Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01683/2010
Datum: 01.09.2010

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

Kandidat: **JANEZ KRANJC**

Naslov: **IZDELAVA SPLETNE APLIKACIJE ZA STROJNO UČENJE**
DEVELOPMENT OF WEB APLICATION FOR MACHINE LEARNING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Cilj diplomskega dela je implementacija sistema za strojno učenje, ki deluje kot spletna aplikacija. V delu naj kandidat predstavi pomembnejše obstoječe sisteme, ki so implementirani kot samostojne, namestitvene aplikacije s poudarkom na sistemu Orange4WS, ki za posamezne algoritme in operacije uporablja spletne servise. Cilj diplomskega dela je ustvariti aplikacijo, ki uporablja iste principe kot Orange4WS z dodatnimi razširitvami, kot so delovanje na večih platformah in napravah (osebni računalnik, mobilni telefon) in enostavna razširljivost. Izdelan program naj vsebuje uporabniški vmesnik za različne platforme, naj omogoča uporabo spletnih servisov kot gradnikov procesa strojnega učenja in operacije nad delovnimi tokovi, ki jih bo mogoče z aplikacijo ustvariti. Kandidat naj podrobno predstavi postopek, s katerim se nek algoritem strojnega učenja pretvori v spletni servis za uporabo v aplikaciji.

Mentor:

prof. dr. Igor Kononenko

Dekan:

prof. dr. Franc Solina

Somentor:

prof. dr. Nada Lavrač

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Janez Kranjc

Izdelava spletne aplikacije za strojno učenje

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Igor Kononenko
Somentorica: prof. dr. Nada Lavrač

Ljubljana, 2010

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Janez Kranjc,

z vpisno številko 63040076,

sem avtor diplomskega dela z naslovom:

Izdelava spletne aplikacije za strojno učenje

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Igorja Kononenka
  in somentorstvom
  prof. dr. Nade Lavrač
- 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 20.9.2010 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju prof. dr. Igorju Kononenku in somentorici prof. dr. Nadi Lavrač za vse nasvete in pomoč pri izdelavi diplomske naloge.

Zahvaljujem se Vidu Podpečanu za vse ideje, ki so mi pomagale pri izdelavi spletne aplikacije.

Kazalo

Seznam uporabljenih kratic in simbolov

Ajax - *Asynchronous JavaScript and XML*
API - *Application Programming Interface*
CLI - *Command Line Interface*
CSS - *Cascading Style Sheets*
DOM - *Document Object Model*
HTML - *Hypertext Markup Language*
HTTP - *Hyper Text Transfer Protocol*
PHP - *PHP: Hypertext Preprocessor*
SOAP - *Simple Object Access Protocol*
WSDL - *Web Services Description Language*
WSRF - *Web Services Resource Framework*
XHTML - *Extensible Hypertext Markup Language*
XML - *Extensible Markup Language*

Seznam prevedenih izrazov

Application programming interface - *Vmesnik za programiranje*
Binding - *Vezava*
Command line interface - *Ukazna vrstica*
Computing node - *Izvajalno vozlišče*
Design perspective - *Oblikovalska perpsektiva*
Document object model - *Objektni model dokumenta*
Event listener - *Poslušalec na dogodke*
Header - *Glava*
Java applet - *Spletna javanska aplikacija*
Java virtual machine - *Virtualni stroj Java*
Pipeline - *Cevovod*
Plugin - *Vtičnik*
Remote procedure call - *Klic oddaljene procedure*
Result perspective - *Perspektiva z izidi*
Selector - *Izbirni ukaz*
Storage node - *Podatkovno vozlišče*
Widget - *Grafični gradnik*
Workflow - *Delotok*
Wrapper - *Ovojnica*

Povzetek

V diplomskem delu je opisana implementacija sistema za strojno učenje, ki deluje kot spletna aplikacija. V delu so najprej opisani že obstoječi sistemi, ki so implementirani kot samostojne, namestitvene aplikacije. Poseben zgled za izdelavo spletne aplikacije je Orange4WS, ki za posamezne algoritme in operacije uporablja spletne servise. Za razvoj spletne aplikacije sta bili izbrani spletni tehnologiji JavaScript in PHP, ki v povezavi z označevalnim jezikom HTML in stilsko predlogo CSS tvorita ogrodje spletne aplikacije. Opisan je razvoj spletne aplikacije, ki deluje v vseh (v času pisanja) širše uporabljenih brskalnikih in omogoča gradnjo in izvajanje delotokov, katerih gradniki so lahko tudi spletni servisi. Predstavljeni so primeri uporabe za uporabnike, navodila za namestitev in postopek razširitve aplikacije s poljubnimi algoritmi in dodatki. V zaključku so predstavljene ideje za nadaljne delo, ki vsebujejo meta učenje nad delotoki in generalizacijo aplikacije na druga področja, kjer je koristna uporaba delotokov.

Ključne besede:

strojno učenje, delotok, spletna aplikacija, spletni servisi

Abstract

The thesis describes the development of a web application for machine learning and data mining. Other stand-alone applications for machine learning are described and used as a basis for the development of the web application. Orange4WS serves as a special example as it is utilizing web services for various algorithms and operations. JavaScript and PHP have been selected, along with the markup language HTML and styling language CSS, as enabling technologies for the web application. The web application is developed for all (at the time of writing) popular web browsers and supports building workflows with web services as its elements. The case studies for users are presented along with instructions for the installation of the system to a web server and instructions for expanding the web application by including various algorithms and add-ons. Ideas which include meta-learning on workflows and the generalization of the web application to suit other areas of expertise where the utilization of workflows is sensible, are explained in the conclusion.

Key words:

machine learning, data mining, workflow, web application, web services

Poglavje 1

Uvod

Okolja za strojno učenje so aplikacije, ki služijo za zajem podatkov, analizo podatkov in izvajanje raznih algoritmov za strojno učenje nad podatki. Vse splošno uporabljene aplikacije za strojno učenje so implementirane kot samostojne namestitvene aplikacije. Z razvojem internetnih tehnologij se je porodila ideja o aplikaciji za strojno učenje, ki bi bila uporabnikom dostopna z internetnega brskalnika. Prednost takšne aplikacije je v tem, da jo uporabnik lahko uporablja brez namestitve, s kateregakoli sistema (osebni računalniki, mobilni telefoni, igralne konzole), ki zadošča zahtevam (dostop do interneta in dovolj sposoben internetni brskalnik). Za analizo podatkov, pridobljenih z interneta, je aplikacija, katere algoritmi se izvajajo na spletnem strežniku, zelo primerna, saj omogoča uporabo tudi na manj zmogljivih napravah.

Prvi sklop diplomske naloge obravnava obstoječe, najbolj razširjene aplikacije za strojno učenje. Njihov pregled je pomemben za določitev dobrih lastnosti, ki smo jih želeli vključiti v naše okolje in za odkritje slabih lastnosti, razvoju katerih smo se želeli izogniti. Pregled vsebuje aplikacije Weka, RapidMiner, KNIME in Orange ter dve razširitve za te aplikacije. To sta Weka4WS in Orange4WS, ki razširjata aplikacije do te mere, da se lahko v njih uporabljajo spletni servisi. Pregled je v veliki meri osredotočen na uporabniški vmesnik, saj je glavni del diplomske naloge prav izdelava takšnega uporabniškega vmesnika, kot ga v aplikacijah za strojno učenje še ni.

Zastavili smo si zahtevo, da je aplikacija za strojno učenje dostopna z internetnega brskalnika, zato smo morali poseči po trenutno aktualnih tehnologijah za spletno programiranje, ki so podrobno opisane v tretjem poglavju. Za izdelavo aplikacije smo uporabili označevalni jezik HTML, stilsko predlogo CSS, skriptna jezika JavaScript in PHP ter komunikacijski jezik SOAP za uporabo spletnih servisov.

V četrtem poglavju je podrobno opisan postopek razvoja spletne aplikacije. Spletno aplikacijo smo logično razdelili na več delov, razvoj in podrobno delovanje vsakega od njih pa smo podrobno opisali. Aplikacijo smo razdelili glede na prostor izvajanja: na uporabnikovem računalniku, na strežniku ali na strežnikih, ki gostujejo spletne servise. Najdaljšega opisa je deležen del, ki se izvaja na uporabnikovem računalniku, saj je tudi najobsežnejši.

Uporabniki spletne aplikacije se delijo v dve skupini: navadni uporabniki in razvijalci. Navadni uporabniki so tisti uporabniki, ki bodo želeli uporabljati funkcionalnosti spletne aplikacije, razvijalci pa so tisti uporabniki, ki želijo razširiti funkcionalnosti sistema do takšne mere, da ustreza njihovim potrebam. Primeri uporabe za obe vrsti uporabnikov so opisani v petem poglavju.

V zaključnem sklopu diplomskega dela naredimo pregled zahtevanih funkcij in navedemo ideje za nadaljnje delo in razširitev aplikacije. Med ideje spadajo meta učenje nad delotoki in analiza podatkovnih tokov.

Poglavje 2

Pregled obstoječih okolij za strojno učenje

Okolja za strojno učenje so vrsta programske opreme, s pomočjo katere lahko uporabnik izvaja algoritme za strojno učenje ter obdelavo in vizualizacijo podatkov. Zasnovane so tako, da uporabnik s pomočjo različnih (skriptnih) programskih jezikov [1] ali pa s pomočjo grafičnega vmesnika [1, 4] oblikuje postopek strojnega učenja, ki ga aplikacije izvajajo. V kolikor za oddaljeno izvajanje algoritmov ni posebej poskrbljeno, celotno izvajanje algoritmov poteka na računalniku, na katerem je okolje nameščeno.

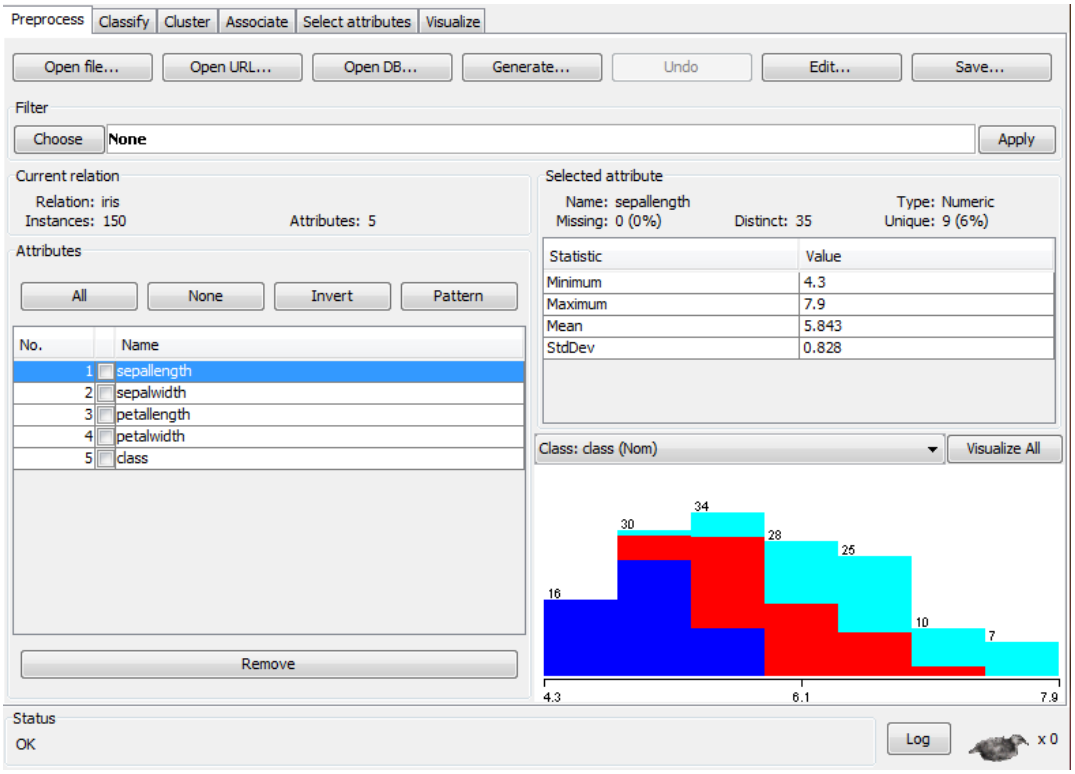
2.1 Okolja za strojno učenje

Za namen razvoja spletne aplikacije smo podrobno pregledali delovanje različnih okolij za strojno učenje. Njihove dobre lastnosti smo uporabili kot ideje pri načrtovanju spletne aplikacije in se poskušali čimbolj izogibati slabostim, ki smo jih pri okoljih odkrili.

2.1.1 Weka

Razvoj okolja za strojno učenje Weka (*Waikato Environment for Knowledge Analysis*) se je začel leta 1993 na Univerzi v Waikatu. Okolje je v celoti implementirano v programskem jeziku Java, zato deluje na vseh sistemih, kjer deluje virtualni stroj Java (*ang. Java Virtual Machine*, s kratico Java VM).

Weka podpira različne standardne algoritme za klasifikacijo, regresijo, razvrščanje, vizualizacijo in obdelavo podatkov. Do teh algoritmov lahko dostopamo preko štirih različnih uporabniških vmesnikov, ki jih nudi okolje Weka.



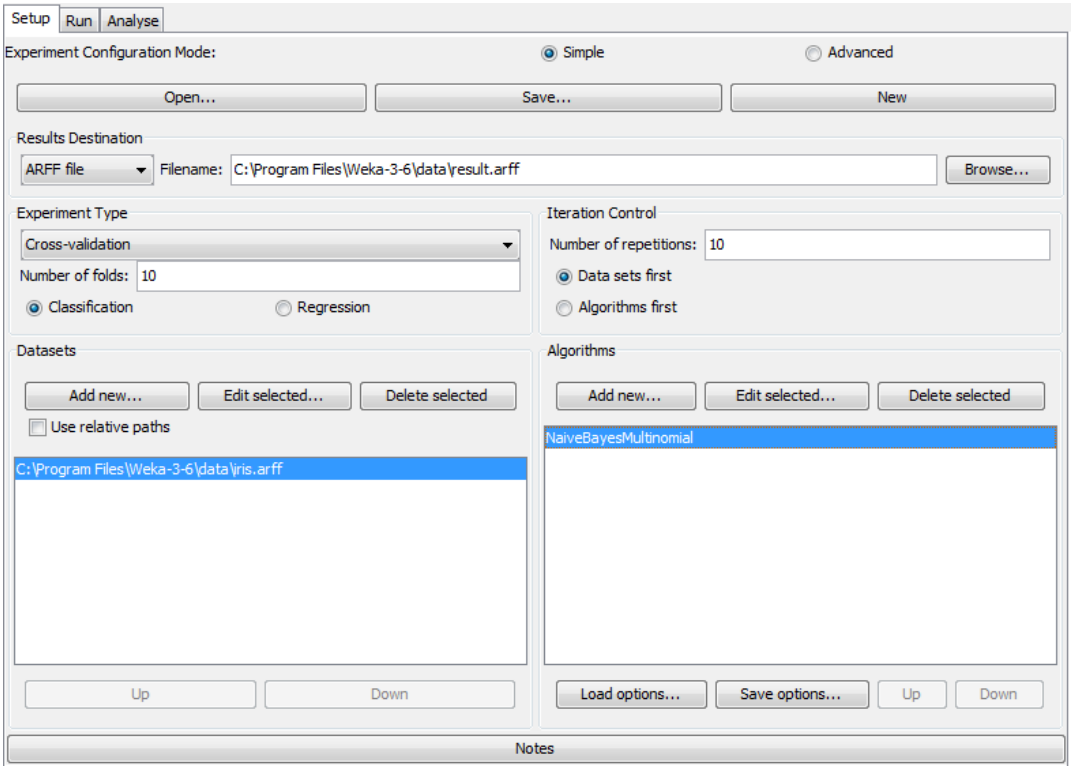
Slika 2.1: Zaslonska maska vmesnika Weka Explorer okolja Weka.

Weka Explorer je grafični uporabniški vmesnik, v katerem je omogočeno generiranje podatkov in nalaganje podatkov z datotek, podatkovnih baz in oddaljenih datotek s spleta. Na teh podatkih lahko izvajamo različne operacije, ki jih izbiramo s pomočjo zavihkov v grafičnem uporabniškem vmesniku (prikazanem na sliki 2.1).

Weka Experimenter je vmesnik za evalvacijo in analizo algoritmov za strojno učenje. Z njegovo pomočjo enostavno ocenjujemo različne algoritme z različnimi metodami evalvacije na eni ali več množicah podatkov (prikaz na sliki 2.2).

Uporabniški vmesnik *Simple CLI* (preprosta ukazna vrstica) omogoča izvajanje algoritmov z vpisovanjem ukazov v ukazno vrstico. Kljub koristnosti takšnega uporabniškega vmesnika za naše novo spletno okolje nismo razvijali, saj je primeren le za izkušene uporabnike.

Uporabniški vmesnik *KnowledgeFlow* omogoča grafično sestavljanje algoritmov v delotok, ki ga okolje izvede. V zgornjem delu grafičnega vmesnika se nahaja širok nabor algoritmov, ki jih nanašamo na delotok. Algoritme med seboj povezujemo z desnim klikom na njihove ikone, kjer izberemo vrsto podatka,



Slika 2.2: Zaslonska maska vmesnika Weka Experimenter okolja Weka.

