UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aleš Mrak

Okolja za odkrivanje znanj iz podatkov v programskem jeziku Python

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Blaž Zupan

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1] V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [**?**]

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

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

Št. naloge: 00351/2012
Datum: 14.09.2012

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

Kandidat: **ALEŠ MRAK**

Naslov: **OKOLJA ZA ODKRIVANJE ZNANJ IZ PODATKOV V PROGRAMSKEM JEZIKU PYTHON**

PYTHON DATA MINING ENVIRONMENTS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi preglejte nekaj tipičnih programskih okolji, ki omogočajo uporabo algoritmov za odkrivanje znanj v podatkih. Okolja
naj imajo vmesnik za programski jezik Python. Poročajte o metodah, ki jih ta okolja implementirajo, o enostavnosti namestitve in primernosti dokumentacije ter o način uporabe. Izdelajte tudi krajšo študijo o odzivnih časih uporabe različnih metod v teh orodjih.

Mentor:

prof. dr. Blaž Zupan

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Aleš Mrak, z vpisno številko **24930479**, sem avtor diplomskega dela z naslovom:

Okolja za odkrivanje znanj iz podatkov v programskem jeziku Python

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Blaža Zupana,
- 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 23 julija 2012 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Blažu Zupanu za predloge ter strokovno pomoč pri izdelavi diplomske naloge ter za vse znanje, ki se ga od njega prejel v času študija.

Iskrena hvala gre tudi mojim staršem in ženi Maji za moralno in finančno podporo skozi vsa študijska leta.

Kazalo

Povzetek

Cilj diplomske naloge je primerjava sistemov za odkrivanje znanj iz podatkov (*data mining*), ki imajo vmesnik v programskem jeziku Python. Veliko odprtokodnih sistemov za odkrivanje znanj iz podatkov so implementirali knjižnice ter njihove programske vmesnike s programskim jezikom Python. Izbrali so ta skriptni programski jezik, ker je hiter in omogoča objektno programiranje, vključevanje drugih programskih knjižnic v Python in je implementiran v vseh glavnih operacijskih sistemih (*Windows, Linux/Unix, OS/2, Mac, itd.*). Naša analiza sistemov za odkrivanje znanj iz podatkov je zajela sedem najbolj uporabljenih sistemov (*Elefant, Mdp, OpenCVLibrary, Orange, Pybrain, Pyml in Shogun*). Preučili smo podporo algoritmov za odkrivanja znanj iz podatkov, podporo podatkovnih formatov, programski vmesnik (*GUI in API*), večopravilnost, podporo podatkovnih baz, odzivne čase ter ostale vidike kot so namestitev sistema, dokumentacij in podpora uporabnikov. Iz te analize smo tudi ugotovili kaj so skupne pomanjkljivosti analiziranih knjižnic in smo podali nekaj priporočil razvijalcem teh sistemov.

Abstract

In the thesis we compare the systems for data mining that have an interface in the programming language Python. Many open-source systems for data mining and library had implemented their software interfaces to the Python programming language. They choose Python because it is fast and provides object-oriented programming, allows for the integration of other software libraries in Python and is implemented in all major operating systems (*Windows, Linux / Unix, OS / 2, Mac, itd.*). Our analysis systems for data mining covers seven most used systems (*Elefant, MDP, OpenCVLibrary, Orange, Pybrain, Pyml and Shogun*). The analysis covered the following properties of the systems data formats, application programming interface (*GUI and API*), multitasking, support for databases, response times and other aspects such as installation, documentation and support for the users. From this analysis, we also find out what are the common shortcomings of the analyzed libraries and we give some recommendations to developers.

Poglavje 1

Uvod

V sedanjem sodobnem svetu, v katerem prevladujejo IT tehnologije, se združbe posebej srečujejo s problemom preobilice podatkov, ki jih pridobijo čez čas z uporabo različnih informacijskih sistemov. Znanstveniki s podatki, ki jih zberejo s preizkusi in ponavljajočimi se operacijami (*npr. vremenski podatki, podatki o genih, itd.*), komercialne združbe kot so banke, trgovine in ostali pravno formalni subjekti, ki z uporabo BI (*Business intelligence*) in ostalih informacijskih sistemov kot so *CRM*, *Campaign* ter *ERP* sistemov, shranjujejo podatke o strankah in njihova obnašanja. Prav tako javne in tajne varnostne organizacije shranjujejo velike količine podatkov o subjektih, ki jih nadzirajo in kontrolirajo. Tako velikih količin podatkov ni možno analizirati brez uporabe novih informacijskih tehnologij, zato se je v koncu prejšnjega stoletja pojavila nova veda za računalniško analizo podatkov, ki so jo imenovali podatkovno rudarjenje (angl. *Data mining*). Te tehnologije nam omogočajo avtomatizirano iskanje novih informacij ali relacij v kopici podatkov. Obstaja več metod podatkovnega rudarjenja. Diplomska naloga se usmerja na področje orodij, s katerimi lahko to kopico podatkov uporabimo za namene podatkovnega rudarjenja, specifično z uporabo programskega jezika Python. Ker je Python odprtokodni sistem so tudi rešitve, ki uporabljajo Python tudi v večini odprtokodni sistemi, ki jih lahko uporabljajo vsi brez restrikcij in komercialnih licenc. Namen

diplomske naloge je ocena teh sistemov po naslednjih kriterijih:

- enostavnost in podpora namestitve sistema
- podpora različnih platform
- dokumentacija sistema
- podpora CRISP DM metodologije
- struktura in uporaba programskega vmesnika
- podpora algoritmov in podatkov
- primerjava časov procesiranja podatkov

Diplomska naloga podaja rezultate pregleda vseh zgoraj omenjenih kriterijev za vsak sistem posebej ter njihovih funkcionalnostih, z možnimi priporočili razvijacem.

Poglavje 2

Opis analiziranih sistemov

Za primerjalno študijo smo pregledali sedem najbolj uporabljenih in priljubljenih programskih sistemov za odkrivanje znanj iz podatkov, ki imajo vmesnik v programskem jeziku Python. Sistemi, ki ustrezajo našemu kriteriju so naslednji:

1. Elefant (*http://elefant.developer.nicta.com.au/*)
2. Mdp (*http://mdp-toolkit.sourceforge.net/*)
3. OpenCV (*http://opencv.org/*)
4. Orange (*http://orange.biolab.si/*)
5. Pybrain (*http://pybrain.org/*)
6. Pyml (*ttp://pyml.sourceforge.net/*)
7. Shogun (*http://www.shogun-toolbox.org/*)

Omenjeni sistemi se med seboj razlikujejo tako po zmožnostih kot po sami arhitekturi. Določeni sistemi so bolj splošni in ne podpirajo samo algoritme za podatkovno rudarjenje, slednje posebej velja za knjižnico OpenCV, ki je namenjena predvsem za multimedijske vsebine, vendar ima podporo za algoritme iskanja znanj. Kot so si ti sistemi različni v implementaciji algoritmov so si raznoliki tudi v implementaciji navodil za uporabo. Predvsem navodil, ki se nanašajo na testne primere, namestitve in navodil programskega vmesnika za razvijalce. Pomembne razlike v teh sistemih

se najbolj odražajo v zgradbi API-ja in integracijo vhodno izhodnih podatkov v sistem ter razlike, ki se nanašajo na (ne)uporabnost uporabniškega grafičnega vmesnika (*GUI*) in odzivnih časov algoritmov, ki jih implementirajo.

V naslednjih podpoglavjih bomo prvo bolj na splošno opisali prej omenjenih sistemov za odkrivanje znanj iz podatkov. Opis bo vseboval zmožnosti in namen posameznih knjižnic, kratko zgodovino razvoja kjer so ti podatki objavljeni. Raziskali bomo kakšno je zadnje stanje ažurnosti razvoja omenjenih sistemov.

Najprej se bomo osredotočili na metodologije, ki jih orodja podpirajo ali ne podpirajo metodologijo CRISP DM (*Cross Industry Standard Process for Data Mining*).

2.1 Metodologija CRISP DM

Na področju odkrivanja znanj iz podatkov obstaja več metodologij za podatkovno rudarjenje, ki nam omogočajo, da se postopki pri iskanju rešitev iskanja znanj opravi po nekem postopku, ki je formalno zapisan in določa standardne korake, ki so potrebni pri uporabi orodij za odkrivanja znanj iz podatkov. Obstajajo različne metodologije in najbolj razširjene metodologije so naslednje:

- CRISP DM (Cross Industry Standard Process for Data Mining),
- SEMMA (sample, explore, modify, model, assess),
- KDD proces (Knowledge Discovery in Databases).

Od zgoraj naštetih se najbolj uporablja CRISP DM ali *Cross Industry Standard Process for Data Mining*. Temelje za nastanek CRISP-DM standarda so postavila podjetja DaimlerChrysler, SPSS in NCR. Leta 1997 so oblikovala konzorcij s ciljem razviti standardni industrijski proces za podatkovno rudarjenje. Namenjen naj bi bil za uporabo v kateremkoli okolju, neodvisno od programskega orodja in gospodarskega področja. Znanje o

praktičnih delovnih procesih na tem področju ter mnenja o tem, kako le-te izboljšati, so pridobili na odprtih delavnicah . Rezultat dela konzorcija je standard CRISP-DM 1.0, ki je nastal leta 2000.

Kot je prikazano na sliki 2.1 vidimo, da CRISP DM metodologijo sestavlja šest stopenj in poti sprehajanj med njimi ter zunanji krog, ki simbolizira iterativni proces podatkovnega rudarjenja. Razumevanje poslovnega procesa je stopnja, ki določa predpogoje in namen, da se lahko lotimo odkrivanju znanj. Ta proces zajema naslednje korake:

- določitev poslovnih ciljev,
- ocena trenutnega stanja,
- določitev ciljev podatkovnega rudarjenja,
- izdelava projektnega načrta.

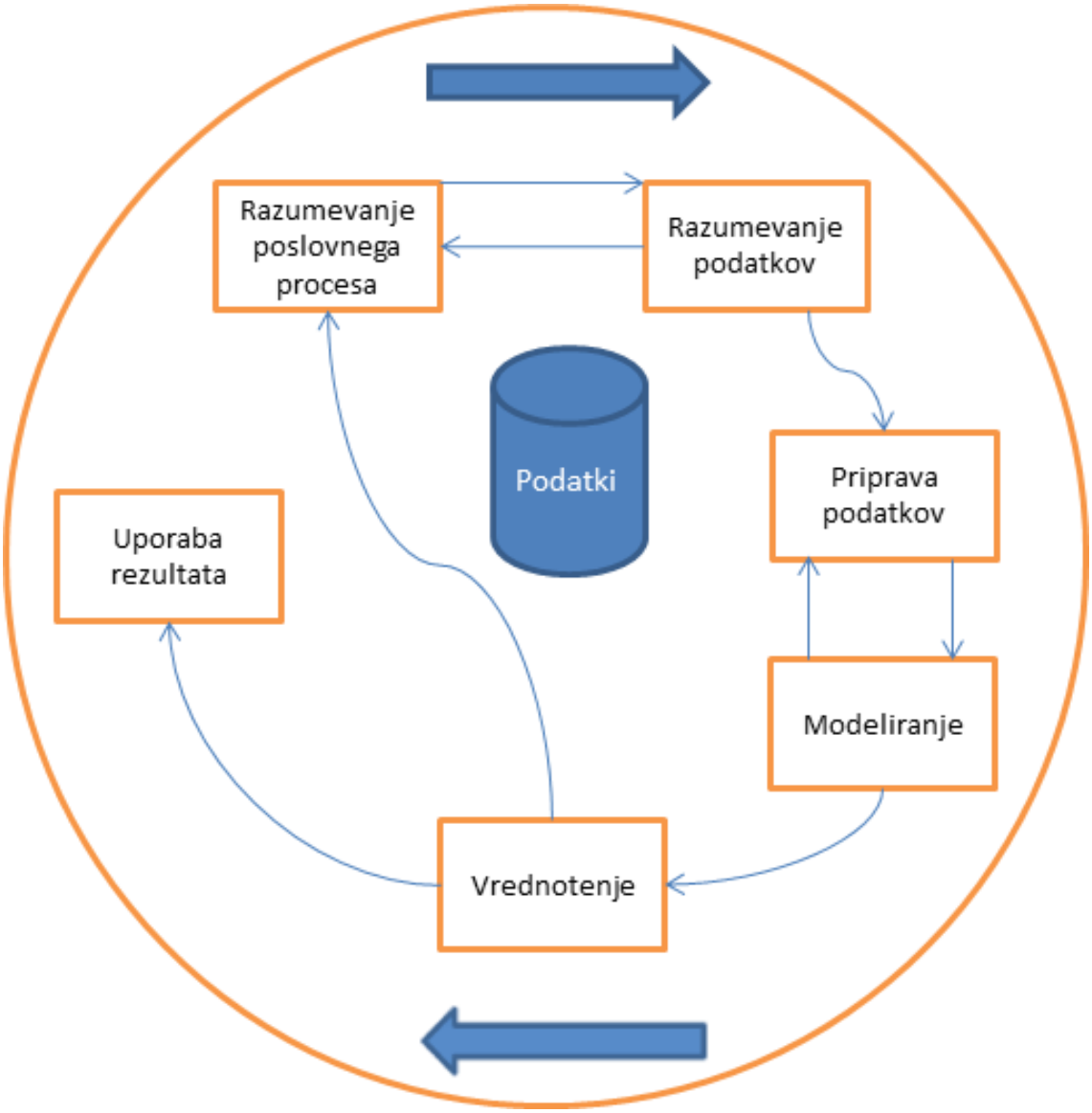
Razumevanje podatkov je stopnja v kateri se določi katere podatke se bo uporabljalo za rudarjenje. Stopnja zajema:

- združevanje podatkov,
- opis podatkov in njihove strukture,
- podrobno raziskavo podatkov,
- ocena kakovosti podatkov.

Priprava podatkov je stopnja v kateri se izvedejo aktivnosti, ki pripeljejo do končne zbirke podatkov. Ti podatki se uporabijo v 4. Razvojni stopnji za izdelavo modela. Ta stopnja zajema:

- čiščenje podatkov,
- izdelava podatkovne strukture,
- integracija zapisov,
- ureditev njihovih vrednosti.

Modeliranje je stopnja v kateri se preizkusi več tehnik ali algoritmov za izdelavo modelov, ki so uporabni za obravnavani poslovni proces. Modeliranje zajema:



Slika 2.1: Razvojne stopnje metodologije *CRISP-DM*.

- izbira tehnik,
- načrt modeliranja,
- izdelava modelov,
- ocenitev modelov.

V stopnji vrednotenja rezultatov ocenjujemo, v kolikšni meri je model usklajen s poslovnimi cilji. Zanima nas, kako dobro model rešuje poslovni problem na novih podatkih. Delovanje modela preskusimo na čim večji množici podatkov. Koraka, ki sta potrebna v stopnji, sta:

- pregled procesa,
- določitev naslednjih korakov.

Stopnja uporaba rezultata je vezana na cilj rudarjenja znanj, ki smo ga določili v prvi stopnji, se odraža po navadi v preprostem poročilu ali preglednici.

2.2 Elefant

Elefant (*Efficient Learning, Large-scale Inference, and Optimisation Toolkit*) je odprtokodni sistem za strojno učenje, ki uporablja licenco MPL (*Mozilla Public Licence*). Sistem implementira algoritme za strojno učenje, ki uporablja moč večprocesorskih operacijskih sistemov kot so Windows, Unix, Linux in Mac OS X. Knjižnica ima implementiran tudi lastni grafični uporabniški vmesnik (*GUI*) za uporabnike, ki hočejo narediti hitre prototipe eksperimentov.

Domača spletna stran sistema Elefant je na naslovu *http://elefant.forge.nicta.com.au*. Elefant postreže tudi vadnice (angl. *tutorials*) o statističnem strojnem učenju na njihovih spetnih straneh (*http://sml.nicta.com.au/isml09.html*) ter dokumentacijo. Sistem je naredila avstralska skupina Statistical Machine Learning Group (*http://sml.nicta.com.au/*) na NICTA centru (*http://www.nicta.com.au*) in podpira naslednje operacijske sisteme Windows (*XP SP2, Vista, 7*), Linux in Max OS X (*10.5*).

Sistem je bil nazadnje ažuriran na datum 17.10.2009 z različico 0.4 kar pomeni, da se sistem verjetno ne razvija več. Elefant podpira samo različico Python 2.6.

2.3 Mdp

Mdp (*Modular toolkit for Data Processing*) je odprtokodni sistem za strojno učenje, ki uporablja licenco BSD (*Berkeley Software Distribution*). Sistem lahko poganjamo v okoljih Linux, Windows ter MacOSX operacijske sisteme. Domača spletna stran sistema Mdp je na naslovu *http://mdp-toolkit.sourceforge.net*. Knjižnica nima lastnega uporabniškega grafičnega vmesnika. Z vidika uporabnika je MDP zbirka nadzorovanih in nenadzorovanih učnih algoritmov in drugih podatkovnih procesnih enot, ki so lahko združeni v obdelavno podatkovna zaporedija in bolj zapletene feed-forward omrežne arhitekture. Z vidika znanstvenega razvijalca je MDP modularni okvir, ki se ga lahko zlahka razširi. Izvajanje novih algoritmov je enostavno in intuitivno, nove implementirane enote ali *units* se samodejno vključujejo v osnovni del knjižnice. Baza razpoložljivih algoritmov se stalno povečuje in vključuje metode obdelave signalov, klasifikatorje in probabilistične metode ter mnogo drugih algoritmov. Mdp podpira paralelizem z implementacijo istočasnega izvajanja paralelnih tokov. Podpira tudi paketno obdelavo podatkov (batch processing) za obdelovanje velikega nabora podatkov. Avtorja prvotnega Mdp sta Pietro Bekes in Tiziano Zito in sta MDP razvila na Inštitutu za teoretično biologijo na Univerzi Humboldt v Berlinu leta 2003. Sistem je bil nazadnje ažuriran 24.10.2011 z različico 3.2 kar verjetno pomeni, da se sistem še naprej razvija. Mdp lahko teče v vseh novejših različicah jezika Python.

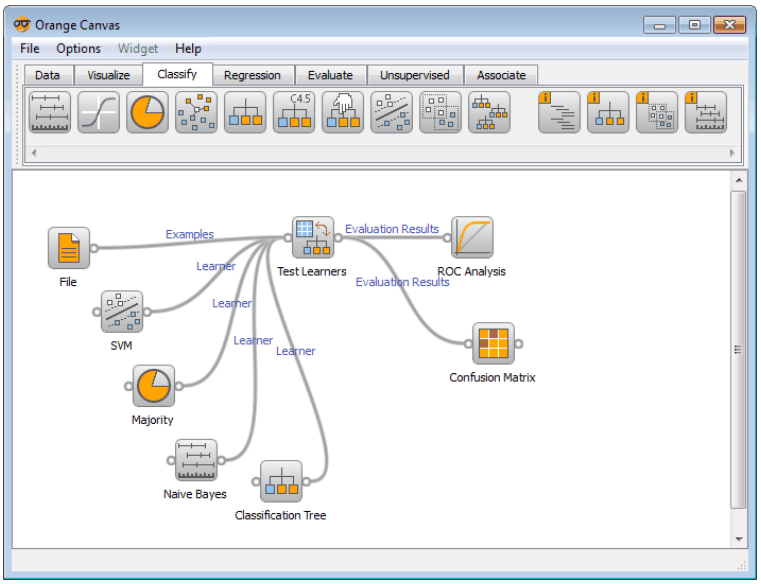
2.4 OpenCVLibrary

Projekt so prvič registrirali na *sourceforge.net* leta 2001. Razvijalci so jo opisali kot knjižnico z več kot 500 različnimi algoritmi, dokumentacijo in primeri uporabe za sisteme realno časovnega računalniškega vida. CV je kratica za *Computer Vision* (računalniški vid) kar pomeni, da je osnovni namen knjižnice za podporo razvijalcem pri razvijanju aplikacij za računalniški vid. Domača spletna stran OpenCV je na naslovu *http://opencv.org*. Knjižnica OpenCV je odprtokodni sistem za strojno učenje, ki uporablja licenco BSD (*Berkeley Software Distribution*). Sistem teče v okoljih Linux, Windows, Android ter MacOSX. Sistem nima grafičnega uporabniškega vmesnika. Danes ima knjižnica že več kot 2500 optimiziranih algoritmov in ima vmesnike za programske jezike C++, C, Python in kmalu tudi za Javo. Knjižnica je napisana v jeziku C in C++. Sama knjižnica vsebuje splošno knjižnico MLL (Machine Learning Library), ki se jo lahko uporablja za strojno učenje. V tej diplomskem delu se bomo osredotočili na knjižnico MLL ter dostopa do te knjižnice preko Python vmesnika. Za knjižnico OpenCV obstaja dve objavljeni knjigi in ti sta *Learning OpenCV* avtorjev Gary Bradski, Adrian Kaehler in *OpenCV 2 Computer Vision Application Programming Cookbook* avtorja Robert-a Laganière-ja. Sistem je bil nazadnje ažuriran na datum 7.5.2012 z različico 2.4 kar lahko pomeni, da se sistem še naprej razvija. Knjižnica OpenCV podpira različici Python 2.6 in 2.7.

2.5 Orange

Orange je celovita komponentna programska oprema za strojno učenje in podatkovnega rudarjenja razvita na Univerzi v Ljubljani, v sodelovanju z odprtokodno skupnostjo. Domača spletna stran sistema Orange je na naslovu *http://orange.biolab.si*. Sistem uporablja licenco GNU (*General Public License*) oziroma njeno različico 3 in naprej. Knjižnica vključuje obsežen sklop sestavnih delov za priprave podatkov, funkcij merjenj dosežkov in

filtriranj, modeliranj, modelov ocenjevanj in raziskovanalnih tehnik. Knjižnica je implementirana v jeziku C++ (*hitrost*) in v programskem jeziku Python (*prožnost*). Orange omogoča analize in podatkovno vizualizacijo za začetnike ter eksperte na področju podatkovnega rudarjenja. Za hitro konfiguracijo sistema ima Orange implementiran svoj uporabniški vmesnik, ki omogoča vizualno programiranje ter uporabo skriptnega jezika Python in je implementiran na knjižnici Qt, ki je podprta na različnih platformah. Orange programski sistem omogoča enostavno implementacijo in integracijo dodatkov (*add-on*) v sistem. Začetki knjižnice Orange segajo v leto 1996 ko sta Univerza v Ljubljani in Inštitut Jožef Štefan začela razvoj sistema ML* sistema za strojno učenje, ki je implementiran v jeziku C++. Leta 1997 je bila dodana podpora za skriptni jezik Python sistemu ML* in dodani novi moduli napisani v tem jeziku s katerim se je sistem preimenoval v Orange. V naslednjih letih so bili algoritmi za strojno učenje napisani v jeziku C++ ali Python. Že leta 2002 se je pojavil prvi prototip novega grafičnega uporabniškega vmesnika s pomočjo knjižnice Pmw. Leta 2003 se je razvil novi grafični uporabniški vmesnik na osnovi knjižnice PyQt Python, ki je omogočal razvoj gradnikov (angl. *widgets*) in povezavo le teh v cevovode (angl. *pipelines*). Leta 2005 je bila dodana razširitev za analizo podatkov za bio-informatiko. Leta 2008 je bila implementirana namestitev za platformo Mac Os X. Leta 2009 je bilo že narejeno in podprtih več kot 100 gradnikov in od tega leta naprej pa je že bila narejena različica 2.0 za Orange in na domači strani ponuja namestitvene pakete, ki slonijo na dnevnem prevajanju kode. Orange podpira različne različice Linux okolij, Apple-ov MAC OS X in različne različice Microsoftovih Windows-ih. Na sliki 2.2 je prikazan primer uporabe Orange grafičnega uporabniškega vmesnika na praktičnem primeru. Sistem se dnevno ažurira in zadnja različica Orange za Windows okolje za časa pričetka pisanja analize je 2.5a4 narejena dne 19.3.2012. Sistem Orange podpira različice Python 2.5, 2.6 in 2.7.



Slika 2.2: primer interakcije objektov v Orange.

2.6 PyBrain

PyBrain je okrajšava za *Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network*. Knjižnica je bila razvita na inštitutu IDSIA (*Istituto Dalle Molle di Studi sull'Intelligenza Artificiale*) v Švici in TUM (*Technische Universität München*) v Nemčiji. Domača stran knjižnice PyBrain je na internetni strani *http://pybrain.org*. Knjižnica podpira Linux, Windows, ter MacOSX operacijske sisteme in uporablja licenco BSD (*Berkeley Software Distribution*) in nima implementiranega lastnega grafičnega uporabniškega vmesnika. PyBrain je modularna knjižnica za strojno učenje v okolju Python. Njen cilj je ponuditi prilagodljiv in preprost način uporabe in vedno močnejših algoritmov strojnega učenja v nalogah in vnaprej določenih različnih okoljih. Knjižnica PyBrain omogoča tudi testiranje in primerjanje rezultatov algoritmov med seboj. Čeprav obstaja kar nekaj knjižnic za strojno učenje, knjižnica PyBrain želi biti modularna knjižnica, ki je zelo enostavna za uporabo in ki bi jo lahko uporabljali študenti na vstopni ravni s tem, da še vedno ponuja prilagodljivost in algoritme za

raziskave, ki zahtevajo večjo kompleksnost. Sistem je bil nazadnje ažuriran dne 16.2.2012 z različico 0.3.1-3 kar pomeni, da se sistem še vedno razvija naprej vendar sta osnovni del programske kode in dokumentacija še vedno iz dne 12.11.2009, kar nakazuje, da ni bilo velikih sprememb ali dodanih novih funkcionalnosti. Knjižnica PyBrain podpira različice Python 2.5, 2.6 in 2.7.

