**Универзитет у Крагујевцу**

**Факултет техничких наука**

**Чачак**



**ДИПЛОМСКИ РАД**

**примена алгоритама машинског учења у програмском језику python у категоризацији земљотреса**

|  |  |  |
| --- | --- | --- |
| Ментор |  | Кандидат |
|  |  |  |
| др Урош Пешовић |  | Борис Михајловић 294/2015 |

У Чачку,

2019. године

# САДРЖАЈ

[1 УВОД 1](#_Toc19229788)

[2 МАШИНСКО УЧЕЊЕ И ИСТРАЖИВАЊЕ ПОДАТАКА 3](#_Toc19229789)

[2.1 Машинско учење 3](#_Toc19229790)

[2.2 Истраживање података 4](#_Toc19229791)

[2.3 Поступак креирања пројекта у области истраживања података 5](#_Toc19229792)

[2.4 Python библиотеке за машинско учење 7](#_Toc19229793)

[3 ПОДАЦИ ЗА КЛАСИФИКАЦИЈУ 8](#_Toc19229794)

[3.1 Земљотреси у периоду од 1965. до 2016. године 8](#_Toc19229795)

[3.2 Претпроцесирање табеле земљотреса 9](#_Toc19229796)

[4 АЛГОРИТМИ КЛАСИФИКАЦИЈЕ 10](#_Toc19229797)

[4.1 kNN aлгоритам класификације 10](#_Toc19229798)

[4.2 Стабла одлучивања 11](#_Toc19229799)

[4.3 SVM алгоритам 12](#_Toc19229800)

[5 ПРАКТИЧНА РЕАЛИЗАЦИЈА 15](#_Toc19229801)

[5.1 Реализација kNN алгоритма 15](#_Toc19229802)

[5.2 Реализација алгоритма стабла одлучивања 18](#_Toc19229803)

[5.3 Реализација SVM алгоритма 24](#_Toc19229804)

[6 МОГУЋНОСТ ПАРАЛЕЛИЗАЦИЈЕ АЛГОРИТАМА МАШИНСКОГ УЧЕЊА 27](#_Toc19229805)

[6.1 PyCOMPS програмски модел 27](#_Toc19229806)

[6.2 Пример паралелизације 28](#_Toc19229807)

[7 ЗАКЉУЧАК 32](#_Toc19229808)

[ЛИТЕРАТУРА 33](#_Toc19229809)

# УВОД

Основна тема овог дипломског рада је истраживање података (енгл. *data mining*), са фокусом на тему класификације земљотреса у периоду од 1965. до 2016. године. Употребљени су најчешћи алгоритми класификације у језику *Python*, чији су параметри мењани како би се направио што бољи модел за класификацију земљотреса. Детаљно је објашњен принцип рада за три алгоритма класификације, као и њихова имплементација на скупу података, односно табели са земљотресима. У раду су коришћене неке од најпопуларнијих библиотека програмског језика *Python*, као нпр. *pandas*, *numpy* и *scikit-learn,* које садрже методе које омогућују истраживање података. Истраживање података представља област науке машинског учења (енгл. *machine learning)* која доживљава експанзију у претходних неколико година због корисних примена у различитим аспектима науке и технике. Као основно радно окружење коришћен је *Jupyter Notebook* који омогућава развој софтвера на више програмских језика. У овом раду је обрађена и примена паралелизма у истраживању података у радном окружењу специјално намењеном за те сврхе.

У другом поглављу овог рада су приказане теоријске основе машинског учења и истраживања података. Описан је поступак креирања пројекта у области истраживања података, као и *Python* библиотеке које се користе у ове намене.

Треће поглавље описује податке који су коришћени за класификацију. У питању су подаци о земљотресима у периоду од 1965. до 2016. године. Такође, објашњен је и поступак који је коришћен приликом претпроцесирања табеле земљотреса.

У четвртом поглављу су описани теоријски принципи алгоритама класификације машинског учења. Конкретно, објашњени су алгоритми *k* најближих суседа, стабла одлучивања и алгоритам машина потпорних вектора.

Пето поглавље описује практичну реализацију наведених алгоритама у програмском језику *Python*. За сваки алгоритам је наведен поступак обраде података, поделе у подскупове и коначне вредности прецизности које су добијене за различите вредности круцијалних параметара појединачних алгоритама.

У шестом поглављу је описана могућност паралелизације наведених алгоритама. Објашњен је PyCOMPS програмски модел паралелне имплементације, који је развијен ради побољшања перформанси извршавања алгоритама.

На крају су дата закључна разматрања у вези са израдом овог рада, као и смернице за даљи развој.

# Машинско учење и истраживање података

## Машинско учење

Машинско учење је наука која представља грану вештачке интелигенције. Једно је од најпопуларнијих грана програмирања претходних година, а експанзија се тек очекује у будућности, где ће много апликација бити засновано на овом учењу.

Са практичног становишта бави се изградњом прилагодљивих рачунарских система који су способни да побољшавају своје перформансе користећи информације из искуства. Са теоријског становишта бави се проучавањем генерализације и конструкцијом и анализом алгоритама који генерализују и стога могу да врше предвиђање њиховим својствима и тиме како та својства зависе од елемената дизајна алгоритма.

Алгоритми који се користе у машинском учењу се могу поделити на алгоритме надгледаног учења, ненадгледаног учења и учења условљавањем. Алгоритми надгледаног машинског учења могу применити оно што су у прошлости научили на новим подацима користећи примере на основу којих предвиђају будуће догађаје. Полазећи од анализе познатог скупа података за тренинг, алгоритам надгледаног учења креира функцију за предвиђање излазних вредности. Алгоритам надгледаног учења такође може да упореди свој излаз са исправним, тј. предвиђеним излазом и пронађе грешке како би у складу са тим модификовао модел.

Супротно томе, алгоритми ненадгледаног учења се користе када информације које се користе за обуку нису класификоване нити етикетиране. Систем не проналази правe излазне вредности, али истражује податке и може извући закључке из скупова података како би описао скривене структуре из некласификованих података.

Алгоритми машинског учења условљавањем су методе учења које у интеракцији са окружењем производе акције и откривају грешке и награде. Претрага грешака и одложено награђивање најрелевантније су карактеристике учења условљавањем. Ови алгоритми омогућавају машинама и софтверским агентима да аутоматски одреде идеално понашање у одређеном контексту како би побољшали његове перформансе. Потребна је једноставна повратна информација да би агент схватио која је акција најбоља, ово је познато као сигнал појачања.

Основни принцип машинског учења је креирање модела којем се додељују огромне количине података. Са довољним скупом података, добро креиран модел машинског учења може да оствари велику прецизност предвиђања резултата. Модел треба да ”предвиди” којој класи припада новоунети податак на основу искуства и информација из великих количина података које су му дате, применом неких од алгоритама машинског учења. Модел машинског учења може бити математички приказ процеса у стварном окружењу. Настаје као резултат тренирања података и користи се израду предвиђања. Главне примене машинског учења у реалним животним сферама су у аутономној вожњи аутомобила, роботици, прогнозирању кретања на берзи, итд.