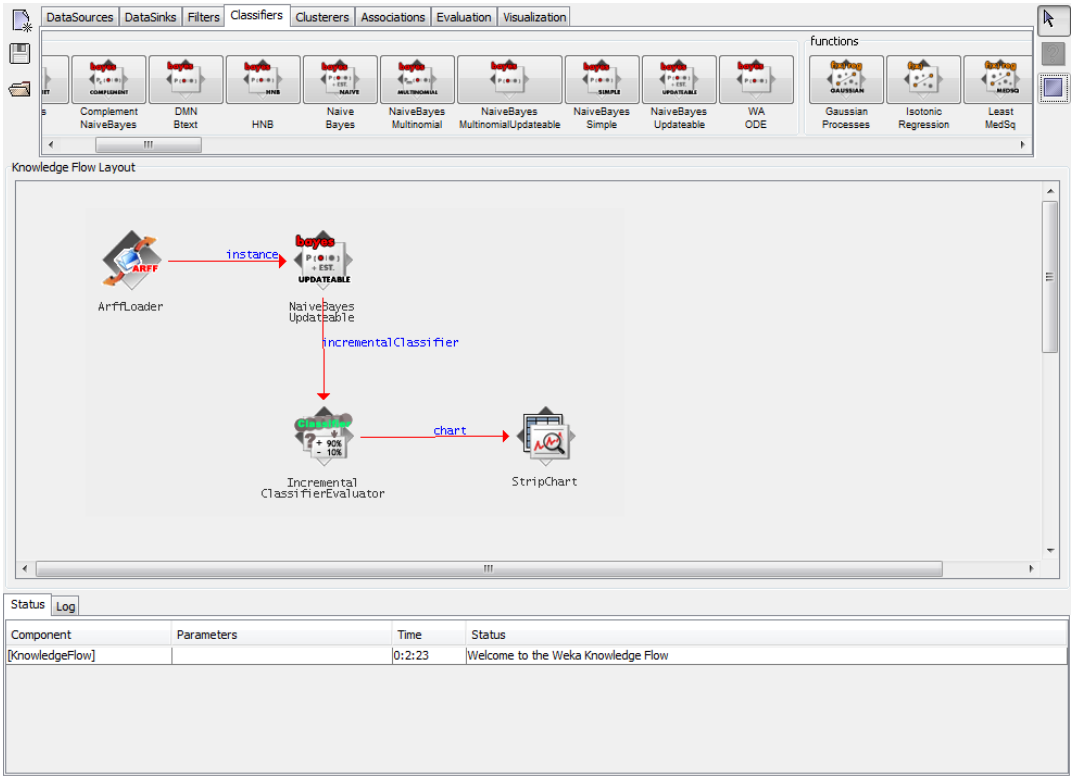
ki jo želimo povezati na nek drug algoritem. Uporabniku le s pregledovanjem ikon algoritmov ni takoj znano, kakšne izhode ponuja nek algoritem in kakšne vhode sprejema. Pri načrtovanju in razvoju spletnega okolja smo zato težili k temu, da so na delotoku vidni vsi vhodi in izhodi.

Dobra lastnost vmesnika KnowledgeFlow je vzporedno izvajanje različnih algoritmov. Ker smo v našem spletnem okolju izkoristili možnost uporabe spletnih servisov, je takšna funkcionalnost pri našem spletnem okolju zelo pomembna.

KnowledgeFlow nudi vse funkcionalnosti že prej omenjenih vmesnikov, zato smo za spletno okolje načrtovali podoben uporabniški vmesnik, ki nudi povezovanje algoritmov v delotoke in izvajanja teh delotokov.

2.1.2 RapidMiner

RapidMiner je odprtokodni sistem za strojno učenje. Pod imenom YALE (*ang. Yet Another Learning Environment*) je bil razvit na Oddelku za umetno

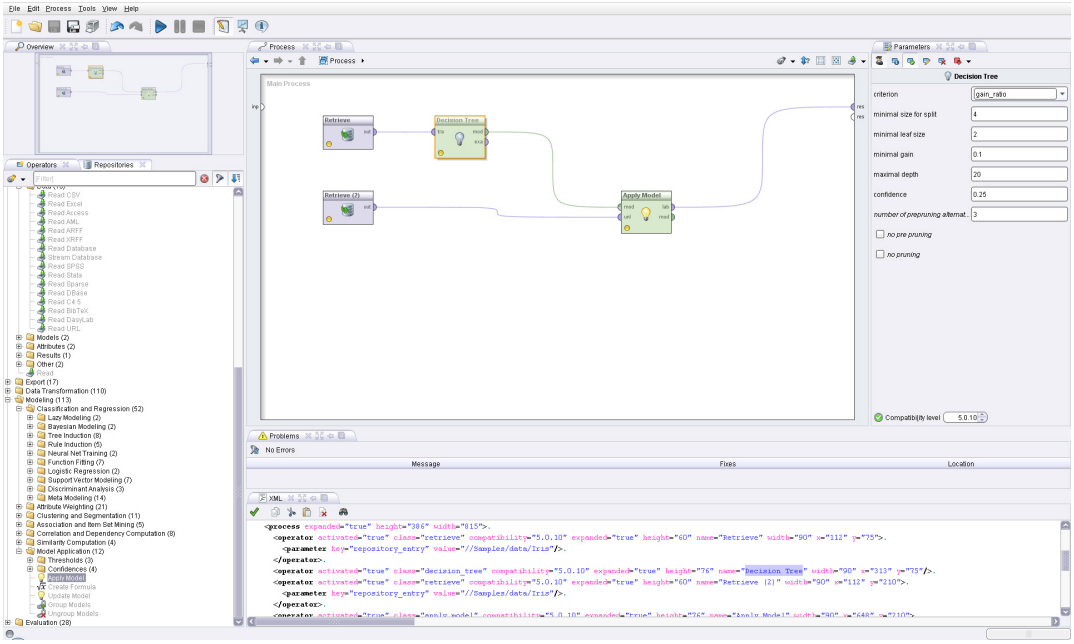


Slika 2.3: Zaslonska maska vmesnika KnowledgeFlow okolja Weka.

inteligenco na Univerzi v Dortmundu leta 2001. Uporablja se lahko bodisi kot samostojna aplikacija, bodisi kličemo njene funkcije iz javanskega programa (npr. razširimo lastno aplikacijo, da vsebuje funkcionalnosti RapidMiner). Okolje je v celoti napisano v programskem jeziku Java, kar pomeni, da deluje na vseh sistemih, kjer deluje virtualni stroj Java (*ang. Java Virtual Machine*, s kratico Java VM).

Grafični uporabniški vmesnik ima dve t.i. perspektivi. V oblikovalski perspektivi (*ang. design perspective*) gradimo delotok, v perspektivi z izidi (*ang. result perspective*) pa lahko vidimo tiste izide in podatke, ki jih v oblikovalski perspektivi eksplicitno izberemo za prikaz. V aplikaciji sta strogo ločena prikaz in obdelava podatkov.

Oblikovalska perspektiva (slika 2.4) vsebuje hierarhično urejene operatorje, ki so gradniki delotokov. Z miško prenesemo operatorje na delotok (v žargonu aplikacije imenovan proces) in jih s klikanjem na vhode in izhode povezujemo med seboj. Med gradnjo procesa lahko tudi urejamo in pregledujemo sam proces v obliki XML (v tej obliki ga aplikacija tudi shranjuje in nalaga). Ob



Slika 2.4: Zaslonska maska oblikovalske perspetkive okolja RapidMiner s primerom delotoka.

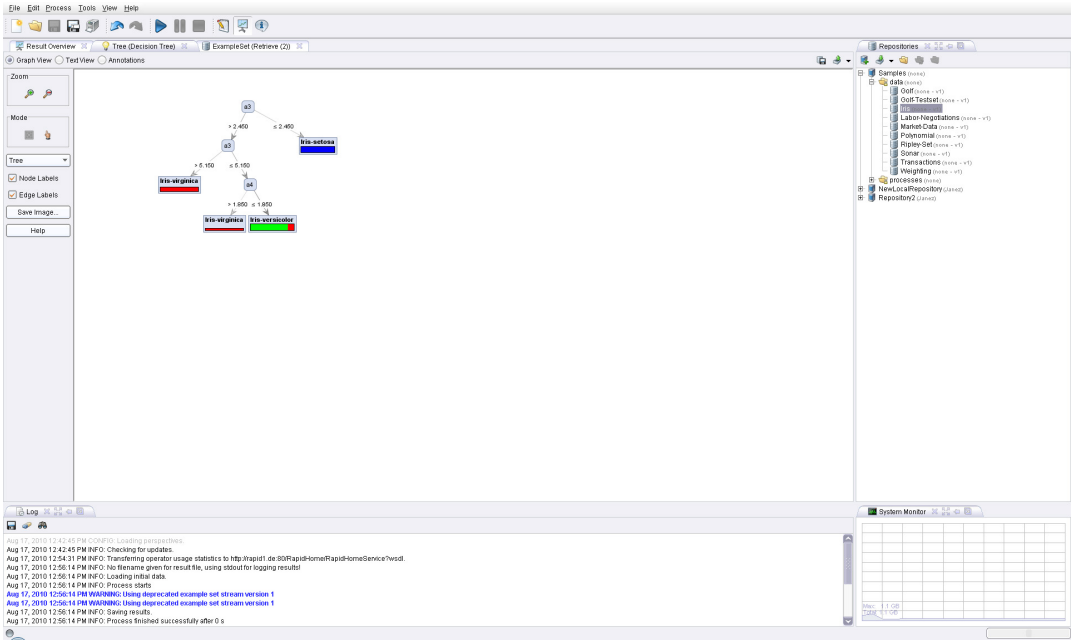
zagonu procesa program izvede proces, ki je zapisan v obliki XML in grafično prikaže izide v perspektivi za izide (slika 2.5).

Osnovni gradniki procesa so operatorji. Okolje RapidMiner vključuje več kot petsto operatorjev. Ti se delijo v naslednje skupine:

- algoritmi za strojno učenje,
- operatorji za predprocesiranje podatkov,
- meta operatorji,
- operatorji za uvoz in izvoz podatkov,
- operatorji za nadzor procesa.

Ugotovljena slabost velikega števila operatorjev je ta, da so nekateri operatorji preveč enostavni in neuporabni (sami po sebi ne naredijo ničesar koristnega). Za dobro rabo takih operatorjev je treba zgraditi zelo kompleksen proces z uporabo operatorjev za nadzor procesa (zanke in pogoji). Manjši nabor kompleksnejših operatorjev bi uporabniku poenostavil izbiro.

Ob povezovanju operatorjev v proces aplikacija preverja, ali so podatki skladni. Ob morebitni napaki nas program opozori in ponudi različne rešitve, ki uspešno rešijo problem. Ta rešitev je običajno dodajanje novega operatorja, ki podatke ustrezno preuredi v podatke, ki jih sprejema vhod nekega drugega operatorja.



Slika 2.5: Zaslonska maska perspektive z izidi okolja RapidMiner s primerom odločitvenega drevesa.

V oblikovalski perspektivi so na desni strani okna za gradnjo procesa vhodi, na katere lahko povežemo različne vrste podatkov, ki nam jih program prikaže v perspektivi za izide. Različne vrste podatkov lahko prikazujemo na različne načine (tekstovno, z grafi).

Proces lahko zaženemo s pomočjo že opisanega grafičnega uporabniškega vmesnika, s pomočjo programa v ukazni vrstici, ki kot vhod sprejme opis procesa XML ali pa s pomočjo javanske funkcije, ki jo vključimo v naš program.

Javanski vmesnik za programiranje (*ang. application programming interface*, s kratico API) lahko uporabimo bodisi tako, da naložimo proces v obliki XML in ga zaženemo, bodisi sami zgradimo proces s pomočjo objektov in krmilimo podatkovni tok.

Okolje ima zelo intuitiven uporabniški vmesnik z veliko pomočmi, kar uporabniku močno olajša učenje in uporabo. To lastnost smo poskušali v

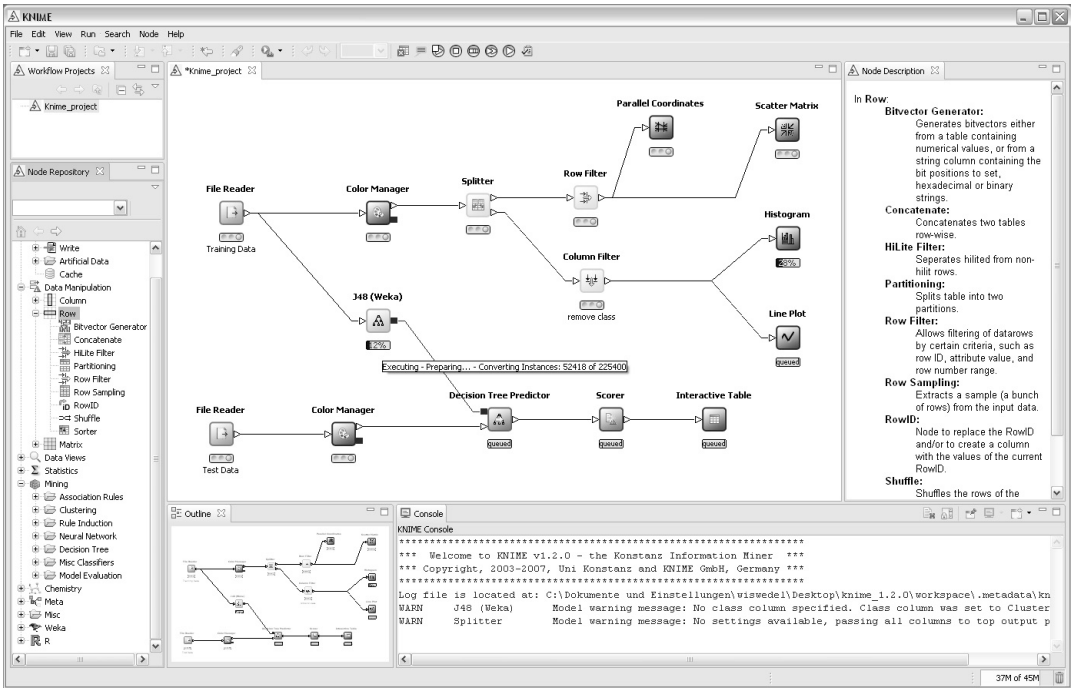
gradnji spletne aplikacije uporabiti (bolj podrobno opisano v razdelku 4.2.5).

Čeprav aplikacija omogoča grajenje delotokov hierarhično, kar pomeni, da del delotoka lahko skrijemo v novo narejeni operator, pa tega operatorja ne moremo shraniti in ga kasneje samostojno uporabljati. V naši aplikaciji smo zato načrtovali sistem, ki omogoča gradnjo podprocesov, ki jih lahko shranjujemo in nalagamo kot samostojne operatorje (bolj podrobno opisano v razdelku 4.2.6).

2.1.3 KNIME

Okolje KNIME so začeli razvijati leta 2004 na Univerzi v Konstanzi. Okolje je osnovano na platformi Eclipse (tj. razširljivo okolje za razvoj aplikacij, razvito v programskem jeziku Java).

V okolju KNIME je uporabniku omogočena gradnja delotokov (v žargonu aplikacije imenovani tudi cevovodi (*ang. pipelines*)). Osnovni gradniki delotokov so vozlišča, ki jih podobno kot pri že prej opisanih aplikacijah nanašamo na okno, ki je namenjeno gradnji delotoka.



Slika 2.6: Zaslonska maska okolja KNIME s primerom delotoka.

Grafični uporabniški vmesnik aplikacije KNIME je razdeljen na več delov

(slika 2.6). Največji in najpomembnejši med njimi je urejevalnik delotoka, kjer poteka snovanje delotoka ter premikanje in povezovanje vozlišč. Različni tipi vozlišč se nahajajo v hierarhično urejeni knjižnici na levi strani grafičnega uporabniškega vmesnika.

Slika 2.7: Primeri vozlišč v okolju KNIME.

Vozlišče je osnovna enota delotoka. Vsako vozlišče ima svoje stanje predstavljeno s semaforjem pod njegovo ikono. Semafor je lahko rdeče, rumene ali zelene barve (slika 2.7). Rdeča barva pomeni, da vozlišče še ni pravilno nastavljeno in ni pripravljeno na izvajanje. Rumena barva nam pove, da se operacija v vozlišču še ni izvedla, a je pravilno nastavljeno. Z zeleno barvo so označena tista vozlišča, ki so bila uspešno izvedena.

Novo vozlišče v delotoku ustvarimo tako, da iz repozitorija z miško povlečemo izbran tip vozlišča v delotok (*ang. drag and drop*). Skupaj z ostalimi vozlišči jih povezujemo tako da z miško narišemo črto od izhoda enega vozlišča do vhoda drugega vozlišča. Vhodi in izhodi vozlišč so kodirani barvno in s simboli (slika 2.7). Tako je že na prvi pogled jasno, kateri vhodi in izhodi so med seboj kompatibilni.

V osnovni namestitvi okolja je približno sto različnih tipov vozlišč [3]. Hierarhično so razdeljeni v tipe vozlišč, ki:

- berejo in pišejo podatke (npr. branje datotek CSV, pisanje razpredelnic Excel),
- manipulirajo podatke (npr. urejanje podatkov, filtriranje podatkov),
- transformirajo podatke (npr. dopolnitev manjkajočih vrednosti),
- vsebujejo algoritme za rudarjenje in učenje (npr. nevronske mreže, odločitvena drevesa),
- izvajajo ostala opravila (npr. izvajanje skript).

Poleg navedenih vozlišč je možno okolje KNIME razširiti z že pripravljenimi vozlišči, ki implementirajo algoritme programa WEKA in algoritme programa R-project. Programiranje lastnega vozlišča sestoji iz razširitve treh abstraktnih razredov [3], za kar obstaja tudi čarovnik v sami aplikaciji.

Dobri lastnosti programa sta jasen prikaz vhodov in izhodov (skladnost je možno ugotoviti že s pogledom na vhod/izhod) in barvni semafor pod samimi vozlišči. Za našo aplikacijo smo načrtovali podobne funkcionalnosti. Problem prikaza skladnosti vhodov in izhodov smo rešili z napisanimi kraticami podatkov na vhodih in izhodih. Stanja vozlišč pa smo ponazorili z ikonami pod vozlišči in obarvanjem vozlišč v primeru napake.

2.1.4 Orange

Aplikacijo Orange so razvili v Laboratoriju za umetno inteligenco na fakulteti za računalništvo in informatiko na Univerzi v Ljubljani.

Orange je knjižnica rutin in objektov, napisana v jeziku C++, ki implementira standardne in manj standardne algoritme za strojno učenje in rudarjenje podatkov [1]. Prav tako je Orange skriptno okolje, v katerem lahko uporabljamo in snujemo nove algoritme. Skriptno okolje uporablja interpretacijski jezik Python, ki je počasnejši, a v njem nekatere stvari naredimo lažje kot v C++. Poleg knjižnice C++ in skriptnega okolja je Orange tudi skupek grafičnih gradnikov (*ang. widgets*), ki uporabljajo rutine iz knjižnice C++ in metod Python. Grafične gradnike lahko povezujemo v okolju, ki se imenuje Orange Canvas, ki služi kot grafični uporabniški vmesnik.