2.7 PyML

PyML je interaktivno objektno usmerjen sistem za strojno učenje napisan v okolju Python. PyML se osredotoča na SVM (*Support Vector Machines*) algoritme za klasifikacijo in regresijo in drugih metod na osnovi jeder. Knjižnica podpira Linux operacijske sisteme, Mac OS X in ne podpira Microsoft Windows operacijskega sistema, ker knjižnica PyML uporablja določeno kodo v C++ jeziku, ki je ni mogoče avtomatično prenesti v Windows okolje. Knjižnica uporablja licenco LGPL (*Lesser General Public License*) in nima implementiranega lastnega grafičnega uporabniškega vmesnika. Domača spletna stran knjižnice Pyml se nahaja na naslovu *http://pyml.sourceforge.net*. Knjižnica je bila razvita v laboratoriju bioinformatike na fakulteti Colorado State University. Sistem je bil nazadnje ažuriran dne 7.10.2011 z različico 0.7.9 kar lahko pomeni, da se sistem še naprej razvija. Knjižnica PyML podpira različice Python 2.5, 2.6 in 2.7.

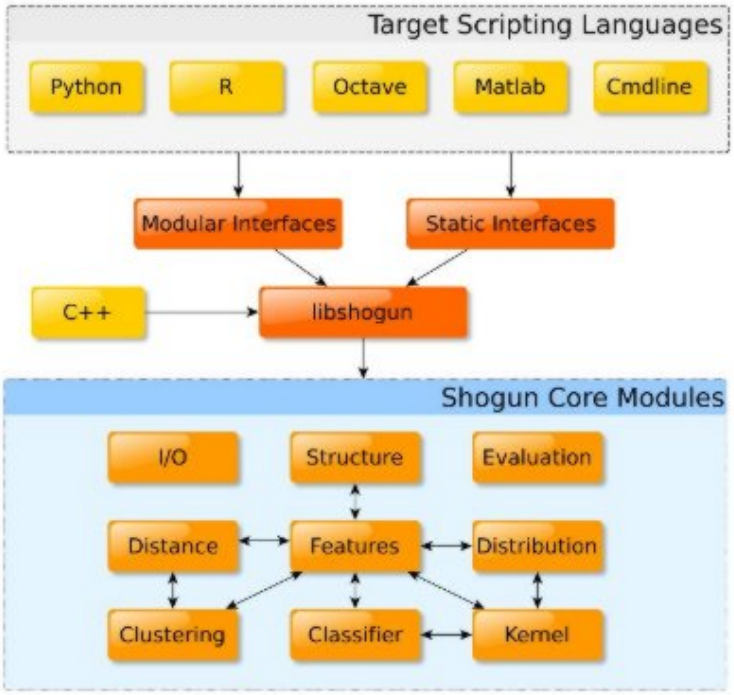
2.8 Shogun

Shogun je odprto kodni sistem napisan v jeziku C++ in ponuja vmesnike za programske pakete Matlab, Octave, Python, R, Java, Ruby in C# in uporablja licenco GPL (*General Public License*) različico 3 in naprej. Ponuja veliko algoritmov in podatkovnih struktur za reševanje problemov s strojnim učenjem. Knjižnica Shogun je obsežna zbirka orodij strojnega učenja s poudarkom na podpori vektorjev (*SVM*). Zagotavlja generični SVM vme-

prog. jezik	porazdelitev kode v procentih
C++	59%
C	15%
Python	10%
Matlab	5%
Other	11%

Tabela 2.1: Uporaba programskih jezikov v implementaciji knjižnice Shogun.

snik na več različnih izvedb SVM-jev, med njimi tudi najsodobnejših kot so knjižnice LibSVM in SVMLight, SVMLin in GPDT. Vsakega od SVM-jev je mogoče kombinirati z različnimi jedri (*kernels*). Primarni fokus Shogun-a je na jedrnih strojih (*kernel machines*) za reševanje problemov z regresijo in klasifikacijo. Ko se je Shogun razvijal so imeli razvijalci v mislih predvsem probleme bioinformatike in to je velikost podatkov, ki so potrebni da se procesirajo podatki, ki imajo lahko tudi do 10 milijonov vzorcev. Na tabeli 5.9 vidimo strukturo sistema Shogun, ki jo sestavljajo različni moduli in vmesniki. Knjižnica Shogun ponuja dva različna programska vmesnika in to sta modularni in statični vmesnik, katera bomo podrobno kasneje opisali v poglavju 4.7.2. Knjižnica nima lastnega grafičnega uporabniškega vmesnika. Na sliki 2.3 pa je prikazana porazdelitev implementacijske programske kode procentualno porazdeljene po različnih programskih jezikih. Domača spletna stran sistema Shogun je na naslovu *http://www.shogun-toolbox.org*. Sistem je bil nazadnje ažuriran dne 1.12.2011 z različico 1.1.0 kar lahko pomeni, da se sistem še naprej razvija. Knjižnica Shogun podpira različice Python 2.5, 2.6 in 2.7.



Slika 2.3: Struktura sistema Shogun.

Poglavje 3

Namestitev

V tem poglavju bomo testirali ter ocenjevali namestitev sistemov za odkrivanj znanj iz podatkov, kateri imajo vmesnik v programskem jeziku Python, ki smo jih že opisali v prejšnjem poglavju. Namestitve omenjenih programskih sistemov so se izvajale na virtualnih računalnikih. Uporabljali smo Oracle VM VirtualBox sistem za poganjanje virtualnih računalnikov (izvirno produkt od podjetja Sun, katerega je Oracle kupil leta 2009). Za uspešno namestitev smo uporabili tri virtualne računalnike, dva Windows XP SP3 operacijska sistema ter eden Linux operacijski sistem. Eden Windows OS bo imel nameščeno različico programa Python 2.6, ki je potreben za knjižnico Elefant, ki ne podpira različice Python večje od 2.6, druga platforma (isto Windows XP OS) pa bo podpiral različico Python 2.7 na katerem so vse ostale knjižnice, ki podpirajo Windows okolje. Na zadnjem in tretjemu virtualnem računalniku pa bo nameščen Linux okolje in različica programa Python 2.7 za knjižnice, ki ne podpirajo Windows okolja in ti knjižnici sta Shogun in Pyml. Naštete programske sisteme za odkrivanj znanj smo ocenili glede podpore operacijskih sistemov ter enostavnosti namestitve v okolje Python. Višja ocena pomeni boljša ocena, nižja pa slabša ocena. Izbrali smo interval od 1 do 5, pet pomeni najboljša možna ocena.

3.1 Elefant

Testna namestitev je potekala v okolju Windows XP. Najnovejša verzija Elefant sistema je 0.4.0 in je iz dne 17.10.2009. Datoteko *elefant-0.4.0.bz2* smo prenesli iz spletnega naslova *http://elefant.forge.nicta.com.au*. Da se Elefant uspešno inštalira so potrebne naslednje pred pogojne namestitve sistemov in knjižic:

- Python 2.6,
- NumPy 1.3 ali kasnejše različice,
- SciPy 0.7.1 ali kasnejše različice,
- Matplotlib 0.99 ali kasnejše različice,
- wxPython 2.8 ali kasnejše različice,
- SWIG 1.3.

Za okolje Windows so potrebni še naslednji pogoji. Datoteka *MSVCP71.dll* mora obstajati v Windows sistemskem imeniku ter sistemski imenik Python-a v katerem se nahaja program *python.exe* naj bo nastavljen v *Path* nastavitvah okoljskih spremenljivk. Sama namestitev je srednje kompleksna ker od uporabnika pričakuje, da pozna detajlno kako se moduli za Python namestijo in nima avtomatične namestitve tipične za Windows okolje. Namestitev poteka preko Python skript, ki se izvajajo po naslednjih korakih:

1. *setup-deps.py*, preveri se ali namesčene vse knjižnice potrebne za delovanje Elefant-a, če ne niso jih avtomatično pobere z interneta,
2. *setup.py build*, s tem ukazom se generira binarna koda,
3. *setup.py test*, testira se *build* vendar je ta korak neobveznen za namestitev),
4. *setup.py install*, sama namestitev programa v Python okolje,
5. *launchelefant.py*, s to skripto se zažene program Elefant, skripta se nahaja v Python Scripts imeniku.

Ker knjižnica podpira samo starejše različice okolja Python in je namestitev ne enostavna smo dali namestitvi knjižnici končno oceno tri.

3.2 Mdp

Testna namestitev je potekala v okolju Windows XP. Najnovejša različica knjižnice MDP za Python je različica 3.2 iz dne 24.10.2011. Datoteko *MDP-3.2.Python2.exe* smo prenesli iz spletnega naslova *http://sourceforge.net/projects/mdp-toolkit/files/mdp-toolkit/3.2/*. Namestitev knjižnice je enostavna saj se knjižnica samodejno namesti v okolje Python. Ni potrebno ročno izvajati skript ali konfigurirati sistemske spremenljivke, da se sistem namesti v okolje Python. Knjižnica Mdp podpira naslednje različice Python-na 2.5, 2.6, 2.7, 3.1, 3.2 in 3.3. Vseeno je potrebna predpogojna namestitev knjižnice *NumPy* različice 1.1 ali kasnejših različic za uspešno delovanje knjižnice Mdp. Po namestitvi se lahko izvede test le te z naslednjimi ukazi v Python okolju:

```
import mdp
mdp.test()
import bimdp
bimdp.test()
```

Ker je namestitev zelo enostavna in avtomatična in ker dobro podpira različne različice okolja Python smo namestitev knjižnice ocenili visoko. Ker pa knjižnica ne poskrbi še za namestitev soodvisnih knjižnic smo namestitvi knjižnice dali oceno 4.

3.3 OpenCVLibrary

Testna namestitev je potekala v okolju Windows XP. Najnovejša različica knjižnice OpenCV je 2.4 iz dne 7.5.2012. Vmesniki knjižnice OpenCV podpirajo Python različico 2.7 in ne podpirajo 64 bitnega okolja Python. Po samodejni namestitvi knjižnice OpenCV v poljuben imenik, je potrebno to knjižnico integrirati v okolje Python in upoštevati pred pogojne zahteve za uspešno delovanja knjižnice. Pred pogojni namestitvi knjižnic potrebnih za uspešno delovanje knjižnice OpenCV sta knjižnici *NumPy* 1.6 in *SciPy*.

V namestitvenem imeniku knjižnice OpenCV je potrebno uvoziti v oko-

lje Python. To naredimo tako, da kopiramo datoteko, ki se nahaja v imeniku *[OpenCV imenik]/build/Python/2.x/cv2.pyd* v imenik okolja Python *[Python]/Lib/site-packages/*. Problem namestitve knjižnice OpenCV v okolje Python je problematično zato, ker ne obstajajo navodila kako to izvesti in ne pomaga tudi dejstvo, da se vmesniki za Python spreminjajo z novimi različicami in je namestitev knjižnice OpenCV v okolje Python različno za vsako večjo spremembo verzije knjižnice OpenCV. Ker ima knjižnica OpenCV omejeno podporo različic okolja Python, podpira samo različice 2.6 in 2.7 in je sama namestitev ne avtomatična v okolje Python in brez navodil kako namestiti to knjižnico smo namestitvi dali končno oceno 3.

3.4 Orange

Testna namestitev je potekala v Windows XP okolju. Namestitveno datoteko *orange-win-w-python-snapshot-hg-2012-03-19-py2.7.exe* smo prenesli iz spletnega naslova *http://orange.biolab.si/download/orange-win-w-python-snapshot-hg-2012-03-19-py2.7.exe*, ki je iz dne 19.3.2012. Orange ima dva tipa namestitev in ti dve sta polna (angl. *snapshot*) in čista (angl. *pure*) namestitev za različici okolja Python 2.6 in 2.7. Različica tipa *snapshot* namesti vse potrebne knjižnice, ki jih Orange potrebuje, da se aplikacija uspešno namesti. Obstaja pa tudi čista različica namestitve, ki je brez knjižnic, ki so potrebne, da se Orange zažene in pričakuje, da so te knjižnice že namesčene. Namestitev tipa *snapshot* je zelo enostavna ne zahteva ročne namestitve dodatnih knjižnic, ki jih Orange potrebuje za samo delovanje sistema vključno z okoljem Python. Avtomatična namestitev doda tudi bližnjico do Orange programa v ozadje uporabnika. Ker ima knjižnica enostavno namestitev knjižnice Orange in soodvisnih knjižnic smo dali namestitvi končno oceno 5.

3.5 PyBrain

Testna namestitev knjižnice je potekala v okolju Windows XP. Najnovejša različica knjižnice PyBrain za Python 2.7 je datoteka *pybrain-pybrain-0.3.1-1-gcee417a.zip* iz dne 29.11.2011, ki smo jo prenesli iz spletnega naslova *https://github.com/pybrain/pybrain/downloads*. Knjižnica PyBrain zahteva najmanj različico Python 2.5 ali kasnejše različice ter knjižnico SciPy 0.6 ali kasnejše različice. Priporoča pa se tudi Matplotlib 0.98 ali kasnejše različice za izrisovanje grafov. Namestitveno datoteko razpakiramo v poljubni imenik nato z naslova *http://peak.telecommunity.com/dist/ez_setup.py* naložimo datoteko `ez_setup.py`, ki jo kopiramo v imenik v katerem smo razpakirali PyBrain datoteke. Nato skripto `ez_setup.py` poženemo v okolju Python. Za namestitev knjižnic Scipy in Matplotlib poženemo naslednje skripte `easy_install scipy.py` ter `easy_install matplotlib.py`. Ponavadi se namestitev Scipy ne zgodi pravilno, manjkajo kakšne dodatne knjižnice kot so (*ATLAS, LAPACK, MKL, ACML*) zato je lažje, da se namesti Scipy preko distribucije *Enthought*.

Nato v imeniku, v katerem smo razpakirali namestitvene datoteke za knjižnico PyBrain, poženemo skripto za namestitev knjižnic v okolje Python z naslednjim ukazom

```
# setup.py install
```

PyBrain namestitev lahko testiramo tako, da se požene naslednjo skripto v okolju Python:

```
import pybrain
```

ali pa se požene naslednjo skripto, ki se nahaja v pybrain/tests imeniku.

```
# python runtests.py
```

Ker je sama nastavitev dosti kompleksna, vendar pa knjižnica podpira širok spekter različic okolij Python, smo dali namestitvi končno oceno 3.

3.6 Pyml

Testna namestitev je potekala v okolju Linux 32 bitne distribucije Ubuntu 11.10. Knjižnice Pyml ne podpira Windows okolja, ker ima določeno kodo v programskem jeziku C++, ki ni kompatibilna za okolje Windows. Najnovejša različica je datoteka *PyML-0.7.9.tar.gz* iz dne 7.10.2011 s spletnega naslova *http://sourceforge.net/projects/pyml/files/*. Pyml zahteva najmanj različico Python 2.5 ali kasnejše različice in da se knjižnica Pyml uspešno namesti v okolje Python sta potrebni knjižici *NumPy* 1.0 ali višje različice in knjižnica Matplotlib, ki je potrebna, če se zahteva da se bodo izrisovali tudi grafi. Namestitev se požene z naslednjimi skriptami v imeniku v katerem smo razpakirali namestitvene datoteke Pyml:

```
# python setup.py build
# python setup.py install
```

Namestitev knjižnice Pyml lahko testiramo z naslednjo skripto:

```
from PyML import *
from PyML.demo import pyml_test
pyml_test.test('svm')
```

Ker knjižnica ne podpira vseh platfrom in ker namestitev ni enostavna in avtomatična je ocena, ki smo ji podali, slabša. Namestitvi smo zaradi tega dali končno oceno 2.

3.7 Shogun

Testna namestitev je potekala v okolju Linux 32 bitne distribucije Ubuntu 11.10. Knjižnica podpira tudi MacOSX operacijski sistem preko posebnega porta vendar isto kot knjižnica Pyml ne podpira Windows okolja, čeprav avtorji oglašujejo, da je možno z dosti napora knjižnico namestiti v sistem Cygwin vendar ni nikjer objavljenih navodil kako to uspešno narediti. Za namestitev knjižnice Shogun rabimo samo osnovni del ter podporo knjižnicam za Python. Knjižnica Shogun poleg Python-a podpira

tudi ostale programske jezike C#, Java, Lua, Octave, Ruby in Matlab. Shogun podpira Python 2.5 in 2.6 ter namestitev Python-a mora biti razvijalska različica (*python-dev*) in jo namestimo s skripto :

`# apt-get install python-dev`. Za namestitev Shogun knjižnic za Python so potrebne naslednje knjižnice, ki morajo biti že nameščene v Python-u:

- Numpy 1.x (python-numpy in python-numpy-ext)
- Matplotlib (za izrisovanje grafov)
- Swig (za objektni Python)

Namestitev kličemo s skriptami:

```
# ./configure--interfaces=python_modular
# make
# sudo make install
```

Statični modul knjižnic se samodejno namesti po prevzetih nastavitvah in ni potrebno posebno določiti, da se namesti kot dodatni vmesnik. Preverimo ali je bila namestitev uspešna z naslednjo skripto:

```
# ../examples/documented/python_modular/graphical/svm.py
```

Ker knjižnica ne podpira vseh platfrom in ker namestitev ni enostavna in avtomatična je ocena, ki smo ji podali slabša. Namestitvi smo zaradi tega dali končno oceno 2.

Poglavje 4

Dokumentacija, programski vmesnik in algoritmi

V tem poglavju bomo pregledali glavne funkcionalnosti posameznih programskih rešitev, ki jih ta diplomska naloga obravnava. Pod te funkcionalnosti bomo pregledali kakšno dokumentacijo ima knjižnica in preiskali odgovore na vprašanje, ali ima dokumentacija naslednje elemente:

- dokumentacija arhitekture,
- tehnična dokumentacija, ki opisuje strukturo programskega vmesnika,
- uporabniška dokumentacija kot so vodniki in ostala dokumentacija.

Pri arhitekturi smo se osredotočili na programski vmesnik in z odgovori na vprašanja, kot so, kako se dostopa in manipulira s podatki, kako se kliče in uporablja algoritme, itd.. Na koncu pa bomo pregledali, katere algoritme in tehnike podpirajo naštete programske rešitve. Posebej moramo poudariti, da smo analizirali tiste aspekte knjižnic, ki pridejo s privzeto namestitvijo in ne dodatnih modulov ali implementacij, ki pridejo z dodatnimi namestitvami drugih knjižnic, ki niso del osnovnega paketa knjižnic in niso obvezne za delovanje same knjižnice, ki jih v tej diplomski nalogi preučujemo.

4.1 Elefant

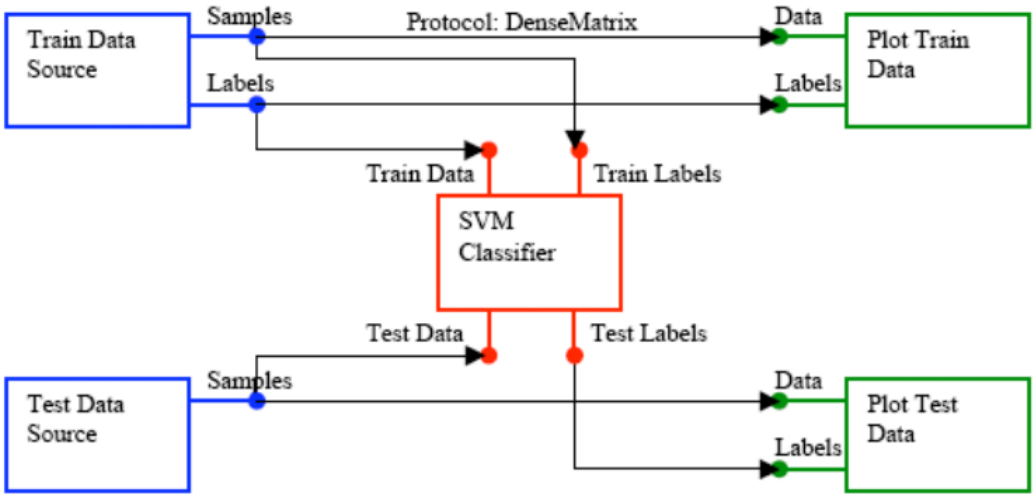
4.1.1 Dokumentacija

Elefant ima na svoji spletni strani naslednjo dokumentacijo:

- uporabniško dokumentacijo (*User manual*) v pdf datoteki,
- dokumentacijo programskega vmesnika v obliki html,
- dokumentacija arhitekture in zasnove (*Component manual*) v pdf datoteki.

Poleg naštetih dokumentov obstaja na spletni strani primer vodnika kako se v sistem dodaja novo funkcionalnost preko integracije modulov v sistem Elefant. Na domači spletni strani je objavljen tudi video posnetek učne ure pod naslovom *Poletna šola - Strojno učenje 2008 z orodjem Elefant*. Vsebina uporabniške dokumentacije nima dodelanega vrstnega reda, je samo skupek navodil za določene algoritme, ki med seboj niso povezani. Primeri, ki so napisani, ne ustrezajo več zadnji različici knjižnice in se ne interpretirajo pravilno, ker se je programski vmesnik spremenil. Boljšo strukturo in preglednost prinaša dokumentacija arhitekture in zasnove v kateri se vsebina začne z opisom osnovnih struktur ter pravil povezovanja le teh. Vsebina dokumenta se nadaljuje z natančnim opisom vhodnih in izhodnih lastnostih ter metod komponent. Dokumentacija programskega vmesnika je zgrajena s programskim orodjem doxygen (*http://www.stack.nl/ dimitri/doxygen/index.html*). Rezultat le tega je skupek html-jev ter povezanih opisov struktur v programski kodi, ki jih je avtor kode dodal kot komentarje v izvorni kodi. Ta način dokumentacije omogoča tako imenovani *drill down* pregled, ki omogoča hiter pregled nad uporabniškim vmesnikom in njegovih podatkovnih struktur.

Ker je dokumentacija, ki je objavljena nedosledna in ne odraža zadnjega stanja programskega vmesnika in ni v pomoč pri pisanju primerov za knjižnico, čeprav je dokumentacija raznolika, ne odtehta te slabosti. Zato smo dali dokumentaciji končno oceno 2.



Slika 4.1: Primer klasifikacije v sistemu Elefant.

4.1.2 Programski vmesnik

Ogrodje (*framework*) Elefant-a je zgrajeno na komponenti zasnovi. Komponento ogrodje omogoča modularno nad gradljivo in ponovno uporabo komponent v sistemu, ki podpira vstavi in poženi (*plug and play*). Elefant ponuja lastni grafični uporabniški vmesnik za vizualno gradnjo primerov. Grafični uporabniški vmesnik je neodvisen od implementacije ogrodja in uporabniški vmesnik sam povpraša komponente, da zapolnijo grafične komponente kot so meniji, ikone in njihovih vhodno izhodnih vrat (*ports*) in protokolov. Ta vhodna in izhodna vrata se uporabljajo za komunikacijo med komponentami.

Primer klasifikacije, ki sloni na ogrodju komponent, je prikazan na sliki 4.1. V njem obstaja komponenta učni podatkovni izvor (*Train Data Source*), ki ima dva vhoda po imenu vzorci (*Samples*) in oznake (*Labels*). Ta dva vhoda sta povezana s komponento *SVM klasifikator* preko protokola *DenseMatrix*. Prednost takega pristopa je, da lahko učni podatkovni izvor uporablja kateri koli podatkovni datotečni format, kot sta naprimer

datotečni formata CSV in Matlab. Poleg tega ima komponenta določen standardni vmesnik v obliki vrat in zna komunicirati s protokolom *DenseMatrix*. Podobno zna komponenta *SVM klasifikator* komunicirati z istim protokolom in ni pomembno kakšen podatkovni format uporablja znotraj komponenta *učni podatkovni izvor*. Komponente v sistemu Elefant morajo imeti vrata, protokole, lastnosti in metode. Vrata se obnašajo kot vmesniki, ki imajo določen namen in funkcionalnost in izpostavljajo funkcionalnost ali storitev komponente preko striktno določenih abstraktnih vmesnikov, ki skrivajo detajle notranje implementacije. Vendar vmesniki niso sami po sebi dovolj, da bi komunicirali z drugimi komponentami. »Govoriti morajo isti jeziki« in to počnejo preko protokolov. Vse komponente morajo določiti vrata in tipe protokolov, ki jih podpirajo posamična vrata. Protokol opisuje, kako komponente komunicirajo z drugimi komponentami, ko so povezane preko določenih vrat. Sistem je zelo podoben mrežnemu komuniciranju, kjer na primer email odjemalec uporablja vrata 110 sistema z namenom, da bi prejemal pošto na oddaljenemu strežniku in s tem oba »govorita«, komunicirata preko POP3 protokola z namenom, da bi se lahko sporazumela.

Elefant sistem podpira naslednje protokole:

- *DenseMatrix*, ta protokol pričakuje podatke v obliki *numpy.ndarray*,
- *DenseVector*, pričakuje podatke v obliki *numpy.ndarray*,
- *SparseMatrix*, pričakuje podatke v obliki *Scipy* matrix obliki,
- *ObjectArray*, ta je v obliki navadne Python liste,
- *OnlineDenseVector*, ta protokol je isti kot *DenseMatrix* s tem, da dodatno zagotavlja funkcionalnost iteratorja. Namesto da se vsi podatki naložijo v pomnilnik omogoča, da se podatki naložijo delno in postopoma na zahtevo,
- *OnlineSparseVector*, isto kot *SparseMatrix* le da se doda funkcionalnost iteratorja,
- *OnlineObjectArray*, isto kot *ObjectArray* le da se doda funkcionalnost iteratorja.