## Истраживање података

Истраживање података (енгл. *data mining*) је научна дисциплина која омогућава издвајање квалитативних информације из великих скупова података, који се налазе у складиштима података. Такође се може рећи да је истраживање података процес проналажења аномалија, образаца и корелација међу великим количинама података. *Data WareHouse* односно складиште података је као концепт настао касних 1980-их. Складишта представљају неку врсту пречишћених података прилагођених извршењу што већег броја различитих извештаја. Складишта података не служе за *online* трансакције где се претежно врши унос или ажурирање нових података. Складишта података се „пуне“ подацима из OLTP (*Online Transaction Processing*) система или других складишта аутоматизованим процедурама које обезбеђују скупљање података, њихову анализу и организовање често у потпуно другачијем хијерархијском моделу од оног оригиналног извора података.

Системи који обезбеђују извештаје који су резултат синтезе великог броја података и обично су подршка неком стратешком одлучивању, називају се OLAP (*Online Analytical Processing*) системи. OLAP системима су извори података управо складишта података. Дакле, складишта података омогућавају OLAP системи, а системи за управљање релационим базама података омогућавају OLTP системи.

Количина података у свету се константно повећава и то је главни разлог зашто истраживање података почиње да изазива велико интересовање. Главни интерес је да се из те велике количине података извуку корисне информације и знања, која се могу искористити у разним областима, као што су менаџмент, контрола производње, анализа тржишта, итд. Иако се корени истраживања података могу наћи у 80-тим годинама прошлог века, све до скоро није било никаквих стандарда у овој области. Тек 1999. године неколико великих корпорација је одлучило да стандардизује ову област. Као резултат ове сарадње је настао такозвани CRISP-DM (*Cross-Industry Standard Process for Data Mining*) стандард у процесу истраживања података (слика 1). Овај стандард се састоји од следећих корака:

* **Схватање организације** – Циљ истраживања је да се пронађу занимљиви резултати који могу бити сврсисходни и схватање који подаци су потребни.
* **Схватање података** – Ова активност има за циљ откивање суштине података, разумевање како су они настали и зашто су битни за пројекат.
* **Припрема података** – Задатак овог корака је да се подаци пречисте да би они постали погоднији за истраживање.
* **Моделирање** – Модели су тачка где истраживање података прелази из припреме и разумевања и сферу развоја и тумачења.
* **Процена** – Уз помоћ бројних математичких и логичких техника уз присуство људског фактора и логике долази се до решења овог корака.
* **Развој** – Ово је корак који обухвата аутоматизовање модела, интеграцију са системима и мерење перформанси.



**Слика 1.** CRISP-DMстандард

## Поступак креирања пројекта у области истраживања података

Сваки пројекат везан за истраживање података захтева основне кораке за његово креирање.

Први корак је унос података, обично се подаци уносе у формату .csv фајла.

Други корак представља „чишћење података“, у овом кораку се врши претпроцесирање података. То се ради у циљу смањења количине података која се додељују моделу, али се мора водити рачуна да ти подаци заиста немају важност, иначе ће модел бити оштећен одузимањем битних података за предвиђање, а самим тим ће се смањити и тачност предвиђања. Уколико постоје колоне које као своје вредности имају текст, сви ти текстуални подаци представљају категоричке вредности и њима је потребно доделити нумеричке вредности, зато што алгоритам прихвата као улазне параметре искључиво нумеричке вредности. Замена категоричких података је моћан начин за укључивање ненумеричких типова података. Категорички подаци односе се на вредности података које представљају категорије - вредности података са фиксним и неуређеним бројем вредности, на пример пол (мушкарац / жена) или сезона (лето / зима / пролеће / јесен). Ове вредности могу бити представљене помоћу варијабли – променљивих које садрже вредности као што су 1 или 0 које представљају присуство или одсуство категоричке вредности.

Трећи корак представља поделу података на тренинг податке и тест податке, ради сигурности да ће модел добро предвиђати. Нека стандардна подела је да тренинг подаци обухватају 70 – 80% података, док тест подаци 20 – 30%.

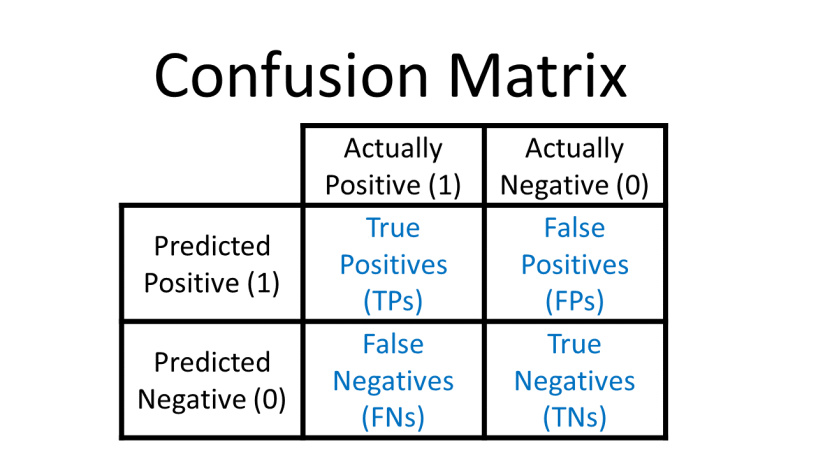
Четврти корак представља креирање поменутог модела и одабир алгоритма за анализу података. *Python* библиотеке у себи имају уграђен велики број алгоритама за машинско учење, али у овом раду ће акценат бити на алгоритмима класификације.

Пети корак је тренирање модела, а то се ради на начин да се функцији за тренирање података доделе тренинг подаци који су додељени у трећем кораку.

Шести корак је само предвиђање.

Седми корак се односи на одређивање процента успешности предвиђања и побољшање резултата. На основу резултата предвиђања и оцене прецизности, оцењује се успешност поступка и врше преправке у уносу параметара алгоритма уколико се сматра да је могуће добити већа прецизност, односно прелази на потпуно други алгоритам. Оцена прецизности је најбитнији излазни параметар алгоритама у истраживању података и најчешће се добија матрицом конфузије (слика 2).

Редови у матрици конфузије одговарају оним вредностима које је алгоритам предвидео, док колоне одговарају стварним вредностима. Свака матрица садржи онолико колона, колико садржи табела над којом вршимо истраживање података. Горњи леви угао матрице представља поље TP – *True Positive* и у њему се налази број погодака алгоритма у односу на праве вредности. У доњем десном углу матрице је поље TN – *True Negative* и у њему је број промашаја алгоритма у односу на праве вредности. Доњи леви угао представља поље FN – *False Negative* и у њему број негативних промашаја, односно број грешака у којима је алгоритам предвидео неку негацију, а реално је ту позитивна вредност и на крају у горњем десном углу FP – *False Positive* , ту се налази број грешака које је предвидео да су позитивне вредности, а заправо су негативне.



