Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tadej Janež

Odkrivanje gruč sorodnih učnih nalog za izboljšanje napovednih modelov posameznih nalog

Doktorska disertacija

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2013

Povzetek

Hkratno učenje več nalog (angl. multi-task learning, MTL) je paradigma strojnega učenja, ki se ukvarja z vzporednim grajenjem modelov za več učnih nalog, ki so si med seboj sorodne. Sorodnost v kontekstu MTL pomeni, da učne naloge prihajajo iz iste problemske domene (npr. imajo skupen prostor atributov). To omogoča, da si pri gradnji modela za določeno učno nalogo pomagamo z informacijami iz podatkov drugih učnih nalog, ki so specifične za skupno problemsko domeno. Večina dosedanjih MTL-pristopov temelji na predpostavki, da so vse učne naloge sorodne in primerne za skupno učenje. Vendar združevanje podatkov dveh nesorodnih nalog lahko vodi do poslabšanja rezultatov na obeh nalogah. Velik izziv na področju MTL je zato vprašanje, kako odpraviti to predpostavko in le selektivno združevati podatke različnih učnih nalog, tako da nesorodne naloge ne vplivajo druga na drugo.

V doktorski disertaciji predstavimo novo MTL-metodo, imenovano *Error-reduction merging* (*ERM*), ki sodi v skupino novejših MTL-pristopov, ki skušajo odpraviti predpostavko, da so vse učne naloge sorodne. *ERM* avtomatsko odkrije gruče sorodnih učnih nalog (le na podlagi podatkov samih) ter združi podatke učnih nalog, ki pripadajo isti gruči, kar vodi k izboljšanju napovednih modelov za posamezne učne naloge. Za definicijo sorodnosti *ERM* uporablja predpostavko, da sta dve učni nalogi sorodni, če ima model, zgrajen na združenih podatkih, manjšo napovedno napako od modelov, zgrajenih na podatkih za posamezni učni nalogi. Števila gruč ni potrebno podati vnaprej, *ERM* ga poišče samodejno. Še ena prednost metode *ERM* v primerjavi z večino MTL-pristopov je, da ni vezana na določeno osnovno učno metodo, ampak lahko znotraj nje uporabimo poljubno metodo nadzorovanega učenja.

V nadaljevanju definiramo tri lastnosti MTL-problema, ki so pomembne z vidika avtomatskega odkrivanja gruč sorodnih učnih nalog z namenom izboljšanja napovednih modelov za posamezno učno nalogo: število učnih nalog, števila primerov posameznih učnih nalog in število gruč različnih tipov učnih nalog. Nato izvedemo obsežno eksperimentalno študijo obnašanja metode *ERM* glede na prej definirane la-

stnosti MTL-problema ter glede na stopnjo šuma v podatkih in uporabljeno osnovno učno metodo znotraj *ERM*. Rezultati na sintetičnih MTL-problemih učenja Boolovih funkcij so pokazali, da se uspešnost metode *ERM* s povečevanjem števila učnih nalog ter s povečevanjem števil primerov posameznih učnih nalog povečuje. Razlika v doseženi ploščini pod ROC-krivuljo (angl. area under ROC curve, AUC) med *ERM* in pristopoma brez združevanja (*NoMerging*) oz. z združevanjem vseh učnih nalog (*MergeAll*) se hitro povečuje. Hkrati pa se njen AUC hitro približuje učenju, kjer je pripadnost gručam podana vnaprej (*Oracle*). S povečevanjem števila gruč različnih tipov učnih nalog ter s povečevanjem stopnje šuma v podatkih se uspešnost metode *ERM* zmanjšuje. V prvem primeru vrednosti AUC metode *ERM* ostajajo večje od metod *NoMerging* in *MergeAll*, vendar se razlika do vrednosti AUC metode *Oracle* povečuje. V drugem primeru so vrednosti AUC metode *ERM* za manjše stopnje šuma še relativno blizu metodi *Oracle*, s povečevanjem stopnje šuma pa začnejo hitro padati. Pri najvišji stopnji šuma *ERM* povsem odpove in deluje podobno ali slabše od metode *MergeAll*. Eksperimenti, kjer smo znotraj *ERM* uporabili različne osnovne učne metode (metodo podpornih vektorjev, odločitveno drevo, k-najbližjih sosedov), so pokazali, da je njena uspešnost v splošnem neodvisna od uporabljene osnovne učne metode. Rezultati na realnih klasifikacijskih in regresijskih MTL-problemih so potrdili, da metoda *ERM* dobro deluje tudi na praktičnih problemih. Ugotovitve študije primera pohitritve učenja robotskega agenta z izvajanjem eksperimentov v kompleksnejšem okolju se v celoti ujemajo z ugotovitvami obnašanja *ERM* na sintetičnih MTL-problemih.

Doktorsko disertacijo zaključimo z interpretacijo binarizacije vrednosti atributov pri gradnji odločitvenih dreves v smislu delovanja *ERM*. V kvantitativni primerjavi metod se je *ERM* izkazala za superiorno. Kvalitativna primerjava je pokazala, da med strukturo dreves, kjer se veje cepijo po atributu *id* (identifikator učnih nalog), in dendrogrami, ki prikazujejo potek združevanja učnih nalog z metodo *ERM*, ni moč najti nobene podobnosti.

Ključne besede

umetna inteligenca, strojno učenje, hkratno učenje več nalog, nadzorovano učenje, Boolove funkcije, avtonomni učeči se agenti, hitrost učenja, binarizacija vrednosti atributov, odločitvena drevesa

Abstract

Multi-task learning (MTL) is a machine learning paradigm concerned with concurrent learning of models for multiple related learning tasks. In the context of MTL, related learning tasks signify tasks from a common problem domain (e.g. sharing the same attribute space). This enables leveraging of the domain-specific information contained in the data of the related learning tasks when building a model for a particular task. Most of the preceding MTL approaches assume that all learning tasks are related and suitable for joint learning. However, merging the data of two unrelated learning tasks may result in worse performance for both tasks. Thus, the question on how to eliminate this assumption and only selectively merge the data of different learning tasks, in order to prevent unrelated tasks influence each other, represents a big challenge in the area of MTL.

The doctoral thesis presents a new MTL method, named *Error-reduction merging* (*ERM*), that belongs to the group of newer MTL approaches which try to eliminate the assumption that all tasks are related. *ERM* automatically discovers clusters of related learning tasks (solely from their data) and merges the data of learning tasks belonging to the same cluster, which leads to improvement of prediction models of individual learning tasks. For defining relatedness, *ERM* uses the assumption that two tasks are related if the model, built on merged data, decreases the prediction error compared to models built on separate data of each learning task. The number of clusters does not need be the given, *ERM* determines it automatically. Another advantage of *ERM* compared to the majority of other MTL approaches is that it is not tied to a particular base learner. Rather, any supervised learning method can be used inside *ERM*.

Next, we define three properties of an MTL problem which are important from the perspective of automatic discovery of clusters of related learning tasks for improvement of prediction models of individual learning tasks: the number of learning tasks, the numbers of examples of individual learning tasks, and the number of clusters of different types of learning tasks. Then we perform an extensive experimental

study of the behavior of *ERM* with respect to the previously defined properties of an MTL problem and, in addition, with respect to the degree of noise in the data and the base learner used inside *ERM*. Results on synthetic MTL problems of learning Boolean functions demonstrate that the success of *ERM* increases with the increased number of learning tasks and with the increased numbers of examples of individual learning tasks. The difference in the area under ROC curve (AUC) between *ERM* and two other approaches, one without merging (*NoMerging*) and the other with all tasks merged together (*MergeAll*), rapidly increases. At the same time, *ERM*'s AUC quickly catches up with the learning, where cluster membership is given in advance (*Oracle*). Increasing the number of clusters of different types of learning tasks and increasing the degree of noise in the data tends to decrease the success of the *ERM*. In the first case, *ERM*'s AUC values remain bigger than AUCs of methods *NoMerging* and *MergeAll*, however, the difference to AUC values of the *Oracle* increases. In the second case, the AUC values of *ERM* remain relatively close to the *Oracle* for smaller degrees of noise. With the increasing degree of noise, however, they quickly decrease. For the largest degree of noise in the data, *ERM* completely falls short and performs similarly or worse than the *MergeAll* method. Experiments, where we used different base learners (support vector machines, decision tree, k-nearest neighbors) inside *ERM* showed that its success is, in general, independent of the base learner used inside. Results on real classification and regression MTL problems confirmed that *ERM* also performs well on such practical problems. The findings of a study on how to speed up the learning of an autonomous robotic agent by performing experiments in a more complex environment are in complete accordance with the findings of *ERM*'s behavior on synthetic MTL problems.

We end the doctoral thesis with an interpretation of the binarization of attribute values when building decision trees in the sense of *ERM*. The quantitative comparison revealed the superiority of *ERM*. Interestingly, the qualitative comparison showed that there is no similarity between the structure of the trees, where branches are split along attribute *id* (identifier of learning tasks), and dendrograms depicting the history of merging of the learning tasks with *ERM*.

Keywords

artificial intelligence, machine learning, multi-task learning, supervised learning, Boolean functions, autonomous learning agents, learning speed, binarization of attribute values, decision trees

Izjava o avtorstvu doktorske disertacije

Spodaj podpisani *Tadej Janež*,
z vpisno številko *63040277*,

sem avtor doktorske disertacije z naslovom

Odkrivanje gruč sorodnih učnih nalog za izboljšanje napovednih modelov posameznih nalog

S svojim podpisom zagotavljam, da:

- sem doktorsko disertacijo izdelal samostojno pod vodstvom mentorja *akad. prof. dr. Ivana Bratka*
- so elektronska oblika doktorske disertacije, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko doktorske disertacije
- in soglašam z javno objavo elektronske oblike doktorske disertacije v zbirki "Dela FRI".

V Ljubljani, dne 12. decembra 2013 Podpis avtorja:

Zahvala

Najprej bi se rad zahvalil svojemu mentorju akad. prof. dr. Ivanu Bratku za vso potrpežljivost, nasvete in komentarje ter svobodo, ki mi jo je puščal pri raziskovalnem delu. Zahvala gre tudi preostalima članoma komisije za oceno doktorske disertacije, doc. dr. Matjažu Kukarju in prof. dr. Matjažu Gamsu, za njune konstruktivne pripombe. Tretjo zahvalo namenjam prijateljici Tjaši Kocjan za njeno skrbno lektoriranje disertacije in veliko mero potrpežljivosti pri branju povsem tuje tematike.

Naslednja zahvala gre članom Laboratorija za umetno inteligenco, ob katerih so se brusile in oblikovale moje ideje za raziskovalno delo. Martinu Možini se zahvaljujem za številne potrpežljive odgovore na moja vprašanja glede strojnega učenja, statističnih testov in programiranja na splošno. Hvaležen sem mu tudi za vso pomoč pri razvoju metode *ERM*. Aleksandru Sadikovu se zahvaljujem za vse trenutke, ko je s svojo prisotnostjo v laboratorij prinašal sproščenost in vedrino ter mi s "pogledom od zunaj" svetoval glede moje znanstvene poti. Aljažu Košmerlju se zahvaljujem za vsa leta, ki sva jih preživela (dobesedno) drug ob drugem ter si v dobrih in slabih trenutnih stala ob strani. Juretu Žabkarju se zahvaljujem, da mi je odprl pogled v zanimiv svet robotike in kvalitativnega modeliranja ter za številne pogovore o življenjskih temah. Vidi Groznik se zahvaljujem za dobro voljo in pripravljenost za pomoč. Timoteju Lazarju se zahvaljujem za (premalo številne) debate o Linuxu in drugih odprto-kodnih zadevah. Mateju Guidu se zahvaljujem za zanimive pogovore, ki so se praviloma vedno končali s šahom. Zahvaljujem se tudi Lanu Žagarju, da mi je predstavil področje hkratnega učenja več nalog ter za vsa kosila, druženja in druge stvari na poti do doktorata. Juretu Žbontarju pa se zahvaljujem za zgled "hard-core Linux geeka" in številne nasvete glede programiranja.

Na koncu se zahvaljujem še Bogu za dar življenja in vse talente, ki jih imam, ter svojim bližnjim, ki so me podpirali na poti do doktorata in bili z mano zelo potrpežljivi, ko mi je zanje zmanjkalo časa. Iskrena hvala moji zaročenki Maji, ki mi vsak dan prinaša veselje, ter vsem družinskim članom: očetu Tomažu, mami Valeriji, sestri Kristini ter bratoma Mateju in Jerneju. Iskrena hvala tudi vsem prijateljem.

Kazalo

1. POGLAVJE

Uvod

Problematika, s katero se ukvarja pričujoča disertacija, spada na področje umetne inteligence, ki je del širšega področja računalniških znanosti. Natančneje, spada na področje *strojnega učenja* (angl. *machine learning*).

Uporabili bomo standardni režim strojnega učenja, ki ga imenujejo *učenje atribut-vrednost* (angl. *attribute–value learning*) [1]. Pri tem režimu strojnega učenja so podatki podani v obliki tabele, kjer vsaka vrstica predstavlja *primer* (angl. *example*) in vsak stolpec *atribut*. Elementi tabele so vrednosti atributov za vsak primer. Velja poudariti, da so pri tem režimu učenja vsi primeri opisani z enakimi atributi, prav tako so opisi primerov neodvisni drug od drugega.

V disertaciji se bomo omejili na *nadzorovano učenje* (angl. *supervised learning*) [2], podpodročje strojnega učenja, katerega cilj je poiskati funkcijo, ki za dane vrednosti atributov (pogosto imenovanih tudi neodvisne spremenljivke) vrne vrednost ciljnega atributa (pogosto imenovanega tudi odvisna spremenljivka). Če je ciljni atribut diskreten (bodisi dvojiški ali večvrednosten), potem mu pravimo *razred* (angl. *class*), problemu učenja pa *klasifikacija* (angl. *classification*). Če je ciljni atribut zvezen, problemu učenja pravimo *regresija* (angl. *regression*). Z izbranim algoritmom nadzorovanega učenja skušamo poiskati vzorce in zakonitosti, ki veljajo za primere iz tabele podatkov, in na njihovi osnovi zgraditi *model*, ki nam bo služil za napovedovanje vrednosti ciljnega atributa za nove primere. V praksi iskana funkcija običajno ni deterministična, zato so zgrajeni modeli lahko le dobri približki le-te.

Konkreten problem nadzorovanega učenja s pripadajočimi podatki bomo v kontekstu disertacije poimenovali *učna naloga* (angl. *learning task*). Sedaj lahko natančno opredelimo področje disertacije — hkratno učenje več nalog.

1.1 Hkratno učenje več nalog

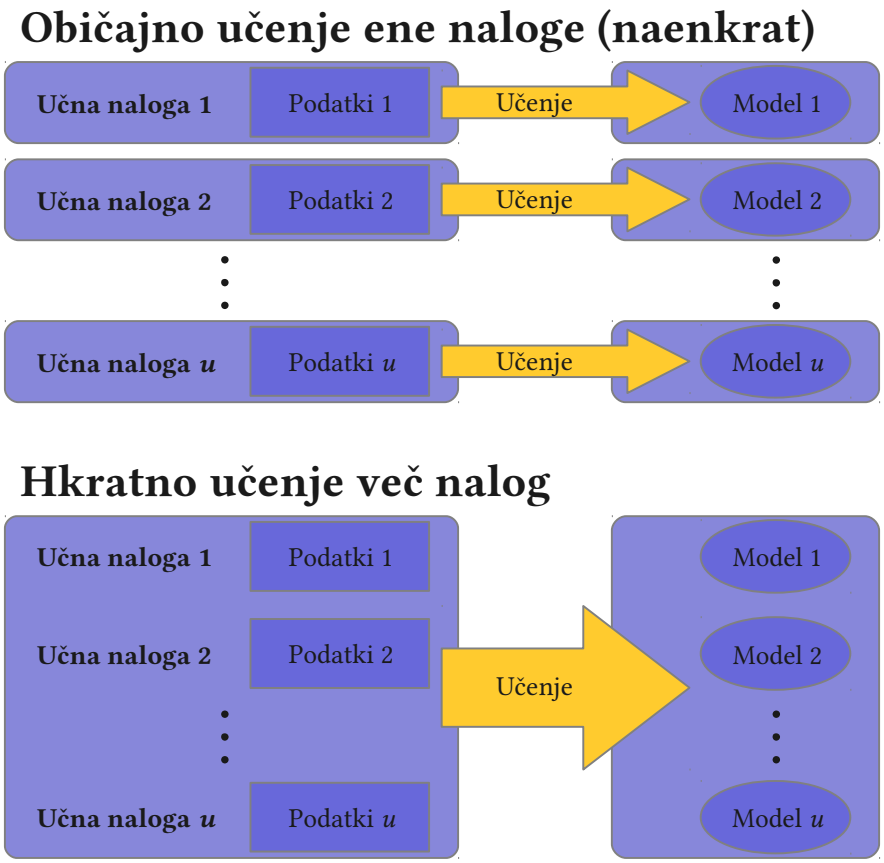
Hkratno učenje več nalog (angl. *multi-task learning*, *MTL*) je paradigma strojnega učenja, ki se ukvarja z vzporednim grajenjem modelov za več učnih nalog, ki so si med seboj sorodne [3, 4]. Sorodnost v kontekstu MTL pomeni, da učne naloge prihajajo iz iste problemske domene (npr. imajo skupen prostor atributov). To omogoča, da si pri gradnji modela za določeno učno nalogo pomagamo z informacijami iz podatkov drugih učnih nalog, ki so specifične za skupno problemsko domeno. Pričakujemo, da bomo na ta način za posamezne učne naloge zgradili splošnejše modele, ki bodo na novih podatkih dajali točnejše napovedi.

Razlika med običajnim t. i. *učenjem ene naloge* (angl. *single-task learning*, *STL*) in MTL-om je prikazana na sliki 1.1. Pri STL-u učni algoritem vsako učno nalogo obravnava ločeno od ostalih in neodvisno zgradi modele učnih nalog. Pri MTL-u pa učni algoritem modele učnih nalog gradi hkrati in si lahko pri izgradnji modela za posamezno učno nalogo pomaga tudi z informacijami iz podatkov ostalih učnih nalog, ki so specifične za skupno problemsko domeno.

Motivacija za tak pristop k učenju izhaja iz opazovanja običajnega STL-pristopa [3]. Pri tem pristopu se večje probleme običajno razbije na niz razmeroma neodvisnih podproblemov in učenje modela za posamezen podproblem poteka neodvisno od ostalih. Vendar je to v nasprotju z intuicijo, ki pravi, da bi s hkratnim učenjem več podproblemov (oz. celotnega problema hkrati) dobili potencialno bogat vir informacij, ki ga ponuja veliko realnih problemov — informacije, ki jih vsebujejo primeri sorodnih podproblemov (v naši terminologiji učnih nalog) iz iste domene. Če bi uspeli razviti učni algoritem, ki bi izkoristil potencial dodanih primerov sorodnih učnih nalog, se lahko nadejamo izgradnje boljših modelov za posamezne učne naloge.

Druga motivacija za MTL-pristop k učenju izhaja iz opazovanja človeškega učenja [5]. Ljudje se srečujemo z nepretrganim nizom učnih nalog. Ko se učimo, se ne naučimo samo določenih učnih konceptov ali motoričnih sposobnosti, ampak tudi *učne pristranskosti* (angl. *learning bias*), tj. kako posploševati. To nam omogoča, da velikokrat stvari pravilno posplošimo, čeprav imamo na voljo zelo malo primerov. Analogno želimo razviti učne metode, ki se bodo skozi izgradnjo modelov za več učnih nalog sposobne naučiti tudi učne pristranskosti, kar bo vodilo do boljšega učenja modelov za nove učne naloge.

V preteklosti so MTL-paradigmo imenovali tudi *učiti se učenja* (angl. *learning to learn*) [5, 6]. Čeprav izraz učiti se učenja napeljuje na nekaj širšega in bolj splošnega,



Slika 1.1: Ponazoritev razlike med običajnim t. i. *učenjem ene naloge* (angl. *single-task learning, STL*) in *hkratnim učenjem več nalog* (angl. *multi-task learning, MTL*). Pri STL-u učni algoritem vsako učno nalogo obravnava ločeno od ostalih in neodvisno zgradi modele učnih nalog. Pri MTL-u pa učni algoritem modele učnih nalog gradi hkrati in si lahko pri izgradnji modela za posamezno učno nalogo pomaga tudi z informacijami iz podatkov ostalih učnih nalog, ki so specifične za skupno problemsko domeno.

ga tukaj omenjamo kot sopomenko za MTL, saj so v [5] zanj podali definicijo, ki je skladna s pojmovanjem MTL:

Definicija (Povzeto po [5]). Naj ima učni algoritem podano:

(1) družino učnih nalog,

(2) primere za vsako učno nalogo,

(3) družino mer zmogljivosti (za vsako učno nalogo ima lahko svojo mero).

Potem pravimo, da se algoritem uči učenja, če se njegova zmogljivost izboljšuje s povečevanjem števila primerov ter s povečevanjem števila učnih nalog.

Poleg tega so članki s področja MTL nekaj časa oba izraza uporabljali kot sinonima [7–9], šele kasneje pa se je bolj uveljavil izraz MTL in so izraz učiti se učenja opustili [10–16].

Skupne korenine z MTL ima tudi paradigma, imenovana *učenje s prenosom* (angl. *transfer learning*) [17]. Učenje s prenosom se od MTL razlikuje po tem, da pri prvem metode skušajo izluščiti znanje iz ene ali več izvornih učnih nalog in ga uporabiti pri učenju modela za ciljno nalogo, medtem ko se pri MTL metode vseh učnih nalog učijo hkrati. Vloge učnih nalog so torej pri MTL simetrične, medtem ko so pri učenju s prenosom nekatere učne naloge izvorne, druge pa ciljne. Z vidika paradigme učenja s prenosom so pomembne zgolj ciljne naloge.

1.2 Sorodno delo

Ključni vidik vseh MTL-metod je vpeljava učne pristranskosti v skupnem prostoru hipotez vseh učnih nalog. Izbira učne pristranskosti odraža naše domneve o sorodnosti učnih nalog. V dosedanjih raziskavah uporabljajo dva glavna načina modeliranja te sorodnosti. Prvi način predpostavlja, da so parametri modelov vseh učnih nalog blizu. Bližino merijo z različnimi normami razlik parametrov učnih nalog (npr. evklidsko, Frobeniusovo, prvo, neskončno) [9, 18–20] ali pa uporabljajo skupno *apriorno verjetnostno porazdelitev* (angl. *probabilistic prior*) za vse modele [6, 21–23]. Drugi način modeliranja sorodnosti učnih nalog pa predpostavlja, da imajo naloge v ozadju prikrito skupno strukturo, tj. skupen latenten nabor atributov [3, 11, 24]. Pristop, ki ga bomo uporabili v disertaciji, ne spada v nobeno od omenjenih skupin. V disertaciji

bomo predpostavili, da sta dve učni nalogi sorodni, če ima model, zgrajen na združenih podatkih obeh učnih nalog, manjšo napovedno napako od modelov, zgrajenih na podatkih posamezne učne naloge.

Večina dosedanjih MTL-pristopov temelji na predpostavki, da so vse učne naloge sorodne in primerne za skupno učenje (oz. da obstaja ekspert, ki lahko določi, katere naloge so sorodne) [3, 9, 11, 19–22, 24]. Vendar združevanje podatkov dveh nesorodnih nalog lahko vodi tudi do poslabšanja rezultatov pri obeh nalogah. Ta fenomen imenujejo *negativni prenos* (angl. *negative transfer*) [17, 25]. Velik izziv na področju MTL je zato vprašanje, kako se znebiti te predpostavke in samo selektivno združevati podatke različnih učnih nalog, tako da nesorodne naloge ne vplivajo druga na drugo. V zadnjem času so se pojavili različni pristopi, ki se skušajo znebiti te predpostavke. V nadaljevanju jih bomo na kratko predstavili.

Prva skupina tovrstnih pristopov privzame, da lahko učne naloge razdelimo v *gruče* (angl. *clusters*) ter da so parametri učnih nalog znotraj iste gruče bodisi blizu (po neki metriki) bodisi imajo skupno apriorno verjetnostno porazdelitev [7, 10, 12, 15, 18]. V [18] predpostavijo, da je razdelitev v gruče podana vnaprej, v [7] je število gruč vhodni parameter, preostale metode [10, 12, 15] pa samodejno poiščejo gruče sorodnih učnih nalog. Vsi omenjeni pristopi privzamejo, da so gruče med seboj neodvisne in da med njimi ni prekrivanj. Podobno idejo sta nekoliko prej (leta 1998) raziskovala tudi Thrun in O'Sullivan [26]. V svojem delu sta sorodnost učnih nalog ocenjevala s tem, kako je znanje (parametri modela), preneseno iz ene učne naloge, izboljšalo model za drugo učno nalogo. Omejila sta se na učenje mere razdalje za metodo *k-najbližjih sosedov* (angl. *k-nearest neighbors*, *k-NN*). Učne naloge so bile razvrščene v gruče, ki so maksimizirale povprečno pričakovano točnost, pri čemer so naloge iz iste gruče uporabljale isto mero razdalje. Optimalna mera razdalje za posamezno gručo je bila tista, ki je minimizirala vsoto napak vseh učnih nalog iz gruče. Število gruč je moralo biti podano vnaprej.

Drugi pristopi, ki vključujejo razvrščanje učnih nalog v gruče, spadajo na področje *regularizacije podprostorov* (angl. *subspace based regularization*) [14, 16, 27]. V omenjenih pristopih predpostavijo, da parametri učnih nalog znotraj iste gruče razpenjajo nizkodimenzionalen podprostor. V [27] in [14] predpostavijo, da med podprostori posameznih gruč ni nobenega prekrivanja, v [16] pa dovoljujejo, da imata dve učni nalogi iz različnih gruč enega ali več baznih vektorjev, ki predstavljajo parametre učnih nalog, skupnih. V [27] predpostavijo, da je število gruč podano, v [14] v primeru, da število gruč ni podano, le-tega ocenijo z internim prečnim preverjanjem, v [16]

pa predpostavijo, da je podano število latentnih baznih vektorjev. Nekoliko drugačen pristop, ki temelji na regularizaciji, so ubrali v [28]. Pri predpostavki, da parametri učnih nalog znotraj gruče ležijo v nizkodimenzionalnem podprostoru, dovoljujejo tudi majhna odstopanja. V definiciji optimizacijskega problema namreč hkrati kaznujejo odstopanja od bločno-redke strukture ter (splošno) redke strukture matrike. Dobljeno matriko parametrov zaradi "nečiste" bločne strukture imenujejo *umazan model* (angl. *dirty model*).

Obstajajo tudi pristopi, ki predpostavljajo, da imamo eno gručo sorodnih učnih nalog in majhno gručo nepovezanih *osamelcev* (angl. *outliers*), ki niso sorodni nobeni izmed učnih nalog [29, 30]. V [29] vpeljejo posplošitev *Gaussovih procesov* [31], t-procese, ki so bolj robustni v prisotnosti osamelcev, sicer pa, podobno kot v [21], privzamejo, da imajo vse sorodne učne naloge skupno apriorno verjetnostno porazdelitev. Druga metoda [30] temelji na regularizaciji, ki hkrati išče nizkodimenzionalen podprostor sorodnih učnih nalog ter *skupinsko-redko* (angl. *group-sparse*) strukturo, ki odkriva osamelce.

Nekateri pristopi se skušajo naučiti celotne kovariančne matrike odvisnosti med učnimi nalogami [13, 32, 33]. Tak pristop poleg modeliranja pozitivne in ničelne odvisnosti omogoča tudi modeliranje negativne odvisnosti med nalogami. V [32] za ocenitev kovariančne matrike uporabijo Gaussove procese, v [33] pa robustnejše t-procese. V [13] gradijo na delu, predstavljenem v [32], pri čemer skušajo njegove pomanjkljivosti odpraviti z vpeljavo novega regularizacijskega člena, tako da kriterijska funkcija postane konveksna.

Pristop, ki ga bomo razvili v disertaciji, imenovan *Error-reduction merging* (*ERM*) [34], prav tako odpravi predpostavko, da so vse učne naloge sorodne. Za razliko od prej naštetih pristopov bomo sorodnost definirali preko zmanjšanja napovedne napake modela, zgrajenega na združenih podatkih (pojem združenih podatkov bomo uvedli kasneje). Predpostavili bomo, da sta dve učni nalogi sorodni, če model, zgrajen na združenih podatkih, zmanjša napovedno napako modelov, zgrajenih na podatkih za posamezni učni nalogi. Števila gruč ne bo potrebno podati vnaprej, metoda ga bo poiskala samodejno, z ustavitvijo postopka združevanja. Prav tako kot večina pristopov bomo predpostavili, da učne naloge lahko razdelimo v gruče, med katerimi ni prekrivanj. Večina MTL-pristopov je vezanih na določeno osnovno učno metodo, pri našem pristopu pa bo znotraj učnega algoritma možno uporabiti poljubno metodo nadzorovanega učenja (npr. metodo podpornih vektorjev, logistično regresijo, odločitveno drevo).

1.3 Pregled disertacije

V naslednjem poglavju (poglavje 2) bomo najprej podali preprost primer izgradnje priporočilnega sistema za filme, ki bo motiviral naš nov pristop k MTL. Hkrati bo služil tudi za lažje razumevanje formalne definicije lastnosti MTL-problema, ki so pomembne z vidika avtomatskega odkrivanja gruč sorodnih učnih nalog z namenom izboljšanja napovednih modelov posameznih nalog. Temu bo sledila še obsežna predstavitev delovanja nove MTL-metode, imenovane *Error-reduction merging* (*ERM*).

V 3. poglavju bomo predstavili rezultate obsežne eksperimentalne študije obnašanja metode *ERM* glede na prej definirane lastnosti MTL-problema ter glede na stopnjo šuma v podatkih in uporabljeno osnovno učno metodo znotraj *ERM*. V prvem delu bomo podali rezultate eksperimentov na sintetičnih MTL-problemih učenja Boolovih funkcij, kjer smo lahko sami izbirali prej omenjene lastnosti MTL-problemov. Nato bo sledila predstavitev rezultatov eksperimentov na dveh realnih klasifikacijskih MTL-problemih, kjer smo se ukvarjali z nalogo prepoznavanja ročno napisanih števk. Zatem pa sledi še predstavitev rezultatov eksperimentov na dveh realnih regresijskih MTL-problemih. Pri prvem gre za nalogo napovedovanja števila točk učencev pri zaključnem preizkusu znanja, pri drugem pa za napovedovanje naklonjenosti oseb nakupu določenega računalnika.

Sledila bo predstavitev študije primera pohitritve učenja robotskega agenta z izvajanjem eksperimentov v kompleksnejšem okolju (poglavje 4). Po natančnem opisu eksperimentalne domene in predstavitvi relacije, ki se je je robot skušal naučiti, bomo podali definicijo različnih kompleksnosti okolja ter predstavili niz scenarijev, v katerih smo izvajali poskuse. Na koncu bomo podali še rezultate in iz njih izhajajoče ugotovitve.

V 5. poglavju bomo predstavili eksperiment, kjer smo binarizacijo vrednosti atributov pri gradnji odločitvenih dreves interpretirali v smislu delovanja *ERM*. Po natančnem opisu eksperimenta bo sledila predstavitev hipotez, rezultatov in ugotovitev kvantitativne primerjave izbranih metod. Zaključili bomo s predstavitvijo hipotez, rezultatov in ugotovitev opisne primerjave zgrajenih modelov.

V zadnjem poglavju (poglavje 6) bomo povzeli glavne prispevke in ugotovitve doktorske disertacije ter podali nekaj iztočnic za nadaljnje delo.

Osnovna različica metode *ERM* (razdelek 2.3) ter študija primera pohitritve učenja robotskega agenta z izvajanjem eksperimentov v kompleksnejšem okolju (poglavje 4) sta bili objavljeni v članku [34].

1.4 Prispevki k znanosti

Doktorska disertacija prinaša naslednje izvirne prispevke k znanosti:

- Razvoj in implementacija metode *ERM*, ki avtomatsko odkrije gruče sorodnih učnih nalog (le na podlagi podatkov samih) ter združi podatke učnih nalog iz iste gruče, kar vodi k izboljšanju napovednih modelov za posamezne učne naloge.
- Definicija različnih lastnosti problema hkratnega učenja več nalog (na kratko, MTL-problema) in na njih sloneča empirična priporočila za uporabo metode *ERM*. Lastnosti MTL-problema (število učnih nalog, števila primerov posameznih učnih nalog, število gruč različnih tipov učnih nalog) so izbrane z vidika avtomatskega odkrivanja gruč sorodnih učnih nalog z namenom izboljšanja napovednih modelov za posamezno učno nalogo. Empirična priporočila za uporabo metode *ERM* temeljijo na eksperimentalni študiji obnašanja metode *ERM* glede na prej definirane lastnosti MTL-problema ter glede na stopnjo šuma v podatkih in uporabljeno osnovno metodo nadzorovanega učenja znotraj *ERM*.
- Interpretacija binarizacije vrednosti atributov pri gradnji odločitvenih dreves v smislu delovanja metode *ERM*.

2. POGLAVJE

Avtomatsko odkrivanje gruč sorodnih učnih nalog

To poglavje bomo začeli z uvodnim primerom izgradnje priporočilnega sistema za filme. Prek njega bomo pokazali, kaj je motivacija za MTL-pristop, kaj je problem združevanja vseh učnih nalog in kako bi v slednjem primeru koristilo odkrivanje gruč sorodnih učnih nalog.

Potem bomo podali formalno definicijo lastnosti MTL-problema, ki so pomembne z vidika avtomatskega odkrivanja gruč sorodnih učnih nalog z namenom izboljšanja napovednih modelov za posamezno učno nalogo.

V zadnjem razdelku (2.3) bomo predstavili delovanje nove MTL-metode, imenovane *Error-reduction merging* (*ERM*), ki avtomatsko odkrije gruče sorodnih učnih nalog (le na podlagi podatkov samih) in združi podatke učnih nalog iz iste gruče ter tako zgradi boljše napovedne modele za posamezne učne naloge. Osnovna različica metode *ERM* je bila objavljena v članku [34].

2.1 Uvodni primer

Preden formalno opredelimo lastnosti MTL-problema, si poglejmo preprost primer. Janez Novak je pred kratkim ustanovil podjetje Oglej.si, ki se ukvarja s ponujanjem ogleda filmov na zahtevo. Uporabniki morajo na začetku ustvariti svoj račun, nato pa lahko (proti plačilu) izbirajo med tisoči filmov in serij, ki se prek spleta pretočijo na njihov računalnik, televizor ali kakšno drugo napravo.

V podjetju vzdržujejo tudi podatkovno bazo filmov, ki vsebuje podatke o naslovu,

letu nastanka, dolžini, žanru, režiserju filma in igralcih, ki v njem nastopajo. Ko uporabnik konča z ogledom filma, je prek uporabniškega vmesnika naprošen, da oceni (na lestvici od 1 do 5), kako mu je bil film všeč. Tako podjetje dobi povratno informacijo, ki jo lahko izkorišča pri priporočanju novih filmov, ki bi utegnili zanimati uporabnika.

Ker je podjetje Oglej.si novo podjetje z malo uporabniki, želi Janez uveljavljenim ponudnikom konkurirati tako, da bo zgradil kvalitetne modele za napovedovanje uporabniških ocen filmov na podlagi podatkov o le-teh. Za vsakega uporabnika ima tabelo podatkov s filmi, ki si jih je ogledal, njihovimi lastnostmi ter oceno, ki jim jo je dodelil. Primera takih tabel podatkov za uporabnika Ano in Jureta sta podana v tabelah 2.1(a) in 2.1(b).

Pri izgradnji modelov je Janez kmalu naletel na težavo: za vsakega uporabnika ima na voljo zelo malo primerov. Posledica tega je, da je model za napovedovanje ocen filmov slab. To pomeni, da med filmi, ki jih podjetje priporoči uporabniku, ni tistih, ki bi si jih le-ta dejansko želel ogledati. Posledično podjetje Oglej.si nima veliko možnosti za uspeh.

Da bi povečal količino podatkov pri gradnji modela, je Janez združil podatke vseh uporabnikov. Vendar je ponovno naletel na težavo: uporabniške ocene filmov so med seboj zelo različne, kar ravno tako povzroči, da je model, zgrajen na združenih podatkih, slab pri napovedovanju ocen filmov.

Janez se je zaradi vsega tega odločil, da bo v podjetju zaposlil Petra, mladega raziskovalca s področja strojnega učenja. Peter je dobil idejo, da bi lahko boljše napovedne modele zgradili tako, da bi uporabnike združevali v gruče, vendar le tiste, ki filmom dajejo sorodne ocene. Tako je odkril, da bi bilo ugodno združiti podatke Špele, Urše in Tine, saj vse tri dajejo visoke ocene krajšim romantičnim komedijam, ostalim filmom pa ne. Drugo gručo uporabnikov bi tvorili Franc, Micka, Štefka in Jože, ki dajejo zelo visoke ocene starejšim filmom, ostalim filmom pa ne. Naslednja gruča uporabnikov bi bili navdušenci nad filmi režiserja Stevena Spielberga, ki njegovim filmom dajejo visoke ocene, ostalim pa ne. Potem bi imeli gručo nekaj uporabnikov, ki dajejo zelo visoke ocene slovenskim komedijam, ostalim pa ne. Na tak način bi moral Peter poiskati še druge gruče sorodnih uporabnikov, ki filmom dajejo podobne ocene.

Če bi Peter uspel razviti metodo za avtomatsko odkrivanje gruč sorodnih uporabnikov, bi podjetje Oglej.si dobilo nezanemarljivo konkurenčno prednost. Mu bo uspelo?

naslov	leto	dolžina	žanr	režiser	igralci	ocena
The Godfather	1972	175 min	kriminalka, drama	F. Coppola	M. Brando ...	3
Poletje v školjki	1986	85 min	mladinski, pustolovski	T. Štiglic	D. Sluga ...	4
Schindler's List	1993	195 min	drama, zgodovinski	S. Spielberg	L. Neeson ...	2
Titanic	1997	194 min	romantična drama	J. Cameron	L. DiCaprio ...	3
Kajmak in marmelada	2003	98 min	drama, komedija	B. Đurić	T. Ribič ...	5
Memento	2000	113 min	triler	C. Nolan	G. Pearce ...	4
Moj ata, socialistični kulak	1987	100 min	drama, komedija	R. Ranfl	P. Bibič ...	4
Vesna	1953	91 min	romantična komedija	F. Čáp	F. Trefalt ...	5

(a) Tabela podatkov za uporabnico Ano.

naslov	leto	dolžina	žanr	režiser	igralci	ocena
Forrest Gump	1994	142 min	romantična drama	R. Zemeckis	T. Hanks ...	2
The Matrix	1999	136 min	akcijski, znan.-fant.	Wachowski	K. Reeves ...	5
The Exorcist	1973	122 min	grozljivka	W. Friedkin	E. Burstyn ...	3
Cvetje v jeseni	1973	122 min	romantična drama	M. Klopčič	P. Bibič ...	3
To so gadi	1977	96 min	komedija	J. Bevc	B. Sotlar ...	5
Smrkci	2011	103 min	animirani, pustolovski	R. Gosnell	*ni relevantno*	1
Gladiator	2000	155 min	akcijski, drama	R. Scott	R. Crowe ...	4
Učna leta izumitelja Polža	1982	89 min	mladinski, komedija	J. Kavčič	M. Petrovčič ...	5
Inception	2010	148 min	akcijski, pustolovski	C. Nolan	L. DiCaprio ...	4
Casablanca	1942	102 min	romantična drama	M. Curtiz	H. Bogart ...	2
Gremo mi po svoje	2010	92 min	mladinski, pustolovski	M. Hočevar	J. Zrnec ...	3
Ben-Hur	1959	212 min	akcijski, drama	W. Wyler	C. Heston ...	5
Braveheart	1995	177 min	akcijski, drama	M. Gibson	M. Gibson ...	4
Levji kralj	1994	89 min	animirani, pustolovski	R. Allers	*ni relevantno*	3

(b) Tabela podatkov za uporabnika Jureta.

Tabela 2.1: Primera tabel podatkov s filmi, njihovimi lastnostmi ter oceno, ki jim jo je dodelil uporabnik.

2.2 Formalna definicija lastnosti MTL-problema

Po uvodnem primeru bomo sedaj še formalno definirali lastnosti MTL-problema. Omejili se bomo le na tiste lastnosti, ki so pomembne z vidika avtomatskega odkrivanja gruč sorodnih učnih nalog z namenom izgradnje boljših napovednih modelov za posamezno učno nalogo. Izbrali smo tri lastnosti: število učnih nalog, števila primerov posameznih učnih nalog in število gruč različnih tipov učnih nalog.

Še prej bomo vpeljali nekaj matematičnih notacij, povezanih z učnimi nalogami, ki jih bomo uporabljali v disertaciji. Naj t označuje učno nalogo in l_t njene podatke iz prostora atributov $\mathcal{X} = X_1 \times X_2 \times \ldots \times X_A$ in ciljnim atributom Y, kjer A predstavlja število atributov. Tabelo podatkov l_t sestavljajo primeri oblike $(\mathrm{x_i}, y_i)$, kjer $\mathrm{x_i} \in \mathcal{X}$ predstavlja vektor vrednosti atributov i-tega primera in $y_i \in Y$ vrednost njegovega ciljnega atributa. Funkcija $f_t : \mathcal{X} \to Y$ je iskana (neznana) funkcija, ki za dane vrednosti atributov primera (x) vrne vrednost ciljnega atributa (y). V disertaciji bomo uporabili standardno predpostavko področja MTL, da imajo vse učne naloge podatke iz istega prostora atributov in isti ciljni atribut.

V uvodnem primeru uporabniki predstavljajo učne naloge, njihove tabele podatkov pa podatke s prostora atributov, ki opisujejo lastnosti filmov. V našem primeru imamo šest atributov: naslov, leto nastanka, dolžina, žanr, režiser in igralci ter ciljni atribut ocena. Primeri so podatki o posameznih filmih s pripadajočo oceno, ki pove, v kakšni meri je bil uporabniku film všeč. Funkcija, ki jo iščemo, za dane lastnosti filma vrne napoved uporabnikove ocene zanj.

Sedaj lahko podamo definicijo prvih dveh lastnosti MTL-problema, ki sta pomembni s prej omenjenega vidika avtomatskega odkrivanja gruč sorodnih učnih nalog z namenom izgradnje boljših napovednih modelov za posamezno učno nalogo.

Definicija 2.1. Naj bodo $t_1, t_2, \ldots, t_T$ učne naloge MTL-problema. Prva lastnost MTL-problema je *število učnih nalog* $T = \left|\{t_1, t_2, \ldots, t_T\}\right|$.

Definicija 2.2. Naj bodo $t_1, t_2, \ldots, t_T$ učne naloge MTL-problema. Potem $n_{t_1} = \left|l_{t_1}\right|$, $n_{t_2} = \left|l_{t_2}\right|, \ldots, n_{t_T} = \left|l_{t_T}\right|$ predstavljajo drugo lastnost MTL-problema: *števila primerov posameznih učnih nalog.*

V našem primeru je število učnih nalog kar število vseh uporabnikov. Števila primerov posameznih učnih nalog pa so števila filmov, ki so si jih posamezni uporabniki ogledali in ocenili.

Tretja lastnost MTL-problema, ki jo bomo definirali, je povezana z različnimi tipi učnih nalog, ki so del MTL-problema. V disertaciji bomo odpravili pogosto uporabljeno predpostavko, da so vse učne naloge sorodne. Tako kot večina pristopov, ki odpravijo to predpostavko, bomo namesto tega predpostavili, da lahko učne naloge razdelimo v gruče sorodnih učnih nalog, med katerimi ni prekrivanj. Glede sorodnosti učnih nalog bomo predpostavili, da so sorodne tiste učne naloge, ki imajo isto (skupno) iskano funkcijo f_t. Rekli bomo, da imajo te učne naloge isti tip in skupaj tvorijo gručo sorodnih učnih nalog.

Definicija 2.3. Naj bodo $t_1, t_2, \ldots, t_T$ učne naloge MTL-problema in $f_{t_1}, f_{t_2}, \ldots, f_{t_T}$ pripadajoče iskane (neznane) funkcije. Naj bodo $c_1, c_2, \ldots, c_C$ gruče različnih tipov učnih nalog MTL-problema, za katere velja:

(1) $c_i \cap c_j = \emptyset \quad \forall i, j \in \{1, 2, \ldots, C\} \wedge i \neq j$,

(2) $c_1 \cup c_2 \cup \cdots \cup c_C = \{t_1, t_2, \ldots, t_T\}$,

(3) $c_i = \{t_k \mid f_{t_k} = f_{t_l} \quad \forall t_l \in c_i\} \quad \forall i \in \{1, 2, \ldots, C\}$.

Tretja lastnost MTL-problema je *število gruč različnih tipov učnih nalog* $C = \left|\{c_1, c_2, \ldots, c_C\}\right|$.

V našem primeru gruče različnih tipov učnih nalog predstavljajo gruče sorodnih uporabnikov, ki filmom dodeljujejo podobne ocene. Primeri takih gruč so uporabnice, ki dodeljujejo visoke ocene krajšim romantičnim komedijam (ostalim pa ne), uporabniki, ki dodeljujejo visoke ocene starejšim filmom (ostalim pa ne), uporabniki, ki dodeljujejo visoke ocene filmom režiserja Stevena Spielberga (ostalim pa ne) itd.

2.3 Metoda *Error-reduction merging (ERM)*

V tem razdelku bomo predstavili delovanje nove MTL-metode, imenovane *Error-reduction merging* (*ERM*). Metoda *ERM* avtomatsko odkrije gruče sorodnih učnih nalog (le na podlagi podatkov samih) in združi podatke učnih nalog iz iste gruče ter tako zgradi boljše napovedne modele za posamezne učne naloge. Začeli bomo z motivacijo, ki nas je vodila pri njenem razvoju. Nadaljevali bomo s predstavitvijo glavnih idej in visokonivojskim pregledom njenega delovanja. Nato bomo natančno opisali njeno delovanje in delovanje pomožnih metod ter podali njihovo *psevdokodo*. Na koncu bomo predstavili še analizo njene časovne zahtevnosti.

2.3.1 Motivacija

Če bi učni algoritem za dani MTL-problem vedel, katere učne naloge imajo isti tip in skupaj tvorijo gručo sorodnih učnih nalog (kot smo ju definirali v razdelku 2.2), potem bi lahko podatke določene učne naloge razširil s podatki vseh ostalih učnih nalog iz iste gruče. Na ta način bi občutno povečal število primerov, ki so na voljo učnemu algoritmu pri izgradnji modela za to učno nalogo. Pri realnih problemih lahko utemeljeno pričakujemo, da se bodo podatki ostalih učnih nalog vsaj delno razlikovali od podatkov izbrane učne naloge. Še več, če skupaj z ostalimi nalogami izbrana učna naloga resnično tvori gručo sorodnih učnih nalog, potem bodo njihovi učni podatki dopolnjevali učne podatke izbrane učne naloge s primeri, ki pokrivajo druge dele prostora atributov. Prav tako njihovi učni podatki ne bodo v nasprotju z učnimi podatki izbrane učne naloge glede primerov, ki pokrivajo iste dele prostora atributov. Potemtakem lahko pričakujemo, da se bo kvaliteta zgrajenega modela povečala.

Vendar učni algoritem nima podatkov o tem, katere učne naloge so sorodne, zato lahko to le domneva. Njegove hipoteze temeljijo na modelih, ki jih je zgradil s pomočjo učnih podatkov, ki pripadajo danim učnim nalogam.

2.3.2 Pregled delovanja

Združevanje podatkov l_{t_i} in l_{t_j} učnih nalog t_i in t_j v kontekstu metode *ERM* pomeni naslednje:

(1) Najprej se unificirajo pripadajoči atributi obeh nalog. Ker v disertaciji uporabljamo predpostavko, da so podatki vseh učnih nalog iz istega prostora atributov (podano v razdelku 2.2), to preprosto pomeni, da se enaki atributi združijo v enega. V uvodnem primeru bi to pomenilo, da se atribut dolžina pri filmih uporabnice Ane združi z atributom dolžina pri filmih uporabnika Jureta v en sam atribut dolžina.

(2) Nato se *staknejo* (angl. *concatenate*) učni primeri obeh nalog. Na primeru Ane in Jureta bi to pomenilo, da se njuni podatki združijo v skupno tabelo.

Združene podatke bomo označevali z $l_{t_i} + l_{t_j}$.

Za določanje koristnosti združevanja učnih nalog t_i in t_j (tj. njunih podatkov) metoda *ERM* zgradi tri modele z enako *osnovno učno metodo* (angl. *base learner*): enega na l_{t_i}, enega na l_{t_j} in enega na $l_{t_i} + l_{t_j}$ ter jih primerja na izbranih kombinacijah podatkov (l_{t_i}, l_{t_j}, $l_{t_i} + l_{t_j}$).

Naj $\overline{\mathrm{E}}(t_r, t_s)$ označuje povprečno napovedno napako modela, zgrajenega na podatkih učne naloge t_r in testiranega na podatkih učne naloge t_s. $\overline{\mathrm{E}}(t_r, t_s)$ v kontekstu metode *ERM* označuje posebno oceno napake, ki je izračunana s pomočjo posplošene različice prečnega preverjanja, ki bo podrobno opisana v razdelku 2.3.3. Sedaj lahko definiramo *zmanjšanje napake* (angl. *error reduction*, *ER*) ob združitvi učnih nalog t_i in t_j:

$$ER_{t_i,t_j} = \frac{n_{t_i}\,\overline{\mathrm{E}}(t_i, t_i) + n_{t_j}\,\overline{\mathrm{E}}(t_j, t_j)}{n_{t_i} + n_{t_j}} - \overline{\mathrm{E}}(t_i + t_j, t_i + t_j). \tag{2.1}$$

Pozitivna vrednost *ER* pomeni, da je uteženo povprečje povprečnih napovednih napak modelov, zgrajenih in testiranih na podatkih posamezne učne naloge, večje od povprečnih napovednih napak modela, zgrajenega in testiranega na združenih podatkih obeh učnih nalog. To pomeni, da bi se z združevanjem učnih nalog t_i in t_j povprečna napovedna napaka na njunih podatkih zmanjšala. Zato je *ER* naš glavni kriterij pri odločanju, ali naj *ERM* združi dve učni nalogi ali ne.

Združevanje učnih nalog z metodo *ERM* poteka na aglomerativen način, tj. *ERM* najprej združi tisti par učnih nalog, ki sta najboljši kandidatki za združevanje, in nadaljuje, dokler ne ostane le še ena (združena) učna naloga ali pa noben par učnih nalog ne izpolnjuje več kriterijev za združevanje. Ta način združevanja učnih nalog je soroden aglomerativnim metodam *hierarhičnega gručenja* (angl. *hierarchical clustering*) [35].

Na tem mestu velja poudariti, da gre pri tem načinu združevanja za *požrešno metodo* (angl. *greedy algorithm*), ki na vsakem koraku naredi lokalno optimalno izbiro (združi par učnih nalog, ki sta najboljši kandidatki za združevanje) z upanjem, da bo to vodilo tudi do globalnega optimuma. Zanj smo se odločili zaradi njegove časovne učinkovitosti (podrobna časovna analiza metode *ERM* je podana v podrazdelku 2.3.6), hkrati pa se je v praksi pokazalo, da z njim dosegamo zelo dobre rezultate (ugotovitve eksperimentalne študije so podane v razdelku 3.4). Tak pristop je idejno podoben principu *dodajanja naprej* (angl. *forward selection*), kot se uporablja pri *postopni regresiji* (angl. *stepwise regression*) [36] ali *izbiri podmnožice atributov* (angl. *feature subset selection*) [37], kjer se je prav tako izkazal za uspešnega. Metodo *ERM* bi lahko enostavno razširili tudi tako, da bi kot kandidate za združevanje upoštevala še vse trojice, četverke, peterke oz. poljubne *n*-erke učnih nalog. Vendar bi bilo to časovno popolnoma prepotratno, saj bi časovna zahtevnost *ERM* s številom učnih nalog eksponentno naraščala.

Definicija. Par učnih nalog (t_i, t_j) je kandidat za združevanje, če izpolnjuje naslednja kriterija:

(1) število primerov vsake od učnih nalog, n_{t_i} in n_{t_j}, je večje od minimalnega števila primerov, določenega s parametrom η (ki bo podrobneje predstavljen v podrazdelku 2.3.5),

(2) vrednost *ER* za združitev učnih nalog t_i in t_j je večja ali enaka nič.

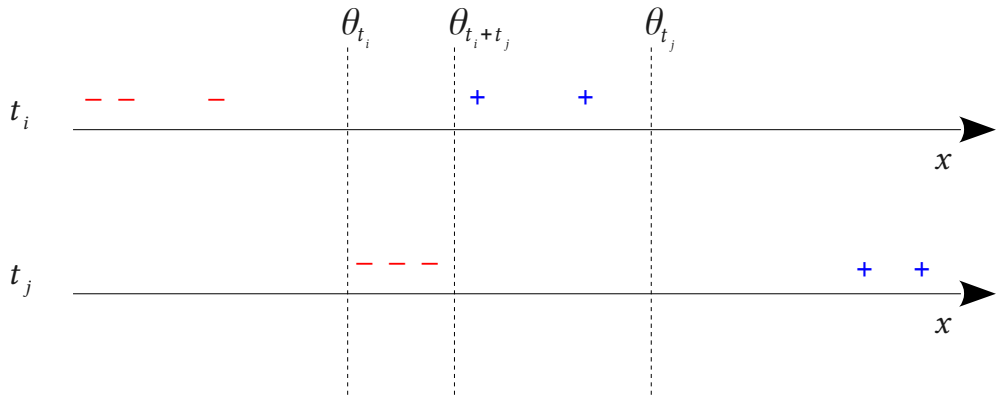
Prvi kriterij je elementaren prag, ki izloči kandidatne pare učnih nalog, katerih število primerov je premajhno. Če ima namreč učna naloga zelo malo primerov, potem ni smiselno ugotavljati, ali ta učna naloga spada v isto gručo sorodnih učnih nalog kot neka druga učna naloga ali ne.

Drugi kriterij primerja povprečne napovedne napake vseh treh modelov (zgrajenih na l_{t_i}, l_{t_j} in $l_{t_i}+l_{t_j}$) na njihovih lastnih podatkih. V skladu z definicijo iz enačbe (2.1) mora biti po tem kriteriju povprečna napovedna napaka modela, zgrajenega in testiranega na združenih podatkih, manjša ali enaka uteženemu povprečju povprečnih napovednih napak modelov, zgrajenih na podatkih posameznih učnih nalog. Ta kriterij zagotavlja, da se z združevanjem učnih nalog t_i in t_j povprečna napovedna napaka na njunih podatkih ne poveča.

Izkaže se, da zgolj opazovanje velikosti *ER* ni najboljši način za izbiro naslednjega para za združevanje. To bomo pokazali s preprostim primerom. Naj bodo primeri točke, ki ležijo na enodimenzionalni premici, funkcija, ki določa razred, pa enostavna dvojiška pragovna funkcija g s parametrom θ:

$$g(x) = \begin{cases} + & \text{če je } x > \theta, \text{ in} \\ - & \text{sicer.} \end{cases}$$

Poleg tega predpostavimo, da imamo kandidatni par učnih nalog (t_i, t_j) s podatki, kot so prikazani na sliki 2.1. Z uporabo osnovne učne metode na vseh kombinacijah podatkov (l_{t_i}, l_{t_j} in $l_{t_i} + l_{t_j}$) smo dobili tri modele, vsakega s svojim približkom parametra θ: θ_{t_i}, θ_{t_j} in $\theta_{t_i+t_j}$ (tudi prikazani na sliki 2.1). Povprečna napovedna napaka vseh treh modelov na svojih podatkih je nič. Torej je tudi *ER* nič. Kriterij za združevanje (2) je izpolnjen, vendar z združevanjem, če gledamo na vrednost *ER*, nič ne pridobimo. Z drugimi besedami, v primerjavi z ostalimi kandidatnimi pari par učnih nalog (t_i, t_j) glede na *ER* ne izgleda preveč obetavno. Toda, če bi opazovali podatke nalog t_i in t_j, bi ugotovili, da se le-ti zelo dobro dopolnjujejo in bi bilo zelo naravno združiti obe učni



Slika 2.1: Preprost primer, ki pokaže pomanjkljivost zgolj opazovanja velikosti *ER*. Primeri so točke, ki ležijo na enodimenzionalni premici, njihov razred je bodisi + (če je $x > \theta$) bodisi − (sicer). Zgornji in spodnji del slike predstavljata podatke učnih nalog t_i in t_j (zaporedoma). Vsaka učna naloga ima pet primerov. Z uporabo osnovne učne metode na vseh kombinacijah podatkov (l_{t_i}, l_{t_j} in $l_{t_i} + l_{t_j}$) smo dobili tri modele, vsakega s svojim približkom parametra θ: θ_{t_i}, θ_{t_j} in $\theta_{t_i+t_j}$.

nalogi. Če bi ju združili, bi namreč njuni primeri skupaj zelo dobro določali θ. Ta primer prikazuje, kako lahko *ER* podceni koristnost združevanja para učnih nalog. Torej lahko rečemo, da *ER* ni dober kriterij za *rangiranje* parov učnih nalog, ki so kandidati za združevanje.

Rešitev za problem iz prejšnjega primera je, da pogledamo, ali držita naslednji neenakosti:

$$\overline{\mathrm{E}}(t_i + t_j, t_i + t_j) \leq \overline{\mathrm{E}}(t_i, t_i + t_j), \tag{2.2}$$

$$\overline{\mathrm{E}}(t_i + t_j, t_i + t_j) \leq \overline{\mathrm{E}}(t_j, t_i + t_j). \tag{2.3}$$

Če se podatki učnih nalog t_i in t_j dobro dopolnjujejo v primerih, ki pokrivajo različne dele prostora atributov, potem bosta zgornji neenakosti držali. Poleg tega kriterij za združevanje (2) (tj. $ER \geq 0$) zagotavlja, da si podatki obeh učnih nalog ne nasprotujejo glede primerov, ki pokrivajo iste dele prostora atributov.

Vrnimo se zopet k prejšnjemu primeru. Poleg že omenjenih povprečnih napovednih napak vseh treh modelov na svojih podatkih, ki so enake nič, lahko izračunamo

še naslednji povprečni napovedni napaki:

$$\overline{\mathrm{E}}(t_i, t_i + t_j) = 3/10,$$
$$\overline{\mathrm{E}}(t_j, t_i + t_j) = 2/10.$$

Vidimo lahko, da obe neenakosti iz enačb (2.2) in (2.3) držita, kar je skladno z intuicijo, da bi bilo koristno združiti učni nalogi t_i in t_j.

Za rangiranje kandidatnih parov učnih nalog metoda *ERM* opazuje, v kakšni meri njihovi podatki pokrivajo iste dele prostora atributov in v kakšni meri pokrivajo različne dele prostora atributov. Zaželeno je namreč, da *ERM* združi tisti par učnih nalog, ki pokrije čim večji del prostora atributov, tako da osnovna učna metoda lahko zgradi čim boljši model za celoten prostor atributov.

Rangiranje kandidatnih parov učnih nalog je narejeno z nizom *t-testov*, ki primerjajo povprečne napovedne napake modelov posameznih učnih nalog s povprečno napovedno napako modela za združeni učni nalogi. Za vsak kandidatni par učnih nalog (t_i, t_j) *ERM* naredi dva parna enostranska t-testa, ki testirata naslednji *ničelni hipotezi*:

$$H_0(t_i) : \overline{\mathrm{E}}(t_i, t_i + t_j) \leq \overline{\mathrm{E}}(t_i + t_j, t_i + t_j), \tag{2.4}$$
$$H_0(t_j) : \overline{\mathrm{E}}(t_j, t_i + t_j) \leq \overline{\mathrm{E}}(t_i + t_j, t_i + t_j), \tag{2.5}$$

ter izračuna njuni *p-vrednosti*. Višja p-vrednost pomeni šibkejšo zavrnitev ničelne hipoteze. Obe ničelni hipotezi, $H_0(t_i)$ in $H_0(t_j)$, želi zavrniti s čim večjo verjetnostjo, zato za najboljši kandidatni par učnih nalog (t_i^*, t_j^*) izbere tistega, ki ima najmanjšo največjo p-vrednost:

$$(t_i^*, t_j^*) = \arg\min_{(t_i, t_j)} \Big\{ \max \Big\{ p\text{-vrednost testiranja } H_0(t_i), p\text{-vrednost testiranja } H_0(t_j) \Big\} \Big\}. \tag{2.6}$$

Po združitvi učnih nalog t_i^* in t_j^* v t_m metoda *ERM* generira vse pare učnih nalog, ki vsebujejo novo učno nalogo t_m, in jih testira, če izpolnjujejo kriterija za združevanje (1) in (2). Poleg tega preveri, če zadoščajo neenakostim iz enačb (2.2) in (2.3). Tisti pari učnih nalog, ki zadoščajo kriterijema za združevanje in omenjenima neenakostma, so dodani v množico kandidatnih parov in postopek rangiranja se ponovi. Ta proces se ponavlja, dokler še obstajajo pari učnih nalog, ki so kandidati za združevanje (podrobnosti tega postopka so opisane v podrazdelku 2.3.5).

Ko je združevanje učnih nalog z metodo *ERM* končano, dobimo množico (združenih) učnih nalog. Vsaka od njih predstavlja gručo sorodnih učnih nalog istega tipa, kot smo jo opredelili v definiciji 2.3. Ker metoda *ERM* ne ve, katere učne naloge v resnici skupaj tvorijo gruče sorodnih učnih nalog, so dobljene (združene) naloge le njen približek. Na koncu *ERM* z osnovno učno metodo za vsako gručo sorodnih učnih nalog zgradi model na vseh njenih podatkih (tj. podatkih vseh učnih nalog dane gruče).

2.3.3 Ocenjevanje napovednih napak s posplošenim prečnim preverjanjem

Pri definiciji *napovedne napake* (angl. *prediction error*) bomo ločili dve možnosti:

(1) V primeru, ko so učne naloge MTL-problema klasifikacijski problemi, definiramo napovedno napako modela M za primer x kot:

$$e_M(x) = 1 - \widehat{P}_M(y^* \mid x), \tag{2.7}$$

kjer je $\widehat{P}_M(y \mid x)$ verjetnost, da ima primer x vrednost razreda enako y, ki jo napove model M, in y^* označuje dejansko vrednost razreda primera x.

(2) V primeru, ko so učne naloge MTL-problema regresijski problemi, definiramo napovedno napako modela M za primer x kot:

$$e_M(x) = \ell(y^*, M(x)), \tag{2.8}$$

kjer ℓ označuje izbrano *funkcijo napake* (angl. *loss function*), y^* označuje dejansko vrednost ciljnega atributa primera x, in $M(x)$ označuje vrednost ciljnega atributa primera x, ki jo napove model M.

Funkcija napake $\ell(y^*, y)$ je poljubna funkcija, ki za dani vrednosti y^* in y vrne nenegativno vrednost, ki je ocena napake. Najpogosteje uporabljeni funkciji napake sta absolutna napaka ($\ell(y^*, y) = |y^* - y|$) in kvadratna napaka ($\ell(y^*, y) = (y^* - y)^2$).

Sedaj lahko podamo psevdokodo metode Generalized-cross-validation (metoda 2.1), posplošene različice *prečnega preverjanja* (angl. *cross-validation*) za izračun

napovednih napak in njihovih povprečij. Posplošitev se nanaša na dejstvo, da se metoda Generalized-cross-validation izogne pristranskim ocenam napovednih napak, do katerih bi prišlo z uporabo običajnega prečnega preverjanja. Kako to doseže, bomo pojasnili v nadaljevanju.

Metoda 2.1 Posplošeno prečno preverjanje za izračun napovednih napak modela, zgrajenega in testiranega na danih podatkih, in njihovega povprečja.

Parameters: $\mathcal{B}$: base learner, k: number of cross-validation folds, ℓ (optional): loss function (if given, it indicates a regression problem).
Input: *learn*: data table containing learning examples, *test*: data table containing testing examples.
Output: E: associative array of prediction errors of examples in *test*, $\overline{\mathrm{E}}$: average prediction error of examples in *test*.

```
1  function Generalized-cross-validation(learn,test)
2      test_folds ← generate k cross-validation folds for test data
3      E ← []                ▷ associative array of prediction errors of examples in test
4      for test_fold ∈ test_folds do
5          M ← B(learn \ test_fold)
6          for x ∈ test_fold do
7              if ℓ is given then
8                  e_M(x) ← ℓ(y*, M(x))
9              else
10                 e_M(x) ← 1 − P̂_M(y* | x)
11             E[x] ← e_M(x)
12     return (E, 1/|test| Σ_{x∈test} E[x])
```

Za delovanje metode Generalized-cross-validation moramo določiti dva parametra: osnovno učno metodo $\mathcal{B}$, s katero bodo zgrajeni modeli, in število ponovitev prečnega preverjanja k. V primeru, da so učne naloge MTL-problema regresijski problemi, je potrebno določiti še funkcijo napake ℓ. V nasprotnem primeru, ko so učne naloge MTL-problema klasifikacijski problemi, pa metoda predpostavlja, da osnovna učna metoda $\mathcal{B}$ vrača modele, ki poleg samih napovedi vračajo tudi verjetnostno porazdelitev za vse vrednosti razreda.

Vhodna argumenta metode Generalized-cross-validation sta dve tabeli podatkov: *learn*, ki vsebuje učne primere, in *test*, ki vsebuje testne primere.

Izhod metode Generalized-cross-validation je par $(\mathrm{E}, \frac{1}{|test|}\sum_{x\in test}\mathrm{E}[x])$, kjer prvi element predstavlja *asociacijsko tabelo* (angl. *associative array*) z napovednimi

napakami modela, zgrajenega na podatkih *learn*, za vse primere iz podatkov *test*, in drugi element njihovo povprečje.

Na začetku metoda primere iz tabele podatkov *test* naključno razbije na k enakovrednih podtabel in tako pripravi podatke za k-kratno prečno preverjanje (vrstica 2). Nato inicializira asociacijsko tabelo E, kamor bo shranjevala izračunane napovedne napake primerov iz podatkov *test* (vrstica 3). Z zunanjo **for** zanko metoda iterira čez vseh k podtabel podatkov *test* (vrstice 4–11). Pri vsaki iteraciji **for** zanke metoda najprej z osnovno učno metodo $\mathcal{B}$ zgradi model M na primerih iz podatkov *learn*, iz katerih poprej odstrani primere iz podtabele *test_fold*, ki jo trenutno testira (vrstica 5). V notranji **for** zanki metoda iterira čez vse primere iz podtabele *test_fold* (vrstice 6–11). Če je funkcija napake ℓ podana, potem napovedno napako modela M za primer x izračuna po formuli iz enačbe (2.8) (vrstica 8), sicer pa po formuli iz enačbe (2.7) (vrstica 10). Na koncu notranje **for** zanke metoda izračunano napovedno napako shrani v asociacijsko tabelo E (vrstica 11).

Razlika med običajnim prečnim preverjanjem [38] in predstavljeno posplošeno različico Generalized-cross-validation je v tem, da slednja kot vhod namesto ene sprejme dve tabeli podatkov. Prva vsebuje učne primere, na katerih se zgradi model, druga pa testne primere, na katerih se zgrajeni model testira. V vsaki ponovitvi prečnega preverjanja metoda preveri, ali so trenutni primeri v podtabeli *test_fold* tudi med učnimi podatki, in jih po potrebi odstrani. Na ta način dobimo boljše ocene napovednih napak, saj testni primeri nikoli niso tudi med učnimi primeri.

Zanimivo je pogledati, kaj se zgodi pri dveh robnih primerih uporabe metode Generalized-cross-validation:

(1) Če sta učna in testna tabela podatkov enaki, potem metoda izvaja standardno prečno preverjanje, saj v tem primeru v vsaki ponovitvi prečnega preverjanja iz učnih podatkov odstrani ravno tiste primere, ki so v trenutni testni podtabeli.

(2) Če učna in testna tabela podatkov nimata skupnih primerov, potem metoda izvaja standardno testiranje, kjer imamo ločene podatke za učenje in testiranje. V tem primeru namreč pri izgradnji modela nikoli ni potrebno odstraniti nobenega primera iz učnih podatkov in tako osnovna učna metoda vedno zgradi enak model. V vsaki ponovitvi prečnega preverjanja izračuna napovedno napako primerov iz ene podtabele testnih primerov, tako da so na koncu vsi testni primeri testirani z enakim modelom, kot če bi uporabili ločene podatke za učenje in testiranje.

Velja še omeniti, da je namen metode Generalized-cross-validation definirati, *kaj* je rezultat posplošenega prečnega preverjanja, in ne, da bi ga morali dejansko implementirati na tak neučinkovit način.

2.3.4 Napovedne napake različnih modelov in izračun p-vrednosti

Za ocenjevanje napovednih napak modelov, zgrajenih in testiranih na različnih kombinacijah podatkov danega para učnih nalog, metoda *ERM* uporablja pomožno metodo Estimate-errors-and-p-values (metoda 2.2). Z $\mathrm{E}_{t_r,\,t_s}$ bomo označili asociacijsko tabelo napovednih napak modela, zgrajenega na podatkih učne naloge t_r in testiranega na podatkih učne naloge t_s. Poleg ocenjevanja napovednih napak ta metoda naredi tudi dva parna enostranska t-testa, ki testirata ničelni hipotezi $H_0(t_i)$ in $H_0(t_j)$, podani v enačbah (2.4) in (2.5) (zaporedoma). Za označevanje p-vrednosti parnega enostranskega t-testa, ki testira ničelno hipotezo: $\overline{\mathrm{E}}(t_r, t_t) \leq \overline{\mathrm{E}}(t_s, t_t)$, bomo uporabili $\mathrm{P}(t_r, t_s; t_t)$.

Vhodni argumenti metode Estimate-errors-and-p-values so množica tabel podatkov vseh učnih nalog MTL-problema $\mathcal{L}$, in par učnih nalog (t_i, t_j), ki sta kandidatki za združevanje.

Izhod metode Estimate-errors-and-p-values je par $(\overline{\mathrm{E}}, \mathrm{P})$, kjer prvi element predstavlja asociacijsko tabelo povprečnih napovednih napak modelov, zgrajenih in testiranih na različnih kombinacijah podatkov para učnih nalog (t_i, t_j), drugi element pa asociacijsko tabelo p-vrednosti parnih enostranskih t-testov, ki testirajo ničelni hipotezi iz enačb (2.4) in (2.5).

Glavni del metode Estimate-errors-and-p-values so klici metode Generalized-cross-validation za izračun napovednih napak modelov, zgrajenih in testiranih na različnih kombinacijah podatkov danega para učnih nalog (vrstice 3–7). V nadaljevanju metoda Estimate-errors-and-p-values izvede še dva parna enostranska t-testa, ki testirata ničelni hipotezi iz enačb (2.4) in (2.5) (vrstici 8–9). Na koncu zbere vse izračunane povprečne napovedne napake modelov in p-vrednosti parnih enostranskih t-testov ter jih shrani v dve asociacijski tabeli (vrstica 10).

2.3.5 Združevanje učnih nalog v gruče

Sedaj lahko predstavimo še delovanje same metode *Error-reduction merging* (*ERM*) (metoda 2.3), ki implementira avtomatsko odkrivanje gruč sorodnih učnih nalog z namenom izboljšanja napovednih modelov za posamezne učne naloge, kot je bilo

Metoda 2.2 Ocenjevanje napovednih napak modelov, zgrajenih in testiranih na različnih kombinacijah podatkov danega para učnih nalog, in izračun p-vrednosti testiranja ničelnih hipotez iz enačb (2.4) in (2.5).

Input: $\mathcal{L}$: set of data tables of all learning tasks of the MTL problem, t_i and t_j: pair of learning tasks that are candidates for merging.

Output: $\overline{\mathrm{E}}$: associative array of average prediction errors of models, built and tested on various combinations of the data belonging to the candidate pair of learning tasks (t_i, t_j), P: associative array of p-values of one-sided t-tests testing the null hypotheses from equations (2.4) and (2.5).

function ESTIMATE-ERRORS-AND-P-VALUES($\mathcal{L}$, t_i, t_j)
  $l_{t_i}, l_{t_j} \leftarrow$ learning sets from $\mathcal{L}$ corresponding to t_i, t_j
  $(\mathrm{E}_{t_i,t_i}, \overline{\mathrm{E}}(t_i, t_i)) \leftarrow$ GENERALIZED-CROSS-VALIDATION(l_{t_i}, l_{t_i})
  $(\mathrm{E}_{t_j,t_j}, \overline{\mathrm{E}}(t_j, t_j)) \leftarrow$ GENERALIZED-CROSS-VALIDATION(l_{t_j}, l_{t_j})
  $(\mathrm{E}_{t_i,t_i+t_j}, \overline{\mathrm{E}}(t_i, t_i+t_j)) \leftarrow$ GENERALIZED-CROSS-VALIDATION($l_{t_i}, l_{t_i} + l_{t_j}$)
  $(\mathrm{E}_{t_j,t_i+t_j}, \overline{\mathrm{E}}(t_j, t_i+t_j)) \leftarrow$ GENERALIZED-CROSS-VALIDATION($l_{t_j}, l_{t_i} + l_{t_j}$)
  $(\mathrm{E}_{t_i+t_j,t_i+t_j}, \overline{\mathrm{E}}(t_i+t_j, t_i+t_j)) \leftarrow$ GENERALIZED-CROSS-VALIDATION($l_{t_i}+l_{t_j}, l_{t_i}+l_{t_j}$)
  $\mathrm{P}(t_i, t_i+t_j; t_i+t_j) \leftarrow$ perform a pair-wise one-sided t-test testing the null hypothesis $H_0(t_i) : \overline{\mathrm{E}}(t_i, t_i + t_j) \leq \overline{\mathrm{E}}(t_i + t_j, t_i + t_j)$ and store the obtained p-value
  $\mathrm{P}(t_j, t_i+t_j; t_i+t_j) \leftarrow$ perform a pair-wise one-sided t-test testing the null hypothesis $H_0(t_j) : \overline{\mathrm{E}}(t_j, t_i + t_j) \leq \overline{\mathrm{E}}(t_i + t_j, t_i + t_j)$ and store the obtained p-value
  collect the computed $\overline{\mathrm{E}}(\cdot, \cdot)$ and $\mathrm{P}(\cdot, \cdot; \cdot)$ values and put them into two associative arrays, $\overline{\mathrm{E}}$ and P
  return $(\overline{\mathrm{E}}, \mathrm{P})$

opisano v podrazdelku 2.3.2. V svoji notranjosti metoda *ERM* za ocenjevanje napovednih napak in izračun p-vrednosti uporablja pomožni metodi Generalized-cross-validation in Estimate-errors-and-p-values, predstavljeni v podrazdelkih 2.3.3 in 2.3.4 (zaporedoma).

Za delovanje metode *ERM* moramo določiti parameter η, ki določa minimalno število primerov posamezne učne naloge, da kandidatni par učnih nalog zadosti kriteriju za združevanje (1), opisanem v podrazdelku 2.3.2.

Metoda *ERM* kot vhodna argumenta sprejme množico vseh učnih nalog MTL-problema $\mathcal{T}$ in množico njihovih tabel podatkov $\mathcal{L}$.

Izhod metode *ERM* je trojica $(\mathcal{T}, \mathcal{L}, \mathcal{M})$, kjer $\mathcal{T}$ predstavlja množico končnih (združenih) učnih nalog (vsaka odkrita gruča učnih nalog je predstavljena s t. i. združeno učno nalogo), $\mathcal{L}$ množico njihovih (združenih) tabel podatkov in $\mathcal{M}$ množico modelov, zgrajenih na podatkih končnih (združenih) učnih nalog.

Na začetku metoda *ERM* inicializira množico C, kamor bo dodajala pare učnih nalog, ki so kandidati za združevanje (vrstica 2). Sledi **for** zanka (vrstice 3–10), ki iterira čez vse pare učnih nalog $(t_i, t_j) \in \mathcal{T}$ in jih doda v množico C, če zadoščajo naslednjim kriterijem:

(1) število primerov vsake od učnih nalog, n_{t_i} in n_{t_j}, je večje od minimalnega števila primerov, določenega s parametrom η (kriterij za združevanje (1)),

(2) vrednost *ER* za združitev učnih nalog t_i in t_j je večja ali enaka nič (kriterij za združevanje (2)),

(3) povprečna napovedna napaka $\overline{\mathrm{E}}(t_i+t_j, t_i+t_j)$ zadošča neenakostma iz enačb (2.2) in (2.3), tj. $\overline{\mathrm{E}}(t_i + t_j, t_i + t_j) \leq \min\left\{\overline{\mathrm{E}}(t_i, t_i + t_j), \overline{\mathrm{E}}(t_j, t_i + t_j)\right\}$.

Razlaga teh kriterijev je podana v podrazdelku 2.3.2.

Potem, ko metoda *ERM* napolni množico C s kandidatnimi pari učnih nalog, začne iterirati čez glavno **while** zanko in s tem nadaljuje, dokler še obstajajo kandidatni pari učnih nalog v množici C (vrstice 11–26). V vsaki iteraciji zanke izbere najboljši kandidatni par učnih nalog (t_i^*, t_j^*), tj. par učnih nalog z najmanjšo največjo p-vrednostjo testne statistike (glej enačbo (2.6)), izračunane z izvedbo dveh parnih enostranskih t-testov, ki testirata ničelni hipotezi $H_0(t_i)$ in $H_0(t_j)$, kot sta definirani v enačbah (2.4) in (2.5) (zaporedoma) (vrstica 12).

Po združitvi učnih nalog t_i^* in t_j^* v novo učno nalogo t_m in združevanju njunih tabel podatkov $l_{t_i^*}$ in $l_{t_j^*}$ v novo tabelo podatkov l_{t_m} (vrstici 14–15), metoda *ERM* po-

Metoda 2.3 *Error-reduction merging* (*ERM*), ki avtomatsko odkrije gruče sorodnih učnih nalog z namenom izboljšanja napovednih modelov za posamezne učne naloge.

Parameters: η: minimal number of examples that each learning task should contain to become a candidate for merging

Input: $\mathcal{T}$: set of all learning tasks of the MTL problem, $\mathcal{L}$: set of data tables of all learning tasks of the MTL problem.

Output: $\mathcal{T}$: set of final (merged) learning tasks (each discovered cluster of learning tasks is represented by a so-called merged learning task), $\mathcal{L}$: set of (merged) data tables of the final (merged) learning tasks, $\mathcal{M}$: set of models built on the data of final (merged) learning tasks.

function *ERM*($\mathcal{T}$, $\mathcal{L}$)
  $C \leftarrow \{\}$ ▷ set of learning task pairs that are candidates for merging
  for each pair $(t_i, t_j) \in \mathcal{T}$ **do**
    $l_{t_i}, l_{t_j} \leftarrow$ learning data from $\mathcal{L}$ corresponding to t_i, t_j
    $n_{t_i} \leftarrow \left|l_{t_i}\right|$, $n_{t_j} \leftarrow \left|l_{t_j}\right|$
    if $n_{t_i} \geq \eta$ **and** $n_{t_j} \geq \eta$ **then**
      run ESTIMATE-ERRORS-AND-P-VALUES($\mathcal{L}, t_i, t_j$) and augment the global associative arrays $\overline{\mathrm{E}}$ and P
      $ER_{t_i,t_j} \leftarrow$ compute *error reduction* of merging objects t_i and t_j (as defined in equation (2.1))
      if $ER_{t_i,t_j} \geq 0$ **and** $\overline{\mathrm{E}}(t_i + t_j, t_i + t_j) \leq \min\left\{\overline{\mathrm{E}}(t_i, t_i + t_j), \overline{\mathrm{E}}(t_j, t_i + t_j)\right\}$ **then**
        $C \leftarrow C \cup \left\{(t_i, t_j)\right\}$
  while $\left|C\right| > 0$ **do**
    $t_i^*, t_j^* \leftarrow \underset{(t_i,t_j) \in C}{\arg\min}\left\{\max\left\{\mathrm{P}(t_i, t_i + t_j; t_i + t_j), \mathrm{P}(t_j, t_i + t_j; t_i + t_j)\right\}\right\}$
    $l_{t_i^*}, l_{t_j^*} \leftarrow$ learning data from $\mathcal{L}$ corresponding to t_i^*, t_j^*
    $t_m \leftarrow$ a new learning task obtained by merging t_i^* and t_j^*
    $l_{t_m} \leftarrow l_{t_i^*} + l_{t_j^*}$
    $\mathcal{T} \leftarrow \mathcal{T} \setminus \left\{t_i^*, t_j^*\right\} \cup \left\{t_m\right\}$
    $\mathcal{L} \leftarrow \mathcal{L} \setminus \left\{l_{t_i^*}, l_{t_j^*}\right\} \cup \left\{l_{t_m}\right\}$
    $C \leftarrow C$ without candidate pairs that contain either t_i^* or t_j^*
    for $t_i \in \mathcal{T} \setminus \left\{t_m\right\}$ **do**
      $l_{t_i} \leftarrow$ learning data from $\mathcal{L}$ corresponding to t_i
      $n_{t_i} \leftarrow \left|l_{t_i}\right|$
      if $n_{t_i} \geq \eta$ **then**
        run ESTIMATE-ERRORS-AND-P-VALUES($\mathcal{L}, t_i, t_m$) and augment the global associative arrays $\overline{\mathrm{E}}$ and P

▷ *Se nadaljuje na strani 26*

Metoda 2.3 *(nadaljevanje) Error-reduction merging* (*ERM*), ki avtomatsko odkrije gruče sorodnih učnih nalog z namenom izboljšanja napovednih modelov za posamezne učne naloge.

▷ *Nadaljevanje z enakim zamikom, kot je v 23. vrstici na strani 25*

24 $ER_{t_i,t_m} \leftarrow$ compute *error reduction* of merging objects t_i and t_m (as defined in equation (2.1))
25 **if** $ER_{t_i,t_m} \geq 0$ **and** $\overline{\mathrm{E}}(t_i + t_m, t_i + t_m) \leq \min\left\{\overline{\mathrm{E}}(t_i, t_i + t_m), \overline{\mathrm{E}}(t_m, t_i + t_m)\right\}$ **then**
26 $C \leftarrow C \cup \left\{(t_i, t_m)\right\}$
27 $\mathcal{M} \leftarrow \{\}$ ▷ set of final (merged) tasks' models
28 **for** $t_i \in \mathcal{T}$ **do**
29 $l_{t_i} \leftarrow$ learning data from $\mathcal{L}$ corresponding to t_i
30 $M_{t_i} \leftarrow \mathcal{B}(l_{t_i})$
31 $\mathcal{M} \leftarrow \mathcal{M} \cup \left\{M_{t_i}\right\}$
32 **return** $(\mathcal{T}, \mathcal{L}, \mathcal{M})$

sodobi množico $\mathcal{T}$, tako da iz nje odstrani t_i^* in t_j^* in vanjo doda t_m (vrstica 16), in množico $\mathcal{L}$, tako da iz nje odstrani $l_{t_i^*}$ in $l_{t_j^*}$ in vanjo doda l_{t_m} (vrstica 17). Poleg tega iz množice C odstrani vse kandidatne pare učnih nalog, ki so vsebovali bodisi t_i^* bodisi t_j^* (vrstica 18).

Vgnezdena **for** zanka (vrstice 19–26) deluje enako kot prva **for** zanka (vrstice 3–10), le da sedaj v množico C dodaja tiste pare učnih nalog, ki vsebujejo novo ustvarjeno učno nalogo t_m in zadoščajo kriterijem za združevanje.

Na koncu metoda inicializira množico $\mathcal{M}$, kamor bo shranjevala modele, zgrajene na podatkih končnih (združenih) učnih nalog (vrstica 27). Sledeča **for** zanka iterira čez vse (združene) učne naloge iz končne množice $\mathcal{T}$ ter na njihovih podatkih zgradi model in ga doda v množico $\mathcal{M}$ (vrstice 28–31).

2.3.6 Časovna zahtevnost

Za analizo časovne zahtevnosti metode *ERM* bomo najprej izvedli analizo časovne zahtevnosti pomožnih metod Generalized-cross-validation in Estimate-errors-and-p-values. Uporabljali bomo standardno *O notacijo* (angl. *big O notation*). S $\tau_{\mathcal{A}}(n)$ bomo označevali limitno časovno zahtevnost (tj. časovno zahtevnost v najslabšem primeru) metode $\mathcal{A}$ za vhod velikosti n.

Glavni del metode Generalized-cross-validation (metoda 2.1) zavzemata obe

for zanki. Zunanja **for** zanka (vrstice 4–11) se ponovi k-krat, kar ustreza številu podtabel prečnega preverjanja. Vsaka iteracija zanke zajema izgradnjo modela M z osnovno učno metodo $\mathcal{B}$ (vrstica 5). Število vhodnih primerov za $\mathcal{B}$ je največ $|learn|$. V notranji **for** zanki (vrstice 6–11) je časovno najpotratnejši izračun napovedi modela M za testni primer x. Tekom celotne metode je število napovedi, ki jih je potrebno izračunati, enako $|test|$. Predpostavili bomo, da je izračun $|test|$ napovedi modela M bistveno hitrejši od njegove izgradnje na vhodnih podatkih velikosti $|learn|$. Sedaj lahko podamo skupno časovno zahtevnost metode Generalized-cross-validation:

$$\tau_{GCV}(k, n) = O(k\, \tau_{\mathcal{B}}(n)), \tag{2.9}$$

kjer n ustreza velikosti tabele podatkov *learn*.

Večji del metode Estimate-errors-and-p-values (metoda 2.2) zavzema pet klicev metode Generalized-cross-validation (vrstice 3–7) s časovno zahtevnostjo, kot je podana v enačbi (2.9). Naj bo $n_{max} = \max_{t \in \mathcal{T}} \left\{ n_t \right\}$, kjer $\mathcal{T}$ predstavlja množico vseh učnih nalog MTL-problema. Izračun p-vrednosti za oba t-testa (vrstici 8–9) ima linearno časovno zahtevnost glede na število primerov n_{t_i} oz. n_{t_j}. S predpostavko, da je časovna zahtevnost izgradnje modela z osnovno učno metodo $\mathcal{B}$ v odvisnosti od števila podanih primerov večja od linearne, lahko izpeljemo časovno zahtevnost metode Estimate-errors-and-p-values:

$$\begin{aligned} \tau_{EEPV}(k, n_{max}) &= 5\ \tau_{GCV}(k, n_{max}) + 2\ O(n_{max}) \\ &= O(k\, \tau_{\mathcal{B}}(n_{max})). \end{aligned} \tag{2.10}$$

Na tem mestu velja omeniti, da lahko z učinkovitejšo implementacijo metode Estimate-errors-and-p-values konstantni faktor 5 pri klicu funkcije Generalized-cross-validation zmanjšamo na 3. Primer take implementacije je v knjižici *PyMTL*, predstavljeni v uvodnem delu 3. poglavja.

Sedaj lahko analiziramo še časovno zahtevnost metode *ERM*. Prva **for** zanka metode *ERM* (vrstice 3–10) za vsak par učnih nalog preveri, ali zadošča kriterijem za združevanje. Časovno daleč najzahtevnejši je klic metode Estimate-errors-and-p-values (vrstica 7). Ostalo je, v primerjavi z njim, zanemarljivo. Osrednji del metode *ERM* predstavlja **while** zanka (vrstice 11–26), ki združuje kandidatne pare učnih nalog. Spomnimo, da *ERM* učne naloge požrešno združuje po principu dodajanje naprej. Na vsakem koraku **while** zanke združi po en par učnih nalog, tako se število preo-

stalih (združenih) učnih nalog vsakič zmanjša za ena. Število vseh ponovitev **while** zanke je torej navzgor omejeno s številom vseh učnih nalog T. Vgnezdena **for** zanka (vrstice 19–26) deluje enako kot prva **for** zanka (vrstice 3–10), zato je tudi pri njej v časovnem smislu pomemben le klic metode Estimate-errors-and-p-values (vrstica 23).

Poglejmo še, kako se spreminja velikost tabel podatkov združenih učnih nalog. Opazimo lahko, da je zgornja meja za velikost tabele podatkov posamezne (združene) učne naloge $i\, n_{max}$, kjer je i števec ponovitev **while** zanke. Na vsakem koraku **while** zanke namreč *ERM* združi po en par učnih nalog. V skrajnem primeru bi se lahko zgodilo, da bi *ERM* vedno združila največjo (že združeno) učno nalogo z eno od preostalih učnih nalog in tako dobila učno nalogo, katere velikost bi bila navzgor omejena z $i\, n_{max}$.

Zadnji del metode *ERM* zajema izgradnjo modelov za vse preostale (združene) učne naloge (vrstice 27–31). Njegova časovna zahtevnost je navzgor omejena z izrazom $T\,\tau_{\mathcal{B}}(T\, n_{max})$, saj je zgornja meja za velikost tabele podatkov po koncu združevanja $T\, n_{max}$. Sedaj lahko izpeljemo časovno zahtevnost metode *ERM*:

$$\begin{aligned}
\tau_{ERM}(k, n_{max}, T) &= \frac{T(T-1)}{2}\,\tau_{EEPV}(k, n_{max}) + \sum_{i=1}^{T} T\,\tau_{EEPV}(k, i\, n_{max}) + T\,\tau_{\mathcal{B}}(T\, n_{max}) = \\
&= \frac{T(T-1)}{2}\,O(k\,\tau_{\mathcal{B}}(n_{max})) + \sum_{i=1}^{T} T\,O(k\,\tau_{\mathcal{B}}(i\, n_{max})) + T\,\tau_{\mathcal{B}}(T\, n_{max}) \leq \\
&\leq O(k\,T^2\,\tau_{\mathcal{B}}(n_{max})) + O(k\,T^2\,\tau_{\mathcal{B}}(T\, n_{max})) + T\,\tau_{\mathcal{B}}(T\, n_{max}) = \\
&= O(k\,T^2\,\tau_{\mathcal{B}}(T\, n_{max})). \qquad (2.11)
\end{aligned}$$

Kot vidimo iz enačbe (2.11), časovna zahtevnost metode *ERM* linearno narašča s številom ponovitev prečnega preverjanja (k) in časovno zahtevnostjo osnovne učne metode ($\tau_{\mathcal{B}}$) ter kvadratno s številom učnih nalog (T).

3. POGLAVJE

Eksperimentalna študija obnašanja metode *ERM*

V tem poglavju bomo preučili obnašanje metode *ERM* glede na različne lastnosti MTL-problema. Naredili bomo eksperimentalno študijo na več MTL-problemih, ki bodo imeli različno število učnih nalog (lastnost iz definicije 2.1), različno število primerov, ki pripadajo posameznim učnim nalogam (lastnost iz definicije 2.2), ter različno število gruč različnih tipov učnih nalog (lastnost iz definicije 2.3). Poleg teh lastnosti nas bo zanimalo tudi, kako uspešna je metoda *ERM* pri različnih stopnjah šuma v podatkih ter kako se spreminja njeno delovanje, če znotraj nje uporabimo različne osnovne učne metode.

V razdelku 3.1 bomo predstavili rezultate eksperimentov na sintetičnih MTL-problemih učenja *Boolovih funkcij*, kjer smo lahko sami izbirali prej omenjene lastnosti MTL-problemov. Nato bo sledila predstavitev rezultatov eksperimentov na dveh realnih klasifikacijskih MTL-problemih (razdelek 3.2), kjer smo se ukvarjali z nalogo prepoznavanja ročno napisanih števk. V razdelku 3.3 pa bomo predstavili še rezultate eksperimentov na dveh realnih regresijskih MTL-problemih. Pri prvem je šlo za nalogo napovedovanja števila točk učencev pri zaključnem preizkusu znanja, pri drugem pa za napovedovanje naklonjenosti oseb nakupu določenega računalnika. Za realne MTL-probleme so njihove lastnosti dane vnaprej (z izjemo omejenega določanja števila primerov učnih nalog z izbiro razmerja delitve na učno in testno množico) ali pa so nepoznane (kot je v primeru števila gruč različnih tipov učnih nalog).

Za oceno zmogljivosti metode *ERM* smo jo v eksperimentih primerjali z naslednjimi tremi MTL-metodami:

NoMerging Ta metoda nikoli ne združi nobenih učnih nalog. Osnovna učna metoda pri gradnji modela vedno uporabi le podatke posamezne učne naloge. Ta MTL-"metoda" nam bo služila kot *izhodišče* (angl. *baseline*) pri ocenjevanju zmogljivosti metode *ERM*.

MergeAll Ta metoda združi vse učne naloge v eno gručo ne glede na to, ali so dejansko istega tipa ali ne. Tako dobijo vse učne naloge enak napovedni model. Tudi ta MTL-metoda nam bo služila kot izhodišče pri ocenjevanju zmogljivosti metode *ERM*.

Oracle Pri tej MTL-metodi predpostavljamo, da imamo *preroka* (angl. *oracle*), ki ve, katere učne naloge skupaj tvorijo gruče sorodnih učnih nalog, tako da jih lahko (brez napak) združi. Ta MTL-metoda nam bo služila kot dober indikator, kakšno zmogljivost lahko doseže metoda *ERM*.

Ob tem velja poudariti, da metoda *Oracle* ne predstavlja teoretične zgornje meje za zmogljivost. Lahko bi se namreč zgodilo, da imamo pri določenem MTL-problemu zelo podobni iskani funkciji, vendar (strogo gledano) še vedno različni. Ti dve funkciji bi predstavljali dve gruči sorodnih učnih nalog, ki jih *Oracle* ne bi nikoli združil, čeprav bi zaradi pomanjkanja podatkov z združitvijo morda dobili boljše napovedne modele.

Vse eksperimente smo večkrat ponovili ter izračunali povprečja dobljenih rezultatov in njihove 95% *konfidenčne intervale*. Ker dobljeni rezultati v resnici predstavljajo povprečne rezultate, izračunane čez vse učne naloge, lahko predpostavimo, da predstavljajo naključen vzorec normalne porazdelitve. Zaradi majhne velikosti vzorca (število ponovitev je bilo za večino eksperimentov v tem razdelku enako 10) in nepoznane variance moramo za izračun 95% konfidenčnega intervala za povprečje uporabiti naslednjo formulo [39]:

$$\left(x_{avg} - t_{\alpha/2,\,n-1}\frac{s}{\sqrt{n}}, x_{avg} + t_{\alpha/2,\,n-1}\frac{s}{\sqrt{n}}\right), \tag{3.1}$$

kjer x_{avg} predstavlja povprečje vzorca, s standardni odklon vzorca, n velikost vzorca in $t_{\alpha/2,\,n-1}$ dvostransko *kritično vrednost* t-porazdelitve z $(n-1)$ prostostnimi stopnjami pri *stopnji značilnosti* (angl. *level of significance*) α. V primeru 95% konfidenčnih intervalov je α enak 0.05.

Konfidenčni intervali so testno povezani s testiranjem statistične značilnosti.

Izrek (Povzeto po [40]). Denimo, da je ničelna hipoteza $\theta = \theta_0$ in alternativna hipoteza $\theta \neq \theta_0$. Če $100(1-\alpha)\%$ konfidenčni interval za θ ne vsebuje θ_0, potem lahko ničelno hipotezo zavrnemo pri stopnji značilnosti α.

Če se torej $100(1-\alpha)\%$ konfidenčna intervala dveh ocen parametra θ (npr. povprečne vrednosti) ne prekrivata, potem je razlika med ocenama statistično značilna pri stopnji značilnosti α. Če sta oceni dobljeni na dveh neodvisnih vzorcih, potem je stopnja značilnosti še precej manjša od α. V [41] poročajo, da neprekrivajoča 95% konfidenčna intervala za oceni, dobljeni na neodvisnih vzorcih, predstavljata statistično značilno razliko pri stopnjo značilnosti ~ 0.01. Velja opozoriti, da obratna trditev ne velja. Namreč, če se $100(1-\alpha)\%$ konfidenčna intervala dveh ocen parametra θ prekrivata, potem je razlika med ocenama lahko statistično značilna pri stopnji značilnosti α ali pa tudi ne [42, 43].

Ko v disertaciji uporabljamo izraz, da se rezultati dveh metod statistično značilno razlikujejo, s tem želimo povedati, da se 95% konfidenčna intervala za povprečji ne prekrivata, kar po [41] predstavlja statistično značilno razliko med povprečjema pri stopnji značilnosti ~ 0.01.

V okviru doktorske disertacije smo razvili *PyMTL (Python library for Multi-task learning)*, Pythonovo knjižnico, ki implementira ogrodje za MTL (vključno z generiranjem podatkov, izvajanjem eksperimentov in vizualizacijo rezultatov). Poleg tega vključuje implementacijo metode *ERM* in pomožnih MTL-metod (*NoMerging*, *MergeAll* in *Oracle*). Temelji na Pythonovih knjižnicah, popularnih v znanstvenih skupnostih, kot so *scikit-learn*, *SciPy*, *NumPy* in *matplotlib*. Je prosto dostopna pod pogoji licence *GNU General Public License*, bodisi različice 3 ali katerekoli kasnejše različice, na portalu GitHub [44].

Poskuse smo izvedli na zmogljivem prenosnem računalniku s procesorjem *Intel Core i7* s hitrostjo 2.80 GHz in pomnilnikom velikost 8 GB. Ko v nadaljevanju navajamo čase izvajanja metode *ERM*, s tem mislimo čas izvajanja na tem računalniku.

3.1 Sintetični MTL-problemi

V prvem delu eksperimentalne študije obnašanja metode *ERM* smo izvedli poskuse na sintetičnih MTL-problemih učenja Boolovih funkcij, kjer smo lahko sami izbirali lastnosti MTL-problemov (definiranih v razdelku 2.2).

Za Boolove funkcije smo se odločili, ker smo želeli izvesti poskuse na klasifikacij-

skih MTL-problemih z diskretnimi atributi, kjer lahko zelo enostavno spreminjamo želene lastnosti MTL-problemov. Hkrati pa so atributi (tj. logične spremenljivke) in namen učenja (iskanje neznane Boolove funkcije) enostavno razumljivi širšemu krogu ljudi z znanstvenega področja disertacije.

Pri generiranju Boolovih funkcij smo uporabili enak postopek kot v [45, 46]. Delovanje postopka določajo naslednji parametri:

- g: število gruč sorodnih učnih nalog,
- tg: število učnih nalog v vsaki gruči,
- a: število spremenljivk (atributov), ki nastopajo v Boolovih funkcijah,
- d: pričakovana dolžina disjunkta,
- *nls*: število različnih tabel podatkov za posamezno učno nalogo,
- n: število primerov posamezne učne naloge,
- *noise*: delež primerov, pri katerih je razred določen naključno.

Za vsako od g gruč sorodnih učnih nalog se najprej generira naključna Boolova funkcija z a spremenljivkami (atributi) v disjunktni normalni obliki. Verjetnost vključitve vsakega od atributov v posamezni disjunkt je $\frac{d}{a}$, pri čemer je verjetnost, da ga negiramo $\frac{1}{2}$. Pričakovana dolžina disjunkta je torej d. Za generiranje približno enakega števila pozitivnih in negativnih primerov je število disjunktov nastavljeno na 2^{d-2}.

Zatem se za vsako od tg učnih nalog vsake gruče generira *nls* različnih tabel podatkov velikosti n. Z njimi lahko simuliramo večkratno ponovitev eksperimenta. S parametrom *noise* se kontrolira delež primerov, pri katerih je razred določen naključno.

Na koncu se za vsako od gruč sorodnih učnih nalog, ki si delijo isto Boolovo funkcijo, generira še množica primerov z vsemi možnimi kombinacijami vrednosti atributov, ki ima potemtakem 2^a primerov. Ta množica predstavlja testne primere, na katerih se testirajo modeli vseh učnih nalog iz dane gruče.

Opisani postopek v knjižnici *PyMTL* [44] implementira funkcija *generate_boolean_data_with_complete_test_sets* v modulu *synthetic_data*.

Za vse eksperimente iz tega razdelka smo parameter a (število atributov) nastavili na 12 in parameter d (pričakovana dolžina disjunkta) na 6. Z izbiro parametra d smo se

odločili za generiranje srednje težko naučljivih Boolovih funkcij (pri vrednosti $d = 1$ bi generirali najenostavnejše Boolove funkcije, pri vrednosti $d = a$ pa najkompleksnejše). Paremeter *nls* (število različnih tabel podatkov za posamezno učno nalogo) smo v vseh eksperimentih nastavili na 10.

Zaradi praktičnih razlogov smo se v glavnini eksperimentov omejili le na eno osnovno učno metodo, na koncu (podrazdelek 3.1.5) pa smo primerjali tudi obnašanje MTL-metod z različnimi osnovnimi učnimi metodami. V podrazdelkih 3.1.1–3.1.4 smo znotraj vseh štirih MTL-metod (*ERM*, *NoMerging*, *MergeAll* in *Oracle*) uporabili *metodo podpornih vektorjev* (angl. *support vector machine*, *SVM*), kot je implementirana v knjižnici za strojno učenje *scikit-learn* [47]. Vrednosti njenih parametrov so bile: *kernel* = poly (tip jedrne funkcije), *degree* = 3 (stopnja polinomskega jedra), *coef0* = 1 (vrednost konstantnega člena polinomskega jedra). Ostali parametri so imeli privzete vrednosti.

Parametra metode *ERM* sta imela v vseh eksperimentih enako vrednost: $\eta = 10$ (minimalno število primerov posamezne učne naloge, da kandidatni par učnih nalog zadosti kriteriju za združevanje (1)), $k = 5$ (število ponovitev prečnega preverjanja).

Zgrajene modele smo ocenjevali s *ploščino pod ROC-krivuljo* (angl. *area under ROC curve*, *AUC*), standardno mero za ocenjevanje kvalitete klasifikacijskih modelov. Uporabi *klasifikacijske točnosti* (angl. *classification accuracy*, *CA*), ki je prav tako standardna mera za ocenjevanje kvalitete klasifikacijskih modelov, smo se zaradi različnih porazdelitev vrednosti razreda za različne MTL-probleme izognili.

3.1.1 Spreminjanje števila učnih nalog

Prvi del eksperimentalne študije na sintetičnih MTL-problemih je bilo opazovanje in primerjava MTL-metod ob spreminjanju števila učnih nalog (lastnost iz definicije 2.1). Pri generiranju Boolovih funkcij smo (poleg vrednosti, ki so podane v začetku razdelka 3.1) izbrali naslednje vrednosti parametrov:

- n (število primerov posamezne učne naloge): 50,
- g (število gruč sorodnih učnih nalog): 5,
- *noise* (delež primerov, pri katerih je razred določen naključno): 0.0.

Postopek generiranja Boolovih funkcij smo ponovili 4-krat, vsakič z drugim naključnim semenom. Tako smo dobili štiri MTL-probleme, na katerih smo izvedli niz

eksperimentov za različne vrednosti parametra *tg* (število učnih nalog v vsaki gruči). Uporabili smo vrednosti: 1, 2, 5, 10 in 20. Čas enega zagona metode *ERM* je bil za *tg* = 1 približno 1 s, za *tg* = 20 pa okoli 3000 s.

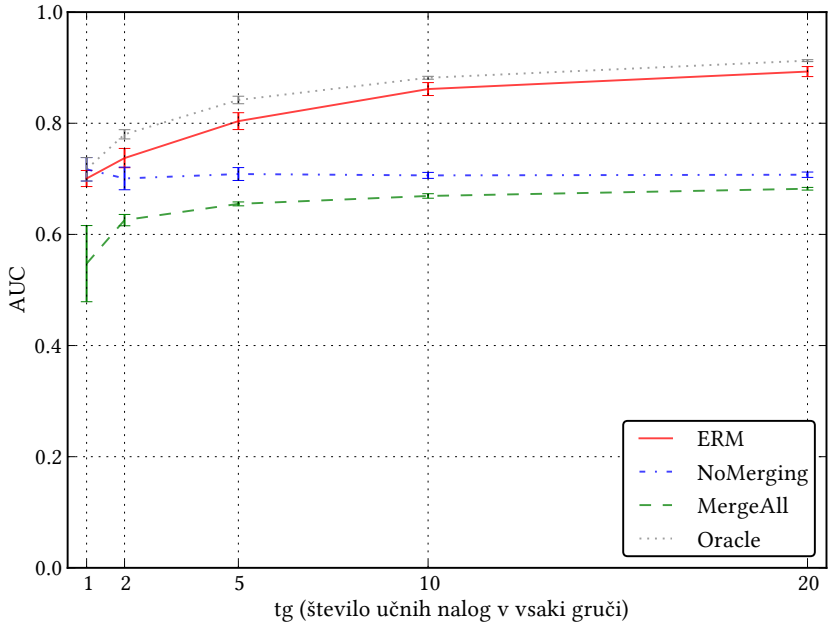
Za oceno AUC za celoten MTL-problem smo izračunali povprečje čez vse učne naloge. Kot je bilo razloženo v uvodnem delu razdelka 3.1, smo za vsako učno nalogo posameznega MTL-problema generirali deset različnih tabel podatkov, s čimer smo simulirali 10-kratno ponovitev eksperimenta. Na koncu smo izračunali povprečja in 95% konfidenčne intervale vseh tako dobljenih povprečnih vrednosti AUC. Rezultati so prikazani na sliki 3.1.

Pričakovano je najboljše rezultate pri vseh MTL-problemih dosegla metoda *Oracle*. Izmed preostalih treh metod se je daleč najbolje odrezala metoda *ERM*, katere AUC se je s povečevanjem števila učnih nalog v vsaki gruči polagoma približeval AUC vrednosti metode *Oracle*. Pri prvih treh MTL-problemih (1–3) se je metoda *NoMerging* odrezala bolje od metode *MergeAll*. Pri zadnjem MTL-problemu (4) pa je na začetku višje vrednosti AUC dosegala metoda *NoMerging*, vendar sta se s povečevanjem števila učnih nalog v vsaki gruči njuni krivulji približevali in na koncu je metoda *MergeAll* dosegla za malenkost večji AUC od metode *NoMerging*.

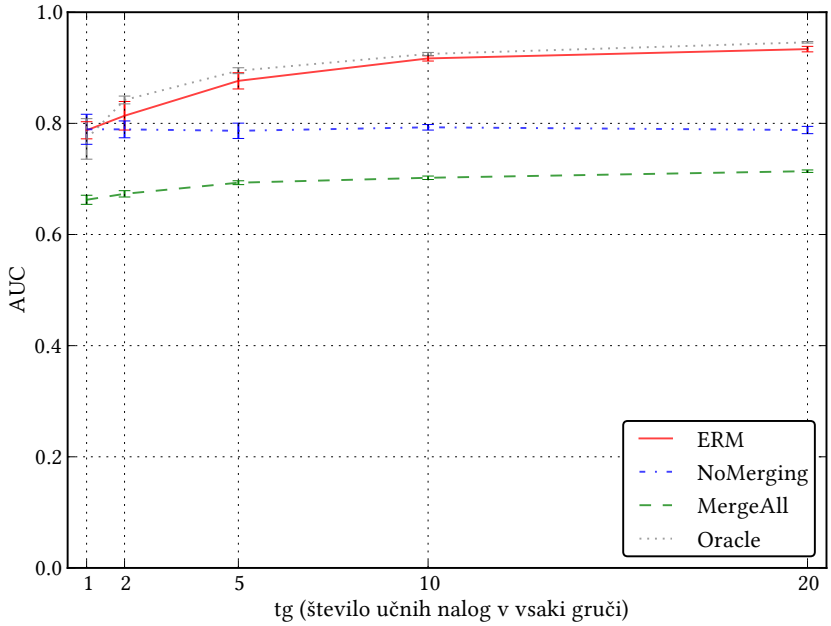
Po pričakovanjih se vrednost AUC metode *NoMerging* s povečevanjem števila učnih nalog v vsaki gruči ne spreminja, saj ta metoda nikoli ne združi nobenih učnih nalog in ji zato večjo število učnih nalog v vsaki gruči nič ne koristi. Začetni nihaj krivulj metode *NoMerging* je posledica majhnega števila različnih tabel podatkov, saj ima takrat posamezna gruča le eno oz. dve učni nalogi s po 50 primeri. Temu pritrjujejo tudi večji 95% konfidenčni interval za povprečja.

Pri vseh MTL-problemih, razen pri MTL-problemu 4, so že pri eni učni nalogi v vsaki gruči vrednosti AUC metod *Oracle*, *ERM* in *NoMerging* statistično značilno boljše od metode *MergeAll*. Rezultat metod *Oracle* in *NoMerging* v primerjavi z metodo *MergeAll* potrjuje, da pri združevanju podatkov vseh učnih nalog pride do negativnega prenosa. Rezultat metode *ERM*, ki je primerljiv z metodama *Oracle* in *NoMerging*, pa dokazuje, da *ERM* (pravilno) ne združuje učnih nalog iz različnih gruč.

Postopno povečevanje AUC metode *ERM* nam prikazuje, da ima s povečevanjem števila učnih nalog v vsaki gruči oz. celotnega števila učnih nalog (lastnost iz definicije 2.1) metoda *ERM* večji potencial za združevanje podatkov, saj lahko pri združevanju podatkov izbira med več učnimi nalogami. Nekoliko nižji AUC v primerjavi z zgornjo mejo, ki jo daje metoda *Oracle*, nakazuje na konzervativnost kriterijev za združevanje, podanih v podrazdelku 2.3.2.

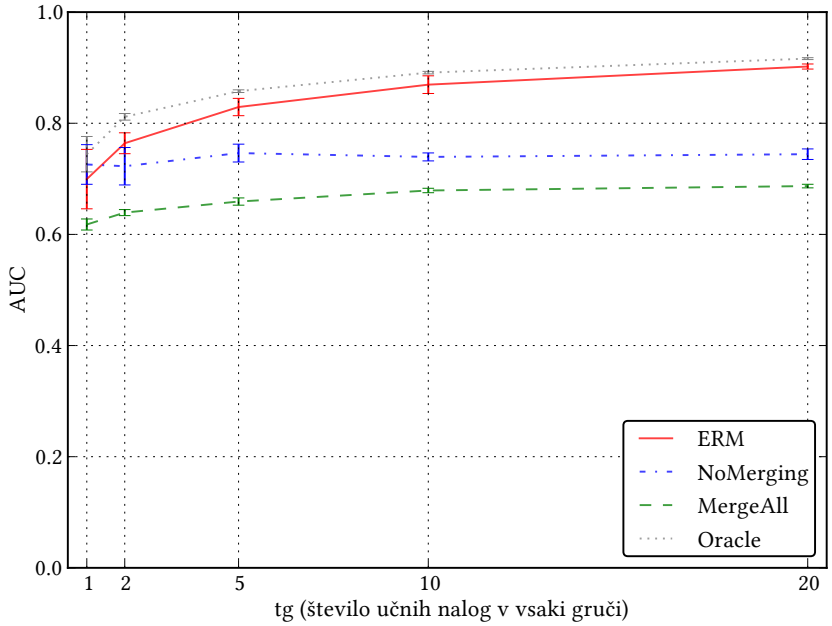


(a) MTL-problem 1

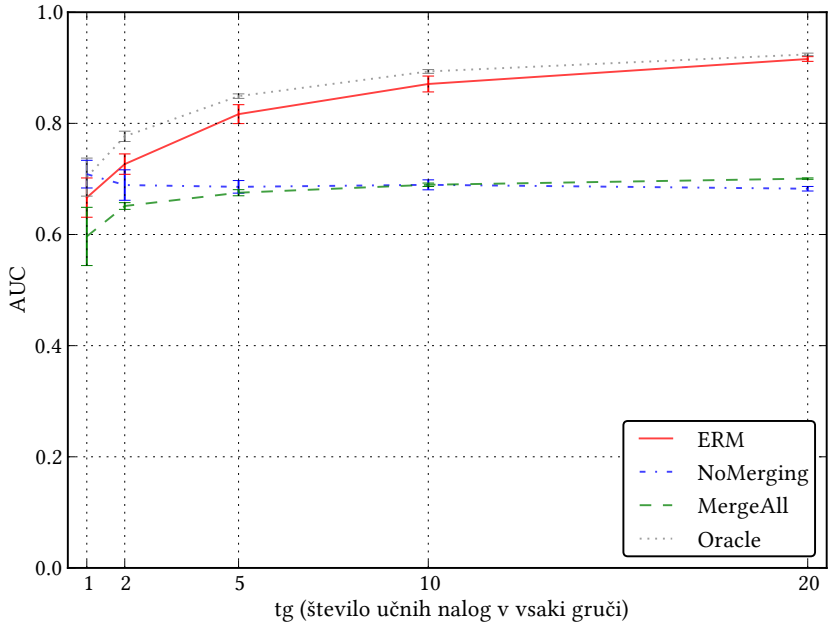


(b) MTL-problem 2

Slika 3.1: Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL- metode ob spreminjanju števila učnih nalog (lastnost iz definicije 2.1). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. *Intervali napake* (angl. *error bars*) prikazujejo 95% konfidenčne intervale za povprečja.