Lastnosti so atributi ali parametri komponent, s katerimi vplivajo na delovanje komponent med izvajanjem (*run time*). Elefant pri branju podatkov podpira naslednje podatkovne formate:

- CSV (*Comma Separated Values*), podpira tudi oddaljen dostop preko spletnega naslova,
- Matlab, podpira tudi oddaljen dostop preko spletnega naslova,
- Random (generator naključnih vrednosti),
- LibSvm datotečni format, podpira tudi oddaljen dostop preko spletnega naslova.

Elefant pri pisanju podatkov zna zapisovati samo v podatkovni format CSV. Elefant oglašuje, da podpira procesiranje velikih količin preko tipa komponent *online* in podatkovnih tipov, vendar knjižnica ne ponuja nobenega primera in izhodi *online* komponent niso uporabni, ker ne omogočajo klasifikacije podatkov, kot to omogočajo komponente, ki niso tipa *online*. Knjižnica tudi ne podpira večprocesne arhitekture za paralelno izvajanje procesiranja algoritmov.

Na sliki 4.2 vidimo primer uporabe grafičnega uporabniškega vmesnika Elefant, ki omogoča tudi nabor, da pretvori narejen sistem v grafičnem uporabniškem vmesniku Elefant-a v izvorno kodo Python-a. Primer na sliki smo pretvorili v kodo Python v naslednjih korakih bomo opisali, kaj so določeni objekti.

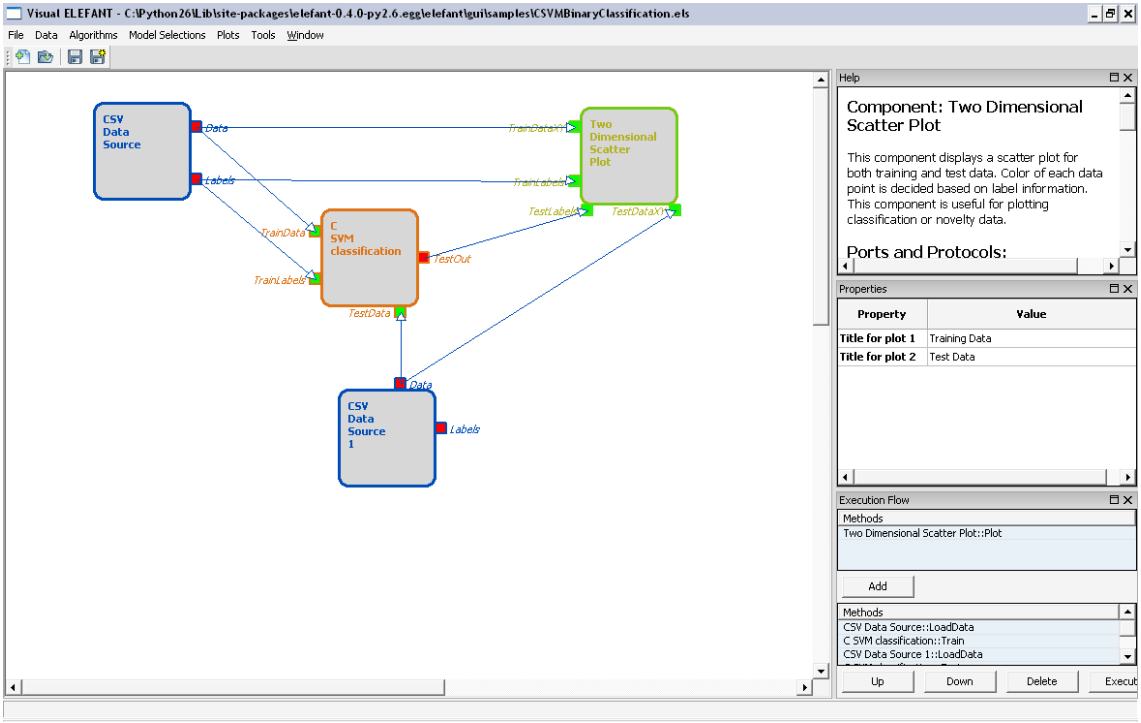
Nastavitev vhodnih podatkov (testnih in učnih) preko CSV datoteke in za nalaganje teh podatkov se uporablja razred *CSVData*.

```
CSV_Data_Source_1 = CSVData()
CSV_Data_Source_1.delimiter = ','
CSV_Data_Source_1.fileName='train_data.csv'

CSV_Data_Source = CSVData()
CSV_Data_Source.delimiter = ','
CSV_Data_Source.fileName = 'test_data.csv'
```

Sledi inicializacija algoritma in njegovega jedra.

```
C_SVM_classification = CSVMClassification()
```



Slika 4.2: Uporabniški grafični vmesnik in zgrajen primer.

```
C_SVM_classification.C = 1.0
kernel = CGaussKernel()
kernel.blocksize = 128
kernel.scale = 1.0
C_SVM_classification.kernel = kernel
```

Inicializaciji grafa sledi spodnja koda.

```
Two_Dimensional_Scatter_Plot = CPlotScatter()
Two_Dimensional_Scatter_Plot.plot1Title = 'Training Data'
Two_Dimensional_Scatter_Plot.plot2Title = 'Test Data'
```

Povezovanje komponent med seboj opravijo sledeči ukazi.

```
Connect(CSV_Data_Source, "Data", Two_Dimensional_Scatter_Plot,\
        "TrainDataXY", "DenseMatrix")
Connect(CSV_Data_Source, "Labels", Two_Dimensional_Scatter_Plot,\
        "TrainLabels", "DenseVector")
Connect(CSV_Data_Source_1, "Data", Two_Dimensional_Scatter_Plot,\
        "TestDataXY", "DenseMatrix")
Connect(CSV_Data_Source_1, "Data", C_SVM_classification,\
        "TestData", "DenseMatrix")
Connect(CSV_Data_Source, "Data", C_SVM_classification,\
        "TrainData", "DenseMatrix")
Connect(CSV_Data_Source, "Labels", C_SVM_classification,\
        "TrainLabels", "DenseVector")
Connect(C_SVM_classification, "TestOut",\
        Two_Dimensional_Scatter_Plot, "TestLabels", "DenseVector")
```

Izvedba procesnega toka je zajeta spodaj.

```
BeginWx()

CSV_Data_Source.LoadData()
C_SVM_classification.Train()
CSV_Data_Source_1.LoadData()
C_SVM_classification.Test()
Two_Dimensional_Scatter_Plot.Plot()

EndWx()
```

4.1.3 Algoritmi

Elefant pri implementaciji algoritmov daje poudarek na metode podpornih vektorjev oz. SVM (*Support Vector Machines)*, ki podpirajo naslednje jedrne funkcije: *dot product, linear, polynomial in tahn*. Ti algoritmi podpirajo

klasifikacijo, regresijo in tudi iskanje osamelcev z algoritmom *Novelty Detection*. Knjižnica podpira tudi preproceriranje podatkov kot je zmanjšanje dimenzij in določevanje relevantnih podatkov (angl. *Feature selection*). Poleg algoritmov SVM knjižnica Elefant podpira tudi Gausovo regresijo in klasifikacijo ter razvrščanje v skupine preko algoritma K-Means in Diffusion Map. Knjižnica v taki obliki, kot je sedaj, ne more uspešno pokrivati operacij, ki so potrebne, da bi lahko popolnoma reševali probleme iskanju znanj iz podatkov z uporabo metodologije CRISP predvsem zaradi nepodpore operacij *prečnega preverjanja* in drugih algoritmov.

4.2 Mdp

4.2.1 Dokumentacija

Mdp (*Modular toolkit for Data Processing*) ima na svoji spletni strani naslednjo dokumentacijo:

- uporabniško dokumentacijo v obliki html in v zapisu datoteke tipa pdf,
- dokumentacijo programskega vmesnika v obliki html, ki je generirana s programskim orodjem Epydoc (*http://epydoc.sourceforge.net/*),
- dokumentacija za razvojnike knjižnice Mdp,
- dokumentirani primeri uporabe knjižnice Mdp,
- dokumentacija dodatnih programskih modulov za Mdp.

Od obstoječih dokumentacij knjižnice Mdp najbolj pogrešamo dokumentacijo arhitekture in zasnove. Pregled same arhitekture je potrebno izluščiti iz primerov uporabe in dokumentacije uporabniškega vodnika. Ker ima knjižnica širok spekter različne dokumentacije vendar s pomanjkljivo uporabniško dokumentacijo in manjkajočem dokumentu arhitekture in zasnove smo dokumentaciji dali končno oceno 3.

4.2.2 Programski vmesnik

Vmesnik sloni na zbirki enot imenovanih *unit*, ki znajo obdelovati podatke. To vključuje algoritme za nadzorovano učenje, nenadzorovano učenje, principalne in neodvisne analitične komponente in klasifikacije. Te enote lahko povežemo v podatkovne tokovne obdelave z namenom, da vzpostavimo cevovode in bolj kompleksne mrežne arhitekture tipe podaj naprej (*feed-forward*). Glede na vhodne podatke knjižnica MPD poskrbi, da se učenje in izvajanje vseh vozlišč (*nodes*) v mreži zgodi v pravilnem vrstnem redu in posreduje vmesne podatke med vozlišči. Knjižnica MDP zna poskrbeti za učinkovitost pri porabi spomina in hitrosti izvajanja operacij. Da se zmanjša zasedba pomnilnika (*memory footprint*) zna knjižnica obdelovati velike količine podatkov. Za zelo velike podatkovne zbirke je možno MDP konfiguritati, da uporablja enojno natančno plavajočo vejico (*single precision point*) namesto dvojne. Knjižnica MDP podpira tudi paralelno izvajanje s podpaketom *parallel*, ki omogoča paralelno implementacijo osnovnih vozlišč in tokov. *Node* je osnovni gradnik MDP aplikacije, ki lahko predstavlja podatkovni procesni element kot na primer učni algoritem, podatkovni filter ali grafični izpis. Vsak *node* ima lahko eno ali več učnih faz.
Primer klicanja metode učenja iz podatkov je:

```
train_data = np.random.random((100, 25)
node = mdp.nodes.PCANode()
node.train(train_data)
node.stop_training()
```

Učenje se neha, ko se pokliče metodo *stop_training*, *execute* ali *inverse*. Ko se učenje vozlišča konča, se lahko pokliče metoda *execute*, ki izvede procesiranje podatkov. V primeru klasifikatorja to pomeni testiranje ali klasifikacijo vhodnih podatkov.

```
x = np.random.random((100, 25))
y_pca = node.execute(x)
```

S konstruktorjem objekta se simulira klic na metodo *execute*.

```
y_pca = node(x)
```

Če je operacija, ki jo je vozlišče izračunalo, invertibilna, potem lahko to vozlišče izvede inverzno transformacijo.

```
print node.is_invertible()
True
x = node.inverse(y_pca)
```

Flow ali tok je sekvenca vozlišč *(nodes)*, ki izvaja učenje po sekvenčnem vrstnem redu in tako zagotavlja izvajanje bolj kompleksnih algoritmov. Tokovi omogočajo avtomatično učenje, izvajanje ali obratnega izvajanja, če je bilo določeno, celotne sekvence. Primer inicializacije toka podaja spodnja vrstica:

```
flow = mdp.Flow([mdp.nodes.PCANode(), mdp.nodes.CuBICANode()])
```

Tokovi se lahko zgradijo tudi s seštevanjem vozlišč:

```
flow = mdp.nodes.PCANode() + mdp.nodes.CuBICANode()
```

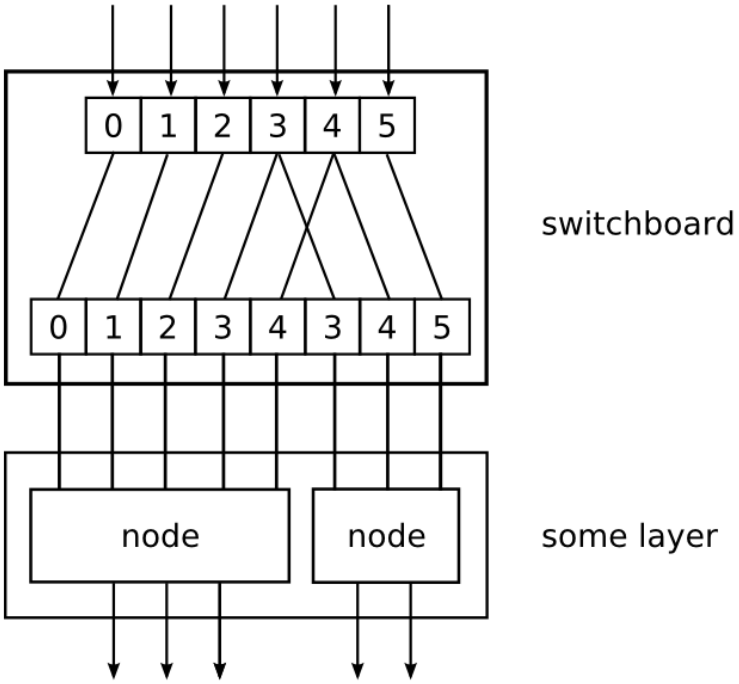
Vozlišča v toku se učijo s klicem *train* in izvede projeciranje podatkov z metodo *execute*:

```
flow.train(x)
out = flow.execute(test)
```

V tem primeru tok simulira kodo kot, če bi jo napisali na roke s posamičnimi vozlišči. Prvo definirano vozlišče, ki je določeno v klicanem toku, vzame vhodne podatke in požene *train* metodo in s tem učenje na teh in po končanem učenju izvede *execute* metodo in njegove izhodne podatke uporabi kot vhodne podatke naslednjega vozlišča v klicu *execute* in tako naprej.

```
pca = mdp.nodes.PCANode(output_dim=5)
pca.train(x)
out1 = pca(x)

ica = mdp.nodes.CuBICANode()
ica.train(out1)
out2 = ica(out1)
```



Slika 4.3: Primer interakcije objekta *switchboard* in *layer*.

Paket *hinet* vsebuje tri osnovne gradnike. Vsi so derivati od osnovnega objekta *Node*. Ti gradniki so *Layer*, *FlowNode* in *Switchboard*. *Layer* ali plast je tip vozlišča, ki se obnaša kot horizontalna verzija toka *(Flow)*. Obnaša se kot ovojnica za množico vozlišč, ki se učijo in izvajajo paralelno.

Primer gradnje *Layer* vozlišča podajamo spodaj:

```
node1 = mdp.nodes.PCANode(input_dim=100, output_dim=10)
node2 = mdp.nodes.SFANode(input_dim=100, output_dim=20)
layer = mdp.hinet.Layer([node1, node2])
```

V zadnjem primeru se prva polovica 200-tih dimenzij avtomatično določi vozlišču *node1* in druga polovica vozlišču *node2*. *Layer* vozlišče se obnaša tako kot vsa ostala vozlišča. *FlowNode* omogoča pretvarjanja tokov v vozlišča in se uporabi v *Layer* vozlišču, kot to prikazuje spodnji primer.

```
node1_1 = mdp.nodes.PCANode(input_dim=100, output_dim=50)
node1_2 = mdp.nodes.SFANode(input_dim=50, output_dim=10)
node1_flow = mdp.Flow([node1_1, node1_2])
```

```
node1 = mdp.hinet.FlowNode(node1_flow)
layer = mdp.hinet.Layer([node1, node2])
```

Switchboard vozlišče omogoča, da se vhodni podatki vežejo na različna vozlišča v plasti.

```
switchboard=mdp.hinet.Switchboard(input_dim=6,\
connections=[0, 1, 2, 3, 4, 3, 4, 5])
x = mdp.numx.array([[2, 4, 6, 8, 10, 12]])
switchboard.execute(x)
```

Po switchboard-u lahko potem sledi plast, ki porazdeli določene vhode na različna določena vozlišča, kot je prikazano na sliki 4.3.

Knjižnica MDP ima tudi paket *parallel*, ki omogoča paralelno izvajanje učenja in izvajanje procesiranja podatkov na tokovih. V osnovi knjižnica MDP podpira paralelno izvajanje z metodami za kopiranje instanc objektov. *ParallelFlow* je tok z implementacijo, ki razdeli učenje ali izvajanje nad podatki v opravila, ki se izvajajo paralelno. MDP pri branju podatkov podpira samo podatkovne strukture *NumPy* in nima svoje podpore za branje datotek CSV, podatkovnih baz in drugih tipov podatkovnih datotek. Vsi vhodni podatki so v tipa *ndarray* razreda, ki je del *NumPy* knjižnice.

4.2.3 Algoritmi

Knjižnica MDP implementira vrsto klasifikatorjev kot so k-Means, Gaussian, Nearest Mean, k-NN, Signum, Perceptron, Simple Markov ter algoritme iscrete Hopfield. Edina regresijska metoda, ki jo ima knjižnica implementirano je linearna regresija. Pri procesiranju podatkov knjižnica naslednje skupine algoritmov: PCA, ICA ter Feature Analysis. Poleg lastnih algoritmov, ki jih knjižnica MDP sama implementira ima tudi implementirane ovojnice okoli ostalih knjižnic, ki implementirajo strojno učenje kot so Shogun, LibSVM in Scikits. Knjižnica v taki obliki kot je sedaj ne more uspešno pokrivati operacij, ki so potrebne, da bi lahko popolnoma reševali probleme iskanju znanj iz podatkov z uporabo metodologije CRISP predvsem zaradi ne podpore operacij *prečnega preverjanja* in ne podpori branju

različnih vhodnih podatkov.

4.3 OpenCV

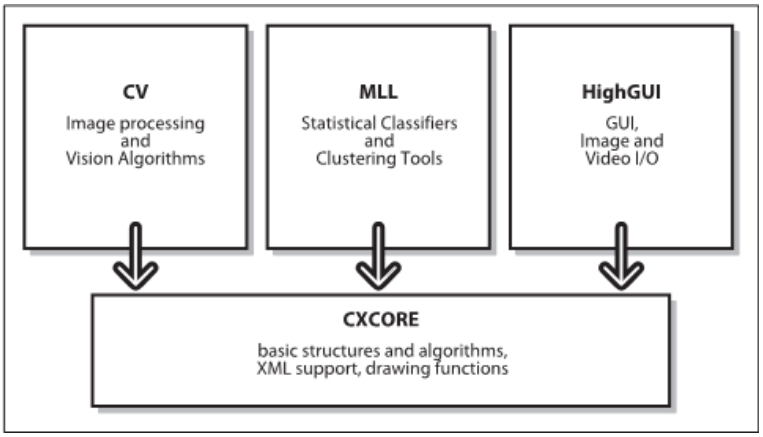
4.3.1 Dokumentacija

Knjižnica OpenCV vsebuje zelo obširno zbirko algoritmov in podpira širok spekter programskih jezikov in zaradi tega obstaja obsežna zbirka dokumentacije na internetu. Poleg dokumentacije na internetnih straneh obstajata za knjižnico OpenCV tudi dve knjigi *Learning OpenCV* in *OpenCV 2 Cookbook*, ki ju je možno kupiti v prosti prodaji. OpenCV ima na svoji spletni strani naslednje dokumentacijo, ki se ob namestitvi knjižnice posnamejo tudi različice v pdf formatu:

- uporabniško dokumentacijo v obliki html,
- dokumentacijo programskega vmesnika v obliki html, ki je generirana s programskim orodjem Sphinx (*http://sphinx.pocoo.org*),
- dokumentirani primeri uporabe OpenCV knjižnice.

Obstaja tudi dokumentacija za programski vmesnik Python, ki se nahaja na spletni strani *http://opencv.willowgarage.com/documentation/python/index.html* vendar je ta del dokumentacije nepopoln, posebej na področju strojnega učenja. Ker je primarni cilj knjižnice podpora aplikacijam za delo z računalniškim vidom je poudarek dokumentacije tako kot same knjižnice za podporo slik, videa in večpredstavnosti ter algoritmov in podpore za računalniški vid. Poleg dokumentacije ima knjižnica OpenCV tudi na spletni strani *sourceforge.net* svoj forum za izmenjavo informacij med razvojniki sistema in uporabniki.

Ker ima knjižnica obsežno dokumentacijo in širok spekter dokumentacij ampak s pomanjkljivo uporabniško dokumentacijo za programski vmesnik Python smo dokumentaciji dali končno oceno 4.



Slika 4.4: Struktura modulov knjižnice OpenCV.

4.3.2 Programski vmesnik

Programski vmesnik v knjižnici OpenCV je zelo širok in podpira širok spekter različnih informacijskih storitev posebej na področju računalniškega vida s tem pa tudi ponuja široko množico podpore večpredstavnosti, delo s slikami, zvokom in videom. V tej poglavju se bomo osredotočili na specifični del knjižnice OpenCV in to je modul, ki se imenuje ML (*Machine Learning*). Šele najnovejša verzija 2.4, ki je trenutno v beta stanju podpira ovojnico okoli modula ML za podporo programskega jezika Python. Pred tem je knjižnica OpenCV podpirala Python vmesnik samo klice na CV modul metod strojnega učenja, ki vsebuje algoritma K-means in Mahalanobis in ni vključevala ovojnice za ML modul. Na sliki 4.4 je prikazana modulna struktura knjižnice.

Večina algoritmov izhaja iz osnovnega razreda *CvStatModel*, ki je definiran v ML modulu in ima definirane naslednje metode, ki jih morajo implementacije razreda implementirati:

- *save*(ime datoteke, ime) shrani naučeni model v format XML ali YMAL,
- *load*(ime datoteke, ime) naloži naučeni model iz formata XML ali YMAL,
- *clear*() zbriše interne podatke, ki so bili shranjeni v objektu,

- *train*(vhodni podatki, zastave, odgovori, ...) klic metode za učenje, vhodni podatki se razlikujejo glede na tip algoritmov,
- *predict*(CvMat sample, ...) klic metode za napoved vhodnih podatkov na že naučenih podatkih.

Vsi algoritmi za učenje imajo za vhodne podatke razred tipa *CvMat*, ki se obnaša kot matrika. Ta matrika vsebuje množico podatkov tipa 32 bitne plavajoče vejice (*32-bit float*) v obliki dvodimenzionalne matrike števil. Če podatki niso številčne vrednosti ampak črkovne vrednosti potem je potrebno pretvoriti črkovne podatke v številčne in narediti pretvorbo, ko se podatki vračajo. Implementacija *predict* metode je prilagojena varnemu nitenju (angl. *safe threading*), kar omogoča nemoteno paralelno izvajanje klica. Z različico 2.4 je knjižnica OpenCV dobila podporo za branje CSV datotek, vendar še ni napisanega vmesnika za Python. Za vhodne izhodne parametre se še vedno uporabljajo podatkovne strukture *NumPy*. OpenCV ne podpira *online* procesiranja podatkov kar pomeni, da se morajo vsi podatki naložiti v pomnilnik, da se jih lahko procesira. In tudi ne podpira paralelizem procesiranja podatkov.

4.3.3 Algoritmi

Algoritmi katere knjižnica OpenCV implementira, se nahajajo v treh različnih modulih. Večina algoritmov je v modulu *ML*, ki vsebuje naivni Bayesov klasifikator, k-NN, SVM, odločitvena drevesa, boosting, naključna drevesa in Expectation-Maximization algoritem. V modulu *CVCORE* sta implementirana naslednja algoritma: Mahalanobis in K-Means. V modulu *CV* pa je implementiran klasifikator Haar. Knjižnica v taki obliki, kot je sedaj, ne more uspešno pokrivati operacij, ki so potrebne, da bi lahko popolnoma reševali probleme iskanju znanj iz podatkov z uporabo metodologije CRISP predvsem zaradi manjkajočih algoritmov in operacij pri procesiranju podatkov pred in po iskanju znanj iz podatkov.

4.4 Orange

4.4.1 Dokumentacija

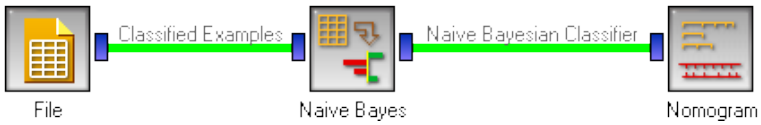
Orange ima na svoji spletni strani *http://orange.biolab.si/documentation* naslednje dokumentacijo:

- katalog in dokumentacija vseh gradnikov (*widgets*) s primeri uporabe,
- dokumentacija za začetnike v obliki vodnika,
- dokumentacija programskega vmesnika, ob namestitvi se posnamejo ta navodila lokalno v obliki html,
- navodila za razvojnike gradnikov,
- navodila za razvoj sistema Orange v wiki formatu.

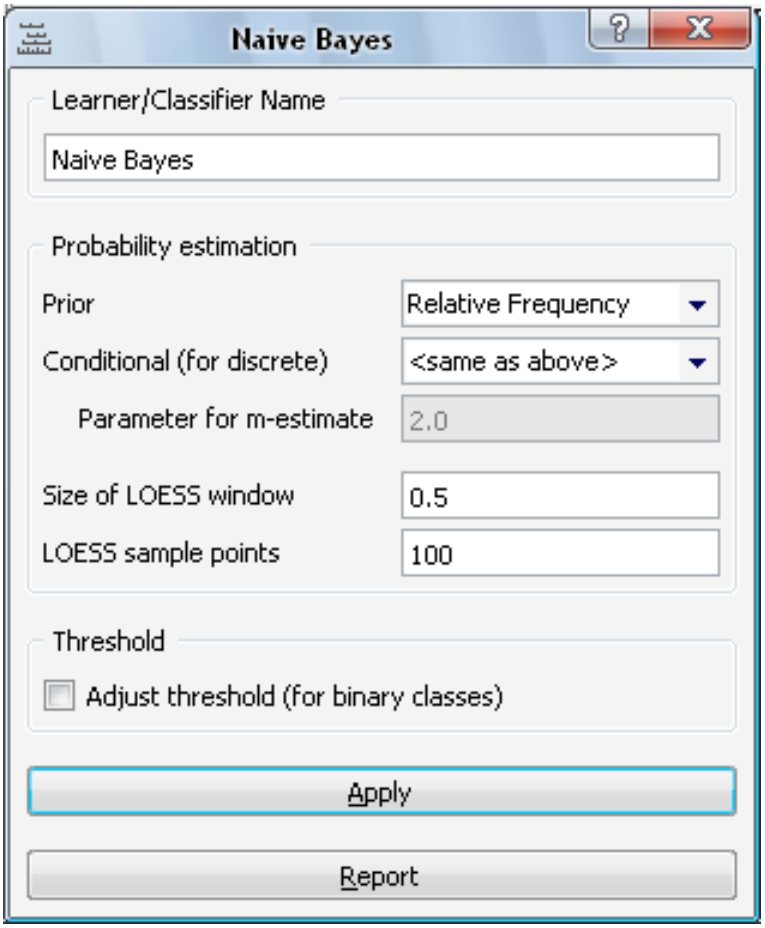
Navodila so zelo obsežna in imajo zelo dobro in strukturiran vodnik za začetnike, ki natančno in postopoma navede in opiše potrebne in možne korake, ko se lotimo uporabljati Orange v programskem okolju Python. Poleg dokumentacije ima Orange tudi na svoji spletni strani svoj forum za izmenjavo informacij med razvojniki sistema in uporabniki. Ker ima knjižnica širok spekter dokumentacij ter dobra navodila za uporabo Orange sistema v okolju Python smo dokumentaciji dali končno oceno 5.