**Слика 2.** Матрица конфузије

## Python библиотеке за машинско учење

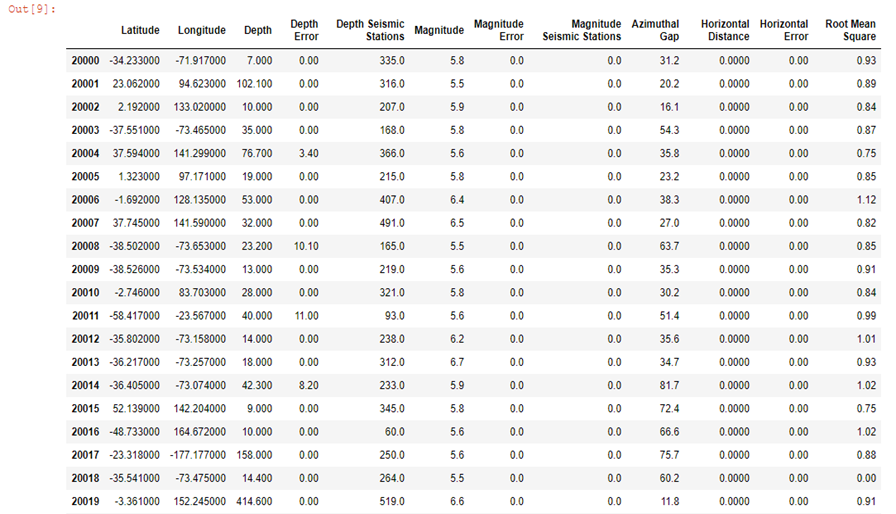
*Python*-ове стандардне библиотеке су веома обимне и нуде широк спектар садржаја. Библиотека садржи уграђене модуле (написане у С програмском језику) који пружају приступ системским функцијама као што су I/О датотеке које би иначе биле недоступне *Python* програмерима, као и модуле написане у *Python*-у који пружају стандардизована решења за многе проблеме који се јављају у свакодневном програмирању. Неки од ових модула специјално су дизајнирани да подстакну и побољшају мобилност *Python* програма.

Библиотека која се у овом раду највише користи је *scikitlearn*. Ово је бесплатна библиотека за машинско учење. Подржава различите алгоритме машинског учења као што су машина потпорних вектора (енгл. *Support Vector Machine*), алгоритам стабла одлучивања (енгл. *Decision Tree*) и *k* најближих суседа (енгл. *k-Nearest Neighbours*), а такође подржава *Python* нумеричке и научне библиотеке. *Pandas* је једна од најчешће коришћених *Python* библиотека у истраживању података. Омогућaва високе перформансе, једноставне структуре и алате за анализу података. За разлику од *numpy* библиотеке која пружа објекте за вишедимензионалне низове, *pandas* ради са објектима у 2D табели у меморији названој oквир података (енгл. *data frames*).

# Подаци ЗА КЛАСИФИКАЦИЈУ

## Земљотреси у периоду од 1965. до 2016. године

Као велику количину улазних података за модел су узети сви земљотреси почев од 1965. године до 2016. године. Табела садржи 21 ред и 23.413 колона. У табели се налази тачно време и датум сваког земљотреса, који је био степен магнитуде, на којој дубини је био епицентар земљотреса, заправо све физичке величине које је могуће измерити, а да имају информације које могу ближе одредити земљотрес (слика 3). Занимљиво је приметити како се са развојем технологије и науке заправо сваке године све више зна о особинама земљотреса.



**Слика 3.** Исечак из табеле земљотреса

Може се приметити да први редови у табели имају само основне податке о земљотресу, док са порастом година колоне престају да буду празне , тако да се заправо простим прегледом табеле може отприлике утврдити које године су откривене нове технологије за мерења сеизмичких промена, а заправо и сами алгоритми класификације.

## Претпроцесирање табеле земљотреса

У класификацији подаци играју главну улогу. Од њихових особина зависи како ће се неки алгоритам класификације понашати, колико ће добар модел да направи, а неки алгоритми чак захтевају експлицитно спремљене податке. Ту се долази до појма претпроцесирања, које представља скуп техника којима се добијају погодни подаци који ће бити улаз за алгоритме класификација. Подаци често долазе у сировој форми, стога се као решење намеће претпроцесирање података, како би били што погоднији за алгоритме. Уклањају се сви редови који су дупликати неких других редова или се уклањају сви редови у којима нема других вредности осим нула. Такође се уклањају одређене колоне које нису неопходне и које не носе битне информације.

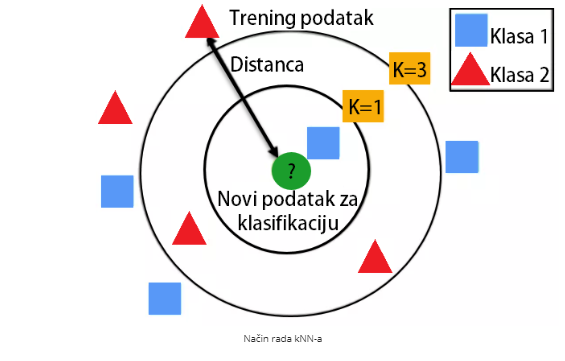
У овом раду због природе података који се обрађују, дуплирани или празни редови нису могући, али зато је пуно колона чије вредности немају никакву вредност у смислу класификовања. Из тог разлога су обрисане колоне: *ID*, *Source*, *Magnitude Source*, *Status*, *Type*, *Date*, *Time*, *Location*, *Source*, *Magnitude type*. На слици 3 може се видети које су све колоне остале. ID колона је вештачка нумеризација сваког земљотреса и не прати никакву природну појаву. За остале колоне важи да имају врло мали број различитих вредности које при том немају неки смисао да би било потребно класификовати их. *Date* колона садржи датуме, али класификација података по датуму је једноставна и није потребно примењивати алгоритме класификације машинског учења да би се земљотреси по том параметру класификовали. Са друге стране, колоне које су остављене су најважнији атрибути земљотреса и у њима готово да нема ниједне исте вредности, осим случајних примера.

# Алгоритми класификације

## kNN aлгоритам класификације

У нашем народу добро је познака изрека “С ким си такав си”. Добро је позната и мисао да ми, као појединци, представљамо “средину, од петоро људи са којима проводимо највише времена”. Слагали се са овим или не, основна идеја ових порука представља идеју за алгоритам kNN или у буквалном преводу К најближих комшија. За сваки елемент из скупа података прво се дефинише раздаљина од суседних елемената.

Класична врста ове класификације користи униформне тежине суседа, односно вредност која је додељена тачки упита је израчуната из већине гласова најближих суседа. Ипак, у неким ситуацијама суседе је потребно гледати на други начин, нпр. у односу на растојање које може бити било која метрика. Најчешће је у питању Еуклидско растојање, које представља најкраћу удаљеност између две тачке у једном простору. Та раздаљина може да представља растојање између две тачке. Параметар *k* бира аналитичар података и он утиче на број суседа за једну тачку која се испитује у његовој околини. На следећој слици је приказан основни принцип класификације kNN алгоритма.