(c) MTL-problem 3



(d) MTL-problem 4

Slika 3.1 (nadalj.): Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila učnih nalog (lastnost iz definicije 2.1). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

Zanimivo odkritje je povečevanje vrednosti AUC metode *MergeAll* s povečevanjem števila učnih nalog v vsaki skupini. Postopna upočasnitev tega povečevanja nakazuje, da je verjetni razlog za ta pojav majhno število primerov na začetku ($50 \times 5 \times 1 = 250$) v primerjavi z ogromnim številom primerov na koncu ($50 \times 5 \times 20 = 5000$). Povečano število primerov osnovni učni metodi (v tem primeru SVM) omogoča izgradnjo boljšega splošnega modela (istega za vse učne naloge), kljub temu da je naučen na podatkih učnih nalog, ki pripadajo različnim gručam.

3.1.2 Spreminjanje števil primerov posameznih učnih nalog

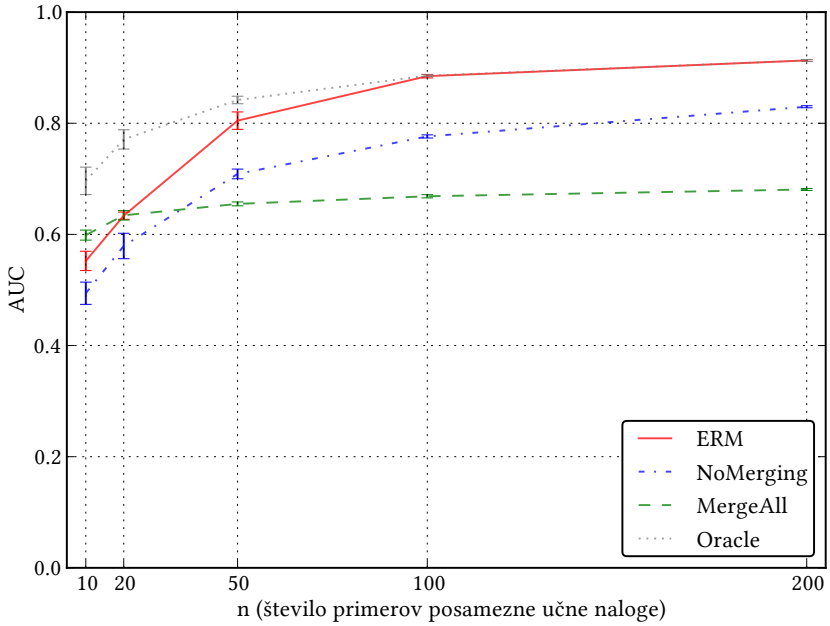
V drugem delu eksperimentalne študije smo opazovali in primerjali delovanje MTL-metod na sintetičnih MTL-problemih ob spreminjanju števil primerov posameznih učnih nalog (lastnost iz definicije 2.2). Pri naših eksperimentih so imele vseh učne naloge enako število primerov, zato bomo odslej govorili o številu primerov posameznih učnih nalog, v splošnem pa enako število primerov za vse učne naloge ni potrebno. Pri generiranju Boolovih funkcij smo (poleg vrednosti, ki so podane v začetku razdelka 3.1) izbrali naslednje vrednosti parametrov:

- *g* (število gruč sorodnih učnih nalog): 5,
- *tg* (število učnih nalog v vsaki gruči): 5,
- *noise* (delež primerov, pri katerih je razred določen naključno): 0.0.

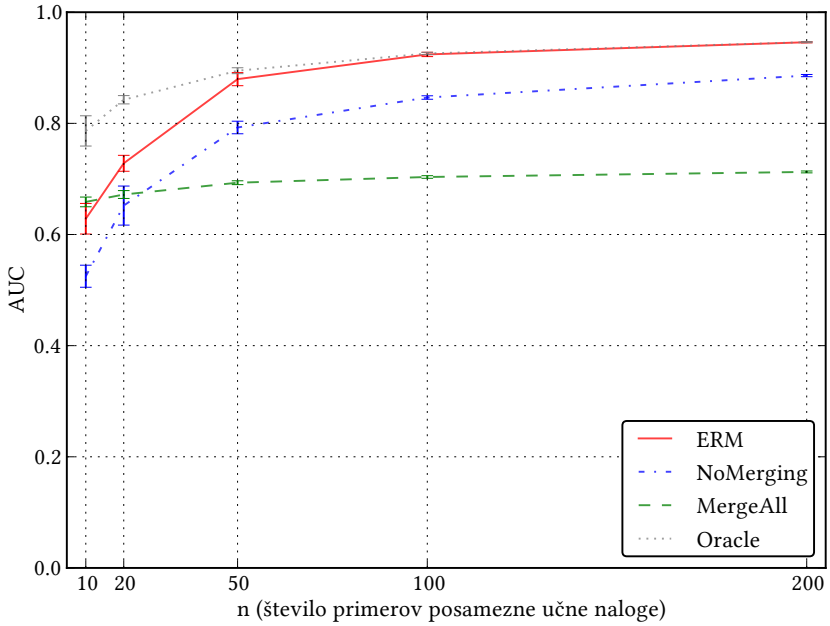
Postopek generiranja Boolovih funkcij smo ponovili 4-krat, vsakič z drugim naključnim semenom. Tako smo dobili štiri MTL-probleme, na katerih smo izvedli niz eksperimentov za različne vrednosti parametra *n* (število primerov posamezne učne naloge). Uporabili smo vrednosti: 10, 20, 50, 100 in 200. Čas enega zagona metode *ERM* je bil za $n = 10$ približno 15 s, za $n = 200$ pa okoli 700 s.

Za oceno AUC za celoten MTL-problem smo izračunali povprečje čez vse učne naloge. Kot je bilo razloženo v uvodnem delu razdelka 3.1, smo za vsako učno nalogo posameznega MTL-problema generirali deset različnih tabel podatkov, s čimer smo simulirali 10-kratno ponovitev eksperimenta. Na koncu smo izračunali povprečja in 95% konfidenčne intervale vseh tako dobljenih povprečnih vrednosti AUC. Rezultati so prikazani na sliki 3.2.

Tudi pri teh poskusih je po pričakovanjih pri vseh MTL-problemih najboljše rezultate dajala metoda *Oracle*. Pri ostalih treh metodah pa so bili rezultati odvisni od

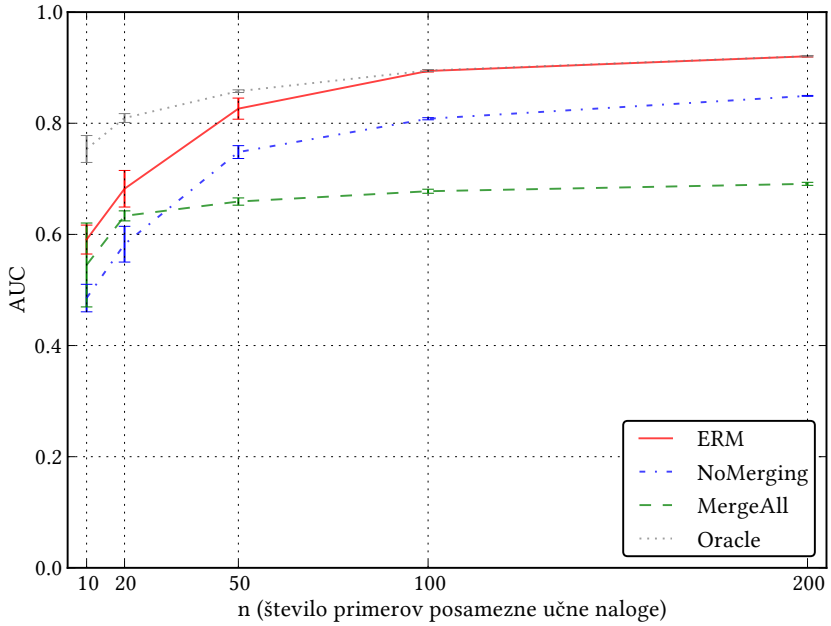


(a) MTL-problem 1

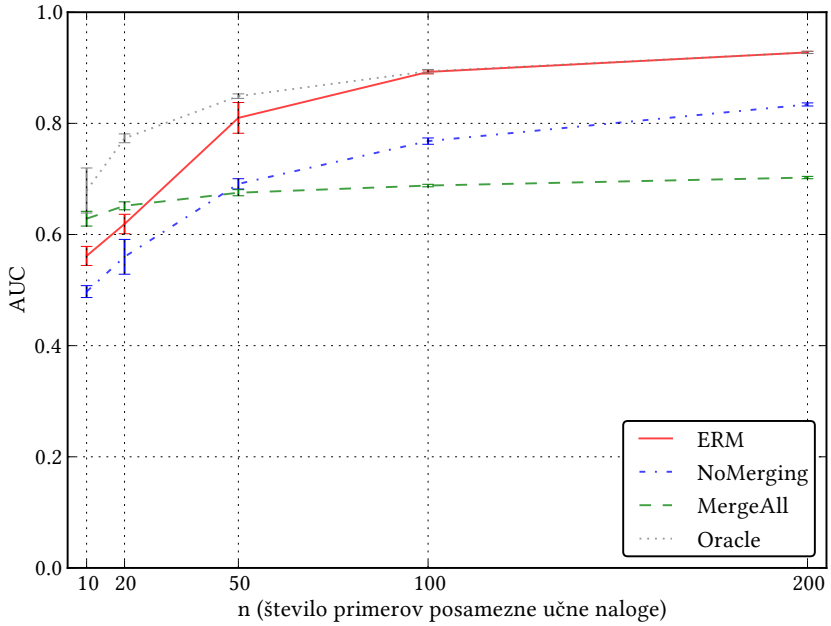


(b) MTL-problem 2

Slika 3.2: Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila primerov posameznih učnih nalog (lastnost iz definicije 2.2). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.



(c) MTL-problem 3



(d) MTL-problem 4

Slika 3.2 (nadalj.): Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila primerov posameznih učnih nalog (lastnost iz definicije 2.2). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

števila primerov posameznih učnih nalog, zato jih bomo v nadaljevanju podrobno analizirali.

Pri 10 primerih na posamezno učno nalogo je bila metoda *MergeAll* praviloma boljša od preostalih dveh metod. Pri MTL-problemih 1 in 4 so njeni rezultati statistično značilno boljši od metod *ERM* in *NoMerging*. Pri MTL-problemu 2 je prav tako boljša od metod *ERM* in *NoMerging*, pri MTL-problemu 3 pa je od nje boljša metoda *ERM*, vendar se v obeh primerih njuna 95% konfidenčna intervala prekrivata. Na drugo mesto (izmed preostalih treh) se praviloma uvršča metoda *ERM*, ki pri vseh MTL-problemih dosega statistično značilno večje AUC vrednosti od metode *NoMerging*.

Pri vseh štirih MTL-problemih se s povečevanjem števila primerov na posamezno učno nalogo vrstni red metod obrne, tako da najboljša postane metoda *ERM*, sledi ji metoda *NoMerging*, metoda *MergeAll* pa pristane na zadnjem mestu. Kdaj se ta obrat zgodi, je odvisno od MTL-problema. Pri MTL-problemu 1 metodi *ERM* in *NoMerging* presežeta AUC metode *MergeAll* med 20 in 50 primeri. Pri MTL-problemu 2 metoda *ERM* preseže AUC metode *MergeAll* že med 10 in 20 primeri, medtem ko metoda *NoMerging* za to potrebuje med 20 in 50 primerov. Pri MTL-problemu 3 ima metoda *ERM* že od začetka večji AUC od metode *MergeAll*, metoda *NoMerging* pa njen AUC preseže med 20 in 50 primeri. Pri MTL-problemu 4 pa metodi *ERM* in *NoMerging* ponovno presežeta AUC metode *MergeAll* med 20 in 50 primeri.

S povečevanjem števila primerov se razlika med AUC vrednostma metod *ERM* in *NoMerging* do števila 100 primerov na posamezno učno nalogo povečuje, potem pa se pri 200 primerih razlika rahlo zmanjša. To je posledica tega, da je takrat z metodo *ERM* že dosežena maksimalna vrednost AUC z dano osnovno učno metodo, medtem ko metodi *NoMerging*, kjer učenje modelov poteka zgolj na posameznih podatkih učnih nalog, povečevanje števila primerov zelo koristi.

Pri vseh štirih MTL-problemih je AUC metode *ERM* hitro naraščal in se približeval AUC vrednosti metode *Oracle*. Od števila 100 primerov na posamezno učno nalogo naprej je metoda *ERM* dosegala enako dobre rezultate kot metoda *Oracle*. To kaže, da je povečanje števila primerov posamezne učne naloge za *ERM* zelo koristno, saj tako lažje odkrije in združi (le) sorodne učne naloge.

Vrednosti AUC metode *MergeAll* s povečevanjem števila primerov rahlo naraščajo, kar je verjetno posledica majhnega števila primerov na začetku ($10 \times 5 \times 5 = 250$) v primerjavi z ogromnim številom primerov na koncu ($200 \times 5 \times 5 = 5000$). Povečano število primerov osnovni učni metodi (v tem primeru SVM) omogoča izgradnjo

boljšega splošnega modela (istega za vse učne naloge), kljub temu da je naučen na podatkih učnih nalog, ki pripadajo različnim gručam.

3.1.3 Spreminjanje števila gruč različnih tipov učnih nalog

V tretjem delu eksperimentalne študije na sintetičnih MTL-problemih smo opazovali in primerjali zmogljivost MTL-metod ob spreminjanju števila gruč sorodnih učnih nalog (lastnost iz definicije 2.3). Pri generiranju Boolovih funkcij smo (poleg vrednosti, ki so podane v začetku razdelka 3.1) izbrali naslednje vrednosti parametrov:

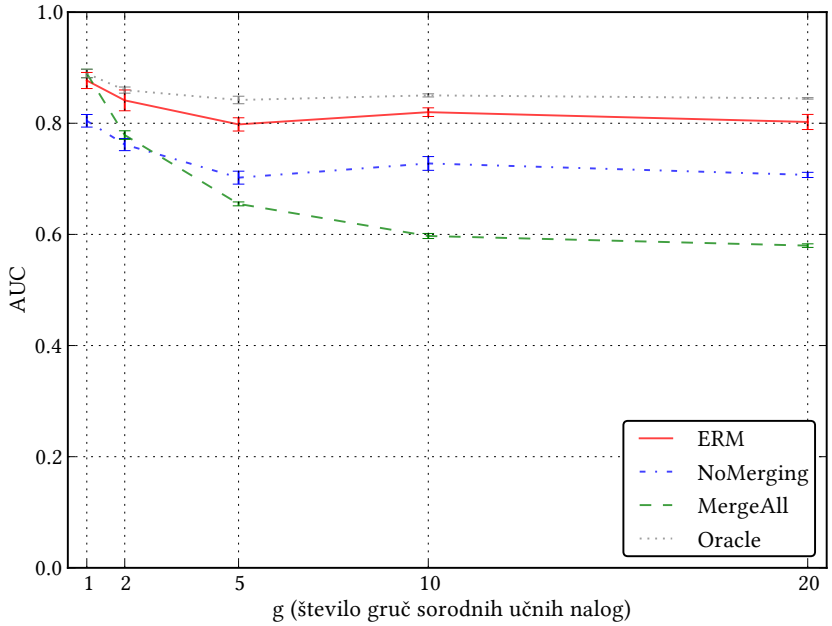
- *n* (število primerov posamezne učne naloge): 50,
- *tg* (število učnih nalog v vsaki gruči): 5,
- *noise* (delež primerov, pri katerih je razred določen naključno): 0.0.

Postopek generiranja Boolovih funkcij smo ponovili 4-krat, vsakič z drugim naključnim semenom. Tako smo dobili štiri MTL-probleme, na katerih smo izvedli niz eksperimentov za različne vrednosti parametra g (število gruč sorodnih učnih nalog). Uporabili smo vrednosti: 1, 2, 5, 10 in 20. Čas enega zagona metode *ERM* je bil za $g = 1$ približno 1 s, za $g = 20$ pa okoli 1000 s.

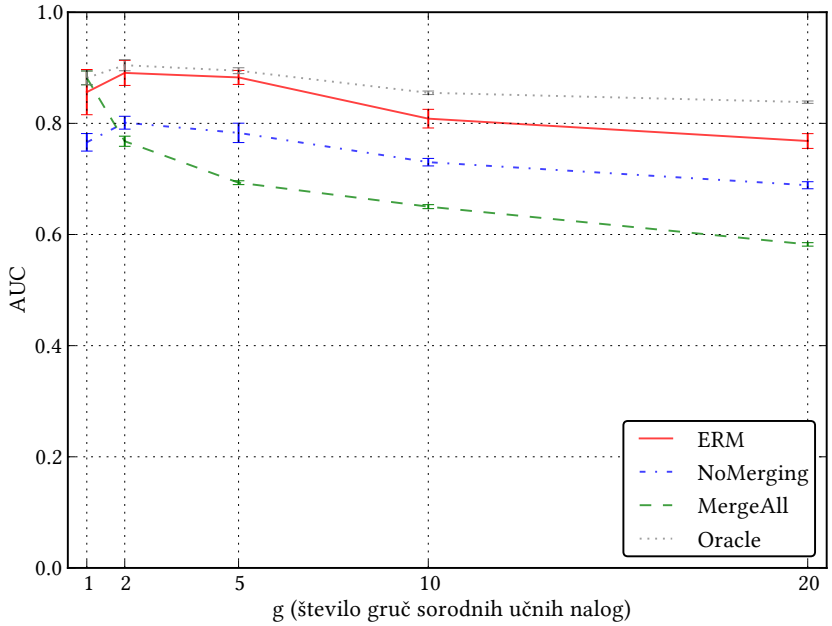
Za oceno AUC za celoten MTL-problem smo izračunali povprečje čez vse učne naloge. Kot je bilo razloženo v uvodnem delu razdelka 3.1, smo za vsako učno nalogo posameznega MTL-problema generirali deset različnih tabel podatkov, s čimer smo simulirali 10-kratno ponovitev eksperimenta. Na koncu smo izračunali povprečja in 95% konfidenčne intervale vseh tako dobljenih povprečnih vrednosti AUC. Rezultati so prikazani na sliki 3.3.

Pri vseh MTL-problemih se po pričakovanjih ponovno najbolje odreže metoda *Oracle*. Sledi ji metoda *ERM*, ostali metodi, *NoMerging* in *MergeAll*, pa se izmenjujeta na 3. oz. 4. mestu glede na število gruč sorodnih učnih nalog.

Nenavadnost, ki jo opazimo pri tej seriji eksperimentov, je začetno valovanje krivulj AUC metode *Oracle*. Pričakovali bi namreč, da bodo njene krivulje več ali manj konstantne, saj se število učnih nalog posamezne gruče in število primerov posamezne učne naloge ni spreminjalo. Natančnejši premislek razkrije, da je začetno valovanje posledica različnih stopenj težavnosti naučljivosti naključno generiranih Boolovih funkcij. Vsaki gruči sorodnih učnih nalog namreč pripada po ena naključno generirana Boolova funkcija. Tako je na začetku povprečna vrednost AUC izračunana le iz

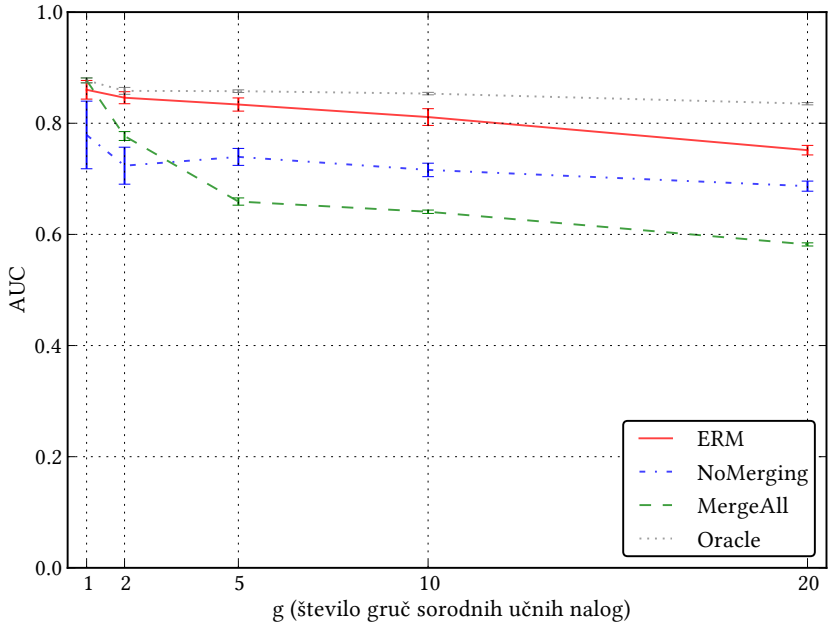


(a) MTL-problem 1

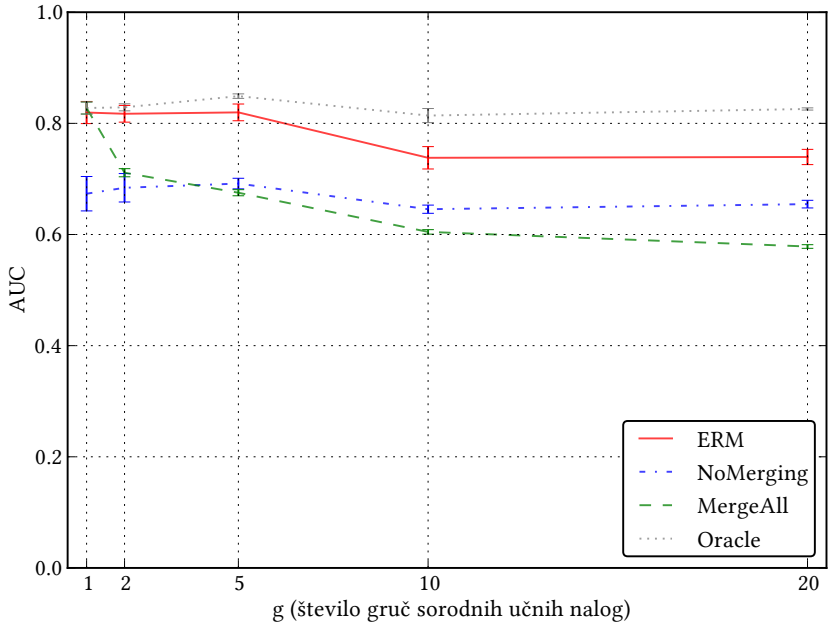


(b) MTL-problem 2

Slika 3.3: Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila gruč sorodnih učnih nalog (lastnost iz definicije 2.3). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.



(c) MTL-problem 3



(d) MTL-problem 4

Slika 3.3 (nadalj.): Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila gruč sorodnih učnih nalog (lastnost iz definicije 2.3). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

modelov za eno samo naključno generirano Boolovo funkcijo, kasneje pa se računa povprečje modelov za večje število (2, 5, 10 in 20) naključno generiranih Boolovih funkcij, kar pomeni da se različne stopnje težavnosti naučljivosti Boolovih funkcij približno izravnajo.

Na podoben način lahko pojasnimo začetno valovanje krivulj AUC metod *ERM* in *NoMerging*. Metoda *NoMerging* za vsako učno nalogo zgradi svoj model, pri čemer še vedno velja, da kvaliteta modelov zavisi od stopnje težavnosti naučljivosti Boolove funkcije, ki je na začetku ena sama, v nadaljevanju pa se njihovo število povečuje. To povzroči, da se povprečne vrednosti AUC s povečevanjem števila gruč postopoma izravnajo. Obnašanje metode *ERM* na začetku sledi metodi *Oracle* (kar se še posebej lepo vidi na sliki 3.3(b)), kar pomeni, da tudi v tem primeru velja, da pri manjšem številu gruč sorodnih učnih nalog vrednosti AUC zaradi različnih stopenj težavnosti naučljivosti Boolovih funkcij bolj nihajo, s povečevanjem števila gruč sorodnih učnih nalog, pa se vrednosti AUC polagoma izravnajo.

Opazimo lahko, da se razlika med metodama *ERM* in *Oracle* s povečevanjem števila gruč sorodnih učnih nalog povečuje. To nakazuje šibkost metode *ERM*, ki ob povečanem številu gruč težje poišče gruče sorodnih učnih nalog, ki ustrezajo prvotnim gručam, kar se kaže v padcu njene zmogljivosti. Torej tretja lastnost MTL-problemov, podana v definiciji 2.3, negativno vpliva na zmogljivost metode *ERM*.

Vrednosti AUC metode *MergeAll* so na začetku pričakovano enakovredne vrednostim metod *Oracle* in *ERM*, saj imamo opravka z eno samo gručo sorodnih učnih nalog, ki jih je seveda smiselno združiti. V nadaljevanju pa vrednosti AUC metode *MergeAll* hitro padajo, tako da je pri petih gručah sorodnih učnih nalog metoda *MergeAll* že najslabša in tako ostane tudi ob nadaljnjem povečevanju števila gruč. Ta rezultat je pričakovan, saj se s povečevanjem števila gruč sorodnih učnih nalog povečuje število naključno generiranih Boolovih funkcij, ki se uporabljajo za generiranje podatkov, kar posledično, kljub povečanemu številu primerov, otežuje gradnjo skupnega modela z dano osnovno učno metodo.

3.1.4 Spreminjanje stopnje šuma v podatkih

V četrtem delu eksperimentalne študije smo opazovali in primerjali delovanje MTL-metod na sintetičnih MTL-problemih ob spreminjanju stopnje šuma v podatkih učnih nalog. Pri generiranju Boolovih funkcij smo (poleg vrednosti, ki so podane v začetku razdelka 3.1) izbrali naslednje vrednosti parametrov:

- n (število primerov posamezne učne naloge): 50,
- g (število gruč sorodnih učnih nalog): 5,
- tg (število učnih nalog v vsaki gruči): 5.

Postopek generiranja Boolovih funkcij smo ponovili 4-krat, vsakič z drugim naključnim semenom. Tako smo dobili štiri MTL-probleme, na katerih smo izvedli niz eksperimentov za različne vrednosti parametra *noise* (delež primerov, pri katerih je razred določen naključno). Uporabili smo vrednosti: 0.0, 0.1, 0.2, 0.3, 0.4 in 0.5. Čas enega zagona metode *ERM* je bil za vse vrednosti parametra *noise* okoli 60 s.

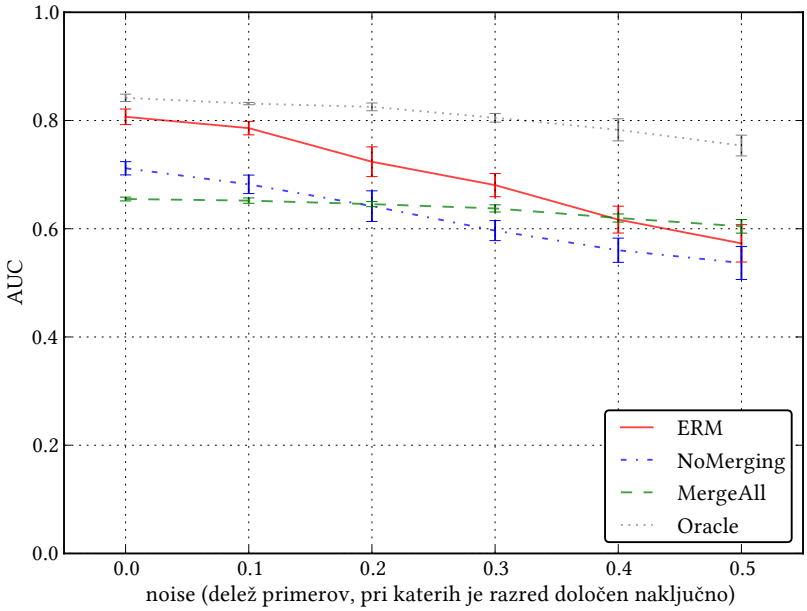
Za oceno AUC za celoten MTL-problem smo izračunali povprečje čez vse učne naloge. Kot je bilo razloženo v uvodnem delu razdelka 3.1, smo za vsako učno nalogo posameznega MTL-problema generirali deset različnih tabel podatkov, s čimer smo simulirali 10-kratno ponovitev eksperimenta. Na koncu smo izračunali povprečja in 95% konfidenčne intervale vseh tako dobljenih povprečnih vrednosti AUC. Rezultati so prikazani na sliki 3.4.

Po pričakovanjih se je najbolje odrezala metoda *Oracle*. Sledijo ji ostale tri metode, katerih vrsti red se s povečevanjem stopnje šuma v podatkih učnih nalog spreminja.

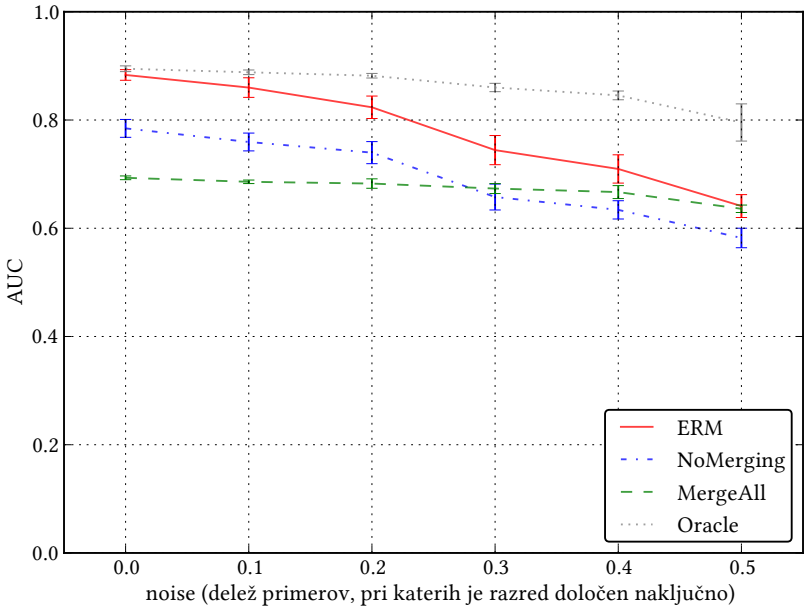
Iz rezultatov metode *Oracle* lahko vidimo, kako stopnja šuma vpliva na zgornjo mejo zmogljivosti z uporabo dane osnovno učne metode (v tem primeru SVM). Vrednost AUC se pri manjših stopnjah šuma počasi zmanjšuje, pri večjih stopnjah šuma pa je njen padec hitrejši. Na koncu (za *noise* = 0.5) je vrednost AUC pri vseh MTL-problemih še vedno relativno visoka (≥ 0.75).

Sorodno obnašanje imajo tudi krivulje AUC metode *MergeAll*. Vrednosti AUC se pri manjših stopnjah šuma zelo počasi zmanjšujejo, pri večjih stopnjah šuma pa je padec hitrejši, vendar počasnejši od padca vrednosti AUC pri metodi *Oracle*. Razlog za ta zanimiv pojav je v tem, da ima metoda *MergeAll* že v osnovi (brez šuma v podatkih) težko delo pri gradnji modela, saj učenje poteka na združenih podatkih vseh petih gruč sorodnih učnih nalog. Vhodni podatki za osnovno učno metodo si torej že brez šuma nasprotujejo, tako da postopno dodajanje šuma le še nekoliko dodatno oteži proces učenja.

Padanje vrednosti AUC pri povečevanju stopnje šuma za metodi *ERM* in *NoMerging* je bilo pričakovano, zanimivo pa je podrobno pogledati njegovo dinamiko. Razlika med vrednostmi AUC metod *Oracle* in *ERM* se s povečevanjem stopnje šuma vztrajno povečuje. Na začetku (za *noise* ≤ 0.1) je *ERM* po absolutnih vrednostih AUC

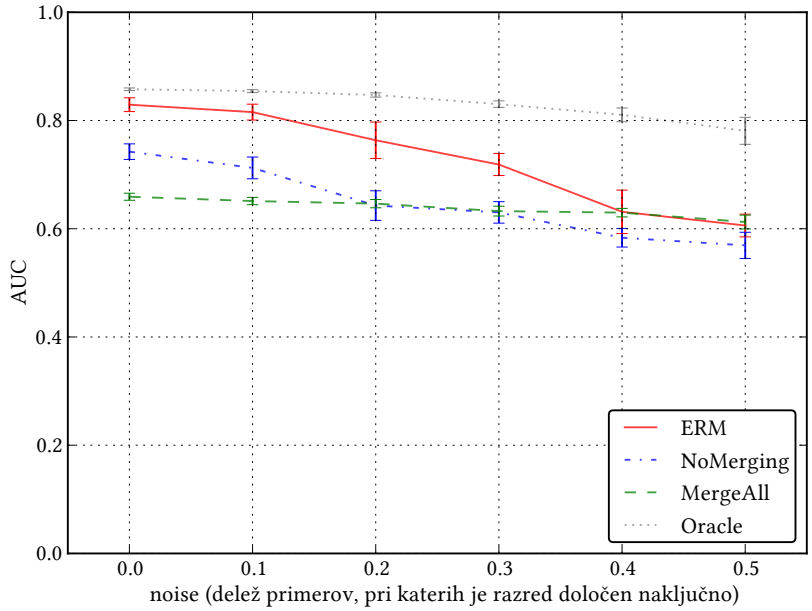


(a) MTL-problem 1

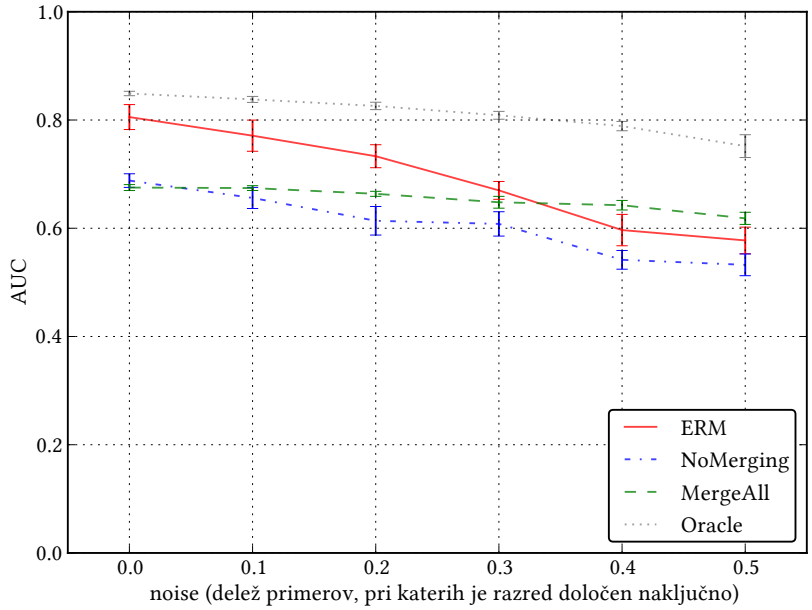


(b) MTL-problem 2

Slika 3.4: Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju stopnje šuma v podatkih učnih nalog (vrednost parametra *noise* označuje delež primerov, pri katerih je razred določen naključno). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.



(c) MTL-problem 3



(d) MTL-problem 4

Slika 3.4 (nadalj.): Rezultati poskusov za različne sintetične MTL-probleme učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju stopnje šuma v podatkih učnih nalog (vrednost parametra *noise* označuje delež primerov, pri katerih je razred določen naključno). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

še relativno blizu (z izjemo MTL-problema 4) metodi *Oracle*. Od tam naprej pa začne njena krivulja AUC hitro padati. Za vrednosti $noise \leq 0.3$ je pri vseh MTL-problemih (z izjemo MTL-problema 4 za $noise = 0.3$) metoda *ERM* še statistično značilno boljša od metod *NoMerging* in *MergeAll*, za večje vrednosti *noise* pa je v nekaterih primerih slabša (za MTL-problema 1 in 4), v enem primeru enaka (za MTL-problem 3) in v enem primeru boljša (za MTL-problem 2) od metode *MergeAll*. Iz rezultatov lahko torej ugotovimo, da pri stopnji šuma $noise = 0.5$ metoda *ERM* že povsem odpove in deluje podobno ali slabše od metode *MergeAll*.

Pri metodi *NoMerging* je dinamika padanja vrednosti AUC nekoliko drugačna. Pri njej že osnovni AUC ni tako visok kot pri metodi *ERM*. Hitrost padanja AUC je zelo različna. Pri MTL-problemu 1 njen AUC pade pod AUC metode *MergeAll* med vrednostma *noise* 0.1 in 0.2, pri MTL-problemu 2 pa med vrednostma 0.2 in 0.3. Pri MTL-problemu 3 sta nekaj časa (za vrednosti *noise* med 0.2 in 0.3) vrednosti AUC obeh metod, *NoMerging* in *MergeAll*, enaki, potem pa vrednosti AUC prve padejo pod vrednosti druge. Pri MTL-problemu 4 AUC metode *NoMerging* pade pod AUC metode *MergeAll* že med vrednostma *noise* 0.0 in 0.1. Tudi ta rezultat je pričakovan, saj ima metoda *NoMerging* za vsako učno nalogo na voljo le 50 primerov. S povečevanjem stopnje šuma v podatkih postaja praktično nemogoče zgraditi napovedni model, ki je boljši od naključnega napovedovanja.

3.1.5 Spreminjanje osnovne učne metode znotraj *ERM*

V zadnjem delu eksperimentalne študije na sintetičnih MTL-problemih smo opazovali in primerjali delovanje MTL-metod ob spreminjanju osnovne učne metode, ki se uporablja znotraj le-teh.

Zaradi praktičnosti smo eksperimente omejili le na MTL-problem 2, na katerem smo izvedli vse eksperimente kot v prejšnjih štirih podrazdelkih (3.1.1–3.1.4), vsakič z drugo osnovno učno metodo. Za primerjavo smo poleg SVM (opisane v uvodnem delu razdelka 3.1) uporabili še *odločitveno drevo* (angl. *decision tree*), kot je implementirano v paketu za strojno učenje in podatkovno rudarjenje *Orange* [48], ter metodo k-NN, kot je implementirana v knjižnici za strojno učenje *scikit-learn* [47]. Za ta izbor osnovnih učnih metod smo se odločili zato, ker smo želeli primerjati nekaj enostavnih in po naravi delovanja različnih metod nadzorovanega učenja, ki so se zmožne naučiti dobrih modelov za problem učenja Boolovih funkcij.

Pri odločitvenem drevesu smo parameter *min_instances* (velikost podmnožice pri-

merov v vozlišču, pri kateri algoritem preneha z nadaljnjim grajenjem drevesa) nastavili na 10 ter parameter *same_majority_pruning* (naknadno rezanje dreves z odstranjevanjem poddreves, katerih listi vse primere klasificirajo v isti razred) na *true*. Ostali parametri so imeli privzete vrednosti.

Pri metodi k-NN smo parameter k (število najbližnjih sosedov, ki jih metoda upošteva pri določanju razreda) nastavili na 10, vrednosti ostalih parametrov pa so imele privzete vrednosti.

3.1.5.1 Spreminjanje števila učnih nalog

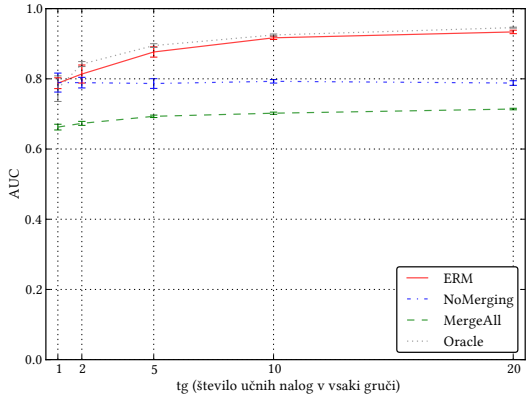
Prvi poskus je zajemal opazovanje obnašanja MTL-metod ob spreminjanju števila učnih nalog za različne osnovne učne metode.

Za ta poskus smo uporabili enake parametre za generiranje Boolovih funkcij, kot so podane v podrazdelku 3.1.1. Prav tako smo uporabili enak način izračuna povprečnih vrednosti AUC in njihovih 95% konfidenčnih intervalov. Rezultati so podani na sliki 3.5. S SVM je bil čas enega zagona metode *ERM* za $tg = 1$ približno 1 s, za $tg = 20$ pa okoli 3000 s. Z odločitvenim drevesom je bil čas enega zagona metode *ERM* za $tg = 1$ približno 1 s, za $tg = 20$ pa okoli 600 s. S k-NN je bil čas enega zagona metode *ERM* za $tg = 1$ približno 1 s, za $tg = 20$ pa okoli 300 s.

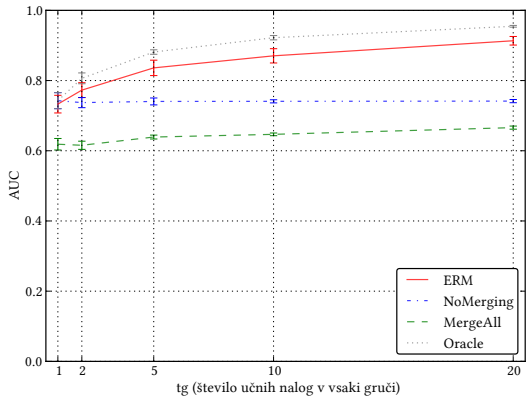
Opazimo lahko, da so trendi naraščanja krivulj AUC ne glede na izbrano osnovno učno metodo (kvalitativno gledano) zelo podobni. Absolutne vrednosti AUC se sicer za različne osnovne učne metode nekoliko razlikujejo, vendar razmerja med MTL-metodami (tj. njihovi rangi) vseskozi ostajajo enaka.

Krivulje AUC so za metodi *NoMerging* in *MergeAll* za vse tri osnovne učne metode skoraj povsem identične, pri metodi *ERM* pa so nekoliko različne. Pri SVM se je *ERM* zelo hitro približala metodi *Oracle* in tam ob povečevanju števila učnih nalog tudi ostala. Pri odločitvenem drevesu je bil AUC metode *ERM* vseskozi statistično značilno nižji od metode *Oracle*. Pri k-NN pa se je s povečevanjem števila učnih nalog razlika med AUC metode *ERM* in *Oracle* rahlo zmanjševala, vendar je do konca ostala statistično značilna.

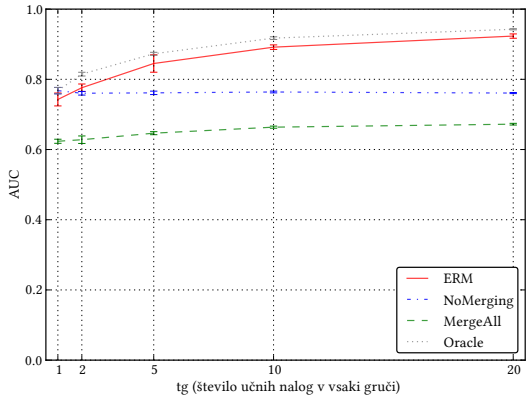
Iz rezultatov lahko torej ugotovimo, da se *ERM* ob spreminjanju števila učnih nalog odlično odreže ne glede na uporabljeno osnovno učno metodo.



(a) SVM



(b) odločitveno drevo



(c) k-NN

Slika 3.5: Rezultati poskusov za različne osnovne učne metode na izbranem sintetičnem MTL-problemu učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila učnih nalog (lastnost iz definicije 2.1). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

3.1.5.2 Spreminjanje števila primerov posameznih učnih nalog

Drugi poskus je zajemal opazovanje obnašanja MTL-metod ob spreminjanju števila primerov posameznih učnih nalog za različne osnovne učne metode.

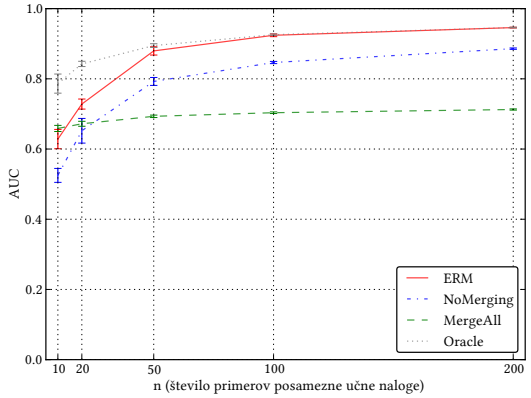
Za ta poskus smo uporabili enake parametre za generiranje Boolovih funkcij kot so podane v podrazdelku 3.1.2. Prav tako smo uporabili enak način izračuna povprečnih vrednosti AUC in njihovih 95% konfidenčnih intervalov. Rezultati so podani na sliki 3.6. S SVM je bil čas enega zagona metode *ERM* za $n = 10$ okoli 15 s, za $n = 200$ pa okoli 700 s. Z odločitvenim drevesom je bil čas enega zagona metode *ERM* za $n = 10$ okoli 20 s, za $n = 200$ pa okoli 80 s. S k-NN je bil čas enega zagona metode *ERM* za $n = 10$ okoli 10 s, za $n = 200$ pa okoli 50 s.

Tudi pri tem poskusu rezultati kažejo, da so trendi naraščanja krivulj AUC s povečevanjem števila primerov posameznih učnih nalog ne glede na izbrano osnovno učno metodo (kvalitativno gledano) zelo podobni.

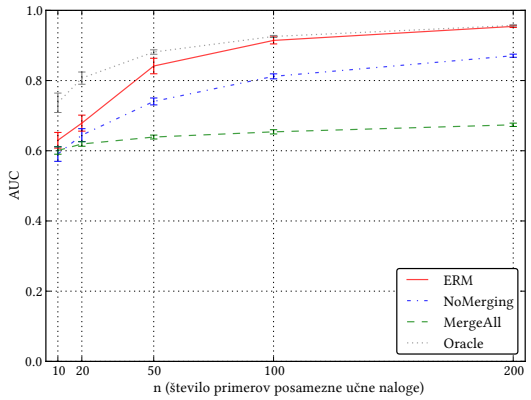
Krivulja AUC metode *MergeAll* ima pri vseh treh osnovnih učnih metodah enako obliko — ob povečevanju števila primerov posameznih učnih nalog zelo počasi narašča. Pri metodi *NoMerging* se krivulje AUC na začetku za različne osnovne učne metode nekoliko razlikujejo, s povečevanjem števila primerov posamezne učne naloge pa so si čedalje bolj podobne. Za osnovni učni metodi SVM in k-NN je AUC metode *NoMerging* pri 10 primerih na učno nalogo zelo nizek (~ 0.5), statistično značilno manjši od vseh ostalih MTL-metod, medtem ko je pri odločitvenem drevesu enakovreden metodama *ERM* in *MergeAll* (~ 0.6). To nakazuje, da je za SVM in k-NN z izbranimi parametri 10 primerov povsem premalo za izgradnjo modela, ki je boljši od naključnega napovedovanja.

Zanimivo je tudi obnašanje metode *ERM* v primerjavi z metodo *Oracle*. Za vse osnovne učne metode *ERM* na koncu (pri 200 primerih na posamezno učno nalogo) doseže metodo *Oracle*. Hitrost naraščanja njene krivulje AUC pa je za različne osnovne učne metode različna — pri odločitvenem drevesu je nekoliko počasnejša kot za ostali dve osnovni učni metodi, saj je njen AUC še pri 100 primerih na učno nalogo statistično značilno manjši od AUC metode *Oracle*. To nakazuje, da je združevanje učnih nalog v gruče z uporabo odločitvenih dreves nekoliko težje in manj učinkovito od uporabe drugih osnovnih učnih metod (npr. SVM ali k-NN).

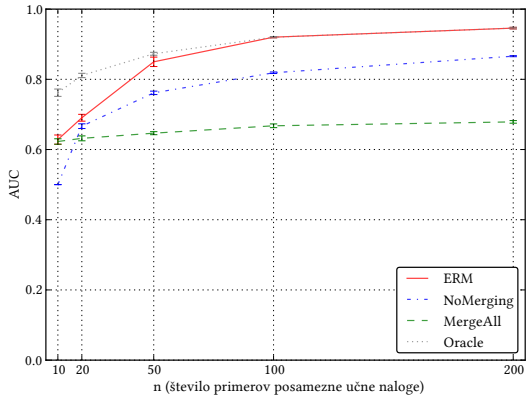
Iz rezultatov lahko zaključimo, da se metoda *ERM* s povečevanjem števila primerov posameznih učnih nalog zelo dobro obnese ne glede na uporabljeno osnovno učno metodo.



(a) SVM



(b) odločitveno drevo



(c) k-NN

Slika 3.6: Rezultati poskusov za različne osnovne učne metode na izbranem sintetičnem MTL-problemu učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila primerov posameznih učnih nalog (lastnost iz definicije 2.2). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

3.1.5.3 Spreminjanje števila gruč različnih tipov učnih nalog

Tretji poskus je zajemal opazovanje obnašanja MTL-metod ob spreminjanju števila gruč različnih tipov učnih nalog za različne osnovne učne metode.

Za ta poskus smo uporabili enake parametre za generiranje Boolovih funkcij, kot so podane v podrazdelku 3.1.3. Prav tako smo uporabili enak način izračuna povprečnih vrednosti AUC in njihovih 95% konfidenčnih intervalov. Rezultati so podani na sliki 3.7. S SVM je bil čas enega zagona metode *ERM* za $g = 1$ približno 1 s, za $g = 20$ pa okoli 1000 s. Z odločitvenim drevesom je bil čas enega zagona metode *ERM* za $g = 1$ približno 1 s, za $g = 20$ pa okoli 550 s. S k-NN je bil čas enega zagona metode *ERM* za $g = 1$ približno 1 s, za $g = 20$ pa okoli 250 s.

Ponovno so trendi naraščanja in padanja krivulj AUC vseh MTL-metod ne glede na uporabljeno osnovno učno metodo (kvalitativno gledano) zelo podobni.

Nihanje krivulj AUC za metode *Oracle*, *ERM* in *NoMerging* smo že razložili v podrazdelku 3.1.3 in je posledica različnih stopenj težavnosti naučljivosti Boolovih funkcij. Na sliki 3.7 lahko vidimo, da je bolj ali manj enako prisoten pri vseh osnovnih učnih metodah.

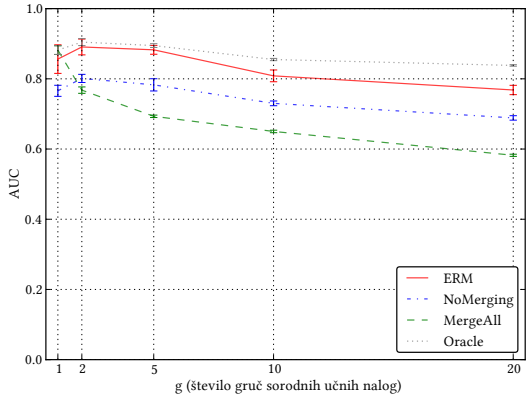
Opazimo lahko, da so krivulje AUC za metode *Oracle*, *NoMerging* in *MergeAll* ne glede na osnovno učno metodo zelo podobne, krivulje metode *ERM* pa se nekoliko razlikujejo. Pri SVM namreč njena krivulja AUC dalj časa ostaja bližje krivulji metode *Oracle* (za eno, dve oz. pet gruč sorodnih učnih nalog se njuni 95% konfidenčni intervali prekrivajo). Pri odločitvenem drevesu in k-NN pa se 95% konfidenčni intervali za metodi *ERM* in *Oracle* prekrivajo le za eno oz. dve gruči sorodnih učnih nalog, od tam naprej pa je AUC metode *ERM* statistično značilno manjši od metode *Oracle*. Pri odločitvenem drevesu in k-NN pri 20 gručah sorodnih učnih nalog njun AUC pade že na podobno vrednost, kot jo dosega metoda *NoMerging*, medtem ko je pri SVM še daleč nad njo (razlika v AUC je 0.08).

Ponovno lahko zaključimo, da metoda *ERM* ob povečevanju števila gruč sorodnih učnih nalog solidno deluje ne glede na uporabljeno osnovno učno metodo.

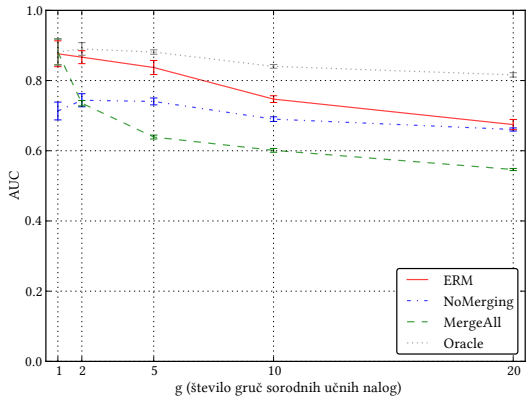
3.1.5.4 Spreminjanje stopnje šuma v podatkih

Četrti poskus je zajemal opazovanje obnašanja MTL-metod ob spreminjanju stopnje šuma v podatkih učnih nalog za različne osnovne učne metode.

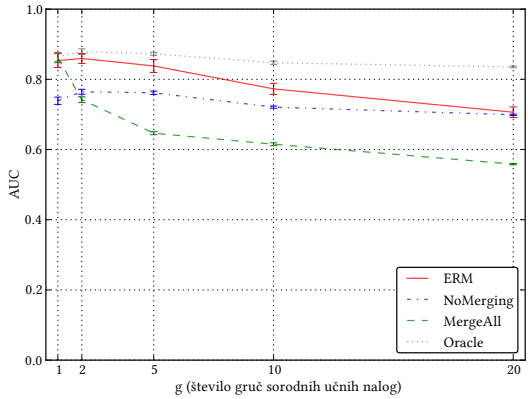
Za ta poskus smo uporabili enake parametre za generiranje Boolovih funkcij, kot so podane v podrazdelku 3.1.4. Prav tako smo uporabili enak način izračuna pov-



(a) SVM



(b) odločitveno drevo



(c) k-NN

Slika 3.7: Rezultati poskusov za različne osnovne učne metode na izbranem sintetičnem MTL-problemu učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju števila gruč sorodnih učnih nalog (lastnost iz definicije 2.3). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

prečnih vrednosti AUC in njihovih 95% konfidenčnih intervalov. Rezultati so podani na sliki 3.8. S SVM je bil čas enega zagona metode *ERM* za vse vrednosti parametra *noise* okoli 60 s. Z odločitvenim drevesom je bil čas enega zagona metode *ERM* za vse vrednosti parametra *noise* okoli 30 s. S k-NN je bil čas enega zagona metode *ERM* za vse vrednosti parametra *noise* okoli 20 s.

Pri tem poskusu se trendi padanja krivulj AUC za različne osnovne učne metode nekoliko razlikujejo. Krivulja metode *Oracle* pri vseh osnovnih učnih metodah pričakovano ostaja nad krivuljami vseh ostalih metod. Zanimiva razlika pa se zgodi pri krivuljah ostalih treh MTL-metod.

Pri SVM je krivulja AUC metode *ERM* vseskozi nad krivuljama ostalih dveh metod in le pri največji stopnji šuma (*noise* = 0.5) ne moremo več z gotovostjo trditi, da je statistično značilno boljša od metode *MergeAll*, saj se njuna 95% konfidenčna intervala prekrivata. Pri odločitvenem drevesu in k-NN pa pride do obrata med vrednostma parametra *noise* 0.2 in 0.3, ko AUC metode *NoMerging* postane večji od AUC metode *ERM*. Na koncu (*noise* = 0.5) je pri obeh omenjenih osnovnih učnih metodah AUC metode *NoMerging* statistično značilno večji od AUC metode *ERM*.

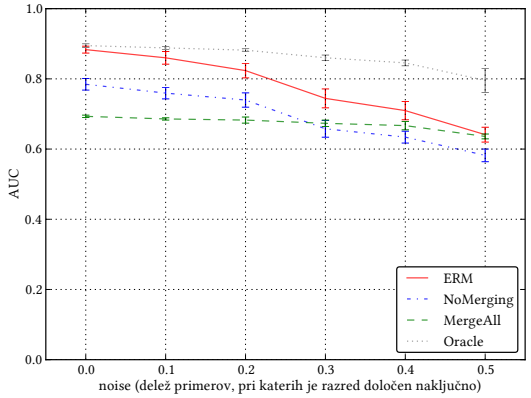
Prav tako se spremeni razmerje med krivuljama AUC metod *NoMerging* in *MergeAll*. Pri SVM krivulja metode *NoMerging* začne med vrednostma *noise* 0.2 in 0.3 strmo padati in je za vrednosti *noise* $\geq$ 0.3 nižja od krivulje metode *MergeAll*. Pri odločitvenem drevesu in metodi k-NN pa med krivuljama AUC metod *NoMerging* in *MergeAll* vseskozi ostaja velika razlika — AUC metode *NoMerging* je povsod statistično značilno večji od metode *MergeAll*.

3.2 Realni klasifikacijski MTL-problemi

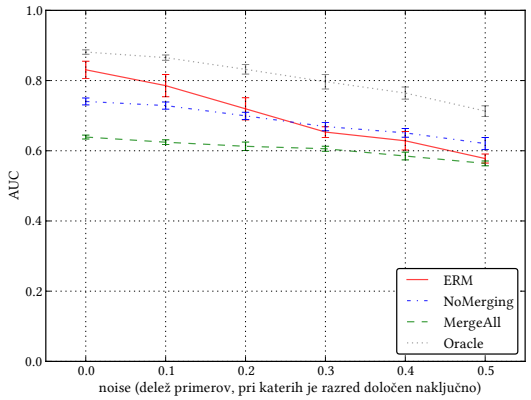
V okviru eksperimentalne študije obnašanja metode *ERM* smo izvedli poskuse na dveh realnih klasifikacijskih MTL-problemih: *USPS digits* in *MNIST digits*. Pri obeh MTL-problemih gre za nalogo prepoznavanja ročno napisanih števk (npr. števk poštnih številk na pismih in razglednicah). Izbrali smo ju, ker sta bila uporabljena v najnovejši literaturi s področja MTL, ki odpravi predpostavko, da so vse učne naloge sorodne, in samo selektivno združuje učne naloge v gruče [14, 16].

Vsak izmed obeh MTL-problemov je prvotno večrazredni klasifikacijski problem. V MTL-problem ga pretvorimo tako, da za vsako števko naredimo eno klasifikacijsko učno nalogo ločevanja med izbrano števko in preostalimi števkami.

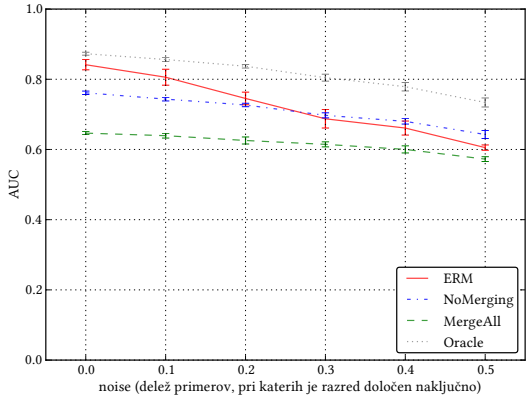
Pri vseh poskusih smo za osnovno učno metodo uporabili *logistično regresijo* s pri-



(a) SVM



(b) odločitveno drevo



(c) k-NN

Slika 3.8: Rezultati poskusov za različne osnovne učne metode na izbranem sintetičnem MTL-problemu učenja Boolovih funkcij, kjer smo primerjali MTL-metode ob spreminjanju stopnje šuma v podatkih učnih nalog (vrednost parametra *noise* označuje delež primerov, pri katerih je razred določen naključno). Prikazana so povprečja desetih ponovitev poskusa, kjer je bila za vsako ponovitev izračunana povprečna vrednost AUC čez vse učne naloge. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

vzetimi parametri, kot je implementirana v knjižnici za strojno učenje *scikit-learn* [47]. Zanjo smo se odločili, ker so jo za ta MTL-problema uporabljali tudi v [14, 16]. Vsak poskus je zajemal primerjavo treh MTL-metod: *ERM*, *NoMerging* in *MergeAll* (delovanje zadnjih dveh je razloženo v uvodnem delu 3. poglavja). Zgrajene modele smo ocenjevali z AUC, standardno mero za ocenjevanje kvalitete klasifikacijskih modelov. Uporabi CA, ki je prav tako standardna mera za ocenjevanje kvalitete klasifikacijskih modelov, smo se zaradi zelo neenakomerne porazdelitve vrednosti razreda ($\sim 1 : 9$) izognili.

3.2.1 Problem *USPS digits*

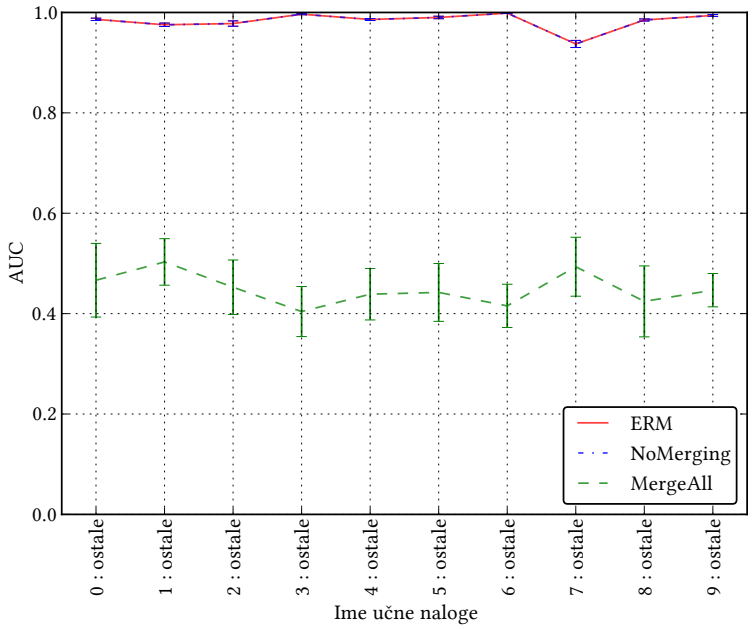
Prvotne podatke za problem *USPS digits* je zbral in opisal Hull leta 1994 v članku [49]. V naših eksperimentih smo uporabili enake podatke, kot so jih uporabili v [14, 16], in so javno dostopni na spletni strani [50] avtorjev prvega članka [14]. Tam so prvotne slike predobdelali z metodo *analize glavnih komponent* (angl. *principal component analysis*, *PCA*), s pomočjo katere so število atributov zmanjšali na 64, s čimer so še vedno ohranili $\sim$ 95% skupne variance. Nato so izmed vseh slik izbrali 2000 slik, na katerih so izvedli eksperimente.

Po pretvorbi prvotnega desetrazrednega klasifikacijskega problema v MTL-problem smo podatke vsake učne naloge razdelili na učno in testno množico, vsaka je imela polovico primerov. Tako smo dobili MTL-problem z desetimi učnimi nalogami, kjer ima vsaka naloga 1000 (učnih) primerov. Ker gre za realen MTL-problem, števila gruč različnih tipov učnih nalog (tretja lastnost MTL-problema, podana v definiciji 2.3) ne poznamo in je lahko katerokoli število med ena in deset.

Postopek delitve podatkov učnih nalog na učno in testno množico smo ponovili 10-krat ter izračunali povprečen AUC za vsako učno nalogo in njegov 95% konfidenčni interval. Rezultati so prikazani na sliki 3.9. Čas enega zagona metode *ERM* je bil približno 15 s.

Metodi *ERM* in *NoMerging* se odrežeta zelo dobro, medtem ko se metoda *MergeAll* izkaže za povsem neuporabno. Podroben pogled pokaže, da so rezultati za metodi *ERM* in *NoMerging* pravzaprav enaki. Do tega pride zato, ker metoda *ERM* pri tem eksperimentu nikoli ne združi nobenega para učnih nalog.

Sprva je bil ta rezultat nekoliko presenetljiv, vendar natančnejša analiza hitro razkrije, zakaj je temu tako. Metoda *ERM* združi par učnih nalog le takrat, ko je povprečna napovedna napaka modela, zgrajenega in testiranega na skupnih podatkih,



Slika 3.9: Rezultati poskusa za MTL-problem *USPS digits*, kjer so bili primeri za vsako učno nalogo razdeljeni na učno in testno množico v razmerju 50 : 50. Prikazana so povprečja desetih ponovitev poskusa, intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

manjša od uteženega povprečja povprečnih napovednih napak modelov, zgrajenih in testiranih na podatkih posamezne učne naloge (kriterij za združevanje (2)). Pri tem MTL-problemu smo podatke učnih nalog skonstruirali s pretvorbo večrazrednega klasifikacijskega problema na MTL-problem. To smo storili tako, da smo originalne podatke za vsako učno nalogo podvojili, kar pomeni, da imajo vse učne naloge enake primere, le razred je vsakič določen drugače: vrednost *true* dobijo primeri, ki predstavljajo trenutno številko, vrednost *false* pa vsi ostali. Če pogledamo primer združenih podatkov za učni nalogi t_i in t_j, $l_{t_i} + l_{t_j}$, imajo ti podatki 2-krat ponovljene enake primere. Primeri, ki imajo v učni nalogi t_i razred enak *true*, imajo v učni nalogi t_j razred enak *false*. In obratno, primeri, ki imajo v učni nalogi t_j razred enak *true*, imajo v učni nalogi t_i razred enak *false*. Ostali učni primeri imajo v obeh učnih nalogah enak razred. Zato združevanje podatkov dveh učnih nalog ni nikoli smiselno in koristno. V tem pogledu *ERM* deluje 100% uspešno, saj nikoli ne združi dveh učnih nalog, katerih model bi bil slabši od obeh modelov za posamezni učni nalogi.

Če primerjamo rezultate za posamezne učne naloge, vidimo, da večjih odstopanj ni. Najslabše se metodi *ERM* in *NoMerging* odrežeta pri ločevanju med številko 7 in preostalimi številkami, vendar je tudi v tem primeru AUC zelo visok (> 0.9).

3.2.1.1 Razbitje na podnaloge

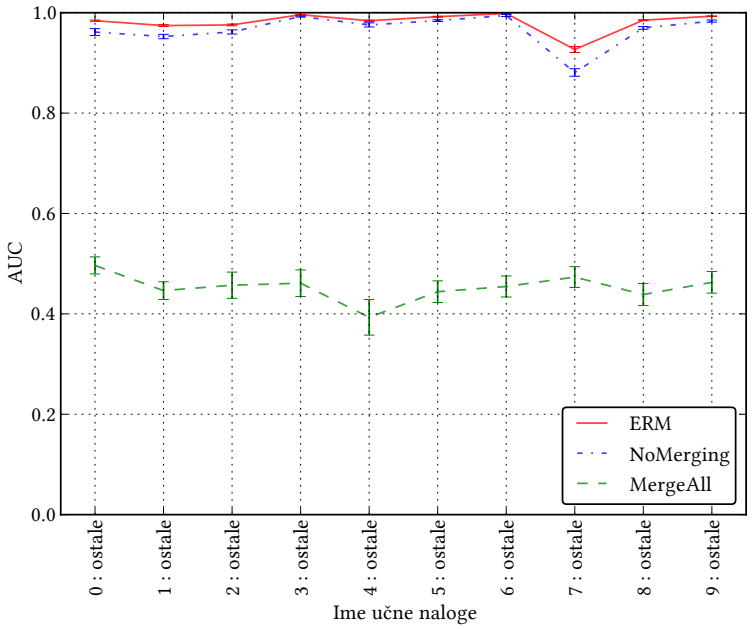
Prvotni MTL-problem, ki smo ga opisali v začetku razdelka, smo v nadaljevanju pretvorili v nov MTL-problem. Enako kot prej, smo začeli z desetimi učnimi nalogami ločevanja med izbrano števko in preostalimi števkami. Podatke vsake učne naloge smo ponovno razdelili na učno in testno množico v razmerju 50 : 50. Nato pa smo učne primere posamezne naloge razdelili v več podmnožic. Vsaka podmnožica učnih primerov je predstavljala novo učno nalogo (v tem smislu ji bomo rekli kar podnaloga prvotne učne naloge). Tako smo dobili nov MTL-problem s precej večjim število učnih nalog. Testiranje modelov, zgrajenih na podatkih podnalog, je potekalo na prvotnih (celotnih) testnih množicah.

Velja poudariti, da nobena MTL-metoda ni vedela, katere učne naloge sodijo skupaj, ker so bile pred razbitjem del iste prvotne učne naloge. Za MTL-metode so bile vse učne naloge enakovredne.

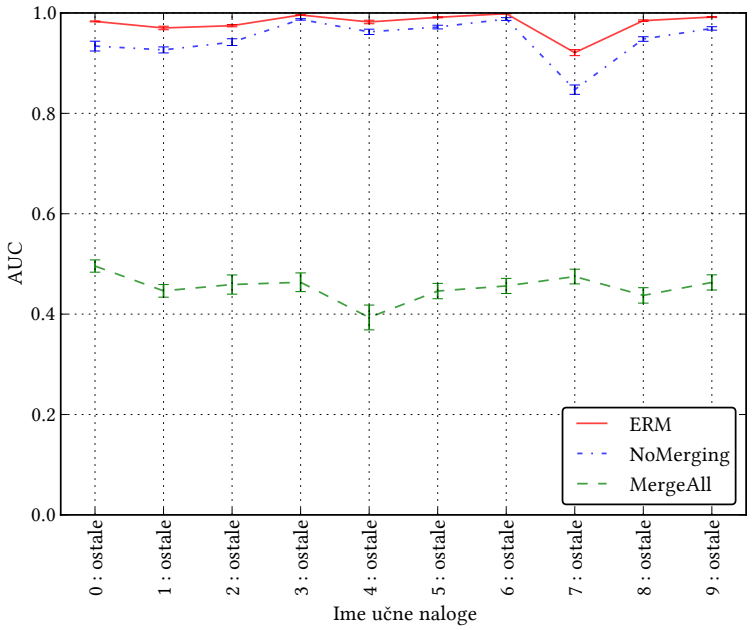
Pri prvem eksperimentu smo prvotne učne naloge razbili na 3–5 (število je bilo za vsako učno nalogo izbrano naključno) podnalog. Povprečni rezultati desetih ponovitev tega poskusa so podani na sliki 3.10(a). Pri drugem eksperimentu smo prvotne učne naloge razbili na 5–10 (število je bilo za vsako učno nalogo izbrano naključno) podnalog. Povprečni rezultati desetih ponovitev poskusa so prikazani na sliki 3.10(b). Čas enega zagona metode *ERM* je bil za prvi eksperiment okoli 200s, za drugi pa okoli 400s.

V obeh primerih se je metoda *ERM* izkazala na najboljšo. Blizu njej je bila še metoda *NoMerging*, medtem ko je metoda *MergeAll* na tem MTL-problemu povsem neuporabna. Večja razlika v vrednosti AUC za metodi *ERM* in *NoMerging* je prisotna pri učnih nalogah, kjer je AUC metode *NoMerging* nekoliko nižji. Ker metoda *NoMerging* za vsako učno nalogo zgradi model le na njenih podatkih, lahko vidimo, da združevanje z metodo *ERM* najbolj koristi takrat, ko imamo za posamezno učno nalogo na voljo premalo podatkov in bi z več podatki lahko zgradili boljši model. Ravno to naredimo s pomočjo metode *ERM*.

Če primerjamo rezultate obeh poskusov, razbitje na 3–5 podnalog in razbitje na 5–10 podnalog, lahko vidimo, da so trendi v obeh primerih enaki, le da so v drugem poskusu rezultati še bolj izrazito v korist metode *ERM*. Pri tem poskusu namreč učne podatke prvotnih nalog razbijemo na več delov, kar pomeni, da ima vsak del manj učnih primerov. Zato brez združevanja (metoda *NoMerging*) dobimo slabše modele. Kljub temu da smo učne naloge razbili na več delov, pa *ERM* še vedno uspešno zdru-



(a) Vsako učno nalogo smo razbili na 3–5 podnalog (število je bilo za vsako učno nalogo izbrano naključno).



(b) Vsako učno nalogo smo razbili na 5–10 podnalog (število je bilo za vsako učno nalogo izbrano naključno).

Slika 3.10: Rezultati poskusa za MTL-problem *USPS digits*, kjer smo prvotne učne naloge razbili na več podnalog. Prikazana so povprečja desetih ponovitev poskusa, intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

žuje učne naloge, katerih podatki sodijo skupaj.

Metoda *ERM* učne naloge združuje v gruče na aglomerativen način, podoben aglomerativnim metodam hierarhičnega gručenja. Zato lahko potek združevanja vizualiziramo z *dendrogramom.* Na sliki 3.11 je prikazan primer dendrograma za 2. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 3–5 podnalog.

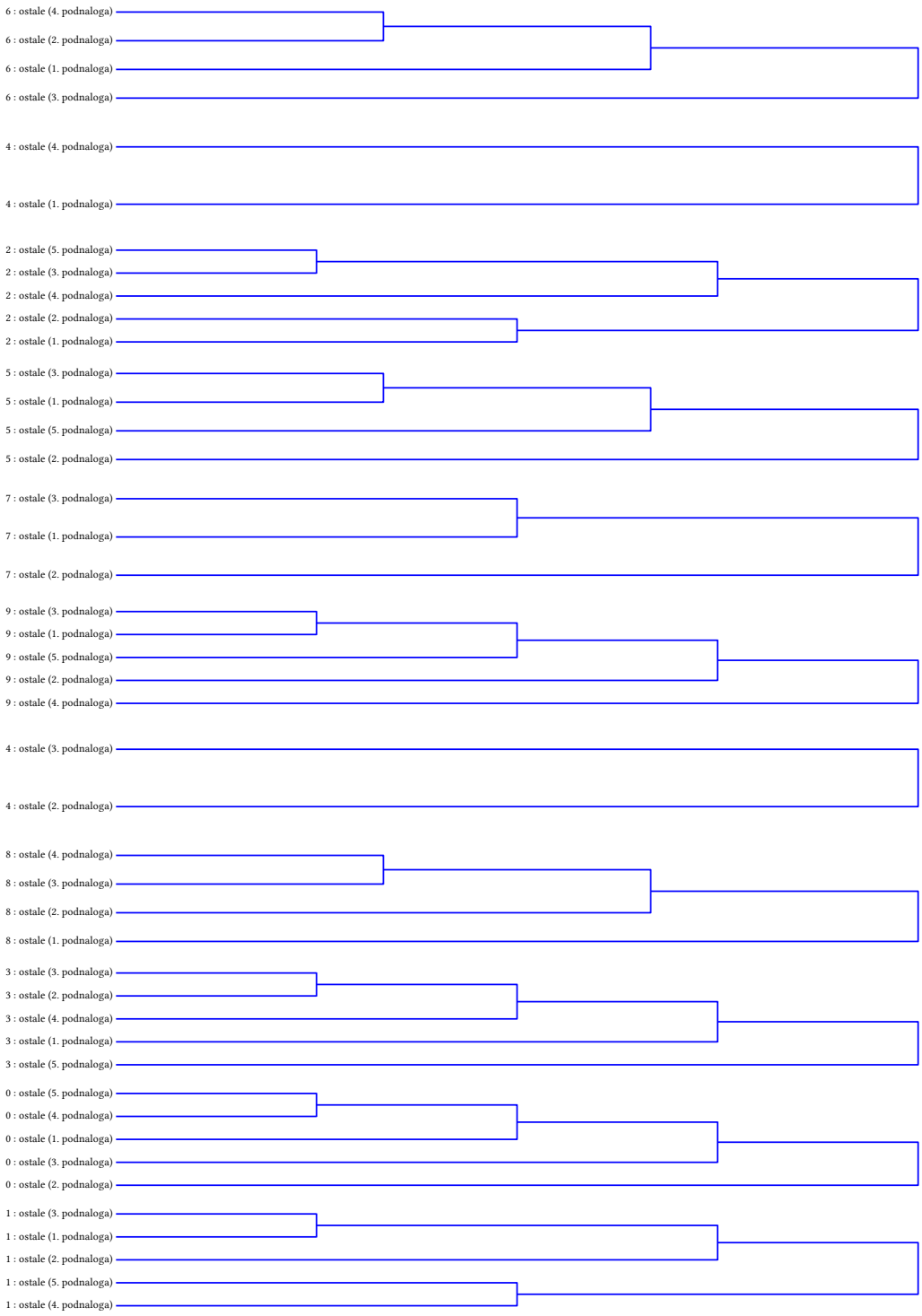
Pri pregledu vseh dendrogramov za eksperiment, kjer smo vsako učno nalogo razbili na 3–5 podnalog, smo opazili, da se na koncu oblikuje od 10 do 11 gruč, ki ustrezajo prvotnim desetim učnim nalogam ločevanja med izbrano števko in preostalimi števkami. Še pomembnejši rezultat, ki smo ga razbrali iz dendrogramov, je, da metoda *ERM* nikoli ne združi dveh učnih nalog, ki pripadata različnim prvotnim učnim nalogam. Z drugimi besedami, *ERM* se nikoli ne zmoti in napačno združi dveh učnih nalog. Zgodi se kvečjemu to, da kdaj nekaterih učnih nalog ne združi, čeprav pripadajo isti prvotni učni nalogi. Tako na primer v 2. ponovitvi eksperimenta *ERM* ni združila vseh podnalog prvotne učne naloge "4 : ostale" (glej sliko 3.11).

Na sliki 3.12 je prikazan še primer dendrograma 10. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 5–10 podnalog.

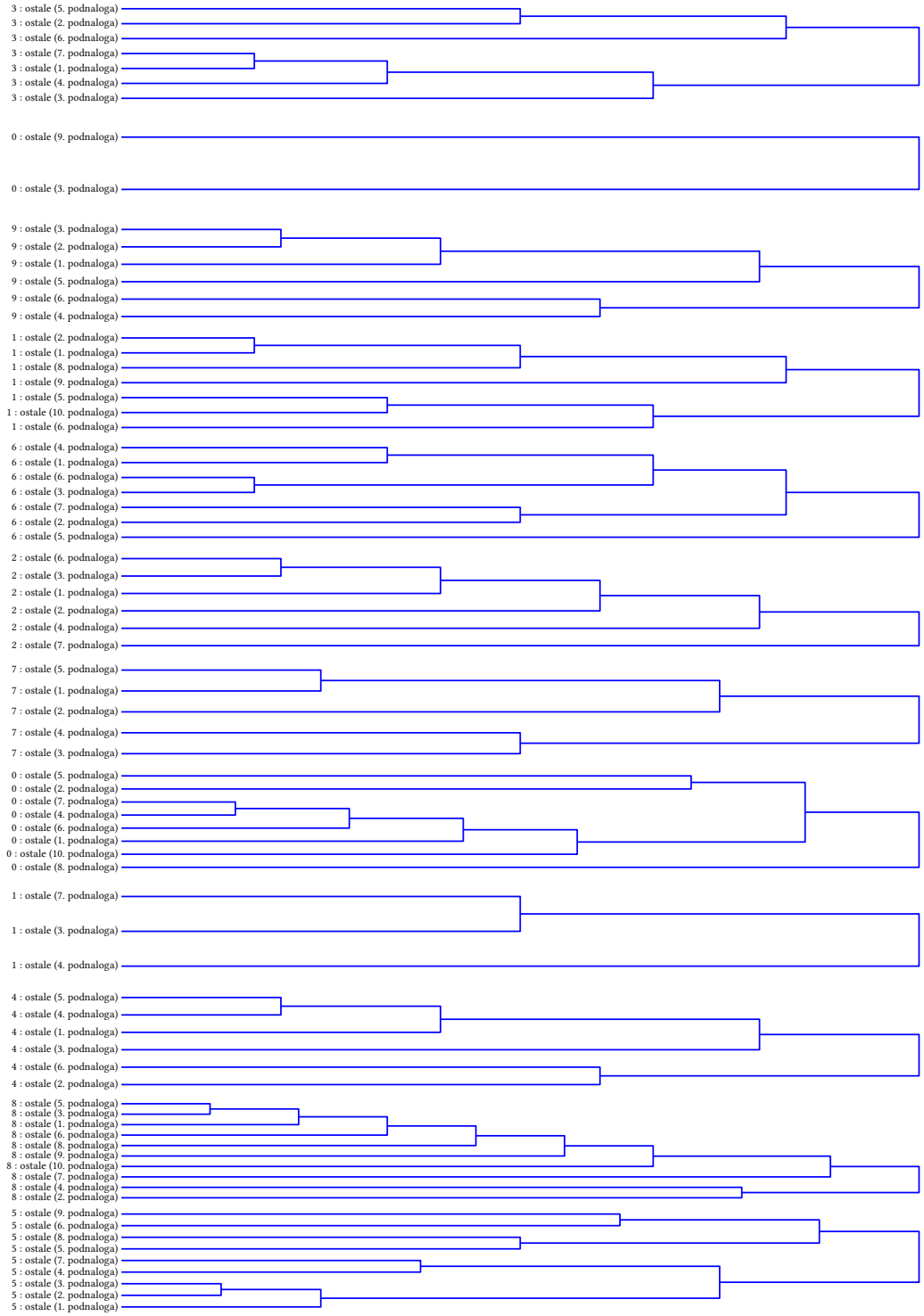
Pri eksperimentu, kjer smo prvotne učne naloge razbili na 5–10 podnalog, smo dobili nekoliko drugačne rezultate kot pri prvem eksperimentu. Pri pregledu vseh dendrogramov smo opazili, da se na koncu oblikuje med 11 in 14 gruč, kar je nekoliko več od prvotnih desetih učnih nalog ločevanja med izbrano števko in preostalimi števkami. Seveda je bila tu naloga metode *ERM* precej težja, saj so bile prvotne učne naloge razbite na veliko več (med 5 in 10) podnalog. Vendar smo tudi pri tem eksperimentu opazili zelo pomemben rezultat: metoda *ERM* (ponovno) nikoli ne združi dveh učnih nalog, ki pripadata različnim prvotnim učnim nalogam. Kvečjemu je *ERM* bolj konzervativna in nekaterih učnih nalog ne združi, čeprav pripadajo isti prvotni učni nalogi. Tak primer se je zgodil v 10. ponovitvi poskusa, ko *ERM* ni združila vseh podnalog prvotnih učnih nalog "0 : ostale" ter "1 : ostale"(glej sliko 3.12).

3.2.1.2 Primerjava s sorodnimi metodami

Rezultate za prvotni MTL-problem (tj. brez razbitja na podnaloge) smo primerjali tudi z rezultati sorodnih metod. Zaradi tehnične zahtevnosti implementacije direktne primerjave s sorodnimi metodami, smo se omejili le na primerjavo z rezultati, podanimi v člankih. Eksperimente z MTL-metodami *ERM*, *NoMerging* in *MergeAll* smo izvedli tako, da smo v največji možni meri sledili opisom iz člankov.



Slika 3.11: Primer dendrograma, ki prikazuje potek združevanja učnih nalog za 2. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 3–5 podnalog. Na y-osi so prikazana imena učnih nalog. Vidimo lahko, da v tej ponovitvi eksperimenta *ERM* ni združila vseh podnalog prvotne učne naloge "4 : ostale".



Slika 3.12: Primer dendrograma, ki prikazuje potek združevanja učnih nalog za 10. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 5–10 podnalog. Na y-osi so prikazana imena učnih nalog. Vidimo lahko, da v tej ponovitvi eksperimenta *ERM* ni združila vseh podnalog prvotnih učnih nalog "0 : ostale" ter "1 : ostale".

Vir rezultatov	Ime metode	Klasifikacijska napaka
pričujoča disertacija	*NoMerging*	0.02 ± 0.00
	MergeAll	0.10 ± 0.00
	ERM	0.02 ± 0.00
Kumar in Daumé III [16]	*NoMerging* (STL)	0.09 ± 0.00
	No-group MTL [11]	0.08 ± 0.00
	Disjoint-group MTL [14]	0.08 ± 0.00
	Group-overlap MTL [16]	0.07 ± 0.00
Kang et al. [14]	*NoMerging* (Single task)	0.10 ± 0.00
	No-group MTL [14]	0.09 ± 0.00
	Disjoint-group MTL [14]	0.08 ± 0.00

Tabela 3.1: Povprečne vrednosti klasifikacijske napake različnih MTL-metod za problem *USPS digits*. Povprečja so izračunana čez vse učne naloge (števke). Intervali napake podajajo standardne odklone povprečij.

Naredili smo primerjavo z rezultati iz člankov [16] in [14]. V obeh člankih so pri testiranju različnih MTL-metod podatke za vsako učno nalogo razdelili tako, da so 1000 primerov uporabili za učenje, 500 primerov za preverjanje izbire parametra G (število gruč različnih tipov učnih nalog) in 500 primerov za testiranje. Ker metoda *ERM* ne potrebuje posebne množice za preverjanje izbire parametra G, smo v naših eksperimentih te primere uporabili za testiranje in je imela testna množica 1000 primerov. Kot v [16] in [14] smo za mero uporabili *klasifikacijsko napako* (angl. *classification error*). Rezultati so podani v tabeli 3.1.

Najbolje sta se odrezali metodi *ERM* in *NoMerging* (kot smo jo implementirali v disertaciji), vse ostale metode so imele bistveno večjo klasifikacijsko napako. Dosežena rezultata sta tudi statistično značilna. Med preostalimi metodami se je nekoliko bolje odrezala metoda *Group-overlap MTL*, nekoliko slabše od vseh ostalih metod pa metodi *MergeAll* in *NoMerging* (kot je bila implementirana v [14]).

Opazimo lahko, da se rezultat za metodo *NoMerging*, ki smo ga dobili v disertaciji, zelo razlikuje od rezultatov iz člankov [16] in [14]. Prvi je med vsemi primerjanimi MTL-metodami med najboljšimi, druga dva pa sta med najslabšimi. Domnevamo, da je do tega prišlo zaradi uporabe različnih implementacij logistične regresije ter drugačnega deljenja primerov na učno in testno množico. V [16] in [14] namreč potrebujejo še posebno množico primerov za preverjanje izbire parametra G, medtem

ko smo v naših eksperimentih te primere uporabili za testiranje.

Primerjava je pokazala, da so rezultati metode *ERM* statistično značilno boljši od sorodnih MTL-metod iz člankov [11, 14, 16]. Pri tem velja omeniti, da so rezultati metode *ERM* enaki metodi *NoMerging*, kot smo jo implementirali v disertaciji, kar pomeni, da k uspehu *ERM* ne pripomore boljše gručenje učnih nalog. Do neke mere lahko to nekonsistentnost pojasnimo s tem, da smo v disertaciji uporabljali boljšo implementacijo logistične regresije ter nekoliko drugačno množico testnih primerov.

3.2.2 Problem *MNIST digits*

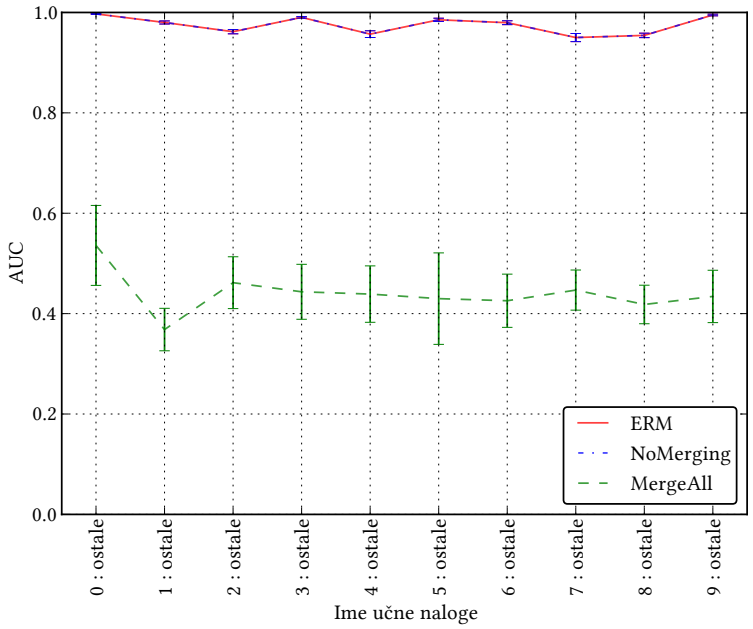
Prvotne podatke za ta problem so leta 1998 pripravili avtorji članka [51]. V disertaciji smo uporabili enake podatke, kot so jih uporabili v [14, 16], in so javno dostopni na spletni strani [50] avtorjev prvega članka [14]. Tam so prvotne slike predobdelali z metodo PCA, s pomočjo katere so število atributov zmanjšali na 87, s čimer so še vedno ohranili $\sim$ 95% skupne variance. Nato so izmed vseh slik izbrali 2000 slik, na katerih so izvedli eksperimente.

Po pretvorbi prvotnega desetrazrednega klasifikacijskega problema v MTL-problem smo podatke vsake učne naloge razdelili na učno in testno množico, vsaka je imela polovico primerov. Tako smo dobili MTL-problem z desetimi učnimi nalogami, kjer ima vsaka naloga 1000 (učnih) primerov. Ker gre za realen MTL-problem, števila gruč različnih tipov učnih nalog (tretja lastnost MTL-problema podana v definiciji 2.3) ne poznamo in je lahko katerokoli število med ena in deset.

Postopek delitve podatkov učnih nalog na učno in testno množico smo ponovili 10-krat ter izračunali povprečen AUC za vsako učno nalogo in njegov 95% konfidenčni interval. Rezultati so prikazani na sliki 3.13. Čas enega zagona metode *ERM* je bil približno 20 s.

Metodi *ERM* in *NoMerging* sta se ponovno, tako kot pri problemu *USPS digits*, odrezali zelo dobro, medtem ko se je metoda *MergeAll* izkazala za povsem neuporabno. Podroben pregled pokaže, da so rezultati za metodi *ERM* in *NoMerging* pravzaprav enaki. Do tega pride zato, ker metoda *ERM* pri tem eksperimentu nikoli ne združi nobenega para učnih nalog. Razlaga, zakaj pride do tega, je enaka kot pri prejšnjem MTL-problemu, *USPS digits*.

Če primerjamo rezultate za posamezne učne naloge, vidimo da večjih odstopanj ni. Nekoliko slabše se metodi *ERM* in *NoMerging* odrežeta pri ločevanju med števko 2 in preostalimi, števko 4 in preostalimi, števko 7 in preostalimi ter števko 8 in preostalimi



Slika 3.13: Rezultati poskusa za MTL-problem *MNIST digits*, kjer so bili primeri za vsako učno nalogo razdeljeni na učno in testno množico v razmerju 50 : 50. Prikazana so povprečja desetih ponovitev poskusa, intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

številkami. Vendar je tudi v teh primerih AUC zelo visok (> 0.9).

3.2.2.1 Razbitje na podnaloge

Enako kot pri MTL-problemu *USPS digits*, smo tudi MTL-problem *MNIST digits* v nadaljevanju pretvorili v nov MTL-problem z razbitjem prvotnih učnih nalog na podnaloge. Začeli smo z desetimi učnimi nalogami ločevanja med izbrano številko in preostalimi številkami. Podatke vsake učne naloge smo razdelili na učno in testno množico v razmerju 50 : 50. Nato smo učne primere posamezne naloge razdelili v več podmnožic. Vsaka podmnožica učnih primerov je predstavljala novo učno nalogo, t. i. podnalogo prvotne učne naloge. Tako smo dobili nov MTL-problem s precej večjim število učnih nalog. Testiranje modelov, zgrajenih na podatkih podnalog, je potekalo na prvotnih (celotnih) testnih množicah.

Velja poudariti, da nobena MTL-metoda ni vedela, katere učne naloge sodijo skupaj, ker so bile pred razbitjem del iste prvotne učne naloge. Za MTL-metode so bile vse učne naloge enakovredne.

Enako kot pri MTL-problemu *USPS digits* smo pri prvem eksperimentu prvotne učne naloge razbili na 3–5 (število je bilo za vsako učno nalogo izbrano naključno)

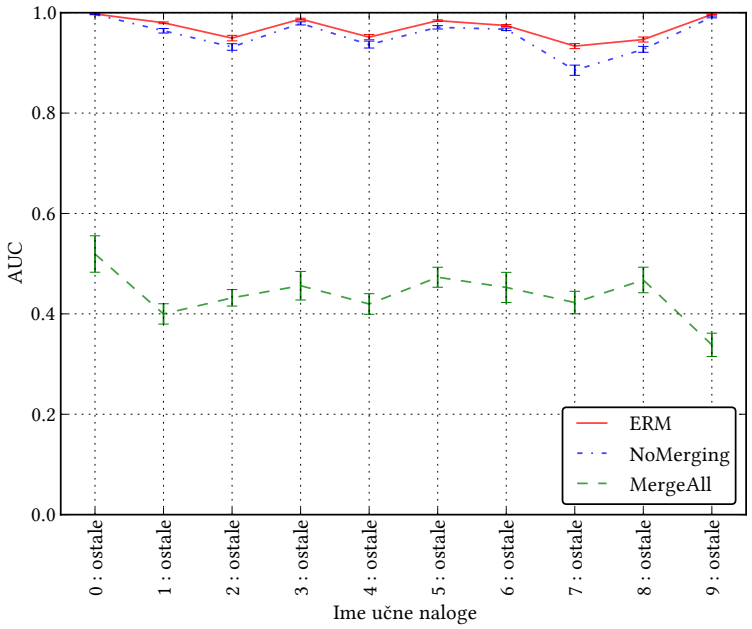
podnalog. Povprečni rezultati desetih ponovitev poskusa so podani na sliki 3.14(a). Pri drugem eksperimentu smo prvotne učne naloge razbili na 5–10 (število je bilo za vsako učno nalogo izbrano naključno) podnalog. Povprečni rezultati desetih ponovitev poskusa so prikazani na sliki 3.14(b). Čas enega zagona metode *ERM* je bil za prvi eksperiment okoli 400 s, za drugi pa okoli 900 s.

Ponovno se je v obeh primerih metoda *ERM* izkazala za najboljšo. Blizu njej je bila še metoda *NoMerging*, medtem ko je metoda *MergeAll* tudi na tem MTL-problemu povsem neuporabna. Večja razlika v vrednosti AUC za metodi *ERM* in *NoMerging* je prisotna pri učnih nalogah, kjer je AUC metode *NoMerging* nekoliko nižji. Ker metoda *NoMerging* za vsako učno nalogo zgradi model le na njenih podatkih, lahko vidimo, da združevanje z metodo *ERM* najbolj koristi takrat, ko imamo za posamezno učno nalogo na voljo premalo podatkov in bi z več podatki lahko zgradili boljši model. Ravno to naredimo s pomočjo metode *ERM*.

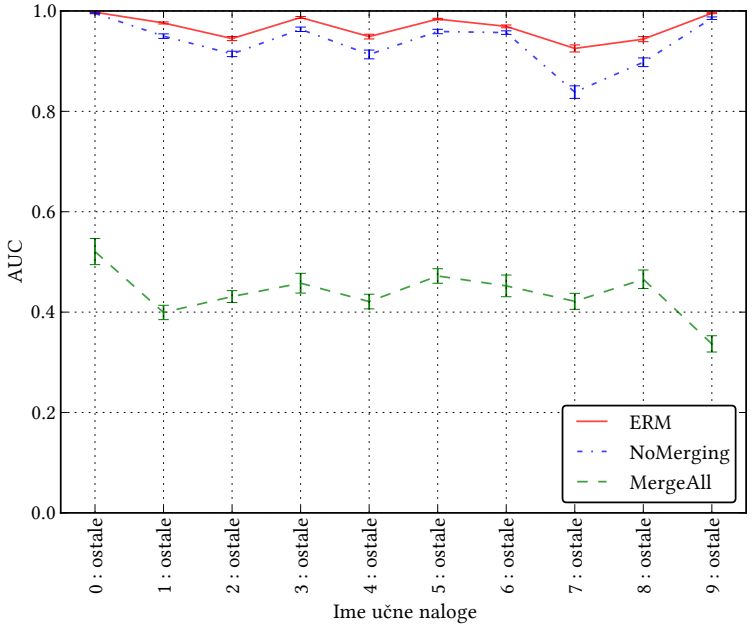
Če primerjamo rezultate obeh poskusov, razbitje na 3–5 podnalog in razbitje na 5–10 podnalog, lahko vidimo, da so trendi v obeh primerih enaki, le da so v drugem poskusu rezultati še bolj izrazito v korist metode *ERM*. Pri tem poskusu namreč učne podatke prvotnih nalog razbijemo na več delov, kar pomeni, da ima vsak del manj učnih primerov. Zato brez združevanja (metoda *NoMerging*) dobimo slabše modele. Kljub temu da smo učne naloge razbili na več delov, pa *ERM* še vedno uspešno združuje učne naloge, katerih podatki sodijo skupaj.

Metoda *ERM* učne naloge združuje v gruče na aglomerativen način, podoben aglomerativnim metodam hierarhičnega gručenja. Zato lahko potek združevanja vizualiziramo z dendrogramom. Na sliki 3.15 je prikazan primer dendrograma za 5. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 3–5 podnalog.

Pri pregledu vseh dendrogramov za eksperiment, kjer smo vsako učno nalogo razbili na 3–5 podnalog, smo opazili, da se na koncu oblikuje od 9 do 11 gruč, ki ustrezajo prvotnim desetim učnim nalogam ločevanja med izbrano števko in preostalimi števkami. Le pri 1. ponovitvi poskusa se je na koncu oblikovalo manj kot deset gruč, kar je posledica tega, da metoda *ERM* nobenega para podnalog učne naloge "6 : ostale" ni združila. Najpomembnejši rezultat, ki smo ga razbrali iz dendrogramov, je, da metoda *ERM* nikoli ne združi dveh učnih nalog, ki pripadata različnim prvotnim učnim nalogam. Z drugimi besedami, *ERM* se nikoli ne zmoti in napačno združi dveh učnih nalog. Pride kvečjemu do tega, da kdaj nekaterih učnih nalog ne združi, čeprav pripadajo isti prvotni učni nalogi. Tako na primer v 5. ponovitvi eksperimenta *ERM* ni združila vseh podnalog prvotnih učnih nalog "3 : ostale" ter "6 : ostale" (glej sliko 3.15).

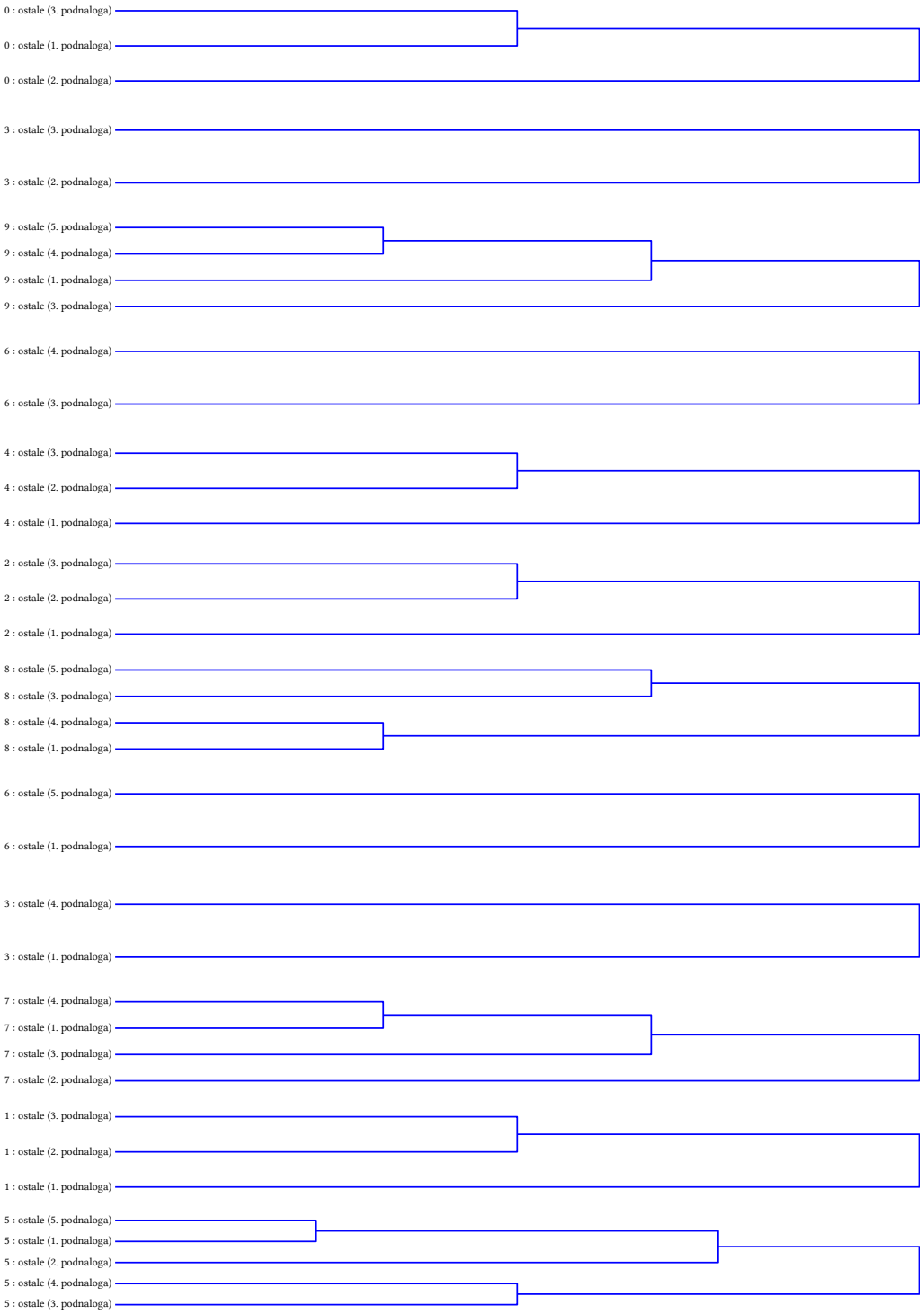


(a) Vsako učno nalogo smo razbili na 3–5 podnalog (število je bilo za vsako učno nalogo izbrano naključno).



(b) Vsako učno nalogo smo razbili na 5–10 podnalog (število je bilo za vsako učno nalogo izbrano naključno).

Slika 3.14: Rezultati poskusa za MTL-problem *MNIST digits*, kjer smo prvotne učne naloge razbili na več podnalog. Prikazana so povprečja desetih ponovitev poskusa, intervali napake prikazujejo 95% konfidenčne intervale za povprečja.



Slika 3.15: Primer dendrograma, ki prikazuje potek združevanja učnih nalog za 5. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 3–5 podnalog. Na y-osi so prikazana imena učnih nalog. Vidimo lahko, da v tej ponovitvi eksperimenta *ERM* ni združila vseh podnalog prvotnih učnih nalog "3 : ostale" in "6 : ostale".

Na sliki 3.16 je prikazan še primer dendrograma za 4. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 5–10 podnalog.

Tudi tukaj, podobno kot pri MTL-problemu *USPS digits*, smo dobili nekoliko drugačne rezultate kot pri prvem eksperimentu, kjer smo učne naloge razbili na manjše število podnalog. V tem eksperimentu se je na koncu oblikovalo med 13 in 15 gruč, kar je nekoliko več od prvotnih desetih učnih nalog. Seveda je bila tu naloga metode *ERM* dosti težja, saj so bile prvotne učne naloge razbite na veliko več (med 5 in 10) podnalog. Vendar lahko tudi pri tem eksperimentu opazimo zelo pomemben rezultat: metoda *ERM* (ponovno) nikoli ne združi dveh učnih nalog, ki pripadata različnim prvotnim učnim nalogam. Kvečjemu je *ERM* bolj konzervativna in nekaterih učnih nalog ne združi, čeprav pripadajo isti prvotni učni nalogi. To se je na primer zgodilo v 4. ponovitvi poskusa, ko *ERM* ni združila vseh podnalog prvotnih učnih nalog "9 : ostale", "1 : ostale", "7 : ostale" ter "2 : ostale" (glej sliko 3.16).

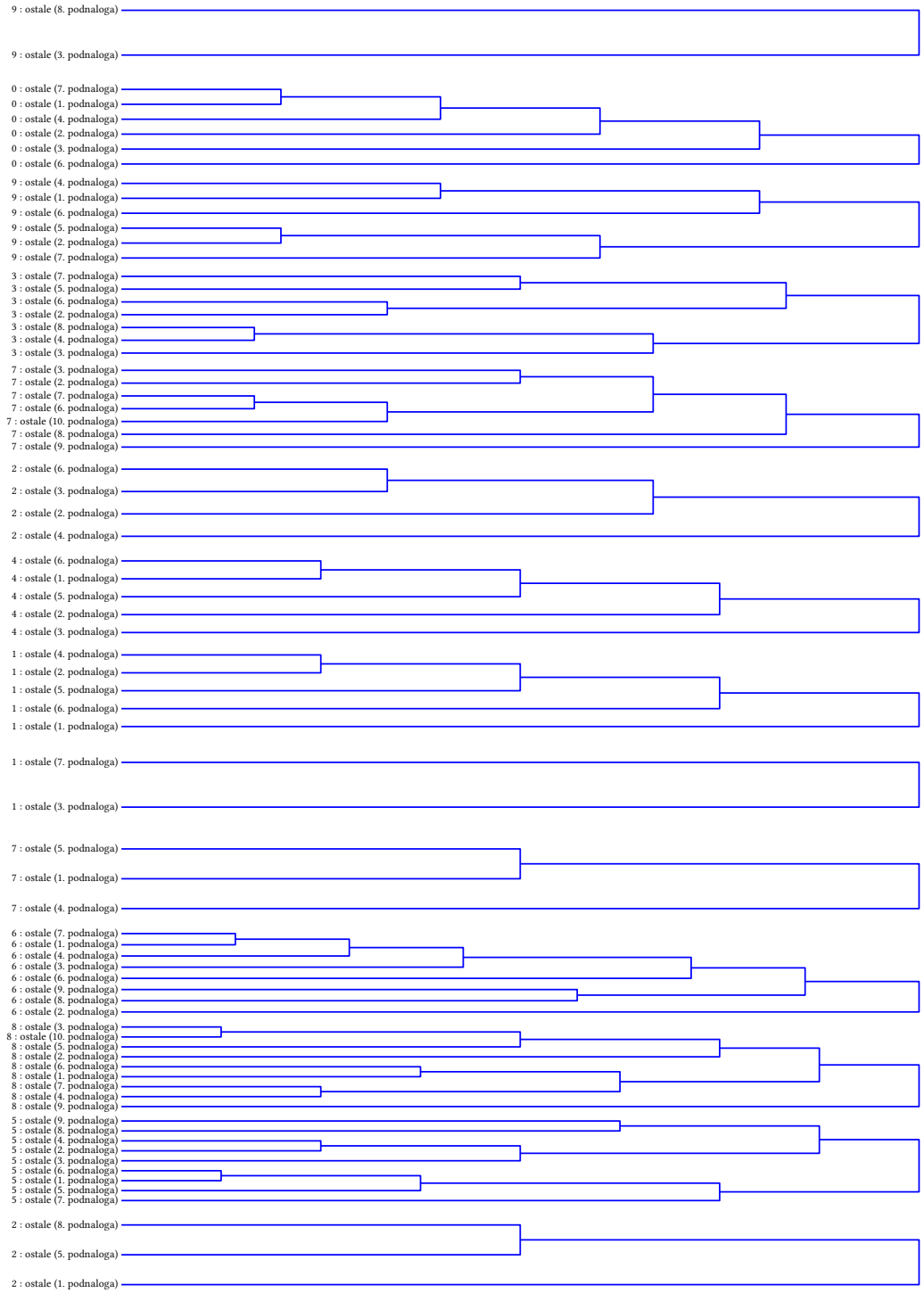
3.2.2.2 Primerjava s sorodnimi metodami

Rezultate za prvotni MTL-problem (tj. brez razbitja na podnaloge) smo primerjali tudi z rezultati sorodnih metod. Zaradi tehnične zahtevnosti implementacije direktne primerjave s sorodnimi metodami smo se omejili le na primerjavo z rezultati, podanimi v člankih. Eksperimente z MTL-metodami *ERM*, *NoMerging* in *MergeAll* smo izvedli tako, da smo v največji možni meri sledili opisom iz člankov.

Naredili smo primerjavo z rezultati iz člankov [16] in [14]. V obeh člankih so pri testiranju različnih MTL-metod podatke za vsako učno nalogo razdelili tako, da so 1000 primerov uporabili za učenje, 500 primerov za preverjanje izbire parametra G (število gruč različnih tipov učnih nalog) in 500 primerov za testiranje. Ker metoda *ERM* ne potrebuje posebne množice za preverjanje izbire parametra G, smo v naših eksperimentih te primere uporabili za testiranje in je imela testna množica 1000 primerov. Kot v [16] in [14] smo za mero uporabili klasifikacijsko napako. Rezultati so podani v tabeli 3.2.

Ponovno sta se najbolje odrezali metodi *ERM* in *NoMerging* (kot smo jo implementirali v disertaciji), vse ostale metode so imele bistveno večjo klasifikacijsko napako. Dosežena rezultata sta tudi statistično značilna. Med preostalimi metodami se je najbolje odrezala metoda *MergeAll*. Njen rezultat je bil prav tako statistično značilno boljši od preostalih metod.

Opazimo lahko, da se rezultat za metodo *NoMerging*, ki smo ga dobili v disertaciji,



Slika 3.16: Primer dendrograma, ki prikazuje potek združevanja učnih nalog za 4. ponovitev eksperimenta, kjer smo vsako učno nalogo razbili na 5–10 podnalog. Na y-osi so prikazana imena učnih nalog. Vidimo lahko, da v tej ponovitvi eksperimenta *ERM* ni združila vseh podnalog prvotnih učnih nalog "9 : ostale", "1 : ostale", "7 : ostale" ter "2 : ostale".

Vir rezultatov	Ime metode	Klasifikacijska napaka
pričujoča disertacija	*NoMerging*	0.03 ± 0.00
	MergeAll	0.10 ± 0.00
	ERM	0.03 ± 0.00
Kumar in Daumé III [16]	*NoMerging* (STL)	0.15 ± 0.00
	No-group MTL [11]	0.14 ± 0.00
	Disjoint-group MTL [14]	0.14 ± 0.00
	Group-overlap MTL [16]	0.13 ± 0.00
Kang et al. [14]	*NoMerging* (Single task)	0.16 ± 0.00
	No-group MTL [14]	0.16 ± 0.00
	Disjoint-group MTL [14]	0.15 ± 0.00

Tabela 3.2: Povprečne vrednosti klasifikacijske napake različnih MTL-metod za problem *MNIST digits*. Povprečja so izračunana čez vse učne naloge (števke). Intervali napake podajajo standardne odklone povprečij.

zelo razlikuje od rezultatov iz člankov [16] in [14]. Prvi je med najboljšimi izmed vseh primerjanih MTL-metod, druga dva pa sta med najslabšimi. Prav tako lahko opazimo, da so vse metode, ki smo jih testirali v disertaciji, dosegle statistično značilno boljše rezultate od metod iz člankov [16] in [14]. Domnevamo, da je do tega prišlo zaradi uporabe različnih implementacij logistične regresije ter drugačnega deljenja primerov na učno in testno množico. V [16] in [14] namreč potrebujejo še posebno množico primerov za preverjanje izbire parametra G, medtem ko smo v naših eksperimentih te primere uporabili za testiranje.

Primerjava je pokazala, da so rezultati metode *ERM* statistično značilno boljši od sorodnih MTL-metod iz člankov [11, 14, 16]. Pri tem velja omeniti, da so rezultati metode *ERM* enaki metodi *NoMerging*, kot smo jo implementirali v disertaciji, kar pomeni, da k uspehu *ERM* ne pripomore boljše gručenje učnih nalog. Nekoliko presenetljivo je, da so bili tudi rezultati metode *MergeAll* statistično značilno boljši od sorodnih MTL-metod. Do neke mere lahko to nekonsistentnost pojasnimo s tem, da smo v disertaciji uporabljali boljšo implementacijo logistične regresije ter nekoliko drugačno množico testnih primerov.

3.3 Realni regresijski MTL-problemi

V okviru eksperimentalne študije obnašanja metode *ERM* smo izvedli tudi poskuse na dveh realnih regresijskih MTL-problemih: *School* in *Computer Survey*. Prvi MTL-problem je eden najbolj znanih in največkrat uporabljenih problemov za testiranje MTL-metod. Med drugim je bil uporabljen v člankih [11, 13–16, 19, 30, 32, 33, 52]. Gre za podatke o učencih iz različnih šol v Londonu. Naloga je napovedovanje števila točk na zaključnem preizkusu znanja. Pri drugem MTL-problemu gre za podatke iz ankete, kjer so osebe podajale svojo naklonjenost nakupu različnim osebnih računalnikov na podlagi njihovih notranjih in zunanjih karakteristik. Naloga je napovedovanje naklonjenosti nakupu računalnika. Tudi ta problem je bil v najnovejši literaturi s področja MTL pogosto uporabljen za primerjavo in testiranje metod, na primer v [11, 16, 52].

Pri vseh poskusih smo za osnovno učno metodo uporabili *grebensko regresijo* (angl. *ridge regression*) z dodanim internim prečnim preverjanjem, ki izmed petih možnih vrednosti (0.1, 0.316, 1, 3.16, 10) regularizacijskega parametra *alpha* izbere najboljšo. Za izbiro parametra *alpha* z internim prečnim preverjanjem smo se odločili zato, ker je število primerov učnih nalog že prvotno zelo različno in se z združevanjem še povečuje, zaradi česar bi uporaba iste vrednosti *alpha* za vse učne naloge vodila v izgradnjo slabših modelov. S tem se časovna zahtevnost učenja z grebensko regresijo poveča za 5-krat, kar posledično upočasni tudi delovanje metode *ERM*. Vendar se je v naših poskusih izkazalo, da je trajanje tovrstnega učenja sprejemljivo. Parameter *normalize* (normalizacija vrednosti atributov) je bil nastavljen na *true*, vsi ostali parametri pa so imeli privzete vrednosti (kot pri implementaciji grebenske regresije v knjižnici za strojno učenje *scikit-learn* [47]). Vsak poskus je zajemal primerjavo treh MTL-metod: *ERM*, *NoMerging* in *MergeAll* (delovanje zadnjih dveh je razloženo v uvodnem delu 3. poglavja). Zgrajene modele smo ocenjevali s standardnima merama za ocenjevanje kvalitete regresijskih modelov: *korenom srednje kvadratne napake* (angl. *root mean square error*, *RMSE*) in *srednjo absolutno napako* (angl. *mean absolute error*, *MAE*).

3.3.1 Problem *School*

Prvotne podatke za ta MTL-problem je zbrala Inner London Education Authority v letih 1985, 1986 in 1987 v okviru obsežne študije učinkovitosti srednjih šol v notranjem Londonu [53, 54]. Podatki vsebujejo rezultate zaključnega preverjanja znanja za 15362

učencev iz 139 srednjih šol v Londonu. Šole so imele različno število učencev, od 22 do 251. V kontekstu MTL vsaka srednja šola predstavlja eno učno nalogo. Učenci predstavljajo primere, ki so opisani z naslednjimi atributi:

- leto preverjanja,
- atributi, vezani na srednjo šolo:
  - delež učencev, ki so upravičeni do brezplačnih šolskih obrokov,
  - delež učencev v najboljši skupini (VR Band 1) pri testu govorno-razumskih sposobnosti (angl. verbal reasoning test),
  - tip šole glede na spol (mešana, za fante, za dekleta),
  - tip šole glede na veroizpoved (javna, anglikanska, rimokatoliška),
- atributi, vezani na učenca:
  - spol (moški, ženski),
  - skupina pri testu govorno-razumskih sposobnosti (VR Band 1 — najboljša, VR Band 2, VR Band 3 — najslabša),
  - etnična pripadnost (Anglija/Škotska/Wales/Irska, Afrika, Bližnji vzhod, Bangladeš, Karibi, Grčija, Indija, Pakistan, SV Azija, Turčija, ostalo).

V naših eksperimentih smo uporabili enake podatke, kot so jih uporabili v [11, 14, 16] in so javno dostopni na spletni strani [55] avtorjev prvega članka [11]. Tam so prvotne kategorične atribute zamenjali s po enim dvojiškim atributom za vsako od vrednosti prvotnega atributa. Tako so dobili 27 atributov. Dodali so še t. i. konstantni atribut (za vse primere ima enako vrednost) z vrednostjo 1, tako da imajo podatki učnih nalog skupaj 28 atributov. Ciljni atribut je število točk, doseženih na zaključnem preizkusu znanja, ki lahko zavzame vrednosti med 1 in 70.

Pri testiranju različnih MTL-metod smo podatke za vsako učno nalogo razdelili na učno in testno množico v razmerju $75 : 25$, kar je enako kot v [11]. Tako smo na koncu dobili MTL-problem s 139 učnimi nalogami, kjer ima vsaka naloga ~ 80 (učnih) primerov, ~ 30 primerov vsake naloge pa smo dali na stran za testiranje. Ker gre za realen MTL-problem, števila gruč različnih tipov učnih nalog (tretja lastnost MTL-problema, podana v definiciji 2.3) ne poznamo in je lahko katerokoli število med ena in 139.

	RMSE	MAE
ERM	10.11 ± 0.08	8.13 ± 0.06
MergeAll	10.18 ± 0.07	8.20 ± 0.05
NoMerging	10.23 ± 0.07	8.22 ± 0.05

Tabela 3.3: Povprečne vrednosti RMSE in MAE različnih MTL-metod za problem *School*. Povprečja so izračunana čez vse učne naloge (šole), utežene glede na število primerov posamezne naloge. Intervali napake podajajo 95% konfidenčne intervale za povprečja.

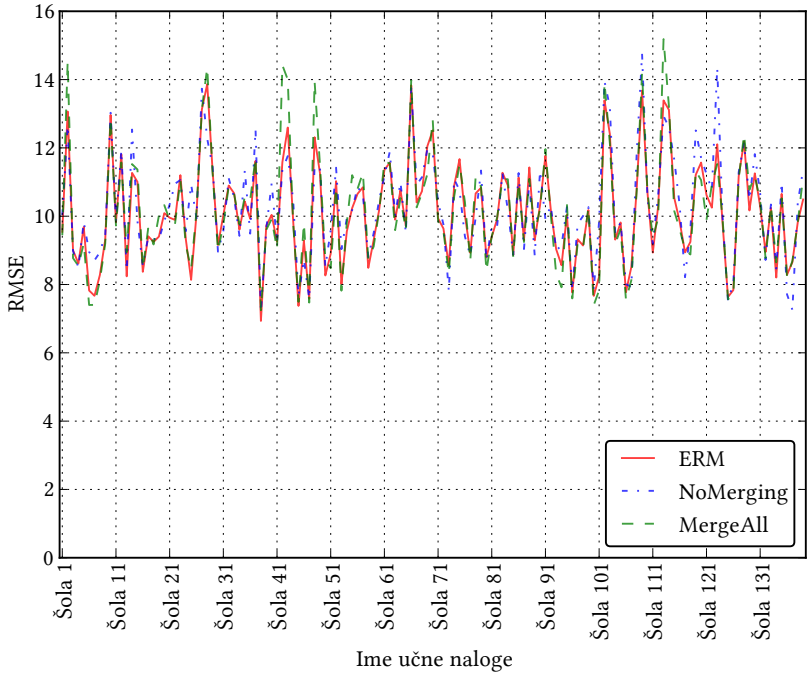
Za oceno RMSE in MAE za celoten MTL-problem smo izračunali povprečje čez vse učne naloge (šole), utežene glede na število primerov posamezne naloge. Poskus smo ponovili 10-krat ter izračunali povprečje in njegov 95% konfidenčni interval. Rezultati so podani v tabeli 3.3. Čas enega zagona metode *ERM* je bil okoli 450 s.

Po obeh merah se je najbolje odrezala metoda *ERM*, sledi metoda *MergeAll*, najslabše rezultate pa smo dosegli z metodo *NoMerging*. Ker se 95% konfidenčna intervala za povprečji metod *ERM* in *MergeAll* za obe meri prekrivata, smo naredili še dvostranski t-test za ugotavljanje statistično značilne razlike njunih povprečij. Vrednost testne statistike je bila za RMSE ≈ 1.32, za MAE pa ≈ 1.57, kar je precej manj od kritične t-vrednosti, ki pri tem številu prostostnih stopenj za $\alpha = 0.05$ znaša ≈ 2.10. To pomeni, da med rezultati metod *ERM* in *MergeAll* ni statistično značilno razlike pri stopnji značilnosti $\alpha = 0.05$.

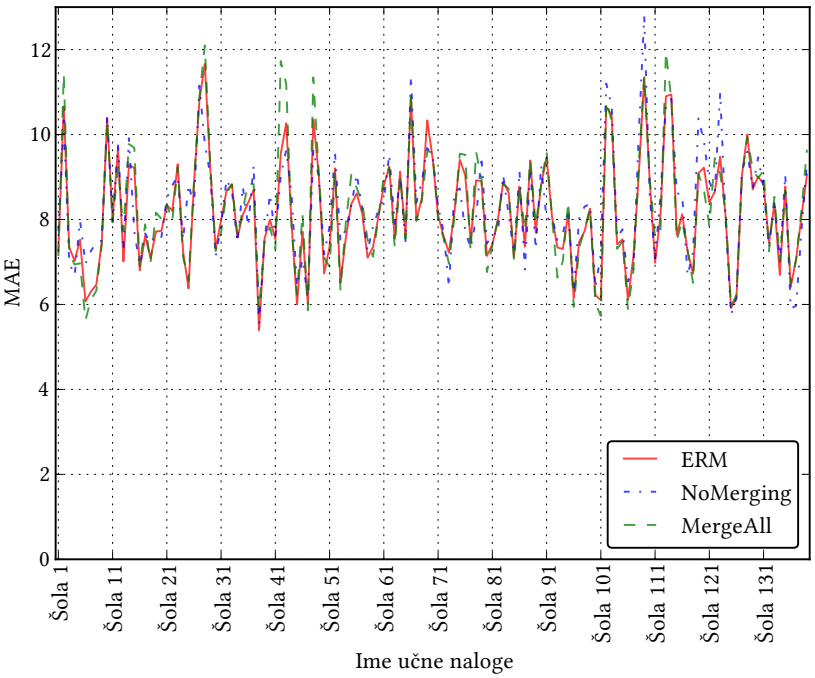
Podrobnejši rezultati, ki prikazujejo dosežen RMSE oz. MAE za vsako učno nalogo posebej, so podani na sliki 3.17. Prva ugotovitev je, da se rezultati za različne učne naloge zelo razlikujejo. Vrednosti RMSE variirajo med 7 in 15, vrednosti MAE pa med 5 in 13.

Na sliki 3.17(a), ki prikazuje rezultate za RMSE, lahko vidimo, da metoda *ERM* za nobeno učno nalogo ne daje najslabših rezultatov (glede na preostali dve metodi). Na sliki 3.17(b), ki prikazuje rezultate za MAE, pa se to zgodi le enkrat. Najboljše rezultate za RMSE dosega metoda *ERM* pri štirih učnih nalogah, najboljše rezultate za MAE pa pri treh učnih nalogah. Lahko bi rekli, da rezultati za metodo *ERM* glede na različne učne naloge najmanj variirajo.

Pri meri RMSE ima največ negativnih odstopanj (bistveno večje vrednosti mere) glede na preostali dve metodi metoda *MergeAll*, pri meri MAE pa metoda *NoMerging*. Večjih pozitivnih odstopanj, tako pri vrednostih RMSE kot pri vrednostih MAE, ni pri nobeni metodi.



(a) Rezultati za RMSE



(b) Rezultati za MAE

Slika 3.17: Posamični rezultati učnih nalog MTL-problema *School*. Prikazana so povprečja (izračunana za vsako učno nalogo posebej) desetih ponovitev poskusa.

Metoda *ERM* učne naloge združuje v gruče na aglomerativen način, podoben aglomerativnim metodam hierarhičnega gručenja. Zato lahko potek združevanja vizualiziramo z dendrogramom. Na sliki 3.18 so prikazani dendrogrami za različne ponovitve eksperimenta.

Opazimo lahko, da se na koncu oblikuje različno število gruč. Za prve štiri ponovitve poskusa je število gruč med 8 in 11. Gruče, ki se oblikujejo, so lahko zelo različnih velikosti. V 1. ponovitvi poskusa (slika 3.18(a)) sta se na primer oblikovali dve zelo veliki gruči (> 25 učnih nalog), dve srednje veliki gruči (> 10 učnih nalog) in nekaj manjših gruč. Opazili smo, da se je v vseh ponovitvah poskusa oblikovala vsaj ena gruča z več kot 30 učnimi nalogami ter vsaj dve gruči s srednje velikim številom učnih nalog (med 10 in 20). Pri takem realnem MTL-problemu pravega števila gruč in pripadnosti posameznih učnih nalog določenim gručam ne poznamo, zato nam lahko ti rezultati pomagajo pri ugotavljanju resnične strukture MTL-problema.

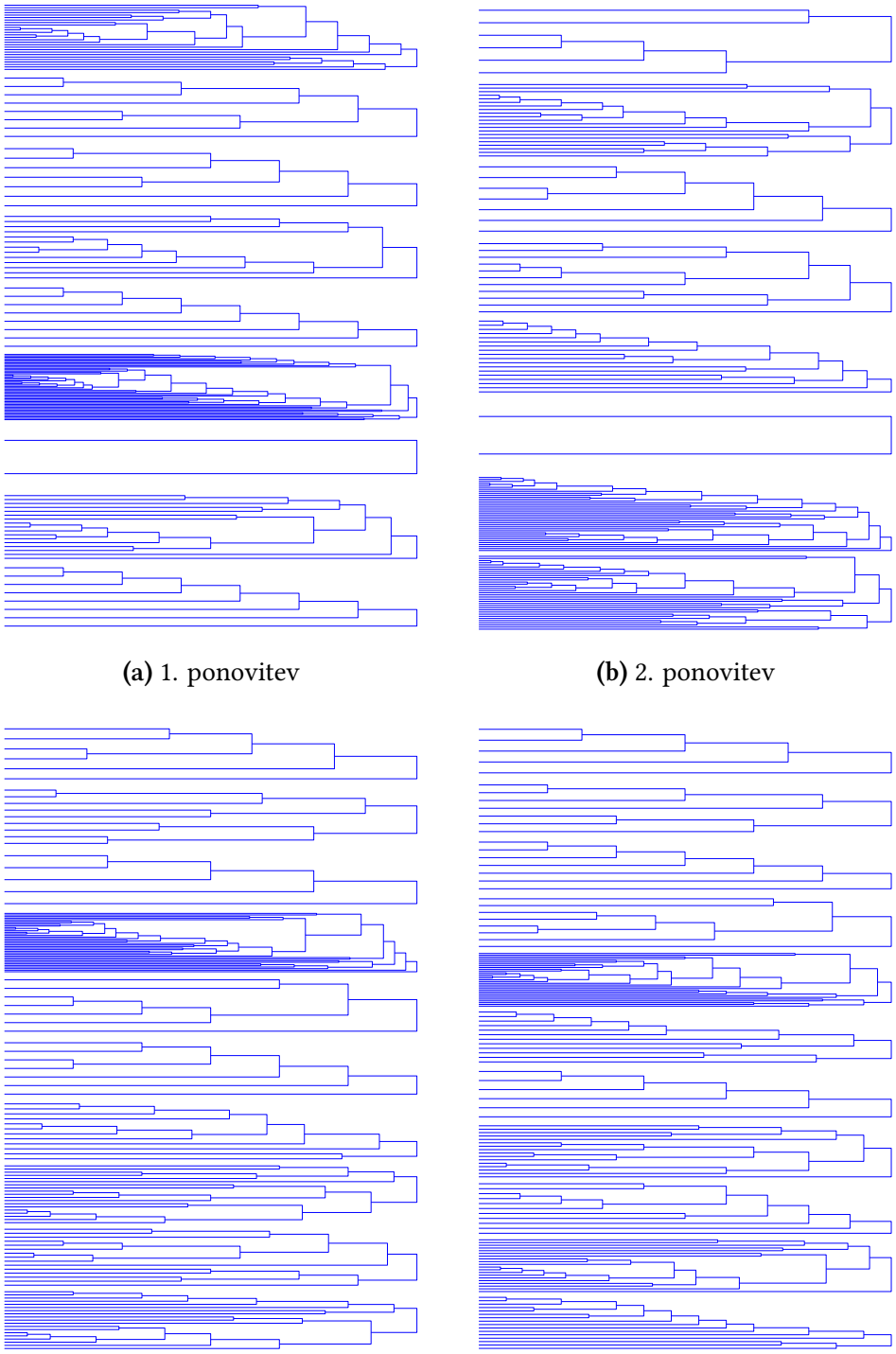
Zanimivo je tudi, koliko parov učnih nalog izmed vseh možnih parov (teh je $\binom{139}{2} = 9591$) je bilo kandidatov za združevanje (glede na kriterije, podane v podrazdelku 2.3.2). V povprečju jih je bilo kar ~ 6000. To nam nakazuje, zakaj je na koncu tako velik delež učnih nalog združenih v gruče.

3.3.1.1 Primerjava s sorodnimi metodami

Za umestitev rezultatov testiranih MTL-metod (*ERM*, *NoMerging* in *MergeAll*) smo jih primerjali z rezultati sorodnih metod. Zaradi tehnične zahtevnosti implementacije direktne primerjave s sorodnimi metodami, smo se omejili le na primerjavo z rezultati, podanimi v člankih. Eksperimente z MTL-metodami *ERM*, *NoMerging* in *MergeAll* smo izvedli tako, da smo v največji možni meri sledili opisom iz člankov.

Prva primerjava zajema rezultate iz članka [16]. V njem so pri testiranju različnih MTL-metod podatke za vsako učno nalogo razdelili na učno in testno množico v razmerju 60 : 40. Pri teh pogojih smo ponovno pognali eksperiment za MTL-metode *ERM*, *NoMerging* in *MergeAll*. Naredili smo 10 ponovitev ter izračunali povprečen RMSE čez vse učne naloge (utežen glede na število primerov posamezne naloge) in njegov standardni odklon. Rezultati so podani v tabeli 3.4.

Najbolje se je odrezala metoda *Group-overlap MTL*, sledi ji metoda *ERM*, na tretjem mestu pa sta metodi *Disjoint-group MTL* in *No-group MTL*. Najslabše se je odrezala metoda *NoMerging*, medtem ko je rezultat metode *MergeAll* le nekoliko slabši od najboljših metod. Opazimo lahko, da se rezultat metode *NoMerging*, ki smo ga dobili



(a) 1. ponovitev

(b) 2. ponovitev

(c) 3. ponovitev

(d) 4. ponovitev

Slika 3.18: Dendrogrami, ki prikazujejo potek združevanja učnih nalog za različne ponovitve eksperimenta za MTL-problem *School*. Imena učnih nalog so zaradi nepreglednosti izpuščena.

Vir rezultatov	Ime metode	RMSE
pričujoča disertacija	*NoMerging*	10.40 ± 0.09
	MergeAll	10.21 ± 0.08
	ERM	10.15 ± 0.09
Kumar in Daumé III [16]	*NoMerging* (STL)	10.67 ± 0.20
	No-group MTL [11]	10.18 ± 0.15
	Disjoint-group MTL [14]	10.18 ± 0.20
	Group-overlap MTL [16]	10.04 ± 0.24

Tabela 3.4: Povprečne vrednosti RMSE različnih MTL-metod za problem *School*. Povprečja so izračunana čez vse učne naloge (šole), utežene glede na število primerov posamezne naloge. Intervali napake podajajo standardne odklone povprečij.

v naših eksperimentih, kar precej razlikuje od rezultata v članku avtorjev Kumar in Daumé III [16]. Domnevamo, da je vzrok, da prišlo do takšne razlike, v tem, da so v [16] uporabili nekoliko drugačno regresijsko metodo. Poleg tega je bila vrednost regularizacijskega parametra v njihovih eksperimentih konstantna, v disertaciji pa smo jo določili z internim prečnim preverjanjem. Druga zanimiva opazka je, da so standardni odkloni rezultatov metod iz članka [16] v povprečju približno dvakrat večji od standardnih odklonov metod iz disertacije.

Število ponovitev eksperimenta v [16] ni bilo podano, zato nismo mogli preveriti, ali so razlike med uspešnostjo metod tudi statistično značilne.

Druga primerjava zajema rezultate iz članka [11]. Način testiranja MTL-metod je enak, kot smo ga uporabili in opisali v podrazdelku 3.3.1. Kot v [11] smo za mero uporabili *pojasnjeno varianco* (angl. *explained variance*). Rezultati so podani v tabeli 3.5.

Tokrat se je najbolje odrezala metoda *MergeAll* (kot smo jo implementirali v disertaciji), na drugem mestu sta bili metodi *ERM* in *MTL-FEAT (linear kernel)*. Dvostranska t-testa za primerjavo povprečij vseh treh metod sta pokazala, da pri dani natančnosti podanih povprečij in standardnih odklonov med metodami ni statistično značilnih razlik pri stopnji značilnosti $\alpha = 0.05$. Najslabše sta se odrezali metoda *NoMerging* (kot smo jo implementirali v disertaciji) ter metoda *MergeAll* (kot je implementirana v [11]). Medtem ko se rezultata za metodo *NoMerging* (kot smo jo implementirali v disertaciji) ter enako metodo iz članka [11], kjer so jo poimenovali Independent, zelo malo razlikujeta, se rezultata za metodo *MergeAll* (kot smo jo implementirali v disertaciji) ter enako metodo iz članka [11], kjer so jo poimenovali Aggregate, bistveno razlikujeta. Implementacija *MergeAll* iz [11] daje najslabši rezultat izmed vseh me-

Vir rezultatov	Ime metode	Pojasnjena varianca
pričujoča disertacija	*NoMerging*	0.23 ± 0.02
	MergeAll	0.28 ± 0.02
	ERM	0.27 ± 0.02
Argyriou et al. [11]	*NoMerging* (Independent)	0.24 ± 0.02
	MergeAll (Aggregate)	0.23 ± 0.01
	MTL-FEAT (variable selection) [11]	0.25 ± 0.02
	MTL-FEAT (linear kernel) [11]	0.27 ± 0.02
	MTL-FEAT (Gaussian kernel) [11]	0.26 ± 0.02

Tabela 3.5: Povprečne vrednosti pojasnjene variance različnih MTL-metod za problem *School*. Povprečja so izračunana čez vse učne naloge (šole), utežene glede na število primerov posamezne naloge. Intervali napake podajajo standardne odklone povprečij.

tod, medtem ko implementacija iz disertacije v eksperimentih daje najboljši rezultat. Del takšne razlike lahko pojasnimo z uporabo različnih regresijskih metod. Prav tako je bile nekoliko drugačno avtomatsko določanje regularizacijskega parametra z internim prečnim preverjanjem. Vseeno to še ne pojasni, zakaj se je metoda *MergeAll* (kot smo jo implementirali v disertaciji) odrezala bolje od vseh ostalih metod, četudi njena prednost pred drugouvrščenima *ERM* in *MTL-FEAT (linear kernel)* ni bila velika. Tokrat lahko opazimo, da so standardni odkloni metod iz članka [11] primerljivi s standardnimi odkloni metod iz disertacije.

Iz obeh primerjav lahko ugotovimo, da metoda *ERM* daje rezultate, ki so primerljivi s sorodnimi MTL-metodami. Druga ugotovitev pa je, da so v implementacijah enakih metod določene nekonsistentnosti, ki jih lahko do neke mere pojasnimo z razliko v uporabljeni osnovni učni metodi ter načinu določanja vrednosti regularizacijskega parametra.

3.3.2 Problem *Computer Survey*

Podatki tega MTL-problema so bili prvotno zbrani in uporabljeni v članku [56]. V disertaciji smo uporabili njihovo pretvorbo v MATLAB-obliko, ki je javno dostopna na spletni strani avtorjev odprtokodnega projekta *pmtkdata* [57], ki nudi zbirko podatkov za knjižnico za strojno učenje *PMTK* [58].

Podatki so bili zbrani z anketiranjem 190 oseb, ki so podali svojo naklonjenost nakupu vsakega od 20 ponujenih osebnih računalnikov. Osebe v MTL-kontekstu

predstavljajo učne naloge, osebni računalniki pa primere. Vsak primer je opisan s 13 atributi, ki zajemajo tako računalnikove notranje kot tudi zunanje lastnosti:

- telefonska pomoč uporabnikom (da, ne),
- velikost pomnilnika (8 MB, 16 MB),
- velikost zaslona (14 palcev, 17 palcev),
- hitrost procesorja (50 MHz, 100 MHz),
- velikost trdega diska (340 MB, 730 MB),
- CD-ROM enota (da, ne),
- predpomnilnik (128 KB, 256 KB),
- barva (bež, črna),
- razpoložljivost (samo naročilo po pošti, samo v računalniški trgovini),
- garancija (1 leto, 3 leta),
- vključena pisarniška programska oprema (da, ne),
- garancija vračila denarja (brez, do 30 dni),
- cena (2000 $, 3500 $).

Vse atribute so rekodirali v dvojiške atribute z vrednostma -1 in 1. Dodali so jim še t. i. konstantni atribut (za vse primere ima enako vrednost) z vrednostjo 1, tako da imajo podatki učnih nalog skupaj 14 atributov. Ciljni atribut je naklonjenost nakupu osebnega računalnika, ki so ga merili na celoštevilski lestvici od 0 do 10.

Pri testiranju različnih MTL-metod smo podatke za vsako učno nalogo razdelili na učno in testno množico v razmerju $75 : 25$, kar je enako kot v [16]. Tako smo na koncu dobili MTL-problem s 190 učnimi nalogami, kjer ima vsaka naloga 15 (učnih) primerov, 5 primerov vsake naloge pa smo dali na stran za testiranje. Ker gre za realen MTL-problem, števila gruč različnih tipov učnih nalog (tretja lastnost MTL-problema, podana v definiciji 2.3) ne poznamo in je lahko katerokoli število med ena in 190.

Za oceno RMSE in MAE za celoten MTL-problem smo izračunali povprečje čez vse učne naloge (osebe). Poskus smo ponovili 10-krat ter izračunali povprečje in njegov

95% konfidenčni interval. Rezultati so podani v tabeli 3.6. Odebeljene so tiste vrednosti RMSE in MAE, ki so statistično značilno boljše od preostalih dveh. Čas enega zagona metode *ERM* je bil okoli 350 s.

	RMSE	MAE
ERM	**2.02** ± 0.03	**1.72** ± 0.03
NoMerging	2.17 ± 0.03	1.83 ± 0.03
MergeAll	2.29 ± 0.01	2.01 ± 0.02

Tabela 3.6: Povprečne vrednosti RMSE in MAE različnih MTL-metod za problem *Computer Survey*. Povprečja so izračunana čez vse učne naloge (osebe). Intervali napake podajajo 95% konfidenčne intervale za povprečja. Odebeljene so tiste vrednosti RMSE in MAE, ki so statistično značilno boljše od preostalih dveh.

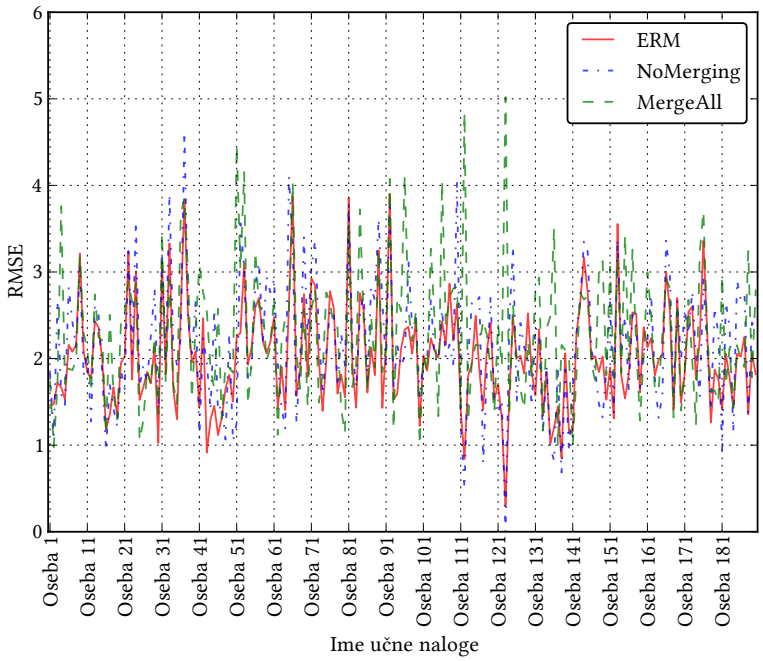
Tudi tokrat se je po obeh merah najbolje odrezala metoda *ERM*. Sledi ji metoda *NoMerging*, najslabše rezultate pa smo dosegli z metodo *MergeAll*. Doseženi rezultati vseh treh metod se statistično značilno razlikujejo (podrobna razlaga izračuna konfidenčnih intervalov ter povezave med konfidenčnimi intervali in statistično značilnostjo je podana v uvodnem delu 3. poglavja).

Podrobnejši rezultati, ki prikazujejo dosežen RMSE oz. MAE za vsako učno nalogo posebej, so podani na sliki 3.19. Takoj lahko opazimo, da se rezultati za različne učne naloge zelo razlikujejo. Vrednosti RMSE in MAE variirajo med 0 in 5.

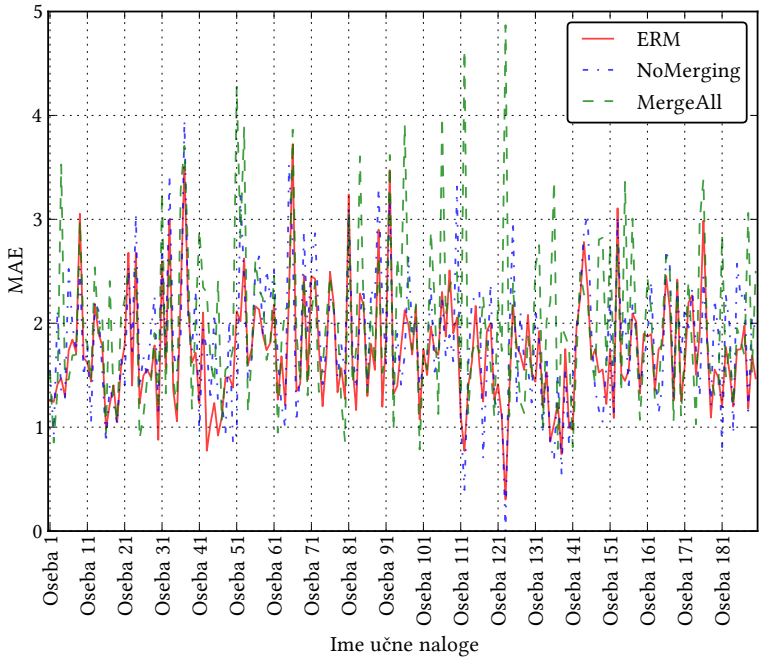
Na sliki 3.17(a), ki prikazuje rezultate za RMSE, lahko vidimo, da metoda *ERM* le za pet izmed 190 učnih nalog doseže slabši rezultat od preostalih dveh metod. Najboljše rezultate izmed vseh treh metod pa doseže za $\sim$ 15 učnih nalog. Na sliki 3.17(b), ki prikazuje rezultate za MAE, je metoda *ERM* slabša od preostalih dveh metoda le za štiri učne naloge. Najbolje pa se odreže pri $\sim$ 15 učnih nalogah. Prav tako lahko opazimo, da rezultati za metodo *ERM*, tako za RMSE kot za MAE, veliko manj variirajo kot pri ostalih dveh metodah.

Pri obeh merah ima največ negativnih odstopanj (bistveno večje vrednosti RMSE oz. MAE) glede na preostali dve metodi metoda *MergeAll*. Nekaj pozitivnih odstopanj glede na preostali dve metodi imata pri obeh merah metodi *NoMerging* in *MergeAll*.

Metoda *ERM* učne naloge združuje v gruče na aglomerativen način, podoben aglomerativnim metodam hierarhičnega gručenja. Zato lahko potek združevanja vizualiziramo z dendrogramom. Na sliki 3.20 so prikazani dendrogrami za različne ponovitve eksperimenta.

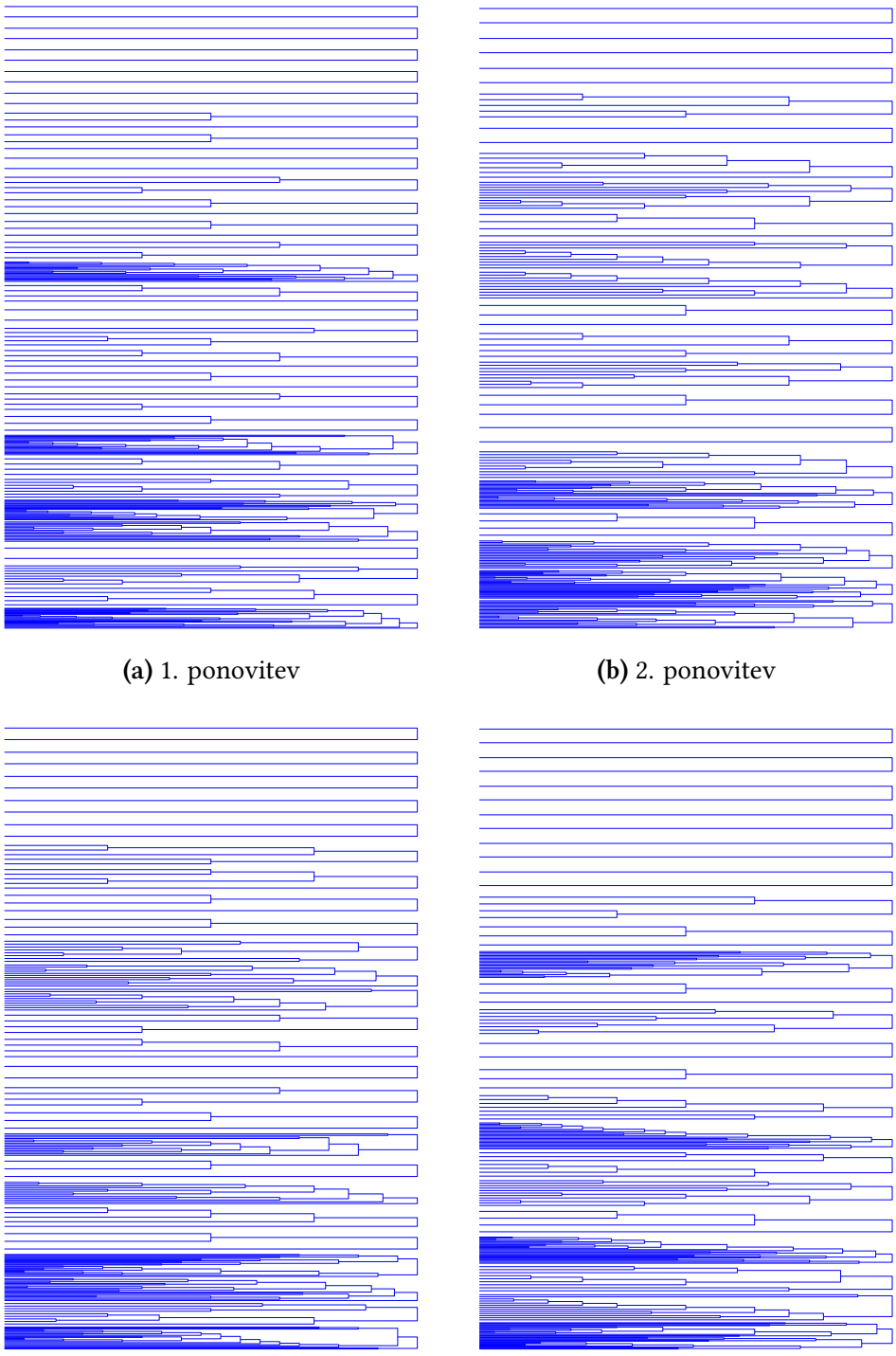


(a) Rezultati za RMSE



(b) Rezultati za MAE

Slika 3.19: Posamični rezultati učnih nalog MTL-problema *Computer Survey*. Prikazana so povprečja (izračunano za vsako učno nalogo posebej) desetih ponovitev poskusa.



(a) 1. ponovitev

(b) 2. ponovitev

(c) 3. ponovitev

(d) 4. ponovitev

Slika 3.20: Dendrogrami, ki prikazujejo potek združevanja učnih nalog za različne ponovitve eksperimenta za MTL-problem *Computer Survey*. Imena učnih nalog so zaradi nepreglednosti izpuščena.

Iz slik lahko razberemo, da se na koncu oblikuje precej različno število gruč — za prve štiri ponovitve poskusa med 22 in 29. Gruče, ki se oblikujejo, so tudi zelo različnih velikosti. Nekatere imajo le po dve učni nalogi, druge imajo srednje veliko število učnih nalog (med 10 in 20), vedno pa obstaja tudi nekaj gruč s po več kot 20 učnimi nalogami. Pri takem realnem MTL-problemu pravega števila gruč in pripadnosti posameznih učnih nalog določenim gručam ne poznamo, zato nam lahko ti rezultati pomagajo pri ugotavljanju resnične strukture MTL-problema.

Ponovno je zanimivo opazovati, koliko parov učnih nalog izmed vseh možnih parov (teh je $\binom{190}{2} = 17955$) je bilo kandidatov za združevanje (glede na kriterije, podane v podrazdelku 2.3.2). V povprečju jih je bilo ~ 6000. Ta delež je sicer manjši kot pri MTL-problemu *School*, vendar še vedno relativno visok, s čimer nakazuje, zakaj je na koncu relativno velik delež učnih nalog združenih gruče.

3.3.2.1 Primerjava s sorodnimi metodami

Za umestitev rezultatov testiranih MTL-metod (*ERM*, *NoMerging* in *MergeAll*) smo jih primerjali s rezultati sorodnih metod. Zaradi tehnične zahtevnosti implementacije direktne primerjave s sorodnimi metodami, smo se omejili le na primerjavo z rezultati, podanimi v člankih. Eksperimente z MTL-metodami *ERM*, *NoMerging* in *MergeAll* smo izvedli tako, da smo v največji možni meri sledili opisom iz člankov.

Naredili smo primerjavo z rezultati iz člankov [16] in [11]. V prvem so uporabili enak način testiranja MTL-metod, kot smo ga uporabili in opisali v podrazdelku 3.3.2, v drugem pa so pri testiranju različnih MTL-metod podatke za vsako učno nalogo razdelili na učno in testno množico v razmerju $80 : 20$. Rezultati so podani v tabeli 3.7.

Najbolje se je odrezala metoda *Group-overlap MTL*, na drugem mestu je bila metoda *MTL-FEAT (Gaussian kernel)*. Sledijo jima metode *MTL-FEAT (linear kernel)*, *MTL-FEAT (variable selection)*, *Disjoint-group MTL* ter *ERM*. Že naštetim metodam sledita metodi *NoMerging* in *MergeAll*, kot sta implementirani v disertaciji, nato pa metoda *NoMerging*, kot so jo implementirali v članku [16], in metoda *NoMerging*, kot so jo implementirali v članku [11]. Opazimo lahko, da so standardni odkloni metod iz disertacije v povprečju za približno polovico manjši od standardnih odklonov metod iz članka [16]. Za metode *MTL-FEAT* standardni odkloni niso podani, saj so eksperiment v [11] ponovili le enkrat.

Ker število ponovitev eksperimenta ni podano oz. je ponovitev ena sama, ne moremo preveriti, ali so razlike med uspešnostjo metod tudi statistično značilne.

Vir rezultatov	Ime metode	RMSE
pričujoča disertacija	*NoMerging*	2.17 ± 0.04
	MergeAll	2.29 ± 0.02
	ERM	2.02 ± 0.05
Kumar in Daumé III [16]	*NoMerging* (STL)	2.70 ± 0.10
	No-group MTL [11]	2.06 ± 0.07
	Disjoint-group MTL [14]	2.01 ± 0.10
	Group-overlap MTL [16]	1.76 ± 0.09
Argyriou et al. [11]	*NoMerging* (Independent)	3.88
	MTL-FEAT (variable selection) [11]	2.01
	MTL-FEAT (linear kernel) [11]	1.93
	MTL-FEAT (Gaussian kernel) [11]	1.85

Tabela 3.7: Povprečne vrednosti RMSE različnih MTL-metod za problem *Computer Survey*. Povprečja so izračunana čez vse učne naloge (osebe). Intervali napake podajajo standardne odklone povprečij.
V članku avtorjev Argyriou et al. standardni odkloni niso podani, saj so eksperiment ponovili le enkrat. Prav tako je razmerje delitve primerov na učno in testno množico 80 : 20, pri ostalih dveh člankih pa 75 : 25.

Rezultat metode *No-group MTL* iz članka [16], ki naj bi bila implementacija metode iz članka [11], je sicer nekoliko slabši od rezultatov, kot jih podajajo v prvotnem članku (rezultati metod *MTL-FEAT*). Ker za slednje ni podan standardni odklon, dopuščamo možnost, da so razlike med rezultati statistično neznačilne. Drugače je pri rezultatih za metodo *NoMerging*, ki jo podajajo vsi trije viri. Tam so odstopanja ogromna. Odstopanje med rezultatoma iz disertacije in članka [16] bi lahko do določene mere pojasnili s tem, da so v [16] uporabili nekoliko drugačno regresijsko metodo ter da je bila vrednost regularizacijskega parametra v njihovih eksperimentih konstantna, v disertaciji pa smo jo določili z internim prečnim preverjanjem. Domnevamo, da je bilo slednje še posebej pomembno, saj je bilo število primerov za posamezno učno nalogo zelo majhno (20), varianca v podatkih različnih učnih nalog pa precej visoka. Zelo velikega odstopanja rezultata metode *NoMerging*, kot je podan v članku [11], ne znamo zadovoljivo pojasniti. Domnevamo, da je problem v tem, da so v [11] eksperiment ponovili le enkrat in je po naključju prišlo do neobičajne delitve primerov na učne in testne množice. Posledica tega so bili zelo slabi rezultati dobljenih modelov.

Primerjava je pokazala, da so rezultati metode *ERM* slabši od metode *Group-overlap MTL* [16] ter primerljivi z rezultati metod *Disjoint-group MTL* [14] in *No-group MTL*

[11]. Nezaupanje v rezultate iz članka [11] povzroča odsotnost večkratne ponovitve eksperimenta ter zelo veliko odstopanje rezultata za metodo *NoMerging* v primerjavi z rezultatoma iz disertacije in članka [16]. Nekonsistentnosti med rezultatoma iz disertacije in članka [16] lahko do določene mere pojasnimo z razliko v uporabljeni osnovni učni metodi ter načinu določanja vrednosti regularizacijskega parametra.

3.4 Ugotovitve

V eksperimentalni študiji smo preučevali obnašanje metode *ERM* glede na različne lastnosti MTL-problema. V prvem delu smo raziskovali njeno obnašanje na sintetičnih MTL-problemih učenja Boolovih funkcij, kjer smo sami izbirali lastnosti MTL-problemov. Generirali smo MTL-probleme z različnim številom učnih nalog (lastnost iz definicije 2.1), različnim številom primerov, ki pripadajo posameznim učnim nalogam (lastnost iz definicije 2.2), ter različnim številom gruč različnih tipov učnih nalog (lastnost iz definicije 2.3). Poleg tega smo preučili tudi njeno uspešnost pri različnih stopnjah šuma v podatkih ter kako se spreminja njeno delovanje, če znotraj nje uporabimo različne osnovne učne metode. V drugem delu smo preučili obnašanje metode *ERM* na dveh realnih klasifikacijskih MTL-problemih prepoznavanja ročno napisanih števk. V tretjem delu smo preučili še njeno obnašanje na dveh realnih regresijskih MTL-problemih. Pri prvem je šlo za nalogo napovedovanja števila točk učencev na zaključnem preizkusu znanja, pri drugem pa za napovedovanje naklonjenosti oseb nakupu določenega računalnika. Za umestitev rezultatov MTL-metod, ki smo jih testirali na realnih klasifikacijskih in regresijskih MTL-problemih, smo jih primerjali še z rezultati sorodnih metod iz člankov [11, 14, 16].

Sintetični MTL-problemi

Najboljši vpogled v delovanje metode *ERM* smo imeli pri eksperimentih na sintetičnih MTL-problemih, saj smo v tem primeru sami nadzorovali njihove lastnosti. Pri poskusu s spreminjajočim številom učnih nalog se je pokazalo, da ima s povečevanjem števila učnih nalog v vsaki gruči oz. celotnega števila učnih nalog (lastnost iz definicije 2.1) metoda *ERM* večji potencial za združevanje podatkov, saj lahko pri združevanju podatkov izbira med več učnimi nalogami. Rezultat tega je, da se pri tem poskusu metoda *ERM* odlično odreže in že pri dveh učnih nalogah v vsaki gruči dosega boljše rezultate od metod *NoMerging* in *MergeAll*, od petih učnih nalog naprej

pa je razlika v AUC že zelo velika (~ 0.1) in s povečevanjem števila učnih nalog le še narašča. Nekoliko nižji AUC metode *ERM* v primerjavi z metodo *Oracle* nakazuje na konzervativnost kriterijev za združevanje, podanih v podrazdelku 2.3.2.

Pri vseh štirih MTL-problemih poskusa s spreminjajočim številom primerov posameznih učnih nalog je AUC metode *ERM* hitro naraščal in se približeval metodi *Oracle*. Za 100 ali več primerov na posamezno učno nalogo je metoda *ERM* dosegala enako dobre rezultate kot metoda *Oracle*. To kaže, da je povečanje števila primerov posamezne učne naloge za ERM zelo koristno, saj tako lažje odkrije gruče sorodnih učnih nalog in združi njihove učne naloge.

Pri poskusu s spreminjajočim številom gruč različnih tipov učnih nalog se je izkazalo, da se razlika med metodama *ERM* in *Oracle* s povečevanjem števila gruč sorodnih učnih nalog povečuje. To nakazuje šibkost metode *ERM*, ki ob povečanem številu gruč težje poišče gruče sorodnih učnih nalog, ki ustrezajo prvotnim gručam, kar se kaže v padcu njene zmogljivosti. Torej tretja lastnost MTL-problemov, podana v definiciji 2.3, negativno vpliva na zmogljivost metode *ERM*.

Pri poskusu, kjer smo povečevali stopnjo šuma v podatkih, je bil padec vrednosti AUC metode *ERM* pričakovan, zanimive pa so ugotovitve glede njegove dinamike. Razlika med vrednostmi AUC metod *Oracle* in *ERM* se s povečevanjem stopnje šuma vztrajno povečuje. Na začetku (za $noise \leq 0.1$) je *ERM* po absolutnih vrednostih AUC še relativno blizu (z izjemo MTL-problema 4) metodi *Oracle*. Od tam naprej pa začne njena krivulja AUC hitro padati. Za vrednosti $noise \leq 0.3$ je pri vseh MTL-problemih *ERM* še statistično značilno boljša od metod *NoMerging* in *MergeAll*, za večje vrednosti *noise* pa je v nekaterih primerih slabša (za MTL-problema 1 in 4), v enem primeru enaka (za MTL-problem 3) in v enem primeru boljša (za MTL-problem 2) od metode *MergeAll*. Iz rezultatov lahko torej ugotovimo, da pri stopnji šuma $noise = 0.5$ metoda *ERM* že povsem odpove in deluje podobno ali slabše kot metoda *MergeAll*.

V zadnjem eksperimentu na sintetičnih MTL-problemih, kjer smo spreminjali osnovno učno metodo znotraj *ERM*, se je izkazalo, da se v splošnem *ERM* odlično odreže, ne glede na uporabljeno osnovno učno metodo. To so namreč potrdili rezultati poskusov spreminjanja učnih nalog, rezultati poskusov spreminjanja števila primerov posameznih učnih nalog ter rezultati poskusov spreminjanja števila gruč različnih tipov učnih nalog. Le pri poskusih s spreminjanjem stopnje šuma v podatkih so bili trendi padanja krivulj AUC za različne osnovne učne metode nekoliko različni. Pri SVM je krivulja AUC metode *ERM* vseskozi nad krivuljama ostalih dveh metod in le pri največji stopnji šuma ($noise = 0.5$) ni več statistično značilno boljša od metode

MergeAll. Pri odločitvenem drevesu in k-NN pa pride do obrata med vrednostma parametra *noise* 0.2 in 0.3, ko AUC metode *NoMerging* postane večji od AUC metode *ERM*. Na koncu (*noise* = 0.5) je pri obeh omenjenih osnovnih učnih metodah AUC metode *NoMerging* statistično značilno večji od AUC metode *ERM*.

Zanimivo odkritje je povečevanje vrednosti AUC metode *MergeAll* pri poskusu s spreminjanjem števila učnih nalog in poskusu s spreminjanjem števila primerov posameznih učnih nalog. Pri obeh poskusih namreč vrednosti AUC (proti pričakovanjem) s povečevanjem števila učnih nalog oz. s povečevanjem števila primerov posameznih učnih nalog naraščajo. Postopna upočasnitev tega povečevanja nakazuje, da je verjetni razlog za ta pojav majhno število primerov združenih podatkov na začetku poskusa (250) v primerjavi z ogromnim številom na koncu poskusa (5000). Povečano število primerov osnovni učni metodi (v tem primeru SVM) omogoča izgradnjo boljšega splošnega modela (istega za vse učne naloge), kljub temu da je naučen na podatkih učnih nalog, ki pripadajo različnim gručam.

Drugo zanimivo odkritje eksperimentov na sintetičnih MTL-problemih je, da je metoda *MergeAll* pri poskusu s spreminjanjem števila primerov posameznih učnih nalog na začetku (pri majhnem številu učnih primerov) boljša od metod *ERM* in *NoMerging*. Pri 10 primerih na posamezno učno nalogo je bila namreč metoda *MergeAll* praviloma boljša od *ERM* in *NoMerging*. S povečevanjem števila primerov na posamezno učno nalogo se je vrstni red metod obrnil, tako da je najboljša postala metoda *ERM*, sledila ji je metoda *NoMerging*, metoda *MergeAll* pa je pristala na zadnjem mestu. Kdaj se je ta obrat zgodil, je bilo odvisno od posameznega MTL-problema. To odkritje nakazuje na potencialno novo MTL-metodo, ki bi na začetku združila vse učne naloge v eno gručo in potem postopoma, ko bi bilo za vsako učno nalogo na voljo več podatkov, iz te gruče odstranjevala tiste učne naloge, ki ne bi več ustrezale določenim kriterijem za združevanje.

Realni klasifikacijski MTL-problemi

Pri obeh realnih klasifikacijskih MTL-problemih prepoznavanja ročno napisanih števk (*USPS digits* in *MNIST digits*) se je izkazalo, da vsaka učna naloga (po ena za vsako števko) predstavlja svojo gručo učnih nalog istega tipa. To pa pomeni, da *ERM* ne more delovati bolje od metode *NoMerging*, saj ni smiselno združiti nobenega para učnih nalog. Izkaže se, da *ERM* deluje 100% uspešno, saj nikoli ne združi dveh učnih nalog.

V nadaljevanju smo pri obeh MTL-problemih prvotni problem pretvorili v nov MTL-problem, kjer smo prvotne učne naloge razbili na različno število (vsakič izbrano naključno) manjših podnalog. Pri obeh novih MTL-problemih se je metoda *ERM* izkazala na najboljšo. Blizu njej je bila še metoda *NoMerging*, medtem ko je metoda *MergeAll* na teh MTL-problemih povsem neuporabna. Večja razlika v vrednosti AUC za metodi *ERM* in *NoMerging* je prisotna pri učnih nalogah, kjer je AUC metode *NoMerging* nekoliko nižji. Ker metoda *NoMerging* za vsako učno nalogo zgradi model le na njenih podatkih, lahko vidimo, da združevanje z metodo *ERM* najbolj koristi takrat, ko imamo za posamezno učno nalogo na voljo premalo podatkov in bi z več podatki lahko zgradili boljši model. Ravno to naredimo s pomočjo metode *ERM*. Pri poskusih z razbitjem na večjo število podnalog so trendi za oba MTL-problema enaki kot pri razbitju na manjše število podnalog, le da so v prvem primeru rezultati še bolj izrazito v korist metode *ERM*.

Primerjava z rezultati sorodnih metod iz člankov [11, 14, 16] je pokazala, da so rezultati metode *ERM* statistično značilno boljši od sorodnih metod. Pri tem velja omeniti, da so rezultati metode *ERM* enaki metodi *NoMerging*, kar pomeni, da k uspehu *ERM* ne pripomore boljše gručenje učnih nalog. Pri MTL-problemu *MNIST digits* so celo vse metode, ki smo jih implementirali v disertaciji, dajale boljše rezultate od sorodnih MTL-metod. Domnevamo, da je vzrok za to uporaba boljše implementacije logistične regresije ter nekoliko drugačna množica testnih primerov.

Realni regresijski MTL-problemi

Pri prvem realnem regresijskem MTL-problemu (*School*), kjer je šlo za nalogo napovedovanja števila točk učencev na zaključnem preizkusu znanja, se je najbolje odrezala metoda *ERM*, sledila ji je metoda *MergeAll*, najslabše rezultate pa smo dosegli z metodo *NoMerging*. Med doseženimi rezultati metod *ERM* in *MergeAll* ni statistično značilno razlike pri stopnji značilnosti $\alpha = 0.05$. Podrobnejša analiza rezultatov za vsako učno nalogo posebej je razkrila, da se le-ti za različne učne naloge zelo razlikujejo. Ugotovili smo, da metoda *ERM* za nobeno učno nalogo ne daje najslabših rezultatov (glede na preostali dve metodi), hkrati pa za nobeno učno nalogo ne daje najboljših rezultatov. Iz tega lahko zaključimo, da rezultati metode *ERM* glede na različne učne naloge najmanj variirajo.

Primerjava z rezultati sorodnih metod iz člankov [11, 16] je razkrila, da metoda *ERM* daje rezultate, ki so primerljivi s sorodnimi MTL-metodami. Druga ugotovi-

tev pa je, da so v implementacijah enakih metod določene nekonsistentnosti, ki jih lahko do neke mere pojasnimo z razliko v uporabljeni osnovni učni metodi ter načinu določanja vrednosti regularizacijskega parametra.

Tudi pri drugem realnem regresijskem MTL-problemu (*Computer Survey*), kjer je šlo za napovedovanje naklonjenosti oseb nakupu določenega računalnika, se je najbolje odrezala metoda *ERM*, sledila ji je metoda *NoMerging*, najslabše rezultate pa smo dosegli z metodo *MergeAll*. Tokrat se doseženi rezultati vseh treh metod statistično značilno razlikujejo. Podrobnejša analiza rezultatov za vsako učno nalogo posebej je tudi tokrat razkrila, da se le-ti za različne učne naloge zelo razlikujejo. Ugotovili smo, da *ERM* le za dve (za RMSE) oz. tri (za MAE) izmed 190 učnih nalog doseže slabši rezultat od preostalih dveh metod, hkrati pa dosega najboljše rezultate za $\sim$ 10 učnih nalog. Opazili smo tudi, da rezultati za metodo *ERM*, tako za RMSE kot za MAE, veliko manj variirajo kot pri ostalih dveh metodah.

Primerjava z rezultati sorodnih metod je pokazala, da so rezultati metode *ERM* slabši od metode *Group-overlap MTL* [16], a primerljivi z rezultati metod *Disjoint-group MTL* [14] in *No-group MTL* [11]. Ponovno smo opazili nekonsistentnosti med rezultati, ki smo jih dobili v disertaciji, ter rezultati iz člankov sorodnih metod za enako osnovno MTL-metodo (*NoMerging*). Medtem ko velikega odstopanja s člankom [11] ne moremo zadovoljivo pojasniti, lahko nekonsistentnost s člankom [16] do določene mere pojasnimo z razliko v uporabljeni osnovni učni metodi ter načinu določanja vrednosti regularizacijskega parametra.

4. POGLAVJE

Študija primera pohitritve učenja robotskega agenta z izvajanjem eksperimentov v kompleksnejšem okolju

V okviru disertacije smo izvedli tudi študijo primera pohitritve učenja robotskega agenta z izvajanjem eksperimentov v kompleksnejšem okolju. Rezultati te študije so bili objavljeni v članku [34].

V uvodnem poglavju (razdelek 4.1) bomo najprej predstavili kontekst, v katerem se je porodila ideja za izvedbo študije. Nato bomo s preprostim motivacijskim primerom pokazali, na kakšen način bi bilo mogoče pohitriti robotovo učenje. V razdelku 4.2 bomo natančno opisali eksperimentalno domeno in predstavili relacijo, ki se je skuša naučiti robot (tj. kako se spremeni njegova razdalja do objekta potem, ko izvede eno od svojih akcij). Definirali bomo tudi pojme *številčna kompleksnost*, *vedenjska kompleksnost* in *vedenjski tip* ter predstavili niz scenarijev z naraščajočo številčno in vedenjsko kompleksnostjo, v katerih se bo robot skušal naučiti prej omenjene relacije. Nato bomo predstavili potek robotovega učenja (razdelek 4.3), ki poteka v neskončni učni zanki. Sledila bosta natančen opis izvedbe eksperimentov (razdelek 4.4) ter predstavitev rezultatov, ki primerjajo zmogljivost metode *ERM* in ostalih MTL-metod (razdelek 4.5). V razdelku 4.6 bomo podali še ugotovitve, ki izhajajo iz predstavljene študije, ter podali možnosti za nadaljnje delo na tem področju.

4.1 Uvod

Pričujoča študija je s področja *učenja z izvajanjem eksperimentov* (angl. *learning by experimentation*), pri katerem se agent (npr. robot) uči relacij med opazovanimi spremenljivkami z izvajanjem eksperimentov v svojem okolju. Z drugimi besedami, agent izvaja akcije in uporablja svoje senzorje za opazovanje učinkov, ki jih imajo le-te na okolje. To mu pomaga odkriti zakone, ki veljajo za okolje in objekte, ki so v njem.

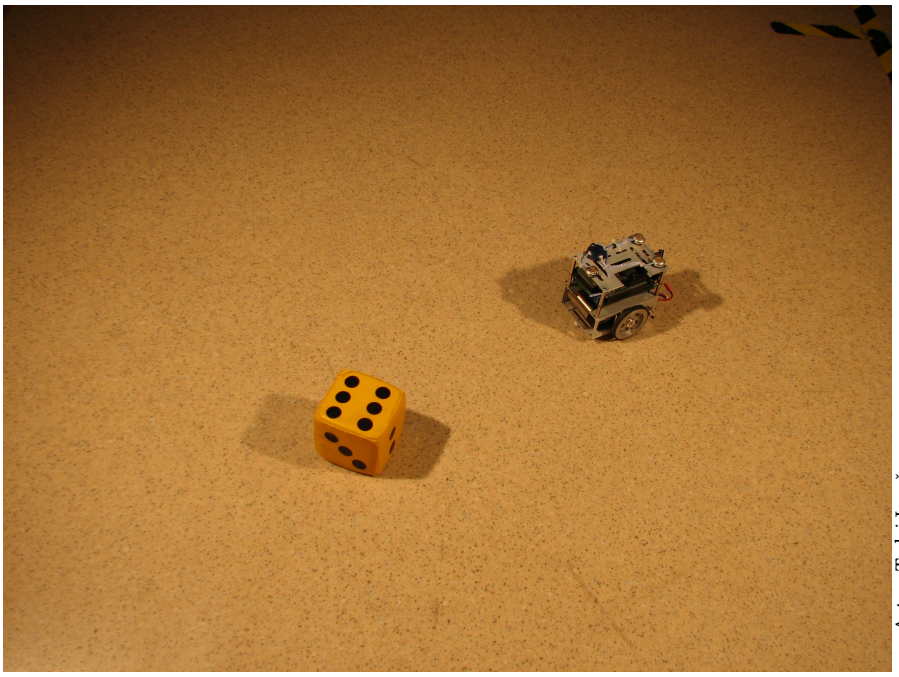
Vzemimo za primer robota, ki se skuša naučiti modela, kako se njegova razdalja do objektov spremeni, ko izvede eno od svojih akcij. Z izvajanjem akcij robot nabira primere in se postopoma nauči modela, ki napoveduje, kako se spremeni robotova razdalja do objektov po izvedbi ene od njegovih akcij. Naučeni model bo lahko robot kasneje uporabil za opravljanje drugih nalog kot npr. ustvarjanje plana za dosego določenega cilja, izogibanje oviram, medtem ko se premika okoli itn.

Vendar lahko učenje traja precej dolgo časa, saj mora robot nabrati dovoljšnjo količino podatkov. Prvo vprašanje, ki ga bomo raziskovali v tej študiji, je, ali je mogoče pohitriti učenje z izvajanjem eksperimentov v kompleksnejšem okolju (tj. okolju z večjim številom različnih objektov), kjer robotovi senzorji opravljajo meritve na več objektih hkrati. Primer, ki ilustrira razliko med preprostim in kompleksnejšim okoljem, je prikazan na sliki 4.1. Izraz pohitritev v kontekstu robotskega učenja pomeni, da robot za dosego določene kvalitete zgrajenih modelov potrebuje manjše število akcij in posledično krajši čas nabira podatke. Naslednje vprašanje, ki ga bomo raziskovali, je, ali je mogoče izvajanje eksperimentov v kompleksnejšem okolju uporabiti kot način paralelnega nabiranja podatkov. V kompleksnejšem okolju bo namreč robot lahko nabral več podatkov v manjšem časovnem obdobju v primerjavi s preprostim svetom z enim samim objektom. Toda, ali bo večja količina podatkov odtehtala večjo kompleksnost učenja v kompleksnejšem okolju? Za ilustracijo nekaterih možnih prednosti in slabosti učenja v kompleksnejšem svetu si oglejmo naslednji motivacijski primer.

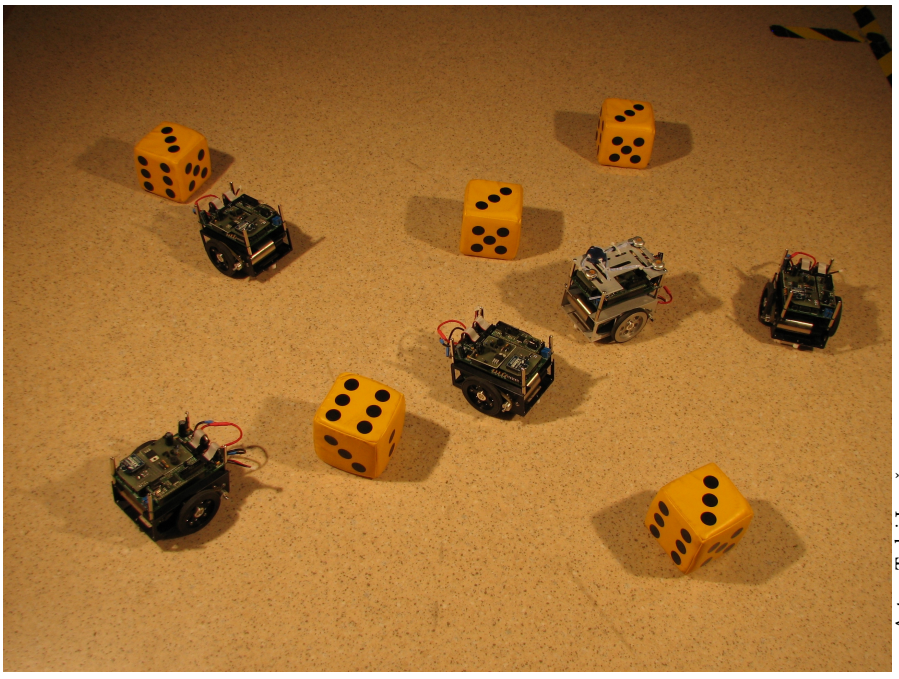
4.1.1 Motivacijski primer

Denimo, da je naloga robota, da se nauči napovedovati, ali se bo po izvedbi ene od svojih akcij približal ali oddaljil od danega objekta. V nadaljevanju bomo uporabljali standardno terminologijo s področja nadzorovanega učenja.

Robot hkrati opazuje tri objekte: A, B in C, kot je prikazano na sliki 4.2. Za vsak

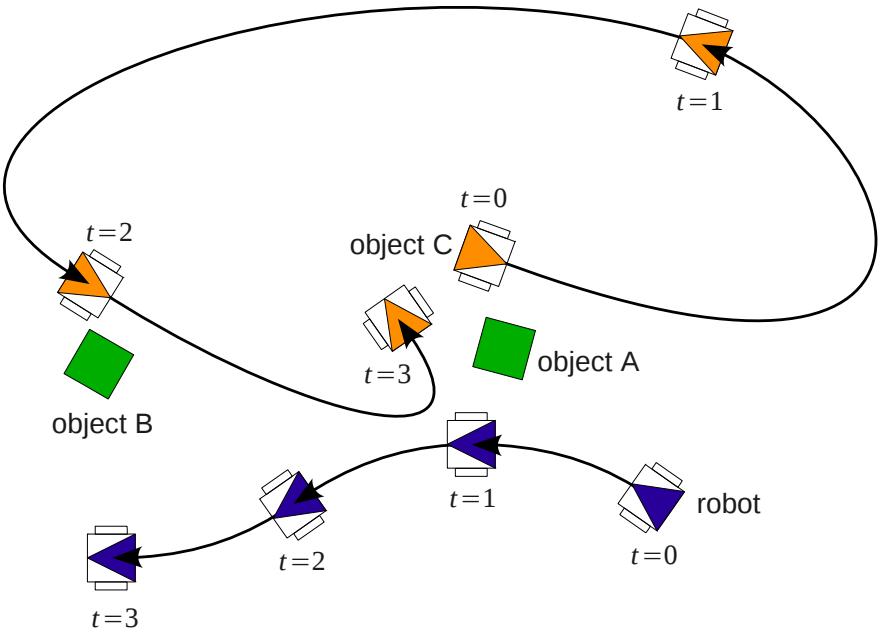


(a) Zelo preprosto okolje: robot in ena kocka.



(b) Kompleksnejše okolje: robot in številni objekti različnih tipov (štiri kocke in štirje drugi roboti).

Slika 4.1: Primer, ki ilustrira razliko med preprostim in kompleksnejšim okoljem. V katerem se bo robot hitreje učil?

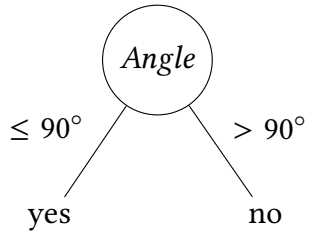


Slika 4.2: Robot sočasno opazuje tri objekte: objekt A (škatla), objekt B (še ena škatla) in objekt C (hitro premikajoči se robot). Iz začetnega položaja ($t = 0$) je robot izvedel tri akcije: "left", "left" in "right" ter se ustavil na končnem položaju ($t = 3$). Medtem se je objekt C prav tako premikal po označeni trajektoriji.

objekt opazuje dva atributa: *Distance* (razdalja med robotom in objektom) in *Angle* (absolutna vrednost kota med robotovo orientacijo in objektom). Atribut *Action* (z možnimi vrednostmi "left", "straight" in "right") označuje smer robotovega premikanja: "straight" pomeni premikanje naravnost naprej, "left" pomeni premikanje naprej z vrtenjem v levo, "right" pa pomeni premikanje naprej z vrtenjem v desno. Razred je atribut *MovedCloser* (z možnima vrednostma "yes" in "no"), ki pove, ali se je robot po izvedbi dane akcije približal objektu ali ne.

Robot nima nobenega znanja o tipu objektov, ki jih opazuje (tj. ne ve, da sta objekta A in B škatli in objekt C hitro premikajoči se robot). Vse, kar pozna, so opazovanja objektov, ki so predstavljena z atributoma *Distance* in *Angle*. V tem motivacijskem primeru bomo privzeli, da robot za izgradnjo napovednega modela uporabi odločitveno drevo z izključenim rezanjem. Na sliki 4.3 je prikazan dober približni model za napovedovanje, ali se bo robot po izvedbi svoje akcije približal nepremikajočemu objektu ali ne.

Denimo, da je robot izvedel tri akcije: "left", "left" in "right", kot je prikazano na sliki 4.2. Med izvajanjem teh akcij je zbral devet primerov, po tri za vsak objekt, kot



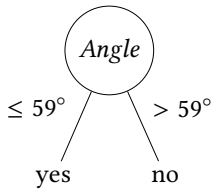
Slika 4.3: Dober približni model za napovedovanje, ali se bo robot po izvedbi svoje akcije približal ne-premikajočemu objektu (tj. objektu *A* ali *B*) ali ne. Velja poudariti, da pri tem odločitvenem drevesu napoved ni odvisna od atributov *Action* in *Distance*.

	Distance	*Angle*	*Action*	*MovedCloser*
objekt *A*	0.37 m	10°	left	yes
	0.17 m	108°	left	no
	0.46 m	179°	right	no
objekt *B*	1.01 m	21°	left	yes
	0.67 m	12°	left	yes
	0.42 m	70°	right	yes
objekt *C*	0.52 m	20°	left	no
	0.81 m	120°	left	yes
	0.53 m	82°	right	no

Tabela 4.1: Tabela podatkov z devetimi primeri, po tremi za vsak objekt, ki jih je robot nabral med izvajanjem akcij, kot je prikazano na sliki 4.2.

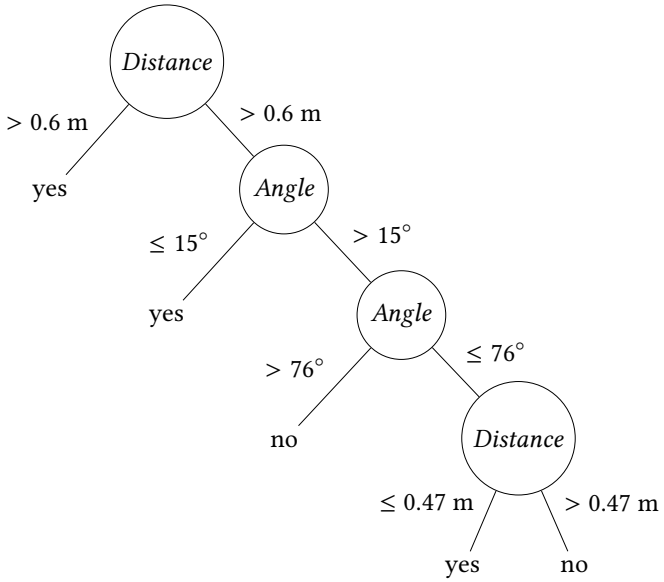
je podano v tabeli 4.1. Na tej točki se je robot ustavil in zgradil odločitveno drevo za vsak objekt. Ker robot ne more zgradi zelo natančnega modela za hitro premikajoči se objekt (tj. objekt *C*), se bomo osredotočili le na napovedna modela za objekta *A* in *B*. Slednja sta podana na slikah 4.4(a) in 4.4(b) (zaporedoma). Odločitveno drevo, zgrajeno na primerih objekta *A*, je podobno dobremu približnemu modelu, predstavljenem na sliki 4.3. Edina razlika je netočen razcep po atributu *Angle* (59° namesto 90°). Po drugi strani je odločitveno drevo, zgrajeno na primerih objekta *B*, precej drugačno od dobrega približnega modela, saj ima le en list, ki vse primere klasificira kot *MovedCloser* = "yes".

V želji po izgradnji bolj točnega modela za objekt *A* ali *B* bi robot lahko skušal združiti (tj. stakniti) podatke posameznih objektov. Na ta način bi dobil več podatkov za učenje, ki bi lahko vodili do izgradnje bolj točnih napovednih modelov. Če bi robot

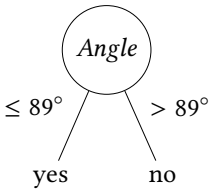


(a) Odločitveno drevo, zgrajeno na podatkih objekta A

(b) Odločitveno drevo, zgrajeno na podatkih objekta B

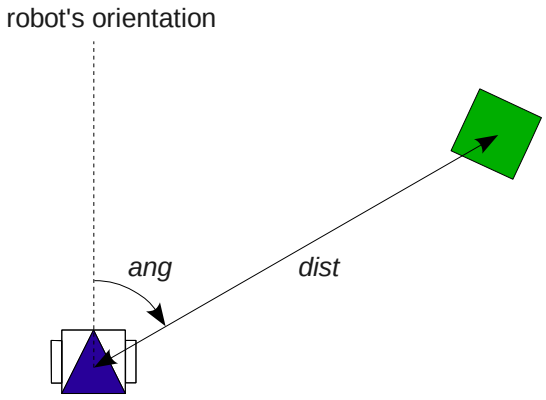


(c) Odločitveno drevo, zgrajeno na združenih podatkih objektov A, B in C



(d) Odločitveno drevo, zgrajeno na združenih podatkih objektov A in B

Slika 4.4: Odločitvena drevesa, ki jih je robot zgradil za različne podatke. Listi označujejo vrednost razreda *MovedCloser* (bodisi "yes" bodisi "no").



Slika 4.5: Robot uporablja stropno kamero za opazovanje razdalje do vsakega izmed objektov (*dist*) ter kota med svojo orientacijo in vsakim izmed objektov (*ang*).

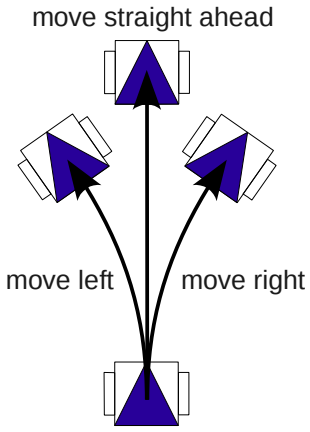
naivno združil vse primere vseh objektov in na njih zgradil odločitveno drevo, bi dobil drevo, ki je prikazano na sliki 4.4(c). To drevo je zelo daleč od želenega drevesa, podanega na sliki 4.3. Pravzaprav je veliko slabši od dreves, zgrajenih na posameznih podatkih objektov A in B. Toda, če bi robot združil le podatke objektov A in B, bi dobil zelo dober napovedni model, prikazan na sliki 4.4(d). To drevo je identično dobremu približnemu modelu na sliki 4.3 z izjemo zanemarljive razlike (1°) v razcepu po atributu *Angle*.

S tem preprostim primerom smo pokazali, kako bi lahko robot z združevanjem podatkov objektov istega tipa zgradil točnejši napovedni model. Še enkrat velja poudariti, da robot ne pozna tipov objektov in zato vprašanje, katere objekte (oz. njihove podatke) združiti, predstavlja težak izziv. Hkrati smo s tem primerom tudi pokazali, da lahko naivno združevanje vseh podatkov vodi do neželenih rezultatov.

4.2 Eksperimentalna domena

Eksperimentalno domeno študije pohitritve učenja robotskega agenta sestavljajo mobilni robot, objekti različnih tipov in stropna kamera. Robot uporablja stropno kamero za opazovanje razdalje do vsakega izmed objektov (označene z *dist*) ter kota med svojo orientacijo in vsakim izmed objektov (označenega z *ang*), kot je prikazano na sliki 4.5. Za računanje razdalj in kotov vse objekte aproksimiramo s točkami — njihovimi težišči.

Na vsakem koraku robot izvede eno od treh akcij (označene z *action*): premik naravnost naprej, premik v levo ali premik v desno, kot je prikazano na sliki 4.6.



Slika 4.6: Robotove akcije: premik naravnost naprej, premik v levo in premik v desno. Na vsakem koraku robot izvede eno od teh akcij.

časovni korak	objekt 1		objekt 2		...	objekt k		...	*action*
	$dist_1$	ang_1	$dist_2$	ang_2	...	$dist_k$	ang_k	...	
1	0.5 m	30°	0.2 m	-10°	...	0.4 m	10°	...	left
2	0.7 m	50°	0.4 m	-30°	...	0.1 m	-10°	...	right
3	0.4 m	80°	0.6 m	-50°	...	0.3 m	20°	...	right
⋮	⋮	⋮	⋮	⋮		⋮	⋮		⋮

Tabela 4.2: Primer sledi podatkov, ki jih robot nabere med učenjem.

Robot se izogiba akcijam, ki vodijo do trkov.

Robot nima vgrajenega koncepta koordinatnega sistema. Vse, kar pozna, so njegove akcije in opazovanja okoliških objektov. Primer sledi podatkov, ki jih robot nabere med učenjem, je podana v tabeli 4.2. Vrednost atributa a ob času t bomo označevali z a^t.

Robot se zaveda, da za vsak objekt opazuje enak nabor atributov ter katere meritve pripadajo določenemu objektu. Na primer, robot ve, da atributa $dist_k$ in $dist_l$ merita isto stvar (robotovo razdaljo do objekta), prvi za objekt k in drugi za objekt l.

4.2.1 Učenje spremembe razdalje do objektov

Robotov cilj je zgraditi model, ki bo napovedoval, za koliko se bo spremenila njegova razdalja do določenega objekta po izvedbi ene od svojih akcij. Namesto napovedovanja točne vrednosti spremembe želimo, da robot zgradi kvalitativni model za napo-

vedovanje kvalitativne spremembe svoje razdalje do objekta. Ta je bodisi *povečanje* (+) bodisi *zmanjšanje* (−) (teoretično obstaja tudi tretja možnost: *brez spremembe* (*o*), vendar se zgodi tako redko, da smo jo združili z možnostjo +).

Povedano bolj natančno, za objekt k bo robot zgradil model M_k, ki bo napovedoval kvalitativno spremembo robotove razdalje do objekta k v naslednjem časovnem koraku ($\text{sgn}(\Delta dist_k^{t+1})$), ki sledi izvedbi akcije ($action^t$) v trenutnem stanju ($dist_k^t$, ang_k^t):

$$\text{sgn}(\Delta dist_k^{t+1}) = M_k(dist_k^t, ang_k^t, action^t),$$

kjer je $\Delta dist_k^{t+1} = dist_k^{t+1} - dist_k^t$.

Za poenostavitev učenja bo robot uporabljal predpostavko, da je M_k le funkcija atributov k-tega objekta (tj. $dist_k$ in ang_k) ter robotove akcije (*action*).

Preprost približni model kvalitativne spremembe robotove razdalje do objekta k lahko strnemo v naslednja tri pravila:

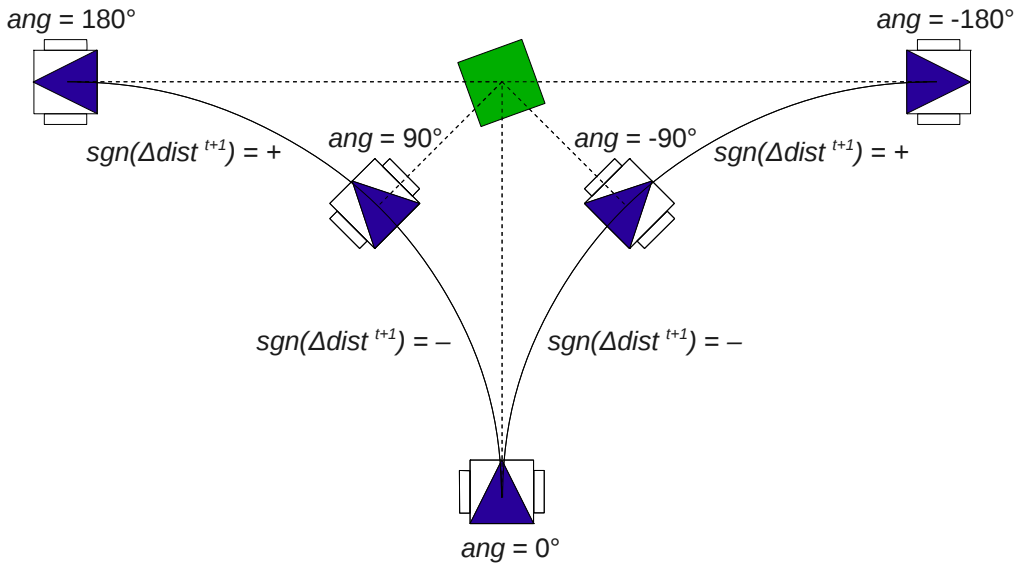
(1) IF $ang_k^t < -90°$ THEN
$\text{sgn}(\Delta dist_k^{t+1}) = +$,

(2) IF $-90° \leq ang_k^t \leq 90°$ THEN
$\text{sgn}(\Delta dist_k^{t+1}) = -$,

(3) IF $ang_k^t > 90°$ THEN
$\text{sgn}(\Delta dist_k^{t+1}) = +$.

Grafična ilustracija teh pravil je podana na sliki 4.7. Točnost podanih pravil na neodvisnih testnih podatkih, skupaj z diskusijo napačno klasificiranih primerov, bomo predstavili v razdelku 4.4.

V naši študiji želimo minimizirati število primerov, ki so potrebni za učenje (tj. koliko primerov robot potrebuje za dosego določene klasifikacijske točnosti), kar je običajen kriterij na področju nadzorovanega učenja. V konkretni eksperimentalni domeni, kjer robot nabira primere, to sovpada s časom, potrebnim za uspešno učenje (tj. koliko akcij mora robot izvesti za dosego določene klasifikacijske točnosti). Ker je minimizacija celotnega časa robotovega izvajanja eksperimentov pogosto razumen in primeren kriterij optimizacije, smo ga izbrali tudi v naši študiji pohitritve učenja robotskega agenta.



Slika 4.7: Grafična ilustracija pravil preprostega približnega modela za kvalitativno spremembo robotove razdalje do objekta. Slika prikazuje primer, kjer robot začne pri kotu $ang = 0°$ in se premika bodisi levo bodisi desno.

4.2.2 Scenariji z naraščajočo številčno in vedenjsko kompleksnostjo

Robotovo učenje bo potekalo v različnih svetovih z naraščajočo *številčno* in *vedenjsko kompleksnostjo*. Številčno kompleksnost smo definirali kot število objektov v robotovem svetu in ustreza prvi lastnosti MTL-problema iz definicije 2.1, tj. število učnih nalog. Podobno smo definirali vedenjsko kompleksnost kot število *vedenjskih tipov* objektov, kjer vedenjski tip objekta predstavlja vse tiste objekte, ki se glede na določeno relacijo, ki se je robot uči (v tem primeru spremembo razdalje do objekta), enako obnašajo. Število vedenjskih tipov objektov ustreza tretji lastnosti MTL-problema iz definicije 2.3, tj. število gruč različnih tipov učnih nalog.

V naši študiji bomo uporabili objekte z dvema vedenjskima tipoma: statični (nepremikajoči se) objekti in premikajoči se objekti. Robot bo zmožen modelirati le spremembo v razdalji do objektov prvega tipa, saj je v tem primeru sprememba odvisna od prejšnjega stanja objekta ($dist^t$, ang^t) in robotove zadnje akcije ($action^t$). Objekti drugega tipa se bodo premikali naključno, zato bo za robota nemogoče zgraditi zelo natančen model za napovedovanje spremembe njegove razdalje do teh objektov. Za predstavnike statičnega vedenjskega tipa smo izbrali škatle, za predstavnike premikajočega vedenjskega tipa pa mobilne robote.

Da bi dobili svetove z naraščajočo številčno in vedenjsko kompleksnostjo, smo

sestavili naslednje scenarije:

- 1 škatla (**1B**),
- 2 škatli (**2B**),
- 4 škatle (**4B**),
- 4 škatle in 4 mobilni roboti (**4B4R**),
- 4 škatle in 8 mobilnih robotov (**4B8R**).

Poleg tega je bil v vsakem scenariju prisoten tudi mobilni robot, ki je opazoval svet in se učil modela za napovedovanje spremembe razdalje do objektov. Na začetku so bili učeči se mobilni robot, škatle in drugi mobilni roboti naključno razporejeni po celotnem pravokotnem polju, na vseh straneh obdanim s steno. Trije primeri scenarijev so prikazani na sliki 4.8.

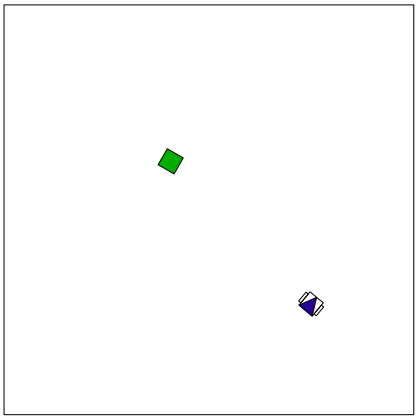
Robotov cilj je v vseh scenarijih enak: naučiti se modela za napovedovanje vrednosti $\mathrm{sgn}(\Delta dist_k^{t+1})$ za vsak objekt v danem scenariju. Z uporabo različnih scenarijev bomo skušali odgovoriti na nekaj zanimivih vprašanj. Je mogoče pohitriti učenje modela za eno škatlo, če imamo na voljo podatke o drugi škatli? Kaj pa v primeru, da imamo na voljo podatke še o treh dodatnih škatlah? Kako prisotnost objektov z drugačnim vedenjskim tipom vpliva na hitrost učenja? Ali bo metoda *ERM* zmogla prepoznati, kateri objekti imajo isti vedenjski tip? Kako se *ERM* obnese, če še nadalje povečamo število objektov z drugačnim vedenjskim tipom?

4.3 Potek učenja

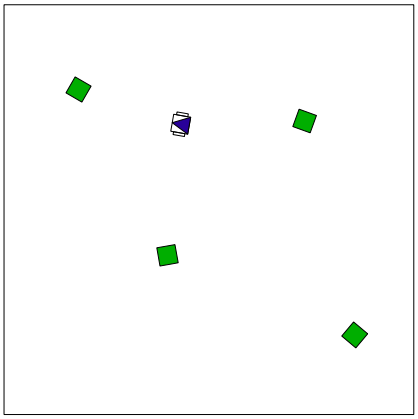
Za učenje robot uporablja metodo *Error-reduction merging* (*ERM*), ki smo jo predstavili v razdelku 2.3. Objekti, ki jih robot opazuje in o njih nabira podatke, v standardni MTL-terminologiji ustrezajo učnim nalogam.

Robot za vsak objekt shranjuje sled podatkov, kot je prikazano v tabeli 4.2. Za pretvorbo sledi podatkov objekta k v običajno tabelo, primerno za nadzorovano učenje, robot za vsak zaporedni par primerov oblike $(dist_k^t, ang_k^t, action^t)$ izračuna vrednost $\mathrm{sgn}(\Delta dist_k^{t+1})$, ki v tej domeni predstavlja razred.

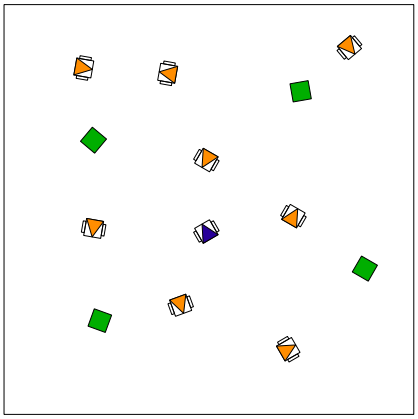
Robotovo učenje poteka v neskončni učni zanki, ki jo podaja procedura LEARNING-LOOP (metoda 4.1).



(a) Scenarij **1B**: učeči se mobilni robot in 1 škatla.



(b) Scenarij **4B**: učeči se mobilni robot in 4 škatle.



(c) Scenarij **4B8R**: učeči se mobilni robot, 4 škatle in 8 drugih mobilnih robotov.

Slika 4.8: Trije primeri scenarijev z naraščajočo številčno in vedenjsko kompleksnostjo (v terminologiji MTL-problemov, z naraščajočim številom učnih nalog ter naraščajočim številom gruč različnih tipov učnih nalog).

Metoda 4.1 Neskončna učna zanka, ki prikazuje potek robotovega učenja.

Require: μ: update interval (i.e. the number of time steps between two successive calls of the *ERM* method)

1 **procedure** LEARNING-LOOP
2 $\mathcal{O} \leftarrow \{\}$ ▷ set of current objects
3 $\mathcal{S} \leftarrow \{\}$ ▷ set of current objects' traces
4 $\mathcal{L} \leftarrow \{\}$ ▷ set of current objects' learning sets
5 $t \leftarrow 0$ ▷ iteration counter
6 **while** *true* **do**
7 $\mathcal{O} \leftarrow$ update $\mathcal{O}$ by creating new objects if the message received by the overhead camera indicates the presence of new objects
8 $\mathcal{S} \leftarrow$ update $\mathcal{S}$ with $\left\{dist_1^t, dist_2^t, ..., dist_{|O|}^t\right\}$ and $\left\{ang_1^t, ang_2^t, ..., ang_{|O|}^t\right\}$ as received by the overhead camera
9 $\mathcal{L} \leftarrow$ update $\mathcal{L}$ with examples computed from the updated $\mathcal{S}$
10 **if** $t \equiv 0 \pmod{\mu}$ **then**
11 $\mathcal{O}', \mathcal{L}' \leftarrow$ make copies of sets $\mathcal{O}, \mathcal{L}$
12 $\mathcal{O}, \mathcal{L}, \mathcal{M} \leftarrow ERM(\mathcal{O}, \mathcal{L})$ ▷ for pseudo-code of *ERM* see method 2.3
13 $\mathcal{O}, \mathcal{L} \leftarrow$ restore original values of $\mathcal{O}, \mathcal{L}$ from $\mathcal{O}', \mathcal{L}'$
14 pick an action from the set of actions (as defined in section 4.2) and execute it
15 $t \leftarrow t + 1$

Za delovanje procedure Learning-loop moramo določiti parameter μ, ki določa interval posodobitve (tj. število časovnih korakov med dvema zaporednima klicema metode *ERM*). Pri izbiri vrednosti tega parametra za praktično rabo se mora uporabnik odločiti, kaj želi. V primeru, da želi čim bolj natančne modele na vsakem časovnem koraku, potem bo nastavil μ na 1, kar pomeni, da bo klic metode *ERM* izveden na vsakem koraku. V primeru relativno hitrega nabiranja novih podatkov pa se lahko zgodi, da bi proces učenja z metodo *ERM* zaostajal za procesom nabiranja podatkov. Takrat je bolj smiselno, da uporabnik nastavi μ na večjo vrednost.

Na začetku procedura Learning-loop inicializira množice $\mathcal{O}$, $\mathcal{S}$ in $\mathcal{L}$ (vrstice 2–4) ter števec iteracij zanke t (vrstica 5). Potem začne ponavljati neskončno **while** zanko (vrstice 6–15). Na vsakem časovnem koraku t robot od stropne kamere dobi nova opazovanja o njegovi razdalji do vsakega objekta, $\left\{dist_1^t, dist_2^t, ..., dist_{|O|}^t\right\}$ ter o kotu med njegovo orientacijo in vsakim objektom, $\left\{ang_1^t, ang_2^t, ..., ang_{|O|}^t\right\}$. Z njimi dopolni trenutne sledi podatkov objektov ter iz njih izračuna nove primere, ki jih doda v tabele podatkov objektov (vrstici 8–9).

Ko je $t \equiv 0 \pmod{\mu}$, procedura naprej naredi rezervni kopiji trenutnih množic $\mathcal{O}$ in $\mathcal{L}$ (vrstica 11) in potem kliče metodo *ERM* (vrstica 12). Ko dobi rezultate metode *ERM*, obnovi množici $\mathcal{O}$ in $\mathcal{L}$ z vrednostma, ki sta ju imeli pred združevanjem (vrstica 13). To bo povzročilo, da se ob naslednjem klicu metode *ERM* združevanje ponovno prične od začetka.

Na koncu **while** zanke procedura izbere eno od možnih robotovih akcij (kot so bile opisane v razdelku 4.2) in jo izvede (vrstica 14) ter poveča števec iteracij za 1 (vrstica 15).

4.4 Opis eksperimentov

Za izvedbo eksperimentov študije pohitritve učenja robotskega agenta smo uporabili robotski simulator *Webots* [59]. Učeči se robot in ostali mobilni roboti so bili implementirani kot preprosti *roboti z diferencialnimi kolesi* (angl. *differential wheeled robots*). Na vsakem časovnem koraku je vsak robot naključno izbral eno od možnih akcij (opisanih v razdelku 4.2) in jo izvedel. Tehnično je bilo izogibanje akcijam, ki vodijo do trkov, implementirano na sledeč način: vsi roboti so imeli spredaj odbijač, ki je služil kot senzor dotika. Kadarkoli je robotov senzor dotika zaznal trk, je robot prenehal z izvajanjem trenutne akcije in izvedel preprost manever za izogibanje trku: premaknil se je vzvratno in se obrnil nekoliko v levo. Potem je nadaljeval z izvaja-

njem akcij. Vsaka akcija, ki jo je robot začel izvajati, je štela kot ena iteracija učne zanke.

Da so ostali mobilni roboti lahko predstavljali objekte z drugačnim vedenjskim tipom glede na relacijo, ki se je je učil robot (tj. napovedovanje spremembe razdalje do objektov), so se morali premikati znatno hitreje od učečega se robota. Če bi se namreč premikali tako hitro kot slednji, bi jih bilo nemogoče ločiti od statičnih škatel, saj robot modelira le kvalitativno spremembo (tj. predznak spremembe) razdalje do objektov.

Sistem za računalniški vid s stropno kamero smo simulirali s posebnim krmilnikom v simulatorju *Webots*, imenovanim *Supervisor*, ki je imel dostop do vseh informacij o učečem se robotu in ostalih objektih. Na vsakem časovnem koraku je *Supervisor* prebral položaje in orientacije učečega se robota in ostalih objektov ter izračunal razdaljo učečega se robota do vsakega izmed objektov (*dist*) ter kot med njegovo orientacijo in vsakim izmed objektov (*ang*). Ti podatki so bili zapakirani v sporočilo, ki je bilo poslano učečemu se robotu. Velja poudariti, da učeči se robot nikoli ni imel podatkov o koordinatah objektov, ampak le podatke o razdalji in kotu do vsakega objekta kot jih je izračunal *Supervisor*.

Pravokotno polje, v katerem se je gibal učeči se robot skupaj z ostalimi objekti, je bilo velikosti 2.0 m $\times$ 2.0 m. Škatle so bile implementirane kot kocke s stranico 0.1 m. Vsi roboti so imeli enako obliko in velikost, ki je bila približno enaka velikosti škatel.

Za oceno zmogljivosti metode *ERM* smo jo v eksperimentih primerjali z metodami *NoMerging*, *MergeAll* in *Oracle*, ki so opisane v uvodnem delu 3. poglavja. Zaradi praktičnih razlogov smo v prvih dveh eksperimentih znotraj vseh štirih MTL-metod uporabili le eno osnovno učno metodo, in sicer *naivni Bayesov klasifikator* (angl. *naïve Bayes classifier*, *NBC*). V tretjem eksperimentu pa smo NBC primerjali še s *C4.5* odločitvenim drevesom in metodo podpornih vektorjev (angl. support vector machine, SVM). Za ta izbor osnovnih učnih metod smo se odločili zato, ker smo želeli primerjati nekaj enostavnih in po naravi delovanja različnih metod nadzorovanega učenja, ki so se zmožne naučiti dobrih modelov za izbrani problem. Pri vseh osnovnih učnih metodah smo uporabili njihove implementacije iz paketa za strojno učenje in podatkovno rudarjenje *Orange* [48].

Metoda NBC je za oceno (pogojnih) verjetnosti zveznih atributov uporabljala *lokalno uteženo regresijo* (angl. *locally weighted regression*, *LOESS*) s parametrom *window_proportion* (delež primerov, ki se uporabijo za lokalno učenje v LOESS), nastavljenim na 0.1. Ostali parametri so imeli privzete vrednosti. Pri metodi *C4.5* so imeli vsi

parametri privzete vrednosti. Parametri metode SVM pa so bili: *svm_type* = *C*-SVC (tip formulacije SVM-problema), $C = 100$ (regularizacijski parameter), *kernel_type* = polynomial (tip jedrne funkcije), *degree* = 3 (stopnja polinomskega jedra), *coef0* = 1 (vrednost konstantnega člena polinomskega jedra). Ostali parametri so imeli privzete vrednosti.

Parametra metode *ERM* sta imela v vseh eksperimentih enako vrednost: η (minimalno število primerov posamezne učne naloge, da kandidatni par učnih nalog zadosti kriteriju za združevanje (1)) je bil nastavljen na 5, k (število ponovitev prečnega preverjanja) pa je bil enak trenutni velikosti podatkov. Slednje pomeni, da smo v teh eksperimentih pravzaprav uporabljali metodo *izpusti enega* (angl. *leave-one-out*). Parameter procedure Learning-loop, μ (število korakov med dvema zaporednima klicema metode *ERM*), je bil nastavljen na 5.

Zgrajene modele smo ocenjevali s klasifikacijsko točnostjo (angl. classification accuracy, CA). Tokrat je bilo razmerje med obema razredoma $\sim 1 : 1$, zato ni bilo potrebe po uporabi ploščine pod ROC-krivuljo (angl. area under ROC curve, AUC). Ker robot ni mogel zgraditi zelo točnih modelov za napovedovanje spremembe svoje razdalje do premikajočih se objektov (tj. mobilnih robotov), nas je zanimal le CA modelov za statične objekte (tj. škatle). Za končni CA smo torej vzeli povprečen CA vseh modelov, ki so napovedovali spremembo robotove razdalje do posamezne škatle. To smo naredili po vsakem klicu metode *ERM* (12. vrstica metode 4.1).

Za izračun CA smo generirali ločene testne podatke, ki so dobro predstavljali celoten prostor atributov. To smo dosegli z gostim vzorčenjem prostora atributov za en objekt in opazovanjem dejanske spremembe v robotovi razdalji do objekta potem, ko izvede eno svojih akcij. Naredili smo mrežni vzorec z 10 začetnimi x koordinatami robota, 10 začetnimi y koordinatami robota, 12 začetnimi robotovimi orientacijami ter 3 akcijami, ki jih je robot izvedel iz začetnega položaja. Objekt, ki ga je robot opazoval, je bil na sredini polja. Na ta način smo dobili 3600 primerov, na katerih smo testirali zgrajene modele in izmerili njihov CA.

Na tem mestu velja omeniti, da preprost približni model kvalitativne spremembe robotove razdalje do objekta (predstavljen v podrazdelku 4.2.1) na testni množici dosega le CA 91.75%. Pri opazovanju napačno klasificiranih primerov smo zasledili nekaj skupnih vzorcev. Vsi napačno klasificirani primeri imajo vrednost *ang* na intervalu $(70°, 100°)$ ali $(-100°, -70°)$. Obstajata dva večja tipa napačno klasificiranih primerov:

(1) Primeri, ki imajo $ang \in (70°, 90°)$ in *action* = left, ter primeri, ki imajo $ang \in (-90°, -70°)$ in *action* = right. V obeh primerih izvedba akcije povzroči povečanje robotove razdalje do objekta (+), ravno obratno od tega, kar napove pravilo (2) (−). Ti primeri predstavljajo 51% napačno klasificiranih primerov.

(2) Primeri, ki imajo $ang \in (90°, 100°)$ in *action* = right, ter primeri, ki imajo $ang \in (-100°, -90°)$ in *action* = left. V obeh primerih izvedba akcije povzroči zmanjšanje robotove razdalje do objekta (−), ravno obratno od tega, kar napovesta pravili (1) in (3) (+). Ti primeri predstavljajo 19% napačno klasificiranih primerov.

Pri ostalih napačno klasificiranih primerih nismo opazili skupnih vzorcev.

Grafi rezultatov v naslednjem razdelku podajajo učne krivulje vseh štirih MTL-metod, ki prikazujejo naraščanje CA v odvisnosti od števila izvedenih akcij robota. Posamezno učno krivuljo smo dobili tako, da smo izbrani eksperiment ponovili 100-krat, vsakič z drugimi naključno izbranimi začetnimi položaji in orientacijami učečega se robota in ostalih objektov, ter na koncu izračunali povprečen CA. Intervali napake predstavljajo 95% konfidenčne intervale za povprečja.

Za izračun konfidenčnih intervalov smo uporabili formulo iz enačbe (3.1). Podrobna razlaga izračuna konfidenčnih intervalov ter povezave med konfidenčnimi intervali in statistično značilnostjo je podana v uvodnem delu 3. poglavja. Tu velja le ponoviti, da ko uporabljamo izraz, da se rezultati dveh metod statistično značilno razlikujejo, s tem želimo povedati, da se 95% konfidenčna intervala za povprečji ne prekrivata, kar po [41] predstavlja statistično značilno razliko med povprečjema pri stopnji značilnosti ~ 0.01.

4.5 Rezultati

V tem razdelku bomo predstavili rezultate študije pohitritve učenja robotskega agenta in skušali odgovoriti na vprašanja, ki smo si jih zastavili na koncu podrazdelka 4.2.2. Rezultati so razdeljeni na tri dele, vsak del se ukvarja z enim vidikom eksperimentalne domene, kot smo jih podali v podrazdelku 4.2.2. Prvi del se ukvarja s povečevanjem številčne kompleksnosti zaradi povečevanja števila škatel v robotovem svetu. V drugem delu je število škatel konstantno, število objektov z drugačnim vedenjskim tipom pa se povečuje, kar pomeni, da se povečujeta tako številčna kot tudi vedenjska kompleksnost sveta. Zadnji del primerja zmogljivost MTL-metod ob uporabi različnih

osnovnih učnih metod v najkompleksnejšem scenariju.

4.5.1 Scenariji z naraščajočim številom škatel

Prvo vprašanje, na katerega smo želeli odgovoriti, je, ali je mogoče pohitriti učenje modela za eno škatlo, če imamo na voljo podatke o drugi škatli. Zato smo primerjali MTL-metode v scenariju z eno škatlo (**1B**) in scenariju z dvema škatlama (**2B**). Rezultati so podani na slikah 4.9(a) in 4.9(b). Na sliki 4.9(a) je le ena učna krivulja, saj v primeru sveta z enim samim objektom vse metode delujejo enako. V scenarijih, kjer imajo vsi objekti isti vedenjski tip (npr. **2B**), MTL-metoda *MergeAll* deluje enako kot metoda *Oracle*, zato smo jo na grafih slike 4.9 izpustili.

Odgovor na naše vprašanje je pritrdilen. Obe metodi, *Oracle* in *ERM*, sta dajali statistično značilno boljše rezultate od metode *NoMerging*, ko je bilo število izvedenih akcij večje od pet. Razlika se je ohranila do konca učne krivulje, vendar se je proti koncu, ko se je število izvedenih akcij približevalo 100, zelo zmanjšala. Tudi na koncu je bila majhna razlika med metodo *NoMerging* in preostalima metodama statistično značilna. To jasno kaže, da uporaba podatkov o drugi škatli omogoča bistveno pohitritev učnega procesa.

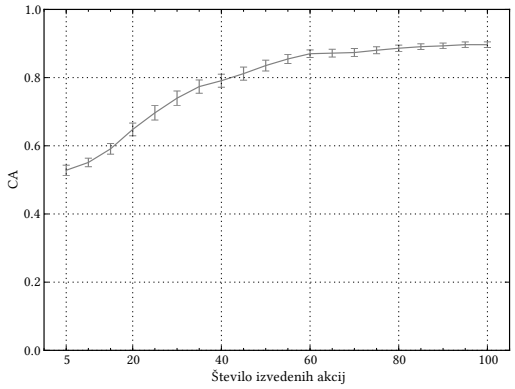
Razlika med vrednostmi CA metod *Oracle* in *ERM* je bila na začetku prav tako statistično značilna, vendar je *ERM* po 20 izvedenih akcijah prvo dohitela. To pomeni, da se metoda *ERM* na začetku, ko je število primerov za vsak objekt zelo majhno, obnaša konzervativno, saj nima dovolj podatkov za združevanje objektov.

Po pričakovanjih učna krivulja metode *NoMerging* v scenariju **2B** ustreza njeni učni krivulji v scenariju **1B**.

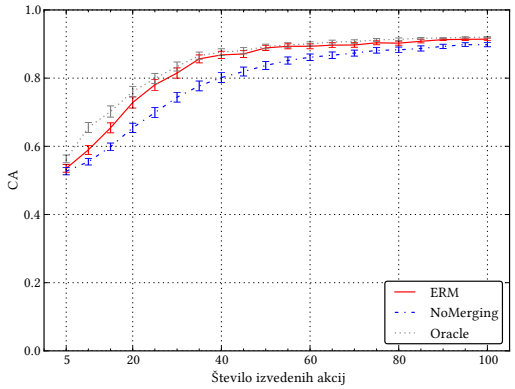
Če v svet dodamo še dve škatli, kot je v scenariju **4B**, razlike med zmogljivostjo metod *Oracle*, *ERM* in *NoMerging* postanejo še bolj očitne. Kot je prikazano na sliki 4.9(c), metoda *ERM* skozi celoten učni proces dosega statistično značilno boljše rezultate od metode *NoMerging*. Razlika med metodama *ERM* in *Oracle* je na začetku precej velika, vendar se s povečevanjem števila izvedenih akcij postopoma zmanjšuje. Kljub temu vseskozi ostaja statistično značilna.

4.5.2 Scenariji s štirimi škatlami in naraščajočim številom drugih mobilnih robotov

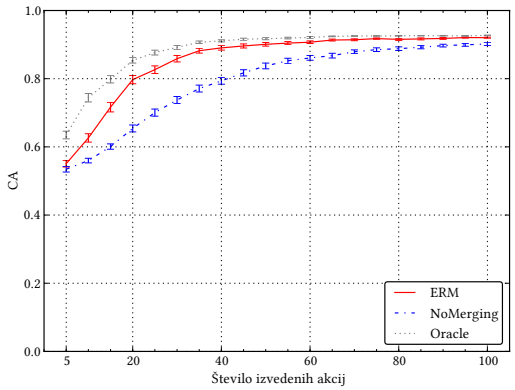
Naslednje vprašanje, ki smo si ga zastavili, je bilo, kako prisotnost objektov z drugačnim vedenjskim tipom vpliva na hitrost učenja. Zato smo primerjali učne krivulje



(a) Scenarij **1B**



(b) Scenarij **2B**



(c) Scenarij **4B**

Slika 4.9: Rezultati poskusov primerjave zmogljivosti MTL-metod v scenarijih s samimi škatlami (tj. scenarijih, kjer so vsi objekti istega vedenjskega tipa). Prikazana so povprečja 100 ponovitev poskusa, kjer je vsaka ponovitev zajemala drugo naključno začetno razporeditev (položaji in orientacije) učečega se robota in ostalih objektov. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja.

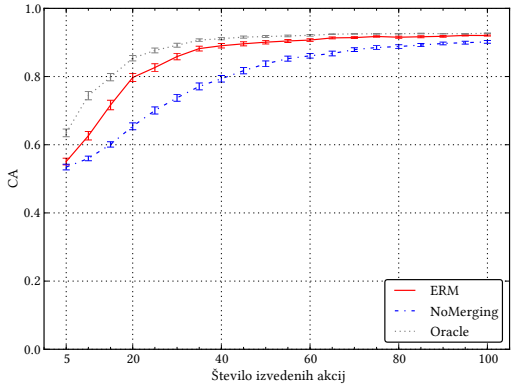
MTL-metod v scenariju s štirimi škatlami (**4B**) in scenariju s štirimi škatlami in štirimi drugimi mobilnimi roboti (**4B4R**). Rezultati so prikazani na slikah 4.10(a) in 4.10(b).

Metoda *ERM* se v scenariju **4B4R** odreže zelo dobro, skoraj enako dobro kot v scenariju **4B**. Njena učna krivulja sicer ne narašča več tako strmo, vendar se razlika med njo in učno krivuljo metode *Oracle* zmanjšuje enako hitro kot prej. To pomeni, da je metoda *ERM* uspešna pri prepoznavanju, kateri objekti imajo statični tip obnašanja in združevanju njihovih podatkov, kar vodi v hitrejšo izgradnjo boljših modelov.

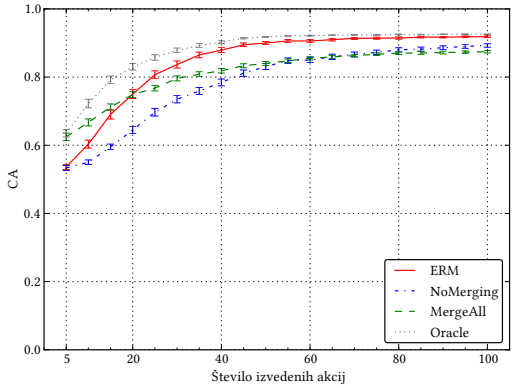
Zanimivo je opazovati učno krivuljo metode *MergeAll* na sliki 4.10(b). Na začetku se je odrezala enako dobro kot metoda *Oracle* in ostala statistično značilno boljša od metode *ERM* do 15. izvedene akcije. Od 25. akcije naprej pa jo je učna krivulja *ERM* spet presegla in tako je ostalo do konca poskusa. Za prvih 45 izvedenih akcij je metoda *MergeAll* dajala statistično značilno boljše rezultate kot metoda *NoMerging*. Hkrati pa je le od 90. akcije naprej metoda *MergeAll* dajala statistično značilno slabše rezultate kot metoda *NoMerging*. Ta rezultat nakazuje, da je lahko v primerih, ko je za objekte na voljo zelo malo podatkov, koristno združiti vse objekte, ne glede na to, ali imajo isti tip obnašanja ali ne.

Še bolj zahteven je bil scenarij s štirimi škatlami in osmimi drugimi mobilnimi roboti (**4B8R**). Rezultati zanj so podani na sliki 4.10(c). Ponovno se je metoda *ERM* odrezala zelo dobro ter dosegala statistično značilno višje vrednosti CA od metode *NoMerging*. Razlika med njeno in učno krivuljo metode *Oracle* je bila le malenkost večja kot pri scenariju **4B4R**. To dokazuje, da je *ERM* še vedno zmožna prepoznati, kateri objekti so škatle (tj. kateri objekti imajo statični vedenjski tip), kljub temu da se je število drugih mobilnih robotov (tj. objektov s statičnim vedenjskim tipom) podvojilo na osem.

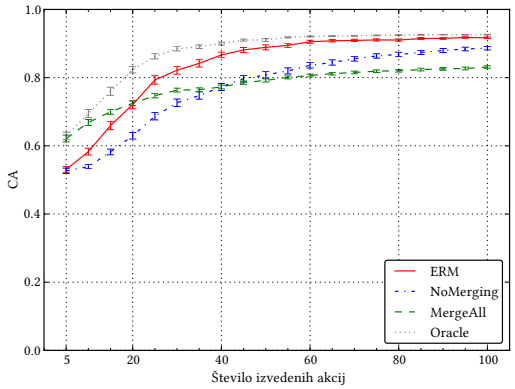
Najopaznejša razlika v zmogljivosti MTL-metod med scenarijema **4B4R** in **4B8R** je padec zmogljivosti metode *MergeAll* v slednjem. To kaže, da združevanje podatkov štirih škatel in osmih drugih mobilnih robotov močno poslabša zmogljivost učenja. Vendar ta vpliv skozi celotno učno krivuljo ni enako velik. Zanimivo je, da je metoda *MergeAll* na začetku (do 20 izvedenih akcij) še vedno dosegala statistično značilno boljše rezultate od metod *ERM* in *NoMerging*. To ponovno nakazuje, da je lahko v primerih, ko je za objekte na voljo zelo malo podatkov, koristno združiti vse objekte, ne glede na to, ali imajo isti tip obnašanja ali ne. Nadalje to nakazuje na potencialno novo MTL-metodo, ki bi na začetku združila vse objekte (oz. učne naloge) in potem postopoma, ko bi bilo za vsak objekt na voljo več podatkov, odstranjevala tiste objekte, ki ne bi več ustrezali združenemu objektu (oz. gruči sorodnih učnih nalog).



(a) Scenarij **4B**



(b) Scenarij **4B4R**



(c) Scenarij **4B8R**

Slika 4.10: Rezultati poskusov primerjave zmogljivosti MTL-metod v scenarijih s štirimi škatlami in naraščajočim številom drugih mobilnih robotov (0, 4 in 8). Prikazana so povprečja 100 ponovitev poskusa, kjer je vsaka ponovitev zajemala drugo naključno začetno razporeditev (položaji in orientacije) učečega se robota in ostalih objektov. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja. Podslika (a) je enaka podsliki 4.9(c) in je tukaj ponovljena zaradi lažje primerjave z ostalimi rezultati.

4.5.3 Primerjava različnih osnovnih učnih metod v scenariju 4B8R

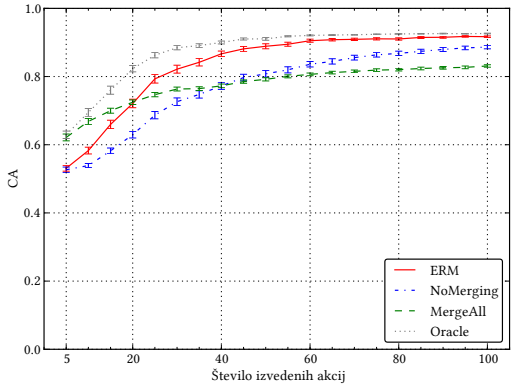
Zadnji niz eksperimentov je primerjal zmogljivost MTL-metod ob uporabi različnih osnovnih učnih metod: NBC, C4.5 in SVM. Primerjali smo jih v najkompleksnejšem scenariju (kot smo jih definirali v podrazdelku 4.2.2) s štirimi škatlami in osmimi ostalimi mobilnimi roboti (**4B8R**). Rezultati so podani na sliki 4.11.

Splošen pregled rezultatov potrjuje, da metoda *ERM* deluje dobro ne glede na uporabljeno osnovno učno metodo. Z metodo C4.5 sicer potrebuje nekoliko več akcij za zmanjšanje razlike do metode *Oracle*, medtem ko z metodo SVM razliko zmanjša približno enako hitro kot z metodo NBC.

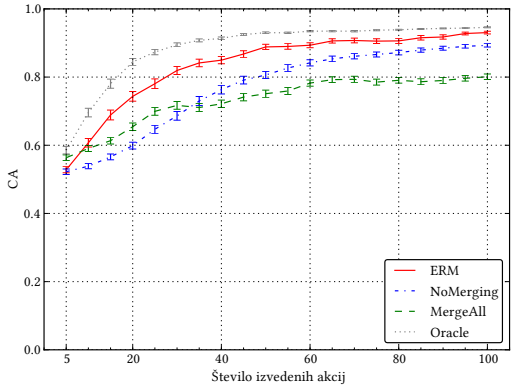
Opazimo lahko znaten padec v zmogljivosti metode *MergeAll* z uporabo osnovne učne metode C4.5, kot je prikazano na sliki 4.11(b). Njena prevlada nad metodo *ERM* je omejena le na prvih pet akcij, od 15. akcije naprej pa so njeni rezultati statistično značilno slabši. To nakazuje, da je metoda C4.5 bolj občutljiva na dodatne nasprotujoče si podatke, ki pripadajo objektom s premikajočim vedenjskim tipom. Ob tem je prav tako zanimivo, da se je metoda *MergeAll* v primerjavi z metodo *NoMerging* odrezala zelo podobno kot pri NBC in je bila za prvih 30 akcij statistično značilno boljša od slednje.

Naslednja zanimiva opazka je zelo slaba zmogljivost metode *NoMerging* pri uporabi osnovne učne metode SVM, kot je prikazano na sliki 4.11(c). V tem primeru se metoda *NoMerging* odreže statistično značilno slabše od vseh ostalih metod. Šele po 90. izvedeni akciji ujame krivuljo metode *MergeAll*, nikoli pa ne pride blizu krivuljama metod *ERM* in *Oracle* (razlika v vrednosti CA je tudi po 100 akcijah večja kot 0.1). Verjetna razlaga za to odstopanje je, da osnovna učna metoda SVM z danimi parametri (opisanimi v razdelku 4.4) potrebuje veliko več primerov, da zgradi modele, ki bi bili tako dobri kot modeli metod NBC ali C4.5. To še dodatno potrjujejo nižji nakloni učnih krivulj ostalih MTL-metod, posebej *ERM* in *Oracle*, v primerjavi s tistimi za NBC ali C4.5.

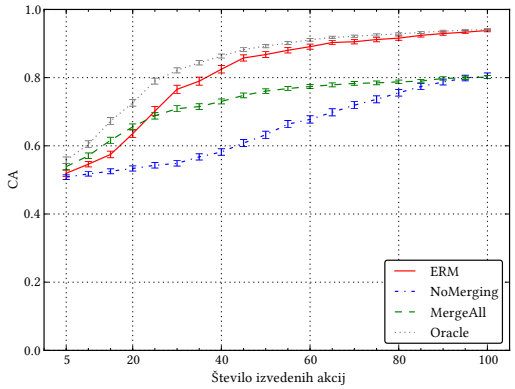
Na koncu velja poudariti, da cilj naših eksperimentov ni bil poiskati optimalne učne krivulje za dane osnovne učne metode. Želeli smo namreč le opazovati, kako se ob spreminjanju osnovne učne metode spreminjajo razmerja med zmogljivostjo primerjanih MTL-metod. Razen manjših odstopanj so ta razmerja med učnimi krivuljami ostala enaka, kar dokazuje, da je delovanje MTL-metod (še posebej *ERM*) neodvisno od uporabljene osnovne učne metode.



(a) NBC



(b) C4.5



(c) SVM

Slika 4.11: Rezultati poskusov primerjave zmogljivosti MTL-metod ob uporabi različnih osnovnih učnih metod (NBC, C4.5, SVM) v scenariju s štirimi škatlami in osmimi drugimi mobilnimi roboti (**4B8R**). Prikazana so povprečja 100 ponovitev poskusa, kjer je vsaka ponovitev zajemala drugo naključno začetno razporeditev (položaji in orientacije) učečega se robota in ostalih objektov. Intervali napake prikazujejo 95% konfidenčne intervale za povprečja. Podslika (a) je enaka podsliki 4.10(c) in je tukaj ponovljena zaradi lažje primerjave z ostalimi rezultati.

4.6 Ugotovitve

V študiji smo raziskovali vprašanje, ali je mogoče pohitriti učenje robotskega agenta v kompleksnejšem okolju. Večja kompleksnost okolja (kot je definirana v razdelku 4.2.2) namreč robotu na vsakem časovnem koraku omogoča izvedbo več hkratnih meritev in s tem zajem večje količine podatkov za učenje. To je lahko koristno, če je v dani domeni določena struktura, ki omogoča združevanje podatkov. V naših eksperimentih so bili to različni objekti, ki so spadali v enega izmed dveh vedenjskih tipov: statični in premikajoči. Po drugi strani pa večja kompleksnost domene z več dodatnimi objekti in vedenjskimi tipi lahko zmede robota in oteži njegovo učenje.

V eksperimentih se je pokazalo, da je metoda *ERM* zmožna odkriti, kateri objekti imajo isti vedenjski tip in združiti njihove podatke, kar vodi do hitrejšega učenja. V MTL-terminologiji to pomeni, da je *ERM* odkrila gruče učnih nalog istega tipa in jih združila, kar je vodilo do izboljšanja napovednih modelov posameznih učnih nalog. Druga ugotovitev je, da *ERM* ob povečevanju števila izvedenih akcij praviloma vedno zelo zmanjša razliko v CA do metode *Oracle*, kjer je pripadnost objektov različnim vedenjskim tipom podana vnaprej. Razlika v manj kompleksnem okolju (scenarij **2B**) ni bila statistično značilna, v bolj kompleksnem okolju (scenariji **4B**, **4B4R**, **4B8R**) pa je bila statistično značilna. Naslednja ugotovitev je, da je metoda *ERM* v vseh pogledih superiorna v primerjavi z metodo *NoMerging*, ki nikoli ne združi podatkov nobenih objektov. Zadnji del eksperimentov pa je še pokazal, da so ugotovljeni trendi veljali ne glede na to, katera osnovna učna metoda je bila uporabljena znotraj *ERM* (oz. ostalih MTL-metod).

Nekoliko nepričakovana ugotovitev študije je, da je metoda *MergeAll* na začetku, ko robot izvede do 15 (v scenariju **4B4R**) oz. 20 akcij (v scenariju **4B8R**), boljša od metode *ERM*, ki smo jo razvili v disertaciji. To nakazuje na potencialno novo MTL-metodo, ki bi na začetku združila vse objekte (oz. učne naloge) in potem postopoma, ko bi bilo za vsak objekt na voljo več podatkov, odstranjevala tiste objekte, ki ne bi več ustrezali združenemu objektu (oz. gruči sorodnih učnih nalog).

5. POGLAVJE

Interpretacija binarizacije vrednosti atributov pri gradnji odločitvenih dreves v smislu delovanja *ERM*

V tem poglavju doktorske disertacije bomo predstavili eksperiment, ki primerja delovanje metode *ERM* s postopkom binarizacije vrednosti atributov pri gradnji odločitvenih dreves.

V literaturi poznamo več postopkov za binarizacijo vrednosti atributov pri gradnji odločitvenih dreves [60–62]. Za zvezne atribute metode za binarizacijo običajno iščejo prag, ki vrednosti atributa razdeli na dva podintervala, tako da maksimizira vrednost izbrane mere za informativnost atributov (npr. *informacijski prispevek* (angl. *information gain*) [63], *Gini-indeks* [64] ali *ReliefF* [65]). Tak pristop med drugim uporabljata [60] in [64].

Pri diskretnih atributih pa nastopi težava zaradi kombinatorične eksplozije pri iskanju najboljše binarizacije. Časovna zahtevnost preverjanja vseh možnih kombinacij delitev vrednosti diskretnega atributa je namreč eksponentna v številu njegovih vrednosti. V [60] zato predlagajo, da se pri binarizaciji diskretnih atributov z malo vrednostmi uporabi izčrpno preiskovanje, v ostalih primerih pa požrešno hevristiko po principu dodajanja naprej (angl. forward selection).

Kot bomo podrobneje razložili v nadaljevanju, smo za izvedbo našega eksperimenta med atribute učnih nalog dodali nov atribut z imenom *id*, ki predstavlja iden-

tifikator učne naloge. Ker so bili ostali atributi dvojiški, se je problem binarizacije vrednosti atributov poenostavil le na binarizacijo vrednosti atributa *id*. Zaradi sorazmerno majhnega števila vrednosti atributa *id* (10), smo lahko v naših eksperimentih za binarizacijo uporabili izčrpno preiskovanje, saj je bilo v danih razmerah dovolj hitro.

V nadaljevanju (razdelek 5.1) bomo najprej podali podroben opis eksperimenta. V njem bomo predstavili sintetične MTL-probleme učenja Boolovih funkcij, ki smo jih generirali za ta poskus, postopek pretvorbe MTL-problema v klasičen STL-problem ter obe metodi za gradnjo odločitvenih dreves, ki smo ju vključili v primerjavo, *Tree* in *ForcedTree*. V razdelku 5.2 bomo predstavili hipoteze, rezultate in ugotovitve kvantitativne primerjave metod *Tree*, *ForcedTree* in *ERM*. Zaključili bomo s predstavitvijo hipotez, rezultatov in ugotovitev opisne primerjave zgrajenih modelov (razdelek 5.3).

5.1 Opis eksperimenta

Tudi pri izvedbi tega eksperimenta smo se, podobno kot v prvem delu eksperimentalne študije obnašanja metode *ERM* (razdelek 3.1), odločili za sintetične MTL-probleme učenja Boolovih funkcij. Podrobnejši opis generiranja Boolovih funkcij je podan v uvodnem delu razdelka 3.1.

Uporabili smo naslednje vrednosti parametrov funkcije *generate_boolean_data_with_complete_test_sets*, ki je del modula *synthetic_data* knjižnice *PyMTL* [44]:

- g (število gruč sorodnih učnih nalog; učne naloge iz iste gruče imajo skupno Boolovo funkcijo): 2,
- tg (število učnih nalog v vsaki gruči): 5,
- a (število spremenljivk (atributov), ki nastopajo v Boolovih funkcijah): 8,
- d (pričakovana dolžina disjunkta): 4,
- nls (število različnih tabel podatkov velikosti n, ki se generirajo za vsako učno nalogo): 10,
- n (število primerov posamezne učne naloge): 100.

Postopek generiranja naključnih Boolovih funkcij smo ponovili 10-krat, vsakič z drugim naključnim semenom. Tako smo dobili deset MTL-problemov s po dvema Boolovima funkcijama, ki ustrezata dvema gručama učnih nalog:

MTL-problem 5.1:

$$f(x_1,\ldots,x_8) = (x_6 \wedge \neg x_5) \vee (x_1 \wedge x_3 \wedge x_6 \wedge \neg x_8) \vee (x_8 \wedge \neg x_1 \wedge \neg x_2 \wedge \neg x_6) \vee (x_3 \wedge x_4 \wedge x_8 \wedge \neg x_1 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7) \quad (5.1a)$$

$$f(x_1,\ldots,x_8) = (x_3 \wedge \neg x_7) \vee (\neg x_2 \wedge \neg x_8) \vee (x_1 \wedge x_5 \wedge x_6 \wedge \neg x_2) \vee (x_2 \wedge x_7 \wedge x_8 \wedge \neg x_3) \quad (5.1b)$$

MTL-problem 5.2:

$$f(x_1,\ldots,x_8) = (x_7 \wedge \neg x_6 \wedge \neg x_8) \vee (\neg x_2 \wedge \neg x_7 \wedge \neg x_8) \vee (x_2 \wedge x_7 \wedge x_8 \wedge \neg x_4) \vee (x_3 \wedge \neg x_2 \wedge \neg x_4 \wedge \neg x_7) \quad (5.2a)$$

$$f(x_1,\ldots,x_8) = (x_1 \wedge x_4 \wedge \neg x_2) \vee (x_6 \wedge x_8 \wedge \neg x_5) \vee (x_1 \wedge x_4 \wedge \neg x_2 \wedge \neg x_6) \vee (x_4 \wedge x_5 \wedge x_7 \wedge \neg x_6) \quad (5.2b)$$

MTL-problem 5.3:

$$f(x_1,\ldots,x_8) = (x_1 \wedge x_5 \wedge x_6 \wedge \neg x_4 \wedge \neg x_7) \vee (x_1 \wedge x_3 \wedge x_6 \wedge x_7 \wedge x_8 \wedge \neg x_4) \vee (x_3 \wedge x_5 \wedge \neg x_1 \wedge \neg x_4 \wedge \neg x_6 \wedge \neg x_8) \vee \neg x_3 \quad (5.3a)$$

$$f(x_1,\ldots,x_8) = (x_2 \wedge x_7 \wedge \neg x_1) \vee (x_1 \wedge x_7 \wedge \neg x_2 \wedge \neg x_3) \vee (x_4 \wedge x_8 \wedge \neg x_2 \wedge \neg x_3) \vee (x_3 \wedge x_4 \wedge x_7 \wedge x_8 \wedge \neg x_5) \quad (5.3b)$$

MTL-problem 5.4:

$$f(x_1,\ldots,x_8) = (\neg x_2 \wedge \neg x_3 \wedge \neg x_7) \vee (x_6 \wedge x_7 \wedge x_8 \wedge \neg x_5) \vee (x_1 \wedge x_3 \wedge x_7 \wedge \neg x_2 \wedge \neg x_4 \wedge \neg x_5) \vee (x_2 \wedge x_5 \wedge x_6 \wedge \neg x_4 \wedge \neg x_7 \wedge \neg x_8) \quad (5.4a)$$

$$f(x_1,\ldots,x_8) = (x_1 \wedge x_6 \wedge \neg x_7) \vee (x_5 \wedge x_6 \wedge \neg x_8) \vee (x_1 \wedge x_6 \wedge x_7 \wedge \neg x_2 \wedge \neg x_3) \vee (x_6 \wedge x_7 \wedge x_8 \wedge \neg x_1 \wedge \neg x_3 \wedge \neg x_4) \quad (5.4b)$$

MTL-problem 5.5:

$$f(x_1,\ldots,x_8) = (x_6 \wedge \neg x_5 \wedge \neg x_8) \vee (x_5 \wedge x_7 \wedge \neg x_1 \wedge \neg x_4 \wedge \neg x_6) \vee (x_5 \wedge x_6 \wedge \neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_8) \vee \neg x_1 \quad (5.5a)$$

$$f(x_1,\ldots,x_8) = (\neg x_4 \wedge \neg x_7) \vee (x_2 \wedge x_6 \wedge \neg x_3 \wedge \neg x_8) \vee (x_1 \wedge x_2 \wedge x_4 \wedge x_6 \wedge \neg x_5 \wedge \neg x_7 \wedge \neg x_8) \vee (x_1 \wedge x_5 \wedge x_6 \wedge x_7 \wedge x_8 \wedge \neg x_3 \wedge \neg x_4) \quad (5.5b)$$

MTL-problem 5.6:

$$f(x_1,\ldots,x_8) = (x_4 \wedge x_5 \wedge x_8) \vee (x_2 \wedge x_3 \wedge \neg x_1 \wedge \neg x_5) \vee (x_1 \wedge x_8 \wedge \neg x_3 \wedge \neg x_6 \wedge \neg x_7) \vee (\neg x_2 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_8) \quad (5.6a)$$

$$f(x_1,\ldots,x_8) = (x_1 \wedge x_2 \wedge x_8) \vee (x_2 \wedge \neg x_4 \wedge \neg x_7) \vee (x_3 \wedge x_7 \wedge x_8 \wedge \neg x_5) \vee (x_3 \wedge x_4 \wedge x_5 \wedge \neg x_2 \wedge \neg x_6) \quad (5.6b)$$

MTL-problem 5.7:

$$f(x_1,\ldots,x_8) = (x_7 \wedge \neg x_4 \wedge \neg x_8) \vee (x_5 \wedge \neg x_2 \wedge \neg x_7 \wedge \neg x_8) \vee (x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_8) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_7) \quad (5.7a)$$

$$f(x_1,\ldots,x_8) = (x_1 \wedge x_2 \wedge x_8) \vee (x_2 \wedge \neg x_4 \wedge \neg x_7) \vee (x_3 \wedge x_7 \wedge x_8 \wedge \neg x_5) \vee (x_3 \wedge x_4 \wedge x_5 \wedge \neg x_2 \wedge \neg x_6) \quad (5.7b)$$

MTL-problem 5.8:

$$f(x_1,\ldots,x_8) = (x_4 \wedge \neg x_2 \wedge \neg x_6) \vee (x_3 \wedge \neg x_2 \wedge \neg x_5 \wedge \neg x_6) \vee (x_2 \wedge x_6 \wedge x_7 \wedge x_8 \wedge \neg x_1) \vee (x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_6 \wedge \neg x_8) \quad (5.8a)$$

$$f(x_1,\ldots,x_8) = (x_1 \wedge \neg x_3 \wedge \neg x_6 \wedge \neg x_8) \vee (x_2 \wedge x_8 \wedge \neg x_3 \wedge \neg x_6) \vee (x_8 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4) \vee (x_1 \wedge x_5 \wedge x_7 \wedge x_8 \wedge \neg x_2) \quad (5.8b)$$

MTL-problem 5.9:

$$f(x_1,\ldots,x_8) = (x_7 \wedge x_8) \vee (x_1 \wedge \neg x_2 \wedge \neg x_5) \vee (x_3 \wedge x_5 \wedge x_7) \vee (x_3 \wedge x_4 \wedge x_5 \wedge \neg x_2 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8) \quad (5.9a)$$

$$f(x_1,\ldots,x_8) = (x_6 \wedge x_7 \wedge x_8) \vee (x_6 \wedge \neg x_3 \wedge \neg x_7) \vee (x_1 \wedge x_3 \wedge x_6 \wedge \neg x_2 \wedge \neg x_7) \vee (x_5 \wedge \neg x_1 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_6) \quad (5.9b)$$

MTL-problem 5.10:

$$f(x_1,\ldots,x_8) = (\neg x_1 \wedge \neg x_3 \wedge \neg x_4) \vee (x_1 \wedge x_3 \wedge x_4 \wedge \neg x_6 \wedge \neg x_8) \vee (x_3 \wedge x_8 \wedge \neg x_1 \wedge \neg x_4 \wedge \neg x_7) \vee (x_1 \wedge x_6 \wedge x_7 \wedge x_8 \wedge \neg x_4 \wedge \neg x_5) \quad (5.10a)$$

$$f(x_1,\ldots,x_8) = (x_1 \wedge \neg x_5 \wedge \neg x_7) \vee (x_3 \wedge x_5 \wedge \neg x_6) \vee (x_3 \wedge \neg x_7 \wedge \neg x_8) \vee (x_1 \wedge x_4 \wedge x_5 \wedge x_6) \quad (5.10b)$$

Izvedli smo niz eksperimentov za različne vrednosti parametra *noise* (delež primerov, pri katerih je razred določen naključno). Uporabili smo vrednosti: 0.0, 0.1, 0.2, 0.3 in 0.5.

Za testiranje smo za vsako učno nalogo generirali množico primerov z vsemi možnimi kombinacijami vrednosti atributov (2^a možnosti). Vsaka testna množica je imela torej 256 primerov.

Za izvedbo binarizacije vrednosti atributov pri gradnji odločitvenih dreves smo podatke učnih nalog za vsak MTL-problem dopolnili in združili. Med atribute učnih nalog smo dodali nov atribut z imenom *id*, ki predstavlja identifikator učne naloge. Vsem učnim primerom, ki pripadajo določeni učni nalogi, smo dodelili skupno unikatno vrednost atributa *id* (npr. "naloga1"). Nato smo primere vseh učnih nalog združili skupaj v eno tabelo podatkov. Prav tako smo primerom, ki smo jih uporabili za testiranje modelov učnih nalog, dodali ustrezno vrednost atributa *id*.

Primerjali smo metodo *ERM* in dve metodi, ki uporabljata binarizacijo vrednosti atributov pri gradnji odločitvenih dreves: *Tree* in *ForcedTree*. Metoda *Tree* uporablja prej opisane združene podatke in zgradi eno (skupno) odločitveno drevo za vse učne naloge. Metoda *ForcedTree* deluje povsem enako kot metoda *Tree*, le da pri gradnji odločitvenega drevesa za prvi razcep vedno uporabi atribut *id*, ne glede na to, ali je po dani meri za določanje informativnosti atributov najbolj informativen ali ne. Velja ponoviti, da metoda *ERM*, za razliko od ostalih dveh metod, uporablja nezdružene podatke brez dodanega atributa *id*.

Za osnovno učno metodo znotraj vseh treh metod smo uporabili odločitveno drevo kot je implementirano v paketu za strojno učenje in podatkovno rudarjenje *Orange* [48]. Mera za določanje informativnosti atributov je bil informacijski prispevek, za

binarizacijo vrednosti atributov smo uporabili metodo izčrpnega preiskovanja. Parameter *min_instances*, ki določa, pri kateri velikosti podmnožice primerov v vozlišču algoritem preneha z nadaljnjo gradnjo drevesa, smo nastavili na 10. Vključili smo tudi naknadno rezanje dreves z odstranjevanjem poddreves, katerih listi vse primere klasificirajo v isti razred. Zgrajene modele smo ocenjevali s ploščino pod ROC-krivuljo (angl. area under ROC curve, AUC).

5.2 Kvantitativna primerjava zmogljivosti

5.2.1 Hipoteze

V prvem delu eksperimenta smo naredili kvantitativno primerjavo metod *Tree*, *Forced-Tree* in *ERM*. Naša hipoteza je bila, da bo metoda *Tree* delovala bolje od metode *Forced-Tree*, saj prva pri izbiri atributa za prvi razcep ni omejena zgolj na različne binarizacije vrednosti atributa *id*, ampak lahko izbere tudi katerega izmed ostalih atributov, če se izkaže za bolj informativnega. Hipoteze o tem, kako se bo odrezala metoda *ERM* v primerjavi z ostalima metodama, si nismo upali postaviti.

5.2.2 Rezultati

Za oceno AUC za celoten MTL-problem smo izračunali povprečje čez vse učne naloge. Kot je bilo razloženo v razdelku 5.1, smo za vsako učno nalogo posameznega MTL-problema generirali deset različnih tabel podatkov, s čimer smo simulirali 10-kratno ponovitev eksperimenta. Na koncu smo izračunali povprečja vseh tako dobljenih povprečnih vrednosti AUC ter njihove 95% konfidenčne intervale.

Za izračun konfidenčnih intervalov smo uporabili formulo iz enačbe (3.1). Podrobna razlaga izračuna konfidenčnih intervalov ter povezave med konfidenčnimi intervali in statistično značilnostjo je podana v uvodnem delu 3. poglavja. Tu velja le ponoviti, da ko uporabljamo izraz, da se rezultati dveh metod statistično značilno razlikujejo, s tem želimo povedati, da se 95% konfidenčna intervala za povprečji ne prekrivata, kar po [41] predstavlja statistično značilno razliko med povprečjema pri stopnji značilnosti $\alpha = 0.01$.

Enak postopek smo ponovili za vsako od vrednosti parametra *noise* (0.0, 0.1, 0.2, 0.3 in 0.5), ki določa delež primerov, pri katerih je razred določen naključno. Rezultati so podani v tabelah 5.1–5.5. Odebeljene so tiste najvišje vrednosti AUC, pri katerih se 95% konfidenčni interval ne prekriva s 95% konfidenčnima intervaloma preosta-

MTL-problem	*Tree*	*ForcedTree*	*ERM*
5.1	0.94 ± 0.02	0.94 ± 0.02	**0.99** ± 0.01
5.2	0.92 ± 0.03	0.92 ± 0.03	**0.99** ± 0.00
5.3	0.96 ± 0.01	0.96 ± 0.01	**0.99** ± 0.01
5.4	0.92 ± 0.02	0.92 ± 0.02	**0.99** ± 0.01
5.5	0.98 ± 0.01	0.99 ± 0.00	**1.00** ± 0.00
5.6	0.87 ± 0.01	0.87 ± 0.01	**0.98** ± 0.00
5.7	0.93 ± 0.01	0.93 ± 0.01	**0.99** ± 0.00
5.8	0.93 ± 0.01	0.92 ± 0.01	**1.00** ± 0.00
5.9	0.96 ± 0.02	0.97 ± 0.01	**1.00** ± 0.00
5.10	0.93 ± 0.01	0.93 ± 0.01	**1.00** ± 0.00

Tabela 5.1: Povprečne vrednosti AUC različnih učnih metod za deset naključno generiranih MTL-problemov, opisanih v razdelku 5.1. Delež učnih primerov z naključno določenim razredom (vrednost parametra *noise*) je 0.0. Intervali napake podajajo 95% konfidenčne intervale za povprečja.

lih metod in so torej statistično značilno višje od preostalih dveh metod pri stopnji značilnosti $\alpha = 0.01$.

Ko v podatkih ni šuma (*noise* = 0.0), se pri vseh desetih MTL-problemih najbolje odreže metoda *ERM* (tabela 5.1). Njen AUC je pri vsakem MTL-problemu statistično značilno večji od vrednosti AUC preostalih dveh metod. Metoda *ForcedTree* je pri dveh MTL-problemih (5.5 in 5.9) za odtenek boljša od metode *Tree*, slednja pa je v pri enem MTL-problemu (5.8) boljša od prve. Razlika med njima nikoli ni statistično značilna. Za ta poskus vse tri metode dosegajo zelo visok AUC: metodi *Tree* in *ForcedTree* ≥ 0.87, metoda *ERM* ≥ 0.98. To pomeni, da takrat, ko v podatkih ni šuma, vse metode zgradijo dobra odločitvena drevesa. Nekoliko bolj povprečne vrednosti AUC nihajo pri metodah *Tree* in *ForcedTree*, pri prvi med 0.87 in 0.98, pri drugi med 0.87 in 0.99, medtem ko pri *ERM* nihajo med 0.98 in 1.00. Podobno so tudi 95% konfidenčni intervali pri metodah *Tree* in *ForcedTree* večji kot pri metodi *ERM*.

Pri naslednjem poskusu je bil delež primerov z naključno določenim razredom 0.1 (tabela 5.2). Enako kot pri poskusu z nezašumljenimi podatki se je tudi tokrat pri vseh desetih MTL-problemih najbolje odrezala metoda *ERM*. Tokrat je bil njen AUC statistično značilno večji od vrednosti preostalih dveh metod pri vseh MTL-problemih, razen pri MTL-problemu 5.5. Ponovno je bila metoda *ForcedTree* pri dveh MTL-problemih (5.3 in 5.9) nekoliko boljša od metode *Tree*, vendar razlika nikoli ni bila statistično značilna. Tudi pri tem poskusu vse tri metode dosegajo zelo visok

MTL-problem	*Tree*	*ForcedTree*	*ERM*
5.1	0.93 ± 0.01	0.93 ± 0.01	**0.99** ± 0.01
5.2	0.89 ± 0.03	0.89 ± 0.03	**0.99** ± 0.01
5.3	0.94 ± 0.01	0.95 ± 0.01	**0.99** ± 0.01
5.4	0.88 ± 0.01	0.88 ± 0.01	**0.97** ± 0.02
5.5	0.97 ± 0.01	0.97 ± 0.01	0.98 ± 0.01
5.6	0.86 ± 0.02	0.86 ± 0.02	**0.97** ± 0.01
5.7	0.91 ± 0.01	0.91 ± 0.01	**0.99** ± 0.01
5.8	0.90 ± 0.02	0.90 ± 0.01	**0.99** ± 0.01
5.9	0.93 ± 0.02	0.94 ± 0.01	**0.99** ± 0.00
5.10	0.90 ± 0.01	0.90 ± 0.01	**0.99** ± 0.01

Tabela 5.2: Povprečne vrednosti AUC različnih učnih metod za deset naključno generiranih MTL-problemov, opisanih v razdelku 5.1. Delež učnih primerov z naključno določenim razredom (vrednost parametra *noise*) je 0.1. Intervali napake podajajo 95% konfidenčne intervale za povprečja.

MTL-problem	*Tree*	*ForcedTree*	*ERM*
5.1	0.91 ± 0.02	0.91 ± 0.02	**0.98** ± 0.01
5.2	0.89 ± 0.02	0.89 ± 0.02	**0.98** ± 0.01
5.3	0.93 ± 0.01	0.93 ± 0.02	**0.97** ± 0.01
5.4	0.87 ± 0.02	0.87 ± 0.02	**0.96** ± 0.01
5.5	0.95 ± 0.01	0.95 ± 0.01	**0.98** ± 0.01
5.6	0.83 ± 0.02	0.83 ± 0.01	**0.95** ± 0.02
5.7	0.87 ± 0.03	0.87 ± 0.03	**0.97** ± 0.01
5.8	0.88 ± 0.02	0.88 ± 0.02	**0.97** ± 0.02
5.9	0.93 ± 0.01	0.93 ± 0.02	**0.97** ± 0.01
5.10	0.88 ± 0.03	0.88 ± 0.03	**0.98** ± 0.01

Tabela 5.3: Povprečne vrednosti AUC različnih učnih metod za deset naključno generiranih MTL-problemov, opisanih v razdelku 5.1. Delež učnih primerov z naključno določenim razredom (vrednost parametra *noise*) je 0.2. Intervali napake podajajo 95% konfidenčne intervale za povprečja.

MTL-problem	*Tree*	*ForcedTree*	*ERM*
5.1	0.85 ± 0.02	0.87 ± 0.02	**0.96** ± 0.01
5.2	0.85 ± 0.02	0.85 ± 0.02	**0.96** ± 0.02
5.3	0.91 ± 0.02	0.91 ± 0.02	0.95 ± 0.02
5.4	0.84 ± 0.03	0.84 ± 0.03	**0.91** ± 0.03
5.5	0.93 ± 0.01	0.94 ± 0.01	0.96 ± 0.02
5.6	0.83 ± 0.02	0.83 ± 0.03	**0.91** ± 0.03
5.7	0.87 ± 0.03	0.87 ± 0.03	**0.96** ± 0.02
5.8	0.82 ± 0.03	0.82 ± 0.03	**0.94** ± 0.03
5.9	0.89 ± 0.03	0.89 ± 0.02	**0.96** ± 0.01
5.10	0.86 ± 0.02	0.86 ± 0.02	**0.96** ± 0.02

Tabela 5.4: Povprečne vrednosti AUC različnih učnih metod za deset naključno generiranih MTL-problemov, opisanih v razdelku 5.1. Delež učnih primerov z naključno določenim razredom (vrednost parametra *noise*) je 0.3. Intervali napake podajajo 95% konfidenčne intervale za povprečja.

MTL-problem	*Tree*	*ForcedTree*	*ERM*
5.1	0.77 ± 0.02	0.78 ± 0.02	**0.88** ± 0.02
5.2	0.77 ± 0.04	0.77 ± 0.04	0.84 ± 0.03
5.3	0.81 ± 0.03	0.80 ± 0.03	0.83 ± 0.02
5.4	0.75 ± 0.02	0.75 ± 0.02	**0.81** ± 0.03
5.5	0.87 ± 0.02	0.87 ± 0.02	0.88 ± 0.02
5.6	0.71 ± 0.02	0.71 ± 0.02	**0.79** ± 0.04
5.7	0.74 ± 0.03	0.74 ± 0.03	**0.83** ± 0.04
5.8	0.72 ± 0.02	0.72 ± 0.02	0.76 ± 0.05
5.9	0.80 ± 0.02	0.80 ± 0.02	0.83 ± 0.02
5.10	0.74 ± 0.03	0.74 ± 0.03	**0.83** ± 0.03

Tabela 5.5: Povprečne vrednosti AUC različnih učnih metod za deset naključno generiranih MTL-problemov, opisanih v razdelku 5.1. Delež učnih primerov z naključno določenim razredom (vrednost parametra *noise*) je 0.5. Intervali napake podajajo 95% konfidenčne intervale za povprečja.

AUC: metodi *Tree* in *ForcedTree* ≥ 0.86, metoda *ERM* ≥ 0.97. To kaže na to, da vse tri učne metode zgradijo dobre modele, kljub temu da je v podatkih nekaj šuma. Ponovno povprečne vrednosti AUC nekoliko bolj nihajo pri metodah *Tree* in *ForcedTree*, med 0.86 in 0.97, medtem ko so pri metodi AUC med 0.97 in 0.99. Podobno je tudi s 95% konfidenčnimi intervali. Opazimo lahko, da so se tokrat, v primerjavi s prejšnjim poskusom za *noise* = 0.0, nekoliko povečali 95% konfidenčni intervali metode *ERM*.

Rezultati poskusa, kjer je bil pri 20% primerov razred določen naključno, so prikazani v tabeli 5.3. Ponovno je bila pri vseh MTL-problemih najboljša metoda *ERM*. Tokrat je bil njen AUC ponovno statistično značilno večji od vrednosti AUC preostalih dveh metod pri vseh desetih MTL-problemih. Pri tem poskusu med metodama *Tree* in *ForcedTree* ni bilo nobenih razlik v povprečnih vrednostih AUC, le njuni 95% konfidenčni intervali so pri nekaterih MTL-problemih nekoliko različni. Tokrat je najnižja povprečna vrednost AUC metod *Tree* in *ForcedTree* padla na 0.83, pri metodi *ERM* pa ostajajo vse vrednosti AUC ≥ 0.95. To kaže na to, da z metodo *ERM* zgradimo dobre modele kljub večji stopnji šuma v podatkih. Ponovno povprečne vrednosti AUC nekoliko bolj nihajo pri metodah *Tree* in *ForcedTree*, med 0.83 in 0.95, medtem ko so pri metodi AUC med 0.95 in 0.98. Prav tako 95% konfidenčni intervali pri metodah *Tree* in *ForcedTree* nekoliko bolj nihajo v primerjavi s 95% konfidenčnimi intervali metode *ERM*.

Pri četrtem poskusu je bil delež primerov z naključno določenim razredom 0.3 (tabela 5.4). Tudi tokrat se je pri vseh MTL-problemih najbolje odrezala metoda *ERM*. Od tega le v dveh primerih *ERM* ni bila statistično značilno boljša od preostalih dveh metod. Pri tem poskusu je bila metoda *ForcedTree* pri dveh MTL-problemih (5.1 in 5.5) nekoliko boljša od metode *Tree*, vendar razlika med njima nikoli ni bila značilna. Pri metodi *ERM* so se povprečne vrednosti AUC znižale na interval med 0.91 in 0.96, medtem ko pri metodah *Tree* in *ForcedTree* podobno velikega padca ni, saj so njune povprečne vrednosti AUC med 0.82 in 0.94. Nihanje povprečnih vrednosti AUC pri metodi *ERM* se je v primerjavi s prejšnjimi poskusi z nižjo stopnjo šuma v podatkih nekoliko povečalo (vrednosti so med 0.91 in 0.96), medtem ko za metodi *Tree* in *ForcedTree* ostaja približno enako (vrednosti so med 0.82 in 0.94). Nadaljuje se trend povečevanja in nihanja 95% konfidenčnih intervalov metode *ERM*, ki je v tem poskusu že enak kot pri preostalih dveh metodah.

Pri zadnjem iz te serije poskusov je imelo 50% primerov razred določen naključno (tabela 5.5). Tako kot pri vseh preostalih poskusih se je tudi tokrat pri vseh MTL-problemih najbolje odrezala metoda *ERM*. Bistveno pa se je zmanjšalo število MTL-

problemov, kjer so vrednosti AUC metode *ERM* statistično značilno večje od preostalih dveh metod. Tokrat se je to zgodilo le pri polovici od desetih MTL-problemov. Metoda *ForcedTree* je bila pri enem MTL-problemu (5.1) nekoliko boljša od metode *Tree*, medtem ko je bila slednja pri prav tako enem MTL-problemu (5.3) nekoliko boljša od prve. Razlika med njima nikoli ni statistično značilna. Občutno so se znižale tudi povprečne vrednosti AUC vseh treh metod. Pri metodah *Tree* in *ForcedTree* na interval med 0.71 in 0.87, pri metodi *ERM* pa na interval med 0.76 in 0.88. Glede na tako veliko stopnjo šuma v podatkih so zgrajeni modeli relativno dobri. Prav tako se je občutno povečalo tudi nihanje povprečnih vrednosti AUC vseh treh metod. Zanimivo je, da so 95% konfidenčni intervali pri tem poskusu največji pri metodi *ERM*.

5.2.3 Ugotovitve

Hipoteze, da bo metoda *Tree* delovala bolje od metode *ForcedTree*, nismo potrdili. Še več, slednja je celo večkrat dosegala za odtenek boljše rezultate od prve, vendar razlika nikoli ni bila statistično značilna. Dobljeni rezultati nakazujejo, da je atribut *id* v večini primerov najbolj informativen in ga tako tudi metoda *Tree* izbere za prvi razcep. Glede na to, da so rezultati metode *ForcedTree* celo malenkost boljši od metode *Tree*, bi lahko rekli, da je atribut *id* najboljša izbira za prvi razcep, čeprav vrednosti dane mere informativnosti atributov kažejo drugače. Dobljena drevesa in njihove prve razcepe bomo podrobneje analizirali v razdelku 5.3.

Nekoliko presenetljivo se je metoda *ERM* v tem eksperimentu izkazala za superiorno. Z njo smo vedno dosegali najboljše rezultate. Pri nižji stopnji šuma je metoda *ERM* dala praviloma vedno tudi statistično značilno boljše rezultate od preostalih dveh metod. Tudi absolutna razlika med njenim AUC in AUC vrednostjo ostalih dveh metod je bila v večini primerov opazno velika (med 0.05 in 0.10). Pri manjših stopnjah šuma ($noise \in \{0.0, 0.1, 0.2\}$) so bili 95% konfidenčni intervali za povprečja metode *ERM* prav tako vedno manjši od 95% konfidenčnih intervalov preostalih dveh metod.

Velja omeniti, da je *ERM* zaradi večje časovne zahtevnosti, podrobno opisane v razdelku 2.3.6, za izgradnjo modelov potrebovala bistveno več časa kot metodi *Tree* in *ForcedTree*. Na zmogljivem prenosnem računalniku z *Intel Core i7* 2.80 GHz procesorjem je metoda *ERM* v večini primerov za eno ponovitev poskusa porabila med 4 in 7 sekundami. Za enake poskuse sta metodi *Tree* in *ForcedTree* v večini primerov porabili le med 0.1 in 0.3 sekunde. To pomeni, da sta bili za cel velikosti red hitrejši od metode *ERM*. Pri uporabi metod v praksi je torej potrebno upoštevati tudi vidik

razmerja med hitrostjo metode in njeno zmogljivostjo.

5.3 Opisna primerjava zgrajenih modelov

5.3.1 Hipoteze

V drugem delu eksperimenta smo zgrajena odločitvena drevesa še podrobneje analizirali in naredili opisno primerjavo med modeli dreves, ki smo jih dobili z metodama *Tree* in *ForcedTree*, ter dendrogrami, ki so rezultat združevanja sorodnih učnih nalog v gruče z metodo *ERM*.

Zanimalo nas je, pri kolikšnem deležu dreves, zgrajenih z metodo *Tree*, je bil za prvi razcep izbran atribut *id* in so bila torej enaka tistim, zgrajenim z metodo *ForcedTree*. Glede na rezultate primerjave zmogljivosti vseh treh metod iz prejšnjega razdelka (5.2), je bila naša hipoteza, da v večini primerov metoda *Tree* za prvi razcep izbere atribut *id*.

Poleg tega smo želeli primerjati tudi strukturo drevesa, kjer se veje cepijo po atributu *id* v primerjavi z dendrogramom, ki prikazuje potek združevanja učnih nalog metode *ERM*. Naša hipoteza je bila, da bomo zasledili podobnosti v tem, kako se cepijo veje dreves glede na vrednosti atributa *id* ter zgodovino združevanja učnih nalog, prikazano v dendrogramu.

5.3.2 Rezultati

Deleži dreves metode *Tree*, kjer je bil za prvi razcep izbran atribut *id*

Kot je razloženo v razdelku 5.1, se je za vsak MTL-problem in za vsako vrednost parametra *noise* generiralo *nls* (v tem eksperimentu 10) različnih tabel podatkov. Na vsaki izmed njih smo zgradili model z metodami *Tree*, *ForcedTree* in *ERM*. Deleži dreves, zgrajenih z metodo *Tree*, kjer je bil za prvi razcep izbran atribut *id*, so podani v tabeli 5.6.

Najprej lahko opazimo, da so deleži v večini primerov zelo visoki. Za MTL-problema 5.2 in 5.10 je ta delež, ne glede na vrednost parametra *noise*, kar 1.0. Najnižji delež dreves s prvim razcepom po atributu *id*, sta imela MTL-problema 5.3 in 5.5. Pri prvem so bili deleži med 0.1 in 0.8, pri drugem med 0.1 in 1.0. Nižje deleže dreves za vrednosti parametra *noise* ≤ 0.1 sta imela še MTL-problema 5.4 in 5.8, med 0.4 in 0.8. Pri MTL-problemu 5.9 pa so bili deleži, ne glede na vrednost parametra *noise*, vedno

	noise				
MTL-problem	0.0	0.1	0.2	0.3	0.5
5.1	1.0	0.9	1.0	0.6	0.9
5.2	1.0	1.0	1.0	1.0	1.0
5.3	0.1	0.2	0.2	0.4	0.8
5.4	0.4	0.8	0.9	1.0	1.0
5.5	0.3	0.5	0.1	0.6	1.0
5.6	1.0	0.8	0.9	0.8	1.0
5.7	1.0	0.9	1.0	1.0	1.0
5.8	0.4	0.8	0.9	1.0	1.0
5.9	0.7	0.8	0.8	0.7	0.9
5.10	1.0	1.0	1.0	1.0	1.0

Tabela 5.6: Deleži dreves, zgrajenih z metodo *Tree*, kjer je bil za prvi razcep izbran atribut *id* (in so torej enaka drevesom metode *ForcedTree*).

nekoliko nižji, med 0.7 in 0.9.

Nizki deleži zaradi t. i. singularnih disjunktov

Podrobnejša analiza razkrije, da je pri obeh MTL-problemih z najnižjim deležem dreves s prvim razcepom po atributu *id*, v eni izmed obeh funkcij prisoten disjunkt z eno samo Boolovo spremenljivko. Pri MTL-problemu 5.3 v funkciji (5.3a) kot singularen disjunkt nastopa atribut x_3, pri MTL-problemu 5.5 pa v funkciji (5.5a) kot singularen disjunkt nastopa atribut x_1. Ročni pregled dreves za vse učne množice obeh MTL-problemov je razkril, da je bil za prvi razcep, v primerih ko ni bil izbran atribut *id*, vedno izbran atribut x_3 oz. x_1.

Za razlago te ugotovitve moramo podrobneje razumeti naravo eksperimenta. MTL-problemi tega eksperimenta so sestavljeni iz dveh Boolovih funkcij v disjunktni normalni obliki, vsaka od njiju ustreza eni gruči petih učnih nalog. Ko je vrednost tega singularnega disjunkta *true*, je tudi vrednost celotne Boolove funkcije *true*. To Boolovo funkcijo si deli polovica učnih nalog MTL-problema, ki k združenim učnim podatkom prispevajo polovico primerov. To pa pomeni, da je tak atribut zelo informativen in ga zato algoritem za gradnjo dreves izbere kot prvega.

Nizki deleži zaradi zelo pogostih pojavitev določenega atributa v disjunktih

Analizirali smo tudi, zakaj se pri MTL-problemih 5.4 in 5.8 pri majhnih vrednostih parametra noise (≤ 0.1) pojavljajo nizki deleži izbire atributa *id* za prvi razcep dreves. Pri obeh MTL-problemih se določen atribut zelo pogosto pojavlja v disjunktih obeh funkcij. Pri MTL-problemu 5.4 se atribut x_6 pojavlja v vseh disjunktih funkcije (5.4b) in v dveh disjunktih funkcije (5.4a), medtem ko se pri MTL-problemu 5.8 prav tako atribut x_6 v negacijski obliki pojavlja v treh disjunktih funkcije (5.8a) in v dveh disjunktih funkcije (5.8b). Ročni pregled dreves za vse učne množice obeh MTL-problemov je razkril, da je bil za prvi razcep, v primerih ko ni bil izbran atribut *id*, vedno izbran atribut x_6.

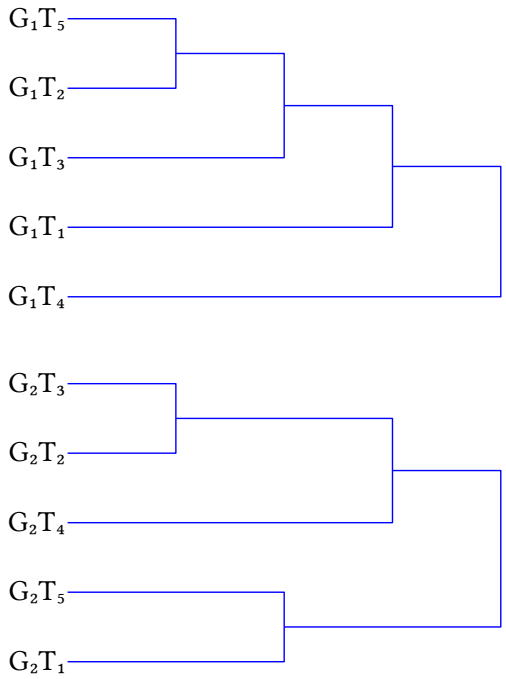
Tudi tukaj velja podobna razlaga kot pri primeru singularnih disjunktov. V primeru, ko se atribut x_6 pojavi v vseh disjunktih funkcije (pri funkciji (5.4b)) in ima vrednost *false*, je tudi vrednost celotne funkcije enaka *false*. Ker si to funkcijo deli polovica učnih nalog MTL-problema oz. polovica učnih primerov, to pomeni, da je ta atribut zelo informativen in ga zato algoritem za gradnjo dreves izbere kot prvega. Podoben efekt se zgodi, ko se atribut x_6 pojavi v večini (treh izmed štirih) disjunktov funkcije (5.8a) in hkrati pojavi tudi v polovici disjunktov druge Boolove funkcije tega MTL-problema. V primeru, da ima x_6 vrednost *true* (ker v disjuktih nastopa kot negacija), imajo vsi disjunkti, v katerih nastopa, vrednost *false* in se obe Boolovi funkciji znatno poenostavita. To pa ponovno pomeni, da je ta atribut zelo informativen in ga zato algoritem za gradnjo dreves izbere kot prvega.

Primerjava strukture dreves, kjer se veje cepijo po atributu *id*, z dendrogrami

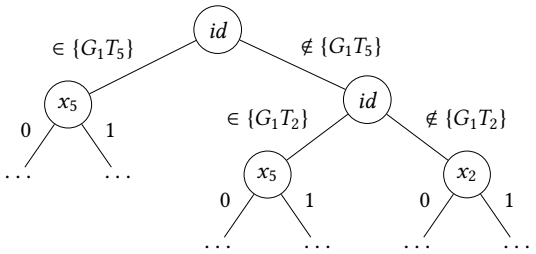
Drevesa, ki smo jih dobili z metodama *Tree* in *ForcedTree*, so bila razmeroma velika — običajno so imela več kot 100 listov. Zato smo se omejili le na izrisovanje poddreves, kjer se veje še cepijo po atributu *id*. To je omogočilo lažjo primerjavo z dendrogrami metode *ERM*.

V prvem delu primerjave strukture dreves, kjer se veje cepijo po atributu *id*, z dendrogrami, ki prikazujejo potek združevanja učnih nalog metode *ERM*, smo se omejili na poskuse, kjer je bil parameter $noise = 0.0$. Primerjali smo drevesa, dobljena z metodo *ForcedTree* (ki so vedno imela prvi razcep po atributu *id*) ter dendrograme metode *ERM*.

Prva ugotovitev primerjave je, da med strukturo dreves in dendrogrami ni nobene podobnosti. Večinoma sta si cepljenje vej dreves glede na vrednosti atributa *id* in



(a) Dendrogram metode *ERM*



(b) Drevo metode *ForcedTree*

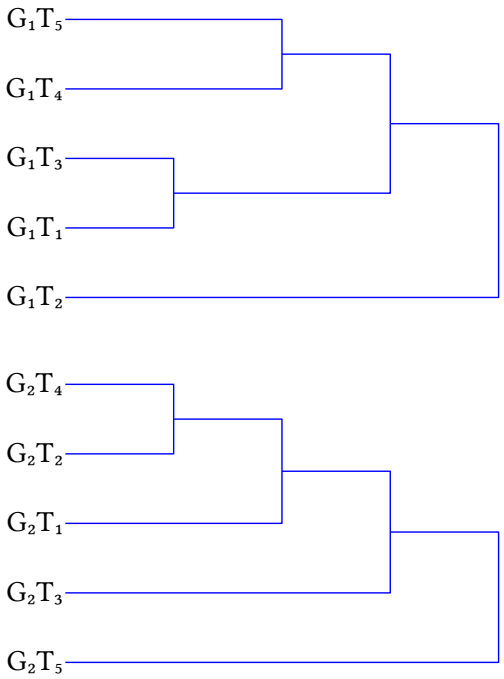
Slika 5.1: Primer popolne različnosti med strukturo dendrograma in drevesa pri 5. ponovitvi poskusa za MTL-problem 5.1. Pri drevesu se najprej odcepi veja s primeri učne naloge G_1T_5 in nato še veja s primeri učne naloge G_1T_2, medtem ko metoda *ERM* ti dve učni nalogi združi kot prvi — najbolj podobni.

zgodovina združevanja učnih nalog, prikazana v dendrogramu, prav nasprotna. Tisto učno nalogo, ki jo drevo najprej odcepi v svojo vejo kot najbolj različno v primerjavi z ostalimi, metoda *ERM* združi med prvimi. Primer tega pojava je podan na sliki 5.1.

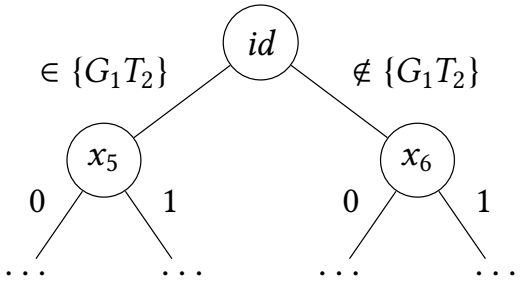
Zelo redki so primeri, ko sta tako metoda za binarizacijo vrednosti atributov pri izgradnji dreves, kot tudi *ERM*, neko učno nalogo spoznala kot najbolj različno od ostalih in se zato le-ta pojavi pri vrhu drevesa oz. dendrograma. Tak primer je prikazan na sliki 5.2.

Optimalna binarizacija običajno neuravnotežena

Nekoliko nas je presenetilo, da je običajno kot rezultat optimalne binarizacije vrednosti atributa *id* delitev na eno samo učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Primeri takih dreves so na sliki 5.3. Redkeje se pojavlja delitev



(a) Dendrogram metode *ERM*



(b) Drevo metode *ForcedTree*

Slika 5.2: Primer podobnosti med strukturo dendrograma in drevesa pri 9. ponovitvi poskusa za MTL-problem 5.4. V zgornji polovici dendrograma je učna naloga G_1T_2 kot zadnja oz. najmanj podobna priključena prvi gruči učnih nalog. Prav tako se veja s primeri učne naloge G_1T_2 kot prva odcepi od primerov ostalih učnih nalog, kar pomeni, da metoda za izbiro atributov oceni, da so ti primeri najbolj različni od ostalih.

na par učnih nalog iz iste gruče v eni veji in vse preostale učne naloge v drugi veji. Še redkeje smo opazili trojice učnih nalog iz iste gruče, najredkeje pa četverke učnih nalog iz iste gruče v eni veji. Kot bomo pojasnili v nadaljevanju, smo slednje našli le pri MTL-problemih 5.3 in 5.5. Delitve, kjer bi bilo vseh pet učnih nalog prve gruče v eni veji in preostalih pet učnih nalog druge gruče v drugi veji, nismo našli.

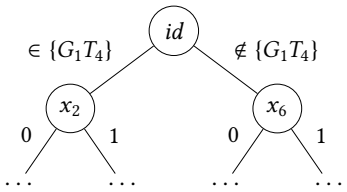
Struktura dreves močno odvisna od konkretnega MTL-problema

Opazili smo, da je struktura dreves, zgrajenih z metodo *ForcedTree*, zelo odvisna od konkretnega MTL-problema.

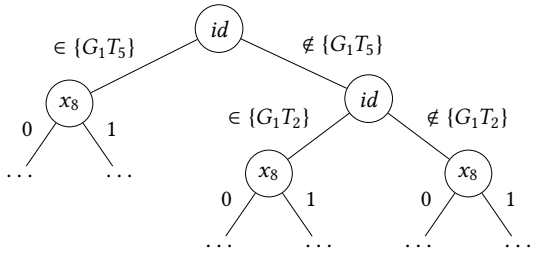
MTL-problema 5.3 in 5.5 sta izstopala po velikem številu pojavitev trojic in četverk učnih nalog iz iste gruče. To so binarizacije vrednosti atributa *id*, kjer v eno vejo skupaj spadajo po tri ali štiri učne naloge iz iste gruče. Pri obeh MTL-problemih so se trojice oz. četverke pojavljale kar v polovici ponovitev eksperimenta. Primer četverke je podan na sliki 5.4. Za razlago tega pojava velja spomniti, da je pri obeh MTL-problemih v eni izmed funkcij prisoten disjunkt z eno samo Boolovo spremenljivko, ki povzroči zelo različno distribucijo pozitivnih in negativnih primerov v obeh gručah učnih nalog. Pri učnih nalogah iz gruče, ki ustreza Boolovi funkciji s singularnim disjunktom, je le-ta blizu $50:50$, medtem ko je za učne naloge iz druge gruče precej bolj neuravnotežena. Posledično je najvišjo vrednost mere za ocenjevanje kvalitete atributov, ki so kandidati za razcep, dobila binarizacija vrednosti atributa *id*, ki v veliki meri ločuje učne naloge ene in druge gruče.

Pri MTL-problemih 5.2, 5.7 in 5.8 smo opazili izrazito veliko število t. i. mešanih parov oz. trojic. To so binarizacije vrednosti atributa *id*, kjer so v eni veji skupaj učne naloge (dve ali več) iz različnih gruč. Pri MTL-problemih 5.2 in 5.8 je bilo po pet mešanih parov oz. trojic, pri MTL-problemu 5.7 pa štirje. Primer mešane trojice je podan na sliki 5.5. Vsi trije MTL-problemi z velikim številom mešanih parov oz. trojic imajo vse disjunkte velikosti vsaj tri. To pomeni, da je precej bolj kompleksno ugotoviti, kateri gruči učnih nalog pripada določena naloga in posledično kratkovidni algoritem za izbiro atributov pri gradnji odločitvenih dreves tu odpove. Hkrati pa tudi za MTL-probleme 5.4, 5.6 in 5.10 velja, da imajo vse disjunkte velikosti vsaj tri, tako da omenjena lastnost ni tudi zadosten pogoj za odkriti fenomen.

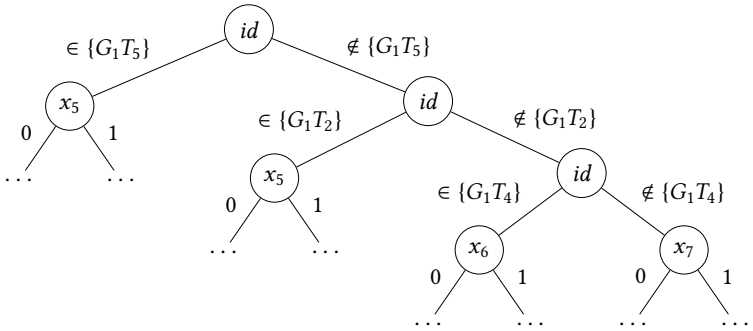
Po izrazito velikem številu delitev vrednosti atributa *id* na posamezne učne naloge (praviloma tudi iz iste gruče) sta izstopala MTL-problema 5.10 in 5.7. Pri prvem je kar v petih ponovitvah poskusa prišlo do 4-kratne delitve na posamezno učno nalogo v



(a) Drevo metode *ForcedTree* za 5. ponovitev MTL-problema 5.4

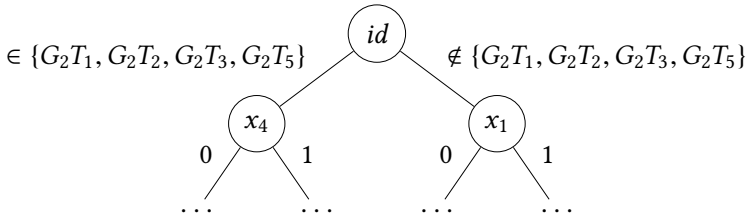


(b) Drevo metode *ForcedTree* za 2. ponovitev MTL-problema 5.6

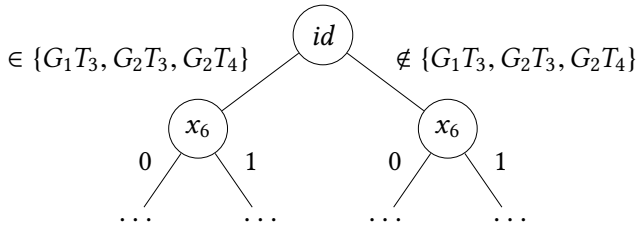


(c) Drevo metode *ForcedTree* za 7. ponovitev MTL-problema 5.1

Slika 5.3: Primeri dreves, kjer se vrednosti atributa *id* delijo na eno samo učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Na podslikah (a), (b) oz. (c) se to zgodi 1-krat, 2-krat oz. 3-krat.



Slika 5.4: Primer četverke učnih nalog iz iste gruče, kjer v levo vejo prvega razcepa drevesa spadajo primeri učnih nalog G_2T_1, G_2T_2, G_2T_3 in G_2T_5 (drevo metode *ForcedTree* za 1. ponovitev MTL-problema 5.5).

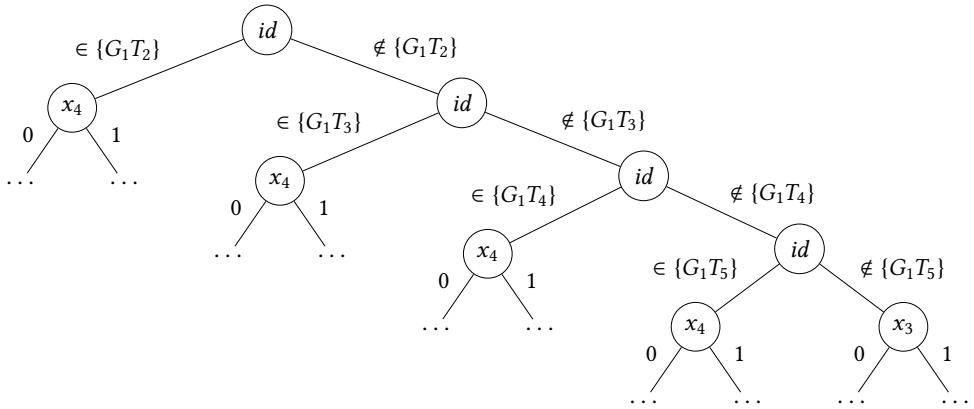


Slika 5.5: Primer t. i. mešane trojice učnih nalog, kjer v levo vejo prvega razcepa spadajo primeri učne naloge G_1T_3 iz prve gruče učnih nalog ter primeri učnih nalog G_2T_3 in G_2T_4 iz druge gruče učnih nalog (drevo metode *ForcedTree* za 2. ponovitev MTL-problema 5.8).

eni veji in preostale učne naloge v drugi veji, pri drugem pa je v treh ponovitvah poskusa prišlo do 4-kratne, v eni ponovitvi pa do 3-kratne prej omenjene delitve. Primer 4-kratne delitve na posamezno učno nalogo v eni veji in preostale učne naloge v drugi veji je podan na sliki 5.6 Podrobnejša analiza dreves je pokazala, da je bila pri omenjenih MTL-problemih za učne naloge iz določene gruče pri nekaterih ponovitvah poskusa porazdelitev razreda izrazito neenakomerna. Na primer, pri ponovitvi poskusa, za katero je drevo prikazano na sliki 5.6, je bil delež primerov učne naloge G_1T_2 (druga učna naloga prve gruče) z razredom *false* enak 0.83. Zato je metoda za binarizacijo vrednosti atributov kot najprimernejšo binarizacijo atributa *id* za prvi razcep drevesa izbrala tisto, kjer je v eni veji le učna naloga G_1T_2, v drugi veji pa vse ostale učne naloge. Delež primerov učne naloge G_1T_3 z razredom *false* je bil 0.80, zato jo je metoda za izbiro in binarizacijo atributov izbrala kot najprimernejšo za drugi razcep drevesa. Podobno sta bili učni nalogi G_1T_4 in G_1T_5 izbrani kot najprimernejši za tretji in četrti razcep, saj sta bila deleža njunih učnih primerov z razredom *false* 0.78 oz. 0.73, medtem ko je bil delež učnih primerov z razredom *false* v preostanku primerov zgolj 0.66. Podobno je visok delež določenega razreda povzročil večkratno delitev na posamezne učno nalogo tudi pri ostalih ponovitvah poskusov za MTL-problema 5.10 in 5.7.

ERM zelo uspešna pri združevanju učnih nalog v gruče

Opazovanje dendrogramov metode *ERM* za različne MTL-probleme je razkrilo, da je metoda *ERM* zelo uspešna pri združevanju učnih nalog v gruče. Praviloma vedno najde prvotno pripadnost učnih nalog gručam. V primerih, ko ni 100% uspešna, pa kvečjemu kakšnih podgruč učnih nalog ne združi v prvotne gruče, nikoli pa ne združi dveh učnih nalog, ki pripadata različnim gručam.

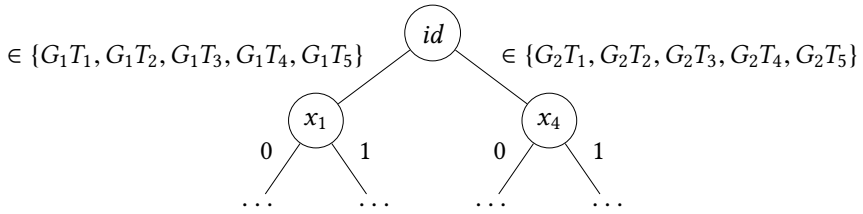


Slika 5.6: Primer 4-kratne delitve vrednosti atributa *id* na posamezno učno nalogo v eni veji in preostale učne naloge v drugi veji (drevo metode *ForcedTree* za 2. ponovitev MTL-problema 5.10).

Primerjava strukture dreves in dendrogramov ob dodajanju šuma v podatke

V drugem delu primerjave strukture dreves, kjer se veje cepijo po atributu *id*, z dendrogrami, ki prikazujejo potek združevanja učnih nalog metode *ERM*, smo opazovali poskuse, kjer je bila stopnja šuma (parameter *noise*) > 0.0. Primerjali smo drevesa, dobljena z metodo *ForcedTree* (ki so vedno imela prvi razcep po atributu *id*) ter dendrograme metode *ERM*.

Pri poskusih, kjer je bil parameter *noise* = 0.1, ni bilo opaziti večjih sprememb glede na poskuse, kjer je bil parameter *noise* = 0.0. Tudi tokrat med strukturo dreves in dendrogrami ni bilo nobenih podobnosti. Še naprej je bila najpogostejša oblika binarizacije vrednosti atributa *id* delitev na eno samo učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Prav tako sta tudi v tem naboru poskusov MTL-problema 5.3 in 5.5 izstopala po velikem številu pojavitev trojic in četverk učnih nalog iz iste gruče. Pri prvem jih je bilo pet, pri drugem pa celo osem. Pri 1. ponovitvi poskusa za MTL-problem 5.5, katerega drevo je prikazano na sliki 5.7, smo prvič (in edinkrat) opazili peterko učnih nalog iz iste gruče v eni veji in preostalih pet učnih nalog druge gruče v drugi veji. Ponovno sta imela MTL-problema 5.2 in 5.8 izrazito veliko t. i. mešanih parov oz. trojic. Pri prvem so bili prisotni v devetih, pri drugem pa v šestih ponovitvah poskusa. Prvič smo opazili tudi četverke mešanih učnih nalog in sicer pri MTL-problemu 5.2. Po izrazito velikem številu delitev vrednosti atributa *id* na posamezne učne naloge je izstopal le še MTL-problem 5.10, kjer je kar v šestih ponovitvah poskusa prišlo vsaj do 3-kratne delitve na posamezno učno nalogo v eni



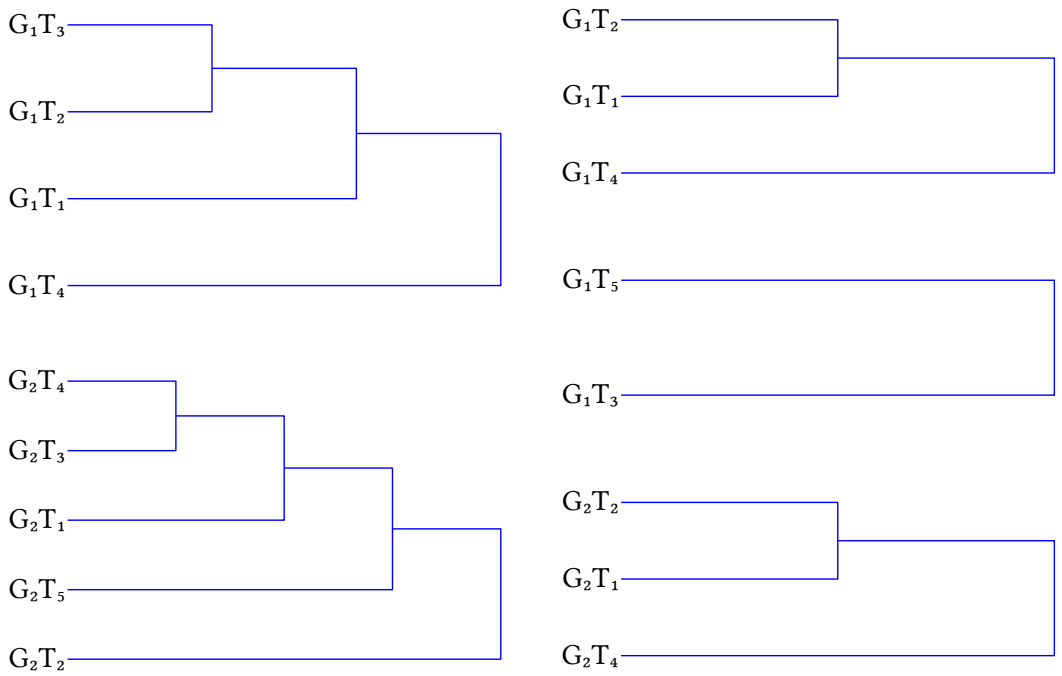
Slika 5.7: Primer peterke učnih nalog iz iste gruče v eni veji in preostalih pet učnih nalog druge gruče v drugi veji. (drevo metode *ForcedTree* za 1. ponovitev poskusa za MTL-problem 5.5).

veji in preostale učne naloge v drugi veji. Opazovanje dendrogramov metode *ERM* za različne MTL-probleme je razkrilo, da le-ta ponovno v večini primerov deluje zelo dobro. Nekoliko več težav pri odkrivanju prvotnih gruč učnih nalog je imela le pri MTL-problemih 5.1 in 5.3, kjer je t. i. dobro gručenje učnih nalog (t.j. gručenje, kjer so bile v vsaki od obeh gruč vsaj štiri od petih učnih nalog) našla le v petih oz. šestih ponovitvah poskusa. V povprečju je delež poskusov, kjer je metoda *ERM* našla dobro gručenje, znašal 0.70. Izjema glede dobrega delovanja metode *ERM* je bil le MTL-problem 5.5, kjer metoda *ERM* ni nikoli našla dobrega gručenja učnih nalog. Velja omeniti, da kljub temu metoda *ERM* ni nikoli napačno združila dveh učnih nalog iz različnih gruč. Gotovo je to tudi razlog, da pri poskusih za *noise* = 0.1 metoda *ERM* le za ta MTL-problem ni dajala statistično značilno boljših rezultatov od metod *Tree* in *ForcedTree* (rezultati v tabeli 5.2). Tokrat smo tudi prvič opazili, da je metoda *ERM* napačno združila učne naloge iz različnih gruč. To se je zgodilo pri 9. ponovitvi poskusa za MTL-problem 5.2 ter pri 6. ponovitvi poskusa za MTL-problem 5.4. Vseeno je v primerjavi s 100 ponovitvami poskusa to zelo majhna številka.

Pri poskusih, kjer smo še nekoliko povečali stopnjo šuma (*noise* = 0.2), ni bilo opaziti večjih sprememb glede na prejšnje poskuse (*noise* = 0.1). Ponovno med strukturo dreves in dendrogrami ni bilo nobenih podobnosti. Najpogostejši tip binarizacije vrednosti atributa *id* ostaja delitev na eno samo učno nalogo v eni veji in vse preostale učne naloge v drugi veji. MTL-problema 5.3 in 5.5 ponovno izstopata po velikem številu pojavitev trojic in četverk učnih nalog iz iste gruče. Pri prvem jih je bilo šest, pri drugem pa štirje. Poleg MTL-problemov 5.2 in 5.8, ki sta imela že v prejšnjem nizu poskusov izrazito veliko število t. i. mešanih parov, trojic oz. četverk, so se le-ti v velikem številu sedaj pojavili tudi pri MTL-problemih 5.7 in 5.9. Pri MTL-problemu 5.2 so se pojavili v petih, pri MTL-problemu 5.7 v šestih, pri MTL-problemu 5.8 v desetih in pri MTL-problemu 5.9 v petih ponovitvah poskusa. Po velikem številu delitev vre-

dnosti atributa *id* na posamezne učne naloge je ponovno izstopal MTL-problem 5.10, kjer je v petih ponovitvah poskusa prišlo vsaj do 3-kratne delitve na posamezno učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Opazovanje dendrogramov metode *ERM* je razkrilo, da je imela le-ta pri tej stopnji šuma že nekoliko več težav pri določanju gruč učnih nalog. Za večino MTL-problemov se je namreč delež poskusov, kjer je metoda *ERM* našla t. i. dobro gručenje učnih nalog (t.j. gručenje, kjer so bile v vsaki od obeh gruč vsaj štiri od petih učnih nalog), zmanjšalo. V povprečju je le-ta znašal 0.56 (prej 0.70). Primer dobrega in slabšega gručenja učnih nalog je podan na sliki 5.8. Kljub temu pa je metoda *ERM* ohranila izredno majhen delež napačno združenih učnih nalog. To se je zgodilo le 2-krat: pri 1. in 5. ponovitvi poskusa za MTL-problem 5.8. Naj še omenimo, da sta se napaki zgodili pri drugem MTL-problemu kot pri poskusih za *noise* = 0.1. Čeprav je bil delež učnih nalog, kjer je metoda *ERM* našla dobro gručenje učnih nalog, tokrat manjše, je metoda *ERM* še vedno pri vseh desetih MTL-problemih dosegala statistično značilno boljše rezultate od metod *Tree* in *ForcedTree* (rezultati so podani v tabeli 5.3).

Tudi pri poskusih, kjer je bila vrednost parametra *noise* = 0.3, ni bilo opaziti večjih sprememb glede na prejšnje poskuse. Ponovno med strukturo dreves in dendrogrami ni bilo nobenih podobnosti. Najpogostejši tip binarizacije vrednosti atributa *id* je ostala delitev na eno samo učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Po velikem številu pojavitev trojic in četverk učnih nalog iz iste gruče ponovno izstopata MTL-problema 5.3 in 5.5, le da jih je bilo tokrat pri drugem šest in pri prvem štirje. Po velikem številu t. i. mešanih parov in trojic ponovno izstopajo MTL-problemi 5.2, 5.8 in 5.9. Tokrat poleg njih izstopa še MTL-problem 5.6, medtem ko MTL-problem 5.7 ne izstopa več. Pri MTL-problemu 5.2 so se pojavili v osmih, pri MTL-problemu 5.6 v petih, pri MTL-problemu 5.8 v šestih in pri MTL-problemu 5.9 v petih ponovitvah poskusa. Ponovno je po velikem številu delitev vrednosti atributa *id* na posamezne učne naloge izstopal MTL-problem 5.10, kjer je v petih ponovitvah poskusa prišlo vsaj do 3-kratne delitve na posamezno učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Analiza dendrogramov metode *ERM* je razkrila, da se je njeno delovanje s povečanjem stopnje šuma zopet nekoliko poslabšalo. Delež poskusov, kjer je metoda *ERM* našla t. i. dobro gručenje učnih nalog, se je za večino MTL-problemov namreč zopet znižal in znašal le še 0.42 (pri vrednosti parametra *noise* = 0.2 je znašal 0.56). Tokrat se je povečalo tudi število ponovitev poskusov, kjer je metoda *ERM* napačno združila dve učni nalogi iz različnih gruč. To se je zgodilo 5-krat pri MTL-problemu 5.4, 3-krat pri MTL-problemu 5.6 in 1-krat pri MTL-problemu 5.8.



(a) Dendrogram metode *ERM* za 4. ponovitev poskusa za MTL-problem 5.5

(b) Dendrogram metode *ERM* za 5. ponovitev poskusa za MTL-problem 5.5

Slika 5.8: Primer dobrega in slabšega gručenja učnih nalog z metodo *ERM*. Na podsliki (a) je primer t. i. dobrega gručenja učnih nalog, saj so v vsaki od obeh gruč vsaj štiri od petih učnih nalog. Gručenje na podsliki (b) je primer slabšega gručenja, saj je metoda *ERM* namesto prvotne prve gruče odkrila dve manjši gruči z dvema oz. tremi učnimi nalogami ter namesto prvotne druge gruče eno manjšo gručo s tremi učnimi nalogami, preostali dve učni nalogi pa ni pridružila nobeni gruči. Še vedno pa velja, da metoda *ERM* ni napačno združila dveh učnih nalog, ki pripadata različnim gručam.

Skupaj torej 9-krat. To se zelo dobro ujema z rezultati iz tabele 5.4, kjer metoda *ERM* ravno za te tri MTL-probleme dosega najnižje povprečne vrednosti AUC: 0.91, 0.91 in 0.94 (zaporedoma).

Pri zadnji seriji poskusov, kjer je bila vrednost parametra *noise* = 0.5, prav tako ni bilo večjih odstopanj od prejšnjih poskusov. Tako kot vedno, tudi tokrat ni bilo opaziti nobenih podobnosti med strukturo dreves in dendrogrami. Najpogostejši tip binarizacije vrednosti atributa *id* je ostala delitev na eno samo učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Tokrat je po velikem številu trojic in četverk učnih nalog iz iste gruče izstopal le še MTL-problem 5.3, kjer so se pojavile šestkrat, medtem ko MTL-problem 5.5 s tremi pojavitvami ni več izstopal. MTL-problemi 5.2, 5.6, 5.8 in 5.9 ponovno izstopajo po velikem številu t. i. mešanih parov, trojic oz. četverk. Pojavljajo so se v sedmih, osmih, osmih in šestih (zaporedoma) ponovitvah poskusa. Poleg teh pa sta po velikem številu mešanih parov, trojic oz. četverk tokrat izstopala še MTL-problema 5.4 in 5.7. Pri prvem so se pojavili v petih, pri drugem pa v osmih ponovitvah poskusa. Ponovno je po velikem številu delitev vrednosti atributa *id* na posamezne učne naloge izstopal MTL-problem 5.10, kjer je v štirih ponovitvah poskusa prišlo vsaj do 3-kratne delitve na posamezno učno nalogo v eni veji in vse preostale učne naloge v drugi veji. Opazovanje dendrogramov metode *ERM* je razkrilo, da se je njeno delovanje občutno poslabšalo. Delež ponovitev poskusov za posamezne MTL-probleme, kjer je metoda *ERM* našla t. i. dobro gručenje učnih nalog, se je v primerjavi s prejšnjo serijo poskusov za večino učnih nalog občutno poslabšal in je v povprečju znašal le še 0.07 (za vrednost parametra *noise* = 0.3 je znašal 0.42). Prav tako se je občutno povečalo število ponovitev poskusov, kjer je metoda *ERM* napačno združila dve učni nalogi iz različnih gruč. Tokrat je bilo to prisotno pri vseh MTL-problemih. V povprečju je delež ponovitev poskusov z napačnim združevanjem kar 0.57. To je skladno z rezultati iz tabele 5.5, kjer so se vrednosti AUC metode *ERM* v primerjavi s prejšnjimi poskusi občutno znižale.

5.3.3 Ugotovitve

Rezultati so popolnoma potrdili prvo hipotezo, ki pravi, da v večini primerov metoda *Tree* za prvi razcep izbere atribut *id*. Deleži dreves metode *Tree*, s prvim razcepom po atributu *id*, so bili za večino MTL-problemov zelo visoki.

Nižje deleže le-teh smo opazili le pri določenih MTL-problemih, ki imajo zelo specifične lastnosti. Ena od njih je bila prisotnost singularnega disjunkta v eni od obeh

Boolovih funkcij (pri MTL-problemih 5.3 in 5.5). Atribut tega disjunkta je zelo informativen, saj je v primeru, da ima vrednost *true*, tudi vrednost celotne Boolove funkcije enaka *true*. To Boolovo funkcijo si deli polovica učnih nalog in posledično polovica učnih primerov, kar pomeni, da je tak atribut zelo informativen in ga metoda za izbiro atributov pri gradnji odločitvenih dreves izbere pred ostalimi. Druga lastnost, na nek način simetrična prvi, pa so zelo pogoste pojavitve določenega atributa (pri MTL-problemih 5.4 in 5.8) v eni od Boolovih funkcij. Tak atribut je zelo informativen, saj je v primeru, da nastopa v (skoraj) vseh disjunktih in ima vrednost *false*, tudi vrednost celotne Boolove funkcije (praviloma) *false*. Ponovno velja, da si Boolovo funkcijo deli polovica učnih nalog in posledično polovica učnih primerov, kar pomeni, da je tak atribut v primerjavi z ostalimi zelo informativen in bo z veliko verjetnostjo izbran za prvi razcep drevesa.

Naša druga hipoteza se je izkazala za povsem napačno. Med strukturo dreves, kjer se veje cepijo po atributu *id*, in dendrogrami, ki prikazujejo potek združevanja učnih nalog z metodo *ERM*, ni bilo moč najti nobene podobnosti. Večinoma so si bili celo nasprotni: tisto učno nalogo, ki jo je drevo najprej odcepilo v svojo vejo kot najbolj različno v primerjavi z ostalimi, je metoda *ERM* združila med prvimi.

Nekoliko nas je presenetilo, da je običajno kot rezultat optimalne binarizacije vrednosti atributa *id* delitev na eno samo učno nalogo v eni veji in vse preostale učne naloge v drugi veji. V skladu s pričakovanji se je pokazalo, da je struktura dreves, zgrajenih z metodo *ForcedTree*, zelo odvisna od konkretnega MTL-problema. MTL-problema 5.3 in 5.5 sta izstopala po velikem številu pojavitev t. i. trojic, četverk oz. peterk učnih nalog iz iste gruče. To so binarizacije vrednosti atributa *id*, kjer v eno vejo skupaj spadajo po tri, štiri oz. pet učnih nalog iz iste gruče. Pri MTL-problemih 5.2 in 5.8 smo v velikem deležu poskusov opazili izrazito veliko število t. i. mešanih parov, trojic oz. četverk. To so binarizacije vrednosti atributa *id*, kjer so v eni veji skupaj učne naloge (dve ali več) iz različnih gruč. Občasno smo jih opazili tudi pri MTL-problemih 5.7, 5.9, 5.6 in 5.4. MTL-problem 5.10 je izstopal po izrazito velikem številu delitev vrednosti atributa *id* na posamezne učne naloge (praviloma tudi iz iste gruče). Običajno je v polovici ponovitev poskusa prišlo do vsaj 3-kratne delitve na posamezno učno nalogo v eni veji in vse preostale učne naloge v drugi veji.

Presenetljiva ugotovitev analize rezultatov eksperimentov je, da se struktura dreves, kjer se veje cepijo po atributu *id*, s povečevanjem stopnje šuma (parametra *noise*), ni kaj dosti spreminjala. Nekoliko se je s povečevanjem stopnje šuma povečalo le število MTL-problemov s pogostimi pojavitvami t. i. mešanih parov, trojic oz. četverk,

drugih sprememb pa ni bilo opaziti. Razlog je verjetno ta, da je šum v povprečju enakomerno prisoten v podatkih vseh učnih nalog, zato le kdaj bolj in kdaj manj oteži izbiro najprimernejše binarizacije vrednosti atributa *id* za naslednji razcep drevesa. V povprečju pa algoritem za izgradnjo odločitvenih dreves še vedno uspe odkriti prvotne zakonitosti v podatkih in posledično zgraditi podobna drevesa kot pri poskusih brez šuma v podatkih.

Bolj pričakovana je bila ugotovitev, da uspešnost združevanja učnih nalog z metodo *ERM* s povečevanjem stopnje šuma pada. Pri podatkih brez šuma je metoda *ERM* zelo uspešna pri združevanju učnih nalog v gruče. Praviloma vedno najde prvotno pripadnost učnih nalog gručam. V primerih, ko ni 100% uspešna, pa kvečjemu kakšnih podgruč učnih nalog ne združi v prvotne gruče, nikoli pa ne združi dveh učnih nalog, ki pripadata različnim gručam. Ko podatke nekoliko zašumimo ($noise \in \{0.1, 0.2\}$), se začne pojavljati napačno združevanje učnih nalog iz različnih gruč. Vendar se je v obeh sklopih poskusov to pojavilo zelo redko — le v 2% poskusov. Pri poskusih z vrednostjo parametra *noise* = 0.3 se je napačno združevanje zgodilo v 9% poskusov, pri poskusih z vrednostjo parametra *noise* = 0.5 pa kar v 57% poskusov. Vzporedno s tem se je zniževal tudi delež poskusov, kjer je metoda *ERM* našla t. i. dobro gručenje učnih nalog (t.j. gručenje, kjer so bile v vsaki od obeh gruč vsaj štiri od petih učnih nalog). Za vrednosti parametra *noise* 0.1, 0.2, 0.3 in 0.5 so bili ti deleži 0.70, 0.56, 0.42 in 0.07 (zaporedoma). Tako obnašanje metode *ERM* je bilo pričakovano. S povečevanjem stopnje šuma v podatkih se je namreč oteževala naloga ugotavljanja, katere pare učnih nalog je smiselno združiti, ker skupaj prinašajo zmanjšanje pričakovane napovedne napake. Po eni strani je *ERM* čedalje težje poiskala t. i. dobro gručenje učnih nalog in je namesto tega vrnila bolj razdrobljeno gručenje z večjim številom manjših gruč. Po drugi strani pa je metoda *ERM* tudi čedalje pogosteje napačno združevala učne naloge iz različnih gruč. Vsi ti opisni kazalci manj uspešnega združevanja učnih nalog z metodo *ERM* ob povečevanju stopnje šuma so popolnoma v skladu z rezultati kvantitativne primerjave zmogljivosti, podanimi v tabelah 5.1–5.5.

5.4 Diskusija

Pomembno spoznanje, ki je rezultat tega eksperimenta, je, da je binarizacija vrednosti atributov pri gradnji odločitvenih dreves, tako kot širše izbira atributov za razcep, z izbrano mero ocenjevanja informativnosti atributov (informacijskim prispevkom) zelo kratkoviden mehanizem. Pri izbiri razcepa namreč ne zna upoštevati vpliva ostalih

atributov (npr. v situaciji, ko je informacijski prispevek dveh atributov gledano posamično zelo majhen, skupaj pa zelo velik).

Poleg tega tudi ne zna upoštevati količine primerov, ki bo po razcepu pripadla eni in drugi veji drevesa. Slednje je v našem eksperimentu zelo pomembno, saj ima posamezna učna nalogo premalo lastnih primerov, da bi z njimi zgradila dober napovedni model, in potrebuje primere drugih učnih nalog, ki pripadajo isti gruči sorodnih učnih nalog. Metoda za gradnjo odločitvenih dreves pa gleda le na čistost primerov v obeh podmnožicah, ki bi nastali po razcepu in bo zato primere učne naloge, ki je navidezno drugačna od ostalih, dal v svoje poddrevo. To povzroči, da je v tistem poddrevesu premalo primerov, medtem ko so primeri vseh ostalih učnih nalog v drugem poddrevesu.

Možnost za premostitev prve hibe (tj. kratkovidnosti) pri binarizaciji vrednosti atributov oz. izbiri atributa za razcep, je t. i. gledanje naprej. To pomeni, da binarizacija vrednosti atributov oz. izbira atributa za razcep ni več le požrešna, ampak gleda tudi v globino. S tem pa lahko upošteva tudi ostale atribute. V prihodnje bi bilo zanimivo preveriti, kaj se zgodi pri binarizacija vrednosti atributov, če za mero informativnosti atributov uporabimo nekratkovidno mero kot je na primer ReliefF [65].

Prav tako bi bilo zanimivo preizkusiti, kako deluje binarizacija vrednosti atributov oz. izbira atributov za razcep, ki ji dodamo hevristiko, ki upošteva še, kako uravnoteženi sta obe podmnožici, ki bi jih dobili po razcepu drevesa.

6. POGLAVJE

Zaključek

Glavni prispevek doktorske disertacije je razvoj in implementacija nove metode za hkratno učenje več nalog (angl. multi-task learning, MTL), imenovane *Error-reduction merging (ERM)*, ki avtomatsko odkrije gruče sorodnih učnih nalog (le na podlagi podatkov samih) ter združi podatke učnih nalog iz iste gruče, kar vodi k izboljšanju napovednih modelov za posamezne učne naloge. Večina dosedanjih MTL-pristopov temelji na predpostavki, da so vse učne naloge sorodne in primerne za skupno učenje (oz. da obstaja ekspert, ki lahko določi, katere naloge so sorodne). Vendar združevanje podatkov dveh nesorodnih nalog lahko vodi tudi do poslabšanja rezultatov na obeh nalogah, kar imenujemo negativni prenos. Metoda *ERM* sodi v skupino novejših MTL-pristopov, ki se skušajo znebiti predpostavke, da so vse učne naloge sorodne. Za razliko od ostalih pristopov *ERM* sorodnost učnih nalog definira preko zmanjšanja napovedne napake modela, zgrajenega na združenih podatkih. Uporablja predpostavko, da sta dve učni nalogi sorodni, če ima model, zgrajen na združenih podatkih, manjšo napovedno napako od modelov, zgrajenih na podatkih za posamezni učni nalogi. Pri *ERM* števila gruč ni potrebno podati vnaprej, metoda ga poišče samodejno. Prav tako kot večina pristopov uporablja predpostavko, da lahko učne naloge razdelimo v gruče, med katerimi ni prekrivanj. Večina MTL-pristopov je vezanih na določeno osnovno učno metodo, znotraj *ERM* pa je moč uporabiti poljubno metodo nadzorovanega učenja (npr. metodo podpornih vektorjev, logistično regresijo, odločitveno drevo).

V disertaciji smo podali tudi definicijo različnih lastnosti MTL-problema, ki so pomembne z vidika avtomatskega odkrivanja gruč sorodnih učnih nalog z namenom izboljšanja napovednih modelov za posamezno učno nalogo:

- število učnih nalog (definicija 2.1),

- števila primerov posameznih učnih nalog (definicija 2.2),
- število gruč različnih tipov učnih nalog (definicija 2.3).

Izvedli smo obsežno eksperimentalno študijo obnašanja metode *ERM* glede na prej definirane lastnosti MTL-problema ter glede na stopnjo šuma v podatkih in uporabljeno osnovno učno metodo znotraj *ERM*.

Z eksperimenti na sintetičnih MTL-problemih učenja Boolovih funkcij smo prišli do naslednjih ugotovitev:

- Uspešnost metode *ERM* se s povečevanjem števila učnih nalog v vsaki gruči oz. celotnega števila učnih nalog povečuje. *ERM* namreč že pri dveh učnih nalogah v vsaki gruči dosega boljše rezultate od metod *NoMerging* in *MergeAll*, od tam naprej pa razlika v AUC samo še narašča. Pri manjšem številu učnih nalog je razlika v AUC glede na metodo *Oracle* večja, potem pa se postopoma zmanjšuje.
- Uspešnost metode *ERM* se s povečevanjem števil primerov posameznih učnih nalog hitro povečuje. Njene vrednosti AUC so vseskozi višje od metode *NoMerging*, metodo *MergeAll* pa tudi hitro presežejo. Za 100 ali več primerov na posamezno učno nalogo je metoda *ERM* dosegala enako dobre rezultate kot metoda *Oracle*.
- Uspešnost metode *ERM* se s povečevanjem števila gruč različnih tipov učnih nalog zmanjšuje. Čeprav vrednosti AUC metode *ERM* ostajajo večje od metod *NoMerging* in *MergeAll*, se razlika v vrednosti AUC glede na metodo *Oracle* s povečevanjem števila gruč sorodnih učnih nalog povečuje.
- Uspešnost metode *ERM* se s povečevanjem stopnje šuma v podatkih pričakovano zmanjšuje. Za manjše stopnje šuma je *ERM* po absolutnih vrednostih AUC še relativno blizu metodi *Oracle*, od tam naprej pa začne njena krivulja AUC hitro padati. Pri najvišji stopnji šuma *ERM* povsem odpove in deluje podobno ali slabše od metode *MergeAll*.
- Uspešnost metode *ERM* je v splošnem neodvisna od osnovne učne metode (v eksperimentih smo uporabili metodo podpornih vektorjev, odločitveno drevo, k-najbližjih sosedov), ki je uporabljena znotraj *ERM*.

Rezultati na realnih klasifikacijskih in regresijskih problemih so potrdili, da metoda *ERM* dobro deluje tudi na praktičnih problemih. Pri obeh problemih prepoznavanja ročno napisanih števk (*USPS digits* in *MNIST digits*) smo po pretvorbi na nove

MTL-probleme z večjim številom t. i. podnalog z metodo *ERM* dosegali najboljše rezultate. Izkazalo se je, da je *ERM* najbolj koristna za tiste učne naloge, za katere vse MTL-metode dosegajo slabše rezultate (in so na nek način težje naučljive).

Pri realnem regresijskem MTL-problemu napovedovanja doseženega števila točk učencev na zaključnem preizkusu znanja (*School*) se je sicer najbolje odrezala metoda *ERM*, vendar se njeni rezultati statistično značilno ne razlikujejo od rezultatov ostalih metod. Tudi pri drugem realnem regresijskem MTL-problemu (*Computer Survey*), kjer je šlo za napovedovanje naklonjenosti oseb nakupu določenega računalnika, se je najbolje odrezala metoda *ERM*. Tokrat so bili njeni rezultati statistično značilno boljši od rezultatov ostalih metod. Pri obeh regresijskih MTL-problemih smo ugotovili, da so rezultati metode *ERM* za posamezne učne naloge najbolj stabilni (tj. najmanj variirajo) ter da metoda *ERM* za večino učnih nalog ne dosega niti najslabših niti najboljših rezultatov.

Primerjava z rezultati sorodnih metod za realne MTL-probleme je razkrila, da *ERM* v določenih primerih (za problema *USPS digits* in *MNIST digits*) dosega boljše rezultate, v drugih primerih (za problem *School*) primerljive rezultate, kdaj (za problem *Computer Survey*) pa tudi slabše rezultate od sorodnih metod. Velja omeniti, da smo pri primerjavi rezultatov različnih implementacij enakih metod našli določene nekonsistentnosti, ki jih lahko do neke mere pojasnimo z razliko v uporabljeni osnovni učni metodi ter izbiri njenih parametrov, sicer pa ostajajo nepojasnjene.

Študija primera pohitritve učenja robotskega agenta z izvajanjem eksperimentov v kompleksnejšem okolju je pokazala, da je metoda *ERM* zmožna odkriti, kateri objekti imajo isti vedenjski tip, in združiti njihove podatke, kar vodi do hitrejšega učenja. V MTL-terminologiji to pomeni, da je *ERM* odkrila gruče učnih nalog istega tipa in jih združila, kar je vodilo do izboljšanja napovednih modelov posameznih učnih nalog. Ugotovitve študije se v celoti ujemajo z ugotovitvami glede obnašanja metode *ERM* na sintetičnih MTL-problemih učenja Boolovih funkcij. Povečevanje števila akcij, ki jih izvede robot, se je namreč izkazalo za analogno povečevanju števila primerov posameznih učnih nalog.

V okviru interpretacije binarizacije vrednosti atributov pri gradnji odločitvenih dreves v smislu delovanja *ERM* smo naredili kvantitativno in kvalitativno primerjavo metode *ERM* z dvema metodama, ki uporabljata binarizacijo vrednosti atributov pri gradnji odločitvenega drevesa. *ERM* se je v kvantitativni primerjavi izkazala za superiorno. Z njo smo vedno dosegali najboljše rezultate, pri nižji stopnji šuma praviloma vedno tudi statistično značilno boljše rezultate od preostalih dveh metod. Rezultati

kvalitativne primerjave zgrajenih modelov so pokazali, da v večini primerov metoda za izgradnjo odločitvenega drevesa za prvi razcep izbere atribut *id*, ki predstavlja identifikator učnih nalog. Poleg tega smo ugotovili, da med strukturo dreves, kjer se veje cepijo po atributu *id*, in dendrogrami, ki prikazujejo potek združevanja učnih nalog z metodo *ERM*, ni bilo moč najti nobene podobnosti. Večinoma so si bili celo nasprotni: tisto učno nalogo, ki jo je drevo najprej odcepilo v svojo vejo kot najbolj različno v primerjavi z ostalimi, je metoda *ERM* združila med prvimi. Potrdili smo znano ugotovitev, da je binarizacija vrednosti atributov pri gradnji odločitvenih dreves, tako kot širše izbira atributov za razcep, z izbrano mero ocenjevanja informativnosti atributov (informacijskim prispevkom) zelo kratkoviden mehanizem. Pri izbiri razcepa namreč ne zna upoštevati vpliva ostalih atributov ali količine primerov, ki bo po razcepu pripadli eni in drugi veji drevesa.

V okviru doktorske disertacije smo razvili *PyMTL (Python library for Multi-task learning)*, Pythonovo knjižnico, ki implementira ogrodje za MTL (vključno z generiranjem podatkov, izvajanjem eksperimentov in vizualizacijo rezultatov). Poleg tega vključuje implementacijo metode *ERM* in pomožnih MTL-metod (*NoMerging*, *MergeAll* in *Oracle*). Je prosto dostopna pod pogoji licence *GNU General Public License*, bodisi različice 3 ali katerekoli kasnejše različice, na portalu GitHub [44].

Tekom raziskav smo naleteli tudi na nekaj novih idej in vprašanj, ki bi jih bilo v prihodnosti vredno raziskati.

Glede na veliko časovno zahtevnost metode *ERM* bi bilo koristno, če bi jo lahko pohitrili. Ena od možnosti za pohitritev je uporaba predhodnega filtriranja parov učnih nalog. Ugotavljanje, ali par učnih nalog zadošča kriterijem za združevanje, namreč vključuje izgradnjo treh modelov z osnovno učno metodo, kar je v časovnem smislu precej potratno. Predhodno filtriranje bi moralo imeti naslednje lastnosti: časovno bi moralo biti manj potratno od ugotavljanja zadoščanja kriterijem metode *ERM*, pomembno bi moralo zmanjšati število parov učnih nalog, za katere bi morala *ERM* kljub vsemu ugotavljati, ali zadoščajo kriterijem za združevanje, hkrati pa ne bi smelo vplivati na končno uspešnost metode *ERM*.

Druga možnost za pohitritev metode *ERM* bi bilo dinamično spreminjanje števila ponovitev prečnega preverjanja (parameter k) znotraj le-te. Ko imajo učne naloge malo podatkov, je za čim boljšo točnost združevanja z metodo *ERM* zaželeno, da je število ponovitev prečnega preverjanja čim večje. Časovno to ni problematično, saj večina osnovnih učnih metod za majhno število primerov hitro deluje. Ko pa imajo učne naloge veliko podatkov, zmanjšanje števila ponovitev prečnega preverjanja bi-

stveno ne vpliva na oceno napovednih napak in posledično kvalitete združevanja z metodo *ERM*. Ima pa velik pozitivni časovni učinek, saj vsaka ponovitev prečnega preverjanja traja dalj časa, saj osnovne učne metode za večje število primerov delujejo počasneje.

Naslednja možnost za nadaljnje delo zajema razširitev metode *ERM* v smislu nesimetričnega združevanja učnih nalog v gruče. To pomeni, da bi za vsako učno nalogo imeli svojo gručo sorodnih učnih nalog, kamor bi spadale tiste učne naloge, ki za dano učno nalogo izboljšajo napovedni model. Različne gruče sorodnih učnih nalog bi se v tem primeru prekrivale. Motivacija za tak pristop je na primer izdelava priporočilnega sistema za filme, kot smo ga podali v razdelku 2.1. Pri takem MTL-problemu lahko namreč naletimo na nekega uporabnika, ki si je ogledal zelo malo filmov in bi mu zelo koristilo, če bi njegove podatke združili s podatki sorodnega uporabnika, ki si je ogledal precej več filmov. Za drugega uporabnika, ki si je ogledal več filmov, pa bi tako združevanje lahko vodilo do poslabšanja rezultatov, saj njegovi podatki že sami zase zadoščajo za izgradnjo dobrega napovednega modela.

Zanimivo odkritje eksperimentov na sintetičnih MTL-problemih ter študije primera pohitritve učenja robotskega agenta z izvajanjem eksperimentov v bolj kompleksnem okolju je, da je metoda *MergeAll* pri poskusu s spreminjanjem števila primerov posameznih učnih nalog na začetku (pri majhnem številu učnih primerov) boljša od metod *ERM* in *NoMerging*. To odkritje nakazuje na potencialno novo MTL-metodo, ki bi uporabljala drugačno hevristiko za potek združevanja učnih nalog: metoda bi na začetku združila vse učne naloge v eno gručo in potem postopoma, ko bi bilo za vsako učno nalogo na voljo več podatkov, iz te gruče odstranjevala tiste učne naloge, ki ne bi več ustrezale določenim kriterijem za združevanje. Idejno bi bil tak pristop podoben principu *vzvratnega odstranjevanja* (angl. *backward elimination*), kot se uporablja pri postopni regresiji ali izbiri podmnožice atributov.

Še ena možnost nadgradnje metode *ERM* je izboljšanje končnih napovednih modelov posameznih učnih nalog. Metoda *ERM* ima namreč omejitev, da za vsako gručo sorodnih učnih nalog zgradi le po en model. To pomeni, da si vse učne naloge iz iste gruče delijo isti model. Za izboljšanje končnih napovednih modelov učnih nalog bi znotraj *ERM* lahko uporabili poljubno standardno MTL-metodo, ki za vsako učno nalogo zgradi svoj model, pri čemer upošteva omejitev, da morajo biti parametri vseh modelov blizu ali da imajo vse učne naloge skupen latenten nabor atributov (podrobnosti v razdelku 1.2). Potem, ko bi bilo združevanje učnih nalog v gruče končano, bi *ERM* nad vsako gručo pognala izbrano standardno MTL-metodo. Pričakujemo, da bi

to vodilo do izboljšanja končnih napovednih modelov za posamezne učne naloge.

Literatura

[1] Hendrik Blockeel. Learning, Attribute-Value. In Werner Dubitzky, Olaf Wolkenhauer, Kwang-Hyun Cho, and Hiroki Yokota, editors, *Encyclopedia of Systems Biology*, pages 1112–1114. Springer, 2013.

[2] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2012.

[3] Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, 1997.

[4] Sebastian Thrun and Lorien Pratt. *Learning to Learn*. Kluwer Academic Publishers, 1998.

[5] Sebastian Thrun and Lorien Pratt. Learning to Learn: Introduction and Overview. In *Learning to Learn*, pages 3–17. Kluwer Academic Publishers, 1998.

[6] Jonathan Baxter. A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling. *Machine Learning*, 28:7–39, 1997.

[7] Bart Bakker and Tom Heskes. Task Clustering and Gating for Bayesian Multitask Learning . *The Journal of Machine Learning Research*, 4:83–99, 2003.

[8] Shai Ben-David and Reba Schuller. Exploiting Task Relatedness for Multiple Task Learning. In Bernhard Schölkopf and Manfred Warmuth, editors, *Learning Theory and Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 567–580. Springer, 2003.

[9] Theodoros Evgeniou and Massimiliano Pontil. Regularized Multi–Task Learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 109–117. ACM, 2004.

[10] Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-Task Learning for Classification with Dirichlet Process Priors . *The Journal of Machine Learning Research*, 8:35–63, 2007.

[11] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[12] Laurent Jacob, Francis Bach, and Jean-Philippe Vert. Clustered Multi-Task Learning: A Convex Formulation. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS) 21*, pages 745–752. 2009.

[13] Yu Zhang and Dit-Yan Yeung. A Convex Formulation for Learning Task Relationships in Multi-Task Learning. In *Proceedings of the twenty-sixth conference on Uncertainty in Artificial Intelligence (UAI)*, pages 733–742, 2010.

[14] Zhuoliang Kang, Kristen Grauman, and Sha Fei. Learning with Whom to Share in Multi-task Feature Learning. In *Proceedings of the twenty-eighth International Conference on Machine Learning (ICML)*, 2011.

[15] Jiayu Zhou, Jianhui Chen, and Jieping Ye. Clustered Multi-Task Learning Via Alternating Structure Optimization. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS) 24*, pages 702–710. 2011.

[16] Abhishek Kumar and Hal Daumé III. Learning Task Grouping and Overlap in Multi-Task Learning. In *Proceedings of the twenty-ninth International Conference on Machine Learning (ICML)*, 2012.

[17] Sinno J. Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[18] Theodoros Evgeniou, Charles A. Micchelli, and Massimiliano Pontil. Learning Multiple Tasks with Kernel Methods. *The Journal of Machine Learning Research*, 6:615–637, 2005.

[19] Jun Liu, Shuiwang Ji, and Jieping Ye. Multi-Task Feature Learning Via Efficient $\ell_{2,1}$-Norm Minimization. In *Proceedings of the twenty-fifth conference on Uncertainty in Artificial Intelligence (UAI)*, pages 339–348. AUAI Press, 2009.

[20] Shibin Parameswaran and Kilian Weinberger. Large Margin Multi-Task Metric Learning. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS) 23*, pages 1867–1875. 2010.

[21] Kai Yu, Volker Tresp, and Anton Schwaighofer. Learning Gaussian Processes from Multiple Tasks. In *Proceedings of the twenty-second International Conference on Machine Learning (ICML)*, pages 1012–1019. ACM, 2005.

[22] Su-In Lee, Vassil Chatalbashev, David Vickrey, and Daphne Koller. Learning a Meta-Level Prior for Feature Relevance from Multiple Related Tasks. In *Proceedings of the twenty-fourth International Conference on Machine Learning (ICML)*, pages 489–496. ACM, 2007.

[23] Hal Daumé III. Bayesian Multitask Learning with Latent Hierarchies. In *Proceedings of the twenty-fifth conference on Uncertainty in Artificial Intelligence (UAI)*, pages 135–142. AUAI Press, 2009.

[24] Rie K. Ando and Tong Zhang. A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. *The Journal of Machine Learning Research*, 6:1817–1853, 2005.

[25] Michael T. Rosenstein, Zvika Marx, Leslie P. Kaelbling, and Thomas G. Dietterich. To Transfer or Not To Transfer. In *NIPS Workshop on Inductive Transfer*, 2005.

[26] Sebastian Thrun and J. O'Sullivan. Clustering Learning Tasks and the Selective Cross-Task Transfer of Knowledge. In *Learning to Learn*, pages 181–209. Kluwer Academic Publishers, 1998.

[27] Andreas Argyriou, Andreas Maurer, and Massimiliano Pontil. An Algorithm for Transfer Learning in a Heterogeneous Environment. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5211 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2008.

[28] Ali Jalali, Pradeep Ravikumar, Sujay Sanghavi, and Chao Ruan. A Dirty Model for Multi-task Learning. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Ze-

mel, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS) 23*, pages 964–972. 2010.

[29] Shipeng Yu, Volker Tresp, and Kai Yu. Robust Multi-Task Learning with t-Processes. In *Proceedings of the twenty-fourth International Conference on Machine Learning (ICML)*, pages 1103–1110. ACM, 2007.

[30] Jianhui Chen, Jiayu Zhou, and Jieping Ye. Integrating Low-Rank and Group-Sparse Structures for Robust Multi-Task Learning. In *Proceedings of the seventeenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 42–50. ACM, 2011.

[31] Carl Rasmussen. Gaussian Processes in Machine Learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 63–71. Springer, 2004.

[32] Edwin Bonilla, Kian Ming Chai, and Chris Williams. Multi-task Gaussian Process Prediction. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems (NIPS) 20*, pages 153–160. MIT Press, 2008.

[33] Yu Zhang and Dit-Yan Yeung. Multi-Task Learning using Generalized t Process. In *Proceeding of the thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 964–971, 2010.

[34] Tadej Janež, Jure Žabkar, Martin Možina, and Ivan Bratko. Learning faster by discovering and exploiting object similarities. *International Journal of Advanced Robotic Systems*, 2013.

[35] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, chapter 14.3.12, pages 520–528. Springer series in statistics. Springer, 2009.

[36] Ronald R. Hocking. A Biometrics Invited Paper. The Analysis and Selection of Variables in Linear Regression. *Biometrics*, 32(1):1–49, 1976.

[37] Isabelle Guyon and André Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[38] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, chapter 7.10, pages 241–248. Springer series in statistics. Springer, 2009.

[39] Kandethody M. Ramachandran and Chris P. Tsokos. *Mathematical Statistics with Applications*, chapter 6.3, pages 310–315. Academic Press, 2009.

[40] Gerald J. Hahn and William Q. Meeker. *Statistical Intervals: A Guide for Practitioners*, chapter 2.7, pages 39–40. Wiley Series in Probability and Statistics. John Wiley & Sons, Ltd., 2011.

[41] Geoff Cumming. Inference by eye: Reading the overlap of independent confidence intervals. *Statistics in Medicine*, 28(2):205–220, 2009.

[42] David Afshartous and Richard A. Preston. Confidence intervals for dependent data: Equating non-overlap with statistical significance. *Computational Statistics & Data Analysis*, 54(10):2296–2305, 2010.

[43] Harvey Goldstein and Michael J.R. Healy. The Graphical Presentation of a Collection of Means. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 158(1):175–177, 1995.

[44] PyMTL (Python library for Multi-task learning). A Python module implementing a Multi-task learning framework built on top of scikit-learn, SciPy, NumPy and matplotlib. URL https://github.com/tjanez/PyMTL. [Online; accessed 12-December-2013].

[45] Ken Sadohara. Learning of Boolean Functions Using Support Vector Machines. In Naoki Abe, Roni Khardon, and Thomas Zeugmann, editors, *Algorithmic Learning Theory*, volume 2225 of *Lecture Notes in Computer Science*, pages 106–118. Springer, 2001.

[46] Pedro Domingos and Michael Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(2-3):103–130, 1997.

[47] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*, 14:2349–2353, 2013.

[49] Jonathan J. Hull. A Database for Handwritten Text Recognition Research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[50] Zhuoliang Kang, Kristen Grauman, and Sha Fei. Supplementary material for paper Learning with Whom to Share in Multi-task Feature Learning, 2011. URL http://www-scf.usc.edu/~zkang/research.html. [Online; accessed 12-December-2013].

[51] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

[52] Alexandre Passos, Piyush Rai, Jacques Wainer, and Hal Daumé III. Flexible Modeling of Latent Task Structures in Multitask Learning. In *Proceedings of the twenty-ninth International Conference on Machine Learning (ICML)*, 2012.

[53] Desmond L. Nuttall, Harvey Goldstein, Robert Prosser, and Jon Rasbash. Differential school effectiveness. *International Journal of Educational Research*, 13(7): 769–776, 1989.

[54] Harvey Goldstein. Multilevel Modelling of Survey Data. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 40(2):235–244, 1991.

[55] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Supplementary material for paper Convex multi-task feature learning, 2008. URL http://ttic.uchicago.edu/~argyriou/code. [Online; accessed 12-December-2013].

[56] Peter J. Lenk, Wayne S. DeSarbo, Paul E. Green, and Martin R. Young. Hierarchical Bayes Conjoint Analysis: Recovery of Partworth Heterogeneity from Reduced Experimental Designs. *Marketing Science*, 15(2):173–191, 1996.

[57] pmtkdata. A collection of MATLAB data sets used by the PMTK project. URL https://code.google.com/p/pmtkdata. [Online; accessed 12-December-2013].

[58] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning series. Mit Press, 2012.

[59] Olivier Michel. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.

[60] Bojan Cestnik, Igor Kononenko, and Ivan Bratko. ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users. In Ivan Bratko and Nada Lavrač, editors, *Progress in Machine Learning: Proceedings of EWSL 87 / 2nd European Working Session on Learning*, pages 31–45. Sigma Press, 1987.

[61] John Mingers. An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning*, 3:319–342, 1989.

[62] J. Kent Martin. An Exact Probability Metric for Decision Tree Splitting and Stopping. *Machine Learning*, 28(2–3):257–291, 1997.

[63] E. B. Hunt, J. Martin, and P. Stone. *Experiments in Induction*. Academic Press, 1966.

[64] Leo Breiman. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall, 1984.

[65] Marko Robnik-Šikonja and Igor Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, 53(1-2):23–69, 2003.