4.4.2 Programski vmesnik

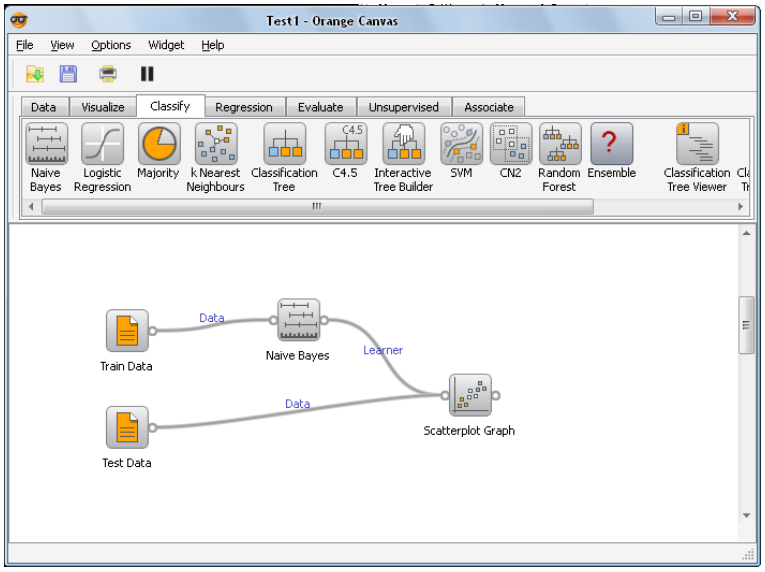
Orange omogoča dva načina uporabe programskega vmesnika. Prvi in najbolj enostaven način je uporaba Orange sistema preko grafičnega uporabniškega vmesnika. Ta vmesnik omogoča konfiguracijo sistema preko gradnikov (*widgets*), ki se jih dodaja v shemo (*scheme*), vsak gradnik ima vhod na levi strani in izhod na desni strani grafičnega objekta, ki je predstavljen kot grafična ikona. Primer grafičnega povezovanja vidimo na sliki 4.5. Na vhod gradnika je možno povezati samo z izhodom drugega gradnika.



Slika 4.5: Primer strukture sheme in povezav.



Slika 4.6: Primer konfiguracije gradnika.

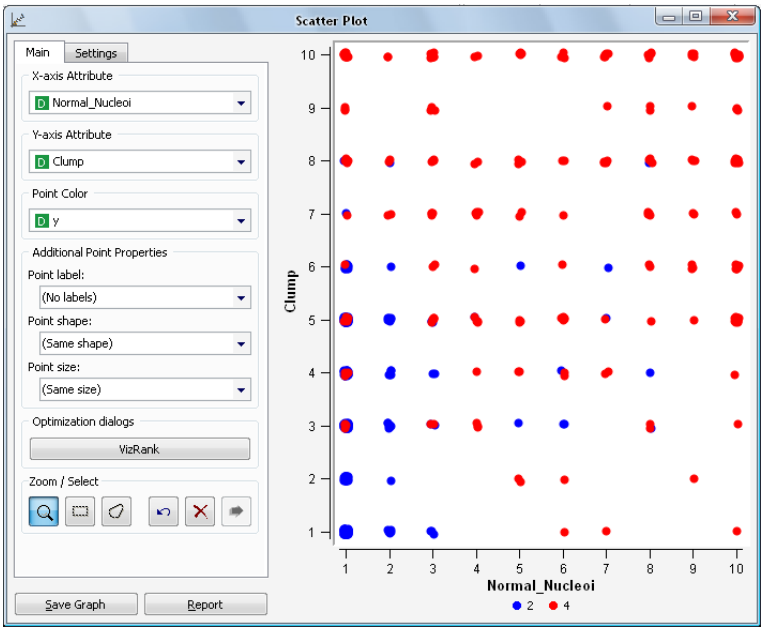


Slika 4.7: Primer klasifikatorja z naivnim Bayesovim klasifikatorjem v sistemu Orange.

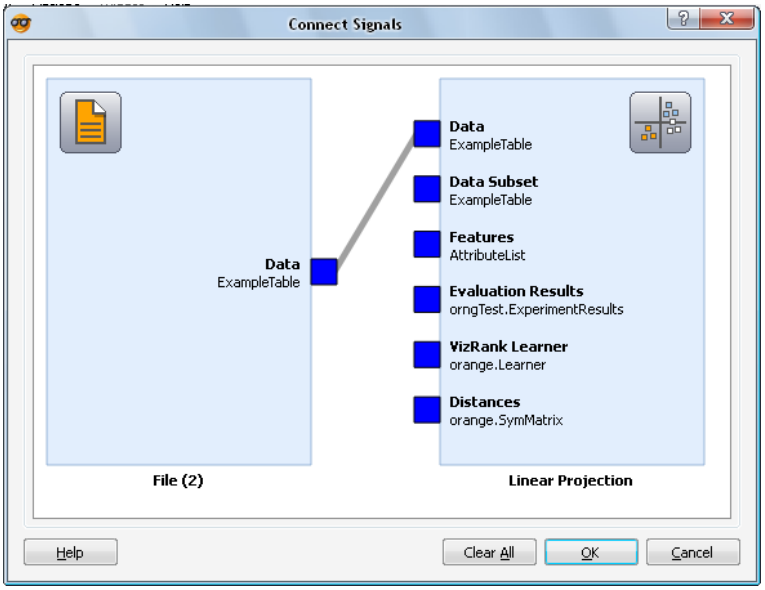
Vsak gradnik poleg vhoda in izhoda ima svojo konfiguracijo, ki določa kako bo gradnik vsebinsko deloval na vhodne in izhodne podatke. Na sliki 4.6 vidimo primer konfiguracije gradnika naivnega Bayes-a. Na sliki 4.7 pa je prikazan osnovni primer uporabe Orange grafičnega vmesnika na praktičnem primeru uporabe naivnega Bayes-a. Končni rezultat primera je viden na sliki 4.8, ko se izriše klasifikacija rezultata na grafu, ki je predstavljen kot gradnik pod imenom *Scatterplot Graph*.

Ker potrebuje večina gradnikov več kot en vhodni podatek posebno, ko se izrisujejo grafi, Orange omogoča, da se povezave med elementi bolj natančno določijo. Zato z dvojnim klikom miške na povezavo prikaže nov dialog s katerim lahko preusmerimo povezave znotraj vhoda in izhoda med dvema gradnikoma tak primer je prikazan na sliki 4.9. Orange podpira naslednje podatkovne tipe:

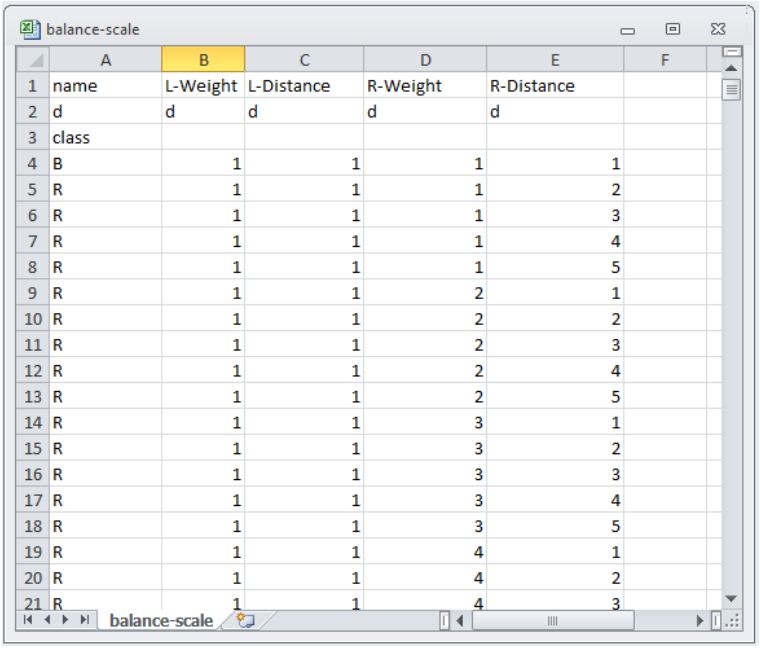
- C4.5,
- Assistant,
- Retis,



Slika 4.8: Primer izrisa grafa.



Slika 4.9: Povezovanje vhodov na izhode.



balance-scale

	A	B	C	D	E	F
1	name	L-Weight	L-Distance	R-Weight	R-Distance	
2	d	d	d	d	d	
3	class					
4	B	1	1	1	1	
5	R	1	1	1	2	
6	R	1	1	1	3	
7	R	1	1	1	4	
8	R	1	1	1	5	
9	R	1	1	2	1	
10	R	1	1	2	2	
11	R	1	1	2	3	
12	R	1	1	2	4	
13	R	1	1	2	5	
14	R	1	1	3	1	
15	R	1	1	3	2	
16	R	1	1	3	3	
17	R	1	1	3	4	
18	R	1	1	3	5	
19	R	1	1	4	1	
20	R	1	1	4	2	
21	R	1	1	4	3	

balance-scale

Slika 4.10: Primer podatkovne datoteke v Orange.

- Tab delimited,
- CSV (*comma separated values*),
- Relacijske podatkovne baze (MySql, Postgres in sqlite).

Osnovni tip datoteke Orange je *tab-delimited* datoteka, ki ima v prvi vrstici datoteke določena imena atributov in razredov. Druga vrstica določa tip atributa (*continuous* ali *discrete*), se pravi, da je lahko vrednost atributa številčna ali diskretna vrednost. Namesto vrednosti *discrete* se lahko napiše okrajšano s črko *d* ali številčne vrednosti *continuous* s črko *c*. Tretja vrstica določa dodatni opis vsakega stolpca in z besedo *class* se določa ali stolp določa razred (*label*) za vrstico. Lahko se določi tudi, da se stolp ne upošteva z besedo *ignore* ali skrajšano s črko *i*. Slika 4.10 prikazuje primer datoteke s podatki in njihovimi meta informacijami. Orange podpira tudi redke podatke (angl. *sparse data*) v obliki basket tipa formata (*.basket*), vrstice v tem formatu so vrednosti ločene s vejico. Drugi način uporabe programskega vmesnika v Orange pa je pisanje kode v Python program-

skem jeziku. Programski vmesnik Orange v Python okolju uporablja svoje tipe za delo s podatki. Podpora za vhodne podatkovne tipe v Orange je obsežna kot je tudi obsežna knjižnica za obdelovanje podatkov in izris grafov. V naslednjem odstavkih bomo opisali glavne gradnike knjižnice Orange.

Razred *Table* nam omogoča enostavno manipuliranje s podatki, nalaganje podatkov iz datotek, izbire, spreminjanje in odstranjevanje podatkov ter generiranje naključnih vrednosti v tabeli. Omogoča združevanje tabel, filtriranje podatkov, pretvarjanje tabel v *NumPy* podatkovne tipe, generiranje kontrolne vsote nad podatki, odpravo podvojenih vrstic in razvrščanje podatkov. Vsak objekt *Table* pripada nekemu *Domain* objektu. *Instance* je razred, ki drži podatke. Vsak objekt tipa *Instance* pripada nekemu objektu tipa *Domain*. Objektu se lahko dodajajo dodatni meta podatki, kot na primer vrednost uteži. Razred *Value* vsebuje vrednost spremenljivke. Vrednost je lahko diskretna, številčna ali drugega tipa. Če je vrednost diskretna ima svoj šifrant možnih vrednostih. Programski vmesnik knjižnice Orange se razdeli na module, ki so družina določenih algoritmov: Modul *Feature*, ki se največ uporablja za računanje uporabnosti, kot je na primer razred *scoring* ta ocenjuje uporabnost značilnosti za napovedovanje odvisnih spremenljivk. Modul *Miscellaneous* ponuja dodatne razrede, ki ne spadajo pod zgoraj omenjene module. Ta modul ima tri objekte, prvi je *CostMatrix* objekt, ki shranjuje ceno klasifikacije ali ne-klasifikacije, cena je lahko pozitivna ali negativna. Drugi razred je *SymMatrix*, ki implementira simetrične matrike določene velikosti. In tretji razred je *Random*, ki implementira generator naključnih števil z implementacijo *Mersenne twister* algoritma. Modul *Evaluation* ponuja vrednotenje napovedovalnih modulov in je razdeljen na dva dela. Modul *Orange.evaluation.testing* vsebuje procedure, ki vzorčijo podatke, naučijo učne algoritme in testirajo modele. Drugi del je modul *Orange.evaluation.scoring*, ki uporablja te podatke, da izračuna različne vrednosti delovanj klasifikatorjev in ostalih algoritmov. Modul *Tuning* je namenjen uglasitvi

parametrov algoritmov z uporabo notranje validacije in uglasitvi pragov (*threshold*) za klasifikatorje. Moduk *Distance* ponuja rešitve pri računanju razdalj med dvema instancama. Orange ne podpira *online* procesiranja podatkov kar pomeni, da se morajo vsi podatki naložiti v pomnilnik, da se jih lahko procesira. In tudi ne podpira paralelizma procesiranja podatkov.

4.4.3 Algoritmi

Orange poleg svojih implementiranih algoritmov ponuja ovojnice okoli algoritmov knjižnici LibSVM in LIBLINEAR, ki niso našteti spodaj ter poleg teh dveh knjižnic si Orange sposoja še implementacijo iz knjižnic kot sta Scikit in Earth R package. Modul *Classification*, deluje na principu dveh objektnih shem. Učni algoritem je implementiran iz osnovnega vmesnika *Orange.classification.Learner*. Ta učni objekt shrani vse parametre od učnega algoritma. Da bi lahko prišli do dodatnih informacij iz podatkov moramo prvo narediti instanco objekta, ki predstavlja učni algoritem in mu nastaviti ustrezne parametre. Šele klicanje takega objekta z učnimi podatki nam poda klasifikator, se pravi učni objekt se ne »nauči« ampak kreira objekt za klasifikacijo podatkov. Modul implementira naslednje klasifikatorje.

- Naive Bayes classifier (*bayes*),
- K-nearest neighbors (*knn*),
- Rule induction (*rules*),
- Support Vector Machines (*svm*),
- Classification trees (*tree*),
- Logistic regression (*logreg*),
- Majority (*majority*),
- Lookup classifiers (*lookup*),
- Classifier from variable,
- Constant Classifier.

Modul *Regression* deluje na podobni način kot modul *classifcation* osnovni učni vmesnik za regresijo je *BaseRegressionLearner*. Modul implementira

naslednje tehnike regresij:

- Linear regression (*linear*),
- Lasso regression (*lasso*),
- Partial least sqaures regression (*PLS*),
- Multivariate Adaptive Regression Splines (*earth*),
- Regression trees (*tree*),
- Mean (*mean*).

Modul *Statistics* ponuja več implementacij za računanje statistik nad podatki. Modul podpira naslednje nabore statistik.

- Basic Statistics for Continuous Features (*basic*),
  min max, avg, dev, n, sum , itd.,
- Contingency table (*contingency*),
- Distributions (*distribution*),
- Probability Estimation (*estimate*).

Modul *Ensemble* ponuja algoritme, ki izboljšujejo delovanje napovedi in podpira naslednje implementacije algoritmov:

- Bagging,
- Boosting,
- Stacking,
- Random Forest.

Modul *Multi-label classification* ponuja več algoritmov, ki rešujejo problem kjer se napoveduje več binarnih spremenljivk (*labels*). Modul *Multi-target* ponuja algoritme, ki težijo k cilju, da podajo boljšo natančnost pri napovedi ali hitrosti pri napovedi več odvisnih spremenljivk hkrati. Modul ponuja naslednje algoritme:

- Multi-target Tree Learner,
- Partial least sqaures regression (*PLS*),
- Multivariate Adaptive Regression Splines (*earth*).

Modul *Association rules and frequent itemsets* ponuja dva algoritma za uporabo asociacijskih pravil, standardni Apriori asociacijski algoritem ter različica Apriori algoritma za atributne-vrednostne podatkovne množice. Modul *Clustering* ponuja tri metode razvrščanja v skupine:

- K-means (*kmeans*),
- Hierarchical (*hierarchical*),
- Consensus (*consensus*).

Modul *Network*, vsebuje implementacije mrež in grafov. Modul *Projections* vsebuje različne transformacije podatkov, kot so:

- Linear projection (*linear*):
  Pricipal Component Analysis (*pca*),
  Fisher discriminant analysis (*fda*),
  FreeViz.
- Multidimensional scaling (*mds*),
- Self-organizing maps (*som*):
  Inference of Self-Organizing Maps,
  Supervised Learning with Self-Organizing Maps.
- Correspondence Analysis (*correspondence*).

S knjižnico Orange bi lahko popolnoma reševali probleme iskanju znanj iz podatkov z uporabo metodologije CRISP predvsem zaradi obilico algoritmov za obdelavo podatkov in močni podpori cross validaciji.

4.5 PyBrain

4.5.1 Dokumentacija

PyBrain ima na svoji spletni strani *http://pybrain.org/docs/* naslednje dokumentacijo:

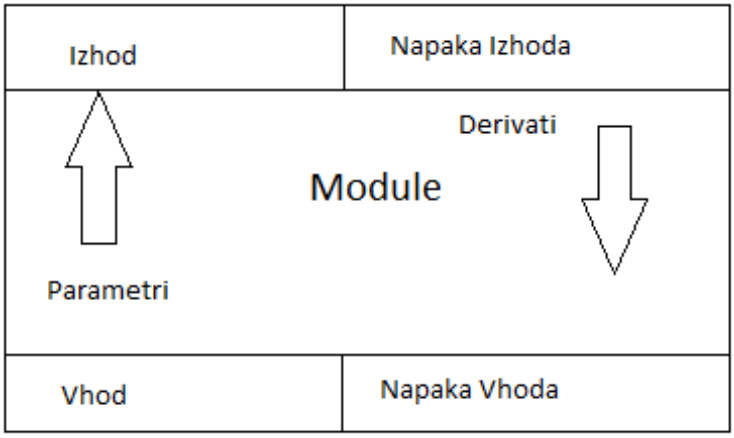
- uporabniško dokumentacijo za začetnike in napredne uporabnike

- dokumentacijo programskega vmesnika v obliki html, ki na hitro in skromno našteje glavne metode razredov

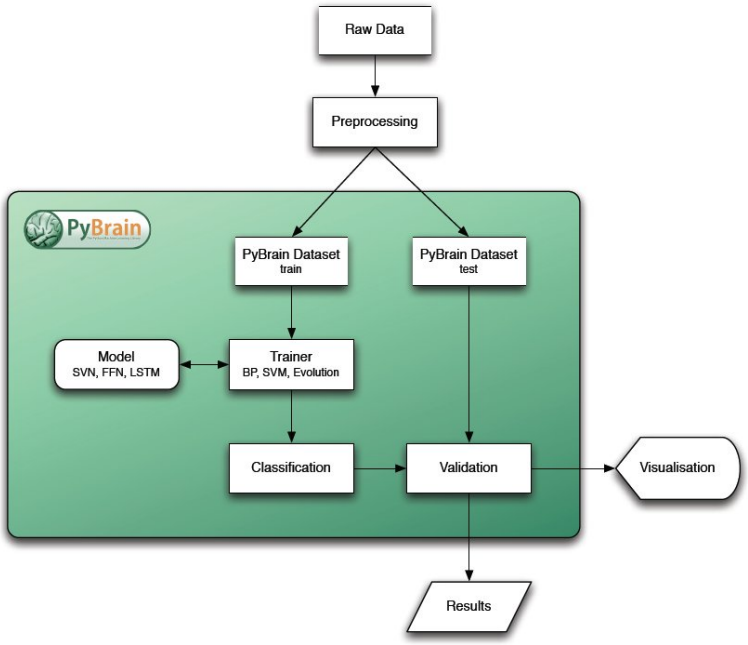
Uporabniška dokumentacija je sestavljena iz treh delov, hitra navodila za uporabo, vodnik za uporabnike ter napredna navodila, ki pokrivajo dodatne module kot so hitre nevronske mreže ter moduli za podporo ODE okolju. Dokumentacija za uporabnike in dokumentacija programskega vmesnika sta zelo pičli in sta napisani minimalistično. Uporabniška dokumentacija vsebuje tudi nekaj strani za razvojnike, ki bi radi razširili knjižnico PyBrain, vendar je tudi ta del zelo skop. Ker ima knjižnica zelo ozek nabor dokumentacij in pomanjkljivo uporabniško dokumentacijo in zelo omejena omembo arhitekture in zasnove smo dokumentaciji dali končno oceno 2.

4.5.2 Programski vmesnik

Osnovni razred v knjižnici PyBrain, ki zaobjame različne algoritme je modul (*Module*). Objekti tega tipa imajo vhodne in izhodne medpomnilnike ter medpomnilnike za napake, ki se uporabljajo v tako imenovanih *error backpropagation* algoritmih. Ti objekti se vključujejo v objekte tipa mrež (*Network*) in se med seboj povezujejo z objekti povezav (*Connection*), ki lahko vsebujejo različne parametre kot je na primer utež, ki je najbolj pogost uporabljani parameter. Objekte *Network* se lahko poljubno povezuje med ostalimi objekti *Module* z omejitvijo, da končni graf povezav tvori direktni aciklični graf. Na sliki 4.11 je prikazana abstraktna struktura modulov. Parametre objektov *Network* se prilagajajo preko razreda *Trainer*, ki uporabljajo podatkovne objekte *Dataset* s katerimi preko učenja optimalno prilagodijo parametre iz primerov. Na sliki 4.12 je prikaz primera procesnega in podatkovnega toka v knjižnici PyBrain. PyBrain omogoča objekte tipa *Dataset*, ki olajšajo delo s podatki, ko je potrebno učenje, testiranje in validiranje podatkov. *Dataset* si lahko predstavljamo kot zbirka imenskih 2D polj. Uporablja se predvsem pri mrežah pri katerih so določeni vhodni



Slika 4.11: Vhodi in izhodi modulov.



Slika 4.12: Potek podatkovnega in procesnega toka v PyBrain.

skriti ter omogoča tudi hitro generiranje naključnih števil. PyBrain podpira naslednje objekte tipa *Dataset*:

- *supervised*, ki se uporablja pri nadzorovanem regresijskem učenju,
- *sequential* se uporablja pri nadzorovanem sekvenčnem regresijskem učenju, starš tega objekta je *supervised*,
- *classification* se uporablja pri nadzorovanem klasifikacijskem učenju.

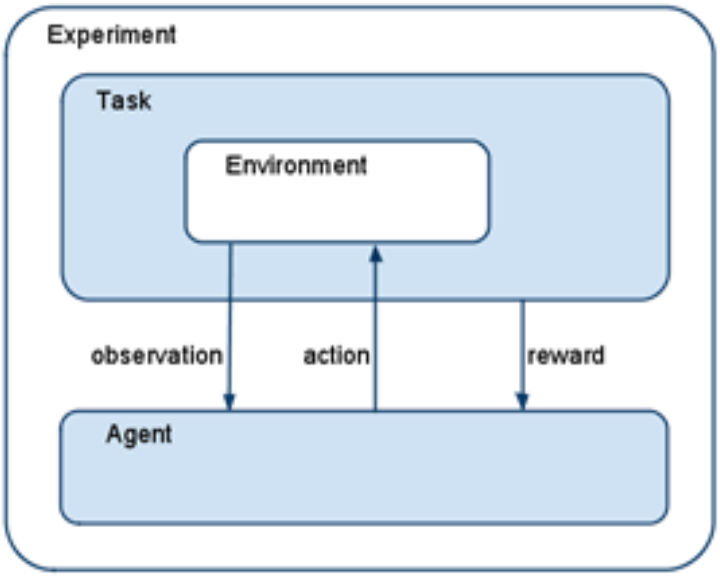
Čeprav PyBrain vsebuje lastne razrede za delo s podatki ti razredi nimajo lastnih metod za nalaganje podatkov iz vhodno/izhodnih sistemov kot so CSV datoteke in podatkovne baze. Zato smo implementirali metode, ki naložijo podatke iz CSV datotek in napolnijo *Dataset* objekte, te metode se nahajajo pod prilogo B. PyBrain ponuja dve vrsti implementacij za reševanje problema optimizacije:

- *BlackBoxOptimizer* je osnovni objekt, ki se uporablja za različne vrste množic,
- *ContinuousOptimizer* je osnovni objekt, ki se uporablja za zvezno optimizacijo.