**Слика 4.** Графички приказ принципа рада kNN алгоритма

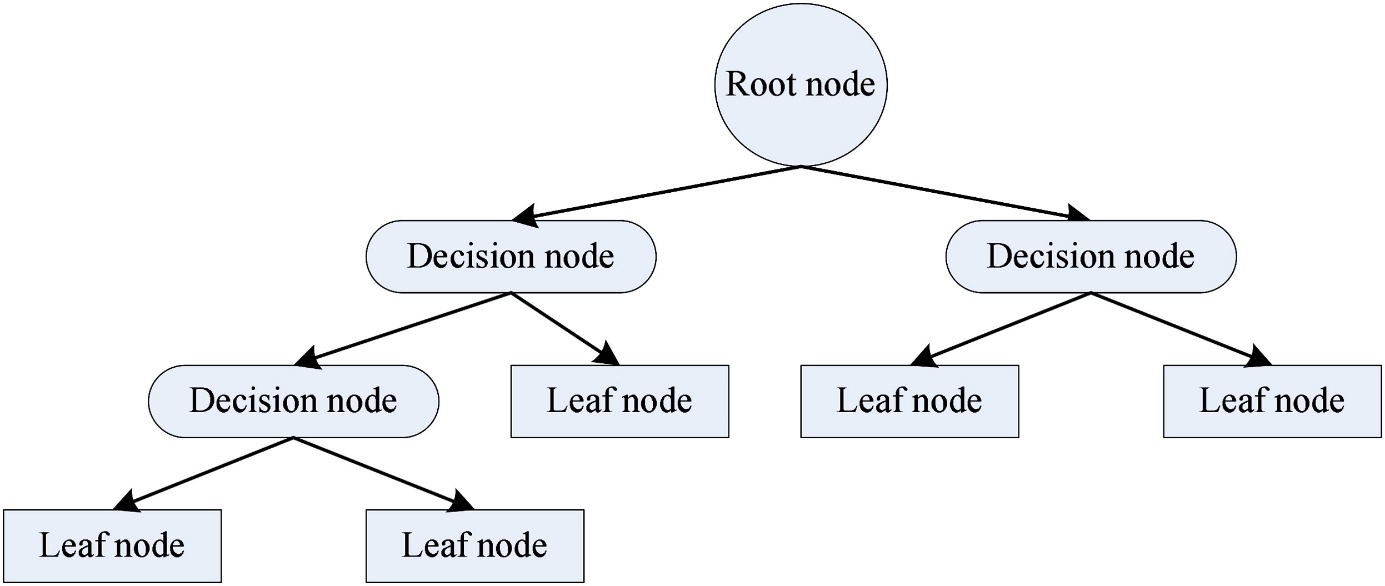
Зелени круг представља нови податак за класификацију, а алгоритам има задатак да одреди да ли припада класи 1 или класи 2. Те класе се добијају тако што се креирају два скупа података са називима класа 1 и класа 2. Они представљају улазне параметре за функције дељења података, тзв. сплитере. Класа 2 су заправо тренинг подаци, а Класа 1 тест подаци. Параметар *k* kNN алгоритма је подешен на вредност 3. Углавном се бирају непарни бројеви. Потребно је увидети које су 3 најближе тачке задатој тест тачки. То су једна тачка класе 1 и 2 тачке класе 2, такође се јасно види да уколико се постави вредност *k* на 1, алгоритам ће са 100% сигурношћу тврдити да тест податак припада класи 1, међутим већ при *k* = 3, прецизност предикције неће бити 100%, јер је окружен са 2 елемента класе 2. Уколико би се десило да нпр. *k* има вредност 25, а 24 суседа буду класе 2, алгоритам ће тест податак класификовати као класе 2, а доћи ће се до закључка да је податак класе 1 који је најближи тесту или аномалија или се догодила нека врста грешке у табели података. Зато се за број *k* никада не узима број 1, већ се гледа да то буду бројеви (3, 5, 7, 9, 11...). Разлог зашто није препоручено узимање парних бројева за вредност *k* се огледа у избегавању нерешеног резултата у броју тачака две класе. Алгоритам ће са времена на време бити врло непрецизан са *k* параметром коме је додељена парна вредност. У овом раду су *k* параметру додељене вредности 3, 5 и 25. Разлог узимања тако велике велике треће вредности за параметар је у сврху утврђивања понашања алгоритма уколико се „број комшија“ драстично промени.

## Стабла одлучивања

Стабла одлучивања сматрају се једним од најпопуларнијих приступа за класификацију података. Истраживачи из разних научно – техничких дисциплина као што су статистика, машинско учење, препознавање образаца и истраживање података бавили су се проблемом раста стабла одлучивања на основу расположивих података. Стабло одлучивања садржи корен, односно чвор који нема улазних грана. Сви остали чворови имају тачно једну улазну грану. Чвор са излазним гранама зове се унутрашњи чвор. Сви остали чворови зову се листови (такође познати као терминални или чворови одлучивања).

У стаблима одлучивања, сваки унутрашњи чвор дели улазне податке (примере) на 2 или више потпростора према одређеним дискретним функцијама улазних вредности атрибута. У најједноставнијем и најчешћем случају, сваки податак узима у обзир један атрибут, тако да је простор података подељен према вредности разматраног атрибута. Сваки лист стабла је придружен некој класи и представља најбољу одговарајућу циљну вредност. Додељивање класе неком податку се врши проласком кроз стабло почев од коренског чвора ка листовима. Истраживачи података се одлучују за мање сложена стабла одлучивања, пошто могу бити сматрана разумљивијим. Сложеност стабла има кључан утицај на његову прецизност. Сложеност стабла се експлицитно контролише помоћу критеријума заустављања који се користи и помоћу примењене методе поткресивања. Обично се сложеност стабла мери према једној од следећих метрика:

* укупан број чворова,
* укупан број листова,
* дубина стабла,
* коришћен број атрибута.

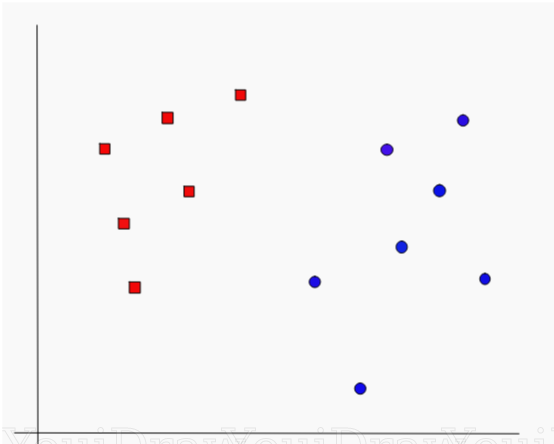


**Слика 5.** Модел стабла одлучивања

Механизми за формирање стабла одлучивања су алгоритми који аутоматски граде стабло одлучивања на основу датог скупа података. У већини случајева циљ је наћи стабло одлучивања такво да је грешка класификације минимална. Стабла одлучивања спадају у непараметарске методе надгледаног учења. Потребно је креирати модел који предвиђа вредност циљне променљиве учењем једноставних правила одлучивања која су закључена из карактеристика података. Стабла одлучивања захтевају минималну припрему података у виду избацивања недостајућих вредности.