Enostavnost skriptnega jezika je prikazana na sliki 2.8.

```
import orange
data = orange.ExampleTable('voting.tab')
model = orange.BayesLearner(data)
for i in range(5):
    print model(data[i]), 'originally', data[i].getclass()
```

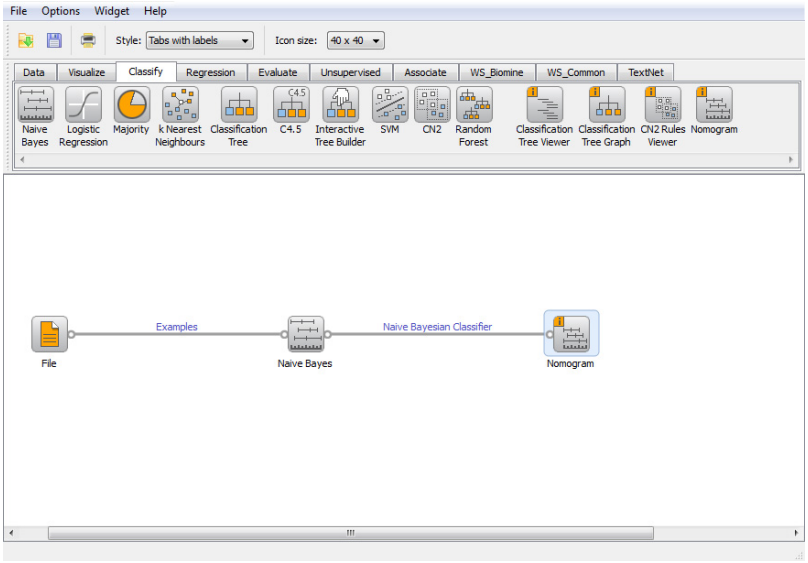
Slika 2.8: Primer strojnega učenja z uporabo knjižnice Orange.

Prvi stavek pove, da bomo uporabljali knjižnico Orange. Množica podatkov je zhranjena v datoteki `voting.tab`. Množica vsebuje podatke o oddanih glasovnicah na volitvah, razredni atribut pa je politična stranka, kateri je bil oddan glas (atribut je binaren, možni vrednosti sta `republican` in `democrat`). V spremenljivko `model` se shrani zgeneriran model Naivnega Bayesovega klasi-

fikatorja. V zanki nato izpišemo prvih pet učnih primerov, njihov napovedani razred in njihov pravi razred. Izpisan izid je prikazan na sliki 2.9.

```
republican originally republican
republican originally republican
republican originally democrat
democrat originally democrat
```

Slika 2.9: Izid klasifikacije z uporabo knjižnice Orange.

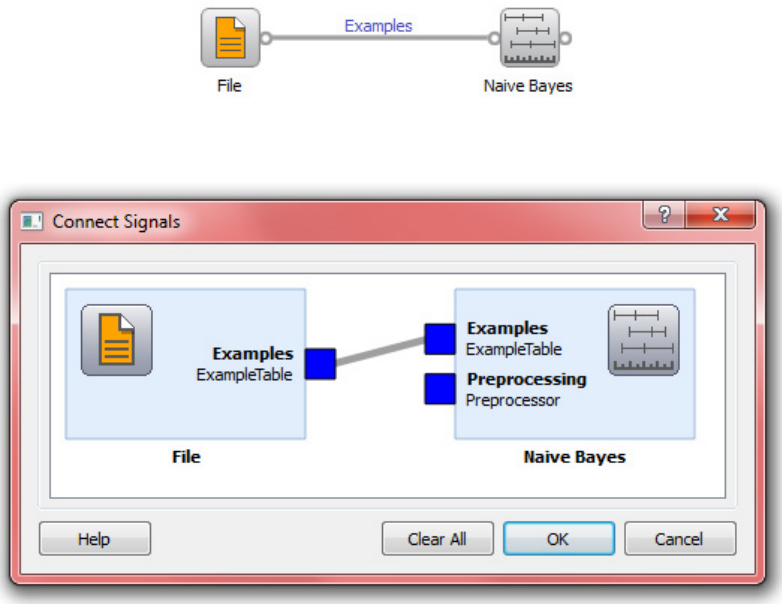


Slika 2.10: Grafični uporabniški vmesnik Orange Canvas s primerom delotoka.

V grafičnem uporabniškem vmesniku Orange Canvas (slika 2.10) s klikanjem na hierarhično urejene grafične gradnike sestavljamo delotok. Grafični gradniki so v osnovnem paketu Orange naslednjih tipov:

- podatkovni,
- vizualizacijski,
- klasifikacijski,
- regresijski,
- evaluacijski,

- povezovalni in
- gradniki za nenadzorovano učenje.



Slika 2.11: Vhodi in izhodi gradnikov delotokov se pri Orange Canvas lahko razločijo šele po dvojnem kliku na povezavo.

V grafičnem vmesniku Orange Canvas so vhodi in izhodi grafičnih gradnikov predstavljeni s krogcem na levi ali desni strani gradnika (glede na to ali gre za vhode ali izhode), ne glede na število le teh. Podrobne vhode in izhode vidimo šele ob dvojnem kliku na povezavo med gradniki (slika 2.11). V spletnem okolju, ki smo ga načrtovali in razvili, smo ta problem rešili tako, da smo omogočili pregled nad vsemi vhodi in izhodi že v osnovnem pogledu (podrobneje opisano v razdelku 4.2.4).

Orange Canvas ne ločuje strogo med obdelavo podatkov in prikazom podatkov (kot RapidMiner). Za predstavitev podatkov uporablja posebne, temu namenjene gradnike. Zaradi enostavnosti smo podoben pristop uporabili tudi pri naši aplikaciji.

2.2 Uporaba spletnih servisov v okoljih za strojno učenje

Ena od izboljšav aplikacij za strojno učenje je vpeljava možnosti porazdeljenega strojnega učenja [5]. Algoritmi se v tem primeru ne izvajajo vsi na enem računalniku, temveč na različnih, oddaljenih računalnikih.

2.2.1 Weka4WS

Okolje Weka4WS je razširitev okolja Weka [6]. Weka nudi veliko zbirko algoritmov za strojno učenje, a se vsi izvajajo lokalno, na uporabnikovem računalniku. Cilj razširitve Weka4WS je bilo razširiti funkcionalnosti okolja Weka na tak način, da omogoča oddaljeno izvajanje algoritmov in tako s pomočjo sočasnega izvajanja skrajšati celoten izvajalni čas.

Predprocesiranje podatkov in prikaz podatkov se pri Weka4WS še vedno izvajata lokalno, algoritmi za klasifikacijo, razvrščanje, in povezovalna pravila pa se lahko zaganjajo na oddaljenih virih.

Algoritmi, ki se oddaljeno izvajajo, so implementirani kot spletni servisi, ki se jih lahko prikaže kot del grafičnega uporabniškega vmesnika. Za ustvarjanje, naslavljanje in uporabo spletnih servisov Weka4WS uporablja družino specifikacij WSRF.

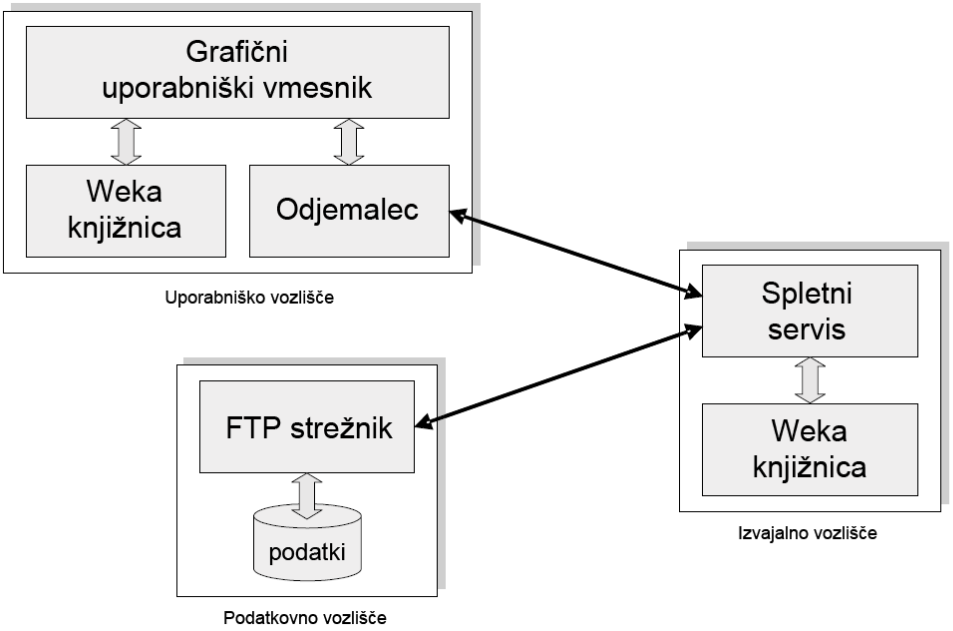
Slika 2.12 prikazuje arhitekturo okolja Weka4WS, ki vsebuje tri različne vrste vozlišč: *podatkovna vozlišča* (*ang. storage nodes*), ki hranijo podatke, ki se bodo analizirali, *izvajalna vozlišča* (*ang. computing nodes*), ki izvajajo algoritme, in *uporabniška vozlišča*, ki predstavljajo lokalne računalnike uporabnikov.

Grafični uporabniški vmesnik uporabniškega vozlišča je razširjen grafični vmesnik okolja Weka, ki podpira izvajanje lokalnih in oddaljenih nalog strojnega učenja. Na sliki 2.13 je prikazan uporabniški vmesnik z dodanim poljem *remote*, ki vsebuje seznam spletnih servisov, ki se lahko pokličejo, in dva gumba za zagon in ustavitev izvajanja operacije na izbranem spletnem servisu.

Dobra lastnost okolja Weka4WS je ta, da za vsak zagnani oddaljeni spletni servis usvari novo nit, kar okolju omogoča sočasno izvajanje večih algoritmov. Pri načrtu in razvoju našega spletnega okolja smo ohranili to funkcionalnost.

2.2.2 Orange4WS

Okolje Orange4WS je zgrajeno na okolju Orange in projektu Python for Web Services (bolj podrobno na Zolera SOAP infrastrukturi), ki nudi knjižnice za



Slika 2.12: Skica arhitekture Weka4WS.

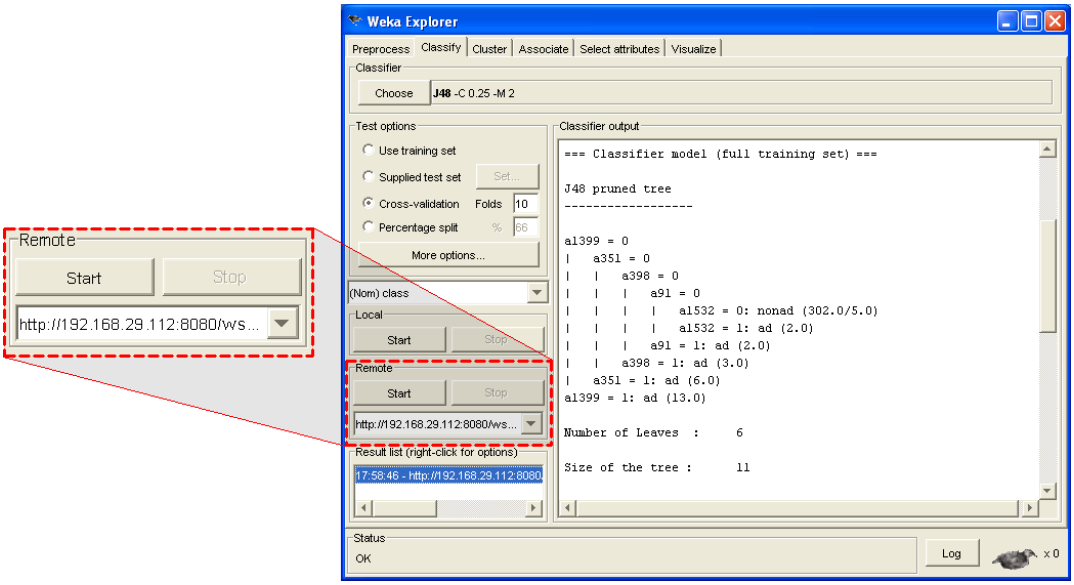
razvoj spletnih servisov z uporabo programskega jezika Python [2].

Okolje Orange nudi grafični uporabniški vmesnik Orange Canvas (opisan v razdelku 2.1.4), v katerem je omogočeno t.i. vizualno programiranje s povezovanjem večih komponent v delotoke, ki so vizalne predstavitve kompleksnih procedur. Orange4WS razširja ta model z vpeljavo spletnih servisov.

Orange4WS je sestavljen iz podpornega (spodnjega) sloja, ki skrbi za izvajanje spletnih servisov, sporočanje napak ter prenašanje in transformiranje podatkov, in zgornjega sloja, ki uporablja nizkonivojske funkcionalnosti, da omogoči spletne servise kot grafične gradnike v Orange Canvas. Vse tehnične podrobnosti, povezane z integracijo spletnega servisa, so povzete v ukazu *uvozi spletni servis*, ki združuje vse potrebne korake za gradnjo novega grafičnega gradnika iz navedenega spletnega servisa.

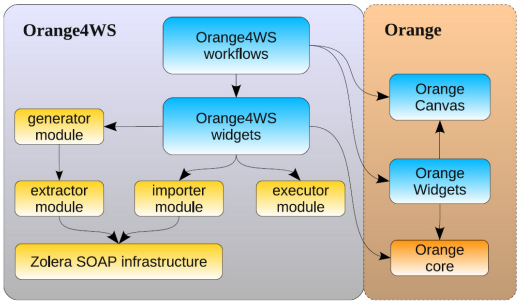
Struktura Orange4WS je skicirana na sliki 2.14. Transformacija podatkov, klicanje spletnih servisov in odzivanje na napake so uporabniku skrite in so z uporabniškega vidika enake kot običajno obnašanje grafičnih gradnikov: sprejemanje podatkov, interno procesiranje, posredovanje izidov.

Python for Web Services Project omogoča pregledovanje spletnih servisov in njihovih vhodov in izhodov. Generacijski modul (prikazan na sliki 2.14)

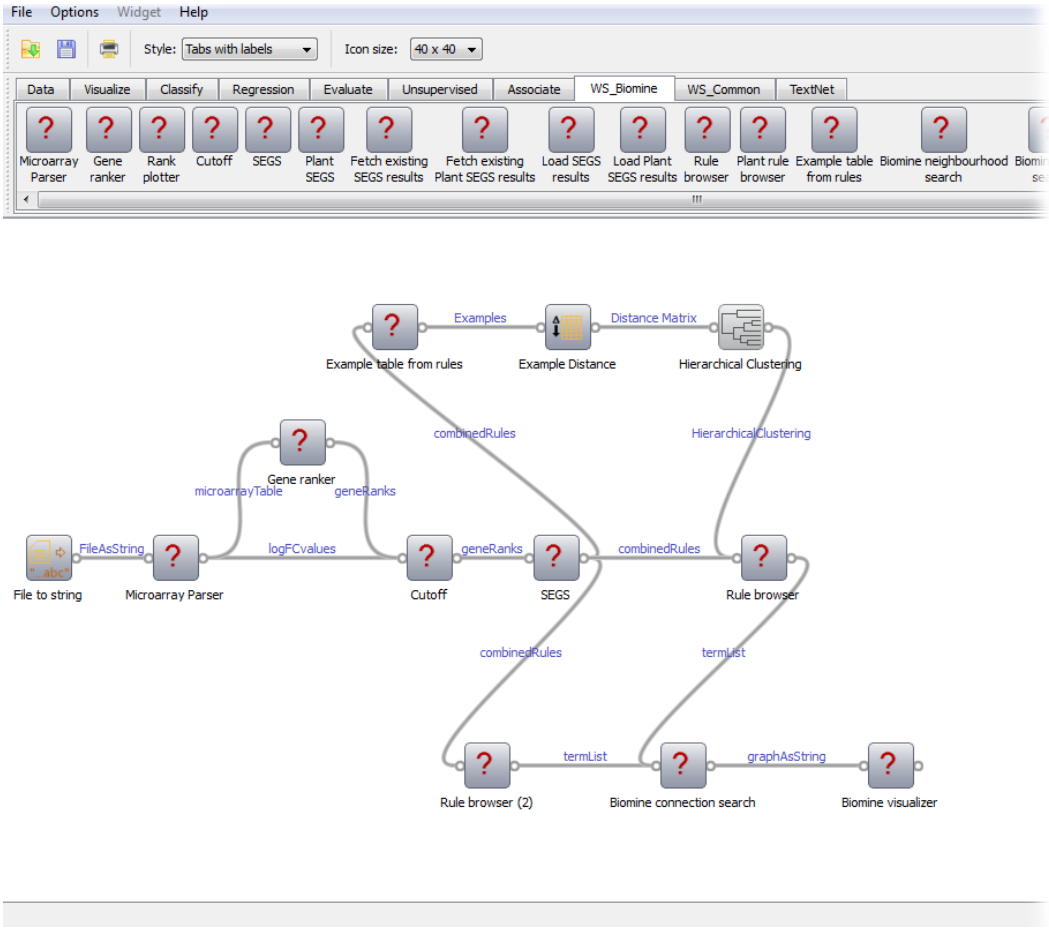


Slika 2.13: Grafični uporabniški vmesnik z dodanim poljem za zaganjanje spletnih servisov.

pretvori vhode in izhode spletnega servisa WSDL v tipizirane kanale, ki jih uporablja Orange Canvas za komunikacijo med grafičnimi gradniki. Ob generiranju gradnikov se vsaka funkcija preslika v natančno en gradnik in vsak parameter v en tipiziran kanal. Grafični gradniki, generirani z Orange4WS, se lahko uporabljajo v povezavi z gradniki, ki v Orange Canvas že obstajajo. Orange Canvas se nelokalnosti gradnikov s spletnimi strežniki ne zaveda. Na sliki 2.15 je prikazan delotok sestavljen iz gradnikov, ki so bili ustvarjeni z Orange4WS in iz običajnih Orange gradnikov.



Slika 2.14: Skica arhitekture Orange4WS.



Slika 2.15: Zaslonska maska Orange Canvas z Orange4WS.

Prav tako kot v Orange4WS smo v naši spletni aplikaciji načrtovali in realizirali generator gradnikov iz spletnih servisov WSDL. Vhodi in izhodi spletnih servisov se pretvorijo v notranje tipe, ki jih uporablja spletna aplikacija.

Poglavje 3

Uporabljena orodja in metode pri razvoju spletne aplikacije

Zaradi spletne narave okolja za strojno učenje, ki smo ga razvili, smo morali poseči po tehnologijah, ki so primerne za izdelavo spletnih strani in spletnih aplikacij. Opisane uporabljene tehnologije nam omogočajo prikaz in interakcijo z uporabniškim vmesnikom, upravljanje podatkovnega toka, izvajanje algoritmov za strojno učenje in uporabo spletnih servisov.