PyBrain podpira tudi RL (*Reinforcement learning*), struktura in relacija objektov je prikazana na sliki 4.13. in na tej sliki je predstavljeno okolje (*Enviroment*), ki predstavlja svet s katerem sodelujejo agenti (*Agent*) in v katerem se dejansko izvaja učenje. Komponenta opravilo (*Task*) pa povezuje okolje in agente skupaj. Komponenta določa kaj je cilj v okolju in kako je agent nagrajen za svoja dejanja. PyBrain podpira tudi n-fold cross validacijo in računanje performanc klasifikatorjev. PyBrain ne podpira *online* procesiranja podatkov kar pomeni, da se morajo vsi podatki naložiti v pomnilnik, da se jih lahko procesira. PyBrain ne podpira paralelizem procesiranja podatkov.

4.5.3 Algoritmi

PyBrain ima implementirane ovojnice okoli LIBSVM knjižnice, vendar se te knjižnice ne namestijo s paketom PyBrain, ampak je potrebno posebej



Slika 4.13: Struktura in relacija objektov v RL.

namestiti specifično verzijo LIBSVM knjižnice vendar katere verzije je potrebno namestiti ni definirano v dokumentaciji, ker z novimi različicami LIBSVM knjižnice primeri ne delujejo. Pri k-Means algoritmu je neznačilno to, da ne podpira PyBrain *DataSet* podatkovnih tipov ampak sprejema samo vhodne podatke v obliki polj knjižnice *Numpy*. Knjižnica PyBrain implementira lastne algoritme klasifikacij kot so Back-Propagation, Evolino, R-Prop in RBF mreže, implementira tudi algoritem grozdenja k-Means. Knjižnica podpira premalo učnih algoritmov in algoritmov procesiranj podatkov, posebno ko niti ne implementira lastnega nalaganja podatkov v podatkovne strukture in s tem ne more uspešno pokrivati operacij, ki so potrebne, da bi lahko popolnoma reševali probleme iskanju znanj iz podatkov z uporabo metodologije CRISP.

4.6 PyML

4.6.1 Dokumentacija

PyML ima na svoji spletni strani *http://pyml.sourceforge.net/tutorial.html* naslednjo dokumentacijo:

- uporabniško dokumentacijo vodnika uporabe knjižnice.

Problem manjkajoče dokumentacije rešuje forum za izmenjavo informacij med razvojniki sistema in uporabniki na *sourceforge.net* strani knjižnice PyML. Ob namestitvi knjižnice PyML se namesti tudi uporabniška dokumentacija vodnika v pdf formatu, ki vsebuje 18 strani. Poleg datoteke pdf se namesti dokumentacija programskega vmesnika v obliki html, ki se generira iz izvorne kode s programsko aplikacijo Epydoc (*http://epydoc.sourceforge.net*). Končna ocena dokumentacije je 1 (dokumentacija je minimalistična in ni v pomoč uporabnikom, ki želijo uporabljati knjižnico za svoje namene).

4.6.2 Programski vmesnik

PyML vsebuje objekte tipa *DataSet*, ki omogočajo lažjo manipulacijo podatkov. Podpira naslednje podtipe:

- *VectorDataSet* vsebuje množice podatkovnih polj,
- *SparseDataSet* vsebuje raztresene množice podatkovnih polj,
- *KernalData* vsebuje pred izračunane matrike jedr,
- *SequenceData* vsebuje polja nizov,
- *PairDataSet* vsebuje vzorce, ki so sestavljeni iz objektnih parov,
- *Aggregate* združuje več množic podatkovnih polj v eno polje.

PyML ponuja nekaj metod kako nalagati podatke v instance tipa *DataSet*. Prva možnost je, da lahko bere podatke iz datotek in pri tem podpira več datotečnih formatov, kot so raztresen format, ki je kompatibilen

s formatom, ki ga uporabljata knjižnici libsvm in svmlight. Lahko uporablja tudi razmejeno datoteko (angl. *delimited file*), ki je lahko ločena z vejico, tabom ali s presledkom. Podpira tudi branje obeh prej omenjenih tipov datotek, ki so stisnjene s knjižnico *gzip* in podatke samodejno prekodira iz gzip stisnjenega formata v izvorno obliko in jih naloži v *Dataset*. Drugi način nalaganje podatkov je kreiranje dvo dimenzionalnih polj iz *NumPy* knjižnice ali iz seznama Python seznamov. Večina klasifikatorjev v PyML knjižnici so jedrni klasifikatorji, kjer se uporabljajo jedrne funkcije, ki računajo podobnost dvojic v vzorcih. Vsak *Dataset* ima svoj jedrni objekt in privzeto jedrno je vedno linearnega tipa. Seveda se lahko določi različni jedrni objekt naknadno na podatkovni množici. Vsi klasifikatorji v PyML so implementirani od istega vmesnika in ponujajo naslednje klice

- *train*(data) nauči klasifikator na vhodnih podatkih,
- *test*(data) testira klasifikator na vhodnih podatkih,
- *trainTest*(data, trainingPatterns, testingPattersm),
- *cv*(data) *prečno preverjanje*, preveri performančne aspekte ,
- *stratified*(data),
- *loo*(data).

Večina klasifikatorjev tudi implementirajo klice, ki klasificirajo posamično vrstico iz množice podatkov:

- *classify*(data, i),
- *decisionFunc*(data, i).

Rezultat klica *test* na klasifikatorjih in križnega vrednotenja (*cross-validation*) se vrne v objektu tipa *Result*. Vsak klasifikator ima svoj specifični tip objekta *Result*. Za regresijo se uporablja *RegressionResults* in za klasifikator pa tip objekta *ClassificationResults*. Objekt *Result* je sestavljen iz Pythonovega seznama, kjer vsak element v seznamu združuje določene sklope podatkov. Ko je rezultat križnega vrednotenja je rezultat število elementov v seznamu enak številu križnih gub (angl. *cross-validation folds*). Ko pa je rezultat uporabe klasifikacijske metode *test*, ima rezultat samo en element.

ClassificationResults vsebuje množico metod, ki omogočajo uporabniku, da pridobi natančne informacije o rezultatu testiranj klasifikatorjev. PyML ne podpira *online* procesiranja podatkov kar pomeni, da se morajo vsi podatki naložiti v pomnilnik, da se jih lahko procesira. In tudi ne podpira paralelizem procesiranja podatkov.

4.6.3 Algoritmi

Knjižnica PyML implementira naslednje algoritme za klasifikacijo: SVM, k-NN in Multi Class družino klasifikatorjev (angl. One Against Rest in One Against One). Knjižnica implementira dva algoritma regresije in to sta Ridge in Support Vector. Poleg regresije in klasifikacijskih algoritmov knjižnica implementira tudi algoritem razvrščanje v skupine k-Means. Knjižnica podpira premalo učnih algoritmov in algoritmov za procesiranje podatkov in s tem ne more uspešno pokrivati operacij, ki so potrebne, da bi lahko popolnoma reševali probleme iskanju znanj iz podatkov z uporabo metodologije CRISP.

4.7 Shogun

4.7.1 Dokumentacija

Dokumentacija Shogun se nahaja na spletni strani *http://www.shogun-toolbox.org/doc/en/current/index.html* in ima naslednje tipe dokumentacij:

- dokumentacija uporabniškega vodnika,
- dokumentirani primeri programske kode za podprte programske jezike vključno s primeri za Python,
- dokumentirana lista implementacij različnih algoritmov,
- dokumentacija programskega vmesnika v obliki html, ki je generirana z aplikacijo Doxygen (*http://www.stack.nl/ dimitri/doxygen*),

- dokumentacija za razvoj novih komponent v knjižnici Shogun.

Za vse zgoraj naštete velja, da so ti dokumenti napisani z zelo skopimi informacijami. Nekako se zdi, da avtorji knjižnice pričakujejo, da se bodo uporabniki naučili uporabljati njihovo knjižnico samo preko primerov, ki so napisani za različne programske jezike in ki niso komentirani po vrsticah kode. Najbolj se občuti pomanjkanje dokumentacije arhitekture in procesov, kjer bi bila opisana arhitektura in splošna uporaba knjižnice. V dokumentaciji za razvoj novih komponent se na primer polovica besedila nanaša, kako je potrebno ali ni potrebno skrbeti za uhajanje pomnilnika, ker knjižnica podpira referenčne kazalce. Skoraj neverjetno se zdi, da so avtorji posvetili toliko besedila na domači strani, ki opisuje kako je knjižnica Shogun bolj bogata v naboru implementacij v primerjavi z drugimi konkurenčnimi knjižnicami, kot pa pri sami dokumentacija knjižnice. Že v tako skopi dokumentaciji uporabniškega vodnika so posamezni primeri naenkrat opisani za vse programske jezike skupaj, kar naredi še tisto malo dokumentacije za nepregledno. Edini dokument, ki je objavljen in primerljiv z ostalimi knjižnicami je dokument programskega vmesnika, urejen po razredih. Ker praktično ni uporabne dokumentacije za knjižnico Shogun in je simbolične narave smo dokumentaciji dali končno oceno 1.

4.7.2 Programski vmesnik

Programska knjižnica se zelo osredotoča na jedrne implementacije in implementacije postopkov SVM (*Support Vector Machines*). Shogun ponuja dva tipa implementacij uporabe objektov: statični in modularni vmesnik. Statični vmesnik naj bi se uporabljal v primeru, ko uporabnik potrebuje orodja za enostavne eksperimente, kot na primer samo izvajanje učenja in evaluacije nad klasifikatorjem in nič več. Modularni vmesnik omogoča, da različne implementacije klasifikatorjev in ostalih objektov delijo in uporabljajo podatke med seboj. Shogun ima svojo implementacijo objektov, ki shranjujejo in manipulirajo podatke in vsi izhajajo iz osnovnega razreda

Features, ki tvori naslednje tri osnove tipe:

- *SimpleFeatures* (dense matrices),
- *SparseFeatures* (sparse matrices),
- *StringFeatures* (množica znakovnih nizev).

Shogun podpira kar nekaj datotečnih formatov, kot so Hdf5, ASCII pri katerem se posamična vrednost polja zapiše v vsako svojo vrstico. Zna brati tudi datoteke, ki so zapisane v binarnem formatu v katerem se datoteka začne z glavo, ki ima definirane štiri znake SG00 ter izmenjujoče podatke s svojimi glavami, podpira pa tudi branje datotek knjižnice *Svmlight*. Čeprav Shogun vsebuje lastne razrede za delo s podatki ti razredi nimajo lastnih metod za nalaganje podatkov iz vhodno/izhodnih sistemov kot so CSV datoteke in podatkovnih baze. Zato smo implementirali metode, ki naložijo podatke iz datotek CSV in napolnijo *Features* objekte. Te metode se nahajajo v Prilogi C.

Tipični koraki pri uporabi Shogun knjižnice z modularnim vmesnikom so:

- nalaganje podatkov v *Features* tipe objektov, učne in testne podatke,
- kreacija poljubnega kernel objekta kateri ima kot vhodne podatke tipa *Features* (testni in učni podatki primerov),
- kreacija objekta *Labels*, ki vsebuje označene primere,
- kreacija poljubnega klasifikatorja kateri konstrukt sprejme kernel, ki smo ga prej kreirali kot vhodni parameter. Poljubni klasifikatorji imajo tudi svoje specifične parametre, ki jih algoritmi pričakujejo,
- na klasifikatorju se kliče učna metoda *train*,
- na klasifikatorju se kliče testna metoda *apply* s testnimi vzorci, ki vrne rezultat tipa *Labels*,
- po končanem testiranju lahko analiziramo rezultat z objektom *PerformanceMeasures*, ki vrača meritve natančnosti, ROC , F-score, itd.

Shogun implementira tudi kar zajetno število meritev, ki merijo zmogljivost algoritmov kot so naprimer družina meritev ROC (*Receiver Opera-*

ting Curve), PRC (*Precision Recall Curve*), DET (*Detection Error Tradeoff*), CC (*Cross Correlation coefficient*), *F-Measure*, merjenje natančnosti algoritmov pri napovedih, itd.

4.7.3 Algoritmi

Knjižnica Shogun se osredotoča, kot smo že omenili, na algoritme tipa SVM, ki pa jih implementira v sklop družin kot so linearni (Standard online perceptron, LDA, LPBoost, SVMPerf, LibLinear, SVMOcas, itd.), binarni (MPDSVM, GPBTSVM, itd.) in več razredni SVM algoritmi (GMNP-SVM, LibSVMs one vs. one multiclas, itd.). Od regresij knjižnica Shogun podpira Vector regresijo in Kernel Ridge regresijo. Algoritme distribucij so implementirani algoritmi HMM (Hidden Markov Models), Histogram, LinearHMM-Markov chains. Za grozdenje podatkov knjižnica Shogun implentira hierarhični in k-Means algoritem. Ostali algoritmi, ki jih knjižnica implementira so različice Multi kernel algoritmov ter k-NN algoritem. Knjižnica je premalo dokumentirana, da bi se lahko izvajalo kompleksne operacije nad podatki, posebno ko niti ne implementira lastnega nalaganja podatkov v podatkovne strukture in s tem ne more uspešno pokrivati operacij, ki so potrebne, da bi lahko popolnoma reševali probleme iskanju znanj iz podatkov z uporabo metodologije CRISP.

Poglavje 5

Odzivni časi in testni podatki

V tem poglavju se bomo osredotočili na odzivne časov algoritmov, ki jih analizirane knjižnice implementirajo. Predvsem bomo testirali algoritme, ki so del osnovne knjižnice. Se pravi, da namestitev paketa zagotovi vse potrebne dele, ki so potrebni za delovanje sistema. Ne bomo testirali algoritmov, ki so del drugih knjižnic in jo knjižnica preko ovojnic implementira in niso že del namestitve osnovne knjižnice. Poglavje odzivnih časov bomo razdelili na tri dele. Prvi je opis podatkov s katerimi bomo testirali odzivne čase. Drugi del vsebuje opis testnih platformah na katerih so se izvajali testi in tretji zadnji del vsebuje izpis rezultatov testiranih algoritmov v obliki tabel.

5.1 Podatki

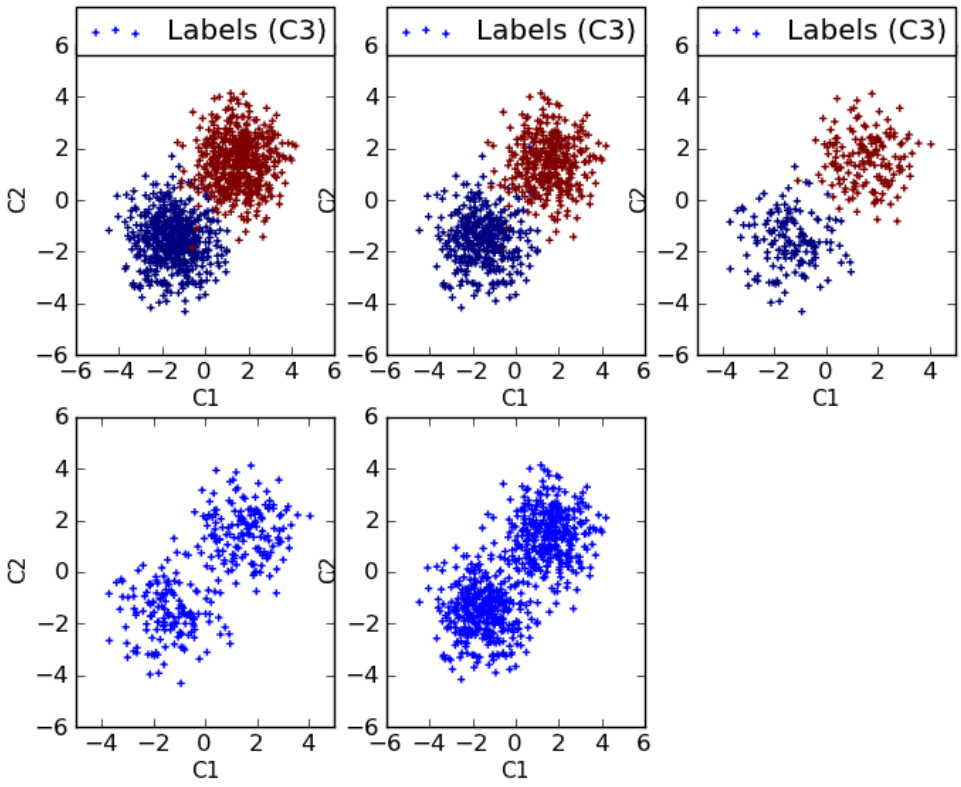
Ker nam je cilj primerjati performančnih razlik algoritmov med seboj moramo te algoritme testirati pod enakimi pogoji. Obstajata dva performančna pogoja, ki ju moramo zagotoviti, da so algoritmi enakovredno testirajo in to sta, da se vsi algoritmi izvajajo na isti strojni opremi ter, da so vhodni testni podatki enaki v sematični obliki za vse algoritme. Ker ima vsaka knjižnica in njeni algoritmi svoj način branja podatkov je potrebno strukturirati podatke tako, da ustrezajo specifičnim zahtevam vhodnih podat-

kov za algoritme, ki jih testiramo. Zato smo se odločili, da bomo podatke, ki jim bomo uporabili za učenje in testiranje generirani za različno število vrstic podatkov na odstotek.

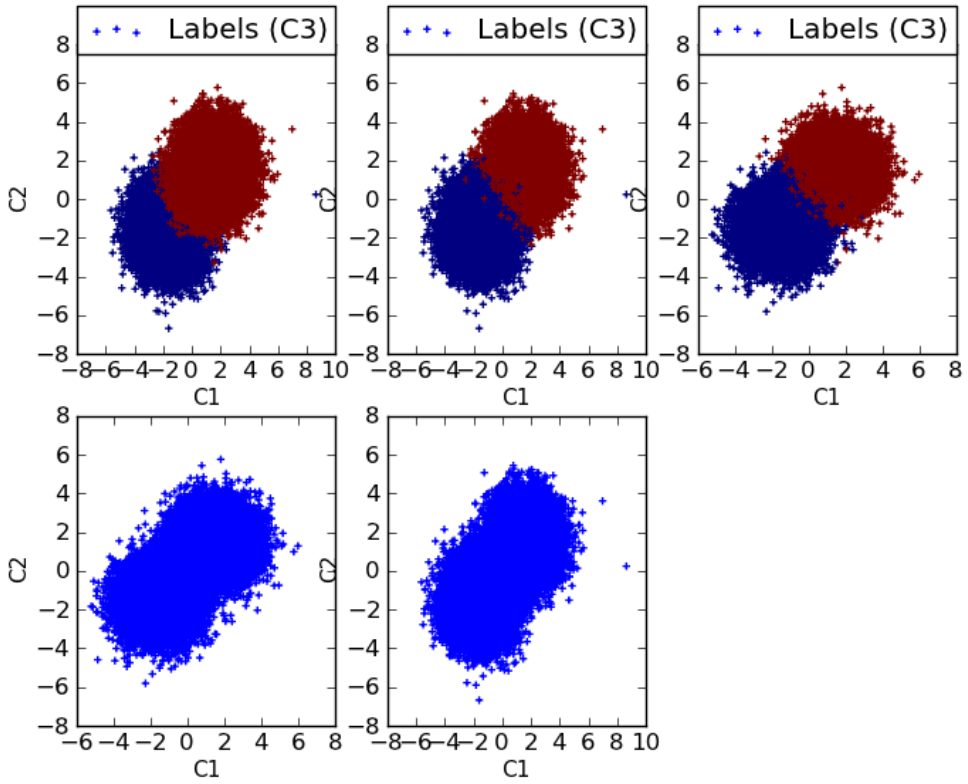
Za generiranje podatkov smo uporabili *Numpy* knjižnico in jih generirali s pomočjo Gaussovega naključnega algoritma s klicom metode *randn*. Izvorna koda za generiranje podatkov učenja in testiranja se nahajajo v Prilogi A. Podatke smo generirali za dve skupini. Prva skupina so binarni podatki pri katerih ima razred samo dve različni vrednosti in drugo vrsto podatkov z več kot dvema različnima vrednostnima razreda. Vsi podatki imajo definirana dva vhodna podatka in razred (2+1 stolpcev), zadnji stolpec določa razred. Poleg teh dveh skupin podatkov smo ti dve množici podatkov razbili še na učne in testne primere. Vzeli smo razmerje 70% od skupne množice podatkov v učne namene in preostalih 30% za testne namene. Za razdelitev podatkov smo si pomagali z metodo *Numpy shuffle*. V Prilogi D se nahaja programska koda Python za razbijanje osnovne datoteke v učne in testne množice in jih zapiše v datoteke z ustreznimi imeni.

Slika 5.1 prikazuje podatke dveh razredov s 1000 vrstic podatkov. Na sliki je prikazano pet grafov, skrajno levi graf zgoraj je prikaz osnovne datoteke z vsemi 1000 vrsticami podatkov. Rdeča in modra barva pa predstavljata pripadnost razreda. Zgornji sredinski graf prikazuje datoteko z učnimi podatki in njihove razrede in ti prestavljajo 70% podatkov osnovne množice. Zgornji desni graf pa prikazuje datoteko s testnimi primeri in njihovimi razredi. Ker nekateri algoritmi knjižnic ne sprejemajo datotek ali polj z razredi se generirajo tudi učne in testne datoteke brez razredov (v primeru določenih algoritmov, ki izvajajo razvrščanje v skupine in podobnih algoritmov). Ti podatki so vidni v spodnjem delu slike grafa, ki so obarvani z modro barvo. Graf v skrajno levem kotu predstavlja testne podatke brez razredov ter graf v skrajno desnem kotu, ki predstavlja učne podatke brez razredov.

Podatke za testiranje performanc algoritmov smo generirali za dvo in

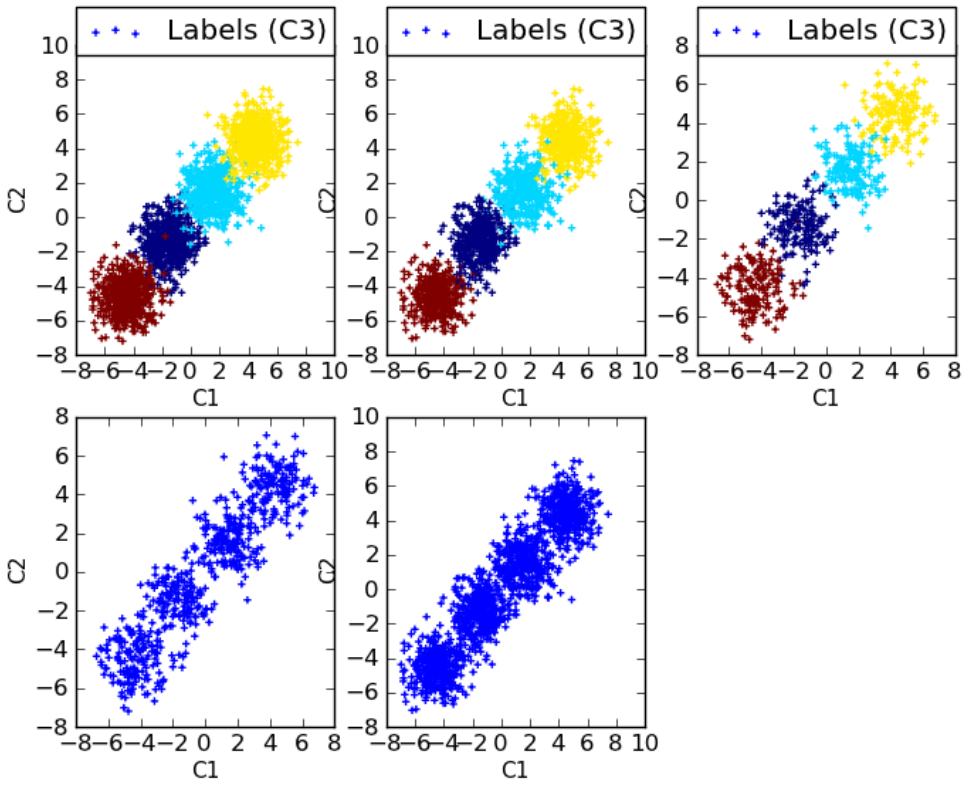


Slika 5.1: Slika datoteke dveh razredov s 1000 vrstic podatkov in njihovih podmnožic.



Slika 5.2: Slika datoteke dveh razredov s 100000 vrstic podatkov in njihovih podmnožic.

več razredne podatke v številu s 1000, 2000, 6000, 12000 in 100000 vrstic podatkov. Privzeta osnovna datoteka ima za vrednosti razredov -1 ali 1. V nekaterih datotekah, ki smo jih generirali iz te osnovne datoteke zaradi določenih knjižnic imajo za razred določene različne vrednosti razreda kot na primer: *true* ali *false* in 0 ali 1. Slika 5.2 pa prikazuje podatke dveh razredov s 100000 vrstic podatkov (struktura grafov je enaka kot iz slike 5.1). Kot smo že omenili smo generirali tudi datoteke, ki imajo več kot dva razreda, v našem primeru smo se odločili za 4 razrede. Primer več razrednih datotek vidimo na sliki 5.3. Za vsako množico podatkov smo generirali naslednje datoteke:



Slika 5.3: Slika datoteke štirih razredov s 1000 vrstic podatkov in njihovih podmnožic.