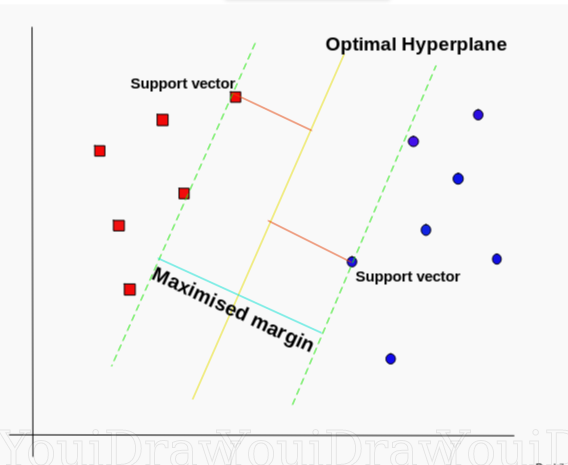
## SVM алгоритам

SVM –*Support Vector Machine* је линеарни модел за класификацију и регресијске проблеме. Може да решава линеарне и нелинеарне проблеме и послужи за многе практичне проблеме. Идеја SVM–а је једноставна: Алгоритам креира линију или хиперраван која раздваја податке у класе. У првој апроксимацији овом алгоритму је циљ проналажење раздвајајуће линије (хиперравни) између података две класе. SVM је алгоритам који узима податке као улаз и уцртава раван која раздваја те класе ако је могуће.



**Слика 6.** Скуп података црвене и плаве класе

На слици 6 су приказани скупови података који припадају двема класама (црвена и плава). Задатак алгоритма је да класификује црвене правоугаонике из плавих елипси. Дакле, задатак је заправо пронаћи идеалну хиперраван која ће тај скуп података раздвојити у две класе. Може се приметити да не постоји јединствена хиперраван која би то урадила, али овај алгоритам има уграђен метод у себи за проналажење идеалне хиперравни. Он функционише тако што проналази тачке које су хиперавни из обе класе. Те тачке се називају потпорни вектори. Након тога се израчунава удаљеност између линије и потпорних вектора. Ова удаљеност се назива маргином. Циљ је максимизирање маргине зато што је хиперраван за коју је маргина максимално заправо идеална хиперраван. На основу пређашњег објашњења се може закључити да SVM покушава да пронађе раздвајање између две класе тако да оно буде што је шире могуће. На слици 7. је све то графички приказано.



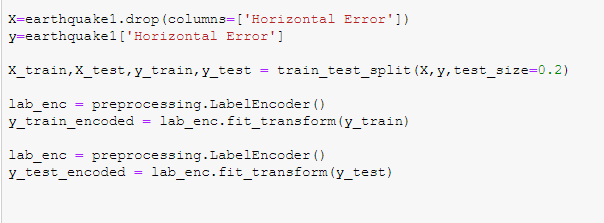
**Слика 7.** Оптимална хиперраван SVM алгоритма

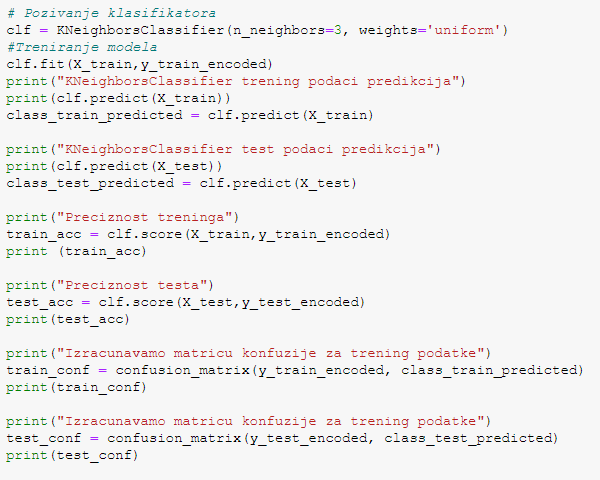
# ПРАКТИЧНА РЕАЛИЗАЦИЈА

## Реализација kNN алгоритма

Приказ кода kNN алгоритма и сви резултати добијени покретањем алгоритма класификације:



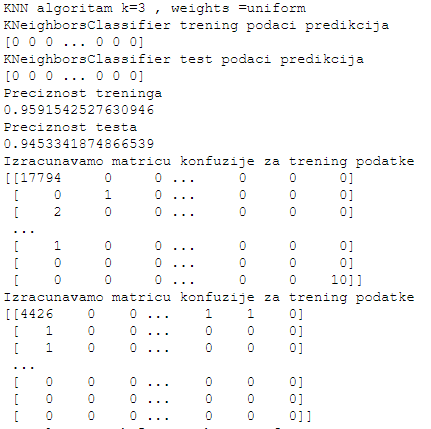


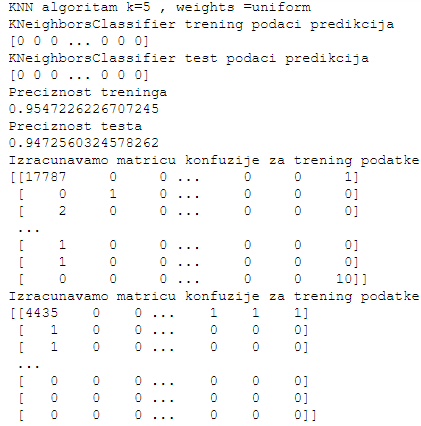


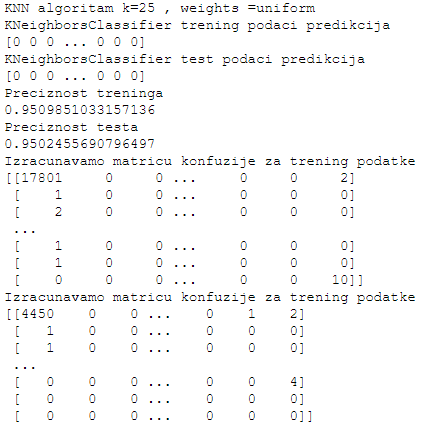
У првом делу кода се може видети позивање свих библиотека неопходних за анализу и класификацију података, поред тога види се и унос табеле података позивом функције *pandas* библиотеке. Након уноса, иде претпроцесирање података, у овом случају брисање одређених колона функцијом *drop*. Као што је већ поменуто, због природе табеле података земљотреса, то је једино могуће урадити у поступку претпроцесирања, осим замене вредности NaN (*Not a Number*) са нулама, због прилагођавања алгоритмима. Оно што је битно напоменути, овај део је исти за сваки алгоритам класификације, па и за друге методе истраживања података.

У другом делу кода извршена је манипулација над подацима. Променљиве **X** и **y** заправо су улазни сет података, односно излазни сет података, респективно. Променљива **у** представља излазну променљиву алгоритма и обухвата колону хоризонталне грешке (*Horizontal Error*). Позивом функције за насумичну поделу табеле података *train\_test\_split*, креиран је подскуп података за тренирање, а затим и подскуп података за тестирање. Функција *train\_test\_split* као повратну вредност враћа *tupple*, што је у ствари скуп нераспакованих објеката у *Python*-у. Подаци су распаковани и додељени у 4 различите променљиве. Следеће што је урађено је прилагођавање излазних података формату који одговара функцији класификације. Без тог дела није могуће истренирати модел.

