UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jernej Virag

Avtomatična kategorizacija predavanj

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor:

izr. prof. dr. Janez Demšar

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01844/2012
Datum: 02.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JERNEJ VIRAG**

Naslov: **AVTOMATIČNA KATEGORIZACIJA PREDAVANJ**
AUTOMATED CATEGORIZATION OF VIDEO LECTURES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V zadnjih letih je na spletu vse več mest, ki uporabnikom ponujajo ogled video posnetkov. Medtem ko se na splošnejših straneh praviloma uporablja iskanje glede na besede v naslovu in opisu, je za strani s posnetki predavanj primernejše razvrščanje v hierarhijo in označevanje (tagging). Vzdrževanje takšnega sistema pa lahko zahteva veliko časa urednikov.

Razvijte sistem, ki bo na podlagi podatkov o video posnetkih predavanj urednikom predlagal ustrezno uvrstitev predavanj. Sistem naj temelji na metodah strojnega učenja, predvsem klasifikacije dokumentov. Razviti sistem preskusite z vidika njegove točnosti in porabe sistemskih virov.

Mentor:

prof. dr. Janez Demšar

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Jernej Virag

z vpisno številko 63060311

sem avtor diplomskega dela z naslovom:

Avtomatična kategorizacija predavanj

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom izr. prof. dr. Janeza Demšarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 9. 10. 2012 Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se svojemu mentorju, izr. prof. dr. Janezu Demšarju, za ideje ter čas pri izdelavi diplomske ter resne odgovore pri vprašanjih dvomljive umnosti. Posebej se zahvaljujem tudi sodelavcem v podjetju Viidea: Petru Kešetu, Juretu Čuhalevu, Goranu Kodrunu in Nejcu Jelovčanu. Posebno zahvalo si zasluži tudi Miha Grčar za vso pomoč pri začetku dela in knjižnice, ki so mi olajšale delo ter ekipa VideoLectures, ki je zgradila portal vreden nadgradnje.

Te poti pa nikakor ne bi mogel niti začeti brez izdatne podpore družine in Ivane Novak. Hvala za vse ure potrpljenja in priganjanja.

Vsem, ki so jim meje njihovega znanja vedno pretesne.

Vsebina

1 Povzetek

Diplomsko delo opisuje implementacijo storitve za klasifikacijo video predavanj z uporabo metod strojnega učenja. Namen storitve je avtomatična kategorizacija predavanj na Viidea platformi, ki poganja spletno stran VideoLectures.net.

Delo opisuje realizacijo storitve v programskem jeziku C# s pomočjo knjižnice Latino. Storitev omogoča izbiro enega izmed štirih vrst klasifikatorjev, ki so opisani v tem delu: naivni Bayesov klasifikator, klasifikator z iskanjem *k* najbližjih sosedov, klasifikator z iskanjem največje entropije in hierarhični klasifikatorja.

Celotna implementacija storitve je sestavljena iz štirih komponent: podatkovnega modela s podatki o predavanjih, modula za kategorizacijo predavanj, modula za predloge podobnih predavanj in spletnega vmesnika. Storitev s svetom komunicira preko protokola HTTP in teče v okoljih .NET CLR in Mono.

Na koncu dela so dodani rezultati testiranja kvalitete klasifikatorjev po dveh različnih merah.

2 Abstract

This bachelor thesis presents service implementation for categorization of video lectures using machine learning methods. The purpose of the service is to automatically categorize lectures on Viidea platform powering VideoLectures.net web portal.

This work describes realization of a web service written in C# programming language using Latino machine learning library. The service implements four types of classification algorithms: naive Bayes classifier, k-nearest neighbour classifier, maximum entropy classifier and hierarchical classifier. All classifiers are described in this work.

Service implementation consists of four components: data model with lecture data, categorization module, lecture recommendations module and web service interface. The service interface uses HTTP protocol and the service itself runs in .NET CLR and Mono environments.

The quality of classifiers is evaluated at the end of this work using two distinct methods.

3 Uvod

Informacijska doba je v eksploziji razširitve hitrih komunikacij in preprostih načinov objave vsebine prinesel tudi probleme z urejanjem in iskanjem informacij. Količina dosegljivih informacij na spletu je dosegla takšno raven, da postaja iskanje relevantnih informacij vedno težje in si moramo pri tem pomagati z vedno bolj dovršenimi algoritmi.

S takimi problemi se tudi soočajo upravitelji izobraževalnih strani z video predavanji. VideoLectures.net, ki temelji na platformi podjetja Viidea, ima natanko take težave: trenutno objavljajo več kot 15.000 pol- do enournih predavanj. Pri taki količini vsebin obiskovalec strani težko najde relevantno predavanje, zato upravitelji strani urejajo vsebine v kategorije. Trenutno obstaja preko 300 kategorij in podkategorij, kar urednikom strani dela precejšnje težave, saj si vsak urednik težko zapomni vse obstoječe kategorije in doda novo predavanje v pravo množico. Posledično se dogaja, da obiskovalci kakšnega predavanja ne najdejo, ker ni bilo dodano v ustrezne kategorije.

Z večanjem števila vsebin in širjenjem ekipe urednikov se bodo take težave samo stopnjevale. Tako sem v pričujoči diplomski nalogi implementiral avtomatični kategorizator, ki bo urednikom spletne strani VideoLectures.net ter ostalih spletnih strani temelječih na platformi Viidea, avtomatično predlagal relevantne kategorije za vsako novo predavanje ter opozoril na morebitne manjkajoče kategorije v starih predavanjih.

Končni cilj dela je implementacija storitve, ki preko protokola HTTP izpostavlja vmesnik do več klasifikatorjev za kategorizacijo predavanj in teče neodvisno od same platforme spletne strani. Storitev mora biti robustna, porabiti čimmanj sistemskih sredstev in teči mora vsaj na operacijskem sistemu Linux.

4 Teoretična podlaga klasifikacije dokumentov

V tem poglavju si bomo ogledali teoretično ozadje klasifikacije besedil – najprej obdelavo dokumentov in njihovo predstavitev v vektorski obliki, zatem pa si bomo ogledali nekaj klasifikacijskih algoritmov.

Dokument je v terminologiji besedilnega rudarjenja samostojno besedilo, ki ga iščemo ali mu pripisujemo lastnosti. V primeru implementacije v naslednjem poglavju bo en dokument besedilo, sestavljeno iz: naslova predavanja, imena avtorja, naslova dogodka ali konference, naslovov prosojnic in opisa predavanja.

Množici vseh dokumentov, ki predstavljajo vhod v algoritem in s tem podlago za izračun modela, pravimo *korpus*.

4.1 Predprocesiranje dokumentov

Besedila, pripravljena za človeško branje, pogosto niso primerna za računalniško obdelavo. V izvorni obliki jih je težko predstavljati na matematični način, obenem pa algoritmom delajo težave jezikovne posebnosti kot so skloni, števila, nakloni in drugi slovnični konstrukti. Besedila so tipično shranjena kot zaporedje številk v neki datoteki ali podatkovni bazi. Najprej to zaporedje številk predstaviti v algoritmom uporabni obliki. Cilj predprocesiranja je ločitev besedila na posamezne besede brez pomensko ekvivalentnih dvojnikov, iz katerih lahko potem zgradimo slovar besed posameznega dokumenta in slovar vseh besed v korpusu.

Predprocesiranje je tipično sestavljeno iz naslednjih stopenj:

1. Tokenizacija
2. Izločitev prepovedanih besed
3. Normalizacija
4. Krnjenje ali lematizacija

4.1.1 Tokenizacija

Tokenizacija je postopek razbitja in filtriranja zaporedja znakov, ki določeno zaporedje znakov razreže na posamezne kose, ki jih imenujemo *token*i. *Token* je zaporedje znakov dokumenta, ki so združeni kot enota za nadaljnje semantično procesiranje. V večini primerov en *token* ustreza eni besedi, vendar bomo v nadaljevanju videli, da to ni vedno nujno.

Tokenizacija poteka tako, da beremo vhodni tok znakov in vsako zaporedje znakov med praznim prostorom označimo kot en *token*. Praznega prostora v posamezne tokene ne vključimo.

Vhod: Urad US-CERT je izjavil, da je »MD5 algoritem zlomljen in neprimeren za nadaljnjo uporabo«.

Izhod: Urad US-CERT je izjavil da je »MD5 algoritem zlomljen in neprimeren za nadaljnjo uporabo«.

Primer 1: Tokenizacija stavka, posamezni *token*i so podčrtani

4.1.2 Izločitev prepovedanih besed

V besedilih večinoma nastopa veliko besed, ki ne nosijo bistvene informacije o vsebini. V slovenščini so to »in«, »v«, »na«, »za«, »po«, ipd. Ker se pojavljajo zelo pogosto, opazno povečajo dimenzije vektorjev za opis besedila.

V tej fazi zato izločimo *token*e, ki predstavljajo tovrstne besede in tako zmanjšamo dimenzionalnost vektorskega prostora in s tem občutno skrčimo potreben pomnilnik ter čas računanja modela in klasifikacije.

Vhod: Urad US-CERT ~~je~~ izjavil ~~da~~ ~~je~~ »MD5 algoritem zlomljen ~~in~~ neprimeren ~~za~~ nadaljnjo uporabo«.

Izhod: Urad US-CERT izjavil »MD5 algoritem zlomljen neprimeren nadaljnjo uporabo«.

Primer 2: Izločitev prepovedanih besed

4.1.3 Normalizacija

Normalizacija je postopek pretvorbe *token*ov v kanonično obliko, kjer odstranimo ločila, velike črke in ostale površinske razlike. Normalizacija tipično vključuje:

1. **Spremembo velikih črk v male**
   S tem poskrbimo, da bodo besede, ki se razlikujejo samo v velikih ali malih črkah upoštevane kot iste pri gradnji slovarja. Pri tem lahko uporabimo slovar imen, ki bo poskrbel, da bodo lastna imena ostala ločena od besed z istim pomenom (npr. »Windows« proti »windows«).

2. **Odstranjevanje naglasov**
   Pogosto se dogaja, da uporaba naglasnih znamenj, preglasov in podobnih jezikovnih konstruktov v tekstih ni konsistentna, zato pri normalizaciji zamenjamo znake z nenaglašenimi različicami.

3. **Odstranjevanje ločil**
   Na zadnji stopnji s posameznih tokenov odstranimo še ločila in vse ostale ne-alfanumerične znake.

Kot že omenjeno, moramo pri normalizaciji biti previdni pri posebnih primerih, saj lahko hitro odstranimo preveč informacije. To je še posebej pomembno pri spreminjanju velikih črk v male, kjer si želimo ohraniti lastna imena ločena od istih besed (podjetje »Apple« proti jabolko »apple«, program »Birokrat« proti osebi »birokrat« ipd.). Podobna težava se zgodi tudi pri odstranjevanju ločil, saj so pogosto ločila del imena – v primeru, da ne upoštevamo izjem, bi vsa tri imena »C«, »C++« in »C#« vsa bila okrajšana na »C«, s tem pa bi izgubili pomembno informacijo o tematiki. Zato na tej fazi pogosto uporabimo slovar lastnih imen ali klasifikacijo imenskih entitet[1] , ki povesta, ali moramo *token*e obdržati v izvorni obliki.

Vhod: Urad US-CERT izjavil »MD5 algoritem zlomljen neprimeren nadaljnjo uporabo«.

Izhod: urad uscert izjavil md5 algoritem zlomljen neprimeren nadaljnjo uporabo

Primer 3: Normalizacija *token*ov

4.1.4 Krnjenje ali lematizacija

Praktično vsi svetovni jeziki poznajo koncept spreminjanja pomensko istih besed v stavku glede na kontekst – najsi je to sklanjanje, spreganje, sprememba naklona, časa ali števila. Pri sestavljanju slovarja besed nam to povzroča težave, saj pomensko iste besede primerjalni algoritem zazna kot drugačne, če so zapisane v drugem času, številu, naklonu …

To težavo rešimo s krnjenjem ali lematizacijo. Pri *krnjenju* odrežemo samo končnice besed in s tem poskušamo dobiti njihove korene, medtem ko pri *lematizaciji* uporabimo algoritem, ki poleg rezanja končnic zna tudi spreminjati besede v njihov koren po načelih besedilne in morfološke analize jezika. Krnjenje bi tako besedo »bil« spremenilo v »b«, lematizacija pa v »je«. Če je le mogoče, zato uporabimo lematizacijo, krnjenje pa se uporablja samo v primeru, če nimamo dostopa do ustreznih algoritmov za jezik besedila.

Vsak jezik potrebuje lasten algoritem za krnjenje ali lematizacijo, torej za vsak jezik na tej stopnji rabimo še informacijo o jeziku dokumenta. Tipično je ta informacija podana vnaprej v obliki metapodatka, uporabimo pa lahko tudi algoritme za detekcijo jezika.

---

[1] *ang. named entity extraction*

Za angleščino je empirično najučinkovitejši *Porter stemmer* [1] avtorja Martina Porterja, za slovenščino in ostale južnoslovanske jezike pa se solidno obnese LemmaGen [2] avtorjev z Inštituta Jožefa Štefana.

Vhod: urad uscert izjavil md5 algoritem zlomljen neprimeren nadaljnjo uporabo

Izhod: urad uscert izjaviti md5 algoritem zlomljen neprimeren nadaljnji uporaba

Primer 4: Rezultat lematizacije slovenščine z LemmaGen

4.2 Vektorski model predstavitve dokumentov

Ker se samih besedil ne da praktično matematično manipulirati v izvorni obliki, dokumente zapišemo v obliki matematičnih vektorjev. V takem vektorju vsaka komponenta (dimenzija) predstavlja eno besedo v besedišču celotnega korpusa. Če so besedila predprocesirana,vsaka komponenta predstavlja koren besede (*termin* – označen s t), vrednost pa število pojavitev tega termina v dokumentu. Ker vsak dokument vključuje samo majhno podmnožico vseh terminov korpusa, je večina vrednosti komponent takih vektorjev enakih 0.

V matematični notaciji označimo dokumente z d, pripadajoče vektorje $\vec{d}$. Postopek za izgradnjo vektorjev poteka v naslednjem vrstnem redu:

1. Izgradnja slovarja vseh terminov t – po predprocesiranju so to koreni besed in samostoječa imena.
2. Vsakemu terminu v slovarju se dodeli indeks i.
3. Za vsak dokument d se ustvari vektor $\vec{d}$

$$\vec{d} = (f_{t_0}, \dots, f_{t_i}, \dots),$$

kjer f_{t_i} predstavlja število pojavitev (frekvenco) termina t z indeksom i v slovarju.

V tem modelu vrstni red besed ni pomemben. Frazi »*jaz grem tja*« in »*grem tja jaz*« ustvarita enak vektor. S tem izgubimo pomemben del informacije, saj tak model ne upošteva pogostih fraz in imen (npr. »Predsednik Republike na obisku v Franciji« je v tem modelu ekvivalent »Predsednik Francije na obisku v Republiki Sloveniji«, čeprav je pomen drugačen). To pomanjkljivost lahko na tem mestu popravimo z generiranjem *n-gramov*.

n-grami so skupki terminov, ki jih zgeneriramo med ustvarjanjem vektorja s kombiniranjem vseh možnih kombinacij n terminov in jih zatem upoštevamo kot ločene termine. V praksi največkrat uporabimo 2- in 3-grame, ker večina fraz in imen ni daljših od treh besed, generiranje večjih kombinacij pa nam močno poveča

dimenzionalnost prostora in s tem pomnilniško in računsko zahtevnost. Generiranje 3-gramov na kratki frazi je prikazano v primeru 5.

Vhod: urad uscert izjaviti

Izhod: urad urad uscert urad uscert izjaviti uscert uscert izjaviti izjaviti

Primer 5: Generiranje 3-gramov besedila

S takim postopkom dobimo množico vektorjev korpusa D z vektorji vseh N dokumentov:

$$D = \{d_1, \dots, d_N\}$$

Na vektorjih dokumentov lahko sedaj izvajamo algebraične operacije – računanje razdalje, skalarnega produkta, iskanje centroidov, ipd. Te operacije so pomemben del klasifikacijskih algoritmov in ta oblika (popravljena s TF/IDF faktorji z naslednjega podpoglavja) bo izhodišče gradnje modela vseh klasifikatorjev.

4.2.1 Obtežitev vektorjev

Vektorji imajo v obliki, razloženi zgoraj, kritično napako: niso normalizirani in zato občutljivi na dolžino besedila. Pri iskanju podobnosti z evklidsko razdaljo ali skalarnim produktom to občutno vpliva na rezultate, saj na razdaljo med dokumenti poleg vsebine vpliva tudi dolžina teh dokumentov. Iz tega razloga tudi ne smemo izvesti preproste normalizacije v obliki

$$f_{t_i} = \frac{f_{t_i}}{\sum_{j \in df} f_{t_j}}, \tag{4.1}$$

saj bo distribucija vrednosti še vedno preveč odvisna od dolžine dokumenta in števila pojavitev fraz.

Komponente vektorjev zato za rešitev tega problema obtežimo z *inverzno frekvenco* (*inverse document frequency* - IDF). Inverzno frekvenco termina označimo z idf_t in jo izračunamo kot

$$\mathrm{idf}_t = \log \frac{N}{df_t}, \tag{4.2}$$

kjer je N število vseh dokumentov v korpusu, df_t pa je število vseh dokumentov, ki vključujejo vsaj eno pojavitev termina t.

IDF faktor za termin je visok, če se le-ta pojavi v malo dokumentih in nizek, če se pojavlja pogosto. To ustreza intuitivni predstavi, da besede, ki se pojavljajo samo v

določeni skupini dokumentov, povejo več o vsebini dokumenta kot tiste, ki se pojavljajo pogosto (to so tipično vezniki in funkcijske besede).

Uporabi faktorja IDF pri obteževanju vektorjev pravimo TF/IDF obteževanje (*tf-idf weighting*), *i*-to komponento vektorja pa izračunamo po enačbi

$$d_i = \text{tf–idf}_{t_i,d} = \text{tf}_{t_i,d} \cdot \text{idf}_{t_i} . \qquad (4.3)$$

Po obteževanju TF/IDF imajo najvišjo vrednost v vektorju tisti termini, ki se pogosto pojavljajo v tem dokumentu in zelo redko v drugih dokumentih. Taki termini pa obenem nosijo največ informacije o tematiki in vsebini dokumenta. Dolžina vektorja je še vedno odvisna od dolžine dokumenta, kar moramo reševati pri računanju klasifikacijskega modela.

4.2.2 Algebraične operacije nad dokumentnimi vektorji

Obravnava v vektorskem prostoru nam prinese možnost uporabe vektorskih operacij s področja linearne algebre in nam s tem odpre nov svet manipuliranja z besedili. Najpomembnejša operacija je gotovo računanje razdalje, saj nam po TF/IDF pretvorbi dokumentov razdalja med vektorji direktno pokaže podobnost besednjakov besedil. V nadaljevanju si bomo ogledali obteženo združevanje vektorjev ter dva splošno razširjena načina računanja razdalje med vektorji: evklidsko razdaljo in skalarni produkt.

Obteženo združevanje vektorjev

Dokumentne vektorje lahko preprosto združimo (*concatenate*) tako, da ustvarimo nov vektor, v katerem zložimo komponente vhodnih vektorjev. Pri tem moramo biti previdni, da v vseh dokumentih vektorje zložimo v istem vrstnem redu, saj tako isti indeks predstavlja isti termin v vseh dokumentih. Pri tem lahko posamezne vektorje pomnožimo z različnimi utežmi.

Vhod:

$$\vec{f} = \{ f_0, f_1, \dots f_n \}, \text{utež } w_f$$
$$\vec{d} = \{ d_0, d_1, \dots d_m \}, \text{utež } w_d$$

Izhod:

$$\vec{f} \parallel \vec{g} = \{ w_f \cdot f_0, w_f \cdot f_1, \dots, w_f \cdot f_n, w_d \cdot d_0, w_d \cdot d_1, \dots, w_d \cdot d_m \}$$

Primer 6: Združevanje vektorjev

Ta operacija se uporablja predvsem v primerih, ko imamo v dokumentu različne tipe besedil (npr. naslov, telo, avtor) in hočemo pri gradnji dokumentnega vektorja določenemu tipu dati prednost.

Evklidska razdalja

Evklidska razdalja (tudi: Evklidova ali druga norma) [3] je najpogostejša »standardna« mera razdalje med dvema točkama.

$$d(\vec{d},\vec{e}) = \sqrt{(d_1 - e_1)^2 + (d_2 - e_2)^2 + \cdots + (d_n - e_n)^2} \tag{4.4}$$

V praksi se poskušamo izogniti računanju kvadratnega korena, ki je draga operacija. Ker nas zanima samo razmerje med razdaljami in ne točna vrednost, si to lahko privoščimo.

Evklidska razdalja upošteva tudi dolžino vektorjev in posledično dolžino besedila, kar pa pri določanju vsebinske podobnosti ni zaželeno. Zaradi te pomanjkljivosti se namesto evklidske razdalje poslužimo skalarnega produkta.

Kosinusna razdalja

Razdalja med pomenom vsebine besedil se izraža kot kot med dvema vektorjema, ki je del skalarnega produkta [4].

Skalarni produkt je definiran kot

$$\vec{d} \cdot \vec{e} = |d||e| \cos\theta \tag{4.5}$$

in iz te enačbe lahko izrazimo kosinus kota. Tako dobimo enačbo

$$\cos\theta = \frac{\vec{d} \cdot \vec{e}}{|\vec{d}||\vec{e}|}. \tag{4.6}$$

Vrednosti $\cos\theta$ pravimo **kosinusna razdalja.** Dolžino vektorja $\vec{d}$ izračunamo kot

$$|\vec{d}| = \sqrt{d_1^2 + d_2^2 + \cdots + d_n^2} \tag{4.7}$$

Sam skalarni produkt dveh vektorjev se pa izračuna kot

$$\vec{d} \cdot \vec{e} = d_1 e_1 + d_2 e_2 + \cdots + d_n e_n. \tag{4.8}$$

S pomočjo teh enačb lahko izračunamo podobnost dveh dokumentov tako, da izračunamo faktor $\cos\theta$. Kosinus je, kot korenjenje, zahtevna operacija, ker pa ravno tako ohranja razmerje, nam kota direktno ni treba izračunati. Tako lahko optimizirano podobnost dveh dokumentov izračunamo z izrazom

$$p(\vec{d},\vec{e}) = \frac{d_1 e_1 + d_2 e_2 + \cdots + d_n e_n}{(d_1^2 + d_2^2 + \cdots + d_n^2)(e_1^2 + e_2^2 + \cdots + e_n^2)} \tag{4.9}$$

Dolžine vektorjev lahko shranimo vnaprej in tako prihranimo še nekaj podvojenega računanja.

Te operacije bodo sestavljale pomemben del klasifikacijskih algoritmov, ki si jih bomo ogledali v naslednjem poglavju.

4.3 Kategorizacija dokumentov

Kategorizacija dokumentov je samo vrsta klasifikacije dokumentov z oznakami. Klasifikacijo opravljamo na podlagi *modela,* ki ga klasifikacijski algoritem zgradi in na podlagi katerega se določajo *oznake* (*labels*) dokumentov. Algoritme za učenje modelov in klasifikacijo v osnovi delimo glede način učenja v tri skupine:

- nadzorovano učenje,
- napol-nadzorovano učenje,
- nenadzorovano učenje.

Značilnost algoritmov za nadzorovano učenje je, da za izgradnjo modela potrebujejo *učno množico*, iz katere se »naučijo« klasifikacijski model. Učna množica je skupek dokumentov, iz katerega izračunamo model za klasifikacijo. Ta učna množica mora imeti oznake že točno določene. Algoritmi za nenadzorovano učenje take učne množice ne potrebujejo in so namenjeni obdelavi podatkov, kjer informacij nimamo podanih vnaprej.

Cilj tega dela je kategorizacija novih dokumentov glede na že obstoječe kategorije, kar pomeni, da so za nas zanimivi samo algoritmi s skupine nadzorovanega učenja in se bomo zato osredotočili samo na njih.

Pri klasifikacijami z oznakami sta pomembni še dve vprašanji:

1. Lahko en dokument nosi več kot eno oznako?
   Poznamo namreč primere, kjer se oznake med sabo izključujejo in ima lahko dokument samo eno oznako, v drugih primerih pa ima lahko en dokument več oznak.

2. V kakšnem medsebojnem odnosu so oznake?
   Oznake so med sabo lahko popolnoma neodvisne (npr. »značke« pri elektronski pošti) ali povezane v graf ali drevo.

Izbira in implementacija algoritmov ter modela je prav tako odvisna od odgovorov na zgornji dve vprašanji. V primeru tega dela so oznake urejene v hierarhijo in en dokument lahko pripada več oznakam.

V tem delu se bomo osredotočili predvsem na **nadzorovano učenje modelov**, kjer en dokument lahko pripada **več klasifikacijskim razredom** , ti pa so urejeni v **hierarhično strukturo**.

V nadaljevanju si bomo ogledali princip delovanja razširjenih klasifikacijskih algoritmov, ki so bili implementirani in preizkušeni.

Ogledali si bomo:

- naivni Bayesov klasifikator,
- klasifikacijo po principu *k* najbližjih sosedov,
- klasifikacijo z iskanjem največje entropije,
- klasifikacijo z odločitvenim drevesom.

V nadaljevanju bo množica kategorij (oz. oznak, razredov – *classes*) označena s $\mathbb{C}$, množica dokumentnih vektorjev z $\mathbb{X}$, *klasifikacijska funkcija*, ki bo dodelila dokumentom razred (v našem primeru točneje kategorijo), pa bo označena z γ.

$$\gamma : \mathbb{X} \rightarrow \mathbb{C}$$

4.3.1 Naivni Bayesov klasifikator

Prva metoda je multinomialna naivna Bayesova metoda [5], ki spada med verjetnostne učne metode in se kot edina med naštetimi ne zanaša na vektorsko obliko dokumentov. Verjetnost dokumenta d v razredu c se izračuna kot

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \qquad (4.10)$$

kjer je $P(t_k|c)$ pogojna verjetnost, da se termin t_k pojavi v dokumentu iz razreda c. $P(t_k|c)$ si lahko interpretiramo kot mero, ki pove, koliko termin t_k prispeva k dokazu, da dokument d pripada razredu c. V tekstovni klasifikaciji moramo poiskati **najboljši** razred za dokument d – v našem primeru bo najboljši razred (c_{best}) najverjetneje tisti, ki ima največjo posteriorno verjetnost

$$c_{best} = \underset{c \in \mathbb{C}}{\operatorname{argmax}} \hat{P}(c|d) = \underset{c \in \mathbb{C}}{\operatorname{argmax}} \left[\hat{P}(c) + \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c) \right] \qquad (4.11)$$

Ker nam dejanske verjetnosti niso znane, smo v enačbi uporabili verjetnosti ocenjene iz naše učne množice - $\hat{P}$. V enačbi 4.11 med seboj pomnožimo pogojne verjetnosti za vsak termin. Te pogojne verjetnosti so decimalna števila, manjša od 1, zato se nam lahko pri hitrem računanju na računalniku zgodi, da pride to t.i. napake podkoračitve[2], kjer se zaradi premajhne natančnosti računanja vrednosti porežejo na 0. S tega razloga pogosto raje uporabimo alternativno obliko, kjer seštevamo logaritme verjetnosti

[2] *ang. floating point underflow*

$$c_{best} = \underset{c \in \mathbb{C}}{\operatorname{argmax}} \left[\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c) \right]. \quad (4.12)$$

Ker lahko logaritemske vrednosti izračunamo vnaprej, seštevanja so pa na trenutnih procesorjih neprimerno hitrejša od množenj, je tak način izračuna tudi časovno manj potraten.

Verjetnost $\hat{P}(c)$ ocenimo kar s frekvenco

$$\hat{P}(c) = \frac{N_c}{N}, \quad (4.13)$$

kjer je N_c število dokumentov v razredu c in N število vseh dokumentov. Pogojno verjetnost pa ocenimo z

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}, \quad (4.14)$$

kjer je T_{ct} število pojavitev termina t v dokumentih učne množice v razredu c, V pa je besednjak celotne učne množice.

Težava z opisanim načinom ocenjevanja verjetnost je v tem, da v enačbi 4.12 množimo verjetnosti terminov med sabo. Posledično bomo v primeru, da kateri izmed terminov pri dokumentu za klasifikacijo manjka v dokumentih v razredu učne množice, dobili verjetnost 0. To je očitno narobe, saj se drugi termini lahko pojavljajo in vplivajo na klasifikacijo. To anomalijo popravimo z *Laplaceovim glajenjem,* kjer vsem razredom dodamo minimalno apriorno verjetnost

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)}. \quad (4.15)$$

Z opisanim algoritmom lahko dodelimo vsakemu dokumentu samo eno oznako. Za dodeljevanje več oznak (*multi-label classification*) ga modificiramo tako, da namesto iskanja razreda z največjo verjetnostjo določimo mejno vrednost verjetnosti in izberemo vse razrede, ki tisto mejno vrednost presežejo.

4.3.2 Klasifikacija s *k* najbližjimi sosedi in mehkim glasovanjem

Klasifikacija s *k* najbližjimi sosedi (v nadaljevanju: kNN[3] klasifikacija) [6] temelji na načinu, kjer dokumentu poiščemo *k* najbližjih dokumentov po razdalji v vektorskem prostoru in mu dodelimo večinski razred teh *k* dokumentov.

Če hočemo klasificirati dokument *d*, to izvedemo tako, da razvrstimo dokumente iz učne množice *V* po oddaljenosti. Za to lahko izberemo eno izmed mer iz prejšnjega

---

[3] *ang.: k-Nearest Neighbour*

poglavja – ponavadi je to kosinusna oddaljenost. Od teh izberemo k najbližjih elementov in dokumentu določimo večinski (najpogostejši) razred, ki mu teh k dokumentov pripada.

Tak pristop ni najbolj optimalen, saj je zelo občutljiv na izbiro spremenljivke k, obenem pa nastane težava pri odločanju v primeru, ko tekmuje večje število dokumentov na enaki razdalji od dokumenta d. Naivna rešitev razrešuje take primere z naključnim dodeljevanjem razredov, Mitchell in Schaefer [6] pa sta našla optimalnejšo rešitev z »mehkim« glasovanjem.

Matematično lahko standardni algoritem kNN definiramo s tremi matrikami: A, R in w. Recimo, da imamo N dokumentov v korpusu in M razredov.

Matrika A pove, kateri dokument iz učne množice pripada kateremu razredu – tako je velika $M \times N$ in vsaka vrstica predstavlja en razred, vsak stolpec pa dokument. Formalno:

$$A_{ij} = \begin{cases} 1, & \text{če } dokument\ d_i\ pripada\ razredu\ c_j \\ 0, & sicer \end{cases}$$

Matrika R je velika $N \times N$ in v vrsticah predstavlja dokumente, pri čemer ima j-ti stolpec vrednost 1, če je dokument v vrstici j-ti po oddaljenosti od dokumenta d, ki ga klasificiramo. Formalno:

$$R_{ij} = \begin{cases} 1, & dokument\ d_{\mathrm{i}}\ iz\ V\ je\ j-ti\ po\ oddaljenosti\ od\ d \\ 0, & sicer \end{cases}$$

Vektor w služi samo kot filter in ima k komponent z vrednostjo 1, ostale so enake 0. Razred, ki ga klasifikator določi dokumentu d, je tedaj

$$c_{best} = \mathrm{argmax}(w^T RA). \tag{4.16}$$

Metoda mehkega glasovanja premeni matriko R, tako da le-ta vključuje poljubno vrednost med 0 in 1. Novo matriko R zgeneriramo tako, da najprej za vsak dokument v učni množici določimo t.i. *fuzzy* faktor D_i

$$D_i(x) = \frac{N(|d_i - x|, \sigma_P)}{N(0, \sigma_P)}, \tag{4.17}$$

kjer je N normalna (Gaussova) porazdelitev, σ_P skalirni faktor, d_i pa oddaljenost i-tega dokumenta učne množice od dokumenta d, ki ga klasificiramo.

Zatem sestavimo matriko R

$$R_{ij} = S\big(D_i, D_{(j)}\big), \tag{4.18}$$

kjer je $D_{(j)}$ faktor, ki pripada j-ti največji razdalji, S pa katerakoli standardna mera podobnosti. Ko imamo matriko R, jo izmenično normaliziramo najprej po vrsticah in zatem po stolpcih dokler vrednosti ne konvergirajo.

Skalirni faktor σ_P izraža deviacijo razdalj od dokumenta d do dokumentov učne množice. Ponavadi izberemo standardno deviacijo razdalj k najbližjih dokumentov.

$$\sigma_P = \frac{\sum_{i=1}^{k}(d_i - \bar{d})^2}{K} \tag{4.19}$$

$$\bar{d} = \frac{\sum_{i=1}^{k} d_i}{K} \tag{4.20}$$

Tak način klasifikacije je robustnejši, še posebej v primerih, ko je vektorski prostor zelo redek (*sparse*) in nima težav z odločanjem pri podobnih dolžinah.

Opisani postopek je uporaben kvečjemu za klasifikacijo vsakega dokumenta v en razred. Za našo uporabo ga moramo torej prilagoditi tako, da namesto iskanja enega samega večinskega razreda k sosedov, poiščemo več razredov in izberemo najboljše kandidate, ki presegajo določeno vrednost glasov pri glasovanju.

4.3.3 Klasifikacija z iskanjem največje entropije

Klasifikacija z iskanjem največje entropije [7] temelji na gradnji modela, ki ima največjo možno entropijo (torej tudi najbolj enakomerno porazdelitev) glede na omejitve, določene iz učne množice. Algoritem začnemo tako, da predpostavimo popolnoma enakomerno porazdelitev verjetnosti pripadnosti dokumenta v razred, potem pa glede na dokumente učne množice popravljamo verjetnosti tako, da odražajo informacije iz učne množice.

Predpostavimo, da je naša učna množica V sestavljena iz N parov $\{(d_1, c_1), \ldots, (d_N, c_N)\}$, kjer so d dokumentni vektorji, c pa pripadajoči razredi. Ker imajo dokumenti več pripadajočih razredov, jih večkrat dodamo v množico.

Z učno množico določimo »empirično« distribucijo

$$\tilde{p}(d,c) \equiv \frac{1}{N} \times \check{s}tevilo\ parov\ (d,c), \tag{4.21}$$

izgraditi pa hočemo stohastični model za

$$p(c|d) = \frac{p(d,c)}{p(d)}. \tag{4.22}$$

Tu še definiramo koncept značilke

$$f_{w,c'}(d,c) = \begin{cases} 0, & \check{c}e\ c \neq c' \\ d_w & sicer \end{cases},$$

kjer je d_w komponenta vektorja d, ki pripada terminu w. Pričakovana vrednost značilke glede na empirično distribucijo je

$$\tilde{p}(f) = \sum_{d,c} \tilde{p}(d,c) \cdot f(d,c). \tag{4.23}$$

Podobno je pričakovana vrednost značilke glede na model p

$$p(f) \equiv \sum_{d,c} \tilde{p}(d)p(c|d) \cdot f(d,c). \tag{4.24}$$

Ker mora model ustrezati empirični distribuciji, velja

$$p(f) = \tilde{p}(f) \tag{4.25}$$

in posledično mora veljati za vsako značilko f

$$\sum_{d,c} \tilde{p}(d)p(c|d) \cdot f(d,c) = \sum_{d,c} \tilde{p}(d,c) \cdot f(d,c). \tag{4.26}$$

Danim omejitvam ustreza neskončno število modelov. Po načelu, omenjenem v prvem odstavku, bomo izbrali tistega, ki ima najenakomernejšo verjetnostno porazdelitev. To bomo izrazili s pogojno entropijo $p(c|d)$

$$H(p) \equiv -\sum_{d,c} \tilde{p}(d)p(c|d) \log p(c|d). \tag{4.27}$$

Tako moramo izbrati model p^*, ki maksimira vrednost $H(p)$. p^* izrazimo v eksponentni obliki

$$p(c|d) = \frac{1}{Z(d)} \exp\left(\sum_i \lambda_i f_i(d,c)\right). \tag{4.28}$$

λ_i so uteži, ki jih moramo oceniti, $Z(d)$ pa je normalizacijska konstanta

$$Z(d) = \sum_c \exp\left(\sum_i \lambda_i f_i(d,c)\right). \tag{4.29}$$

Preostane nam samo še izračun uteži λ_i, ki ga izvedemo z algoritmom *izboljšanega iterativnega skaliranja*[4]. Algoritem lahko v kratkem opišemo v treh korakih:

1. Začni z $\lambda_i = 0$ za vse $i \in \{1,2,\dots,n\}$
2. Za vsak $i \in \{1,2,\dots,n\}$
   a. Naj bo $\Delta\lambda_i$ rešitev za

$$\left[\sum_{d,c} \tilde{p}(d)p(c|d)f_i(d,c)\right] \exp(\Delta\lambda_i \sum_i f_i(d,c)) = \tilde{p}(f_i) \qquad (4.30)$$

   b. $\lambda_i = \lambda_i + \Delta\lambda_i$
3. Vrni se na korak 2. dokler niso vse vrednosti λ_i skonvergirale

S tem postopkom smo izgradili model, ki ima vse potrebno, za klasificiranje dokumentov s pomočjo enačbe 4.22.

Pri klasifikaciji izračunamo verjetnost *p(c|d)* za vsak razred *c* in predlagamo tiste razrede, ki presegajo določeno mejo verjetnosti.

4.3.4 Hierarhična klasifikacija tekstovnih dokumentov

Hierarhični klasifikator [8] izkorišča dejstvo, da so razredi razporejeni v drevo. Za vsako vozlišče drevesa bomo izračunali dva binarna klasifikatorja:

1.) lokalnega, ki bo določil ali dokument pripada razredu vozlišča,
2.) pod-drevesnega, ki bo določil, ali se bo nadaljevalo preverjanje pripadnosti še v naslednikih vozlišča.

Na tem mestu najprej definiramo koncept *Pokritja*. Pokritje razreda c_i je množica razredov, ki pripadajo poddrevesu s korenom v c_i vključno s samim c_i. Za katerikoli dokument d_j velja $d_j \in c_i$, če pripada razredu c_i; $d_j \in Pokritje(c_i)$ pa velja, če in samo če d_j pripada kateremukoli razredu iz $Pokritje(c_i)$. Definiramo še $Starš(c_i)$, ki pomeni starša razreda c_i v drevesu razredov.

Na osnovi teh predpostavk definiramo klasifikatorje v vsakem vozlišču drevesa razredov. Ker gradimo binarne klasifikatorje, moramo definirati *pozitivne* (+) in *negativne*(-) primere dokumentov iz učne množice *V*. Glede na vrsto vozlišča definiramo klasifikatorje na štiri načine:

- Pod-drevesni klasifikator v korenu drevesa razredov c_{koren}
  - Pozitivni – vsi $d_j \in Pokritje(c_{koren})$
  - Negativni – enako število primerov kot pozitivnih, tako da $d_j \notin Pokritje(c_{koren})$

[4] *ang.: improved iterative scaling*

- Pod-drevesni klasifikator za notranje vozlišče c_i
  - Pozitivni – vsi $d_j \in Pokritje(c_i)$
  - Negativni – vsi $d_j \notin Pokritje(c_i) \wedge d_j \in Pokritje(Starš(c_i))$
- Lokalni klasifikator za notranje vozlišče c_i
  - Pozitivni – vsi $d_j \in c_i$
  - Negativni – vsi $d_j \notin c_i \wedge d_j \in Pokritje(c_i)$
- Lokalni klasifikator za razred c_i v listu drevesa
  - Pozitivni – vsi $d_j \in c_i$
  - Negativni – vsi $d_j \notin c_i \wedge d_j \in Pokritje(Starš(c_i))$

V tem delu bomo za klasifikacijski algoritem v vozliščih izbrali *Support Vector Machines* [9], ki so se za klasifikacijo besedil izkazali kot hitri in učinkoviti [10].

4.4 Ocenjevanje učinkovitosti kategorizacije

V prejšnjem poglavju smo videli, da lahko dokumente kategoriziramo z množico klasifikacijskih algoritmov, ki so različno kvalitetni. Da se bomo lahko odločili za najboljšega, moramo najprej definirati mero, po kateri bomo lahko izmerili kvaliteto rezultatov klasifikacije. V nadaljevanju si bomo ogledali dve meri, s katerima bomo v praktični implementaciji ocenili kvaliteto rezultatov kategorizacije.

Obe meri temeljita na k-kratnem prečnem preverjanju[5], kjer učno množico razdelimo na k delov. Nato k-krat ponovimo postopek učenja in testiranja, pri čemer vsakič uporabimo $(k\text{-}1)$ delov za učenje modela, en del pa uporabimo kot vhod v klasifikatorje in z njim ocenimo pravilnost klasifikacij. Najpogosteje se uporablja 10-kratno prečno preverjanje.

Mero bomo izbrali glede na namen implementacije. V praktični implementaciji se bo rezultat kategorizacije pokazal uporabniku pri nalaganju videa predavanja in uporabnik bo moral ustrezne kategorije ročno potrditi. Posledično moramo zasnovati mere tako, da bodo dobro ocenile klasifikator, ki bo v omejeni množici rezultatov vrnil vse ustrezne kategorije, ne bodo pa kaznovale vračanja preveč napačnih kategorij v tej množici.

4.4.1 Ocenjevanje s klasifikacijsko točnostjo

Pri prvi izmed mer bomo pri preizkušanju določili delež vrnjenih pravilnih kategorij. Predpostavimo, da imamo dokumente d_i iz testne množice T, ki imajo določeno množico pravih kategorij C_T in množico kategorij, ki jih je vrnil klasifikator C_P. Definirajmo še funkcijo $In(c, C)$ s

$$In(c, C) = \begin{cases} 1, & c \in C \\ 0, & c \notin C \end{cases}.$$

[5] *ang.: k-fold cross-validation*

Ocena klasifikatorja je tedaj

$$Score = \frac{\sum_{d \in T} \sum_{c \in C_{dT}} In(c, C_{dP})}{\sum_{d \in T} N_{C_{dT}}}, \tag{4.31}$$

kjer je $N_{C_{dT}}$ število elementov množice pravih kategorij C_{dT} dokumenta d.

Ta mera dobro izraža naš cilj – pove nam delež pravilno najdenih pravih kategorij dokumentov, ki jih iščemo. Ne upošteva pa dejstva, da so kategorije urejene v hierarhično strukturo, in da je določitev starša ali otroka kategorije »manj napačna«, kot je umestitev dokumenta v popolnoma napačno poddrevo. Posledično ima klasifikator, ki dela predloge v lokalni okolici drevesa prave kategorije enako oceno kot tisti, ki vrača nesmiselne predloge.

4.4.2 Ocenjevanje z upoštevanjem hierarhije

Slabost prejšnje metode popravlja mera, ki sta jo predlagali S. Nowak in H. Lukashevich [11]. Pri tej meri za vsako izmed napačno predlaganih kategorij na drevesu poiščemo najbližjo pravo kategorijo in vsaki izmed zgrešenih pravih kategorij poiščemo najbližjo predlagano kategorijo. Te razdalje potem odštejemo od ocene pravilnosti dokumenta.

Pred uporabo te mere moramo izračunati uteži na povezavah med vozlišči drevesa kategorij. Ceno povezave a_i izračunamo z enačbo

$$a_i = \frac{2^{i-1}}{2 \cdot \sum_{i=1}^{N} 2^{i-1}}, \tag{4.32}$$

kjer je N višina drevesa, i pa trenutna globina v drevesu. Ta enačba poskrbi, da dolžina poti med poljubnima vozliščema v drevesu ni nikoli večja od 1.

Enako kot prej imamo dokumente d_i iz testne množice T, kjer vsakemu pripada množica pravih kategorij C_T in množica predlaganih kategorij C_P. Definirajmo še množico zgrešenih kategorij kot $C'_T = C_T \setminus (C_P \cap C_T)$ in množico napačno določenih kot $C'_P = C_P \setminus (C_P \cap C_T)$.

Cena vseh zgrešenih kategorij dokumenta se izračuna kot

$$W(C_P, C_T) = \sum_{c_i \in C_P,} \min_{c_j \in C_T} cost(c_i, c_j) + \sum_{c_j \in C_P,} \min_{c_i \in C_T} cost(c_i, c_j) \tag{4.33}$$

kjer $cost(c_i, c_j)$ predstavlja dolžino poti med vozliščema c_i in c_j. Ocena W bo enaka 1, če je klasifikator napačno določil vse razrede, in so pravi ter dodeljeni razredi v listih popolnoma ločenih poddreves pod korenom drevesa kategorij.

Končna ocena klasifikacije dokumenta se določi kot

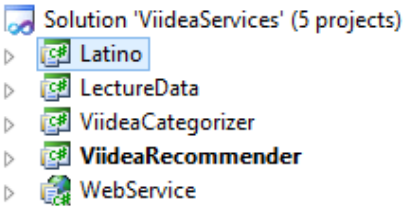
$$score_d = \left(1 - \frac{W(C_P, C_T)}{|C_P \cup C_T|}\right)^{\alpha}. \quad (4.34)$$

α je v tem primeru *Shenov α-faktor* [12], ki določa, kako močno bodo kaznovano določanje bližnjih kategorij v primerjavi s pravimi.

Za določitev ocene klasifikatorja moramo izračunati še povprečje ocen klasifikacije posameznih dokumentov v testni množici. Ta mera je od prejšnje opazno boljša, ker pravilneje oceni klasifikatorje, ki dajejo dobre, a ne čisto točne predloge.

5 Implementacija kategorizacije video predavanj

V nadaljevanju si bomo ogledali konkretno implementacijo storitve za kategorizacijo video predavanj na spletnem portalu VideoLectures[6]. Storitev je implementirana v programskem jeziku C# in je sestavljena iz štirih glavnih komponent. To so:



Slika 1: Struktura implementacije

- **Podatkovni model**, ki vključuje razrede, ki opisujejo predavanja, kategorije in avtorje ter funkcije za prenos in obdelavo podatkov s strani VideoLectures. Druge komponente se zanašajo na to komponento za pridobivanje podatkov.
- **Modul za predlaganje predavanj**, ki vključuje algoritme za iskanje podobnih predavanj, ki se pojavijo na spletni strani pod predlogi povezanih videov. Ta modul ne spada v obseg tega dela in si ga zato ne bomo podrobneje ogledali.
- **Modul za kategorizacijo predavanj**, ki vključuje algoritme za kategorizacijo, opisane v (4.3), ter implementacijo metod za ocenjevanje, opisanih v (4.4). Za lažje meritve se ta modul prevede tudi v samostoječo izvršilno datoteko, ki privzeto opravi 10-kratno križno preverjanje izbranega klasifikatorja.
- **Spletna storitev**, ki izpostavi API, na katerega se lahko spletna stran (v tem primeru VideoLectures) poveže in uporablja funkcionalnost kategorizacije in predlogov ter sporoča spremembe metapodatkov o predavanjih na to storitev. Implementirana je s knjižnico ServiceStack, ki izpostavi vmesnik čez protokole SOAP, JSON, XML in HTTP.

Vsi moduli se močno zanašajo na knjižnico Latino, ki je splošna knjižnica za reševanje problemov s področja podatkovnega rudarjenja (slika 1).

Implementirana storitev ima dva glavna namena:

1.) Za poljubno predavanje vrniti seznam predlogov predavanj, ki bodo prikazana ob strani (podobno kot »*suggestions*« na spletni strani YouTube)
2.) Za poljubno predavanje vrniti seznam predlogov kategorij, v katere spada. Namen tega je uredniku predlagati nekaj najprimernejših kategorij za video predavanje, da mu ni treba listati čez seznam par sto kategorij in izbirati pravih.

[6] http://www.videolectures.net

V tem delu se bomo osredotočili na drugi namen.

5.1 Podatkovna domena

Spletna stran VideoLectures trenutno vsebuje preko 17.000 posnetih predavanj, kategoriziranih v več kot 500 kategorij z vsaj 11.000 avtorji.

Dogodek predstavlja skupino predavanj (v večini primerov je to neka eno- ali večdnevna konferenca), večinoma z isto tematiko. *Predavanje* predstavlja video, ki ga pogosto spremljajo prosojnice. Predavanje lahko pripada dogodku ali pa je samostoječe. Prav tako ima predavanje enega ali več avtorjev in pripada nič ali več kategorijam. *Avtor* je oseba, ki je predavala na predavanju. *Kategorija* pa združuje predavanja s podobno tematiko in ima vedno *starša* in nič ali več *otrok*. Izjema je edino korenska kategorija `Top`, ki staršev nima.

V podatkovnih strukturah spletne strani so dogodki in predavanja predstavljeni z modelom `Lecture`, avtorji z modelom `Author` in kategorije z modelom `Category`.

Podatki so na spletni strani na voljo v JSON formatu v štirih datotekah:

- **`authors.json`** – Slovar vseh poznanih avtorjev
- **`categories.json`** – Slovar vseh poznanih kategorij
- **`lectures.json`** – Slovar vseh poznanih dogodkov in predavanj
- **`edges.json`** – Slovar, ki povezuje predavanja iz `lectures.json` s kategorijami v `categories.json` in avtorji iz `authors.json`

authors.json

```
{
   ...,
   "A8":{
         "url":"marko_hawlina",
         "name":"Marko Hawlina",
         "gender":"M",
         "organization":"Faculty of Medicine, University of Ljubljana",
         "email":"marko.hawlina@example.com",
         "refs":{
             "homepage":"http://ophthalmology2004.rsklan.com/"
         },
         "text":{
            "desc":"Marko Hawlina is Professor ..."
        },
   "A9": ...
}
```

Primer 7: Format datoteke authors.json

V `authors.json` (prikazana v primeru 7) najdemo slovar parov ključ/vrednost, kjer je ključ ID avtorja s predpono »A«, vrednost pa je slovar podatkov o avtorju. Podatki na voljo so

- `url` – pot do strani avtorja na spletni strani VideoLectures
- `name` – ime avtorja
- `gender` – spol, M - moški, F - ženski
- `organization` – organizacija, ki jo avtor predstavlja
- `email` – elektronski naslov avtorja
- `refs` – slovar dodatnih referenc na druge vire o avtorju, kot so domača stran ipd.
- `text` – slovar besedilnih podatkov o avtorju, trenutno vključuje samo polje `desc` z biografskim opisom avtorja

categories.json

```
{
   ...,
   "C20":{
       "url":"Top/Computer_Science/Machine_Learning/Boosting",
       "refs":{
           "wiki":"http://en.wikipedia.org/wiki/Boosting"
       },
       "edges":{
           "parent":"C16"
       },
       "text":{
           "title":"Boosting"
       },
   },
   "C21":...
}
```

Primer 8: Format datoteke categories.json

V `categories.json` (prikazana v primeru 8) najdemo slovar kategorij, kjer so ključi IDji kategorij s predpono »C« ter vrednosti slovarji s podatki o kategoriji:

- `url` – pod do prikaza kategorije na spletni strani VideoLectures, hkrati tudi celotna pot po drevesu kategorij
- `refs` – reference na druge vire o kategoriji. Trenutno vključujejo samo polje `wiki` s povezavo na informacije o tematiki kategorije na Wikipediji
- `edges` – vključuje samo polje `parent`, ki ima za vrednost ID starševske kategorije v drevesu kategorij
- `text` – vključuje besedilne opise kategorije, trenutno `title`, ki vsebuje ime kategorije in `desc` z daljšim opisom

lectures.json

```
{
  "L201":{
      "lang":"en",
       "url":"solomon_berendt_twum",
       "type":"tutorial",
       "views":2124,
       "enabled":1,
       "public":1,
       "recorded":"2006-10-06T10:00:00",
       "published":"2007-02-25",
       "text":{
              "title":"Semantic Web Usage Mining",
              "desc":"In this tutorial we will review...",
              "slides":["Semantic Web Usage Mining \u2013 Overview and Case
                        Studies", "Goals and top-level questions",?", ...]
       }
},
...
}
```

Primer 9: Format datoteke `lectures.json`

V `lectures.json` (primer 9) je slovar vseh predavanj in dogodkov na spletni strani. Ključi so IDji predavanj s predpono `L`, vrednost pa je slovar podatkov o predavanju:

- `lang` – dvočrkovna vrednost, ki opisuje jezik predavanja
- `url` – je pot na spletni strani do predavanja (t.i. *slug*)
- `type` – vrsta vnosa, dogodki imajo vrednost `event`, ostala predavanja pa imajo vrednosti glede na tip: `lecture`, `tutorial`, `keynote`, ...
- `views` – število ogledov predavanja do sedaj
- `enabled` – zastavica, ki določa ali je predavanje omogočeno – torej je vidno in se pojavlja v seznamih
- `public` – zastavica, ki določa če je predavanje javno vidno
- `recorded` – čas in datum snemanja predavanja v obliki ISO 8601[7]
- `published` – datum objave na portalu v obliki ISO 8601
- `text` – slovar tekstovnih opisov predavanja

[7] *to je oblika leto-mesec-danTura:minute:sekunde. Npr. »2006-10-06T16:32:22«*

- `title` – naslov predavanja
- `desc` – besedni opis vsebine predavanja
- `slides` – seznam naslovov prosojnic v prezentaciji

edges.json

```
{
...,
"L17":{
        "parent":"L2",
       "categories":{"C36":1.0000},
       "authors":{"A3":"author"},
       "related":{"L1998":23,"L15930":18,"L1974":12,"L1888":12,...}
},
...
}
```

Primer 10: Format datoteke `edges.json`

Datoteka `edges.json` (primer 10) opisuje graf odnosov med predavanji, kategorijami in avtorji. Zapisana je v obliki slovarja, ki ima za ključe IDje predavanj iz datoteke `lectures.json` in kot vrednosti povezave.

Vrednosti:

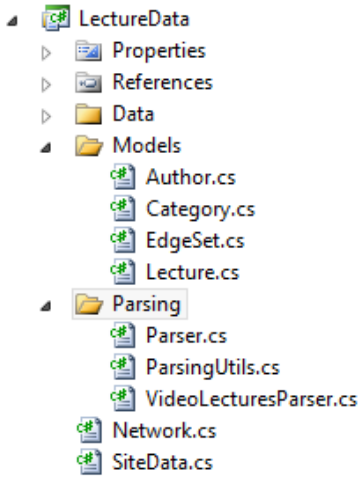
- `parent` – ID dogodka, ki mu predavanje pripada
- `categories` – slovar kategorij, ki jim predavanje pripada. Ključi so IDji kategorij iz datoteke `categories.json`, vrednosti pa pozitivne in negativne uteži, ki jih določajo uporabniki z glasovanjem
- `authors` – slovar avtorjev predavanja, ključi so IDji avtorjev iz datoteke `authors.json`, vrednosti pa določajo vrsto avtorstva: `author`, `coauthor`, `organizer`
- `related` – povezana predavanja glede na oglede uporabnikov. Določijo se tako, da se izračuna pot obiskovalca med predavanji na spletni strani[8], zatem pa se izberejo predavanja, ki se pojavijo največkrat, do globine 10 po tej poti. Ključ je ID predavanja, vrednost pa število uporabnikov, pri katerih se pojavi predavanje znotraj razdalje 10.

[8] *t.i. click-stream*

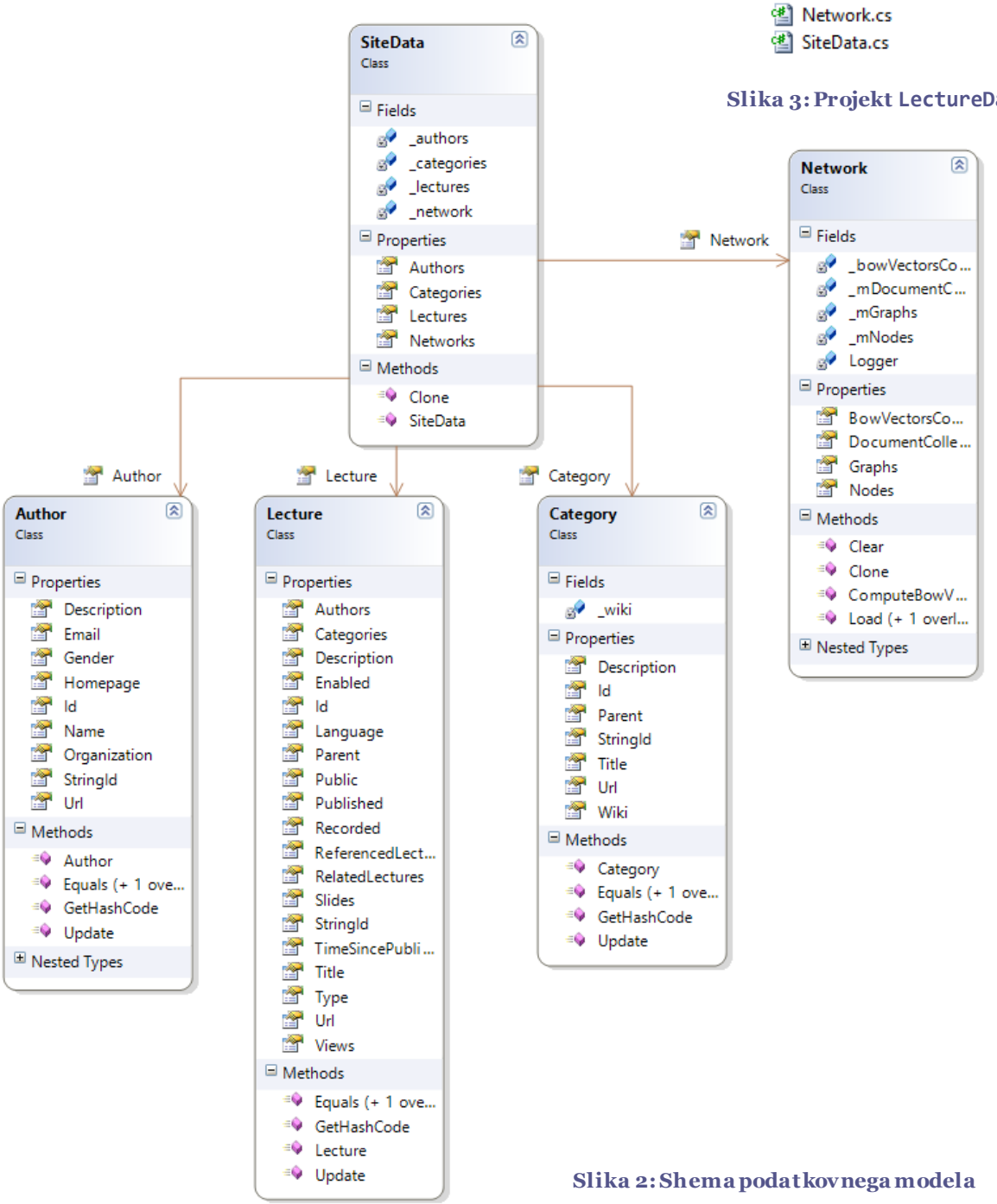
S pomočjo teh štirih datotek lahko rekonstruiramo celotno vsebino spletne strani VideoLectures. Manjkajo le video posnetki sami in vsebina prosojnic. Na osnovi vsebine teh datotek potem sestavimo lastni model podatkov znotraj aplikacije.

5.2 Podatkovni model

Za predstavitev in obdelavo podatkov z vmesnika JSON spletne strani skrbi projekt `LectureData`. Vključuje orodja za obdelavo v (5.1) omenjenih JSON datotekah ter razrede, ki podatke predavanj predstavijo. Shema razredov, ki tvorijo podatkovni model, je prikazana v slikah 3 in 2.



Slika 3: Projekt `LectureData`



Slika 2: Shema podatkovnega modela

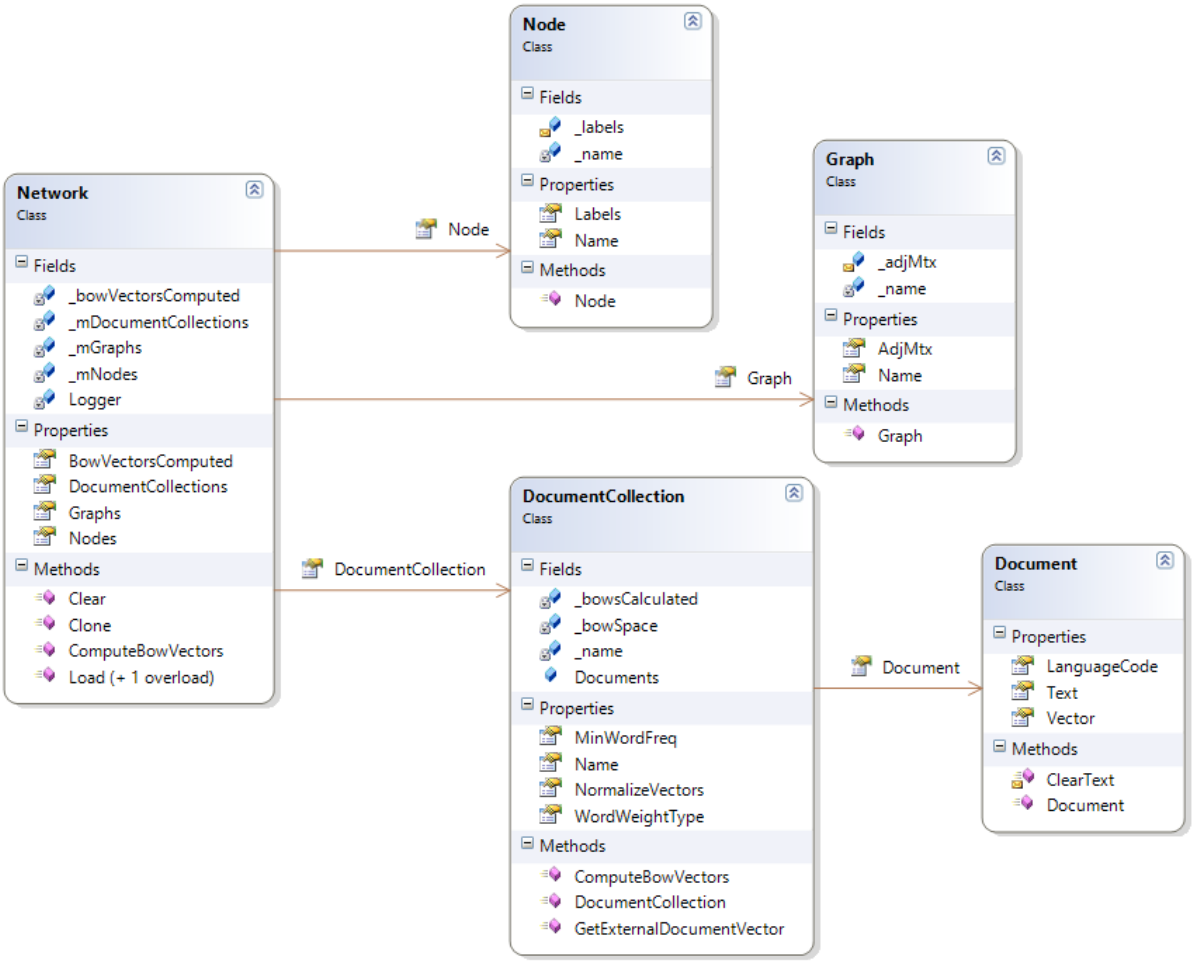
`SiteData` je razred, ki predstavlja konsistentno sliko podatkov spletne strani. Lastnosti `Authors`, `Lectures` in `Categories` so slovarji tipa `Dictionary`, ki imajo za ključe celoštevilske IDje (brez predpon »A«, »L« in »C«), kot vrednost pa ima objekt ustreznega tipa. Posebnost je le razred `Network`, ki si ga bomo podrobneje ogledali kasneje.

Vsak izmed razredov `Author`, `Lecture` in `Category` vključuje lastnosti, ki so analogne poljem, opisanih v (5.1). Posebnost sta le lastnosti `Id` in `StringId`. `StringId` vsebuje ID predmeta skupaj z ustrezno črkovno predpono (»C«, »L«, »A«), `Id` pa je celoštevilskega tipa in vsebuje le številčni del IDja predmeta. Vsak izmed razredov pa vključuje tudi metodo `Update()`, ki kot parameter sprejme instanco drugega objekta istega tipa in posodobi lastnosti, tako da se ujemajo s parametrom. Metoda se uporablja za posodabljanje podatkov s klici s spletne strani, ne da bi bilo potrebno zamenjati instanco objekta v vseh slovarjih in ostalih podatkovnih strukturah modelov za predlaganje in kategorizacijo predavanj.

```
public void Update(Author author)
{
   if (Id != author.Id)
      throw new ArgumentException("Author IDs must match for update!");

   Name = author.Name;
   Url = author.Url;
   Gender = author.Gender;
   Organization = author.Organization;
   Email = author.Email;
   Homepage = author.Homepage;
   Description = author.Description;
}
```

Primer 11: Funkcija Update razreda Author

Razred `Network` si zasluži bolj podroben pregled.



Slika 4: Razred `Network`

Razred `Network` (slika 4) je namenjen opisu podatkov v obliki grafov in njihovi predstavitvi v vektorskem prostoru. Lastnosti razreda so

- `DocumentCollections` – seznam objektov `DocumentCollection`, ki predstavlja zbirko besedilnih dokumentov
- `Graphs` – seznam objektov `Graph`, ki predstavlja graf vozlišč
- `Nodes` – seznam objektov `Node`, ki predstavljajo vozlišča v grafu

Vsak objekt tipa `Graph` vsebuje ime in matriko podobnosti tipa `SparseMatrix<double>` s knjižnice Latino. Vsak `Node` ima ime in seznam oznak, ki ji pripadajo.

Vsak `DocumentCollection` predstavlja vektorski model vsebovanih dokumentov, kot smo si ga ogledali v (4.2). Vsebuje seznam objektov `Document`, ki vsak predstavlja en besedilni dokument in vsebuje dvočrkovno kodo jezika besedila, besedilo samo in po izračunu besedilni vektor dokumenta tipa `SparseVector<double>` knjižnice Latino. Besedilni vektorji se izračunajo s klicem metode `ComputeBowVectors()`, lastnost `_bowsCalculated` pa določa, ali imajo dokumenti te vektorje že izračunane.

Lastnosti, ki kontrolirajo izračun vektorjev dokumentov, so

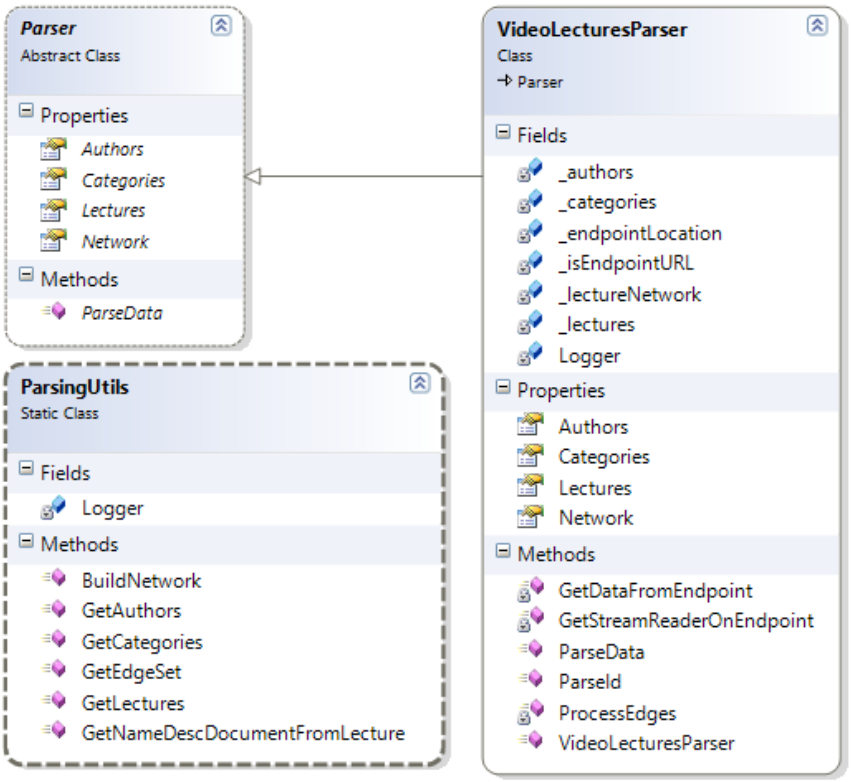
- `MinWordFreq` (*double*) – minimalna frekvenca besede v dokumentu, da postane del vektorja
- `NormalizeVectors` (*boolean*) – če je ta zastavica postavljena, so dokumentni vektorji normalizirani, tako da so vrednosti komponent v intervalu [0, 1.0]
- `WordWeightType` (*enum*) – določa vrsto obteževanja vektorjev, veljavne vrednosti so
  - `TermFrequency` – vektorji niso obteženi
  - `TfIdf` – vektorji so obteženi s TF/IDF mero opisano v 4.2.1
  - `LogTfIdf` – vektorji so obteženi z logaritmom TF/IDF mere iz 4.2.1

Po klicu metode `ComputeBowVectors()` v razredu `Network`, se pokliče metoda `ComputeBowVectors()` na vsakem `DocumentCollection` razredu. Po izračunu vseh vektorjev, se napolnijo lastnosti `Vector` razredov `Document` v zbirkah dokumentov in se zastavice `BowVectorsComputed` postavijo na `true`.

Izračunane vektorje potem uporabljajo vsi implementirani algoritmi za predlaganje in kategorizacijo predavanj.

5.2.1 Izgradnja podatkovnega modela

Podatkovni model se iz JSON datotek zgradi s pomočjo razredov `Parser`, `VideoLecturesParser` in `ParserUtils` (prikazani v sliki 5).



Slika 5: Razredi `Parser`, `VideolectureParser` in `ParsingUtils`

`Parser` (primer 12) je abstraktni razred, ki samo implementira vmesnik za generično obdelovanje spletnih strani. Izpostavi metodo `ParseData()`, ki začne obdelovanje vhodnih datotek ter lastnosti `Authors`, `Categories`, `Lectures` in `Network`, ki po obdelavi vsebujejo slovarje parov ID / objekt ustreznih vsebin.

```
public abstract class Parser
{
    public abstract Network Network { get; }
    public abstract Dictionary<int, Category> Categories { get; }
    public abstract Dictionary<int, Author> Authors { get; }
    public abstract Dictionary<int, Lecture> Lectures { get; }
    public abstract Network ParseData();
}
```

Primer 12: Abstraktni razred Parser

Iz razreda `Parser` potem deduje `VideoLecturesParser`, ki dejansko obdela podatke spletne strani VideoLectures. V konstruktorju mu lahko podamo URL do končne točke na spletu ali pa pot do direktorija z JSON datotekami na lokalnem disku. Ob klicu `ParseData()` se naložijo JSON datoteke in se s pomočjo metod razreda `ParseUtils` sprocesirajo v objekte in shranijo kot lastnosti razreda.

Metode `GetAuthors()`, `GetLecures()`, `GetEdgeSet()`, `GetCategories()` vse sprejmejo JSON zapis iz ustrezne datoteke in vrnejo slovar objektov razredov `Author`, `Lecture`, `EdgeSet` in `Category`. Iz teh slovarjev lahko zatem metoda `GetNetwork()` ustvari ustrezen razred `Network` z dokumenti. Privzeto ustvari dve zbirki dokumentov (`DocumentCollection`) z imenoma `Names` in `Names_Descriptions`. Vsak dokument predstavlja eno predavanje, besedilno polje pa je v primeru zbirke `Names` napolnjeno samo z imenom predavanja in avtorjev, v primeru `Names_Descriptions` pa je tekstovno polje napolnjeno z naslovom predavanja, imeni avtorjev, naslovom starševskega dogodka, opisom predavanja in naslovi vseh prosojnic. Metoda `GetNameDescDocumentFromLecture` sprejeme kot parameter objekt `Lecture` in zgradi objekt razreda Document, ki ustreza zbirki dokumentov `Names_Descriptions`.

Končni rezultat gradnje modela je objekt `SiteData`, ki vsebuje podatke o vseh (trenutnih) avtorjih, kategorijah, predavanjih ter pripravljene tekstovne dokumente z izračunanimi TF/IDF vektorji.

5.3 Modul za predloge podobnih predavanj

Projekt `ViideaRecommender` vsebuje modul, ki za vsako predavanje izračuna zanimiva podobna predavanja glede na avtorja, kategorije, opis ter opazovano obnašanje dosedanjih gledalcev tega predavanja (slika 6).

Algoritem za predloge predstavlja abstraktni razred `Recommender`, ki je prikazan v primeru 13.

```
public abstract class Recommender
{
   public abstract bool IsCalculationInProgress();
   public abstract void RecalculateSimilarities(SiteData siteData);


   public abstract List<Tuple<double, int>> GetSimilaritiesForLecture(
                                                              int lectureId);
   public abstract Hashtable GetAllSimilarities();
   public abstract bool ModelReady { get; }
   public abstract string Name { get; }
   public abstract DateTime ModelCalculatedOn { get; }
}
```

Primer 13: Abstraktni razred Recommender

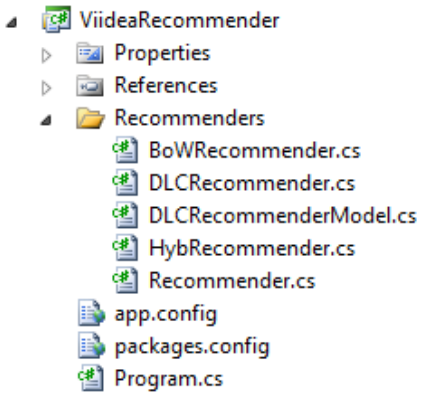
Model za predloge se izgradi z metodo `RecalculateSimilarities`, ki kot parameter prejme objekt `SiteData` z vsemi podatki o predavanjih. Izračun modela traja nekaj časa, za nemoteno delovanje pa se od implementacij algoritmov pričakuje, da bodo med računanjem novega modela še vedno vračali predloge, določene iz starega modela.

Stanje modela vračajo metode
- `ModelReady`, ki vrne `true`, če je katerikoli model izračunan in algoritem lahko vrača predloge
- `IsCalculationInProgress()` vrne `true`, če se trenutno nov model računa
- `ModelCalculatedOn` vrne čas in datum izgradnje modela, ki trenutno vrača predloge

Metoda `GetSimilaritiesForLecture` sprejme ID predavanja in urejen seznam najboljših 30 predlogov. Seznam je sestavljen iz terk, kjer je prvi element podobnost (*score*, odvisen od algoritma) ter drugi element ID predavanja. `GetAllSimilarities()` pa vrne zgoščeno tabelo, kjer so ključi vsa poznana predavanja, vrednosti pa enake kot rezultat klica `GetSimilaritiesForLecture` za tisto predavanje. Od algoritmov se pričakuje, da lahko sprejemajo klice z metod z večih niti hkrati (morajo biti *thread-safe*).

Kot je razvidno iz vmesnika, trenutna implementacija ne podpira vračanja predlogov za predavanja, ki niso bila del izračuna modela.

Trenutno so implementirani trije algoritmi za predloge, dva, ki sta bila predstavljena na European Conference of Machine Learning and Principles of Practice of Knowledge Disctovery in Databases (Atene, Grčija 2011) ter eden preprost, ki služi kot kontrola.



Slika 6: Projekt ViideaRecommender

Prvi algoritem je osnovan na deformiranih linearnih kombinacijah in je bil objavljen v »Two Recommendation Algorithms Based on Deformed Linear Combinations« [13] avtorja Alexandra D'yakonova. Za izračun podobnosti se uporabi obtežena linearna kombinacija večih vektorjev: TF/IDF vektorja `Names_Descriptions` iz zbirke dokumentov, vektorjev pripadnosti avtorjem in kategorijam ter vektorjev, ki določajo časovno razliko med objavo predavanj in število uporabnikov, ki so si določen par predavanj ogledali v zaporedju. Implementiran je v razredih `DLCRecommender` (glavna logika) in `DLCRecommenderModel` (model predlagalnika).

Drugi algoritem za določitev podobnosti izračuna posebej prirejene TF/IDF vektorje (kjer se naslov predavanja, avtorji ter naslov dogodka večkrat ponovi), ki jih zatem dodatno obteži glede na število hkratnih ogledov določenega para predavanj ter časovno razliko med objavo para predavanj. Objavljen je bil v »A Hybrid Approach for Cold-start Recommendations of VideoLectures« [14] avtorjev Spyromitros-Xioufis, Stachtiari, Tsoumakas in Vlahavas. Implementiran je v razredu `HybRecommender`.

Tretji algoritem služi kot kontrola delovanja in preprosto izračuna kosinusne razdalje med pari TF/IDF vektorji predavanj iz zbirke dokumentov `Names_Descriptions`. Implementiran je kot razred `BoWRecommender`.

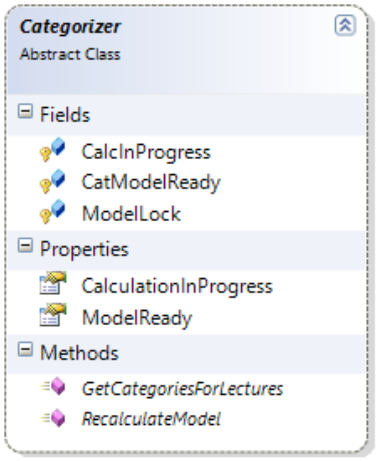
Poleg algoritmov projekt sestavlja tudi `Program.cs`, kjer je implementirana koda, ki se prevede v program `ViideaRecommender.exe`, ki iz direktorija ali URLja z JSON datotekami lahko izračuna predloge in jih izpiše v izhodno JSON datoteko.

5.4 Modul za kategorizacijo

Kategorizacija predavanj je implementirana v modulu `ViideaCategorizer`. Sestavljen je iz implementacije večih klasifikatorjev in implementacij razredov za ocenjevanje kvalitete klasifikacij (slika 8).

5.4.1 Abstraktni vmesnik

Vsak klasifikator mora razširjati abstraktni razred `Categorizer`, ki definira zunanji vmesnik kategorizatorjev. Razredni diagram je prikazan na sliki 7.



Slika 7: Razred `Categorizer`

Abstraktni razred že definira tri privatna polja:

- `CalcInProgress` (*bool*), ki označuje, ali klasifikator trenutno računa nov model,
- `CatModelReady` (*bool*), ki označuje, da ima klasifikator izračunan model in lahko vrača kategorije ter
- `ModelLock` (*ReaderWriterLock*), ki je ključavnica tipa `ReaderWriterLock`, ki je namenjena zaklepanju dostopa do modela in dovoljuje več hkratnih dostopov za branje in samo en ekskluziven dostop za pisanje

Poleg polj sta že implementirani lastnosti `ModelReady`, ki vrne vsebino polja `CatModelReady` in zaščiti dostop z uporabo `ModelLock` ključavnice ter `CalculationInProgress`, ki samo vrne vsebino polja `CalcInProgress`. Oba dostopa sta mogoča samo za branje.

Interakcijo s klasifikatorjem omogočata abstraktni metodi `GetCategoriesForLectures` in `RecalculateModel`, ki sta prikazani v primeru 14.

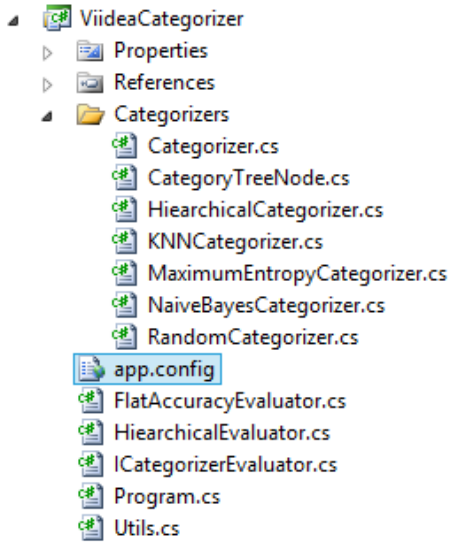
```
public abstract void RecalculateModel(SiteData data,
                                      Set<int> ignoreLectures = null);

public abstract Dictionary<int, List<Tuple<double, int>>>
                              GetCategoriesForLectures(IList<Lecture> lectures);
```

Primer 14: Podpisa abstraktnih metod `RecalculateModel` in `GetCategoriesForLectures`

`RecalculateModel` sproži izračun novega modela v klasifikatorju. Sprejme dva parametra: `data`, ki je tipa `SiteData` in vključuje podatke o predavanjih spletne strani, kot je opisano v poglavju 5.2.
Drugi, neobvezni, parameter `ignoreLectures` je množica ID-jev predavanj, ki naj se pri izgradnji modela ignorirajo. Uporaben je predvsem za testiranje klasifikatorjev, saj nam ni potrebno pripraviti nove instance podatkov strani `SiteData` za vsako iteracijo n-kratnega križnega preverjanja.

`GetCategoriesForLectures` vrne predloge kategorij za predavanja, ki so podana kot seznam objektov tipa `Lecture`. Rezultat metode je slovar, ki ima za ključ ID predavanja, vrednost pa je seznam terk, kjer ima vsaka terka vrednost tipa `double`, ki predstavlja primernost kategorije ter `int`, ki predstavlja ID kategorije. Število rezultatov je odvisno od klasifikatorja, pričakuje se pa, da ni veliko (ranga 10-20).

Od konkretne implementacije zgoraj naštetih metod se pričakuje, da bodo varne za dostop z večih niti in bodo hkratne dostope ščitile z uporabo ključavince `ModelLock`. Prav tako se od implementacije pričakuje, da bo med izračunom novega modela po klicu `RecalculateModel`, klasifikator še vedno vračal rezultate glede na stari model.



Slika 8: Projekt ViideaCategorizer

5.4.2 Implementacije klasifikatorjev

Za namene tega dela je bilo implementiranih pet različnih klasifikatorjev, ki so v večini opisani v poglavju 4.3. Struktura projekta je prikazana na sliki 8. Vsi klasifikatorji izračune izvajajo v več nitih in so sposobni izkoristiti večjedrne procesorje za hitrejše izračune s pomočjo `Parallel` konstruktov v jeziku C#.

Naključni klasifikator

Naključni klasifikator je implementiran v razredu `RandomCategorizer` in za razliko od ostalih ne zgradi nobenega modela, ampak si samo zapomni IDje kategorij iz objekta `SiteData`. Pri klicu metode `GetCategoriesForLectures` vrne pet naključnih kategorij za vsako predavanje.

Uporaben je predvsem kot kontrola implementacij razredov za testiranje klasifikatorjev.

Naivni Bayesov klasifikator

Naivni Bayesov klasifikator je implementiran v razredu `NaiveBayesCategorizer` in je originalna implementacija algoritma iz poglavja 4.3.1. Pri klicu `RecalculateModel` se izračunajo in shranijo pogojne verjetnosti za kategorije in vektorji, ki predstavljajo vsoto frekvenc terminov za vsako kategorijo.

Pri klicu `GetCategoriesForLectures` se za vsako predavanje izračuna verjetnost pripadnosti kategoriji po enačbi 4.15. Za vsako predavanje se vrne 30 kategorij z največjo verjetnostjo. Implementacija je prikazana v primeru 15.

```
foreach (var categoryVector in _categoryVectors)
{
   var score = Math.Log(_priorProbabilities[categoryVector.Key]) +
               // Calculate log sum of terms contained in lecture vec
               lectureVector.Select(idxDat => categoryVector
                                              .Value.TryGet(idxDat.Idx, 0.0))
                     .Where(termScore => termScore > Double.Epsilon)
                     .Sum(termScore => Math.Log(termScore));
   lectureResults.Add(new Tuple<double, string>(-score,
                                                categoryVector.Key));
}

var result = lectureResults.OrderByDescending(res => res.Item1)
                           .Take(30)
```

Primer 15: Izračun kategorij v naivnem Bayesovem klasifikatorju

Klasifikator s k najbližjimi sosedi

Kategorizator s k najbližjimi sosedi (v nadaljevanju kNN – *k-Nearest Neighbour*) uporablja implementacijo kNN klasifikatorja v knjižnici Latino in je realiziran kot razred `KNNCategorizer`. Algoritem je bil opisan v poglavju 4.3.2. V učno množico se doda vektor vsakega predavanja za vsako kategorijo, ki ji to predavanje pripada. Oznaka (*label*) primera v učni množici je kar ID kategorije. Izsek kode je prikazan v primeru 16.

```
var dataset = new LabeledDataset<string, SparseVector<double>>();
var lectureVecs = new Dictionary<int, SparseVector<double>>();

// Fill lectureVecs dictionary so that keys are lecture IDs and values are
lecture vectors

foreach (var lecture in lectures)
{
    foreach (var category in lecture.Categories)
    {
        dataset.Add(new LabeledExample<string, SparseVector<double>>(
                        category.StringId,
                        lectureVecs[lecture.Id])
                    );
    }
}
```

Primer 16: Priprava učne množice za klasifikatorje v knjižnici Latino

Pri vračanju kategorij za predavanje se izbere do 10 sosedov, ki imajo oceno boljšo od določenega faktorja (empirično je bilo ugotovljeno, da je glede na metrike, opisane v 4.4.1 in 4.4.2, najboljša vrednost 0.45) in se jih posortirane vrne, ko je to prikazano v primeru 17.

```
var predictions = _classifier.Predict(lectureVector);
var results = predictions.Where(dat => dat.First > CutoffFactor)
                  .OrderByDescending(dat => dat.First)
                  .Take(10)
                  .Select(dat => new Tuple<double, string>(dat.First,
                                                           dat.Second))
```

Primer 17: Določanje kategorij s kNN klasifikatorjem

Klasifikator z iskanjem največje entropije

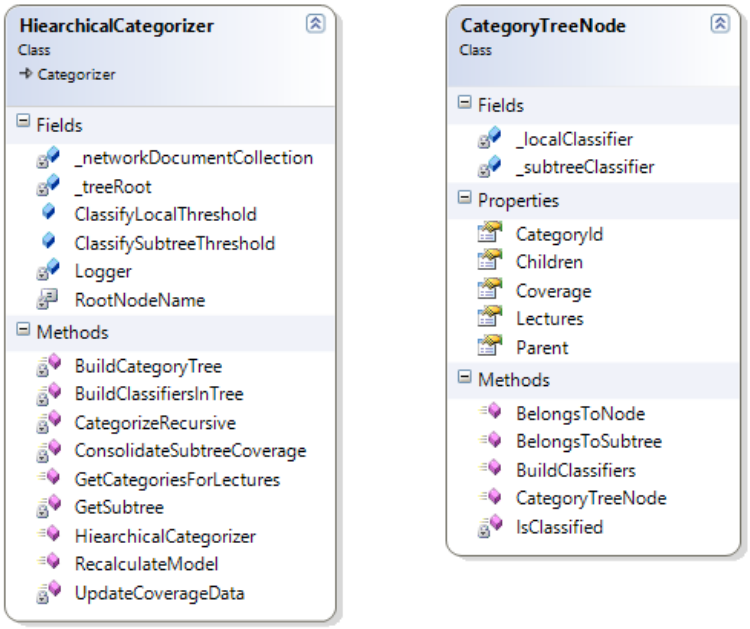
V razredu `MaxEntropyCategorizer` je s pomočjo knjižnice Latino implementiran klasifikator, ki je opisan v poglavju 4.3.3. Večina implementacije klasifikatorja je zaobjeta v razredu `MaxEnt`. Učna množica se zgradi enako, kot je prikazano v primeru 16, vrne pa 10 najboljših rezultatov. Empirično testiranje je pokazalo, da 10 iteracij iskanja največje entropije predstavlja optimalno razmerje med kvaliteto rezultatov glede na metrike, opisane v 4.4.1 in 4.4.2, in časom računanja modela. Klasifikator vrne 10 kategorij z najboljšim rezultatom.

```
var dataset = DataUtils.BuildLabeledCategoryDataset(data.Network,
                                                    data.Lectures.Values,
                                                    ignoreLectures);
var maxClassifier = new MaximumEntropyClassifier<string>();
maxClassifier.CutOff = 1;
maxClassifier.NumIter = 10;
maxClassifier.MoveData = true;
maxClassifier.NumThreads = Environment.ProcessorCount;
maxClassifier.Train(dataset.ConvertDataset<BinaryVector>(true));
```

Primer 18: Gradnja modela klasifikatorja z maksimalno entropijo

Hierarhični klasifikator

Hierarhični klasifikator temelji na načelih, opisanih v poglavju 4.3.4, sestavljata ga pa razreda `HiearhicalCategorizer` in `CategoryTreeNode`. Razredni diagram obeh razredov je prikazan na sliki 9.



Slika 9: Razredni diagram hierarhičnega klasifikatorja

Učenje modela v hierarhičnem klasifikatorju poteka v več stopnjah:

1. V funkcij `BuildCategoryTree` se iz podatkov v `SiteData` objektu zgradi drevo kategorij, ki je sestavljeno iz primerkov razreda `CategoryTreeNode`. V vsakem vozlišču se postavijo naslednje lastnosti:
   - `CategoryId`, ki vsebuje ID kategorije, ki jo to vozlišče predstavlja
   - `Children`, ki je seznam vseh otrok tega vozlišča
   - `Parent`, ki kaže na starša tega vozlišča

   Med grajenjem drevesa se vozlišča, primerki razreda `CategoryTreeNode`, sproti vstavljajo v razpršeno tabelo `nodeDictionary`, kjer so ključi IDji kategorije in se uporablja za hiter dostop do vozlišča drevesa.

2. Po izgradnji drevesa se v funkciji `BuildCategoryTree` algoritem sprehodi čez vsa predavanja v objektu `SiteData` in s pomočjo `nodeDictionary` razpršene tabele napolni sezname predavanj `Lecture` v vozliščih drevesa.

3. Zatem se v funkciji `ConsolidateTreeCoverage` rekurzivno sprehodi po drevesu in od listov proti korenu vozliščem napolni lastnost `Coverage`, ki je množica (`Set`, točneje `HashSet`) IDjev predavanj, ki pripadajo vozlišču ali vozliščem v poddrevesih otrok.

Na tej točki imamo zgrajeno drevo kategorij iz objektov `CategoryTreeNode`, ki imajo pravilno postavljene vse lastnosti. Zdaj je potrebno zgraditi lokalne klasifikatorje v vozliščih.

4. Pokliče se funkcija `BuildClassifiersInTree`, ki se paralelno (večnitno) sprehodi čez vsa vozlišča in v njih zgradi dva binarna SVM klasifikatorja – klasifikator vozlišča in klasifikator poddrevesa. Za implementacijo lokalnih klasifikatorjev se uporablja C knjižnica SvmLightLib [15], ki predstavlja hitro implementacijo SVM binarnih klasifikatorjev.
   Klasifikator vozlišča se zgradi iz množice vektorjev predavanj, ki pripadajo samo trenutnemu vozlišču in se shrani v lokalno spremenljivko `_localClassifier`.
   Klasifikator poddrevesa pa se zgradi iz vektorjev predavanj celega pokritja trenutnega vozlišča[9].

Po izgradnji lokalnih klasifikatorjev je model pripravljen na uporabo.

Klasifikacija primerov potem poteka podobno. Metoda `CategorizeRecursive` (prikazana v primeru 19) se rekurzivno spusti po drevesu in na vozliščih kliče najprej funkcijo `BelongsToNode`, ki pove, ali primer pripada kategoriji in zatem kliče funkcijo `BelongsToSubtree`, ki pove, ali se splača nadaljevati sestop po drevesu.

---

[9] *Pokritje je definirano v poglavju 4.3.4 na strani 20*.

Rezultat se doda kandidatom, če zaupanje presega konstanto, določeno v `ClassifyLocalTreshold`.

```
private static void CategorizeRecursive(CategoryTreeNode node,
                                        SparseVector<double> lectureVector,
                                        List<Tuple<double, string>> results)
{
   double confidence;
   if (node.BelongsToNode(lectureVector,
                          out confidence,
                          classifyTreshold:ClassifyLocalThreshold))
   {
      results.Add(new Tuple<double, string>(confidence, node.CategoryId));
   }

   if (node.BelongsToSubtree(lectureVector,
                             out confidence,
                             classifyTreshold:ClassifySubtreeThreshold))
   {
      foreach (var categoryTreeNode in node.Children)
      {
         CategorizeRecursive(categoryTreeNode, lectureVector, results);
      }
   }
}
```

Primer 19: Metoda CategorizeRecursive

Kot rezultat metode `GetCategoriesForLectures` se vrne 10 elementov z najboljšo oceno iz seznama `results`.

5.5 Spletni vmesnik

Modul za kategorizacijo in modul za predloge se samostojno prevedeta v izvršljivi datoteki, ki iz vhoda v obliki datotek JSON, opisanih v 5.1, ustvarita izhodno JSON datoteko s predlogi predavanj oz. s kategorijami. Za testiranje in merjenje je tak pristop ustrezen, je pa zelo nepraktičen za uporabo pri živi spletni strani. To težavo rešuje modul, ki v obliki spletne storitve izpostavi vso funkcionalnost kategorizatorjev in predlagalnikov.

Spletni vmesnik je zgrajen z uporabo knjižnice .NET ServiceStack [16], ki je odprtokodna knjižnica za gradnjo spletnih storitev v .NET. Razlog za izbiro je enostavna gradnja vmesnikov REST, ki uporabljajo JSON, SOAP ali XML kot obliko prenosa podatkov. Alternativa ASP.NET ob času implementacije še ni bila dobro podprta v okolju Mono, kar bi pomenilo, da vmesnik ne bi tekel na Linuxu ali OS X.

Modul s spletnim vmesnikom opravlja dve funkciji:

- Izpostavlja vmesnik za predloge in kategorizacijo preko HTTP, zato da lahko spletne strani v živo pridobivajo predloge in kategorije za predavanja
- Shranjuje podatkovni model spletne strani v podatkovno bazo PostgreSQL in izpostavlja vmesnik HTTP za posodabljanje teh podatkov

Shranjevanje podatkov si zasluži podrobnejšo razlago. Generiranje datotek JSON z vsemi predavanji, kategorijami in avtorji na strani velikosti VideoLectures traja nekaj minut in zahteva precej procesorskega časa ter izvajanje relativno kompleksnih poizvedb na podatkovni bazi. Posledično je ta čas odzivnost spletne strani opazno slabša. Zato tega postopka ni praktično izvajati tekom dneva, še posebej ne pogosto, kar pomeni, da so podatki vedno nekoliko zastareli. Vidimo tudi, da se večina teh podatkov praktično nikoli ne spremeni – tedensko se doda nekaj novih predavanj, pri nekaj predavanjih ali avtorjih pa se spremenijo polja. Redno izvažanje vseh podatkov zato ni smiselno.

Posledično modul spletnega vmesnika shranjuje vse relevantne podatke za izgradnjo modelov predlagalnikov in kategorizatorjev (točneje, vse podatke v objektu SiteData) v lastni PostgreSQL podatkovni bazi in potem sprejema samo obvestila o spremembah metapodatkov. Storitev lahko tako zagotavlja ažurne predloge, brez da bi obremenjevala glavno spletno stran.

5.5.1 Vmesnik spletne storitve

Vsi klici se izvajajo kot zahtevki GET ali POST preko protokola HTTP in vsakemu klicu pripada svoj URL.

ServiceStack obliko rezultata zahtevka določi glede na vsebino »Accept:« glave v HTTP zahtevku. Trenutno upoštevani MIME tipi so:

- `application/json` za format JSON
- `application/xml` za format XML

- `text/jsv` za format JSV
- `text/csv` za tekst ločen z vejicami
- `text/html` za polepšano HTML stran

V kolikor MIME tip ni prepoznan, bo privzeto vrnjena stran HTML s človeško berljivimi rezultati klica. Vsi klici vrnejo polje `Status`, ki ima vsebino »`OK`« za uspešno izveden klic ali pa kodo napake.

Veljavni klici za upravljanje s podatki strani so

- `/reload/files` – iz JSON datotek v obliki, razloženi v 5.1 v direktoriju `App_Data`, se naložijo podatki in se napolni podatkovna baza. Vsi morebitni obstoječi podatki v bazi se pobrišejo.

- `/reload/db` – Podatki se ponovno naložijo iz podatkovne baze, klic je predvsem uporaben za odpravljanje težav pri razvoju.

- `/rebuild/` - ukaže vsem predlagalnikom in klasifikatorjem za kategorije, naj ponovno zgradijo svoje modele iz podatkov

- `/update/` - posodobi podatkovni model v storitvi. Klic za posodobitev mora biti tipa HTTP POST in lahko vsebuje polja `Authors`, `Lectures`, `Categories` in `Edges`. Vsako od polj (če obstaja) mora vsebovati zapis JSON, ki ustreza zapisu v datotekah JSON, opisanih v 5.1. Pri posodobitvi se prepišejo vsi podatki, kar pomeni, da mora biti pri spremembi (npr. naslova predavanja) poslan celoten zapis predavanja z enakim ID za ključ. Enako velja za odnose med kategorijami, avtorji in predavanji – če hočemo izbrisati kategorijo, mora biti v `Edges` ponovno poslana struktura, ki opisuje povezave za vsa predavanja, ki so prej pripadala tisti kategoriji.

Klica za ponovno nalaganje podatkov se uporabljata predvsem, ko se spremeni večji delež podatkov oz. se zaradi napake posodabljanje ni pravilno izvajalo. Klic za ponovno gradnjo modelov pa se tipično poganja periodično. Ker klic zahteva veliko sistemih virov (vzporedni izračun večih modelov za predlaganje in kategorizacijo polno zasede tudi 8 procesorskih jeder in poraba pomnilnika med izgradnjo lahko poskoči na več GB), se v praksi kliče samo v časih nizke porabe virov na strežniku.

Za pridobivanje predlogov predavanj so na voljo URLji:

- `/recommend/`*`id_predavanja`*`/`
- `/recommend/`*`ime_predlagalnika`*`/`*`id_predavanja`*`/`
- `/recommend/`*`ime_predlagalnika`*`/`*`id_predavanja`*`/true/`
  Pri teh vnosih se mora *`id_predavanja`* zamenjati z dejanskim številčnim IDjem predavanja, za katerega hočemo pridobiti predloge.

ime_predlagalnika pa se zamenja s kratkim imenom predlagalnika, ki naj predloge vrne (trenutno so to `dlc`, `hyb`, `bow` za vse tri opisane v 5.3).
Če je naslov zaključen s `true`, bo storitev vrnila izpis za razhroščevanje z vsemi podatki predavanj.

Snapshot of RecommendationRequest generated by ServiceStack on 11.9.2012 18:25:46

view json datasource from original url: http://localhost:19052/recommend/dlc/13317/true in other formats: json xml csv jsv

Status	OK
Lecture Id	13317
Lecture Title	Geometric Tools for Graph Mining of Large Social and Information Networks
Lecture Authors	Michael Mahoney,
Event Title	Tutorials
Model Updated	2012/09/11

Recommendations

Id	Title	Authors	Event Name	Similarity
9273	Large Graph-Mining: Power Tools and a Practitioner's Guide	Christos Faloutsos,Gary L Miller,Charalampos E. Tsourakakis,	Tutorials	3.44987151783492
13324	Mining Heterogeneous Information Networks	Jiawei Han,	Tutorials	3.35249033498451
14070	Structure is Informative: On Mining Structured Information Networks	Jiawei Han,	European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)	3.29916557545657
3877	Mining Large Graphs: Laws and Tools	Jure Leskovec,	The 18th European Conference on Machine Learning (ECML) and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)	3.24835634145971
5998	Graph Mining and Graph Kernels	Karsten Michael Borgwardt,Xifeng Yan,	The 14th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining	3.16087680896825
13293	Metric Forensics: A Multi-Level Approach for Mining Volatile Graphs	Keith Henderson,	Industry / Government Sessions	3.16056339498602
15832	Web Mining	Aris Gionis,Ricardo Baeza-Yates,	Twenty-Second International Joint Conference on Artificial Intelligence	3.15457705904929
9588	Scalable Link Mining and Analysis on Information Networks	Philip S. Yu,	ILP/MLG/SRL collocated International conferences/workshops on learning from relational, graph-based and probabilistic knowledge	3.14248382434889

Slika 10: Primer izpisa predlagalnika za razhroščevanje

Za kategorizacijo predavanj je potrebno poslati zahtevek POST na enega izmed naslednjih naslovov:

- `/categorize/`
- `/categorize/`*`ime_kategorizatorja`*`/`
- `/categorize/`*`ime_kategorizatorja`*`/true/`

Vsebina zahtevka POST je v enakem formatu kot pri klicu `/update/` - polja `Lectures`, `Authors`, `Categories` in `Edges`. *`Ime_kategorizatorja`* določa, kateri izmed naloženih kategorizatorjev bo vrnil rezultate za zahtevek. Na voljo so vsi opisani v tem delu; `rnd` - naključni, `bys` – naivni Bayesov, `knn` - kNN, `maxent` – z iskanjem največje entropije, `hie` – hierarhični klasifikator.

Kategorizator bo predlagal kategorije za vsako izmed predavanj poslanih v polju `Lectures`. Pri dekodiranju zahtevka se v polju `Edges` avtomatsko referencirajo obstoječi avtorji in predavanja, če niso poslani skupaj s tem zahtevkom. Zaradi tega so zahtevki lahko manjši.

Rezultat zahtevka je slovar v polju `Categories`, ki ima kot ključe IDje predavanj, vrednosti so pa slovarji predlogov. V slovarjih predlogov je ključ ID kategorije, vrednost pa je decimalna vrednost zaupanja klasifikatorja. V rezultatu je vključeno še ime klasifikatorja in točen čas in datum izgradnje klasifikatorjevega modela, na osnovi katerega so bili narejeni predlogi (primer 20).

Dodatek parametra `true` k URLju vključi način za razhroščevanje, ki namesto IDjev predavanj in kategorij vrne njihova polna imena in jih grafično prikaže (slika 11).


```
{
    "Status": "OK",
    "Categorizer": "MaxEnt",
    "ModelUpdated": "/Date(1347389346299+0000)/",
    "Categories": {
        "13317": {
            "16": 11.6446176613643,
            "27": 3.72559011084326,
            "28": 3.23005473895591,
            "29": 3.43006623475748,
            "36": 8.61728032446888
        }
   }
}
```


Primer 20: Rezultat klica kategorizatorja v formatu JSON

Snapshot of CategorizationRequest generated by ServiceStack on 11.9.2012 18:03:00

view json datasource **from original url:** http://localhost:19052/categorize/maxent/true **in other formats:** json xml csv jsv

Status	OK
Categorizer	MaxEnt
Model Updated	2012/09/11

Categorizations

Lecture Title	Categories

Geometric Tools for Graph Mining of Large Social and Information Networks

Category	Confidence
Top/Computer_Science/Machine_Learning	11.6446176613643
Top/Computer_Science/Data_Mining	8.61728032446888
Top/Computer_Science/Semantic_Web	3.72559011084326
Top/Computer_Science/Machine_Learning/Clustering	3.43006623475748
Top/Computer_Science/Network_Analysis	3.23005473895591

Innovation EverywhereâWhy the World Isnât Flat Enough

Category	Confidence
Top/Computer_Science/Machine_Learning	3.1221429139624
Top/Computer_Science/Semantic_Web	2.93671636387939
Top/History	2.69166275946376
Top/Computer_Science/Data_Mining	2.46661967384494
Top/Technology/Innovation	2.36561408350066

The Six Webs, 10 Years On

Category	Confidence
Top/Computer_Science/Semantic_Web	2.2955660972154
Top/Computer_Science/Machine_Learning	2.28157052914244
Top/Computer_Science	1.93493653579492
Top/Computer_Science/Data_Mining	1.82103130941256
Top/Computer_Science/Artificial_Intelligence	1.77850000732475

Slika 11: Primer izpisa kategorizatorja za razhroščevanje

5.5.2 Uporaba vmesnika

Glavna spletna stran (v našem primeru VideoLectures) mora za ohranjanje ažurnosti podatkov in ustrezno pridobivanje povratne informacije klicati spletni vmesnik. Klici bi se naj sprožali ob naslednjih dogodkih:

- **ob nalaganju strani predavanja** – klic `/recommend` za predloge podobnih predavanj. Rezultat klica se shrani v medpomnilnik.

- **ob dodajanju predavanja** – klic `/categorize` za predloge kategorij; klic `/update` ob shranjevanju za posodobitev podatkov storitve

- **ob brisanju predavanja** – klic `/update` za odstranitev podatkov

- **periodično; nekajkrat na dan, če je prišlo do sprememb** – brisanje medpomnilnika predlogov in klic `/rebuild`, da se ponovno zgradijo modeli predlagalnikov in kategorizatorjev

Ker se podatki na spletni strani VideoLectures redko posodabljajo (doda se kvečjemu nekaj predavanj enkrat do dvakrat na teden), prihaja do klicev za posodobitev precej redko. Posledično je tudi zaradi uporabe medpomnjenja obremenitev storitve nizka.

6 Meritve

Pri izbiri najustreznejšega klasifikatorja za podporo spletne storitve nas najbolj zanimajo naslednji parametri:

- kvaliteta rezultatov, ki jih klasifikator daje,
- čas izračuna modela,
- čas klasifikacije,
- poraba pomnilnika in CPUja med računanjem in med čakanjem na ukaz.

Pri izbiri algoritma za uporabo na spletni strani, kot je npr. VideoLectures, moramo upoštevati vse te parametre. Klasifikator, ki daje odlične rezultate ni uporaben, če mora uporabnik čakati dlje časa, kot bi lahko isto odločitev sprejel sam. Enako neuporaben je klasifikator, ki porabi več sistemskih sredstev, kot jih imamo na voljo. V nadaljevanju si bomo ogledali rezultate meritev prej omenjenih parametrov za implementacije klasifikatorjev, opisanih v 5. poglavju.

6.1 Metodologija in testna množica

Vse meritve so bile opravljene na sistemu s sledečimi specifikacijami:

- Intel Core i5-2500; 4 jedra pri 3.3 GHz s TurboBoost[10] frekvenco do 3.7 GHz,
- 12 GB DDR3 sistemskega pomnilnika,
- OCZ Vertex 3 MaxIOPS 128GB SSD za shranjevanje podatkov,
- 64-bitna različica operacijskega sistema Windows 8.

Za testiranje je bilo implementirano posebno stikalo izvršilne datoteke modula za kategoriziranje, ki je sprožilo 10-kratno križno preverjane. Točen postopek je bil:

1. Naloži JSON datoteke podatkovnega seta VideoLectures.net v SiteData objekt.
2. Ponovi 10x:
    1. Naključno razdeli predavanja na dve množici po načelu 90%/10%.
    2. Uporabi 90% predavanj za učenje modela klasifikatorja.
    3. Pridobi kategorije za preostalih 10% predavanj.
    4. Določi uspešnost klasifikacije.
3. Vrni izmerjene čase izgradnje modela, klasifikacije in povprečen rezultat uspešnosti klasifikacije.

Implementacija algoritma je prikazana v primeru 21.

[10] *Glej* [17]

Podatki za testiranje so bili zajeti s strani VideoLectures.net 8. septembra 2012 in so vsebovali

- 17757 predavanj
- 11590 avtorjev
- 548 kategorij

```
if (_autoSplit)
{
    var results = new List<double>();
    var modelTimings = new List<double>();
    var categTimings = new List<double>();

    for (int i = 0; i < EvaluationFolds; i ++ )
    {
       // Memory cleanup
       System.GC.Collect();

       categorizeLecIds = AutoSplit(siteData);
       var sw = new StopWatch();
       // Build categorizer model, passing test lecture IDs as ignore param
       categorizer.RecalculateModel(siteData, new Set<int>(categorizeLecIds));
       modelTimings.Add(sw.TotalMilliseconds);

       // Memory cleanup to keep memory results accurate
       System.GC.Collect();
       var lectures = categorizeLecIds.Select(id =>
                                              siteData.Lectures[id]).ToList();

       sw = new StopWatch();
       var categories = categorizer.GetCategoriesForLectures(lectures);
       categTimings.Add(sw.TotalMilliseconds);

       var ca = EvaluateResults(siteData, categories);
       logger.Info("Main", "Class. accuracy of categorizer {0} was {1}.",
                                                 categorizer.ToString(), ca);
       results.Add(ca);
    }

    var mean = results.Average();
    var evalTimes = modelTimings.Average();
    var categTimes = categTimings.Average();
    logger.Info("Main", "Results: {0}", String.Join(" ", results));
    logger.Info("Main", "Build times: {0}", String.Join(" ", modelTimings));
    logger.Info("Main", "Categorization times: {0}",
                                          String.Join(" ", categTimings));
    logger.Info("Main", "Average model build time {0:n} ms.", evalTimes);
    logger.Info("Main", "Average categorization time {0:n} ms.", categTimes);
    logger.Info("Main", "Mean classification accuracy after {0}-fold
                validation was {1:0.00}%.", EvaluationFolds,  mean * 100.0);
}
```

Primer 21: Implementacija ocenjevanja klasifikatorjev

Omembe vredno je še dejstvo, da je drevo kategorij visoko le pet ravni.
Dodatek A vsebuje tabelo vseh zbranih meritev.

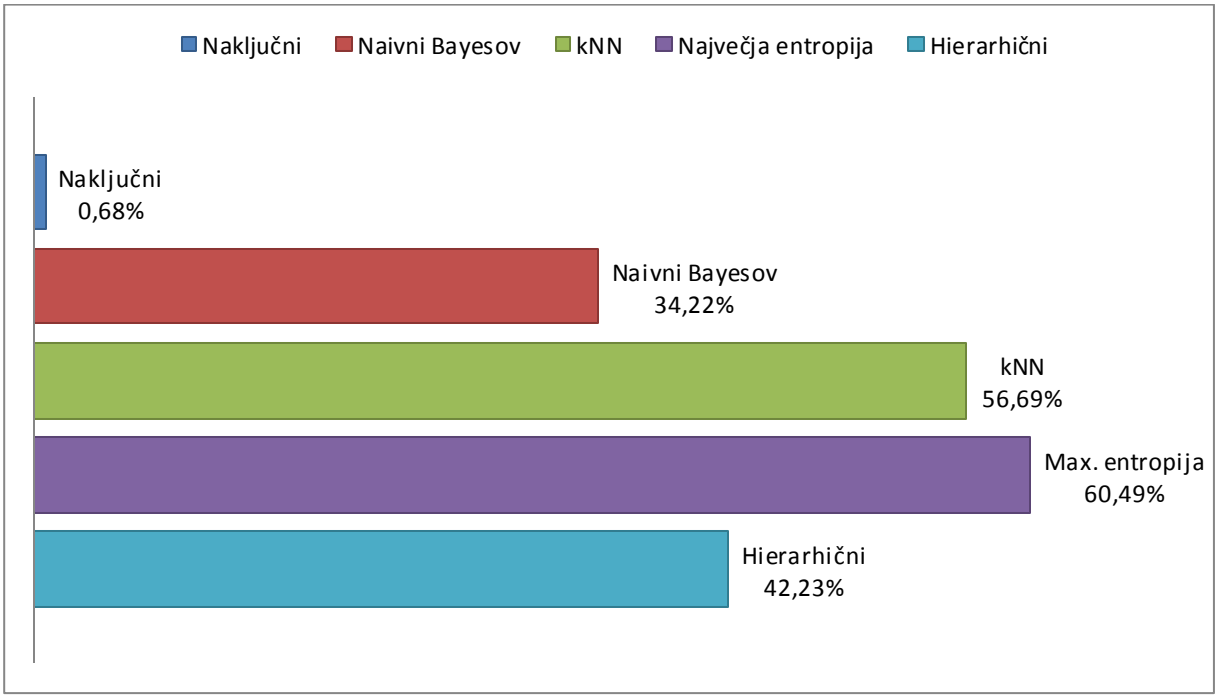
6.2 Učinkovitost kategorizatorjev

Učinkovitost kategorizatorjev je bila določena z obema metodama, razloženima v poglavju 4.4.

6.2.1 Klasifikacijska točnost

Klasifikacijska točnost je bila določena z algoritmom, opisanim v 4.4.1. Algoritem vrne rezultate v razponu med 0,0 in 1,0. Rezultat 1,0 pomeni, da so bile vsem predavanjem dodeljene vse pravilne kategorije in nobena napačna, rezultat 0,0 pa pomeni, da nobenemu predavanju ni bila dodeljena nobena pravilna kategorija.

Za večjo preglednost je rezultat predstavljen v odstotkih v grafu 1.



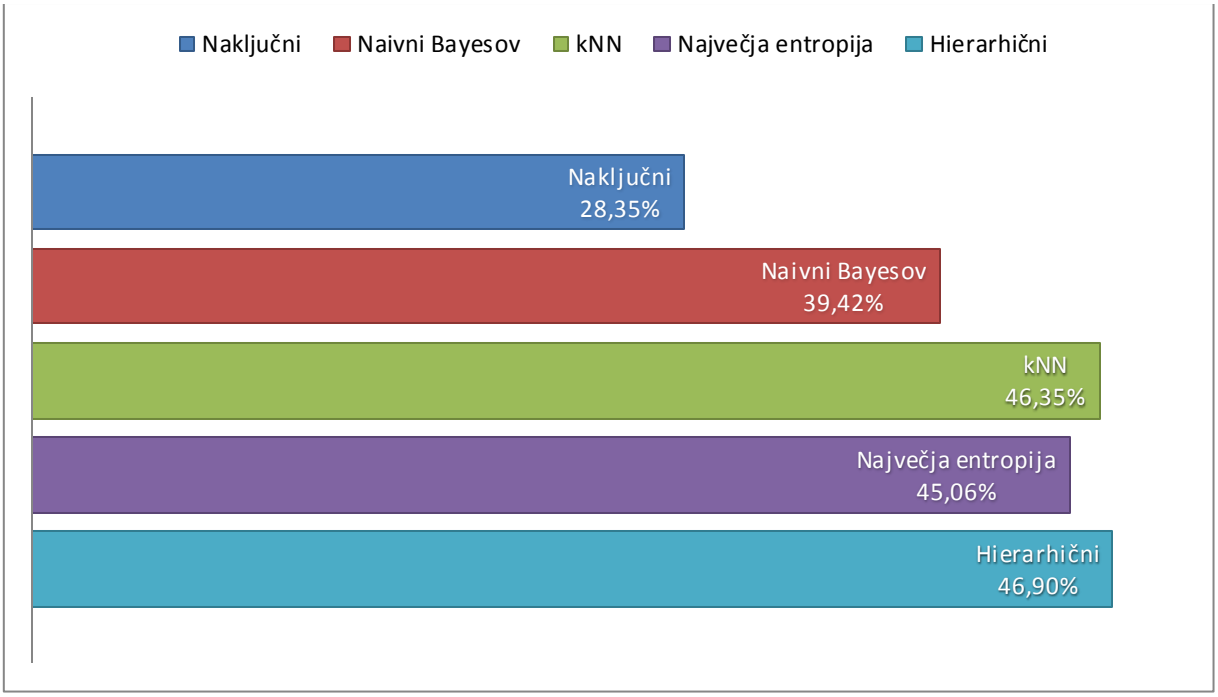
Graf 1: Rezultat meritev klasifikacijske točnosti

Iz rezultatov je razvidno, da se je daleč najboljše obnesel klasifikator z maksimalno entropijo, medtem ko hierarhični in naivni Bayesov klasifikator vidno zaostajata.

6.2.2 Ocena z upoštevanjem hierarhije

Klasifikatorji so bili ocenjeni še z algoritmom opisanim v razdelku 4.4.2. V implementaciji je bil Shenov α-faktor določen na vrednost 3.5, ki se je empirično izkazal kot parameter, ki uteži rezultate v prid točnih rezultatov in še ne degenerira testiranja v preprosto določanje klasifikacijske točnosti.

Kot prej, je rezultat tega algoritma ocena med 0,0 in 1,0, kjer 1,0 pomeni idealni klasifikator in 0,0 takega, ki klasificira vse kategorije vseh predavanj v največji možni oddaljenosti od pravilne. Rezultati so prikazani na grafu 2.

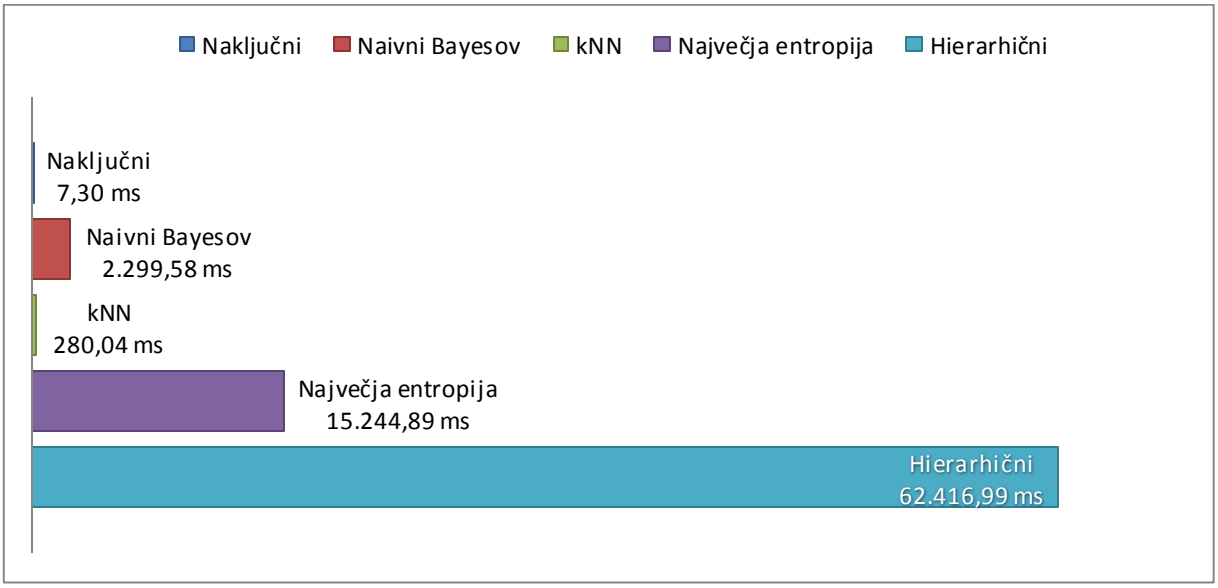


Graf 2: Rezultat meritev z upoštevanjem hierarhije

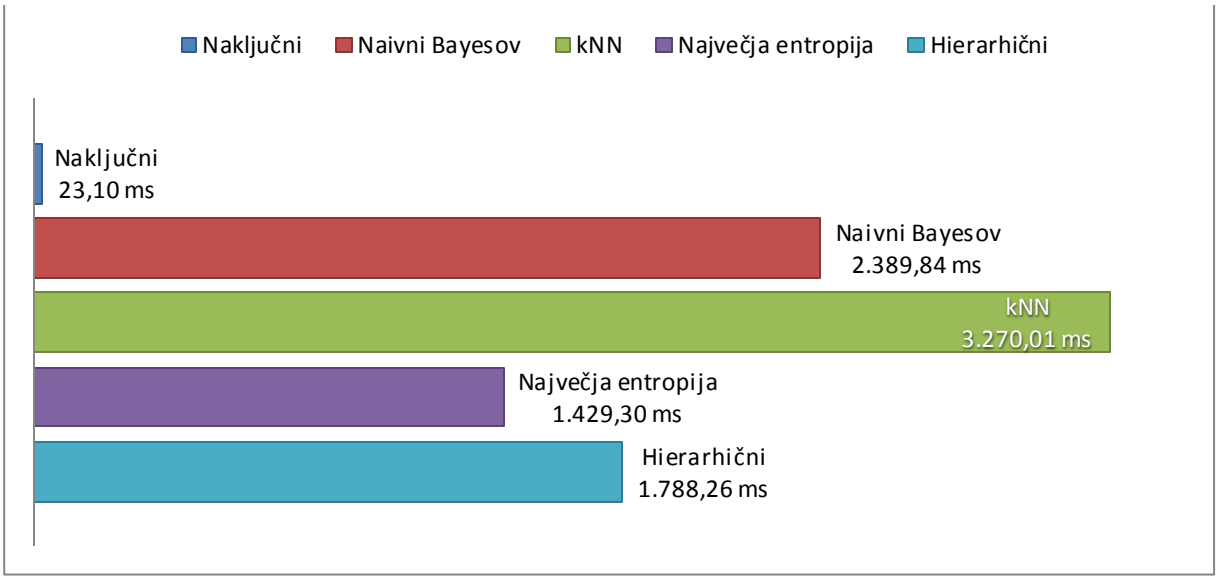
Rezultati so pričakovani. Ker ta način ocenjevanja povečuje rezultat tudi za netočno klasifikacijo, je rezultat naključnega klasifikatorja boljši. Boljše se obnese tudi hierarhični klasifikator, ki edini upošteva hierarhično strukturo drevesa kategorij in so zato zgrešene klasifikacije bližje pravim rezultatom kot pri drugih klasifikatorjih. Vseeno pa je razlika v učinkovitosti med najboljšimi tremi klasifikatorji minimalna.

6.3 Čas izgradnje modela in klasifikacije

Za algoritme je pomembno da so tudi hitri, saj uporabniki pričakujejo odzive v realnem času. Zato so bili za klasifikatorje izmerjeni časi izgradnje modela in časi klasifikacije testne množice. Časi izgradnje modelov so prikazani na grafih 3 in 4.



Graf 3: Časi gradnje modelov klasifikatorjev



Graf 4: Časi klasifikacije testne množice

Pri časih gradnje modela izstopata dva klasifikatorja – kNN in hierarhični. kNN klasifikator se izkaže kot izjemno hiter, kar gre na račun tega, da razen shranjevanja vektorjev ni potrebno postoriti ničesar drugega.
Pri hierarhičnem klasifikatorju pa je razvidno, da računanje več kot 1000 SVM modelov traja precej časa in je zato skoraj desetkrat počasnejši od drugega najpočasnejšega klasifikatorja.

Tudi pri časih klasifikacije pa so rezultati tudi precej jasni. Ker kNN klasifikator opravi večino dela pri klasifikaciji, je opazno počasnejši.

6.4 Poraba sistemskih virov

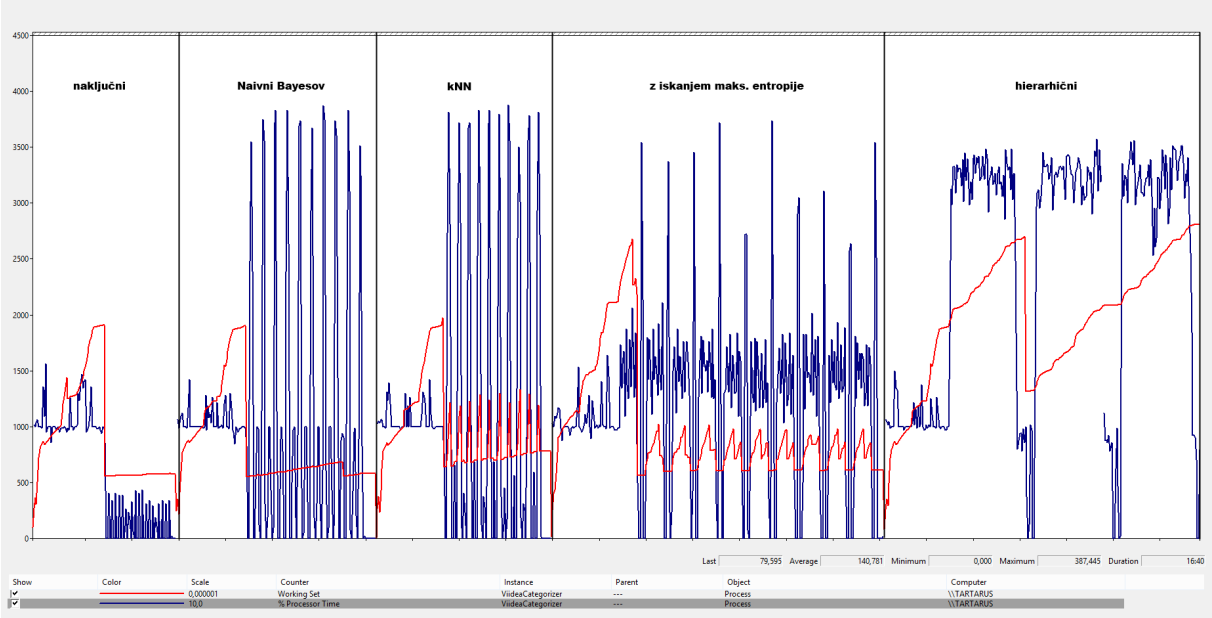
Poraba sistemskih virov je bila izmerjena s programom Performance Monitor, ki je nameščen skupaj z Windows 8. Izmerjena je bila uporaba CPU jeder in poraba fizičnega pomnilnika[11] med izgradnjo, čakanjem na kategorizacijo in kategorizacijo samo. Pred vsako izgradnjo modela in pred vsakim zagonom klasifikacije je eksplicitno pognan *garbage collector*. Poraba virov je prikazana v grafu 5, večji graf z več podrobnostmi pa je na voljo v dodatku B.

Z modro barvo je narisana uporaba jeder CPU v odstotkih v 10x skali (t.j. 4000 pomeni 400% izrabo jedra CPU, to je polno zasedenost testnega procesorja), z rdečo pa je narisana poraba pomnilnika v MB. Zaradi pomanjkanja prostora so pri hierarhičnem kategorizatorju prikazane le tri iteracije.

[11] t.i. *working set*, glej [18]

Iz grafa lahko razberemo nekaj vzorcev:

- Gradnja dokumentnih vektorjev porabi precej pomnilnika (približno 2GB) in se zaradi hkratnih dostopov do inverznega indeksa zelo slabo paralelizira. Polno zasedeno je le eno procesorsko jedro.
- Ko je izgradnja dokumentnih vektorjev končana in se zavržejo vsi nepotrebni podatki (ostanejo le IDji predmetov, povezave in dokumentni vektorji), poraba pomnilnika močno pade, objekt SiteData z vsemi potrebnimi podatki zaseda približno 500MB.



Graf 5: Poraba pomnilnika in procesorja (Dodatek B vsebuje večjo različico grafa)

- Izgradnja modela se pri vseh klasifikatorjev dobro paralelizira in procesorska jedra so skoraj polno obremenjena, kar potrjuje 10 visokih vrhov porabe CPUja pri vsakem kategorizatorju.
- Model hierarhičnega klasifikatorja porabi opazno več pomnilnika kot vsi ostali, klasifikator z iskanjem največje entropije in naivni Bayesov klasifikator sta pomnilniško najbolj ekonomična.
- Zaradi deljenega dostopa do inverznega indeksa zahteva izgradnja dokumentnega vektorja za klasificirano predavanje ekskluzivno zaklepanje. Pri klasifikaciji je zato obremenjeno kvečjemu eno jedro, kar kažejo nizki vrhovi porabe CPU med visokimi.
- Ker traja klasifikacija pri klasifikatorju z iskanjem maks. entropije dlje časa, se stopnja paralelizacije v tem primeru boljša in se obremenjenost jeder giblje okoli 200%.

Skupno lahko opazimo da se pri porabi procesorja vsi klasifikatorji obnesejo približno isto, prav tako pa so, z izjemo hierarhičnega klasifikatorja, približno enako gospodarni s pomnilnikom.

7 Zaključki

Cilj naloge je bil določiti in implementirati najboljši klasifikator, ki bi urednikom strani VideoLectures pomagal kategorizirati novo naložena predavanja. V ta namen smo si ogledali teoretično ozadje in C# implementacijo štirih klasifikatorjev in izmerili njihovo učinkovitost.

Na podlagi meritev, zbranih v 6. Poglavju, zaključujemo, da je za praktično uporabo na spletni strani VideoLectures najprimernejši klasifikator s k najbližjimi sosedi (kNN).

Za takšen zaključek izhaja iz večih opažanj:

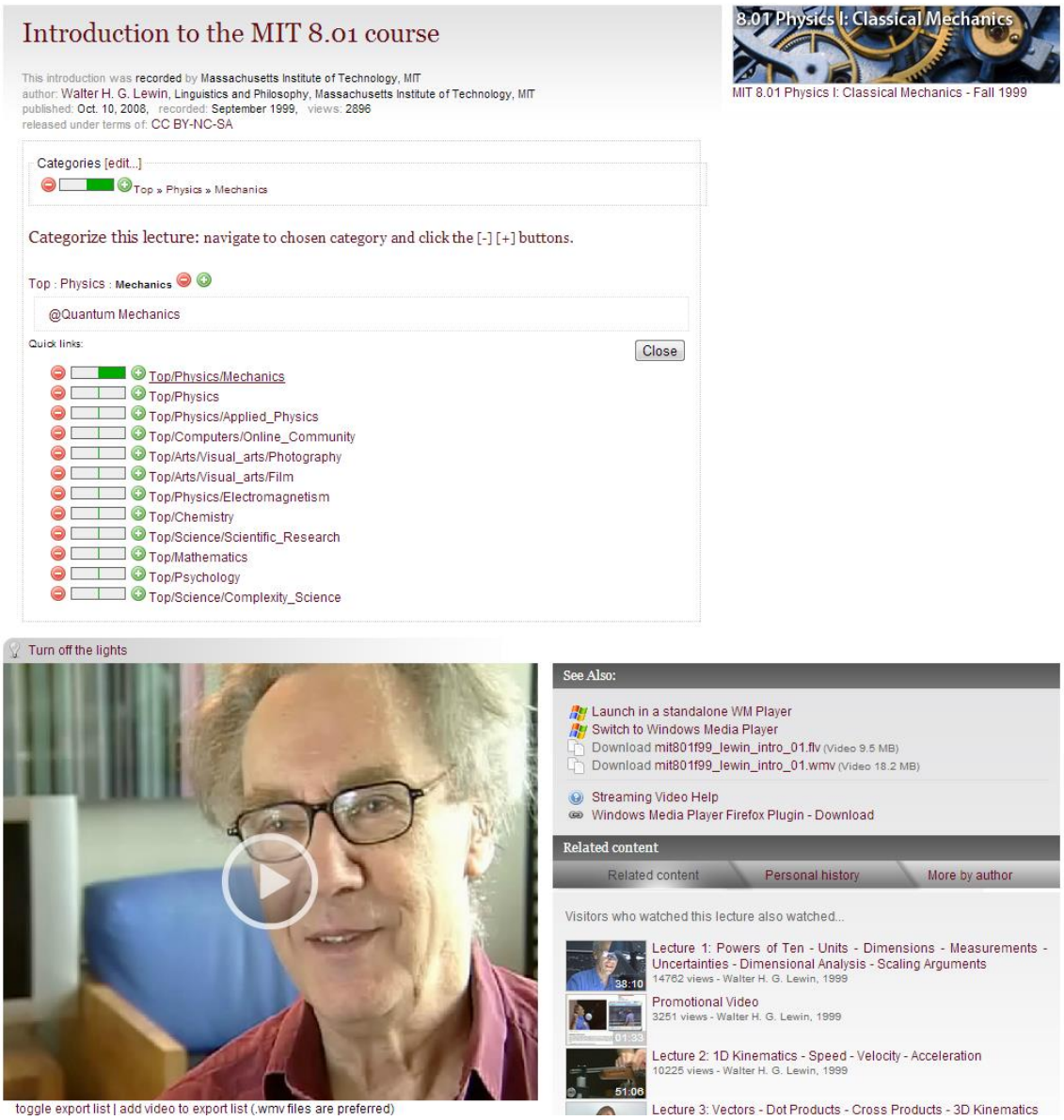
- kNN kategorizator se je dobro obnesel po obeh kriterijih ocenjevanja učinkovitosti. V nobenem primeru ni bil najboljši, je pa imel konsistentno dober rezultat.
- Med vsemi klasifikatorji je porabil najmanj časa za izgradnjo modela. S tako kratkim časom izgradnje modela se preračunavanje lahko dela v realnem času – takoj ko urednik doda novo predavanje.
- Po času klasifikacije je bil najhitrejši klasifikator z iskanjem največje entropije. Ker kNN klasifikator odloži večino računanja na čas klasifikacije, je bil najpočasnejši – klasifikacija približno 1.700 predavanj je trajala 3.2 sekunde. V praksi se izkaže, da klasifikacija enega predavanja traja približno 5ms, klasifikacija predavanja s klasifikatorjem z iskanjem največje entropije pa 3ms. Razlika za realno-časovno klasifikacijo torej ni bistvena.

Tehtnico v prid kNN klasifikatorja nagne predvsem izjemno hiter čas gradnje modela – pri tako kratkem času gradnje modela si lahko privoščimo ponovno izgradnjo modela po dodajanju vsakega predavanja. Ker uredniki tipično nalagajo predavanja v tematskih paketih (kar je posledica tega, da so v veliki večini posneta na tematskih konferencah), se lahko model klasifikatorja po določitvi kategorij za vsako predavanje sproti popravlja. Rezultat je opazno boljša uporabniška izkušnja, kljub objektivno slabšim rezultatom meritev.

7.1 Rezultat

Rezultat tega dela je implementacija spletne storitev za klasificiranje predavanj po podatkovnem modelu spletne strani VideoLectures.net. Implementacija trenutno na spletnem strežniku z operacijskim sistemom Linux teče v okolju Mono in pomaga urednikom kategorizirati nova predavanja.

Vmesnik, ki prikazuje rezultate klasifikatorja, je prikazan na sliki 12. Uredniku se prikaže seznam predlaganih kategorij, od katerih izbere primerne s klikom na zeleni gumb s simbolom »+«, očitno slabe predloge pa lahko negativno obteži z izbiro rdečega gumba s simbolom »-«.



Slika 12: VideoLectures.net; vmesnik za kategorizacijo predavanja

7.2 Ideje za nadaljnjo delo

Načrtovane nadgradnje storitve obsegajo:

1. Razširitev podatkovnega modela, da bo podpiral več strani, baziranih na platformi Viidea.

   Viidea platforma, ki poganja stran VideoLectures.net, namreč poganja tudi video arhive Kiberpipe[12], Hekovnika[13] in še nekaj konferenčnih strani. Ker so vsi ostali video arhivi opazno manjši (Kiberpipin, drugi največji, vsebuje

[12] http://video.kiberpipa.org/
[13] http://video.hekovnik.si/

približno 400 predavanj), je smiselno implementirati skupen model za klasifikacijo. Pri tem izziv leži v pravilnem vračanju kategorij, smiselnih za vsako spletno stran.

2. Izboljšanje modela napovedi
   Učinkovitost vseh klasifikatorjev, implementiranih v tej nalogi, je bila relativno nizka, uporaba uteži glede na tip informacije in uporaba morebitnih dodatnih informacij z drugih virov (prepis besedila s posnetka predavanja, Wikipedia strani povezane s temo itd.) bi lahko izboljšale napovedi.

3. Upoštevanje glasovanja urednikov
   Vmesnik za urednike omogoča glasovanje pri izbiri kategorije – poleg izbire lahko urednik izbere tudi negativni glas (glej sliko 12). Ti glasovi trenutno niso upoštevani, čeprav bi jih lahko klasifikator uporabil kot uteži pri vhodu.

4. Implementacija živega posodabljanje modela
   Trenutno ponovna izgradnja modela zahteva klic API funkcije, čeprav pri vseh vrstah klasifikatorjev to ni nujno potrebno – model je moč namreč širiti brez ponovnega preračunavanja vsega.

Prostora za nadgradnjo rešitve je še veliko, za določitev prioritet širjenja rešitve pa je trenutno predvsem potrebno zbrati in oceniti odziv uporabnikov in urednikov spletne strani.

8 Kazala

8.1 Kazalo slik

8.2 Kazalo primerov

8.3 Kazalo grafov

9 Bibliografija

[1] M. Porter, 2006. [Elektronski]. Available: http://tartarus.org/~martin/PorterStemmer/. [Poskus dostopa 12 junij 2012].

[2] Inštitut Jožefa Štefana, „LemmaGen," 2010. [Elektronski]. Available: http://lemmatise.ijs.si/. [Poskus dostopa 12 junij 2012].

[3] E. W. Weisstein, „Distance," [Elektronski]. Available: http://mathworld.wolfram.com/Distance.html.

[4] E. W. Weisstein, „Dot Product," [Elektronski]. Available: http://mathworld.wolfram.com/DotProduct.html.

[5] C. D. Manning, P. Raghavan in H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.

[6] H. B. Mitchell in P. A. Schaefer, „A "Soft" K-Nearest Neighbor Voting Scheme," *International Journal of Intelligent Systems,* str. 459-468, April 2001.

[7] K. Nigam, J. Lafferty in A. McCallum, „Using Maximum Entropy for Text Classification," v *IJCAI-99, the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[8] A. Sun in E.-P. Lin, „Hierarchical Text Classification and Evaluation," v *IEEE International Conference on Data Mining*, San Jose, CA, USA, 2001.

[9] C. Cortes in V. Vapnik, „Support-vector networks," *Machine Learning,* zv. 20, str. 273-297, 1995.

[10] T. Joachims, „Text categorization with support vector machines: learning with many relevant features," v *10th European Conference on Machine Learning*, Chemnitz, Germany, 1998.

[11] S. Nowak in H. Lukashevich, „Multilabel Classification Evaluation using Ontology Information," v *First ESWC Workshop on Inductive Reasoning and Machine Learning on the Semantic Web*, Heraklion, Greece, 2009.

[12] X. Shen, M. Boutell, J. Lou in C. Brown, „Multilabel machine learning and its application to semantic scene classification," v *Storage and Retrieval Methods and Applications for Multimedia*, 2004.

[13] A. D'yakonov, „Two Recommendation Algorithms Based on Deformed Linear Combinations," v *European Conference on Machine Learning and Principles*

and Practice of Knowledge Discovery in Databases, Athens, Greece, 2011.

[14] E. Spyromitros-Xioufis, E. Stachtiari, G. Tsoumakas in I. Vlahavas, „A Hybrid Approach for Cold-start Recommendations of Videolectures," v *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Athens, Greece, 2011.

[15] M. Grčar, „SvmLightLib," October 2009. [Elektronski]. Available: http://mihagrcar.org/svmlightlib/. [Poskus dostopa 4 September 2012].

[16] Service Stack, „ServiceStack, Opensource .NET and Mono REST Web Services framework," Service Stack, 2012. [Elektronski]. Available: http://www.servicestack.net/.

[17] Intel corporation, „Intel Turbo Boost Technology," [Elektronski]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html.

[18] Microsoft, „MSDN - Working Set," 7 September 2012. [Elektronski]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/cc441804(v=vs.85).aspx.

	Klasifikacijska natančnost											
												Povprečje
Naključni	**Rezultat**	0,00838	0,007	0,009009	0,006818182	0,004716981	0,00755	0,008547	0,004021	0,009451	0,002339	0,68%
	Čas izgradnje	11,0069	8,0046	7,001	7,0053	7,001	7,0047	6,0002	7,0007	7,0007	7,0013	7,40 ms
	Čas klasifikacije	26,9983	23,006	23,0029	23,0033	23,0101	22,0038	22,0084	24,0034	23,0085	22,0028	23,20 ms
Naivni Bayesov	**Rezultat**	0,33854	0,3612	0,3426573	0,333523701	0,308384918	0,35961	0,339932	0,333333	0,340843	0,364115	34,22%
	Čas izgradnje	2357,33	2328,3	2279,3238	2298,3268	2287,3275	2272,32	2288,325	2264,354	2267,321	2305,335	2.294,83 ms
	Čas klasifikacije	2320,33	2391,3	2198,3117	2322,3293	2248,3185	2387,34	2249,319	2295,327	2322,33	2397,34	2.313,23 ms
kNN	**Rezultat**	0,56987	0,5661	0,577765	0,58489426	0,565019875	0,5663	0,568261	0,545812	0,561804	0,562937	56,69%
	Čas izgradnje	259,051	259,04	250,0355	252,0391	259,0377	324,046	255,0368	365,0512	255,0368	347,048	282,54 ms
	Čas klasifikacije	4345,62	3320,5	3163,4496	3167,4492	3064,4352	3282,47	3051,432	3080,438	3283,466	3107,441	3.286,67 ms
Največja entropija	**Rezultat**	0,61073	0,6072	0,6028329	0,614619883	0,593078759	0,59932	0,607735	0,61756	0,609994	0,586047	60,49%
	Čas izgradnje	16751,9	15004	15190,638	14704,6275	15071,0398	15336,7	15357,77	14868,44	15059,17	14942,92	15.228,67 ms
	Čas klasifikacije	1513,21	1414,2	1484,211	1400,1969	1377,1945	1398,2	1450,205	1417,201	1467,208	1411,2	1.433,30 ms
Hierarhični	**Rezultat**	0,43545	0,4196	0,4280045	0,42865463	0,404317386	0,41593	0,417867	0,420992	0,42312	0,429476	42,23%
	Čas izgradnje	57630,4	55643	54272,798	56082,0061	54826,7815	56691	53711,69	53827,75	54276,78	56441,01	55.340,34 ms
	Čas klasifikacije	1710,25	1719,2	1737,2484	1727,246	1762,2498	1701,24	1776,259	1774,253	1760,255	1815,257	1.748,35 ms
		Hierarhični evaluator										
												Povprečje
Naključni	**Rezultat**	0,28101	0,2702	0,2793048	0,274644148	0,294372646	0,29124	0,283147	0,283526	0,291611	0,285726	28,35%
	Čas izgradnje	12,0069	6,0006	6,0009	7,0007	7,001	7,001	7,0016	7,0013	7,0013	6,0006	7,20 ms
	Čas klasifikacije	27,9991	21,002	23,0026	22,0031	23,003	22,0031	22,0028	23,0029	23,0032	23,0033	23,00 ms
Naivni Bayesov	**Rezultat**	0,39221	0,3824	0,383298	0,398025374	0,406401306	0,3907	0,397057	0,401667	0,392516	0,397569	39,42%
	Čas izgradnje	2348,33	2348,3	2276,3237	2269,3224	2287,3154	2355,33	2322,335	2284,324	2275,329	2276,323	2.304,33 ms
	Čas klasifikacije	2644,37	2317,3	2309,3271	2245,3184	2744,3892	2625,37	2739,383	2291,325	2258,314	2489,353	2.466,45 ms
kNN	**Rezultat**	0,46808	0,4665	0,4672656	0,455283135	0,459350402	0,46398	0,457082	0,46391	0,458207	0,475699	46,35%
	Čas izgradnje	273,102	262,04	272,0386	248,0361	260,0388	251,036	260,0366	344,05	348,0605	257,0362	277,55 ms
	Čas klasifikacije	4404,62	3148,4	3156,4474	3170,4494	3137,4431	3149,45	3010,428	3171,45	3120,431	3064,435	3.253,36 ms
Največja entropija	**Rezultat**	0,45921	0,4394	0,4385994	0,452918769	0,46152325	0,45821	0,453917	0,453551	0,446964	0,441344	45,06%
	Čas izgradnje	15104,4	15851	14917,627	15548,1677	15380,7034	15110,4	15195,41	15151,13	15149,15	15202,79	15.261,12 ms
	Čas klasifikacije	1432,2	1403,2	1499,2131	1407,1954	1400,1981	1416,2	1455,207	1387,198	1428,202	1424,202	1.425,30 ms
Hierarhični	**Rezultat**	0,47181	0,4726	0,4772905	0,463920887	0,477492803	0,47775	0,464918	0,469305	0,454295	0,460309	46,90%
	Čas izgradnje	68786,2	66756	65941,024	68974,5938	66676,8481	72154,5	74835,67	69702,44	70344,85	70763,89	69.493,65 ms
	Čas klasifikacije	1912,27	1788,3	1808,2567	1784,2532	1901,2757	1766,26	1829,269	1818,263	1881,273	1792,254	1.828,16 ms

Dodatek B: Graf porabe CPU in pomnilnika