- osnovni datoteki z ustrezno količino vrstic (1000, 2000, itd), ki smo ju poimenovali *twoclass_(št.vrstic).csv* in *multiclass_(št.vrstic).csv*, ti datoteki vsebujeta vse podatke in njihove pripadajoče razrede,
- datoteki *twoclass_(št.vrstic)_test.csv* in *multiclass_(št.vrstic)_test.csv* vsebujeta testne primere in njihove pripadajoče razrede (30% naključnih vzorcev osnovne datoteke),
- datoteki *twoclass_(št.vrstic)_test_wc.csv* in *multiclass_(št.vrstic)_test_wc.csv* vsebujeta testne primere vendar brez njihovih pripadajočih razredov (30% naključnih vzorcev osnovne datoteke),
- datoteki *twoclass_(št.vrstic)_train.csv* in *multiclass_(št.vrstic)_train.csv* vsebujeta učne primere in njihove pripadajoče razrede (70% naključnih vzorcev osnovne datoteke),
- datoteki *twoclass_(št.vrstic)_train_wc.csv* in *multiclass_(št.vrstic)_train_wc.csv* vsebujeta učne primere vendar brez njihovih pripadajočih razredov (70% naključnih vzorcev osnovne datoteke),
- datoteki *twoclass_(št.vrstic)_train_labels.csv* in *multiclass_(št.vrstic)_train_labels.csv* vsebujeta le pripadajoče razrede podatkov iz datoteke *twoclass_(št.vrstic)_train_wc.csv* ali *multiclass_(št.vrstic)_train_wc.csv* po istem zaporedju podatkov (70% naključnih vzorcev osnovne datoteke).

Za potrebe posameznih knjižnic smo generirali še različice zgoraj omenjenih datotek. Pri transformaciji podatkov so podatki ostali enaki vendar smo jih zapisali v drugi format in transformirane dototeke so naslednje:

- za Shogun knjižnico smo generirali CSV datoteke, ki imajo presledek za ločilnik. Te datoteke smo imenovali *twoclass_sp_(št.vrstic)_*_.csv* in *multiclass_sp_(št.vrstic)_*_.csv*. Zaradi Shogun knjižnice smo generirali *label* datoteko v kateri so shranjene samo vrednosti razreda, brez ostalih spremenljivk (atributov),
- za Orange knjižnico smo generirali CSV datoteke, ki imajo za razred

določeno vrednost v določeno v črkovnih znakih, te datoteke smo imenovali *twoclass_str_(št.vrstic)_*_.csv*. Za Orange knjižnico smo naredili tudi različico TAB datotek, ki imajo ločilnik tab kontrolni znak in vrednost razreda določeno kot 0 ali 1 , te datoteke smo imenovali *twoclass_01_(št.vrstic)_*_.tab*.

5.2 Testne platforme

Za tesno platformo smo izbrali Oracle VM VirtualBox, ki omogoča postavljanje in zaganjanje virtualnih računalnikov, ki imajo lahko naložene različne operacijske sisteme. Za potrebe testiranja in namestitve knjižnic smo kreirali tri platforme na katere smo inštalirali knjižnice, ki smo jih analizirali. Za primarno okolje smo izbrali Windows okolje, ker pa dve knjižnici ne podpirata našega primarnega okolja smo za sekundarno okolje izbrali Linux. Med knjižnicami, ki smo jih testirali v Windows okolju obstaja knjižnica, ki ne podpira novejših različic Python-a zato smo še postavili okolje na katerem je inštalirana samo starejša različica Pyhton-a. Konfigurirali smo tri platforme in na prvo platformo smo namestili Windows XP SP3, ki je imel nameščen različico Python 2.7 in na katerem smo namestili naslednje knjižnice: Mdp, OpenCV, Orange in PyBrain. Na drugi platformi smo namestili Windows XP SP3, ki je imel nameščen različico Python 2.6 in na katerem smo namestili knjižnico Elefant. Na tretji in zadnji platfomi smo namestili 32 bitno distribucijo Linux Ubuntu 11.10, ki je imela nameščen različico Python 2.7 in na katerem smo namestili knjižnici PyML in Shogun.

Vse platforme so bile konfigurirane na isto virtualno strojno konfiguracijo. Konfiguracija je bila naslednja:

- 1x procesor,
- 4 GB pomnilnika RAM,
- 15 GB velikost trdega diska,

- 128 MB grafičnega pomnilnika s podporo grafičnega pospeševanja in podporo pospeševanja 3d funkcij.

Zgoraj omenjene platforme so potekale na istem testnem fizičnem računalniku, ki je imel naslednjo strojno opremo:

- procesor 2.2 GHz Intel Core i7-2670QM (4 procesorjev ali 8 HT enot),
- 8 GB RAM pomnilnika (DDR3),
- 1 TB trdi disk (5400 rpm),
- grafična kartica AMD Radeon HD 6770M z DDR5 pomnilnikom.

Testni računalnik je imel nameščen 64 bitni operacijski sistem Windows 7 Home ter Oracle VM VirtualBox različice 4.1.10.

5.3 Odzivni časi knjižnic

V tabelah, ki se nahajajo pod prilogo F smo predstavili rezultate glavnih in podpornih algoritmov, ki jih knjižnice podpirajo in so v delujočem stanju. Klici algoritmov so se izvajali na različnih različic zapisov podatkov, ki pa so bili izpeljani iz osnovne podatkovne datoteke, kot smo to opisali v poglavju 5.1, se pravi testirali smo na podatkih z 1000, 2000, 6000, 12000 ter 100000 vrsticami. V tabeli so zapisani naslednji parametri testa:

- *algoritem* prestavlja ime algoritma ali ime procesa za procesiranje podatkov,
- *čas učenja*, čas v sekundah, ki je bil potreben, da se je algoritem naučil, če pa je algoritem, ki procesira podatke, pa je to čas, ki je bil potreben, da se je algoritem izvedel,
- *čas testiranja*, čas v sekundah, ki je bil potreben, da je algoritem klasificiral ali izračunal napoved testnih podatkov, če je testna datoteka prazna, potem je tudi ta podatek brez vrednosti,
- *učna datoteka*, ime učne datoteke, ki smo jo uporabili za vhodne podatke učenja,

- *testna datoteka*, ime testne datoteke, ki smo jo uporabili za vhodne podatke testa,
- *čas nalaganja učne dat.*, meritveni čas nalaganje učne datoteke v sekundah,
- *čas nalaganja testne dat.*, meritveni čas nalaganje testne datoteke v sekundah.

Izvorno kod v programskem jeziku Python, ki smo uporabili za merjenje časa se nahaja v Prilogo E.

5.4 Diskusija

Če vzamemo ožji izbor rezultatov algoritmov knjižnic, ki smo jih objavili v prejšnjem poglavju lahko algoritme, ki se pojavljajo v večini knjižnic povzamemo v naslednje družine in njihove rezultate knjižnic od najhitrejših do najpočasnejših. Ožji izbor algoritmov, ki smo jih primerjali med seboj so sledeči: SVM (Support Vector Machine), k-NN (k-Nearest Neighbor), razvrščanje v skupine k-Means, hierarhično razvrščanje v skupine in Bayes klasifikacija. Pregledali smo rezultate za družino podatkov, ki imajo 100000 vrstic ter 12000 vrstic ker določeni algoritmi niso zmogli procesirati 100000 vrstic zaradi napak pomnilnika. V prvi koloni so imena knjižnic v drugi koloni tabele pa so časi merjeni v sekundah (učenje algoritma).

Iz teh tabel je razvidno, da je knjižnica OpenCV po hitrosti izvajanja algoritmov najhitrejša od ostalih knjižnic. Naslednja knjižnica, ki se približa OpenCV je knjižnica Shogun. Ti dve knjižnici se večinoma pojavljata na vrhu, predvsem OpenCV. Najbolj porazni rezultati pa so vidni v knjižnici Elefant ne toliko zaradi hitrosti algoritmov ampak v sami optimizaciji algoritmov pri obdelavi velikih in srednje velikih podatkov. Večina algoritmov v knjižnici ni mogla obdelati niti podatkov s 6000 vrsticami. Naslednja tabelo, ki smo naredili iz testov, ki smo jih prikazali v prejšnjem poglavju so časi nalaganja učnih podatkov. V prvem stolpcu so imena knjižnic v drugem stolpcu so rezultati pri nalaganju podatkov s 100000

Knjižnica	čas učenja (s)
OpenCV	0,5
PyML	56
Shogun (stat)	730
Shogun (mod)	1391
Orange	zmanjkalo pomnilnika
Elefant	zmanjkalo pomnilnika
PyBrain	nima
Mdp	nima

Tabela 5.1: prikazuje čas učenja algoritmov SVM s 100000 vrstic podatkov.

Knjižnica	čas učenja (s)
OpenCV	0,07
PyML	0,54
Shogun (mod)	10,3
Shogun (stat)	10,5
Orange	40,6
Elefant	zmanjkalo pomnilnika
PyBrain	nima
Mdp	nima

Tabela 5.2: prikazuje čas učenja algoritmov SVM s 12000 vrstic podatkov.

Knjižnica	čas učenja (s)
Shogun (mod)	0,0010
OpenCV	0,0012
Mdp	0,0050
Shogun (stat)	0,0300
Orange	0,0430
PyML	0,0580
Elefant	nima
PyBrain	nima

Tabela 5.3: prikazuje čas učenja algoritmov k-NN s 100000 vrstic podatkov.

Knjižnica	čas učenja (s)
OpenCV	0,000095
PyML	0,000115
Shogun (mod)	0,000290
Shogun (stat)	0,000380
Mdp	0,000490
Orange	0,003400
Elefant	nima
PyBrain	nima

Tabela 5.4: prikazuje čas učenja algoritmov k-NN s 12000 vrstic podatkov.

Knjižnica	čas učenja (s)
Shogun (stat)	0,000001
OpenCV	0,003100
Mdp	0,004900
Elefant	0,053000
Orange	0,680000
PyBrain	72065,00
Shogun (mod)	zmanjkalo pomnilnika
PyML	nima

Tabela 5.5: prikazuje čas učenja algoritmov k-Means s 100000 vrstic podatkov.

Knjižnica	čas učenja (s)
Shogun (stat)	0,000001
Mdp	0,000490
OpenCV	0,000700
Elefant	0,000800
Orange	0,080000
Shogun (mod)	9,600000
PyBrain	1280,000
PyML	nima

Tabela 5.6: prikazuje čas učenja algoritmov k-Means s 12000 vrstic podatkov.

Knjižnica	čas učenja (s)
Shogun (stat)	0,0000012
Shogun (mod)	9,6000000
Orange	63,800000
Mdp	nima
OpenCV	nima
Elefant	nima
PyBrain	nima
PyML	nima

Tabela 5.7: prikazuje čas učenja algoritmov Hierarchical s 12000 vrstic podatkov.

Knjižnica	čas učenja (s)
OpenCV	0,0046
Shogun (mod)	0,0050
Orange	2,3000
Shogun (stat)	nima
Mdp	nima
Elefant	nima
PyBrain	nima
PyML	nima

Tabela 5.8: prikazuje čas učenja algoritmov Bayes s 100000 vrstic podatkov.

Knjižnica	št. vrstic 100000	št. vrstic 12000
Shogun	0,012	0,09
Orange	0,050	0,50
PyBrain	0,120	1,10
Mdp	0,130	1,20
OpenCV	0,130	1,10
Elefant	0,260	2,20
PyML	0,260	2,10

Tabela 5.9: prikazuje čas nalaganja podatkov s 12000 in 100000 vrstic podatkov.

in v tretjem rezultati nalaganja s 12000 vrstic. Iz tabele je razvidno, da najhitrejše nalaganje podatkov omogoča knjižnica Shogun, takoj za to je knjižnica Orange. Iz tabele je tudi razvidno, da si knjižnice PyBrain, Mdp in OpenCV delijo kodo za nalaganje podatkov in to je s pomočjo knjižnice *Numpy* in metode *loadtxt*, zato so si časi enakovredni. Knjižnici Elefant in PyML imata tudi svoji implementaciji za nalaganje vhodno/izhodnih podatkov vendar sta ti dve knjižnici dosti počasnejši pri nalaganju podatkov kot sta Orange in Shogun.

Poglavje 6

Sklep

V tem diplomskem delu smo analizirali sedem knjižnic za odkrivanje znanj iz podatkov za programski jezik Python. Iz analiz, ki smo jih izvedli in objavili v prejšnjih poglavjih je razvidno, da je bila večina knjižnic implementirana za podporo pri razvoju novih algoritmov za akademske članke ter kot orodje za učenje študentov na izobraževalnih inštitucijah. Pri analizi smo opazili slabo podporo procesiranju veliki količini podatkov in tudi tiste knjižnice, ki oznanjajo, da podpirajo ta način to implementirajo samo na zelo omejeni množici algoritmov. Opazili smo tudi pomanjkanje podpori več procesnih računalnikov. Vsem razvijalcem knjižnic za odrivanje znanj iz podaktov, ki smo jih analizirali v tem diplomskem delu bi priporočili, da se osredotočijo na razvoj novih in spremembo starih algoritmov, ki bodo podpirali tako imenovane *online* procesiranje podatkov ter da bodo implementirali podporo več procesnih računalnikov, ki so v današnjem času norma in tudi zaradi tega ker procesiranje večje količine podatkov potrebuje več časa za končni rezultat z več procesorsko podporo pa bi ta čas spravili na čase, ki bi bili sprejemljivi za uporabnike.

Če bi morali izbrati eno od knjižnic, ki ponuja največ funkcionalnosti, da ni najpočasnejša knjižnica, ponuja natančno in uporabno dokumentacijo in je enostavna za namestitev bi izbrali knjižnico Orange. Knjižnica Orange ponuja največ raznolikih družin algoritmov, ponuja grafični upo-

rabniški vmesnik poleg vmesnika za Python in ponuja dokumentacijo in navodila za uporabnike, katere so izmed najbolje dodelani od vseh izbranih knjižnic in ima eno od najhitrejših nalaganj podatkov. Presenetljivo je tudi, da ima Orange hitrejše nalaganje podatkov kot jo ima knjižnica *Numpy* in to za faktor več kot dvakrat. Orange ponuja tudi lastno implementacijo prikazovanja rezultatov v obliki interakcij med atributi in ostalih parametrov kar jo postavlja v svoj razred, vse ostale omenjene knjižnice imajo za prikazovanje lastnih rezultatov zunanje knjižnice in omejeno izbiro izrisa rezultatov. Čeprav Orange nima najhitrejših algoritmov je pa med algoritmi na sredini kar se tiče hitrosti algoritmov in je očitno najboljša izbira, če bi potrebovali sistem s katerim bi lahko obdelovali podatke po metodi CRISP.

Vse glavne značilnosti, lastnosti in podpora funkcionalnosti knjižnic, ki smo jih analizirali smo predstavili v tabeli 6.1.

Funkcionalnost	Elefant	MDP	OpenCV	Orange	PyBrain	PyML	Shogun
Enostavnost namestitve / podpora Python platform	3	3	4	5	3	2	2
Dokumentacija	2	3	4	5	2	1	1
Grafični uporabniški vmesnik	da	ne	ne	da	ne	ne	ne
Podpora metodologiji CRISP-DM	ne	ne	ne	da	ne	ne	ne
Podpora klasifikatorjev	da	da	da	da	da	da	da
Podpora regresije	da	da	da	da	da	da	da
Podpora SVM	da	ne	da	da	da	da	da
Podpora Multiclass klasifikaciji	da	ne	da	da	da	da	da
Podpora grozdenju (clustering)	da	da	da	da	da	da	da
Podpora MLP	ne	da	da	ne	da	ne	ne
Podpora Cross validaciji	ne	ne	ne	da	da	da	da
Podpora za performančnih in ostalih meritev	ne	ne	ne	da	da	da	da
Podpora za paralelno izvajanje	ne	da	ne	ne	ne	ne	da
Podpora za lastne podatkovne strukture	da	ne	ne	da	da	da	da
Implementacija podpori nalaganju IO podatkov	da	ne	ne	da	ne	ne	ne
Javni forum za uporabnike	ne	ne	ne	da	ne	da	ne
Podpora velikih podatkovnih množic	ne	da	ne	ne	ne	ne	ne
Windows OS	da	da	da	da	da	ne	ne
Linux	da	da	da	da	da	da	da
Mac OS X	da	da	da	da	da	da	da

Tabela 6.1: Glavne značilnosti, lastnosti in podpora funkcionalnosti knjižnic

Literatura

[1] Curk T., Demšar H., X. Qikai, Leban G., Petrovič U., Bratko I., Shaulsky G., Zupan B. *Microarray data mining with visual programming*, Bioinformatics (Oxford University Press), strani 396–398, februar 2005.

[2] Bradski G., Kaehler A. *Learning OpenCV: Computer Vision with the OpenCV Library*, (O'Reilly Media), oktober 2008.

[3] Laganière R *OpenCV 2 Computer Vision Application Programming Cookbook*, (Packt Publishing), maj 2011.

[4] Konda P *Izvedba podatkovega rudarjeja v bančištvu z uporabo metodologije crisp-dm*, 2009.

[5] Marsland S., *Machine Learning: An Algorithmic Perspective*, (Chapman and Hall/Crc), April 2009.

[6] Schaul T., Bayer J., Wierstra D., Sun Y., Felder M., Sehnke F., Rückstieß T., Schmidhuber J. *PyBrain*, Journal of Machine Learning Research, 2010.

[7] Zito T., Wilbert N., Wiskott L., Berkes P, *Modular toolkit for Data Processing (MDP): a Python data processing frame work*, Front. Neuroinform, 2008.

[8] Bradski, G., *The OpenCV Library*, Dr. Dobb's Journal of Software Tools, 2008.

[9] Webers C., Gawande K., Smola A., Vishwanathan, Günter S., Hui Teo C., Qinfeng Shi J., McAuley J., Song L., Le Q., *Elefant User Manual (Revision 0.1)*, NICTA, 2009.

[10] Gawande K., Webers C., *PyPETSc User Manual (Revision 1.0)*, NICTA, 2009.

[11] Sonnenburg S., Raetsch G., Henschel S., Widmer C., Behr J, Zien A., Bona F., Binder A., Gehl C., Franc V., *The SHOGUN Machine Learning Toolbox*, Journal of Machine Learning Research, junij 2010.

Poglavje 7

Priloge

7.1 Priloga A: Programska koda za generiranje učnih in testnih podatkov

```
def CreateBinaryRandomData(rowsPerClass,dist,show=True,\
name='graph'):
    data = concatenate((randn(2,rowsPerClass)-dist,\
        randn(2,rowsPerClass)+dist), axis=1)

    a = chararray(rowsPerClass,itemsize=10)
    for i in range(rowsPerClass):
        a[i]='-1'

    b = chararray(rowsPerClass,itemsize=10)
    for i in range(rowsPerClass):
        b[i]='1'

    labels = concatenate((a,b))

    if show==True:
        pylab.figure()
        cols = {-1: "blue", 1: "green", 0: "red"}
        pylab.scatter(data[0],data[1], c=map(lambda x:cols[x],\
                labels))
        plt.savefig(filename = os.path.join(path, \
                name+'_scatter.png'))
        pylab.show()
    return data,labels

def CreateBinary01RandomData(rowsPerClass,dist,show=True,\
```

```
name='graph'):
    data = concatenate((randn(2,rowsPerClass)-dist, \
        randn(2,rowsPerClass)+dist), axis=1)

    a = chararray(rowsPerClass,itemsize=10)
    for i in range(rowsPerClass):
        a[i]='0'

    b = chararray(rowsPerClass,itemsize=10)
    for i in range(rowsPerClass):
        b[i]='1'
    labels = concatenate((a,b))

    if show==True:
        pylab.figure()
        cols = {-1: "blue", 1: "green", 0: "red"}
        pylab.scatter(data[0],data[1], c=map(lambda x:cols[x],\
                labels))
        plt.savefig(filename = os.path.join(path, name+\
                '_scatter.png'))
        pylab.show()
    return data,labels



def CreateBinaryStringRandomData(rowsPerClass,dist,show=True,\
name='graph'):
    data = concatenate((randn(2,rowsPerClass)-dist,\
        randn(2,rowsPerClass)+dist), axis=1)

    a = chararray(rowsPerClass,itemsize=10)
    for i in range(rowsPerClass):
        a[i]='false'

    b = chararray(rowsPerClass,itemsize=10)
    for i in range(rowsPerClass):
        b[i]='true'

    labels = concatenate((a,b))

    if show==True:
        pylab.figure()
        cols = {'false': "blue", 'true': "green"}
        pylab.scatter(data[0],data[1], c=map(lambda x:cols[x],\
                labels))
        plt.savefig(filename = os.path.join(path, name+\
                '_scatter.png'))
        pylab.show()
    return data,labels
```

```
def CreateMultiClassRandomData(rowsPerClass,dist,show=True,\
name='graph'):
    data = concatenate((randn(2,rowsPerClass)-dist,\
        randn(2,rowsPerClass)+dist), axis=1)
    data = concatenate((data, randn(2,rowsPerClass)+3*dist),\
        axis=1)
    data = concatenate((data, randn(2,rowsPerClass)-3*dist),\
        axis=1)

    a = empty(rowsPerClass)
    a.fill(0)

    b = empty(rowsPerClass)
    b.fill(1)

    c = empty(rowsPerClass)
    c.fill(2)

    d= empty(rowsPerClass)
    d.fill(3)

    labels = concatenate((concatenate((a, b)),c))
    labels = concatenate((labels,d))

    if show==True:
        pylab.figure()
        cols = {-1: "blue", 1: "green", 0: "red"}
        pylab.scatter(data[0],data[1], c=map(lambda x:cols[x],\
                labels))
        plt.savefig(filename = os.path.join(path, name+\
                '_scatter.png'))
        pylab.show()
    return data,labels
```

7.2 Priloga B: Nalaganje podatkov za PyBrain knjižnico

```
def np_get_dataset_sup(fileName,last_column_class,headers=False):

    if headers==True:
        array = numpy.loadtxt(fileName, delimiter=',', skiprows=1)
    else:
        array = numpy.loadtxt(fileName, delimiter=',', skiprows=0)
    # assume last field in csv is single target variable
    # and all other fields are input variables
```

```
    number_of_columns = array.shape[1]

    if last_column_class==True:
        dataset = SupervisedDataSet(number_of_columns - 1, 1)
    else:
        dataset = SupervisedDataSet(number_of_columns, 1)

    if last_column_class==True:
        dataset.setField('input', array[:,:-1])
        dataset.setField('target', array[:,-1:])
    else:
        dataset.setField('input', array[:,:])
    return dataset

def csv_get_dataset_sup(fileName,last_column_class,headers=False):
    reader = csv.reader(open(fileName, 'rb'))

    nfields = 0
    header = reader.next()
    fields = dict(zip(header, range(len(header))))
    nfields = len(fields)

    # assume last field in csv is single target variable
    # and all other fields are input variables
    if last_column_class == True:
        dataset = SupervisedDataSet(nfields - 1, 1)
        if headers==False:
            dataset.addSample(header[:-1], header[-1])

        for row in reader:
            dataset.addSample(row[:-1], row[-1])
    else:
        dataset = SupervisedDataSet(nfields, 0)
        if headers==False:
            dataset.addSample(header[:],0)
        for row in reader:
            dataset.addSample(row[:],0)
    return dataset

def csv_get_dataset_unsup(fileName,last_column_class,\
headers=False):
    reader = csv.reader(open(fileName, 'rb'))

    nfields = 0
    header = reader.next()
    fields = dict(zip(header, range(len(header))))
    nfields = len(fields)
    # assume last field in csv is single target variable
    # and all other fields are input variables
```

```
    if last_column_class == True:
        dataset = UnsupervisedDataSet(nfields - 1)
        if headers==False:
            dataset.addSample(header[:-1])

        for row in reader:
            dataset.addSample(row[:-1])
    else:
        dataset = UnsupervisedDataSet(nfields)
        if headers==False:
            dataset.addSample(header[:])
        for row in reader:
            dataset.addSample(row[:])
    return dataset


def csv_get_data(fileName,last_column_class,headers=False):
    reader = csv.reader(open(fileName, 'rb'))

    nfields = 0
    header = reader.next()
    fields = dict(zip(header, range(len(header))))
    nfields = len(fields)

    data = []
    targ = []

    # assume last field in csv is single target variable
    # and all other fields are input variables
    if last_column_class == True:
        if headers==False:
            data.append(header[:-1])
            targ.append(header[-1])

        for row in reader:
            data.append(row[:-1])
            targ.append(row[-1])
    else:
        if headers==False:
            data.append(header)
            targ.append(header[-1])
        for row in reader:
            data.append(row[:-1])
            targ.append(row[-1])

    return data,targ

def csv_get_dataset_sqe(fileName,last_column_class,headers=False,\
just_train=True):
```

```
    reader = csv.reader(open(fileName, 'rb'))

    nfields = 0
    header = reader.next()
    fields = dict(zip(header, range(len(header))))
    nfields = len(fields)

    # assume last field in csv is single target variable
    # and all other fields are input variables
    if just_train == True:
        dataset = SequentialDataSet(0, 1)
    else:
        dataset = SequentialDataSet(nfields-1, 1)
    dataset.newSequence()
    if headers==False:
        if just_train == True:
            dataset.addSample([], header[-1])
        else:
            dataset.addSample(header[:-1], header[-1])

    for row in reader:
        if just_train == True:
            dataset.addSample([], row[-1])
        else:
            dataset.addSample(row[:-1], row[-1])

    return dataset

def csv_get_dataset_cls(fileName,last_column_class=True,\
headers=False,convert=True):
    reader = csv.reader(open(fileName, 'rb'))
    list_data = list(reader)

    rows = len(list_data)
    cols = 0
    if rows>0:
        cols = len(list_data[0])
    if cols == 0: return None

    if headers==True:
        list_data.pop()


    # assume last field in csv is single target variable
    # and all other fields are input variables
    classes = numpy.unique([row[cols-1] for row in list_data])
    d=dict(zip(classes,range(len(classes))))

    attribs = cols-1
```

```
    ldata = []

    for row in list_data:
        ldata.append(row[:-1])

    data = numpy.array(ldata)
    ds=ClassificationDataSet(attribs,nb_classes=len(classes),\
        class_labels=classes)
    for i in range(len(data)):
        ds.appendLinked(data[i],d[list_data[i][-1]])
    if convert==True:
        ds._convertToOneOfMany()

    return ds
```