Приказ резултата након тестирања алгоритама за различите вредности параметра *k*:







Након прве класификације резултати су били одлични, прецизност је велика како за тренинг податке (95,92%), тако и за тест податке (94,53%). Касније су промењене вредност параметра *k* на 5 и 25, да би се утврдило како ће модел да се понаша при различитим бројем „комшија“. Показало се да је модел стабилан и да за све вредности даје добре резултате. Проценат прецизности, не зависи само од алгоритма, зависи и од тога која се колона одабере као излазни параметар. Конкретно, за колону *Магнитуда*, када се она изабере као колона која ће бити сет излазних података, прецизност података је 3 пута лошија, из разлога што јачину, односно магнитуду има свака инстанца ове тaбеле, а хоризонталну грешку веома мали број инстанци. Разлог томе је што је мерење извршено савременом технологијом, а што је особина за класификовање ређа, прецизност је већа. Приказ остварене прецизности алгоритма за различиту вредност параметра *k* је дата у табели 1.

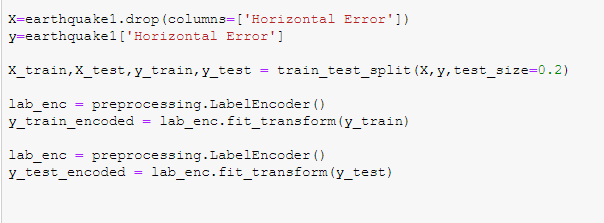
Табела 1. Прецизност kNN алгоритма у различитим ситуацијама

|  |  |  |  |
| --- | --- | --- | --- |
| ***k*** | **3** | **5** | **25** |
| **Прецизност тренинг подскупа** | 95,91% | 95,47% | 95,09% |
| **Прецизност тест подскупа** | 94,53% | 94,72% | 95,02% |

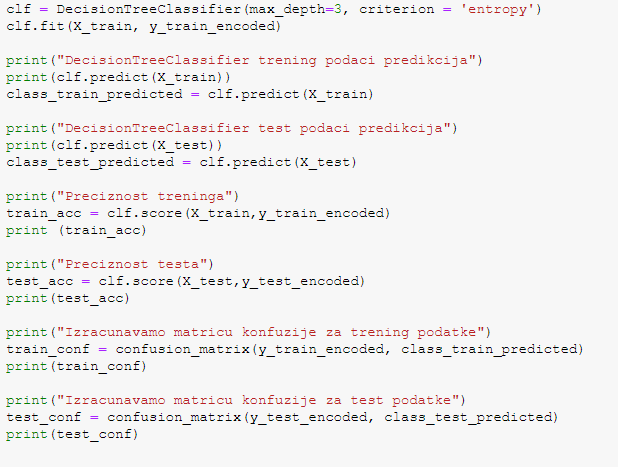
## Реализација алгоритма стабла одлучивања

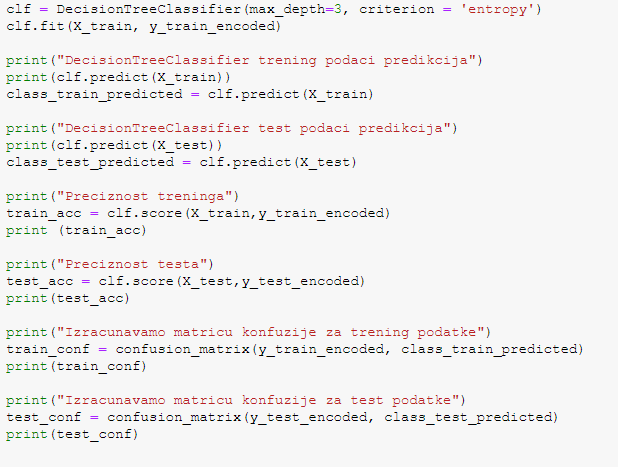
Приказ кода алгоритма стабла одлучивања и сви резултати добијени покретањем алгоритма класификације:



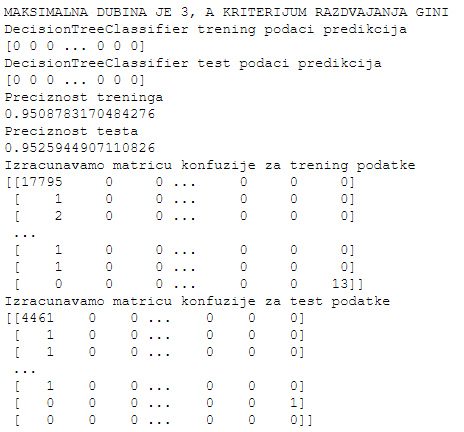


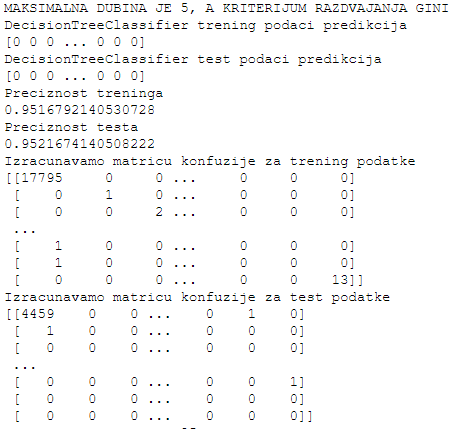
Овај део кода је исти за сваки алгоритам и детаљно је објашњен у претходном примеру.

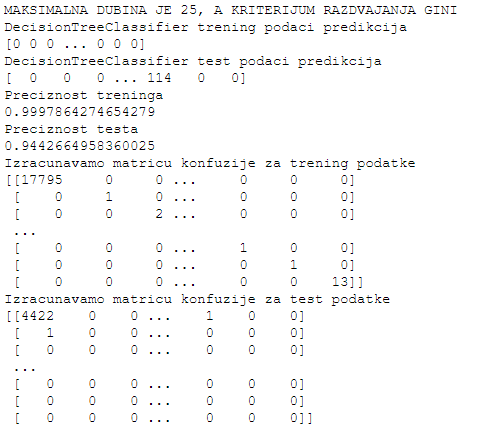


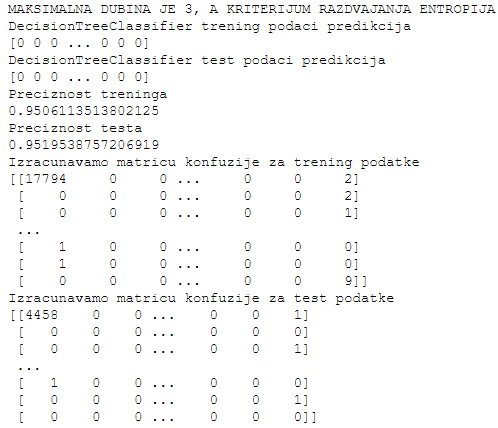


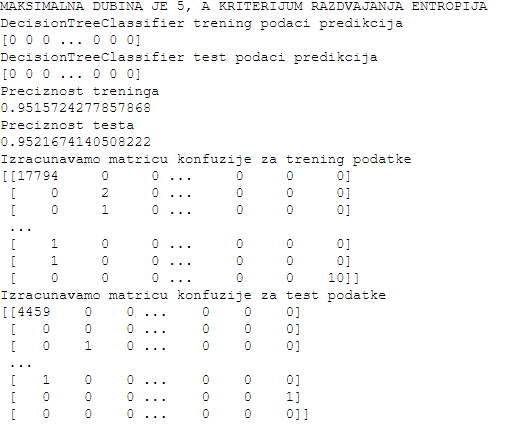
Са слике кода може се приметити да је као параметар алгоритма мењана максимална дубина стабла *max\_depth*, као и критеријум раздвајања *criterion*. За тај параметар је конкретно мењано између *Gini* индекса и *Entropije*, док је за вредност максималне дубине стабла узимано 3, 5 и 25. Ентропија је 0 ако сви узорци чвора припадају истој класи, а ентропија је максимална ако имамо равномерну дистрибуцију класа. Другим речима, ентропија чвора (ако се састоји од једне класе) је 0, јер је вероватноћа 1, а ентропија се заснива на логаритамској функцији, (*log*(1) = 0). Ентропија достиже максималну вредност када све класе у чвору имају једнаку вероватноћу. Слично ентропији, *Gini* индекс је максималан ако су класе савршено измешане. Баш као и у првом примеру, потребно је видети како се алгоритам понаша у сличним ситуацијама, као и како реагује на драстичну промену параметара. Резултати добијени покретањем алгоритма су следећи:

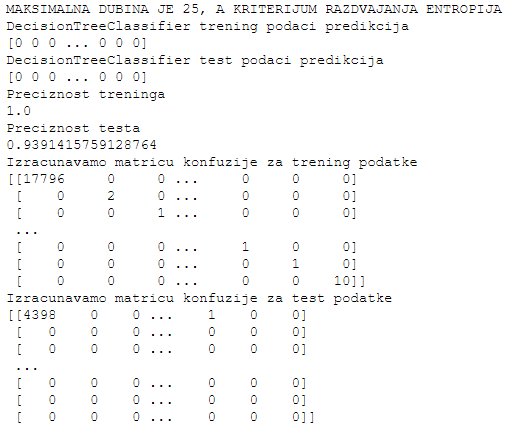












Излазни резултати алгоритма су приказани у табели 2.

Табела 2. Прецизности алгоритма стабла одлучивања у различитим ситуацијама

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Критеријум** | ***entropy*** | | | ***gini*** | | |
| **Max\_depth** | **3** | **5** | **25** | **3** | **5** | **25** |
| **Прецизност тренинг подскупа** | 95,06% | 95,15% | 100% | 95,08% | 95,16% | 99,97% |
| **Прецизност тест подскупа** | 95,19% | 95,21% | 93,91% | 95,25% | 95,21% | 99,42% |

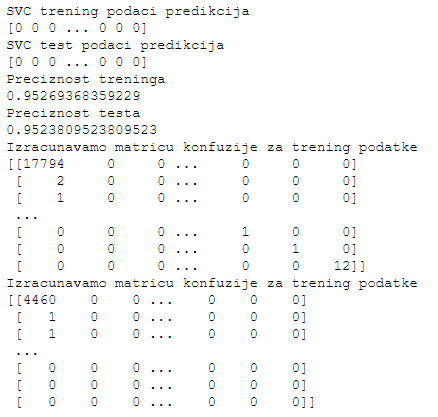
Резултати алгоритма показују да модел са великом прецизношћу предвиђа, како за различите дубине стабла одлучивања, како и за различите критеријуме раздвајања. Интересантно је приметити да при повећању дубине стабла на 25, прецизност тренинга расте и у случају ентропије је 100%, док код *gini* критеријума готово да је 100%. Међутим, проценат тачности предвиђања тест података пада при овако драстичном повећању дубине стабла, а такве резултате је потребно избећи тако да се може закључити да су мале дубине стабла оптималне за овај скуп података.

## Реализација SVM алгоритма

Приказ кода SVM алгоритма и резултати добијени класификацијом:







Као и код осталих алгоритама код је исти, осим параметра који се додељује код тренирања модела. Тај параметар је заправо кернел. SVM алгоритми користе скуп математичких функција које су дефинисане као кернел. Функција кернела је да узме податке као улаз и трансформише их у тражени облик. Ове функције могу бити различитих типова, на пример линеарна, нелинеарна, полиномна, радијална основна функција (RBF) и сигмоидна. Линеарни кернел се користи када су подаци линеарно раздвојиви, односно могу се одвојити помоћу једне линије. То је један од најчешћих кернела који се користе. Најчешће се користи када постоји велики број функција у одређеном скупу података. Са резултата класификације се може видети да је алгоритам врло ефикасан за дати скуп података, прецизност тренинг подскупа је 95,27%, а тест подскупа 95,24%.

# могућност паралелизације алгоритама машинског учења

Због велике количине вишедимензионалних података, конвенционални алгоритми и алати за машинско учење често су непрактични за употребу у многим индустријским сферама. За решавање овог проблема алгоритми су често или поједностављени или се покрећу паралелно преко рачунарске инфраструктуре с више машина, нпр. рачунарског кластера.

У великим индустријским постројењима прикупљене количине података толико су велике да су конвенционални алгоритми машинског учења поприлично непрактични за рачунање у реалном времену. Постоје два уобичајена начина решавања ових проблема:

1. поједностављивање алгоритама,
2. паралелно извршавање алгоритама.

Барселонски суперкомпјутерски центар креирао је модел програмирања названа COMPS и његову верзију програмског језика *Python* која се зове PyCOMPS, са идејом да се поједностави процес паралелног програмирања и учини га лакши и доступнијим корисницима.

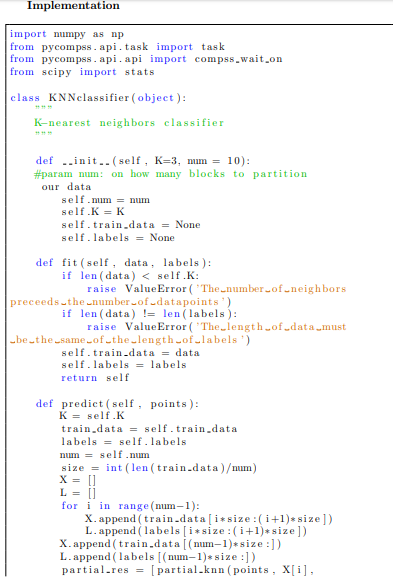
## PyCOMPS програмски модел

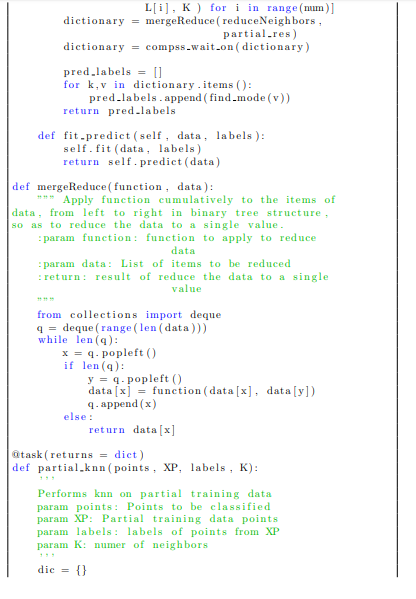
Подразумевана имплементација програмског језика *Python*, *CPython*, не може покренути вишеструко извршавање нити, а већина постојећих решења је или неизмерно компликована или недовољно ефикасна. Да би се решио овај проблем научници су креирали PyCOMPS , паралелни програмски *framework* за *Python* апликације. PyCOMPS апликација је секвенца *Python* скрипти. У моделу је корисник углавном одговоран за идентификовање функција које се извршавају као асинхрони паралелни задаци. Систем за извршавање у реалном времену је задужен за аутоматско искоришћавање инхерентне конкурентности скрипти, откривање и извршавање зависности података између задатака и мреста где су ови задаци доступни изворима, који могу бити чворови у кластеру, *cloud*-у или решетки.