3.1 Uporabljene tehnologije pri razvoju grafičnega uporabniškega vmesnika

Ker želimo, da bi bila celotna aplikacija uporabniku dostopna iz spletnega brskalnika, je zelo pomembna izbira tehnologij za gradnjo uporabniškega vmesnika. Pri izbiri tehnologij nismo imeli v mislih le enostavnosti in intuitivnosti uporabniškega vmesnika samega, temveč smo težili tudi k temu, da aplikacija deluje v vseh, v času razvoja aplikacije, razpoložljivih brskalnikih. Razvoj sam in glavno testiranje je potekalo na brskalniku Mozilla Firefox 3.6, skrbeli pa smo, da aplikacija deluje tudi na drugih brskalnikih.

Seznam podprtih brskalnikov:

- Mozilla Firefox
- Internet Explorer 7 in 8
- Safari
- Opera

- Google Chrome

Postavitev elementov uporabniškega vmesnika je tako napisana v jeziku HTML, njihov izgled pa je napisan v jeziku CSS. S tem smo ločili izgled aplikacije od njegove funkcionalnosti in omogočili enostavno menjavo stilov in barvnih shem za bolj privlačen izgled aplikacije. Sama manipulacija objektov, definiranih v kodi HTML aplikacije, pa je omogočena s skriptnim jezikom JavaScript, ki se izvaja na odjemalčevem računalniku, in sicer kar v brskalniku. Tu smo zopet morali skrbeti za brezhibno delovanje v različnih brskalnikih, saj ima vsak brskalnik svoj interpreter JavaScript. Delovanje tega dela aplikacije je bilo preizkušeno v že zgoraj navedenih brskalnikih.

Ker bo navzven celotna aplikacija izgledala kot spletna stran, je zelo pomembna zahteva tudi to, da se vsa interakcija z uporabnikom dogaja, ne da bi se spletna stran osvežila. Tej zahtevi smo ustregli z uporabo knjižnic JavaScript, ki uporabljajo metodo asinhronega JavaScript in XML (*ang. Asynchronous JavaScript and XML*, s kratico Ajax).

3.1.1 HTML/CSS in JavaScript

Za opis postavitve elementov smo uporabili vrsto dokumenta XHTML™ 1.0, ki je razširitev jezika HTML4 [7]. HTML4 je standarden generaliziran označevalni jezik (*ang. Standard Generalized Markup Language*, s kratico SGML), ki se podreja mednarodnemu standardu ISO 8879, ki je bil v času pisanja najbolj upoštevan standard za objavljanje na svetovnem spletu [7].

Dokumenti XHTML se podrejajo standardu XML [7], kar pomeni da je objektni model dokumenta (*ang. Document object model*, s kratico DOM) mogoče pregledovati, urejati in preverjati na enak način kot XML dokumente. To nam močno olajša dinamično dodajanje in urejanje objektnega modela dokumenta same aplikacije med izvajanjem (problem dodajanja novih elementov v delotok se prevede na problem vstavljanja elementov v drevesno strukturo objektnega modela dokumenta).

Predloge izgleda vsakega od elementov smo opisali v dokumentih CSS 2.1 (*ang. Cascading Style Sheets*). Določanje stilov elementov poteka tako, da se elementi dokumenta XHTML sestavijo v drevo, nato pa se posamezne stilske predloge dodajajo elementom na podlagi posebnih izbirnih ukazov (*ang. selectors*), s katerimi izberemo enega ali več elementov v dokumentu [8].

Za dodajanje, spreminjanje in urejanje elementov v dokumentu XHTML smo uporabili skriptni jezik JavaScript, ki se v celoti izvaja na odjemalčevem računalniku. V naši spletni aplikaciji JavaScript opravlja dve pomembni nalogi. Na prvem mestu JavaScript skrbi za interakcijo z uporabnikom. Ta skriptni

jezik je dogodkovno usmerjen (*ang. event based*), kar pomeni da na vsakem elementu lovimo dogodke in ustrezno na njih reagiramo (npr. klikanje na elemente, premikanje miške po elementih, označevanje elementov, pritiskanje tipk na tipkovnici). Njegova druga naloga v naši aplikaciji pa je, da interaktira s strežnikom in spletnimi servisi. Vsa interakcija, ki poteka z uporabnikom in spletnimi servisi, mora potekati brez osveževanja celotne aplikacije (tj. spletne strani), kar bi pomenilo izgubo podatkov za uporabnika, saj bi se aplikacija povrnila v začetno stanje. Temu se izognemo z uporabo metod Ajax, ki nam omogočajo asinhrono komuniciranje med brskalnikom in strežnikom, ne da bi se spletna stran v celotni ponovno naložila. Rezultate komunikacije uporabniku prikazujemo z manipulacijo objektnega modela dokumenta (npr. vstavljanje besedila v element, ki prikazuje stanje aplikacije, dodajanje elementov, odstranjevanje elementov).

3.1.2 jQuery in jQuery UI

Da smo olajšali delo s pisanjem kode JavaScript, smo uporabili knjižnico jQuery. Ta knjižnica nam nudi štiri pomembne olajšave.

Iskanje elementov v drevesu DOM je zelo poenostavljeno, saj lahko uporabljamo enake ukaze kot pri pisanju CSS (*ang. selectors*), s katerimi izberemo enega ali več elementov. Preprosto lahko izberemo vse elemente, ki pripadajo nekemu delu aplikacije (npr. vse operatorje, ki so trenutno na kanvasu) in na njih postavimo poslušalce za dogodke (*ang. event listeners*).

Manipulacija elementov je enostavna, saj nam knjižnica nudi zbirko ukazov za spreminjanje atributov elementov. Tako lahko na primer nekemu elementu enostavno spremenimo barvo (npr. ob kliku na element želimo uporabniku posredovati informacijo, da je ta element sedaj "označen", zato mu spremenimo barvo).

Knjižnica jQuery nam omogoča tudi poenostavljeno uporabo asinhrone komunikacije brskalnika s strežnikom, kar nam olajša izvajanje algoritmov in prikazovanje rezultatov uporabniku. S tem je omogočeno izvajanje več algoritmov hkrati, kar je posebej koristno pri uporabi spletnih servisov, kjer lahko prihranimo veliko časa.

Kljub temu, da imajo vsi brskalniki svoj lasten interpreter JavaScript, vsi ukazi, ki jih nudi knjižnica jQuery, delujejo na vseh že opisanih brskalnikih (poglavje 3.1), za kar skrbi knjižnica sama. Tako smo pri prilagajanju aplikacije na različne brskalnike lahko odmislili celotno kodo JavaScript in se osredotočili le na CSS in HTML, ki sta za prilagajanje bolj težavna.

Dodatek h knjižnici jQuery pa je knjižnica jQuery UI, ki nudi vtičnike (*ang.*

plugins) za gradnjo uporabniškega vmesnika. Ta knjižnica nam omogoča enostavno odpiranje modalnih oken za interakcijo z uporabnikom (npr. nastavljanje parametrov določenega operatorja), premikanje operatorjev na kanvasu z miško (*ang. drag and drop*). Tako kot jQuery, jQuery UI skrbi, da se uporabniški vmesnik obnaša enako v različnih brskalnikih.

3.2 Uporabljene tehnologije za izvajanje algoritmov

Izvajanje algoritmov v spletni aplikaciji poteka tako, da brskalnik asinhrono kliče funkcije, ki so shranjene na strežniku kot skripte PHP (*ang. PHP: Hypertext preprocessor*), ki bodisi posredujejo zahtevo naprej spletnemu servisu, bodisi same opravijo določeno opravilo. V vsakem primeru pa vrnejo poročilo o uspehu ali neuspehu, kar uporabniški vmesnik posreduje uporabniku.

3.2.1 PHP

PHP je večnamenski odprtokodni skriptni programski jezik, ki večinoma teče na spletnih strežnikih. Sprva je bil namenjen za dinamično generiranje kode HTML (od tukaj pomen kratice PHP: Hypertext Preprocessor).

Interpreter PHP lahko deluje kot program v ukazni vrstici ali kot dodatek k spletnemu strežniku, ki za vsak zahtevek neke dinamične spletne strani zažene novo instanco procesa PHP, ki pripravi primerno kodo HTML, ki se pošlje odjemalcu. Celotno izvajanje skript PHP se dogaja na spletnem strežniku. Odjemalec nikoli ne vidi kode PHP, vidi le rezultat, ki ga skripta vrne.

Posebnost interpreterja PHP je, da izvaja le kodo, ki je znotraj posebnih mej (*ang. delimiter*), kar pomeni da lahko kodo PHP pišemo znotraj datoteke HTML.

V primeru (slika 3.1) je prikazana koda HTML z vstavljeno kodo PHP, kot je shranjena na strežniku. Interpreter PHP bi vrnil nedotaknjeno kodo HTML, razen kode, ki je napisana med mejami `<?php` in `?>`. Takšno kodo bi interpreter izvedel in vrnil samo rezultate, ki bi jih vrnil stavek `echo`.

Za potrebe spletne aplikacije bi lahko uporabili tudi katerega od drugih skriptnih jezikov, ki se lahko izvajajo kot dodatki k spletnemu strežniku (npr. Python, ASP.net). Jezik PHP je bil izbran zaradi njegove enostavnosti in razširjenosti.

V razviti spletni aplikaciji so notranja delovanja operatorjev realizirana kot skripte PHP, ki bodisi izvedejo določeno nalogo, bodisi uporabijo posebne

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
  <?php
    echo "Hello World";
  ?>
  </body>
</html>
```

Slika 3.1: Primer kode v skriptnem programskem jeziku PHP.

protokole za uporabo oddaljenih spletnih servisov. V vsakem primeru pa vrnejo sporočilo, ki ga obdela del aplikacije JavaScript (več o arhitekturi aplikacije v razdelku 4.1).

3.2.2 WSDL in SOAP

Opisovalni jezik za spletne servise (*ang. Web Services Description Language*, s kratico WSDL) je jezik, ki temelji na jeziku XML in nudi model opisovanja spletnih servisov.

WSDL opiše spletni servis tako, da specificira njegovo lokacijo in operacije (metode), ki jih servis nudi. Dokument WSDL je v principu dokument XML, ki vsebuje množico definicij, s katerimi opiše spletni servis. Te definicije vključujejo elemente, kot so tipi podatkov, tipi sporočil, operacije in komunikacijski protokoli, ki jih uporablja opisovani spletni servis.

Različne vrste operacij, ki jih opišemo, se delijo glede na čas in vrstni red zahtevka in odgovora. Poznamo operacije, ki so enosmerne (operacija dobi zahtevek, a ne vrne odgovora), tipa zahtevek-odgovor (operacija dobi zahtevek in vrne odgovor), tipa zahtevaj odgovor (operacija pošlje zahtevek in čaka na odgovor), tipa notifikacija (operacija le pošlje sporočilo in ne čaka na odgovor).

S pomočjo t.i. vezav (*ang. bindings*) povežemo spletni servis z določenem protokolom (v našem primeru protokol SOAP), ki ga uporabljamo za uporabo spletnih servisov.

SOAP je protokol, ki temelji na jeziku XML in omogoča aplikacijam izmenjavo informacij preko protokola HTTP (bolj enostavno: SOAP je protokol za uporabo operacij spletnih servisov).

Za razvoj aplikacij je pomembna internetna komunikacija med programi.

Današnje aplikacije uporabljajo klic oddaljenih procedur (*ang. Remote Procedure Call*, s kratico RPC) med objekti, za kar protokol HTTP ni bil načrtovan. SOAP nudi komunikacijo med aplikacijami, ki tečejo na različnih operacijskih sistemih, različnih tehnologijah in so napisane v različnih programskih jezikih.

Sporočilo SOAP je običajen dokument XML, ki vsebuje naslednje elemente:

- ovojnico, ki identificira dokument XML kot sporočilo SOAP,
- glavo, ki vsebuje informacije, ki so specifične za aplikacijo (npr. avtentikacija, plačilo),
- telo, ki vsebuje klic procedure in podatke o odgovoru,
- element, ki vsebuje odgovore na napake in podatke o stanju.

Na sliki 3.2 je prikazan primer skeleta sporočila SOAP.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>mmmm
```

Slika 3.2: Primer skeleta sporočila SOAP.

Metoda SOAP je zahtevek/odgovor HTTP, ki se podreja pravilom kodiranja SOAP. Zahtevek SOAP je lahko HTTP POST ali HTTP GET. Delovanje protokola SOAP si najlažje predstavljamo s pomočjo "formule":

$$HTTP + XML = SOAP$$

V razviti spletni aplikaciji smo uporabili knjižnico `NuSoap`, ki skrbi za pravilno nastavitev glave zahtevka HTTP, pošiljanje zahtevkov preko zahtevkov HTTP POST in GET in pravilno prebiranje opisov spletnih servisov WSDL.

3.3 Povezava med uporabniškim vmesnikom in strežniškim delom spletne aplikacije

Za povezavo med uporabniškim vmesnikom in strežniškim delom aplikacije smo uporabljali protokol HTTP. Vse vhode, ki jih določen operator potrebuje, smo zapakirali v zahtevek HTTP POST, ki se s pomočjo knjižnice jQuery pošlje vnaprej določeni skripti PHP.

Pri pošiljanju informacij je najbolj pomembno, da se pri pošiljanju zahtevka in prejemanju odgovora aplikacija ne osveži, kar bi pomenilo vrnitev v prvotno stanje. Stanje bi torej morali pred vsakim klicem shraniti, ga poslati skupaj z zahtevkom in ob vrnitvi rezultata ponovno skonstruirati celoten delotok in prejšnje stanje aplikacije, kot ga je uporabnik imel pred zagonom nekega operatorja. Da smo se temu izognili, smo uporabili skupek tehnologij Ajax (Asynchronous JavaScript And XML). Kljub imenu pa uporaba jezika XML pri pošiljanju ni obvezna, prav tako ni obvezna asinhronost. V našem primeru smo uporabljali asinhrone klice, kar nam omogoči sočasno izvajanje večih skript (vsaka pa se na strežniku izvede kot nova instanca interpreterja PHP, od katerih nekatere asinhrono kličejo spletne servise s pomočjo protokola SOAP).

Kot omenjeno, je Ajax skupek tehnologij, ki uporablja kombinacijo HTML in CSS za prikaz informacij, JavaScript za manipulacijo objektnega modela dokumenta za dinamični prikaz informacij in omogočanje interakcije s prikazanimi informacijami uporabniku. JavaScript in objekt XMLHttpRequest nudita možnost za izmenjavo podatkov asinhrono med spletnim brskalnikom in strežnikom, s čimer se izognemo osvežitvi celotnih strani.

Asinhrono nalaganje spletnih vsebin je prvič postalo popularno, ko so se začele uporabljati javanske aplikacije (*ang. Java applet*) leta 1995. To je omogočalo prevedeni kodi na odjemalčevi strani nalaganje podatkov asinhrono s spletnega strežnika po tem, ko je bila spletna stran naložena. Leta 1996 so ustvarjalci brskalnika Internet Explorer uvedli element `<iframe>`, ki je omogočal asinhrono nalaganje spletnih vsebin (del elementa HTML je bil okvir, v katerega se je naložil drug dokument HTML). Leta 1999 so razvijalci brskalnika Internet Explorer razvili objekt XMLHTTP ActiveX, katerega svojo

različičo pa so razvijalci ostalih brskalnikov (Mozilla, Safari in Opera) razvili pod imenom objekt XMLHttpRequest, ki je omogočal asinhrono pošiljanje zahtevkov HTTP v ozadju. Možnost pošiljanja zahtevkov HTTP v ozadju je postala najbolj popularna leta 2004, ko jo je Google uporabil v svojih spletnih aplikacijah Gmail in Google Maps.

Brez tehnologij Ajax bi bila razvoj in uporaba spletne aplikacije, razvite v tem diplomskem delu, zelo otežena, tako za razvijalca (več programiranja funkcionalnosti za ohranjanje trenutnega stanja), kot tudi za uporabnike (daljši nalagalni časi, nezmožnost sočasnega izvajanja).

Poglavje 4

Razvoj spletnega okolja za strojno učenje

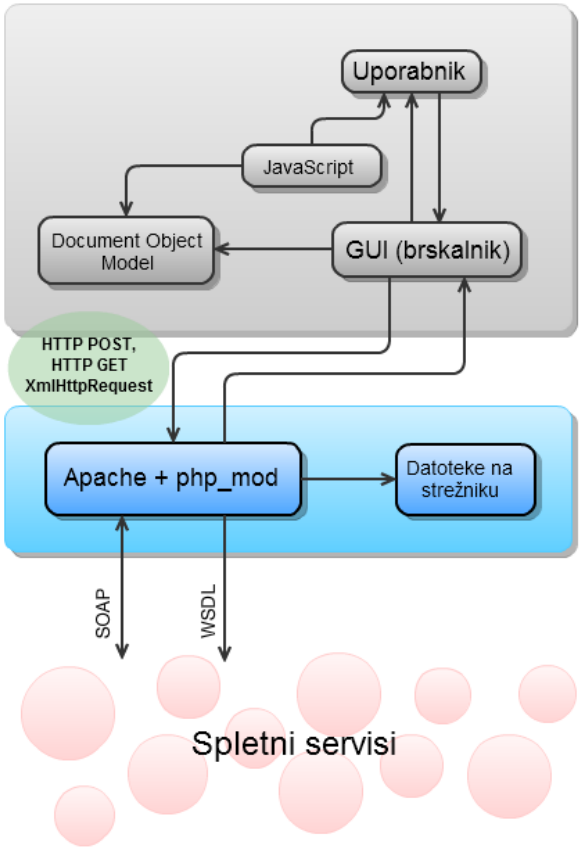
Razvoj spletnega okolja za strojno učenje je bil razdeljen v več faz. Vsaka faza je ustrezala eni od plasti spletne aplikacije, ki so bolj podrobno opisane v tem poglavju. Razvoja smo se lotili od zgoraj navzdol, kar pomeni, da smo najprej izdelali uporabniški grafični vmesnik, kasneje pa mu dodali vse potrebne funkcije. Faze so potekale v enakem vrstnem redu, kot so opisane plasti aplikacije v tem delu in skicirane na sliki 4.1.

4.1 Shema arhitekture spletnega okolja

Zgradbo aplikacije najlažje predstavimo v plasteh (slika 4.1). Najvišja plast je grafični uporabniški vmesnik, ki je opisan v jeziku HTML. Njegova funkcionalnost je zagotovljena s pomočjo skriptnega jezika JavaScript, ki deluje na uporabnikovem računalniku (v internetnem brskalniku). Funkcije JavaScript tako manipulirajo objektni model dokumenta, ki je definiran v datoteki HTML. Na različne elemente HTML dodajajo poslušalce dogodkov, preko katerih se izvajajo vnaprej definirane funkcije.

S pomočjo funkcij JavaScript se asinhrono kličejo skripte PHP, ki se v celoti izvajajo na strežniku. S pomočjo tehnik Ajax lahko funkcije JavaScript pošiljajo zahtevek skriptam PHP na strežniku, ne da bi se aplikacija osvežila (osvežitev aplikacije bi pomenila povrnitev na začetno stanje, čemur se želimo izogniti).

Omenjene skripte PHP prejmejo vhod preko zahtevka POST (definiranega v protokolu HTTP), ki nam omogoča pošiljanje velike količine podatkov (omejene z nastavitvami strežnika). Te skripte PHP lahko same opravijo določeno



Slika 4.1: Prikaz zgradbe spletnega okolja v večih plasteh. Elementi na ozadjih različnih barv se lahko nahajajo na različnih računalnikih, ni pa to potrebno. Na sliki so poleg ločenih delov aplikacije prikazane tudi tehnologije, ki so uporabljene pri teh delih.

funkcijo in vrnejo rezultate, ki jih obdela JavaScript in pravilno prikaže uporabniku (z dodajanjem, odstranjevanjem in spreminjanjem stila elementom v objektnem modelu dokumenta), ali pa služijo samo kot ovojnica (*ang. wrapper*) za uporabo spletnih servisov. Poleg podatkov, ki jih skripta dobi preko zahtevka POST, lahko dostopa do določenih datotek na strežniku in do relacijske podatkovne baze.

Začasne datoteke, ki obstajajo zato, da je pretok podatkov hitrejši (problem je bolj podrobno opisan v razdelku 4.2.9) se nahajajo na istem srednjem nivoju. Te datoteke se nahajajo na strežniku in uporabniku niso dostopne. Njihovo vsebino lahko vidi s pomočjo posebnih operatorjev, ki so temu namenjeni.

Na najnižjem nivoju se nahajajo spletni servisi, ki pa se lahko nahajajo kjerkoli na spletu. Klic spletnih servisov se izvede s skript na strežniku in ne z uporabnikovega računalnika. Prednost porazdeljenih sistemov je med drugim tudi sočasno izvajanje [5], ki nam pri velikih delotokih skrajša čas izvajanja.

4.2 Uporabniški del aplikacije

Uporabniški del aplikacije je tisti del, ki ga uporabniki vidijo in uporabljajo. Zgrajen je s tehnologijami HTML, CSS in JavaScript (podroben opis je v razdelku 3.1.1).

4.2.1 Grafični uporabniški vmesnik

Grafični uporabniški vmesnik smo načrtovali tako, da smo upoštevali dva pogoja. Prvi pogoj je bila enostavnost in intuitivnost uporabe, drugi pa upoštevanje omejitev, ki nam jih vsiljujejo brskalniki. Uporabniški vmesnik je zelo preprost, sestavljajo pa ga trije glavni deli.

- V orodni vrstici imamo v oblikah gumbov možnosti, s katerimi nadzorujemo delotok (shranjujemo, nalagamo in brišemo). Delotok lahko tudi zaženemo (izvedemo vse algoritme).
- Na levi strani pod orodno vrstico se nahaja seznam operatorjev in algoritmov, katere prenašamo na kanvas s klikom na njih. Ta seznam se dinamično širi, ko dodajamo nove operatorje in algoritme v program.
- Največji del uporabniškega vmesnika zavzema kanvas, na katerega nanašamo algoritme in jih povezujemo v delotoke. Delotok je torej usmerjen graf brez ciklov, katerega vozlišča so algoritmi, povezave pa povezani vhodi in izhodi različnih instanc na kanvas nanešenih algoritmov.

4.2.2 Orodna vrstica

Gumbi v orodni vrstici se skozi celotno izvajanje aplikacije ne spreminjajo, zato smo jih predhodno definirali kar v dokumentu HTML in vsakemu določili posebno identifikacijsko ime (ID), s pomočjo katerega smo na te gumbe nastavili poslušalce dogodkov (klikov). Vnaprej smo v kodi JavaScript definirali, kako naj se poslušalci na dogodke odzovejo. Vsak poslušalec čaka klik na svoj gumb in nato izvede svojo nalogo.

Gumbi in njihove naloge v orodni vrstici so sledeči:

- nov delotok - pobriše vse operatorje in povezave na kanvasu,
- odpri delotok - naloži že prej shranjen delotok, da ga lahko uporabnik nadgradi ali zažene
- shrani delotok - shrani delotok na strežnik, da lahko uporabnik delo na njem nadaljuje ob kasnejšem času,
- zaženi delotok - začne se izvajanje vseh algoritmov v primernem vrstnem redu,
- nastavitve programa - odpre modalno okno, v katerem lahko uporabnik spreminja posamezne nastavitve programa (npr. način risanja povezav),
- izbriši označeni element - iz kanvasa se odstrani tisti element, ki je trenutno označen.

Poleg gumbov je v orodni vrstici tudi prostor, kamor se napiše besedilo s trenutnim oz. zadnjim stanjem. Po vsakem končanem dejanju se opis dejanja zapiše v vrstico s stanjem, ki pa se glede na vrsto stanja (uspešno, neuspešno) ustrezno obarva (rumeno, rdeče, v tem vrstnem redu).

4.2.3 Kanvas

Kanvas je prostor, na katerem gradimo delotok. Ob zagonu aplikacije je kanvas prazen. Na levi strani kanvasa se nahaja seznam operatorjev, ki jih lahko na kanvas nanesemo s klikom na izbran operator.

Operatorje lahko med seboj povezujemo s klikom na njihove vhode in izhode. Na povezavah so med izvajanjem podatki, zato smo si zadali zahtevo, da lahko povežemo samo tiste vhode in izhode, ki so enakega tipa (ali tipa, ki se lahko prilagodi določenemu tipu). Druga zahteva pri povezovanju pa je ta, da se operatorji na kanvasu ne smejo povezovati na tak način, da bi tvorili cikle.

V objektnem modelu dokumenta je kanvas predstavljen kot element `<div>`, ki je definiran že v delu aplikacije HTML. Ob začetku izvajanja aplikacije je element popolnoma prazen. S pomočjo JavaScript se dinamično ob izvajanju na kanvas dodajajo operatorji in povezave. Prav tako se ob brisanju odstranjujejo.

Tekom izvajanja aplikacije se lahko ustvari nov primer kanvasa, ki predstavlja podproces. Ogled različnih kanvasov je omogočen s klikanjem po zavihkih, ki se nahajajo nad kanvasom (podrobneje opisano v razdelku 4.2.6).

4.2.4 Operatorji

Operator je element delotoka, ki ima naslednje atribute:

- ime,
- ikona,
- seznam vhodov,
- seznam izhodov,
- trenutni podatki na izhodu,
- seznam parametrov in nastavitev,
- ciljno funkcijo, ki naj se izvede,
- stanje izvajanja,
- tip operatorja,
- koordinate vizualne predstavitve operatorja.

Aplikacija ima ob zagonu prazno tabelo operatorjev, v katero se vnašajo primeri na kanvas nanešenih operatorjev. Ta tabela skupaj s tabelo povezav tvori celoten opis delotoka.

V aplikaciji poznamo tri različne skupine (tipe) operatorjev. Prva skupina so operatorji, ki so vnaprej definirani in realizirani s spletno aplikacijo. V to skupino spadajo osnovni algoritmi za strojno učenje in operatorji za uvoz in izvoz podatkov. V drugi skupini so operatorji, ki služijo za gradnjo hierarhije delotokov, ki so podrobneje opisani v razdelku 4.2.6. Tretja skupina so operatorji, ki se tvorijo iz spletnih servisov (ti so opisani v razdelku 4.2.7).

Prva skupina operatorjev je vnaprej definirana v datoteki XML, ki se prebere ob zagonu aplikacije.

Na sliki 4.2 je prikazan primer opisa operatorja za izvajanje modela strojnega učenja nad testnimi primeri v jeziku XML.

Operatorjeva funkcija je klasifikacija ali regresijska napoved testnih primerov z nekim modelom strojnega učenja. Oba vhoda (primeri in model) sta obvezna. Operator na izhod vrne množico že napovedanih primerov. Poleg vhodov, izhodov in parametrov je podana tudi ciljna datoteka PHP, ki naj se izvede ob zagonu operatorja.

```
<widget>
    <name>Model application</name>
    <action>applymodel.php</action>
    <inputs>
        <input type='model' required='true'>mod</input>
        <input type='dataset' required='true'>ds</input>
    </inputs>
    <outputs>
        <output type='dataset'>ds</output>
    </outputs>
    <preferences>
        <pref type='checkbox'
              name='ignore'
              defaultvalue='checked'>Ignore ex. w/ missing data
        </pref>
    </preferences>
    <icon>model-application.gif</icon>
</widget>
```

Slika 4.2: Primer opisa operatorja v obliki XML.

V opisu XML operatorja manjkajo njegove koordinate in stanje izvajanja, saj imajo ta podatek le primeri operatorjev. V datoteki XML hranimo le splošne opise operatorjev in jih uporabljamo izključno za tiste operatorje, ki imajo vnaprej znano število in vrsto vhodov ter izhodov. To so tisti operatorji, ki so definirani kot funkcije PHP in ne kličejo spletnih servisov. Operatorji, ki kličejo spletne servise, se definirajo dinamično med izvajanjem aplikacije.

Seznam operatorjev se naloži kot seznam `<ul>`, čigar elementom dodamo poslušalce na dogodke, ki ob kliku na njih sprožijo funkcijo, ki na kanvas doda primer operatorja.

Ob kliku na operator iz seznama se v tabelo operatorjev doda operator. V tabelo se vpišejo vsi podatki, saj primer operatorja nima reference na splošen opis operatorja. Vsak gumb v seznamu operatorjev nosi informacijo o identifikacijski številki tipa operatorja, od katerega dobi vso potrebno informacijo za zagon algoritma. Splošni tipi operatorjev so shranjeni samo v tabelah v kodi aplikacije JavaScript (za razliko od primerov operatorjev, ki so predstavljeni v tabeli in v objektnem modelu dokumenta).

V objektni model dokumenta se doda nov element, ki opisuje zgradbo vizualne predstavitve operatorja. Dinamično se zgradi koda HTML, ki je

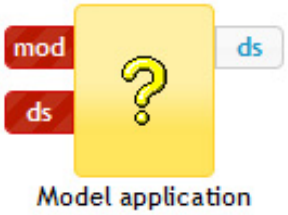
odvisna od števila vhodov in izhodov, imena in ikone. Primer vstavljenega primera prej opisanega operatorja v jeziku HTML je prikazan na sliki 4.3.

```
<div class="widget" rel="1" id="widget1">
    <div class="inputs">
        <div id="input-1-0" rel="model">mod</div>
        <div id="input-1-1" rel="dataset">ds</div>
    </div>
    <div class="widgetcenter">
        <img src="widget-icons/question-mark.gif">
    </div>
    <div class="outputs">
        <div id="output-1-0" rel="dataset">ds</div>
    </div>
    <div>Model application
        <br />
        <img class="loadingimage" src="statusimages/blank.gif">
    </div>
</div>
```

Slika 4.3: Primer opisa operatorja v označevalnem jeiku HTML.

Za vsak dodani operator se ustvarita nova elementa v objektnem modelu dokumenta, ki predstavljata dve okni. Eno okno potrebujemo za nastavljanje parametrov operatorja (preprosti vhodi, ki niso izhod drugih operatorjev) in drugo za pregled rezultatov operatorja.

Na operator se nastavijo poslušalci dogodkov. Pri operatorjih smo definirali štiri dogodke. To so premikanje operatorja po kanvasu, označevanje operatorja, označevanje vhoda ali izhoda operatorja in odpiranje kontekstnega menuja. Premikanje po kanvasu želimo izvesti tako, da lahko operator z miško s pridržanim klikom vlečemo in spustimo (*ang. drag and drop*). To nam omogoča knjižnica jQuery UI (opisana v razdelku 3.1.2), ki izbranim elementom (v našem primeru operatorjem) doda funkcionalnost premikanja znotraj elementa, v katerem se nahaja (tj. kanvas). Označevanje operatorja je realizirano tako, da ob kliku na operator (brez premikanja miške) dodamo novo stilsko predlogo elementu (obarva se z rumeno barvo, slika 4.4). Pri označitvi se poskrbi tudi za morebitno odstranitev označbe na ostalih operatorjih (označen je lahko samo en operator). Na enak način smo realizirali označevanje vhodov in izhodov. Označen je lahko samo en vhod in samo en izhod. Hkratna označitev vhoda in izhoda (v primeru da sta podatkovna tipa prilagodljiva



Slika 4.4: Slika označenega operatorja, kot ga prikažejo brskalniki.

in da povezava med njima ne bi povzročila cikla v delotoku) pomeni vstavljanje povezave med dvema operatorjema.

Za odpiranje kontekstnega menuja smo morali tako kot za operator sam najprej definirati njegov izgled v jeziku HTML in ga vstaviti v objektni model dokumenta (kar v datoteko HTML in ne dinamično med nalaganjem). Ta element je ob zagonu aplikacije skrit, ob desnem kliku na operator se njegovi stilski predlogi dinamično zamenjajo koordinate, s čimer povzročimo premik kontekstnega menuja na položaj miškinega kazalca. Takrat se spremenijo tudi poslušalci na klik elementov kontekstnega menuja. S tem smo dosegli, da se isti element prilagaja različnim operatorjem, odvisno od tega, na katerem se je odprtje kontekstnega menuja sprožilo. Znotraj kontekstnega menuja je uporabniku na voljo zagon operatorja (podrobneje opisan v razdelku 4.2.9), nastavitev preferenc, pregled rezultatov operatorja in izbris operatorja.

Ob brisanju operatorja se iz objektnega modela dokumenta odstrani vse elemente, ki predstavljajo operator, in vse elemente, ki predstavljajo tiste povezave, ki so povezane bodisi na vhod bodisi na izhod izbrisanega operatorja. Ob izbrisu se ponovno preverijo vsi pogoji (za možen zagon delotoka morajo imeti vsi operatorji podatke na vhodih, ki to zahtevajo) in ustrezno se obarvajo vhodi operatorjev (zahtevani vhodi, katerih pogoji niso izpolnjeni se obarvajo z rdečo barvo). Iz tabele operatorjev se izbranemu operatorju nastavi zastavica, da je operator izbrisan. Enako se zgodi tudi s povezavami.

Za prikaz parametrov in rezultatov določenega operatorja smo pripravili posebna okna, ki se odpirajo ob izbiri želene možnosti iz kontekstnega menuja ali pa ob dvojnem kliku na operator (velja za nastavitev parametrov). Okna so, za razliko od kontekstnega menuja, v objektnem modelu dokumenta predstavljena večkrat. Za vsak operator se v posebnem vsebovalniku v objektnem

modelu ustvarita dve okni, ki sta ob nastanku operatorja skriti. Knjižnica jQuery UI (podrobneje opisana v razdelku 3.1.2) nam omogoča enostavno dodajanje funkcionalnosti okna tem elementom. Ta okna se lahko odpirajo, zapirajo in premikajo znotraj vidnega polja brskalnika. Okna lahko vsebujejo poljubne elemente HTML, zato smo pri oknu s parametri dodali element `<form>`, ki nam omogoča vpis podatkov s pomočjo vnosnih polj, kljukic in gumbov. Vsak primer operatorja torej hrani podatke o parametrih, ki se ob zagonu v zahtevku POST pošljejo konkretni skripti PHP, ki je določena s splošnim opisom operatorja. V splošnem opisu operatorja so določeni tudi vsi različni parametri, ki jih za določen tip operatorja lahko nastavljamo.

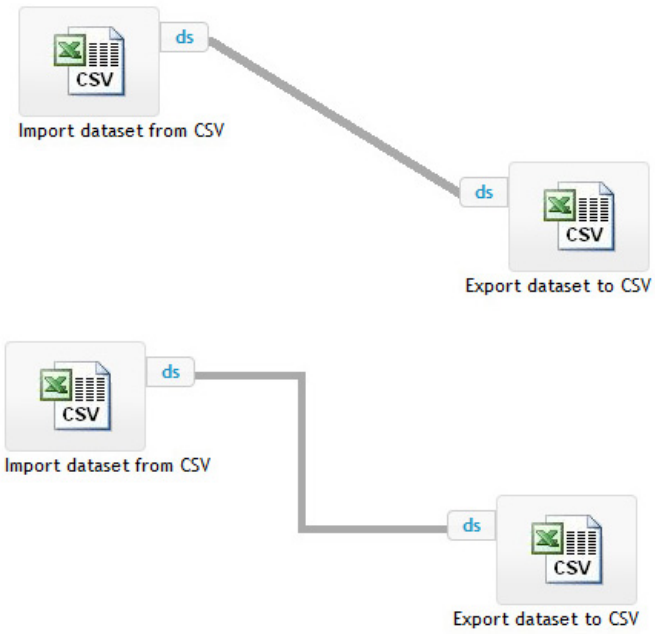
V oknu, kjer se hranijo rezultati operatorja, lahko prikazujemo poljubno vrsto podatkov (slike, grafi, besedila). Ob nastanku operatorja je v oknu za prikaz rezultatov le besedilo, ki nas obvešča, da operator še ni vrnil rezultatov.

4.2.5 Povezave

Povezava je prvina delotoka z dvema atributoma. To sta vhod in izhod (različnih) operatorjev. Na drugem nivoju aplikacije (podrobneje opisano v razdelku 4.1) je povezava predstavljena kot objekt tipa `connection`, njegovi primeri pa se hranijo v tabeli povezav.

Povezava se na kanvas doda takrat, ko sta hkrati označena vhod in izhod operatorja. Še preden se povezava ustvari, se preveri, ali se tip podatka na izhodu lahko prilagodi tipu podatka na vhodu. V kolikor je pogoju zadoščeno, se najprej v tabelo povezav doda povezava, potem pa se z iskanjem v globino preišče del delotoka, ki vsebuje operator na vhodnem delu povezave (operator, v katerega tečejo podatki) in pregledujemo vse povezave, ki vodijo izven operatorja, dokler se iskanje ne zaključi ali dokler ne najdemo operatorja na izhodnem delu povezave (operator, iz katerega tečejo podatki). Če je v delotoku zaznan cikel, se povezava odstrani in uporabniku se prikaže sporočilo o napaki. V primeru, da cikla ni, pa se povezava nariše.

Povezava je na prvem nivoju aplikacije predstavljena z več elementi HTML. Pri risanju črt smo imeli več možnosti. Ker je element HTML `<div>` pravokoten, bi lahko temu elemenu nastavili ozadje, kjer črta poteka iz levega zgornjega kota v desni spodnji kot in to ozadje raztegnili tako, da bi bila zgornja leva točka na položaju izhoda operatorja in spodnja desna na položajo vhoda operatorja. Tako bi lahko realizirali enostavno in hitro risanje črt. Problem bi nastal pri označevanju črt, saj bi se pravokotni elementi med seboj prekrivali in klikanje na njih ne bi bilo več enolično. Druga možna rešitev je deljenje črte v več elementov, tako da vsak element prikaže del črte (tako bi lahko za



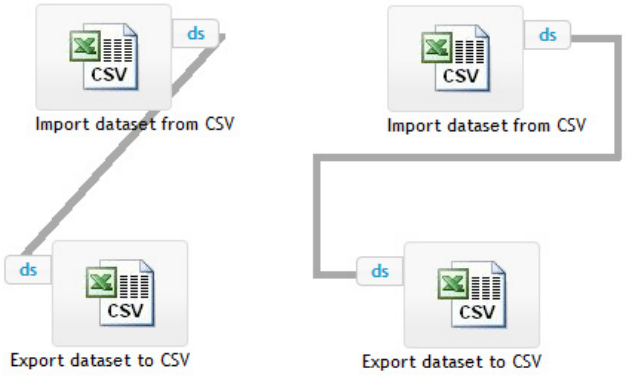
Slika 4.5: Povezave, kot jih prikažejo internetni brskalniki.

vsako točko na črti uporabili svoj element `<div>` in mu nastavili primerne koordinate). Ob velikem številu povezav bi tako dobili veliko elementov, kar bi upočasnilo delovanje internetnega brskalnika. Za risanje ravnih črt smo uporabili elemente, ki so bili večji od ene točke, tako da je aplikacija za povprečno črto potrebovala približno sto elementov. Veliko manjšo porabo elementov in hitrejše risanje smo dosegli z risanjem lomljenih ravnih črt (slika 4.5). Za en odsek smo uporabili en element. Za povezana operatorja, kjer je izhodni operator pred vhodnim, aplikacija izriše tri elemente in za povezana operatorja, kjer je vhodni operator za vhodnim, izriše pet elementov (prikaz na sliki 4.6).

4.2.6 Hierarhija delotokov

Za boljšo preglednost delotokov in ustvarjanje manjših večkrat uporabljenih delotokov za večkratno uporabo smo načrtovali in razvili tri posebne operatorje, ki uporabniku omogočajo gradnjo podprocesov, ki navzen izgledajo kot en sam operator.

Prvi izmed posebnih operatorjev je podproces. Ta vrsta operatorja se ob zagonu že nahaja v seznamu operatorjev (definirana je že statično v datoteki HTML). Ob kliku nanj se na trenutnem kanvasu ustvari primer operatorja,



Slika 4.6: Primerjava lomljenih in ravnih črt pri neobičajni postavitvi operatorjev.

hkrati pa se ustvari tudi novi kanvas, ki predstavlja notranje delovanje tega operatorja. Nad prostorom za kanvas se ustvari tudi novi zavihek, ki nam omogoča lažjo navigacijo med glavnim delotokom in podprocesom. Aplikacija v spremenljivki JavaScript hrani podatek o trenutno aktivnem kanvasu, na katerega uporabnik nanaša operatorje. V primeru, da je aktiven kanvas, ki predstavlja podproces, se vsi ustvarjeni operatorji nanesejo na ta kanvas, tudi če gre za kanvas, ki predstavlja podproces. Nivo, do katerega lahko nizamo podprocese, je neomejen.

Operator, ki predstavlja podproces, je ob nanosu na kanvas brez vhodov in izhodov. Ti se dodajo dinamično ob dodajanju drugega in tretjega posebnega operatorja znotraj podprocesa. To sta vhod (*ang. input*) in izhod (*ang. output*). Operator vhod ima en sam izhod in služi kot povezava med podprocesom in procesom, v katerem je vsebovan. Ob nanosu tega operatorja se operatorju, ki predstavlja podproces, ustvari nov vhod. Tip nastalega vhoda operatorja podprocesa in tip izhoda operatorja vhoda je ob nastanku nedoločen. V trenutku, ko izhod operatorja vhoda povežemo z vhodom nekega operatorja znotraj procesa, se spremenita tip izhoda in tip nastalega vhoda na operatorju podprocesa. V primeru, da je na vhod operatorja podprocesa že priključen izhod nekega drugega operatorja, se ponovno preveri, ali se izhod lahko prilagodi novemu tipu vhoda, in obvesti uporabnika.

Podobno kot operator, ki predstavlja vhod podprocesa, smo razvili operator, ki predstavlja izhod podprocesa. Ta operator ima en vhod. Ob nastanku pa primernemu operatorju, ki predstavlja podproces, v katerem se nahaja, doda izhod, ki je takšnega tipa, kot izhod operatorja, na katerega je povezan.

S tem smo dosegli, da podprocesi niso odvisni od procesa, v katerem so vsebovani, kar nam omogoča lažjo prenosljivost. Tak podproces lahko enostavno shranimo kot nov operator, ki bo ob svojen nastanku zgeneriral celotno hierarhijo podprocesov, kot da bi bil to en sam operator.

Vsi trije posebni operatorji, ki služijo za gradnjo podprocesov, nimajo pripadujoče skripte PHP, ki bi se zagnala na strežniku ob zagonu posameznega operatorja ali celotnega delotoka. Operatorji služijo le kot navodilo delu aplikacije JavaScript, ki krmili zaganjanje operatorjev (razdelek 4.2.9).

4.2.7 Generator operatorjev iz opisov spletnih servisov

Poseben operator, ki sproži klic spletnega servisa, lahko generiramo s pomočjo skripte PHP. Na dnu seznama operatorjev se nahaja gumb za uvoz spletnih servisov. Ob kliku na gumb se odpre posebno okno, v katerem se nahaja polje za vpis naslova opisa servisa WSDL.

Ob kliku na gumb za uvoz spletnega servisa se asinhrono pokliče skripta PHP, ki iz opisa razbere ime operacije, ki jo spletni servis ponuja, njene vhode in izhode. Za branje opisa smo uporabili knjižnico NuSoap, ki z opisa WSDL razbere vse vhode in izhode. Skripta PHP vrne prirejen opis v obliki XML, ki ga del aplikacije JavaScript uporabi pri gradnji naslednjega interaktivnega okna. Okno se odpre in uporabniku prikaže seznam vhodov, za vsakega od njih pa se uporabnik lahko odloči, kakšna naj bo njegova vloga (lahko je predstavljen kot vhod operatorja ali kot parameter operatorja).

Ob uporabnikovi potrditvi se na seznam operatorjev doda operator, ki kliče spletni servis. Vsi operatorji, ki uporabljajo spletne servise, kličejo isto skripto PHP. Od navadnih operatorjev se ločijo po tem, da imajo tudi skrite parametre, ki jih uporabnik ne vidi. Ti parametri so naslov spletnega servisa, naslov operacije in prevajalna tabela, ki za vsak vhod hrani podatek o imenu vhoda na strani spletnega servisa.

4.2.8 Možnost shranjevanja in nalaganja delotokov in posameznih operatorjev

Shranjevanje delotoka smo realizirali tako, da se v dokument XML zapišejo stanja vseh pomembnih notranjih spremenljivk programa. Ker vsak primer operatorja v sebi nosi dovolj informacije (ne potrebuje povezave na tip operacije, katerega primer ustvarja) je dovolj, da v standardizirano datoteko XML zapišemo vse aktivne operatorje in vse aktivne povezave.

V delu aplikacije JavaScript program naredi obhod po seznamu aktivnih operatorjev in po seznamu povezav. Primer zapisa XML za vse operatorje in povezave je prikazan na sliki 4.7.

```
<widgets>
    <widget>
       <top></top>
       <left></left>
       <id></id>
       <name></name>
       <icon></icon>
       <inputs>
           <input />
       </inputs>
       <outputs>
           <output />
       </outputs>
       <preferences>
           <pref />
       </preferences>
       <action></action>
       <canvas></canvas>
       <counter></counter>
       <type></type>
       <deleted></deleted>
    </widget>
</widgets>
<connections>
    <connection />
</connections>
```

Slika 4.7: Struktura dokumenta XML, ki vsebuje vse podatke o delotoku.

Zapis XML se ob shranjevanju zapisuje v spremenljivko, ki se po zaključku pisanja pošlje po protokolu HTTP POST skripti `save.php`, ki brskalniku vrne takšno glavo (*ang. header*), da uporabnika prisili v naložitev datoteke na uporabnikov računalnik (*ang. force download*).

Ob kliku na ikono za nalaganje delotoka se uporabniku odpre okno, v katerem izbere datoteko XML s svojega računalnika, ki se z zahtevkom HTTP POST pošlje na strežnik. V tem trenutku se aplikacija naloži na novo, le

da v sebi nosi informacijo o imenu datoteke, ki je bila naložena. JavaScript sproži zahtevek HTTP GET do naložene datoteke, jo prebere in iz datoteke XML zgradi delotok. Vsi operatorji imajo enake identifikacijske številke kot v shranjenem delotoku.

Shranjevanje in nalaganje posameznih operatorjev in podprocesov smo realizirali podobno, a s to razliko, da se aplikacija ne naloži ponovno ob nalaganju posameznega operatorja ali podprocesa. Naložitev datoteke je realizirana s pomočjo elementa `<iframe>`, saj asinhrono pošiljanje datotek s pomočjo objekta XMLHttpRequest ni mogoče.

Element `<iframe>` je pravzaprav okvir v katerega se naloži nov HTML dokument in se lahko ponovno nalaga neodvisno od okvirja samega [7], ki ostaja nedotaknjen.

V spletni aplikaciji se zahtevek HTTP POST pošlje elementu `<iframe>`, ki naloži datoteko na strežnik. Glavni okvir medtem preverja stanje in ob končanju nalaganja sproži funkcijo, ki zahtevek obdela in izbriše notranji okvir iz objektnega modela dokumenta. Okvir je za uporabnika neviden, zato daje nalaganje enak vtis kot pošiljanje datoteke s pomočjo Objekta XMLHttpRequest.

Datoteka XML, ki opisuje operator ali podproces se shrani v tabelo shranjenih uporabniških operatorjev in ime operatorja ali podprocesa se zapiše v seznam operatorjev, ki jih lahko dodajamo na kanvas. Ob kliku na njegovo ime se na kanvasu ustvari nov primer operatorja.

V primeru da gre za podproces je treba posebej paziti na pravilno označevanje operatorjev, ki sedaj niso več enaki kot v shranjeni datoteki XML. Med ustvarjanjem primera se ustvari prevajalna tabela, ki deluje kot funkcija, ki preslika shranjene identifikacijski številke v nove identifikacijske številke, ki se zaporedoma dodeljujejo novim operatorjem. S tem smo omogočili da lahko podproces shranimo in ga večkrat nanesemo na kanvas, neodvisno od že nanešenih operatorjev.

4.2.9 Zaganjanje delotokov in posameznih operatorjev

Glavni del aplikacije je zaganjanje operatorjev. Vsak operator lahko zaženemo na dva načina: z zagonom celotnega delotoka ali z zagonom operatorja samega.

Pred zagonom posameznega operatorja aplikacija preveri, ali je zadoščeno vsem predpogojem. Pogojem operatorja je zadoščeno natanko takrat, ko ne obstaja tak obvezen vhod, na katerem bodisi ni povezave, ali pa ko na izhodu operatorja, s katerim je povezan, ni podatkov. V primeru nezadoščanja pogojem se uporabniku pošlje sporočilo o napaki. Ko pa je pogojem zadoščeno, se

izvajanje lahko prične.

Izvajanje operatorja se vrši v obliki asinhronega zahtevka POST. Vsi podatki na vhodih in parametri operatorja se zakodirajo v zahtevek HTTP POST, ki se pošlje skripti, ki je navedena v operatorjevem atributu `<action>`. Operatorji tipa vhod, izhod in podproces so pri tem izjema, saj ne pride do asinhronega zahtevka POST. Ti operatorji le prenašajo podatke med delotokom samim. Podproces ob zagonu zažene vse svoje podoperatorje, čigar predpogojem je zadoščeno. Običajno so to operatorji, ki predstavljajo vhode. Ti operatorji ob zagonu na svoj izhod postavijo tiste podatke, ki se nahajajo na izhodu operatorja, ki je povezan na ustrezen vhod operatorja, ki predstavlja podproces. Ob zagonu operatorja, ki predstavlja izhod, se na izhod podprocesa prenesejo tisti podatki, ki se nahajajo na izhodih, ki so povezani na vhod obravnavanega operatorja.

Pred klicem operatorjeve funkcije se stanje operatorja spremeni v stanje izvajanja. Pod njegovo grafično podobo se prikaže animiran krog, ki ponazarja izvajanje. Ob neuspehu se grafična podoba operatorja obarva rdeče, ob uspehu pa se krog zamenja z zeleno kljukico, ki ponazarja, da so na vseh izhodih podatki in da se je operator uspešno izvedel. Ob uspešnem ali neuspešnem zagonu se stanje vsem naslednjim operatorjem (tj. tistim, ki so s svojimi vhodi posredno ali neposredno povezani na izhode obravnavanega operatorja) spremeni v neizvedeno (na izhodih nimajo aktualnih podatkov). Posebna lastnost operatorjev, ki predstavljajo podprocese, je ta, da med svojim izvajanjem ves čas preverjajo, ali imajo vsi njegovi podoperatorji podatke na izhodih. Preverjanje poteka vsakih sto milisekund, zato da ne obremenjujemo brskalnika, ki bi sicer postal neodziven.

Če uporabnik izbere izvajanje celotnega delotoka, se v primeru, da je del delotoka že bil izveden, uporabnika vpraša, ali naj se izvede celoten delotok ali le tisti operatorji, ki še niso bili izvedeni (preverjajo se stanja operatorjev). V primeru, da uporabnik izbere izvajanje celotnega delotoka, se vsem operatorjem stanje nastavi na neizvedeno. Aplikacija potem poišče vse operatorje, čigar predpogojem je zadoščeno, in jih zažene. Ob zaključku izvajanja teh operatorjev zopet poišče operatorje, ki se lahko izvedejo. Aplikacija to ponavlja, dokler takšni operatorji obstajajo.

4.3 Tok podatkov

Vsi podatki, ki prehajajo med operatorji, so nizi znakov. Ob vsakem zagonu operatorja se vsi podatki pošljejo v zahtevku POST operatorju, ta pa jih v

obliki niza znakov vrne delu aplikacije JavaScript. Niz znakov se ustrezno razdeli na različne izhode, kamor se podatki tudi shranijo (spomin brskalnika).

Knjižnica jQuery nam omogoča shranjevanje podatkov k elementom v objektnem modelu dokumenta. Na vsak primer izhoda se s posebnim ukazom jQuery shrani podatek, ki ga mora ta izhod hraniti. Podatki se vedno hranijo na izhodih. Vhodi se pri toku podatkov uporabljajo izključno za to, da hranimo informacijo o tem, na kateri izhod je poljuben operator povezan. Ob zagonu operatorja se torej preverijo vse vhodne povezave in s posebnim ukazom jQuery se preberejo vsi podatki na izhodih teh povezav, ki se skupaj s parametri, ki so hranjeni v ustreznem elementu `<form>`, pretvorijo v zahtevek POST, ki se pošlje skripti PHP.

Ob vsakem zagonu operatorja se izvršita dva prenosa podatkov. Prvi se izvrši ob klicu operatorja, drugi pa ob vračanju rezultatov. Vsak prenos podatkov pomeni fizičen prenos podakov z uporabnikovega računalnika na strežnik preko spletnega brskalnika (bolj podrobno preko zahtevka POST protokola HTTP).

Veliko število prenosov podatkov ne predstavlja problema, dokler je količina teh podatkov majhna. V primeru velikih količin podatkov se torej uporablja le ime začasne datoteke, ki se nahaja na strežniku.

Pri vsaki manipulaciji podatkov je potrebno posebno pozornost posvetiti ohranjanju konsistentnosti podatkov. Vsi operatorji, ki manipulirajo podatke, morajo narediti novo začasno datoteko in vse spremembe zapisati v to datoteko in posredovati kot izid to novo ime. S tem omogočimo, da izvirna datoteka ostane nedotaknjena, saj moramo uporabniku dopustiti možnost, da jo v prvotni obliki uporabi kot vhod kakšnega drugega operatorja. Vsi operatorji morajo torej upoštevati to pravilo, da lahko zapisujejo le nove datoteke, ne smejo pa spreminjati že obstoječih.

4.4 Strežniški del aplikacije

Strežniški del aplikacije je tisti del, do katerega uporabnik eksplicitno nima dostopa. Lahko se (in običajno se) nahaja na drugem računalniku, kot uporabnikov brskalnik, s katerim dostopa do grafičnega uporabniškega vmesnika. Strežniški del aplikacije za izvajanje skript uporablja interpreter PHP (opisan v razdelku 3.2.1).

4.4.1 Zaganjanje operatorjev in vračanje rezultatov

Vsakemu operatorju (razen operatorjev, ki so namenjeni upravljanju podprocesov) pripada datoteka PHP, v katerem je zapisana operacija, ki se mora izvesti ob izvajanju tega operatorja. V opisih operatorjev XML se ime datoteke nahaja v spremenljivki `action`, sama datoteka s tem imenom pa mora obstajati na strežniku. Ob zagonu operatorja se zahtevek HTTP POST pošlje na naslov, kjer se nahaja ta datoteka.

Operatorjeva skripta prejme vhode in parametre v tabeli POST. Ko se skriptin algoritem zaključi, vrne podatke v tekstovni obliki, ki jih loči s posebnim znakom. Na začetku rezultata sta lahko ključni besedi `RESULT` ali `OK`. Ob ključni besedi `OK` se podatki rekonstruirajo in shranijo na primerne izhode. Enako se zgodi pri ključni besedi `RESULT`, le da se sproži tudi odprtje okna, kjer se prikaže izid (to je namenjeno operatorjem, ki vizualizirajo podatke in izide).

Vsi ostali možni izidi, ki ne vsebuje teh dveh ključnih besed, se obravnavajo kot napaka v operatorju in se v statusni vrstici prikažejo uporabniku. Med takšne napake spadajo tudi napake, ki jih vrne interpreter PHP, zato lahko testiramo skripte PHP kar s pomočjo te spletne aplikacije.

Primer izida, ki ga vrne operator, čigar naloga je uvoz množice učnih primerov iz datoteke:

```
OK|temp001.tab
```

Na vhod in izhod lahko vpeljemo tudi kompleksnejše podatkovne tipe. Katerikoli objekt, ki ga uporablja skriptni jezik PHP, lahko pretvorimo v zaporedje (tj. enolično pretvorimo v niz znakov, iz katerega lahko kasneje rekonstruiramo enak objekt) in postavimo na izhod. PHP nudi funkcije za pretvarjanje v zaporednje in sestavljanje objektov iz zaporedja znakov, tako lahko v različnih operatorjih uporabljamo enake objekte. Primer takšnega objekta bi bilo odločitveno drevo, ki ga zgradi nek operator in za klasifikacijo uporabi drug operator.

4.4.2 Zagon spletnih servisov

Operatorji, ki so klici spletnih servisov, se od navadnih operatorjev razlikujejo samo po tem, da imajo tudi skrite parametre, ki jih uporabnik sam ne more nastavljati. Ti parametri so uporabljeni v skripti PHP, ki je zadolžena za klicanje spletnih servisov.

Med podatke, ki so hranjeni v skritih parametrih, spada naslov spletnega servisa, ime operacije, imena vhodov in ime izhodov. Vsak vhod, ki je predstavljen v delotoku, se pravilno preslika v vhod spletnega servisa (s pomočjo prevajalne tabele, ki vsako vrstno število vhoda preslika v ime vhoda na strani spletnega servisa). Ob klicu servisa se vrnejo rezultati, ki se morajo v pravilnem vrstnem redu shraniti na izhode. S pomočjo prevajalne tabele, ki imena izhodov pretvori v vrstni red izhodov, lahko izhode pravilno vrnemo delu aplikacije JavaScript.

Ob zagonu operatorja s spletnim servisom se del aplikacije JavaScript ne zaveda, da gre za takšen operator. Vhodi, izhodi in parametri delujejo na enakem principu kot običajni operatorji. Klic spletnega servisa in prirejanje vhodnih in izhodnih podatkov se zgodi v skripti PHP (`callwebservice.php`), ki je za to zadolžena.

Poglavje 5

Primeri uporabe in razširljivost

Spletno okolje za strojno učenje ima dve ciljni vrsti uporabnikov. Prva vrsta so uporabniki, ki želijo uporabljati funkcionalnosti, ki jih okolje nudi, druga pa so razvijalci, ki bi radi razširili funkcionalnosti okolja na način, ki ustreza njihovim potrebam. To poglavje vsebuje primere uporabe za obe vrsti uporabnikov in navodila za namestitev aplikacije na spletni strežnik.

5.1 Navodila za namestitev spletnega okolja

Za delovanje spletnega okolja je potreben spletni strežnik z interpreterjem PHP. Za razvoj in testiranje aplikacije smo uporabili brezplačni spletni strežnik Apache z vklopljenim modulom `mod_php`, ki vsebuje interpreter PHP.

Za hitro nastavitev na operacijskem sistemu Windows 7 smo uporabili paket Xampp, ki vsebuje Apache, PHP in podatkovno bazo MySQL v enem samem namestitvenem programu.

Po končani namestitvi in uspešnem zagonu spletnega strežnika je potrebno na strežniku ustvariti posebno mapo, v kateri želimo, da se nahaja okolje za strojno učenje. V to mapo naložimo datoteke, ki so sestavni del okolja. Datotečna struktura okolja je prikazana na sliki 5.1.

Pri nalaganju datotek moramo biti pozorni na to, da mora imeti spletni strežnik ustrezne pravice, da lahko ustvarja datoteke v mapi `uploads`, kamor se shranjujejo tudi začasne datoteke.

Okolje za strojno učenje po namestitvi zaženemo tako, da spletni brskalnik usmerimo na naslov, kjer se aplikacija na spletnem strežniku nahaja.

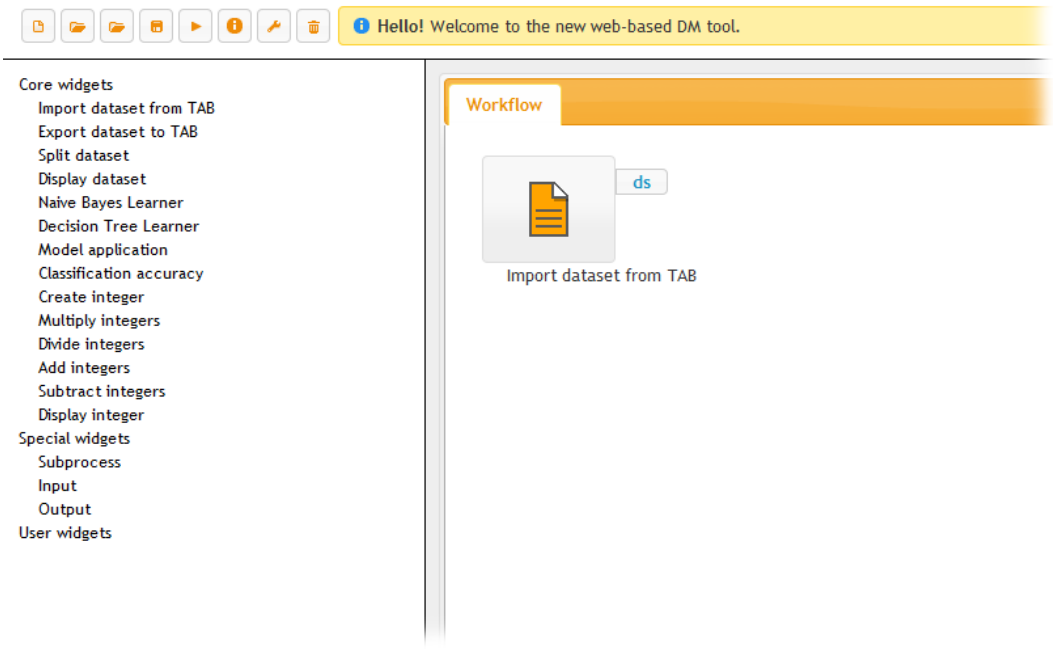
```
/css/style.css
/css/fileuploader.css
/css/reset.css
/css/jquery.contextmenu.css
/images/loading.gif
/js/fileuploader.js
/js/graphics.js
/js/jquery-1.4.2.min.js
/js/jquery-ui-1.8rc3.custom.min.js
/js/jquery.contextmenu.js
/js/script.js
/nusoap/nusoap.php
/uploads/
/widgets/callwebservice.php
/widgets/widgets.xml
/widget-icons/
/save.php
/savesub.php
/phpupload.php
/import.php
/index.php
```

Slika 5.1: Datotečna struktura okolja za strojno učenje.

5.2 Povezovanje operatorjev v delotoke in izvajanje delotokov

Uporabnik želi ustvariti in zagnati delotok, v katerem bo na neki množici podatkov izračunal klasifikacijsko točnost Naivnega Bayesovega klasifikatorja. S pomočjo grafičnega uporabniškega vmesnika izbere operatorje, ki jih za takšen delotok potrebuje, jih med seboj poveže in jih zažene.

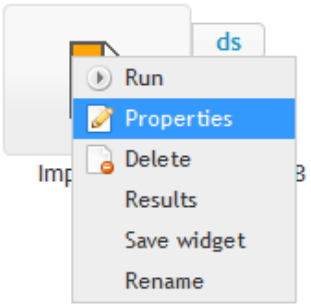
Na levi strani se nahaja množica operatorjev, iz katere uporabnik najprej izbere uvoz podatkov z datoteke TAB (*ang. import dataset from TAB*). Ob kliku na ta operator se na kanvasu nariše primer operatorja (slika 5.2), ki ga uporabnik lahko po kanvasu poljubno premika z držanjem miškinega gumba (*ang. drag and drop*).



Slika 5.2: Uporabnik s klikom na ime operatorja v levem seznamu povzroči, da se primer operatorja pojavi na kanvasu.

Uporabnik z desnim klikom na operator odpre kontekstni menu z večimi možnostmi (slika 5.3), od katerih izbere `preferences`. Odpre se mu posebno okno, v katerem se nahaja gumb za nalaganje datotek (*ang. upload a file*). Ob kliku na gumb se odpre posebno okno brskalnika, ki uporabniku omogoča izbiro datoteke z lastnega računalnika. Uporabnik z računalnika izbere da-

toteko `titanic.tab`, ki vsebuje podatke o potnikih ladje RMS Titanic (njihov potovalni razred, spol, starost in razredni atribut, ki nam pove, ali so preživeli potopitev ladje). Skrajšana različica te datoteke in splošen opis datoteke tipa TAB sta navedena v dodatku A.



Slika 5.3: S klikom na desni miškin gumb na operator uporabnik povzroči odprtje kontekstnega menuja z več možnostmi.

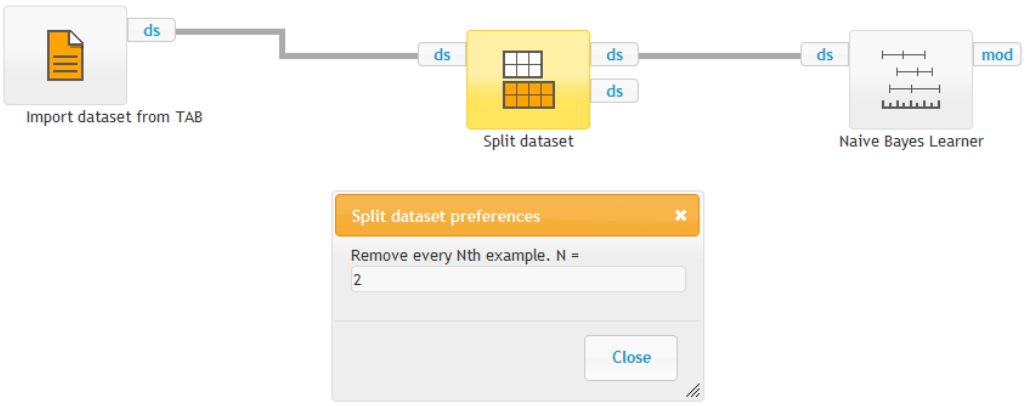
Datoteka se začne nalagati na strežnik po tem, ko jo uporabnik izbere. Nalaganje datotek je analogno pisanju vrednosti parametra v okno s parametri. Kljub temu, da se po nalaganju datoteka že nahaja na strežniku, pa v delotok še ni bila uvožena, saj se operator še ni pognal. Uporabnik lahko operator ročno zažene s pomočjo kontekstnega menuja ali pa zažene celoten delotok s klikom na gumb v orodni vrstici.

Uporabnik nadaljuje z nanašanjem operatorjev na kanvas. Odloči se, da bo množico primerov razdelil na testno in učno množico. Uporabil bo operator, ki razdeli množico primerov na dva dela. Operator deluje na takšen način, da vsak n-ti primer odstrani iz množice in ga vstavi v drugo množico. Vrednost n uporabnik nastavi s parametrom.

Po razdelitvi podatkov na učno in testno množico uporabnik na kanvas nanese operator, ki iz podanih učnih primerov zgradi model Naivnega Bayesovega klasifikatorja (slika 5.4).

Da bi uporabnik izračunal klasifikacijsko točnost takšnega modela mora najprej klasificirati testno množico podatkov. To naredi s pomočjo operatorja, ki izvrši model na podatkih (*ang. model application*). Na kanvas nanese ta operator in ga poveže z modelom, ki ga je generiral operator, ki gradi model. Na drugi vhod poveže izhod operatorja, ki je podatke razdelil na testno in učno množico (uporabi tisti izhod, ki ga ni uporabil za učenje).

Za izračun klasifikacijske točnosti uporabnik uporabi poseben operator,



Slika 5.4: Primer delotoka, ki zgradi model Naivnega Bayesovega klasifikatorja iz učne množice (tj. polovica vseh podatkov).

ki na vhodu prejme dve klasificirani množici podatkov (množica resničnih primerov in množica primerov, ki jih je klasificiral naivni Bayesov klasifikator). Celoten delotok je prikazan na sliki 5.5.

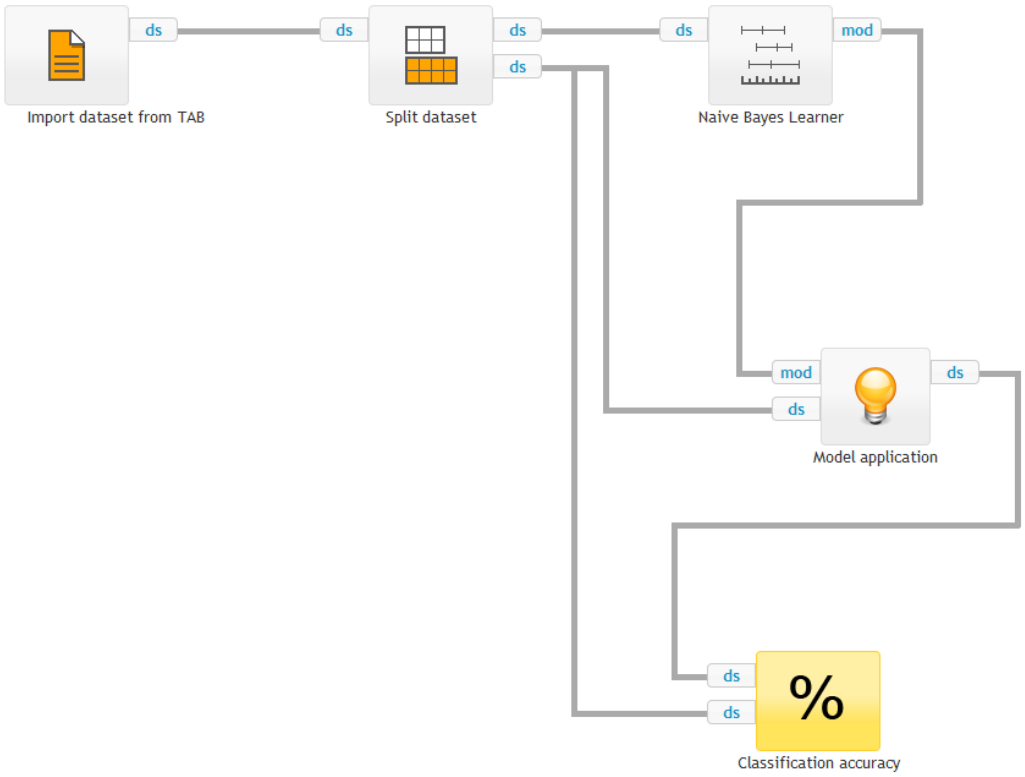
Uporabnik lahko zažene vsak operator posebej ali pa zažene celoten delotok. Ob zagonu se zaženejo tisti operatorji, ki se v danem trenutku lahko zaženejo (tisti, katerih predpogojem je zadoščeno in nimajo nepovezanih obveznih vhodov). Ob zaključku izvajanja se odpre okno z izidom, v katerem je napisan količnik pravilno klasificiranih primerov (slika 5.6).

V primeru, da uporabnik želi pregledati množico podatkov in sam primerjati klasifikacijo primerov, lahko uporabi operator, ki prikazuje podatke. Za vsako množico podatkov uporabnik ustvari svoj operatorja operatorja in jih ustrezno poveže na podatke in zažene. Izid takšnega delotoka je prikazan na sliki 5.7.

5.3 Dodajanje novega operatorja, ki je implementacija algoritma za strojno učenje

Dodajanje novega operatorja poteka v dveh korakih. V aplikacijo moramo dodati opis operatorja, nato pa sprogramirati še skripto PHP, ki se bo poklicala vsakič, ko bo operator zagnan.

V prvem koraku dodamo opis operatorja v datoteko `widgets.xml`, ki vsebuje podatke o vseh operatorjih. Primer opisa operatorja, ki implementira



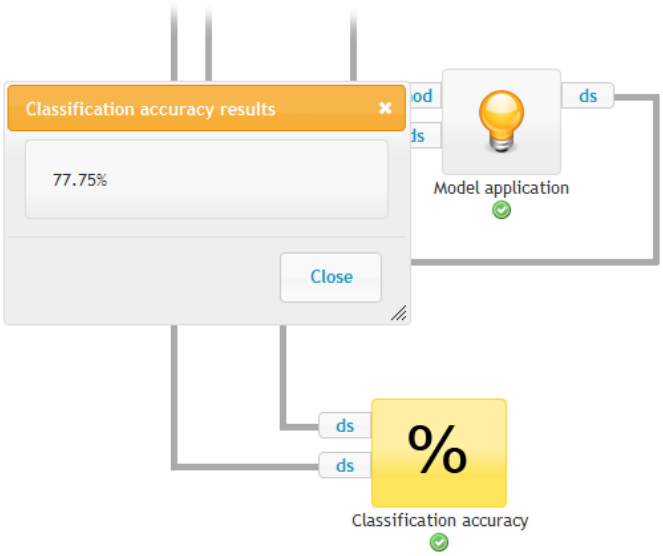
Slika 5.5: Primer delotoka, ki izmeri klasifikacijsko točnost Naivnega Bayesovega klasifikatorja na konkretnih podatkih.

algoritem za strojno učenje Naivni Bayes, je prikazan na sliki 5.8.

Operator ima en vhod in en izhod. Na vhod dobi učno množico podatkov, iz katere zgradi model, ki ga lahko posreduje drugim operatorjem, ki ga znajo uporabljati. V spremenljivki `action` se nahaja ime datoteke, ki jo bo interpretiral PHP, preden vrne rezultat brskalniku. Datoteka se mora nahajati v mapi `widgets`.

Za programiranje skripte lahko uporabimo vzorec skripte, ki prvi vhod preslika na prvi izhod (slika 5.9).

Vzorec na sliki 5.9 lahko razširimo tako, da odpre datoteko, katere ime je podano na vhodu, in iz nje prebere celotno množico podatkov. Za vsak razred se izračunajo ustrezne apriorne verjetnosti in pogojne verjetnosti za vsak atribut, ki se uporablja pri klasifikaciji. Vse verjetnosti shranimo v tabelo, ki jo pretvorimo v zaporedje in oddamo na izhod. Operator, ki bo model uporabljal, mora pravilno rekonstruirati model in ga ustrezno uporabiti pri klasifikaciji



Slika 5.6: Primer izida, ki ga vrne operator, ki meri klasifikacijsko točnost.

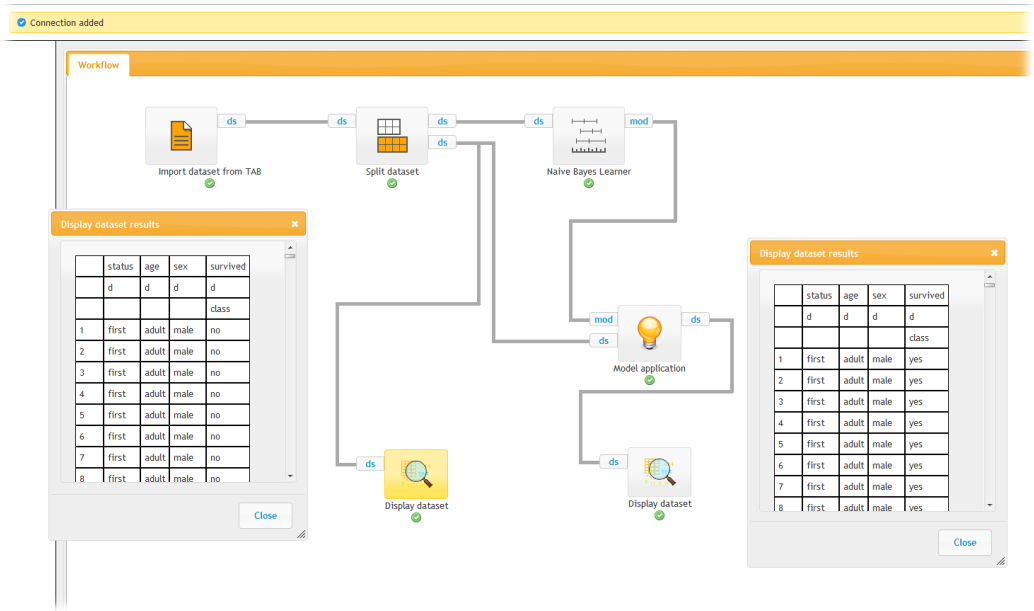
primerov. Primer implementacije algoritma Naivni Bayes, ki deluje z učnimi primeri, podanimi v obliki datoteke TAB, je prikazan v dodatku B.

5.4 Dodajanje in uporaba operatorjev s pomočjo spletnih servisov

Uporabnik želi ustvariti in zagnati delotok, v katerem bo uporabil spletni servis, ki služi lematizaciji besed različnih jezikov. Uporabnik sproži postopek uvoza spletnega servisa s klikom na gumb, ki se nahaja pod seznamom operatorjev. Ob kliku na gumb v pogovorno okno vpiše naslov spletnega servisa (slika 5.10).

Spletni servis za lematizacijo ima dva vhoda. Prvi vhod `inputText` služi za vnos vhodnega teksta, drugi vhod `language`, pa za vnos jezika, v katerem je napisan vhodni tekst. Uporabnik ve, da bo vhodni tekst dobil s pomočjo nekega drugega operatorja, zato v naslednjem pogovornem oknu določi, da bo vhod `inputText` pripeljal na vhod, vhod `language` pa bo vnesel kot parameter (slika 5.11).

Ob kliku na gumb za uvoz se v seznam operatorjev doda nov operator.



Slika 5.7: Delotok z dvema operatorjema, ki vizualizirata podatke v obliki tabel.

Ob kliku na ta operator se na kanvas doda operator `lemmatize`, ki uporablja spletni servis. S pomočjo ostalih operatorjev uporabnik zgradi delotok (slika 5.12) in ga zažene na enak način, kot je opisano v razdelku 5.2.

5.5 Postopek za postavitev spletnega servisa, ki implementira algoritem za strojno učenje

Uporabnik želi ustvariti spletni servis za uporabo v spletnem okolju za strojno učenje. Opisani spletni servisi WSDL niso odvisni od programskega jezika, zato ima uporabnik prosto izbiro programskega jezika.

V primeru da se uporabnik odloči, da bo spletni servis napisal v programskem jeziku PHP, lahko uporabi enostaven vzorec za splošni spletni servis (na sliki 5.13 je prikazano ogrodje kode za spletni servis Lemmatize). Če se datoteka PHP nahaja na naslovu `http://localhost/ws.php`, se njen opis WSDL nahaja na naslovu `http://localhost/ws.php?wsdl`.

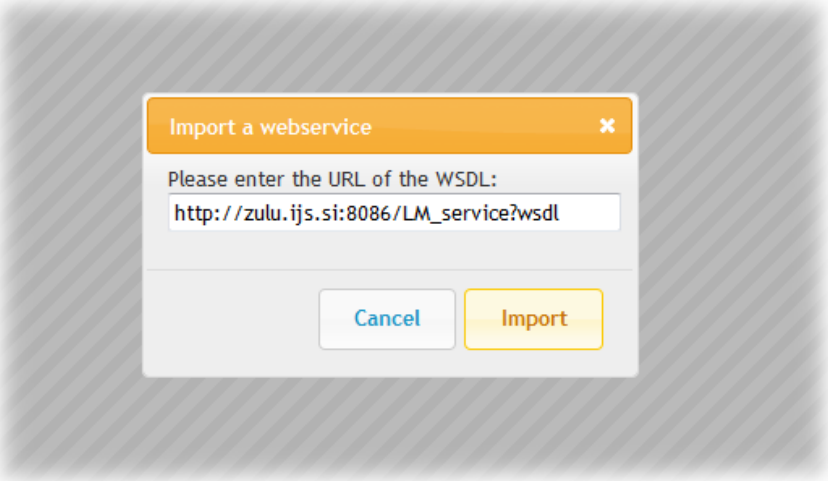
Uporabnik lahko z razširjanjem te kode (npr. z uporabo kode v dodatku B) ali z uporabo drugega programskega jezika implementira poljubni algoritem.

```
<widget>
    <name>Naive Bayes Learner</name>
    <action>naivebayes.php</action>
    <inputs>
        <input type='dataset' required='true'>ds</input>
    </inputs>
    <outputs>
        <output type='model'>mod</output>
    </outputs>
    <icon>naive-bayes.gif</icon>
</widget>
```

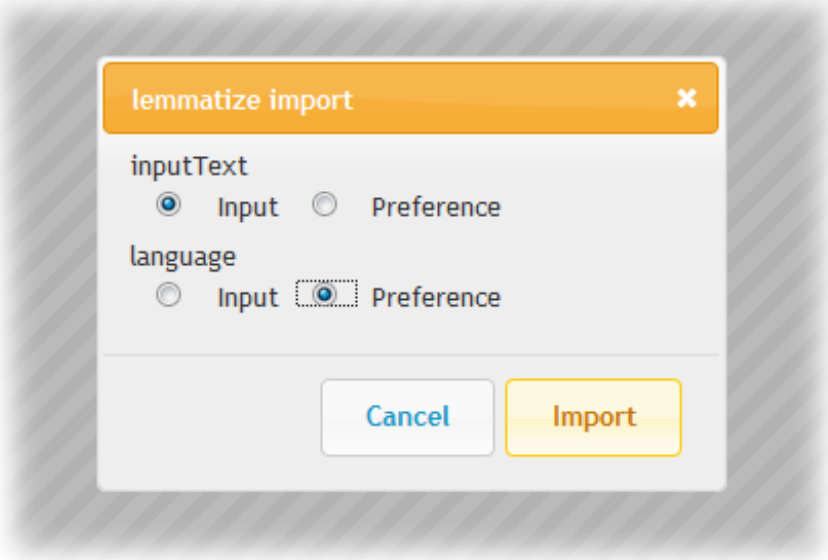
Slika 5.8: Opis operatorja XML, ki implementira algoritem Naivni Bayes.

```
<?php
\$output0 = \$_POST['input0'];
echo "OK|".\$output0;
?>
```

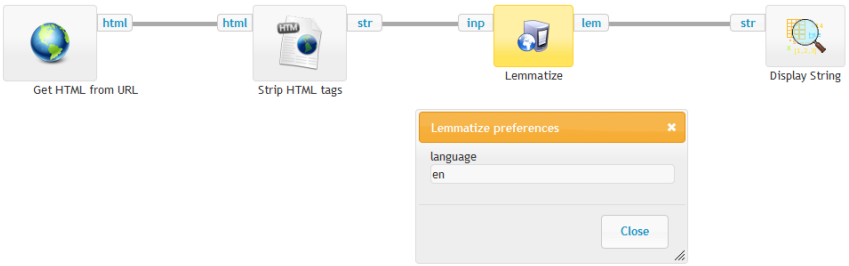
Slika 5.9: Vzorec skripte PHP, ki prvi vhod preslika na prvi izhod.



Slika 5.10: Pogovorno okno za vpis naslova, na katerem se nahaja opis spletnega servisa.



Slika 5.11: Pogovorno okno za nastavitev vlog vhodov spletnega servisa.



Slika 5.12: Delotok, katerega operator *lemmatize* uporablja spletni servis.

```
<?php
// Knjiznica NuSoap
require_once('nusoap.php');
// Ustvarimo novo instanco SOAP servisa
$server = new soap_server();
$server->configureWSDL('lemmatizewsdl', 'urn:lemmatizewsdl');
// Metoda Lemmatize
$server->register('lemmatize',              // ime metode
    array('inputText' => 'xsd:string',
          'language' => 'xsd:string'),      // vhodni parametri
    array('lemmatizedText' => 'xsd:string'),// izhodni parametri
    'urn:lemmatize',
    'urn:lemmatizewsdl#lemmatize',
    'rpc',
    'encoded',
    'Opis spletnega servisa'
);
// Php funkcija, ki naj se izvede ob klicu
function lemmatize($inputText,$language) {

  // opravi lematizacijo

}
if (isset($HTTP_RAW_POST_DATA)) {
    $server->service($HTTP_RAW_POST_DATA);
} else {
    $server->service('');
}
?>
```

Slika 5.13: Vzorec skripte PHP, ki deluje kot spletni servis.

Poglavje 6

Sklepne ugotovitve in ideje za nadaljnje delo

V diplomskem delu je opisan razvoj aplikacije za strojno učenje. Grafični uporabniški vmesnik aplikacije se zgleduje po že opisanih aplikacijah v drugem poglavju in vsebuje tudi nekaj unikatnih funkcij.

Aplikacija vsebuje urejevalnik delotokov, ki jih sestavljajo operatorji, ki jih med seboj lahko povezujemo in zaganjamo. Omogočeno je hkratno izvajanje večih algoritmov (izognili smo se težavam Orange4WS), uporabo spletnih servisov kot gradnikov delotokov in enostavno razširjanje aplikacije bodisi z dodajanjem algoritmov v spletno aplikacijo s pisanjem skript PHP, bodisi z uvažanjem spletnih servisov, ki so implementirani v poljubnem programskem jeziku. Operatorje lahko dodajamo tudi s shranjevanjem delotokov v operator, ki ga lahko nanašamo v nadaljnje delotoke (lahko pa so njegovi elementi tudi takšni operatorji), kar nam omogoča veliko svobodo in preglednost pri gradnji.

Aplikaciji v zaključni fazi manjka le velik izbor algoritmov, kot jih imajo druge aplikacije. Za nadaljni razvoj je potrebno vpeljati na enega od prej opisanih načinov veliko novih, koristnih operatorjev. Najlažje bi to naredili tako, da bi operatorje kakšne druge aplikacije (npr. Orange) zavili v spletne servise in jih uvožene uporabljali v aplikaciji.

Druga možnost za nadaljnji razvoj pa bi bila avtomatična izgradnja delotokov na podlagi načrtovanja in raziskovanja ontologije operatorjev. S tem bi uporabniku olajšali gradnjo delotokov in strojno učenje približali uporabnikom z manj znanja. Za takšno delo je potrebna velika množica operatorjev.

Aplikacija trenutno ni usposobljena za delo s tokovi. Za delo s tokovi in uporabo algoritmov za inkrementalno učenje bi bilo potrebno uvesti takšne funkcionalnosti, ki bi omogočile shranjevanje delotoka na strežnik, ki bi na

neko časovno enoto preverjal spremembe na virih (vhodnih operatorjih) in ob spremembah prožil tiste operatorje, katerih vhodi so se ob iteraciji spremenili. Dobro bi lahko izkoristili lastnost, da se spletna aplikacija nahaja na strežniku. Uporabnik bi sestavljen delotok shranil na strežnik in nastavil časovno obdobje, v katerem naj se izvaja. Takrat bi lahko svoj računalnik uporabljal v druge namene (brskalnik bi lahko zaprl) in po preteku časovnega obdobja bi preveril rezultate (ali jih celo dobil po elektronski pošti).

Na koncu pa bi omenili še, da kljub temu, da gre za spletno aplikacijo za strojno učenje, jo je mogoče generalizirati na splošno spletno aplikacijo za delo z delotoki. Namen aplikacije je odvisen od nabora operatorjev. Če bi se v operatorjih nahajali algoritmi za obdelavo slik (npr. odstranjevanje rdečih oči), bi lahko govorili o spletni aplikaciji za obdelavo slik. Preprosta razširljivost aplikacije omogoča, da postane takšna aplikacija, kot jo uporabniki potrebujejo. V primeru da bi aplikacijo razširili do te mere, da bi njeni uporabniki in razvijalci prosto objavljali svoje delotoke in razširitve, bi se njena uporabnost znala hitro razširiti na veliko področij obdelave podatkov.

Dodatek A

Opis in primer datoteke tipa TAB

Datoteka TAB je vrsta datoteka, ki vsebuje množico primerov z diskretnimi ali zveznimi atributi in enim razrednim atributom. V prvi vrstici datoteke so določena imena atributov, v drugi vrstici njihovi tipi (diskretni ali zvezni), v tretji vrstici pa je pod razrednim atributom napisana ključna beseda `class`. Vsak podatek je ločen s tabulatorjem. V naslednjih vrsticah pa je vsak primer ločen z novo vrstico, vrednosti atributov pa s tabulatorjem.

Sledi okrnjen primer datoteke, ki je navedena pri primeru uporabe spletnega okolja v razdelku 5.2:

```
status          age         sex         survived
d               d           d           d
                                        class
first           adult       male        yes
first           adult       male        no
first           adult       male        no
first           adult       female      yes
first           adult       female      yes
first           adult       female      yes
first           child       male        yes
second          adult       male        no
third           adult       male        yes
third           adult       male        no
crew            adult       male        no
crew            adult       male        no
```

Dodatek B

Implementacija Naivnega Bayesa za delovanje v spletni aplikaciji

```
<?php
function indexOf($needle, $haystack) {
    for ($i=0;$i<count($haystack);$i++) {
        if (trim($haystack[$i]) == trim($needle)) {
            return $i;
        }
    }
    return false;
}
if (file_exists("../uploads/".$_POST['input0'])) {
    $file="../uploads/".$_POST['input0'];
    $fh = fopen($file, 'r');
    $rawData = fread($fh, filesize($file));
    fclose($fh);
    $rows = explode("\n",trim($rawData));
    $attributeNames = explode("\t",$rows[0]);
    $types = explode("\t",$rows[1]);
    $roles = explode("\t",$rows[2]);
    $attributes = array();
    $classId = indexOf("class",$roles);
    $classYes = "";
    $classNo = "";
```

```
    $yesCount = 0;
    $noCount = 0;
    $count = 0;
    $row = explode("\t",$rows[3]);
    $classYes=$row[$classId];
    for ($j=4; $j<count($rows); $j++) {
        $row = explode("\t",$rows[$j]);
        if ($classYes!=$row[$classId]) {
            $classNo=$row[$classId];
            $noCount++;
        } else {
            $yesCount++;
        }
        $count++;
    }
for ($i=0; $i<count($attributeNames); $i++) {
    if ($i!=$classId) {
        $attribute = array();

        $attribute['name']=$attributeNames[$i];
        $attribute['id']=$i;
        $attribute['type']=$types[$i];
        $attribute['role']=$roles[$i];

        $attribute['yes']=array();
        $attribute['no']=array();

        $attributes[$i]=$attribute;
    }
}
$yesApriori = $yesCount/$count;
$noApriori = $noCount/$count;
for ($i=3; $i<sizeof($rows);$i++) {
    $row = explode("\t",$rows[$i]);
    foreach ($attributes as $attributeId => $attribute) {
        if ($row[$classId]==$classYes) {
            if (isset($attributes[$attributeId]['yes'][$row[$attributeId]])) {
                $attributes[$attributeId]['yes'][$row[$attributeId]]++;
            } else {
```

```
                $attributes[$attributeId]['yes'][$row[$attributeId]]=1;
            }
        } else {
            if (isset($attributes[$attributeId]['no'][$row[$attributeId]])) {
                $attributes[$attributeId]['no'][$row[$attributeId]]++;
            } else {
                $attributes[$attributeId]['no'][$row[$attributeId]]=1;
            }
        }
    }
}
foreach ($attributes as $attributeId => $attribute) {
    foreach ($attributes[$attributeId]['yes'] as $value => $count) {
        $attributes[$attributeId]['yes'][$value]=$count/$yesCount;
    }
    foreach ($attributes[$attributeId]['no'] as $value => $count) {
            $attributes[$attributeId]['no'][$value]=$count/$noCount;
        }
    }
    $model = array();
    $model['type']="bayes";
    $model['classYes']=$classYes;
    $model['classNo']=$classNo;
    $model['attributes']=$attributes;
    $model['yesApriori']=$yesApriori;
    $model['noApriori']=$noApriori;
    $model['classId']=$classId;
    echo "OK|".serialize($model);
} else {
    echo "Dataset not found";
}
?>
```

Slike

Literatura

[1] Demšar, J., Zupan, B.: "Orange: From Experimental Machine Learning to Interactive Data Mining", *White Paper (www.ailab.si/orange)*, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2004.

[2] Podpečan, V., Juršič, M., Žáková, M., Lavrač, N.: Towards a service-oriented knowledge discovery platform., SoKD'09, Bled, Slovenija, September 2009.

[3] Berthold, M., Cebron M., Dill F., Gabriel T., Kötter T., Meinl T., Ohl P., Sieb C., Thiel K., Wiswedel B.: "KNIME: The Konstanz Information Miner", *White Paper (http://www.inf.uni-konstanz.de)*, ALTANA Chair for Bioinformatics and Information Mining, Univerza Konstanz, 2007.

[4] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

[5] Skillicorn, D., Talia, D.: "Mining Large Data Sets on Grids: Issues and Prospects", *Computing and Informatics*, št. 21 zv. 4, str. 347-362, 2002.

[6] Domenico Talia, Paolo Trunfio, Oreste Verta: "Weka4WS: a WSRF-enabled Weka Toolkit for Distributed Data Mining on Grids" v zborniku *9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005)*, Porto, Portugal, Oktober 2005, LNAI št. 3721, str. 309-320, Springer-Verlag, 2005.

[7] S. Pemberton et al.: "XHTML™1.0 The Extensible HyperText Markup Language (Second Edition): A Reformulation of HTML 4 in XML 1.0", *W3C Recommendation (http://www.w3.org/TR/2002/REC-xhtml1-20020801)*, Avgust 2002.

[8] B. Bos, H. W. Lie, C. Lilley, I. Jacobs: "Cascading Style Sheets, level 2 (CSS2) Specification" *W3c Recommendation (http://www.w3.org/TR/CSS2/)*, September 2009.