7.3 Priloga C: Nalaganje podatkov za Shogun knjižnico

```
def createTrainTestValidation(filename,train,test,val,deli=','):
    data = numpy.loadtxt(filename,delimiter=deli,skiprows=0,\
        dtype='string')
    rows = len(data)
    attrs = data.shape[1]

    print "%s file has rows: %d and %d attributes" % \
        (filename,rows,attrs)

    numpy.random.shuffle(data)

    train_data = data[:int(rows*train)]
    test_data = data[int(rows*train):int(rows*(train+test))]

    test_without_class = test_data.transpose()
    test_without_class = test_without_class[:attrs-1]
    test_without_class = test_without_class.transpose()

    new_filename = os.path.splitext(filename)[0]

    ext = 'csv'
    if deli=='\t':
        ext = 'tab'

    numpy.savetxt(new_filename+'_train.%s'%ext,train_data,\
        delimiter=deli, fmt="%s")
    numpy.savetxt(new_filename+'_test.%s'%ext,test_data,\
        delimiter=deli, fmt="%s")
    numpy.savetxt(new_filename+'_test_wc.%s'%ext,\
```

```
        test_without_class, delimiter=deli, fmt="%s")

    if val > 0.0:
        val_data = data[int(rows*(train+test)):int(rows*\
                (train+test+val))]
        numpy.savetxt(new_filename+'_validation.csv',val_data,\
                delimiter=deli, fmt="%s")

    train_data_without_labels = train_data.transpose()
    train_data_without_labels = \
        train_data_without_labels[:attrs-1]
    train_data_without_labels = \
        train_data_without_labels.transpose()
    numpy.savetxt(new_filename+'_train_wc.csv',\
        train_data_without_labels, delimiter=deli, fmt="%s")

    train_labels = train_data.transpose()
    train_labels = train_labels[attrs-1:attrs]
    train_labels = train_labels.transpose()
    numpy.savetxt(new_filename+'_train_labels.csv',train_labels,\
        delimiter=deli, fmt="%s")
```

7.4 Priloga D: Razbijanje osnovne datoteke na učne in testne množice

```
def createTrainTestValidation(filename,train,test,val,deli=','):
    data = numpy.loadtxt(filename,delimiter=deli,skiprows=0,\
        dtype='string')
    rows = len(data)
    attrs = data.shape[1]

    print "%s file has rows: %d and %d attributes" %\
        (filename,rows,attrs)

    numpy.random.shuffle(data)

    train_data = data[:int(rows*train)]
    test_data = data[int(rows*train):int(rows*(train+test))]

    test_without_class = test_data.transpose()
    test_without_class = test_without_class[:attrs-1]
    test_without_class = test_without_class.transpose()

    new_filename = os.path.splitext(filename)[0]
```

```
    ext = 'csv'
    if deli=='\t':
        ext = 'tab'

    numpy.savetxt(new_filename+'_train.%s'%ext,train_data,\
        delimiter=deli, fmt="%s")
    numpy.savetxt(new_filename+'_test.%s'%ext,test_data,\
        delimiter=deli, fmt="%s")
    numpy.savetxt(new_filename+'_test_wc.%s'%ext,\
        test_without_class, delimiter=deli, fmt="%s")

    if val > 0.0:
        val_data = data[int(rows*(train+test)):int(rows*(train+\
                test+val))]
        numpy.savetxt(new_filename+'_validation.csv',val_data,\
                delimiter=deli, fmt="%s")

    train_data_without_labels = train_data.transpose()
    train_data_without_labels = \
        train_data_without_labels[:attrs-1]
    train_data_without_labels = \
        train_data_without_labels.transpose()
    numpy.savetxt(new_filename+'_train_wc.csv',\
        train_data_without_labels, delimiter=deli, fmt="%s")

    train_labels = train_data.transpose()
    train_labels = train_labels[attrs-1:attrs]
    train_labels = train_labels.transpose()
    numpy.savetxt(new_filename+'_train_labels.csv',\
        train_labels, delimiter=deli, fmt="%s")
```

7.5 Priloga E: Metoda za merjenje izvajanje Python kode

```
def Profile(code,count,printText,setup):
    t = timeit.Timer(code,setup)
    try:
        time = t.timeit(count)
        timePerPass = (time /count )
        print printText + " took %.3f usec/pass" % timePerPass
        return timePerPass

    except MemoryError:
        print 'MemoryError'
        return -2.0
```

```
    except Exception as inst:
        print type(inst)
        return -3.0
```

7.6 Priloga F: Tabele merjenj hitrosti algoritmov

7.6.1 Elefant

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Epsilon SVM Regression	4,60	0,0012	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,129	0,007
Standard Gaussian Regression	0,29	0,10	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,022	0,007
Laplacian SVM Regression	3,41	0,0065	twoclass_1000_train.csv	twoclass_1000_test.csv	0,024	0,010
Nu SVM Regression	4,51	0,0231	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,023	0,008
Gaussian Process Hetero Scedastic Regression	1,65	0,0828	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,001	0,007
SVM Binary Classification	0,62	0,013	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,022	0,007
Nu SVM Binary Classification	1,85	0,0075	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,040	0,007
Gaussian Process Multiclass Classification	1,44	0,0084	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,040	0,007
Gaussian Process Extreme Multiclass Classification	1,35	0,0075	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,022	0,006
Diffusion Map Clustering	2,88		twoclass_1000_train.csv		0,021	
Kmeans Clustering	0,03		twoclass_1000_train_wc.csv		0,034	
Feature Selection BAHSIC	0,01		twoclass_1000_train_wc.csv		0,019	
Nonparametric quantile estimation	0,65	0,0068	twoclass_1000_train.csv	twoclass_1000_test_wc.csv	0,026	0,007

Tabela 7.1: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Epsilon SVM Regression	26,53	0,01	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,08	0,01
Standard Gaussian Regression	0,41	0,60	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,05	0,01
Laplacian SVM Regression	23,03	0,03	twoclass_2000_train.csv	twoclass_2000_test.csv	0,04	0,02
Nu SVM Regression	33,05	0,02	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,09	0,01
Gaussian Process Hetero Scedastic Regression	0,05	0,04	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,001	0,03
SVM Binary Classification	3,32	0,07	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,05	0,01
Nu SVM Binary Classification	10,60	0,01	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,05	0,01
Gaussian Process Multiclass Classification	4,14	0,037	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,04	0,01
Gaussian Process Extreme Multiclass Classification	4,11	0,04	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,05	0,01
Diffusion Map Clustering	13,31		twoclass_2000_train.csv		0,05	
Kmeans Clustering	0,0014		twoclass_2000_train_wc.csv		0,04	
Feature Selection BAHSIC	0,03		twoclass_2000_train_wc.csv		0,04	
Nonparametric quantile estimation	4,28	0,03	twoclass_2000_train.csv	twoclass_2000_test_wc.csv	0,09	0,02

Tabela 7.2: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Epsilon SVM Regression	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,13	0,04
Standard Gaussian Regression	6,14	12,42	twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,13	0,04
Laplacian SVM Regression	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test.csv	0,13	0,12
Nu SVM Regression	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,13	0,04
Gaussian Process Hetero Scedastic Regression	0,04	0,33	twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,00	0,04
SVM Binary Classification	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,13	0,04
Nu SVM Binary Classification	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,13	0,04
Gaussian Process Multiclass Classification	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,24	0,04
Gaussian Process Extreme Multiclass Classification	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,23	0,04
Diffusion Map Clustering	zmanj. pomn		twoclass_6000_train.csv		0,13	

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Kmeans Clustering	0,004		twoclass_6000_train_wc.csv		0,18	
Feature Selection BAHSIC	zmanj. pomn.		twoclass_6000_train_wc.csv		0,19	
Nonparametric quantile estimation	zmanj. pomn.		twoclass_6000_train.csv	twoclass_6000_test_wc.csv	0,24	0,04

Tabela 7.3: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Epsilon SVM Regression	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,28	0,15
Standard Gaussian Regression	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,27	0,18
Laplacian SVM Regression	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test.csv	0,27	0,12
Nu SVM Regression	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,27	0,08
Gaussian Process Hetero Scedastic Regression	1,69	zmanj. pomn.	twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,00	0,08
SVM Binary Classification	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,26	0,08
Nu SVM Binary Classification	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,28	0,09
Gaussian Process Multiclass Classification	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,26	0,09
Gaussian Process Extreme Multiclass Classification	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,27	0,08
Diffusion Map Clustering	zmanj. pomn.		twoclass_12000_train.csv		0,25	
Kmeans Clustering	0,01	zmanj. pomn.	twoclass_12000_train_wc.csv		0,23	
Feature Selection BAHSIC	zmanj. pomn.		twoclass_12000_train_wc.csv		0,23	
Nonparametric quantile estimation	zmanj. pomn.		twoclass_12000_train.csv	twoclass_12000_test_wc.csv	0,28	0,08

Tabela 7.4: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Epsilon SVM Regression	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,245395714	0,75273701
Standard Gaussian Regression	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,315373881	0,689134234
Laplacian SVM Regression	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test.csv	2,329382086	1,00212094
Nu SVM Regression	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,307617588	0,660064897
Gaussian Process Hetero Scedastic Regression	1,710603163	zmanj. pomn.	twoclass_100000_train.csv	twoclass_100000_test_wc.csv	0,002051099	0,669114091
SVM Binary Classification	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,254757797	0,69726739
Nu SVM Binary Classification	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,256708883	0,666825253
Gaussian Process Multiclass Classification	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,266224897	0,684710487
Gaussian Process Extreme Multiclass Classification	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,269433126	0,721660434
Diffusion Map Clustering	zmanj. pomn.		twoclass_100000_train.csv		2,201039518	
Kmeans Clustering	0,052547461	zmanj. pomn.	twoclass_100000_train_wc.csv		2,050396349	
Feature Selection BAHSIC	zmanj. pomn.		twoclass_100000_train_wc.csv		2,000271543	
Nonparametric quantile estimation	zmanj. pomn.		twoclass_100000_train.csv	twoclass_100000_test_wc.csv	2,253338064	0,722080041

Tabela 7.5: prikazuje rezultat testiranja z osnovno množico s po 100000 vzorci.

7.6.2 MDP

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
KMeans Classifier	0,00011	0,27	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005
Signum Classifier	0,00198		twoclass_1000_train.csv		0,011	
Perceptron Classifier	0,00016	0,13	twoclass_1000_train.csv	twoclass_1000_test.csv	0,012	0,005
Gaussian Classifier	0,00009	0,17	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005
KNN Classifier	0,00014	0,24	twoclass_1000_train.csv	twoclass_1000_test.csv	0,012	0,005
NearestMean Classifier	0,00008	0,17	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005
Linear Regression	0,00009	0,21	twoclass_1000_train.csv	twoclass_1000_test.csv	0,010	0,005
PCA	0,00009	0,17	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005
CuBICA	0,00009	0,20	twoclass_1000_train.csv	twoclass_1000_test.csv	0,010	0,004
TDSEP	0,00010	0,26	twoclass_1000_train.csv	twoclass_1000_test.csv	0,010	0,005
Independent Slow Feature Anal.	0,00009	0,23	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005

Tabela 7.6: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
KMeans Classifier	0,00014	0,43	twoclass_2000_train,csv	twoclass_2000_test,csv	0,022	0,009
Signum Classifier	0,00013	0,40	twoclass_2000_train,csv	twoclass_2000_test,csv	0,021	0,009
Perceptron Classifier	0,00014	0,40	twoclass_2000_train,csv	twoclass_2000_test,csv	0,022	0,009
Gaussian Classifier	0,00012	0,49	twoclass_2000_train,csv	twoclass_2000_test,csv	0,021	0,009
KNN Classifier	0,00013	0,41	twoclass_2000_train,csv	twoclass_2000_test,csv	0,022	0,010
NearestMean Classifier	0,00014	0,40	twoclass_2000_train,csv	twoclass_2000_test,csv	0,021	0,010
Linear Regression	0,00012	0,47	twoclass_2000_train,csv	twoclass_2000_test,csv	0,022	0,009
PCA	0,00013	0,26	twoclass_2000_train,csv	twoclass_2000_test,csv	0,021	0,009
CuBICA	0,00012	0,27	twoclass_2000_train,csv	twoclass_2000_test,csv	0,022	0,009
TDSEP	0,00014	0,41	twoclass_2000_train,csv	twoclass_2000_test,csv	0,021	0,010
Independent Slow Feature Anal.	0,00040	0,33	twoclass_2000_train,csv	twoclass_2000_test,csv	0,022	0,010

Tabela 7.7: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
KMeans Classifier	0,00029	1,22	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Signum Classifier	0,0098	1,05	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Perceptron Classifier	0,00034	1,06	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Gaussian Classifier	0,0019	1,46	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03
KNN Classifier	0,00031	1,25	twoclass_6000_train.csv	twoclass_6000_test.csv	0,08	0,03
NearestMean Classifier	0,00029	1,46	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03
Linear Regression	0,00027	0,99	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
PCA	0,00027	1,62	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
CuBICA	0,00027	1,00	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03
TDSEP	0,00031	1,45	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,08
Independent Slow Feature Anal.	0,00035	0,77	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03

Tabela 7.8: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
KMeans Classifier	0,0005	3,17	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
Signum Classifier	0,0192		twoclass_12000_train.csv		0,17	
Perceptron Classifier	0,0005	2,05	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
Gaussian Classifier	0,0005	2,01	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
KNN Classifier	0,0005	2,00	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
NearestMean Classifier	0,0005	1,99	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
Linear Regression	0,0005	2,61	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
PCA	0,0005	2,00	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
CuBICA	0,0008	2,93	twoclass_12000_train.csv	twoclass_12000_test.csv	0,15	0,07
TDSEP	0,0002	2,01	twoclass_12000_train.csv	twoclass_12000_test.csv	0,14	0,06
Independent Slow Feature Anal.	0,0005	2,45	twoclass_12000_train.csv	twoclass_12000_test.csv	0,14	0,06

Tabela 7.9: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
KMeans Classifier	0,005	19	twoclass_100000_train.csv	twoclass_100000_test.csv	1,19	0,52
Signum Classifier	0,168		twoclass_100000_train.csv		1,25	
Perceptron Classifier	0,008	24	twoclass_100000_train.csv	twoclass_100000_test.csv	1,27	0,62
Gaussian Classifier	0,005	20	twoclass_100000_train.csv	twoclass_100000_test.csv	1,26	0,51
KNN Classifier	0,005	28	twoclass_100000_train.csv	twoclass_100000_test.csv	1,28	0,54
NearestMean Classifier	0,005	28	twoclass_100000_train.csv	twoclass_100000_test.csv	1,20	0,54
Linear Regression	0,005	28	twoclass_100000_train.csv	twoclass_100000_test.csv	1,29	0,54
Principal Component Analysis	0,005	19	twoclass_100000_train.csv	twoclass_100000_test.csv	1,18	0,54
CuBICA	0,005	27	twoclass_100000_train.csv	twoclass_100000_test.csv	1,35	0,54
TDSEP	0,006	23	twoclass_100000_train.csv	twoclass_100000_test.csv	1,30	0,53
Independent Slow Feature Analysis	0,006	23	twoclass_100000_train.csv	twoclass_100000_test.csv	1,25	0,54

Tabela 7.10: prikazuje rezultat testiranja z osnovno množico s po 100000 vzorci.

7.6.3 OpenCV

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
K-Nearest Neighbors	0,00007	0,0043	twoclass_1000_train.csv	twoclass_1000_test.csv	0,012	0,006
Random Trees	0,0017	0,0008	twoclass_1000_train.csv	twoclass_1000_test.csv	0,016	0,006
Decision Trees	0,0005	0,0007	twoclass_1000_train.csv	twoclass_1000_test.csv	0,016	0,005
Gradient Boosted Trees	0,17	0,033	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005
Extremely randomized trees	0,03	0,006	twoclass_1000_train.csv	twoclass_1000_test.csv	0,010	0,005
Boosting	0,30	0,026	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005
Support Vector Machines	0,00	0,0005	twoclass_1000_train.csv	twoclass_1000_test.csv	0,012	0,005
Feed Forward Neural Network	10,19	0,0023	twoclass_1000_train.csv	twoclass_1000_test.csv	0,010	0,005
Kmeans Clustering	0,00008		twoclass_1000_train.csv		0,016	
Normal Bayes Classifier	0,03	0,0003	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005

Tabela 7.11: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
K-Nearest Neighbors	0,0001	0,012673678	twoclass_2000_train.csv	twoclass_2000_test.csv	0,021	0,011
Random Trees	0,0013	0,001476724	twoclass_2000_train.csv	twoclass_2000_test.csv	0,024	0,011
Decision Trees	0,0009	0,001484826	twoclass_2000_train.csv	twoclass_2000_test.csv	0,021	0,009
Gradient Boosted Trees	0,31	0,071378904	twoclass_2000_train.csv	twoclass_2000_test.csv	0,020	0,009
Extremely randomized trees	0,053	0,014065754	twoclass_2000_train.csv	twoclass_2000_test.csv	0,021	0,010
Boosting	0,46	0,049711905	twoclass_2000_train.csv	twoclass_2000_test.csv	0,021	0,010
Support Vector Machines	0,0047	0,001637638	twoclass_2000_train.csv	twoclass_2000_test.csv	0,021	0,009
Feed Forward Neural Network	20	0,005409626	twoclass_2000_train.csv	twoclass_2000_test.csv	0,021	0,009
Kmeans Clustering	0,000123		twoclass_2000_train.csv		0,021	
Normal Bayes Classifier	0,00012	0,000558171	twoclass_2000_train.csv	twoclass_2000_test.csv	0,021	0,009

Tabela 7.12: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
K-Nearest Neighbors	0,00	0,10	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03
Random Trees	0,004	0,005	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Decision Trees	0,003	0,004	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Gradient Boosted Trees	1,05	0,24	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Extremely randomized trees	0,15	0,05	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Boosting	1,69	0,20	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Support Vector Machines	0,03	0,00	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03
Feed Forward Neural Network	60,82	0,01	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03
Kmeans Clustering	0,0002		twoclass_6000_train.csv		0,06	
Normal Bayes Classifier	0,0003	0,0014	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03

Tabela 7.13: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
K-Nearest Neighbors	0,00009	0,41	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
Random Trees	0,01	0,01	twoclass_12000_train.csv	twoclass_12000_test.csv	0,15	0,05
Decision Trees	0,01	0,01	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
Gradient Boosted Trees	2,38	0,59	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
Extremely randomized trees	0,32	0,10	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
Boosting	4,09	0,45	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
Support Vector Machines	0,07	0,01	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
Feed Forward Neural Network	118,66	0,03	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
Kmeans Clustering	0,0007	0,0007	twoclass_12000_train.csv		0,13	
Normal Bayes Classifier	0,0005	0,0033	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05

Tabela 7.14: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
K-Nearest Neighbors	0,0012	27,82	twoclass_100000_train.csv	twoclass_100000_test.csv	1,07	0,46
Random Trees	0,09	0,09	twoclass_100000_train.csv	twoclass_100000_test.csv	1,06	0,48
Decision Trees	0,09	0,08	twoclass_100000_train.csv	twoclass_100000_test.csv	1,10	0,50
Gradient Boosted Trees	27,49	5,98	twoclass_100000_train.csv	twoclass_100000_test.csv	1,13	0,49
Extremely randomized trees	2,74	0,94	twoclass_100000_train.csv	twoclass_100000_test.csv	1,14	0,49
Boosting	3,92	1,58	twoclass_100000_train.csv	twoclass_100000_test.csv	1,06	0,46
Support Vector Machines	0,50	0,08	twoclass_100000_train.csv	twoclass_100000_test.csv	1,07	0,46
Feed Forward Neural Network	1000,82	0,23	twoclass_100000_train.csv	twoclass_100000_test.csv	1,06	0,46
Kmeans Clustering	0,0031		twoclass_100000_train.csv		1,10	
Normal Bayes Classifier	0,0046	0,03	twoclass_100000_train.csv	twoclass_100000_test.csv	1,07	0,46

Tabela 7.15: prikazuje rezultat testiranja z osnovno množico s po 100000 vzorci.

7.6.4 Orange

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Naive Bayes classifier	0,02	0,0005	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Decision Trees	0,01	0,004	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Nearest Neighbour	0,0002	0,04	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Majority	0,00003	0,0001	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
CrossValidation Evaluation	0,13		twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	
Linear Regression	0,004	0,01	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Associate Rule	0,0004		twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	
Feature Selection	0,01		twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,005	
Compare Feature Selection	0,40		twoclass_str_1000_train.csv		0,004	
Bagging	0,02	0,43	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,006	0,002
Support Vector Machines	0,26	0,0023	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,005	0,002
Rule Induction	0,05		twoclass_str_1000_train.csv		0,004	
Logistic Regression	0,0015	0,001	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Lasso Regression	0,53	0,0005	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
PLS Regression	0,52	0,0005	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Earth Regression	0,08	0,13	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,005	0,002
Regression Trees	0,01	0,00	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Simple Regression Trees	0,0012	0,0002	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Mean Regression	0,00004	0,00012	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
Multilabel Binary Relevance	0,04	0,001	twoclass_01_1000_train.tab	twoclass_01_1000_test.tab	0,006	0,003
Label Powerset Classification	0,04	0,001	twoclass_01_1000_train.tab	twoclass_01_1000_test.tab	0,006	0,002
MultikNN Classification	0,46	0,001	twoclass_01_1000_train.tab	twoclass_01_1000_test.tab	0,006	0,003
BR-kNN Classification	0,45	0,001	twoclass_01_1000_train.tab	twoclass_01_1000_test.tab	0,006	0,002
KMeans Clustering	0,01		twoclass_str_1000_train.csv		0,006	
Hierarchical Clustering	0,37		twoclass_str_1000_train.csv		0,006	
Linear Projection PCA	0,003		twoclass_str_1000_train.csv		0,004	
Unsupervised Self-organizing maps	3,36		twoclass_str_1000_train.csv		0,004	
Self-organizing maps	0,35	0,06	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,004	0,002
FreeVizLearner	0,16	0,00	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,005	0,002

Tabela 7.16: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Naive Bayes classifier	0,043	0,001	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,004
Decision Trees	0,023	0,01	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,004
Nearest Neighbour	0,001	0,19	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,004
Majority	0,0001	0,0003	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,005
CrossValidation Evaluation	0,279		twoclass_str_2000_train.csv		0,016	
Linear Regression	0,009	0,01	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,010	0,004
Associate Rule	0,001		twoclass_str_2000_train.csv		0,008	
Feature Selection	0,014		twoclass_str_2000_train.csv		0,010	
Compare Feature Selection	0,772		twoclass_str_2000_train.csv		0,010	
Bagging	0,027	2,04	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,004

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Support Vector Machines	1,082	0,01	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,012	0,003
Rule Induction	0,132		twoclass_str_2000_train.csv		0,009	
Logistic Regression	0,004	0,002	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,004
Lasso Regression	0,822	0,001	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,008	0,004
PLS Regression	0,837	0,001	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,003
Earth Regression	0,557	0,26	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,004
Regression Trees	0,021	0,02	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,008	0,004
Simple Regression Trees	0,005	0,000	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,008	0,004
Mean Regression	0,0002	0,000	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,004
Multilabel Binary Relevance	0,080	0,002	twoclass_01_2000_train.tab	twoclass_01_2000_test.tab	0,013	0,005
Label Powerset Classification	0,098	0,002	twoclass_01_2000_train.tab	twoclass_01_2000_test.tab	0,013	0,005
MultikNN Classification	1,879	0,005	twoclass_01_2000_train.tab	twoclass_01_2000_test.tab	0,012	0,006
BR-kNN Classification	2,152	0,002	twoclass_01_2000_train.tab	twoclass_01_2000_test.tab	0,026	0,011
KMeans Clustering	0,015		twoclass_str_2000_train.csv		0,012	
Hierarchical Clustering	1,629		twoclass_str_2000_train.csv		0,014	
Linear Projection PCA	0,004		twoclass_str_2000_train.csv		0,010	
Unsupervised Self-organizing maps	6,063		twoclass_str_2000_train.csv		0,009	
Self-organizing maps	0,565	0,11	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,009	0,005
FreeVizLearner	0,557	0,005	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,017	0,005

Tabela 7.17: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Naive Bayes classifier	0,10	0,00	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,04	0,02
Decision Trees	0,10	0,13	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Nearest Neighbour	0,0026	1,88	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Majority	0,0001	0,001	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
CrossValidation Evaluation	1,00		twoclass_str_6000_train.csv		0,03	
Linear Regression	0,06	0,03	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,05	0,01
Associate Rule	0,001		twoclass_str_6000_train.csv		0,03	
Feature Selection	0,05		twoclass_str_6000_train.csv		0,03	
Compare Feature Selection	2,49		twoclass_str_6000_train.csv		0,03	
Bagging	0,09	19,74	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Support Vector Machines	10,88	0,05	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Rule Induction	0,49		twoclass_str_6000_train.csv		0,03	
Logistic Regression	0,01	0,01	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Lasso Regression	2,21	0,003	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
PLS Regression	2,49	0,01	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Earth Regression	5,65	0,87	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Regression Trees	0,15	0,18	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,05	0,02
Simple Regression Trees	0,02	0,001	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Mean Regression	0,0003	0,001	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
Multilabel Binary Relevance	0,26	0,01	twoclass_01_6000_train.tab	twoclass_01_6000_test.tab	0,04	0,02
Label Powerset Classification	0,32	0,01	twoclass_01_6000_train.tab	twoclass_01_6000_test.tab	0,04	0,02
MultikNN Classification	21,48	0,01	twoclass_01_6000_train.tab	twoclass_01_6000_test.tab	0,04	0,02
BR-kNN Classification	21,60	0,01	twoclass_01_6000_train.tab	twoclass_01_6000_test.tab	0,05	0,02
KMeans Clustering	0,07		twoclass_str_6000_train.csv		0,05	
Hierarchical Clustering	15,02		twoclass_str_6000_train.csv		0,04	
Linear Projection PCA	0,01		twoclass_str_6000_train.csv		0,03	
Unsupervised Self-organizing maps	16,68		twoclass_str_6000_train.csv		0,03	
Self-organizing maps	1,33	0,33	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01
FreeVizLearner	4,94	0,03	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,01

Tabela 7.18: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Naive Bayes classifier	0,215	0,01	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,06	0,02
Decision Trees	0,30	0,53	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,06	0,02
Nearest Neighbour	0,003	7,54	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Majority	0,0003	0,001	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,06	0,02

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
CrossValidation Evaluation	2,39		twoclass_str_12000_train.csv		0,06	
Linear Regression	0,21	0,06	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,06	0,02
Associate Rule	0,003		twoclass_str_12000_train.csv		0,05	
Feature Selection	0,11		twoclass_str_12000_train.csv		0,06	
Compare Feature Selection	5,08		twoclass_str_12000_train.csv		0,06	
Bagging	0,19	79,80	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Support Vector Machines	40,57	0,14	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Rule Induction	1,64		twoclass_str_12000_train.csv		0,06	
Logistic Regression	0,02	0,01	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Lasso Regression	5,86	0,01	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
PLS Regression	3,58	0,01	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Earth Regression	26,45	1,60	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Regression Trees	0,29	0,58	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,06	0,02
Simple Regression Trees	0,06	0,002	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Mean Regression	0,001	0,001	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
Multilabel Binary Relevance	0,54	0,01	twoclass_01_12000_train.tab	twoclass_01_12000_test.tab	0,07	0,03
Label Powerset Classification	0,72	0,01	twoclass_01_12000_train.tab	twoclass_01_12000_test.tab	0,07	0,03
MultikNN Classification	82,77	0,01	twoclass_01_12000_train.tab	twoclass_01_12000_test.tab	0,09	0,03
BR-kNN Classification	85,71	0,01	twoclass_01_12000_train.tab	twoclass_01_12000_test.tab	0,09	0,03
KMeans Clustering	0,08		twoclass_str_12000_train.csv		0,07	
Hierarchical Clustering	63,76		twoclass_str_12000_train.csv		0,08	
Linear Projection PCA	0,02		twoclass_str_12000_train.csv		0,05	
Unsupervised Self-organizing maps	28,04		twoclass_str_12000_train.csv		0,05	
Self-organizing maps	2,54	0,69	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02
FreeVizLearner	19,75	0,08	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,05	0,02

Tabela 7.19: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Naive Bayes classifier	**2,35**	0,04	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,54	0,22
Decision Trees	5,77	27,09	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,48	0,21
Nearest Neighbour	0,04	641,36	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,61	0,22
Majority	0,002	0,01	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,48	0,20
CrossValidation Evaluation	30,83		twoclass_str_100000_train.csv		0,55	
Linear Regression	zman. pomn.		twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,48	0,19
Associate Rule	0,02		twoclass_str_100000_train.csv		0,52	
Feature Selection	1,11		twoclass_str_100000_train.csv		0,61	
Compare Feature Selection	63,37		twoclass_str_100000_train.csv		0,48	
Bagging	3,11	5931,86	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,73	0,23
Support Vector Machines	zman. pomn.		twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,48	0,21
Rule Induction	54,99		twoclass_str_100000_train.csv		0,58	
Logistic Regression	0,70	0,09	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,54	0,20
Lasso Regression	100,65	0,05	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,54	0,20
PLS Regression	87,32	0,04	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	1,34	0,29
Earth Regression	zman. pomn.		twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,50	0,21
Regression Trees	5,44	35,24	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,49	0,21
Simple Regression Trees	1,28	0,02	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,50	0,21
Mean Regression	0,002	0,01	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,48	0,20
Multilabel Binary Relevance	5,20	0,09	twoclass_01_100000_train.tab	twoclass_01_100000_test.tab	0,86	0,26
Label Powerset Classification	7,25	0,09	twoclass_01_100000_train.tab	twoclass_01_100000_test.tab	0,82	0,27
MultikNN Classification	8196,22	0,09	twoclass_01_100000_train.tab	twoclass_01_100000_test.tab	1,07	0,26
BR-kNN Classification	7644,61	0,09	twoclass_01_100000_train.tab	twoclass_01_100000_test.tab	0,59	0,26
KMeans Clustering	0,68		twoclass_str_100000_train.csv		0,60	
Hierarchical Clustering	zman. pomn.		twoclass_str_100000_train.csv		0,61	
Linear Projection PCA	0,14		twoclass_str_100000_train.csv		0,49	
Unsupervised Self-organizing maps	945,66		twoclass_str_100000_train.csv		0,49	
Self-organizing maps	20,95	5,65	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,53	0,24
FreeVizLearner	653,51	4,94	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,51	0,21

Tabela 7.20: prikazuje rezultat testiranja z osnovno množico s po 100000 vzorci.

7.6.5 PyBrain

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Feed Forward Network	303,80	0,04	twoclass_1000_train.csv	twoclass_1000_test.csv	0,012	0,005
Recurrent Network	294,84	0,02	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,005
Feed Forward Network Classification	158,39	0,02	twoclass_str_1000_train.csv	twoclass_str_1000_test.csv	0,006	0,003
Evolino Network	1,44	0,09	twoclass_1000_train.csv	twoclass_1000_test.csv	0,011	0,006
K-Means	9,55		twoclass_1000_train.csv		0,012	

Tabela 7.21: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Feed Forward Network	609,19	0,07	twoclass_2000_train.csv	twoclass_2000_test.csv	0,022	0,009
Recurrent Network	591,16	0,05	twoclass_2000_train.csv	twoclass_2000_test.csv	0,022	0,010
Feed Forward Network Classification	317,95	0,03	twoclass_str_2000_train.csv	twoclass_str_2000_test.csv	0,011	0,005
Evolino Network	2,45	0,19	twoclass_2000_train.csv	twoclass_2000_test.csv	0,022	0,010
K-Means	36,90		twoclass_2000_train.csv		0,021	

Tabela 7.22: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Feed Forward Network	1821,48	0,22	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Recurrent Network	1780,91	0,13	twoclass_6000_train.csv	twoclass_6000_test.csv	0,06	0,03
Feed Forward Network Classification	961,35	0,11	twoclass_str_6000_train.csv	twoclass_str_6000_test.csv	0,03	0,02
Evolino network	9,12	0,62	twoclass_6000_train.csv	twoclass_6000_test.csv	0,07	0,03
K-Means	329,05		twoclass_6000_train.csv		0,07	

Tabela 7.23: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Feed Forward Network	3644,95	0,41	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
Recurrent Network	3566,54	0,27	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,05
Feed Forward Network Classification	1907,26	0,20	twoclass_str_12000_train.csv	twoclass_str_12000_test.csv	0,07	0,03
Evolino network	22,38	1,24	twoclass_12000_train.csv	twoclass_12000_test.csv	0,13	0,06
K-Means	1280,25		twoclass_12000_train.csv		0,13	

Tabela 7.24: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Feed Forward Network	23801,14	3,45	twoclass_100000_train.csv	twoclass_100000_test.csv	1,10	0,46
Recurrent Network	23300,06	2,24	twoclass_100000_train.csv	twoclass_100000_test.csv	1,09	0,46
Feed Forward Network Classification	11770,89	1,79	twoclass_str_100000_train.csv	twoclass_str_100000_test.csv	0,55	0,24
Evolino network	zman. pom.	10,95	twoclass_100000_train.csv	twoclass_100000_test.csv	1,15	0,50
K-Means	72065,75		twoclass_100000_train.csv		1,07	

Tabela 7.25: prikazuje rezultat testiranja z osnovno množico s po 100000 vzorci.

7.6.6 PyML

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Support Vector Machines	0,02	0,003	twoclass_1000_train.csv	twoclass_1000_test.csv	0,02	0,01
Support Vector Regression	0,12	0,002	twoclass_1000_train.csv	twoclass_1000_test.csv	0,03	0,02
K-nearest Neighbor Classifier	0,0001	0,04	twoclass_1000_train.csv	twoclass_1000_test.csv	0,04	0,01
Ridge Regression Classifier	2,50	1,23	twoclass_1000_train.csv	twoclass_1000_test.csv	0,02	0,01

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
One Against the Rest Classifier	0,88	0,02	multiclass_1000_train.csv	multiclass_1000_test.csv	0,04	0,02
One Against the One Classifier	0,02	0,04	multiclass_1000_train.csv	multiclass_1000_test.csv	0,05	0,02
Model Selection SVM (C param)	1,24		twoclass_1000_train.csv		0,03	
Recursive Feature Elimination	0,01		twoclass_1000_train.csv		0,02	
Feature selection Filter	0,03		twoclass_1000_train.csv		0,06	
Feature selection Random	0,0002		twoclass_1000_train.csv		0,02	
Chain Operation Classifier	0,01	0,004	twoclass_1000_train.csv	twoclass_1000_test.csv	0,02	0,02
Preprocessing Standardizer	0,004		twoclass_1000_train.csv		0,02	

Tabela 7.26: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Support Vector Machines	0,01	0,01	twoclass_2000_train.csv	twoclass_2000_test.csv	0,05	0,02
Support Vector Regression	0,46	0,004	twoclass_2000_test.csv	twoclass_2000_test.csv	0,06	0,02
K-nearest Neighbor Classifier	0,0001	0,01	twoclass_2000_train.csv	twoclass_2000_test.csv	0,06	0,02
Ridge Regression Classifier	9,75	4,90	twoclass_2000_train.csv	twoclass_2000_test.csv	0,04	0,02
One Against the Rest Classifier	6,31	0,03	multiclass_2000_train.csv	multiclass_2000_test.csv	0,09	0,03
One Against the One Classifier	0,08	0,08	multiclass_2000_train.csv	multiclass_2000_test.csv	0,09	0,04
Model Selection SVM (C param)	2,66		twoclass_2000_train.csv		0,05	
Recursive Feature Elimination	0,003		twoclass_2000_train.csv		0,05	
Feature selection Filter	0,01		twoclass_2000_train.csv		0,05	
Feature selection Random	0,0003		twoclass_2000_train.csv		0,04	
Chain Operation Classifier	0,05	0,01	twoclass_2000_train.csv	twoclass_2000_test.csv	0,05	0,03
Preprocessing Standardizer	0,01		twoclass_2000_train.csv		0,05	

Tabela 7.27: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Support Vector Machines	0,12	0,02	twoclass_6000_train.csv	twoclass_6000_test.csv	0,12	0,06
Support Vector Regression	1,95	0,01	twoclass_6000_test.csv	twoclass_6000_test.csv	0,19	0,09
K-nearest Neighbor Classifier	0,00	0,33	twoclass_6000_train.csv	twoclass_6000_test.csv	0,13	0,06
Ridge Regression Classifier	188,61	46,02	twoclass_6000_train.csv	twoclass_6000_test.csv	0,13	0,05
One Against the Rest Classifier	88,82	0,10	multiclass_6000_train.csv	multiclass_6000_test.csv	0,27	0,11
One Against the One Classifier	0,39	0,24	multiclass_6000_train.csv	multiclass_6000_test.csv	0,26	0,11
Model Selection SVM (C param)	11,18		twoclass_6000_train.csv		0,14	
Recursive Feature Elimination	0,01		twoclass_6000_train.csv		0,14	
Feature selection Filter	0,02		twoclass_6000_train.csv		0,13	
Feature selection Random	0,001		twoclass_6000_train.csv		0,13	
Chain Operation Classifier	0,35	0,02	twoclass_6000_train.csv	twoclass_6000_test.csv	0,13	0,06
Preprocessing Standardizer	0,02		twoclass_6000_train.csv		0,18	

Tabela 7.28: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Support Vector Machines	0,54	0,03	twoclass_12000_train.csv	twoclass_12000_test.csv	0,26	0,13
Support Vector Regression	8,17	0,02	twoclass_12000_test.csv	twoclass_12000_test.csv	0,34	0,15
K-nearest Neighbor Classifier	0,0001	1,40	twoclass_12000_train.csv	twoclass_12000_test.csv	0,27	0,12
Ridge Regression Classifier	zman. pomn.		twoclass_12000_train.csv	twoclass_12000_test.csv	0,29	0,11
One Against the Rest Classifier	246,55	0,60	multiclass_12000_train.csv	multiclass_12000_test.csv	0,48	0,20
One Against the One Classifier	4,36	1,03	multiclass_12000_train.csv	multiclass_12000_test.csv	0,92	0,40
Model Selection SVM (C param)	80,57		twoclass_12000_train.csv		0,31	
Recursive Feature Elimination	0,03		twoclass_12000_train.csv		0,59	
Feature selection Filter	0,09		twoclass_12000_train.csv		0,56	
Feature selection Random	0,001		twoclass_12000_train.csv		0,63	
Chain Operation Classifier	3,97	0,11	twoclass_12000_train.csv	twoclass_12000_test.csv	0,63	0,26
Preprocessing Standardizer	0,10		twoclass_12000_train.csv		0,58	

Tabela 7.29: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Support Vector Machines	56,08	0,41	twoclass_100000_train.csv	twoclass_100000_test.csv	3,18	1,95
Support Vector Regression	415,13	0,15	twoclass_100000_test.csv	twoclass_100000_test.csv	2,87	1,33
K-nearest Neighbor Classifier	0,06	92,47	twoclass_100000_train.csv	twoclass_100000_test.csv	2,15	0,87
Ridge Regression Classifier	zman. pomn.		twoclass_100000_train.csv	twoclass_100000_test.csv	2,10	0,91
One Against the Rest Classifier	4592,60	2,27	multiclass_100000_train.csv	multiclass_100000_test.csv	4,83	1,74
One Against the One Classifier	111,89	4,26	multiclass_100000_train.csv	multiclass_100000_test.csv	4,40	1,83
Model Selection SVM (C param)	13281,05		twoclass_100000_train.csv		2,24	
Recursive Feature Elimination	0,13		twoclass_100000_train.csv		2,21	
Feature selection Filter	0,36		twoclass_100000_train.csv		2,13	
Feature selection Random	0,01		twoclass_100000_train.csv		2,12	
Chain Operation Classifier	110,79	0,32	twoclass_100000_train.csv	twoclass_100000_test.csv	2,09	0,90
Preprocessing Standardizer	0,48		twoclass_100000_train.csv		2,19	

Tabela 7.30: prikazuje rezultat testiranja z osnovno množico s po 100000 vzorci.

7.6.7 Shogun

Shogun knjižnica ima implementirana dva programska vmesnika zato smo naredili test za oba vmesnika. Vsi vhodni podatki algoritmov v knjižnici Shogun potrebujejo še dodatno vhodno datoteko z imenom, ki se konča na *_labels.csv in ki ni definirana v spodnjih tabelah. To je datoteka, ki vsebuje rezultate razreda učne datoteke.

Statični vmesnik

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
LIBSVM	0,08	0,03	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,002	0,0004
GMNPSVM	0,15	0,04	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0003
GPBTSVM	0,09	0,02	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
LIBSVM_MULTICLASS	0,07	0,02	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0003
LIBSVM_ONECLASS	0,04	0,0003	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
MPDSVM	0,38	0,03	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0002
KNN	0,0001	0,04	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0003
PERCEPTRON	ni konvergiral		twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
HIERARCHICAL	0,000006		twoclass_sp_1000_train_wc.csv		0,001	
KMEANS	0,000001		twoclass_sp_1000_train_wc.csv		0,001	
LIBSVR	0,15	0,03	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,002	0,0006
SVRLIGHT	0,29	0,03	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0006

Tabela 7.31: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
LIBSVM	0,51	0,11	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,006	0,001
GMNPSVM	0,77	0,14	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,000
GPBTSVM	0,61	0,12	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001
LIBSVM_MULTICLASS	0,30	0,11	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001
LIBSVM_ONECLASS	0,17	0,00	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001
MPDSVM	2,03	0,12	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001
KNN	0,00	0,13	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001
PERCEPTRON	ni konvergiral	0,00	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001
HIERARCHICAL	0,000002		twoclass_sp_2000_train_wc.csv		0,001	
KMEANS	0,000001		twoclass_sp_2000_train_wc.csv		0,001	
LIBSVR	0,34	0,17	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
SVRLIGHT	1,16	0,12	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001

Tabela 7.32: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
LIBSVM	3,06	1,02	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
GMNPSVM	7,44	1,04	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,005	0,002
GPBTSVM	3,18	0,92	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
LIBSVM_MULTICLASS	2,62	0,93	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
LIBSVM_ONECLASS	2,31	0,00	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,004	0,002
MPDSVM	74,56	0,93	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,005	0,002
KNN	0,0003	1,22	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,007	0,002
PERCEPTRON	ni konvergiral		twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
HIERARCHICAL	0,000001		twoclass_sp_6000_train_wc.csv		0,004	
KMEANS	0,000001		twoclass_sp_6000_train_wc.csv		0,004	
LIBSVR	2,85	0,93	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
SVRLIGHT	14,16	1,00	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002

Tabela 7.33: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
LIBSVM	10,55	3,74	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,003
GMNPSVM	25,79	4,27	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,013	0,004
GPBTSVM	11,14	3,67	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,011	0,004
LIBSVM_MULTICLASS	10,40	3,78	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,004
LIBSVM_ONECLASS	9,39	0,00	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,008	0,004
MPDSVM	395,28	3,67	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,011	0,004
KNN	0,0004	4,75	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,011	0,004
PERCEPTRON	ni konvergiral		twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,011	0,004
HIERARCHICAL	0,000001		twoclass_sp_12000_train_wc.csv		0,056	
KMEANS	0,000001		twoclass_sp_12000_train_wc.csv		0,009	
LIBSVR	11,65	3,74	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,004
SVRLIGHT	62,33	3,75	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,004

Tabela 7.34: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
LIBSVM	739,42	258,61	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,104	0,03
GMNPSVM	713,99	151,52	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,095	0,03
GPBTSVM	774,64	257,53	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,095	0,03
LIBSVM_MULTICLASS	4897,46	3436,78	multiclass_sp_100000_train_wc.csv	multiclass_sp_100000_test_wc.csv	0,215	0,071
LIBSVM_ONECLASS	839,62	0,01	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,070	0,03
MPDSVM	63933,77	256,72	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,099	0,03
KNN	0,03	365,97	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,094	0,029
PERCEPTRON	ni konvergiral		twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,094	0,03
HIERARCHICAL	zman. pomn.		twoclass_sp_100000_train_wc.csv			
KMEANS	0,000001		twoclass_sp_100000_train_wc.csv		0,070	
LIBSVR	1030,03	280,51	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,095	0,030
SVRLIGHT	11451,33	297,77	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,094	0,0295

Tabela 7.35: prikazuje rezultat testiranja z osnovno množico s po 100000 vzorci.

Modularni vmesnik

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
AveragedPerceptron	0,05	0,0001	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
GaussianNaiveBayes	0,0001	0,0002	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
GMNPSVM	0,14	0,04	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
GPBTSVM	0,08	0,05	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
KNN	0,00	0,06	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,002	0,0004
larank	0,08	0,11	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
LibSVM	0,08	0,06	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0001
LibSVMMultiClass	0,40	0,64	multiclass_sp_1000_train_wc.csv	multiclass_sp_1000_test_wc.csv	0,002	0,0008
LibSVMOneClass	0,05	0,03	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
Mpdsvm	0,16	0,05	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
Subgradientsvm	0,55	0,10	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,080	0,0286
Svmlin	0,0004	0,0002	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,002	0,0001
Svmocas	0,0008	0,0002	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
Svmsgd	0,0004	0,0002	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0005
Hierarchical	0,05		twoclass_sp_1000_train_wc.csv		0,001	
KMeans	0,06		twoclass_sp_1000_train_wc.csv		0,001	
MKLBin	0,32	0,02	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0003
MKLMulti	1065,90	0,67	multiclass_sp_1000_train_wc.csv	multiclass_sp_1000_test_wc.csv	0,002	0,0009
LibSVR	0,09	0,05	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,001	0,0004
SVRLight	0,22	0,05	twoclass_sp_1000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,002	0,0006

Tabela 7.36: prikazuje rezultat testiranja z osnovno množico s po 1000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
AveragedPerceptron	0,09	0,0003	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,004	0,001
GaussianNaiveBayes	0,0001	0,0003	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
GMNPSVM	0,69	0,14	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
GPBTSVM	0,34	0,22	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
KNN	0,0001	0,30	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
larank	0,30	0,38	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,004	0,001
LibSVM	0,35	0,21	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,004	0,001
LibSVMMultiClass	2,14	2,53	multiclass_sp_2000_train_wc.csv	multiclass_sp_2000_test_wc.csv	0,007	0,002
LibSVMOneClass	0,28	0,11	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,002	0,001
Mpdsvm	1,32	0,21	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
Subgradientsvm	0,31	0,0003	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,004	0,001
Svmlin	0,0008	0,0005	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,002
Svmocas	0,0003	0,0027	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
Svmsgd	0,0006	0,0004	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
Hierarchical	0,23		twoclass_sp_2000_train_wc.csv		0,002	
KMeans	0,23		twoclass_sp_2000_train_wc.csv		0,003	
MKLBin	1,05	0,10	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,004	0,001
LibSVR	0,33	0,22	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001
SVRLight	0,92	0,21	twoclass_sp_2000_train_wc.csv	twoclass_sp_2000_test_wc.csv	0,003	0,001

Tabela 7.37: prikazuje rezultat testiranja z osnovno množico s po 2000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
AveragedPerceptron	0,31	0,002	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,007	0,002
GaussianNaiveBayes	0,0004	0,001	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
GMNPSVM	7,80	1,14	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
GPBTSVM	2,84	1,93	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
KNN	0,00	2,20	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,003
larank	8,83	3,78	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,006	0,002
LibSVM	2,93	2,15	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,008	0,002
LibSVMMultiClass	16,89	25,83	multiclass_sp_6000_train_wc.csv	multiclass_sp_6000_test_wc.csv	0,014	0,005
LibSVMOneClass	3,06	1,10	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,005	0,002
Mpdsvm	63,80	2,05	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,007	0,003
Subgradientsvm	0,29	0,002	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,008	0,003
Svmlin	0,003	0,003	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,007	0,002
Svmocas	0,001	0,003	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,007	0,002
Svmsgd	0,002	0,005	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,009	0,002
Hierarchical	2,51		twoclass_sp_6000_train_wc.csv		0,005	
KMeans	2,60		twoclass_sp_6000_train_wc.csv		0,005	
MKLBin	60,90	1,31	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,008	0,003
LibSVR	2,97	1,88	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,007	0,002
SVRLight	14,28	1,98	twoclass_sp_6000_train_wc.csv	twoclass_sp_6000_test_wc.csv	0,009	0,003

Tabela 7.38: prikazuje rezultat testiranja z osnovno množico s po 6000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
AveragedPerceptron	0,60	0,01	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,003
GaussianNaiveBayes	0,001	0,002	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,004
GMNPSVM	27,45	4,38	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,003
GPBTSVM	10,87	7,42	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,014	0,004
KNN	0,0003	9,13	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,004
larank	32,64	13,29	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,003
LibSVM	10,34	7,43	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,011	0,004
LibSVMMultiClass	62,19	91,14	multiclass_sp_12000_train_wc.csv	multiclass_sp_12000_test_wc.csv	0,027	0,008
LibSVMOneClass	11,69	3,82	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,009	0,004
Mpdsvm	193,94	7,31	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,003
Subgradientsvm	0,46	0,19	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,369	0,035
Svmlin	0,004	0,012	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,015	0,004
Svmocas	0,001	0,013	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,013	0,004
Svmsgd	0,004	0,012	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,013	0,004
Hierarchical	9,59		twoclass_sp_12000_train_wc.csv		0,069	
KMeans	9,64		twoclass_sp_12000_train_wc.csv		0,009	
MKLBin	199,82	4,37	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,025	0,004
LibSVR	11,99	7,54	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,012	0,025
SVRLight	57,02	7,51	twoclass_sp_12000_train_wc.csv	twoclass_sp_12000_test_wc.csv	0,018	0,006

Tabela 7.39: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.

Algoritem	čas učenja	čas testiranja	učna datoteka	testna datoteka	čas nalaganja učne dat.	čas nalaganja testne dat.
AveragedPerceptron	4,60	0,39	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,18	0,05
GaussianNaiveBayes	0,01	0,06	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,16	0,05
GMNPSVM	790,32	170,19	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,15	0,05
GPBTSVM	771,41	522,38	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,15	0,05
KNN	0,001	719,01	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,18	0,05
larank	2282,62	917,22	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,15	0,05
LibSVM	1391,84	523,25	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,15	0,05
LibSVMMultiClass	8484,98	6460,12	multiclass_sp_100000_train_wc.csv	multiclass_sp_100000_test_wc.csv	0,34	0,11
LibSVMOneClass	1260,49	267,49	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,16	0,09
Mpdsvm	48707,75	523,33	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,16	0,05
Subgradientsvm	0,46	0,38	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,28	0,11
Svmlin	0,04	0,76	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,16	0,06
Svmocas	0,01	0,79	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,15	0,05
Svmsgd	0,03	0,79	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,15	0,05
Hierarchical	zman. pomn.		twoclass_sp_100000_train_wc.csv		0,15	
KMeans	zman. pomn.		twoclass_sp_100000_train_wc.csv		0,07	
MKLBin	zman. pomn.		twoclass_sp_100000_train_wc.csv	twoclass_sp_1000_test_wc.csv	0,11	0,03
LibSVR	962,69	519,91	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,10	0,03
SVRLight	6311,87	567,48	twoclass_sp_100000_train_wc.csv	twoclass_sp_100000_test_wc.csv	0,15	0,05

Tabela 7.40: prikazuje rezultat testiranja z osnovno množico s po 12000 vzorci.