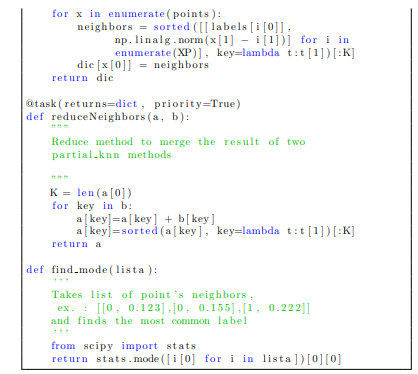
Мрест у рачунарству (енгл. *spawn*) односи се на функцију која учитава и извршава нови подређени процес. Текући процес може да сачека да се дете процес прекине или да настави да извршава паралелно рачунање. Стварање новог потпроцеса захтева довољно меморије у којој се може извршити и подређени процес и тренутни програм.

## Пример паралелизације

Код програмирања са PyCOMPS – ом, први корак корисника је дефинисање задатака. Апликација се може састојати од већег броја функција, а неке од тих функција могу бити рачунски веома захтевне. Такве функције су добри кандидати за задатке који ће се паралелно извршавати. На слици је представљен пример примене алгоритма за *k* PyCOMPS . Раније је детаљно објашњен kNN алгоритам, а на слици испод може се видети имплементација паралелизма над овом класификацијом. Принцип је такав да се алгоритам раздвоји на функције и одлучи коју од њих треба дистрибуирати на који начин. С обзиром на природу kNN-а, то се не мора радити у фази тренинга, јер је напоменуто да се код kNN-а тренирање података врши само једном, након тога више пута је могуће само вршити предвиђања, али паралелизација је укључена у фазу предвиђања. Идеја је поделити оригиналне скупове података у делове и посматрати их као самосталне проблеме, изводи се мини kNN паралелно сваки за себе, да би се на крају вршило спајање резултата. Слика приказује идеју алгоритма и детаљно објашњење је у наставку.



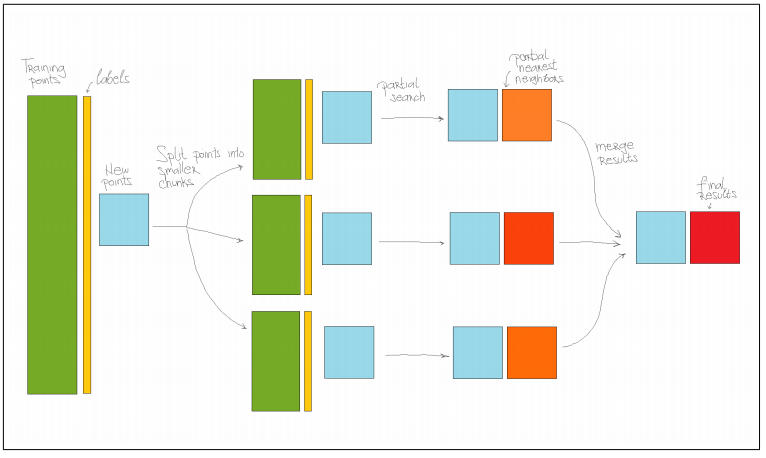




Поред провере димензија података, чувају се само тачке које припадају тренинг подскупу који се касније користи за одређивање припадности нових тачака.

Метода .*predict*() је сложена и у њу су укључене све функције које нису наведене. Параметар *num* адресира број делова на које се желе поделити податке приликом обраде. Подаци се деле на одређене делове и чувају у листи X, а из те листе се узимају подаци за рад парцијалног kNN-а. Ове функције су задаци и извршаваће се паралелно, сви делови података биће обрађивани истовремено. За нове улазне податке су потребне тачке у односу на које се врши класификација и један део података над којим ће програм вршити смањени kNN. Када се израчунају сви делимични kNN-ови , над свим деловима података, излаз, односно речници се преносе функцији *mergeReduce* која спаја све добијене речнике у један и он као такав садржи ознаку односно лабелу за све тачке.

Метода .*fit\_predict*() изводи .*fit*() и .*predict*() методе и врши обраду над истим подацима, враћајући ознаке које нису нужно исте као оригиналне ознаке.



**Слика 8.** Приказ извршавања паралелних задатака

На блок шеми изнад је сликовито објашњен цео поступак поделе података у процесу паралелизације и на крају спајање података у један резултат који би требао да има исте вредности као да је рађен стандардним методама, али са мањим временом извршавања. Нажалост, овај *framework* је врло сложен и његова практична имплементација би превазишла оквире овог рада.

# закључак

У овом раду је приказано неколико алгоритама којима се описани подаци могу класификовати. Искоришћене су неке од најчешћих метода класификација, стратешки су им мењани параметри, са циљем да се направи модел који би убудуће могао што боље да класификује податке, са којима до тад није имао додирних тачака. Сви алгоритми класификације су најпре детаљно теоријски објашњени, а након тога је и извршена практична имплементација.

Овај рад се може унапредити проширивањем скупа метода класификација које су вршене над њим, али за то је рачунар са високим перформансама паралелне обраде, јер су присутне велике количине података, а алгоритми који добро класификују најчешће захтевају веће перформансе. Још један начин за унапређење овог рада је поменути PyCOMPS *framework* за паралелизацију алгоритама класификације, који је сам по себи довољно сложен да се један цео рад може написати на ту тему. Овим се отвара простор и идеја да то буде тема за даље истраживање у којем ће се остварити практична имплементација PyCOMPS *framework*-a над истим скупом података као у овом раду и упоредити добијене резултате са класичним методама класификације из овог рада и донети одређене закључке.

# ЛИТЕРАТУРА

1. “Data Science from Scratch with Python: Concepts and Practices with NumPy, Pandas, Matplotlib, Scikit-Learn and Keras”, AI Publishing, 2019.
2. Andreas Muller, Sarah Guido, “Introduction to Machine Learning with Python: A Guide for Data Scientists”, O’Reilly, 2016.
3. Aurelien Geron, “Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”, O’Reilly, 2017.
4. Daniel Chen, “Pandas for Everyone: Python Data Analysis: Python Data Analysis (Addison-Wesley Data & Analytics Series)”, Addison-Wesley Professional, 2018.
5. Enric Tejedor et al., “PyCOMPSs: Parallel computational workflows in Python”, The International Journal of High Performance Computing Applications, Vol. 31, No. 1, pp. 66-82, 2015.
6. Jake VanderPlas, “Python Data Science Handbook: Essential Tools for Working with Data”, O’Reilly, 2016.
7. Joel Grus, “Data Science from Scratch: First Principles with Python”, O’Reilly, 2015.
8. Sebastian Raschka, “Python Machine Learning”, Packt Publishing, 2015.
9. Trevor Hastie, Robert Tibshirani, Jerome Friedman, “The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)”, Springer, 2016.
10. Barcelona Supercomputing Center, “COMP Superscalar”, доступно на: <https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar>