УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Илија Докнић

Класификација облака тачака употребом алгоритама машинског учења

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2021

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		Илија Докнић
Ментор, **МН**:		др. Миро Говедарица
Наслов рада, **НР**:		Класификација облака тачака употребом алгоритама машинског учења
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		2021
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		Поглавља 8; Страна 95; Цитата 51; Табела 5; Слика 59; Једначина 27;
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника
Предметна одредница/Кључне речи, **ПО**:		Машинско учење, облак тачака, сегментација, неуронске мреже
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		У оквиру ове студије извршена је класификација облака тачака коришћењем метода машинског учења. Извршена је сегментација облака тачака као корак у предпроцесирању. Такође је одрађена компрација између тренирања на сегментисаним и несегментисаним облацима тачака. У истраживању су примењени алгоритми неуронсхик мрежа и насумичне шуме као и логистичка регресија
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	
	Члан:	Потпис ментора
	Члан, ментор:	**др. Миро Говедарица, редовни професор**

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monographic publication
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Bachelor Thesis
Author, **AU**:	Ilija Doknic
Mentor, **MN**:	Miro Govedarica PhD
Title, **TI**:	Classification of the point cloud using machine learning algorithms
Language of text, **LT**:	Serbian
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Republic of Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2021
Publisher, **PB**:	Author's reprint
Publication place, **PP**:	Novi Sad, Dositeja Obradovica sq. 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	Chapters:8; Pages 99; Citations 51; Pictures :59; Equations:27; Tables:5;
Scientific field, **SF**:	Electrical Engineering
Scientific discipline, **SD**:	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, **S/KW**:	Machine learning, point cloud, segmentation, neural networks
UC	
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, **N**:	
Abstract, **AB**:	The bachelor thesis describes the process of classifying point clouds using algorithms of machine learning. Segmentation was applied as a step in data preprocessing. Also, a comparison was done between training on segmented and non-segmented point clouds. In research, neural networks and random forest algorithms were applied as well as logistic regression.
Accepted by the Scientific Board on, **ASB**:	
Defended on, **DE**:	
Defended Board, **DB**: President:	
Member:	
Member, Mentor:	PhD Miro Govedarica

Menthor's sign

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА	Лист/Листова: IV/112

Врста студија:	☒ Основне академске студије ☐ Основне струковне студије
Студијски програм:	**Геодезија и геоматика**
Руководилац студијског програма:	

Студент:	**Илија Докнић**	Број индекса:	
Област:	**Ласерско скенирање терена и објеката**		
Ментор:	**Проф. др Миро Говедарица**		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

КЛАСИФИКАЦИЈА ОБЛАКА ТАЧАКА УПОТРЕБОМ АЛГОРИТАМА МАШИНСКОГ УЧЕЊА

ТЕКСТ ЗАДАТКА:

На основу стечених теоријских и практичних знања из области машинског учења и ласерског скенирања терена и објеката израдити модел класификације ЛИДАР података методама машинског учења. Предложити и дефинисати организацију података и поступак рада у оквиру модела класификације. Описати алгоритме и поступке у примени тих алгоритама у класификацији облака тачака. Посебну пажњу обратити на карактеристике метода машинског учења и анализу прикупљених података методама даљинске детекције.

У циљу верификације модела класификације на основу расположивих тест података извршити верификацију на одабраном подручју. Посебно обратити пажњу на оцену квалитета класификованих података. Урадити оцену тачности класификације.

Руководилац студијског програма:	Ментор рада:
	Проф. др Миро Говедарица

Примерак за: ☐ - Студента; ☐ - Ментора

Захвалност

Захваљујем се свима који су безрезервно помогли израду овог дипломског рада, а посебно свом ментору др. Миру Говедарици. Помоћ су пружили Игор Русковски, Марија Ђаковић, Жељко Бугариновић и Наталија Галовић.

Највећу захвалност за подршку током студирања и израде дипломског рада, дугујем својим родитељима Нади и Радојици, породици и пријатељима.

САДРЖАЈ

Списак слика

Списак табела

Списак једначина

Скраћенице

LIDAR	*- Light detection and ranging*
ANN	*-Artifical neural networks,*
CART	*- (Classification and regression tree),*
ReLu	*-Linear rectified unit,*
DBSCAN	*- density-based spatial clustering of applications with noise*
RMSProp	*- Root Mean Squared Propagation*
ADAM	*-Adaptive moment estimation*
TIN	*- Triangular irregular network*
SVM	*- Support Vector Machines*
ГНСС	*-Глобални навигациони сателитски систем*
ИНС	*- Инцерцијални навигациони системи*
БДП	*- Бруто домаћег производа*

1. Увод

LiDAR технологија (LiDAR – Light Detection and Ranging) је тренутно једна од најактуелнијих поља истраживања у Геоинформатици. Развој технологија као што су ГНСС, ИНС и развој рачунара уопште омогућио је LiDAR системима да у малом временском периоду прикупе огромне количине квалитетних података. Међутим, са развитком технологије за аквизицију података, долази до питања како те силне податке обрадити. Тренутно постоје многи алгоритми за њихово процесирање и класификацију, али и поред њих, човек мора да одради напорну мануелну рекласификацију да би резултати били на задовољавајућем нивоу.

Машинско учење, вештачка интелигенција, дубоко учење са друге стране су изрази које се могу чути свакодневно у разговору инжењера као и људи из осталих сфера. Могућност која је дата рачунарима да у неком погледу мисле и понашају се као људи, увела је револуцију у рачунарском свету. Многи алгоритми су дизајнирани на овим принципима и примењују се у разним задацима.

Циљ овог рада је да се испита примена алгоритама машинског учења за класификацију LiDAR података, како би добили бољу класификацију и свели људски мануелни рад на минимум. Самим тим омогућили бисмо и аутоматску класификацију огромних количина података. Други циљ овог рада је да се испита улога сегментације као корака предпорцесирања у класификацији.

2. LiDAR системи

2.1 Основне карактеристике LiDAR система

LiDAR (LiDAR – Light Detection and Ranging) је метода мерења из ваздуха, која на основу ласерског зрака и осталих интегрисаних уређаја даје тродимензионалне координате за све тачке на земљи од којих се ласерски зрак одбио [2].

Како LiDAR заправо добија тродимензионалне координате? Користи се принцип заснован на мерењу времена које прође од тренутка када је емитован ласерски зрак до тренутка када уређај прими рефлектовани зрак. На основу добијеног времена и на основу познате брзине простирања ласерског зрака, могуће је израчунати дистанцу од уређаја па до објекта тј. тачке од које се ласерски зрак одбио. Сада, када је познато измерено растојање од скенера до објекта, неопходно је још и да сама позиција скенера буде позната као и његова оријентација у тренутку емитовања таласа да би добили X,Y,Z координате. Оријентацију уређаја као и његову позицију можемо да добијемо на основу ИНС уређаја, односно на основу ГНСС уређаја. Сада када знамо све ове параметре, можемо да их комбинујемо и на основу њих израчунамо X,Y,Z координате тачке објекта.

Основне компоненте LiDAR система су дакле

- Ласер
- ИНС
- ГНСС
- Рачунар

Ласер се користи за мерење растојања до датог објекта.

$$r = t * c / 2$$

Једначина 2.1 Мерење растојања до датог објетка

r– представља растојање од сензора до објекта

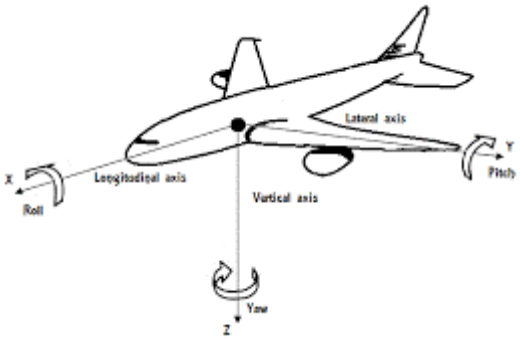
t– представља време које је потребно да ласерски зрак стигне до објекта одбије се и и дође назад до сензора

c– брзина ласерског зрака, једнака је брзини светлости

Инерцијални навигациони систем (*Inertial navigational system-INS*)је уређај за мерење параметара навигације у простору. Он користи жироскопе и уређаје за мерење растојања. ИНС бележи ротацију авиона око све три осе X,Y,Z

- X оса је оса у правцу лета авиона (*Roll*)
- Y оса налази се у хоризонталној равни и управна је на X осу (*Pitch*)

Z оса се налази у вертикалној равни и управна је на X и Y осу(*Jaw*)

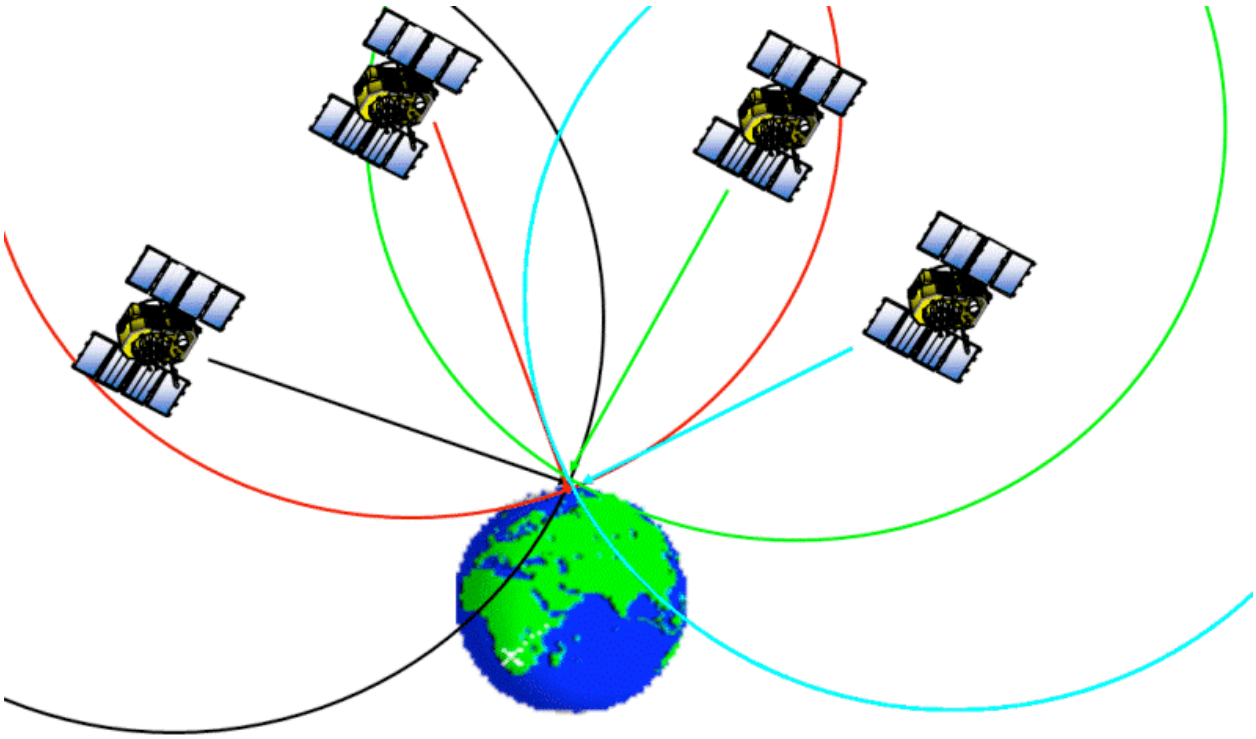


Слика 2.1 Мерење ротације око три осе

ИНС уређај је неопходан да би се могла израчунати оптимална трајекторија тј. да би се знала позиција авиона у сваком тренутку.[3]

ГНСС укључује констелацију сателита који круже Земљином орбитом и константно емитује сигнале на основу кога корисници ГНСС система могу да одреде своје тродимензионалне координате. [4]

Одређивање дистанце је базирано на геометријском проблему, где користимо дистанце од корисника до најмање четири сателита којима познајемо координате. Дистанце су израчунате на корисничком пријемнику користећи емитовани сигнал као и навигационе податке које трансмитује сателит. На основу ових дистанци одређује се позиција тачности око једног метра. Већу прецизност можемо постићи користећи напредније технике позиционирања. Једна од техника је да користимо референтну станицу са познатим координатама која са налази у близини нашег уређаја. Референтна станица прима скоро исте сигнале као и наш уређај и на основу разлике координата добијених из ГНСС мерења као и познатих координата тачке, станица шаље корекцију нашем уређају. Овом техником мерења са слањем корекција може да се постигне тачности од 5 до 10 центиметара [5].



Слика 2.2 Мерење позиције на основу минимум четри сателита. Дистанце од сателита до уређаја су представљене кружницама

Рачунар у LiDAR систему се користи да би складиштио сва ова мерења и интегрисао их и претворио у X,Y,Z координате. С обзиром да сам систем може да сними и до преко милион тачака у једној секунди, неопходно је да рачунар има добру меморију како би сачувао све те податке.

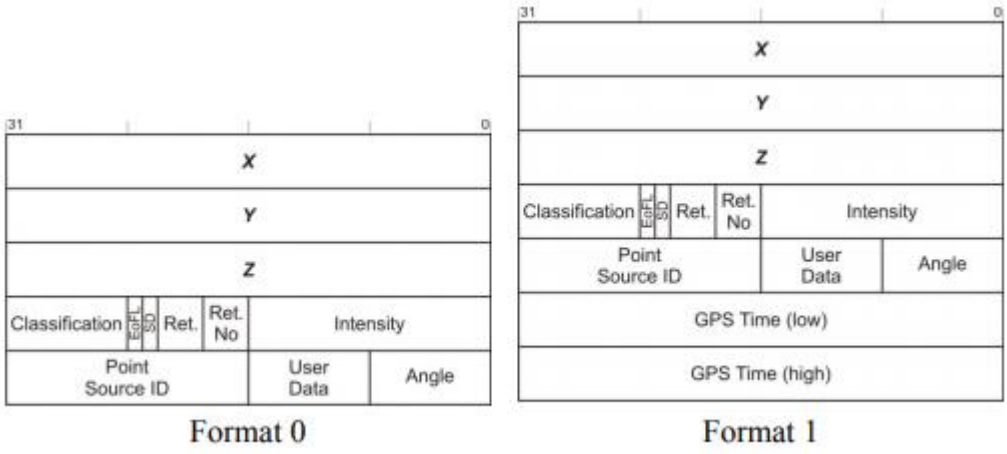
Један од најчешћих начина за представљање Лидар мерења је облак тачака. Облак тачака је скуп енормног броја тачака где свака садржи X,Y,Z координате као и интензитет одбитка. Ово је најближа представа онога што је скенер на терену забележио [2].

Стандарди формат за чување облака тачака је ЛАС фајл, и он је де факто индустријски стандард за чување лидар података. Иако постоје више различитих формата, сви формати деле три главна дела.

• Заглавље у коме се чувају генералне информације о Лидар подацима као што је лас формат, број снимљених тачака, параметри скалпирања ...

• Поље променљиве дужине који садржи метаподатке, пројекцију, кориснички подаци

• Поље фиксне величине које садржи податке о координатама тачака, и других атрибута као што су гпс време, интензитет, класа тачке итд [7].

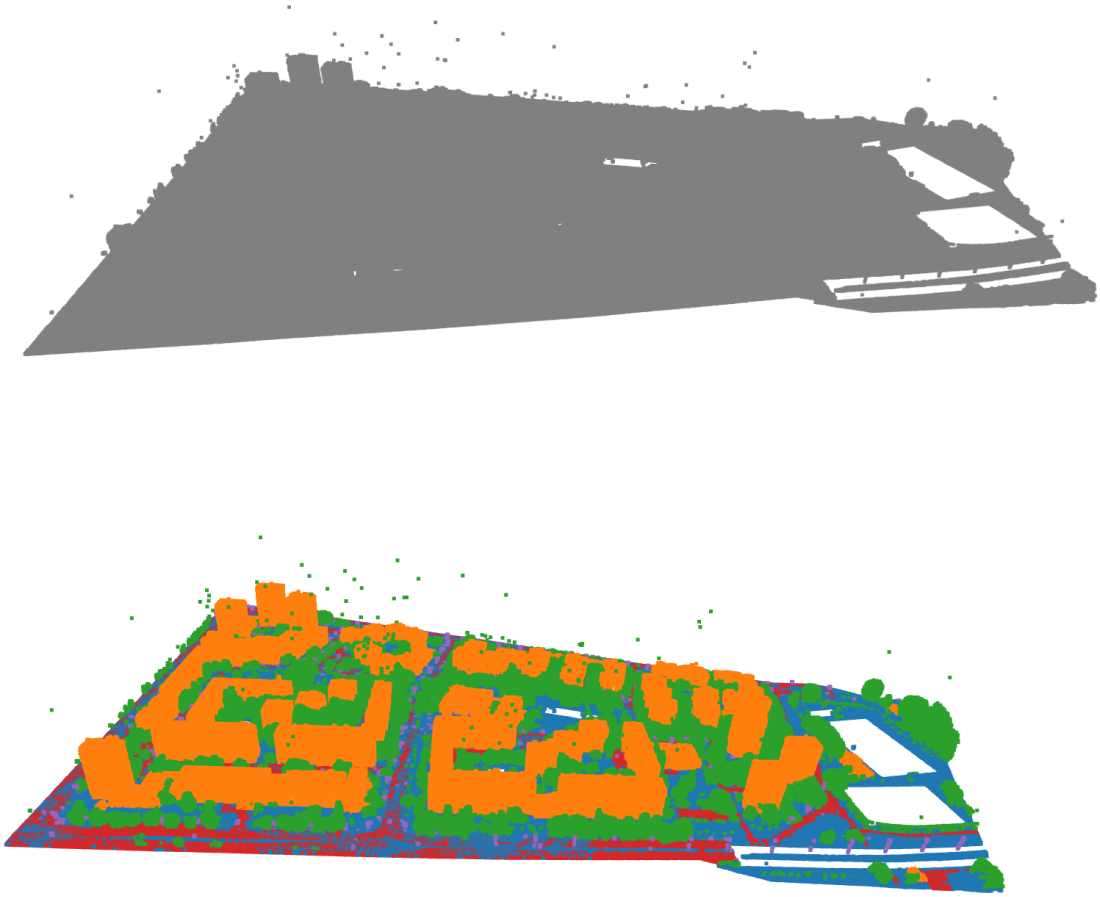


Слика 2.3 Поље фиксне дужине које садржи информације о тачкама дефинисане на основу ЛАС формате верзије 1.1

2.2 Класификација облака тачака

Један од најзначајнијих корака у обради LiDAR података, јесте њихова класификација. Класификација представља сврставање сваке тачке у једну од предефинисаних класа. Она се дели на аутоматску, полуаутоматску и мануалну класификацију.

Због недостатака аутоматских метода и захтеване тачности, најчешњи модел класификације у пракси јесте полуаутоматска класификација. Она ради тако што се прво искористе одређене класификационе рутине за аутоматску класификацију, а затим је човек својим мауелним радом поправља и употпуњује. У првој фази се углавном издвајају тачке које су резултат грешака или припадају тлу или зградама, док се у другој фази мануелног рада углавном класификују ситнији објекти (стубови, расвета, клупе..) и исправљају грешке из прве фазе.



Слика 2.4 Некласификовани и класификовани облак тачака

Класификационе рутине које се најчешње користе у пракси су

- Ниске тачке
- Изоловане тачке
- Тачке које су у ваздуху
- Тачке тла
- Тачке које не припадају тлу
- Тачке зграда

Класификације ниских тачака (*Low points*) проналази тачке које су доста ниже у односу на тачке у окружењу. Ове тачке се налазе испод земље и оне се тратирају као грешке тј. шум.

Рутина рада тако што пореди висину једне тачке са висином сваке друге тачке у датом окружењу. Тачке које знатно ниже у односу на остале тачке се сврставају у ову класу.

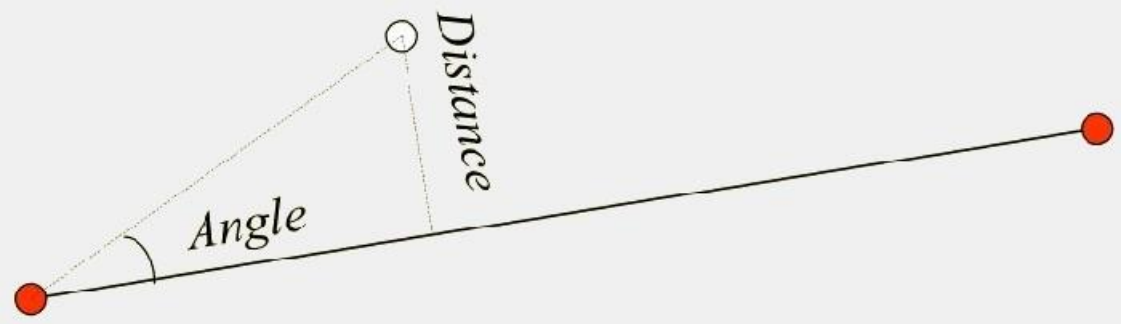
Овакав приступ функционише када тражимо једну тачку која одступа у односу на остале. Међутим, алгоритам се показао да добро функционише и у класификацији групе тачака које су знатно ниже у онодсу на остале.

Класификација изолованих тачака (*Isolated points*) класификују тачке које немају много тачака у свом 3D радијусу. Ова рутина омогућава да се изоловане тачке које су највероватније резултат грешке, било да се налазе испод земље или у ваздуху.

Класификација тачака у ваздуху (*Air points*) проналази оне тачке које знатно више у односу на средњу висину околних тачака. Када класификујемо једну тачку, овај принцип ће пронаћи све тачке у задатом радијусу. Затим ће израчунати средњу висину тачака и стандардно одступање. Тачке ће бити смештена у ову класу једино ако је висини изнад одређених параметара. Ове тачке обично представљају птице, честице у ваздуху или неке друге изворе шума, и оне уносе велике грешке у облак тачака.

Класификација тачака тла *(Ground points*) је једна од најважнијих метода класификације облака тачака. Процес почиње тако што се изаберу тачке које сигурно припадају класи тло. На основу ових тачака модел прави TIN модел површи (*Triangular irregular networks)*. Затим се креће са итеративним додавањем нових тачака на овај модел, што чини модел доста реалнијим. Тачка се додаје уколико је довољно близу троугаоној равни. Колико блиско тачка мора бити троугаоној равни зависи од параметара итерације. То су итерациони угао и итерациона дистанца.

Итерациони угао је максимални угао између тачке, њене пројекције на троугаону раван и најближе тачке. Ово је главни параметар који контролише колико тачака ће бити смештене у тачке тла. Што је угао итерације мањи, то ће метод бити крутији када додаје нове тачке и неће пратити природне неравнине у терену. Стога, када радимо класификацију облака тачака, који представља равницу, користићемо мањи итерациони угао (мањи од 4 степена). Супрото, када класификује облка тачака који представља брдовити терен, користићемо већи итерациони угао (око 10 степени)[47].



Слика 2.5 Илустрација итерационог угла и дистанце у рутине класификације тла

Класификације тачака које не припадају тлу (Non-ground) класификује оне тачке које имају другу тачку у непосредној близини и друге тачке под стрмим углом силазно. У принципу, класификује оне тачке које не могу бити „гроунд“ тачке зато што је нагиб ка другој тачки веома стрм.Ова рутина упоређује центар сваке тачке са другим тачкама у задатом ху растојању. Ако вертикални угао од једне тачке до центра друге тачке прелази дозвољени лимит, тачка ће бити класификована [2].

Рутина објекти препознаје кровове на основу крове павни у оквиру задатих параметара, који дефинишу грубост кровне равни и одступања и њој. Тачке које представљају кров сврставају се у класи Buildings. За извршавање ове функције неопходно је претходно обавити класификацију тачака терена.

Горе наведене рутуне се углавном групишу у једну комадну тј. у марко програм. Након завршетка класификације неопходно је да се одради визуална инспекција и исправе направљене грешке.

3. Машинско учење

Машинско учење је наука програмирања рачунара тако да имају способност да науче из података. Артур Самуел је дао генералну дефиницију машинском учења и она гласи:

Машинско учење је поље изучавања које даје рачунару могућност да учи без потребе да га експлицитно испрограмирамо

Формалну дефиницију дао је Том Мичел

„За компијутерски програм кажем да учи на основу искуства E које се односе на задатак T и меру перформанси П, уколико се његове перформансе на задатку T, мерене у односу на П, унапређују са искуством E“

Да би илустровали ову дефиницију користићемо чест пример примене машинског учења а то је класификација ручно писаних цифара, што би у нашем случају задатак Т. Да би наш модел научио да класификује одређене писане цифре у одговарајуће класе, неопходно је да му дамо неки примере на основу којих ће он моћи да закључи која је која цифра. Ово примери са којима „хранимо“ алгоритам се зову тренинг подаци, и у горе поменутој дефиницији представља искуство Е. Да би сазнали да ли наш модел учи или не, морамо да уведемо меру перформанси П. То би у нашем случају могла да буде прецизност, тј. однос тачно класификованих цифара и укупног броја цифара [8].

3.1 Употреба алгоритама машинског учења

Како расте популарност алгоритама машинског учења тако расте и могућност њихове примене. У почетку алгоритми су се користили за препознавање цифара, уклањање „спам“ мејлова, просте игрице као што су мице итд.

Данас су алгоритми машинског учења свеприсутни. Почнимо од тога да скоро сав садржај и рекламе на друштвеним мрежама нам управо препоручују ови алгоритми (recommended systems). Системи за откључавање наших уређаја помоћу лица и помоћу отиска прста су имплементирани користећи ове алгоритме. Они се такође користе за препознавање говора, кластеровање, предвиђање цена на берзи, аутоматској вожњи итд.

Што се тиче геоинформационих система, машинско учење се користи у сврхе класификације сателитских снимака, препознавања одређених објеката на снимку и сегментације снимка [9].

Слика 3.1 Општа примена машинског учења (слике горе) и примена у геоинформационим системима

3.2 Подела алгоритама машинског учења

Према начину на који алгоритам учи, он може грубо да се сврста у две категорије

- Надгледано учење
- Ненадгледано учење

3.2.1 Надгледано учење

Надгледано учење је учење у којем алгоритму дајемо податке које садрже жељене класе тј. решење за сваку инстанцу. Свака инстанца има вектор карактеристика (feature vector), који садржи низ нумеричких атрибута за сваки објекат као и решење тј. класа којој сваки објекат припада. Шта алгоритам ту покушава да уради јесте да нађе математичку функцију која ће дати вектор мапирати на право решење. Грубо говорећи алгоритам ово ради тако што му прво дамо неки улаз (вектор карактеристика), он затим израчунава излаз

(класу) на основу својих параметара, и ми тај излаз тј. решење поредимо за решењем које је нам дато. На основу информације да ли се излаз из модела поклапа са датим излазом, ми ажурирамо параметре модела тако да се израчунати излаз и дати излаз поклапају што више.

Типични пример за Надгледано учење је класификација. На пример, хоћемо да класификујемо сателитске снимке у оне које представљају урбано подручје и оне које представљају неурбано подручје. Ту би поред сваког сателитског снимка имали лабелу која означава којој класи он припада. Онда би пустили сателитске снимке кроз алгоритам и за сваки снимак би добили класу коју је алгоритам њему доделио. Затим би поредили да ли се предвиђене класе поклапају са датим класама, и преправили би параметре модела да се што више предвиђених класа поклапа са датим класама.

Још један облик Надгледаног учења јесте регресија. Код регресије ми не покушавамо да предвидимо којој класи нека инстанца припада, већ покушавамо да предвидимо одређену нумеричку вредност која је асоцирана са објектом. Пример за регресију би био, када би на основу података о некој некретнини, њеној величини, старости, локацији, покушали да предвидимо њену цену.

За свако надгледано учење су нам потребне четири ставке:

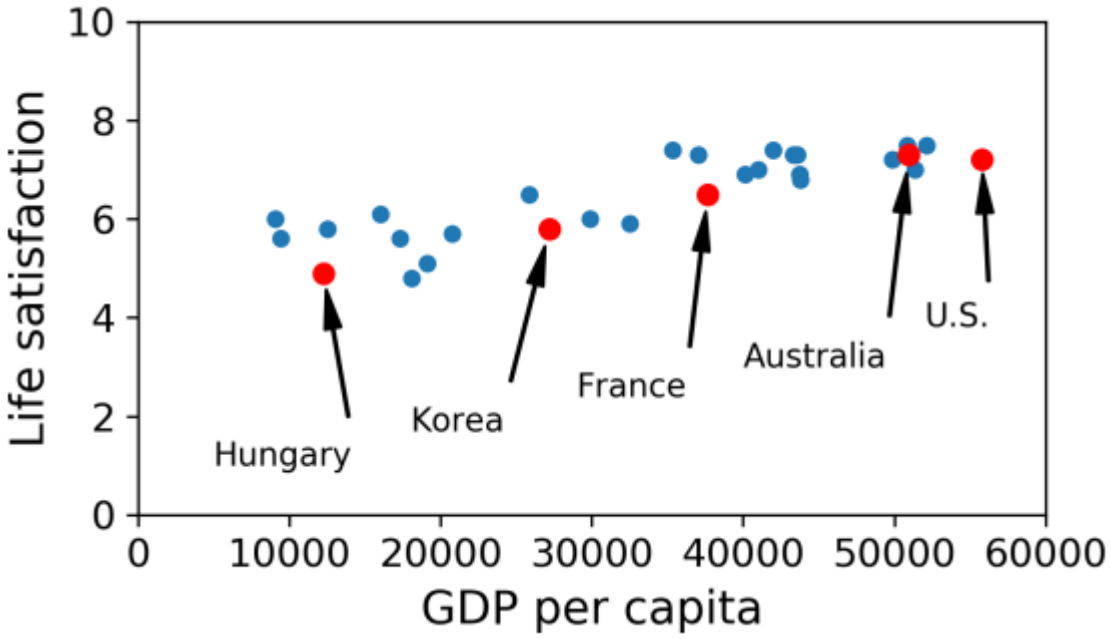
1. Подаци
2. Модел
3. Објективна функција
4. Оптимизациона функција

Подаци које треба да користимо су они какве очекујемо да ће наш алгоритам предвиђати када га будемо користили у пракси. Модел се односи на сам статистички модел који користимо за предвиђање. Објективна функција се користи да измеримо колику грешку прави наш модел, док оптимизациона функција служи да минимизује објективну функцију, и самим тим покуша да нам да оптималнији модел.

Најчешће коришћени алгоритми машинског учења су:

- Линеарна регресија
- Логистичка регресија
- Метода потпорних вектора (*SVM*)
- Стабла одлучивања и Алгоритам случајне шуме
- Неуронске мреже

Најпростији модел надгледаног учења је линеарни модел. Пример који ћемо користити да објаснимо линеарни модел је предвиђање животног задовољства у држави на основу БДП-а. Подаци о овом односу су приказани на слици 3.2.



Слика 3.2 Приказ односа БДП-а и животне сатисфакције

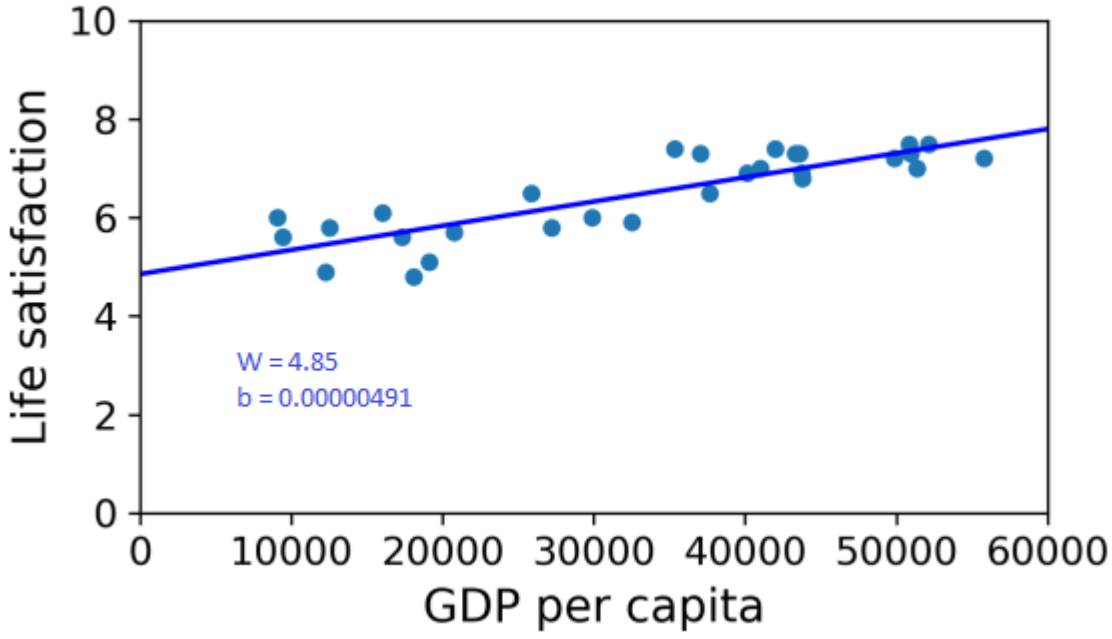
Хоризонтална оса представља БДП државе по глави становника а вертикална оса представља животно задовољство. На графику можемо да увидимо генерални тренд, где што је држава богатија то су њени становници сретнији. Овај тренд ћемо покушати да моделујемо линеарним моделом тј. покушаћемо да нађемо линију која га најбоље осликава.

$$\text{животно задовољство} = w * \text{БДП} + b$$

Једначина 3.1 Прост линеарни модел

Променљива w на говори какав је однос између БДП-а и животне сатисфакције тј. за повећање БДП за један долар колико ће се животна сатисфакција променити. Уколико је w позитиван број,то значи да нама модел говори да за пораст у БДП-у расте и животна сатисфакција. Супротно уколико би променљива w била негативна,наш модел би рекао да са порастом у БДП-у опада животна. сатисфакција. Параметар w се зове још и нагиб. Променљива b представља вредност одсечка тј. вредности коју додајемо на сваки производ параметра w и вредности БДП-а. Одсечак служи да помери криву горе или доле.

Линија која се најбоље осликава тренд међу датим подацима , приказана је на слици 8. На њој су такође приказани параметри који најбоље осликавају тренд у подацима.



Слика 3.3 График простог линеарног модела

Када смо добили овај модел, можемо га искористити за предвиђање животне сатисфакције неке државе. Потребно је само да знамо колики је њен БДП по становнику.

Ово је био илустративни пример линеарне регресије, зарад бољег разумевања надгледаног учења. Додатно објашњење линеарне регресије, како се дефинише колико је који модел добар и како се долази до најбољег модела ће бити објашњено у наставку рада.

3.2.2 Ненадгледано учење

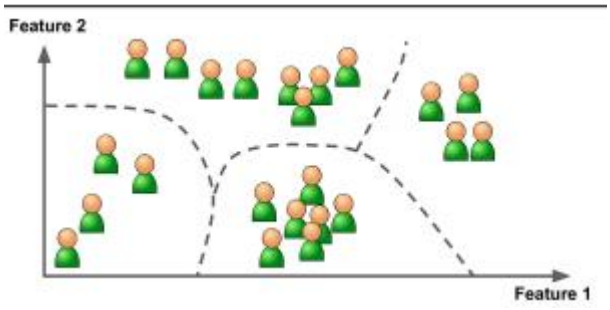
Највећа мана код Надгледаног учења јесте то што нам подаци са ознакама често нису доступни а и сам процес ручног означавања је често мукотрпан. Овде наступају алгоритми ненадгледаног учења. Алгоритми ненадгледаног учења не захтевају податке који су означени. Ови алгоритми проучавају везе између података и на основу ових везе и на основу онога што се од алгоритма тражи он доноси одлуку.

Алгоритми Ненадгледаног учења могу да се грубо поделе у четири целине:

1. Кластеровање
2. Редукција димензионалности
3. Визуализација
4. Детекција аномалија

Кластеровање представља груписање сличних инстанци у једну групу тј. кластер. Циљ ових алгоритама је да минимизују сличности унутар кластера, а максимизују разлику између инстанци из различитих кластера.

Класичан пример кластеризације је разврставање купаца у различите сегменте на основу њихове сличности.



Слика 3.4 Пример кластеризације купаца на основу њихове сличности

Ови алгоритми углавном користе неку меру сличности унутар кластера и њу покушавају да минимизују.

Разлика између кластеровања и класификације јесте то што код класификације имамо тачно предодређени број класа у које инстанце треба да сврстамо, док код кластеризације то није случај. Када извршимо кластеризацију може да се деси да алгоритам пронађе групе и конекције које ми нисмо уопште очекивали, и то може да нас доведе до корисних закључака у вези са подацима

Типични представници ових алгоритам су:

• DBSCAN алгоритма

• KMeans алгоритам

• Хијерархијска стабла

Редукција димензионалности представља поступак коме је циљ да смањи комплексност података без да изгубимо много информација. Један начин да се ово постигне јесте да спојимо више атибута који су међусобно зависни у један атрибут. Смањивање комплексности података је корисно ако се користи као корак обрађивања у надгледаном машинском учењу. Ови алгоритми убрзавају следујуће алгоритме, али такође, штеде меморију а могу и да уклоне сметње из података.

Редукција димензионалности се може такође користи за визуализацију података. Уколико све атрибуте сведемо на два (евентуално 3), њихов однос можемо да представимо у декартовом координатном систему и да на основу тога извучемо додатне закључке.

Детекција аномалија представља алгоритам чија је сврха да детектује инстанце које се пуно разликују од „нормалних инстанци“. Алгоритам ради тако што му покажемо нормалне инстанце и он научи шта може да очекује, и када дође инстанца која се пуно разликује он може то да препозна. Ови алгоритми су корисни код праћења трансакција да види које су трансакције сумњиве и потенцијалне преваре, а такође може да се користи као корак обраде који би избацио грубе грешке из података.

3.3 Дефинисање модела машинског учења и селекција модела

3.3.1 Објеткивне функције

Да би измерили колико је поклапање између предвиђања која је наш модел израчунао и датих решења морамо да дефинишемо функцију која се зове објективна функција. Одабир објективне функције зависе од типа проблема који имамо. Оно што је важно да напоменемо јесте да свака објективна функција треба да буде диференцијабилна и непрекидна. Уколико имамо регресиони проблем најчешћи избор за објективну функцију је средња квадратна грешка.

$$J = \sqrt{\frac{\sum_{i=1}^{n}((w * x + b) - y)^2}{n}}$$

Једначина 3.2 Средња квадратна грешка

Функција J означава функцију губитка. Параметар w означава тежине које су асоциране са сваким атрибутом, b представља додадтну вредност а у представља лабеле које су асоциране са сваком инстанцом x.

Уколико имамо бинарни класификациони проблем користићемо функцију која се зове бинарни укрштени ентропски губитак.

$$J = -\frac{1}{n}\sum_{i=1}^{n} y_i * \log(h_i) + (1 - y_i) * \log(1 - h_i)$$

Једначина 3.3 Бинарни укрштени ентрописки губитак

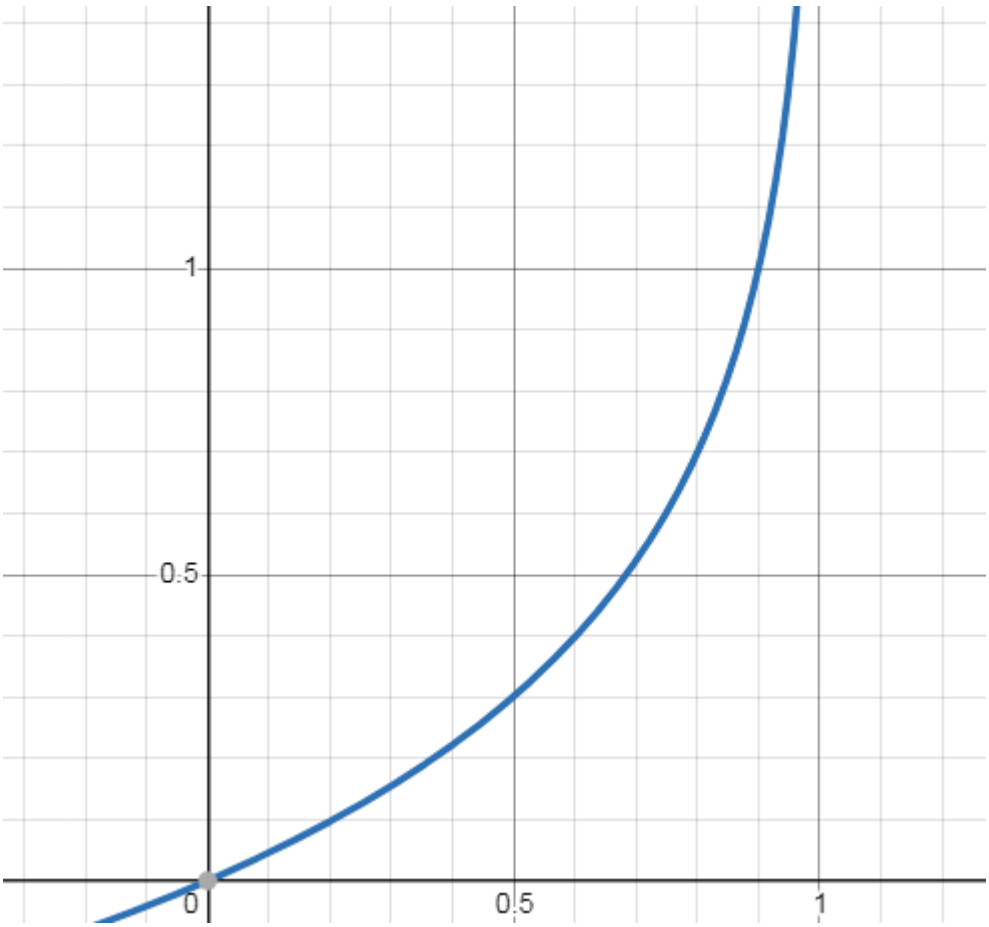
Променљива y представља лабелу за сваку инстанцу и она има вредности 0 или 1. Функција h представља предвиђање за сваку инстанцу и њене вредности су у распону од 0 до 1 и представља вероватноћу да инстанца припада класи од интереса (ово предвиђање је објашњено у поглављу Логистичка регресија). Објективна функција би била једнака нули уколико би за сваку инстанцу, y_i било једнако h_i. (уколико је y_i један,функција h_i идеално би дала вероватноћу од 1.0 да инстанце припада класи од интереса).

Разлог зашто користимо негативан логаритам је следећи. Уколико би вредност функције h била 1, логаритам те функције је био нула, док уколико би вредност функције била 0, логаритам би тежио бесконачности.

Слика 3.5 Функција log(h)

Ова особина нам је корисна зато што, ако је y вредност 1, а нама је алгоритам предвидео управо јединицу, грешка као што је и логично би била нула. Уколико би пак алгоритам предвидео 0, наша грешка би тежила бесконачности. Управо ова велика казна за погрешна предвиђања је разлог зашто користимо негативан алгоритам.

Ово се односило на први сабирак у суми, тј. случај када је y једнако јединици. Уколико је y једнако 0 први сабирак је небитан (исто као што је небитан други сабирак када је y једнак јединици), и гледа се само други сабирак. Логика за други сабирак је иста као и за први само што логаритамску функцију пресликамо у односу на вредност 0.5. Дакле ако је је y вредност 0, и вредност функције h такође нула грешка ће бити була, док ако је и вредност функције h тежи јединици, грешка ће тежити бесконачности.



Слика 3.6 Функција log(1-h)

Генерализација бинарног укрштеног губитка је укрштени ентрописки губитак који се користи за израчунавање губитка када треба предвиђамо K класа. ($K > 2, K \in N,$)

$$J = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_k \log(h_k)$$

Једначина 3.4 Укрштени ентрописки губитак

Горе наведене објективне функције су функције губитка (*loss function*). Функције губитка дају велику вредност за моделе који даје лоша предвиђања, а малу вредност за добра предвиђања.

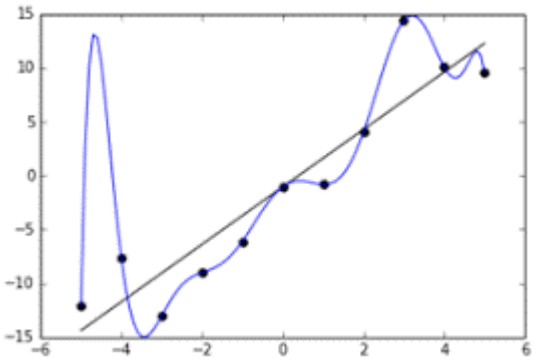
3.3.2 Изнадподесност

У претходном поглављу смо поменули функцију губитка и рекли смо да што су предвиђања боља нама је функција губитка већа и обрнуто. Да ли је онда добро да наша функција губитка тежи нули, односно да дамо коректно предвиђање за сваку инстанце? Не нужно. Уколико имамо јако мали губитак на подацима на којима смо тренирали наш модел, то врло вероватно значи да је наш модел изнадподесан.

Изнадподесност (*overfitting*) је концепт у науци о подацима који се дешава када статистички модел егзактно уклопи тренинг податке. Када се ово догоди алгоритам нажалост не може да да добра решења за податке који никад раније није видео и самим тим функција коју алгоритам треба да обавља је компромитована.

У самом процесу тренирања, модел користи тренинг сет како би подесио своје параметар зарад што боље генерализације тј. предвиђања на новом скупу података. Међутим, уколико наш алгоритам тренира превише дуго, или је модел превише комплексан, модел почиње да учи „буку“ у подацима. Под буком у подацима се сматра вредности који не представљају праве особине података, већ су резултат насумичности.[29]

На слици 3.7 црном линијом је приказан тренд у подацима док плава функција представља изнадподесну фунцкију.



Слика 3.7 Изнадподесна функција

Како уопште можемо да знамо да је ли је наш модел „оверфитовао“? Начин на који то можемо да сазнамо јесте тако што ћемо алгоритму дати податке које никада пре није видео. Уколико нам је тачност резултата на тренинг подацима велика, а тачност на тест подацима врло мала, то је највероватније знак да смо оверфитовали наш модел.

Податке које модел никад није видео углавном добијамо тако што пре самог подешавања модела, цео наш скуп података поделимо на два скупа, један који ћемо користити за тренирање, и један скуп који ћемо користити да тестирамо наш модел. Тестирање нам даје резултате, тј. квалитет предвиђања какав можемо да очекујемо када будемо користили модел у пракси. Тестирање над тест подацима треба да одрадимо само једном. Уколико бисмо тестирали модел на тест подацима више пута и изабрали модел који прави најмању грешку на тест подацима, онда би тачност модела коју смо добили била пристрасна тј. представљало би модел бољим него што то заиста јесте. Разлог томе јесте то што смо ручно изабрали модел са датим параметрима који одговарају баш том тест сету.

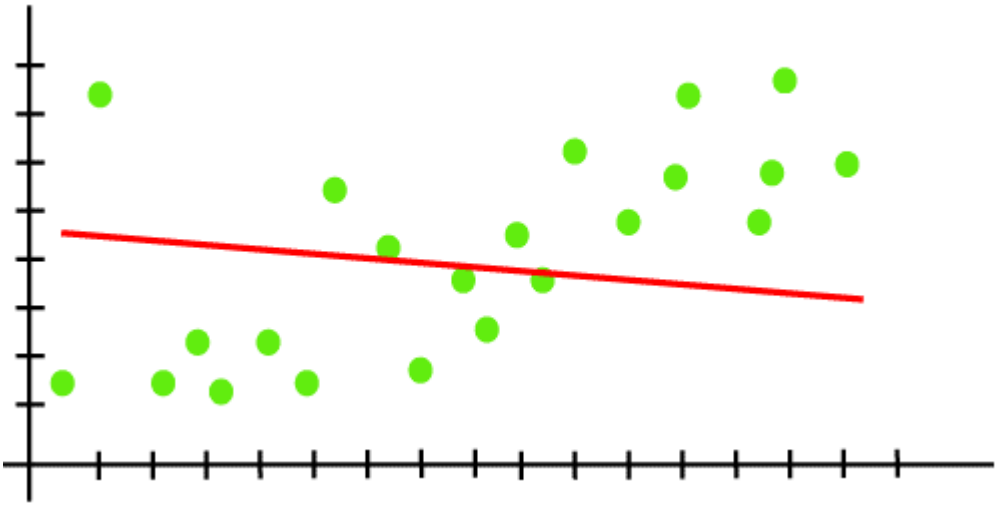
Управо због овога уводимо још један скуп података који се зове валидациони скуп података. Валидациони скуп података добијамо тако што из тренинг сета узмемо одређени број података. У самом процесу селекције одговарајућег модела, и тражењу оптималних

параметара, користимо тренинг податке за тренирање, а валидационе податке користимо да проверимо да ли смо оверфитовали, као и то да видимо који статистички модел са којим параметрима је најбољи за наш задатак.

Начин на који ћемо поделити наше податке у подскупове у многоме зависи од самог задатка машинског учења, али и од тога какве податке имамо. Опште правило је да узмемо од 60 до 80 посто података за тренирање, а остатак равномерно расподелимо за валидацију и тестирање.

3.3.3 Исподподесност

Исподподеснот (*underfitting*) представља феномен када наш алгоритам прави велику грешку и на тренинг и на тестинг подацима. Ово се често дешава када је наш модел превише прост, када модел има превише регуларизације, или када подаци немају довољно корисних атрибута.



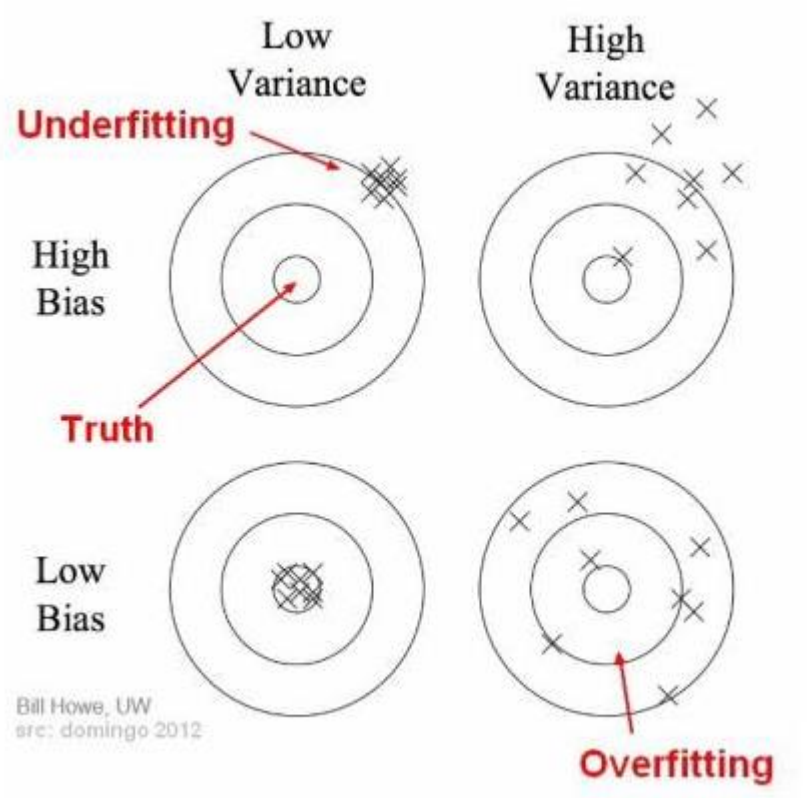
Слика 3.8 Пример исподподесности модела

Ундерфитовање је лакше да се примети зато што су потреби само тренинг подаци да би га дијагностиковали.

3.3.4 Компромис између грешке и варијансе

Грешка (*bias*) у машинском учењу представља просечну разлику између нашег предвиђања и тачних резултата. За моделе са великом грешком кажемо да су исподподесни, јер не успевају нађу патерне у подацима. Варијанса са друге стране представља варијабилитет у предвиђању података. Модели са великом варијансом обраћају превише пажње на тренинг податке и не успевају да генерализују предвиђања на новим подацима тј. наш модел је изнадподесан.

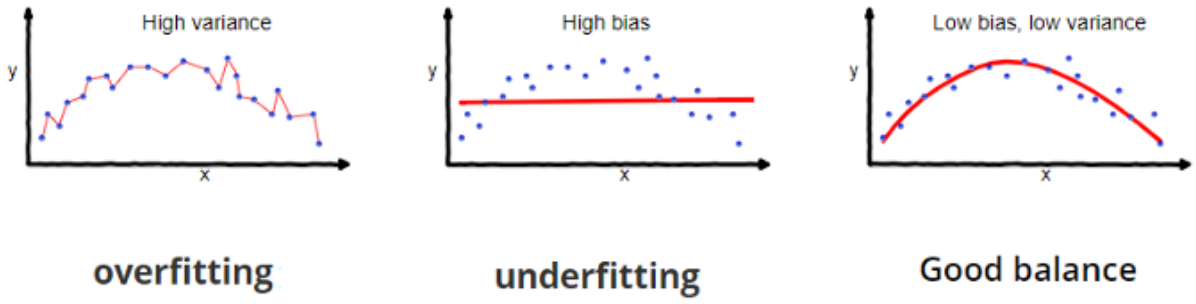
Слика 3.9 приказује пример изнадподесности и исподподесности на примеру гађања мете. Идеалан случај је приказан на доњој левој мети, када су наши хитци тачни а и поновљиви. Доња десна мета приказује хитце коју су релативно тачни али нису прецизни односно поновљили, те је то метафора за изнадподесност (велика варијанса у предвиђању). Горња десна мета је метафора за исподподесност, зато што хитци не успевају да погоде центар.



Слика 3.9 Приказ изнадподесности и исподподесности

Ако је наш модел превише прост и ако има премало параметара онда ће он имати велику грешку а малу варијансу. Са друге стране, ако је наш модел превише комплексан онда ће наш модел имати малу грешку али ће зато имати велику варијансу. Дакле, ми морамо да нађемо најбољи однос између изнадподесности и исподподесноти. Тај компромис се огледа у тражењу адекватне комплексности самог модела [31].

На слици 14 видимо три случаја која сусрећемо када тренирамо наш модел. Лева слика приказује случај када оверфитујемо наш модел. Слика у средини приказује случај када наш модел није научио из података довољно добро и има велику грешку, док десна слика приказује оптималан модел.



Слика 3.10 Пример изнадподесности, изподподесности и оптималног модела

Уколико модел андерфитује, постоји неколико ствари које можемо да урадимо. Најочигледнија ствар која се може урадити јесте да изаберемо комплекснији модел, који ће моћи боље да научи патерн у подацима. У случајевима када имамо регуларизацију на нашем моделу, она се може смањити или укинути (Види Регуларизација). Такође могу да се додају додатни атрибути нашим подацима, избаце непотребни атрибути, или исправе грешке у подацима. На основу ових промена у подацима модел би могао лакше да научи патерн. Треба имати на уму да уколико је модел исподподесан, додавање додатних података неће помоћи моделу, и било би само губљење времена.

Са друге стране, ако је познато да смо оверфитовали модел, шта може да се учини да се то поправи? Најочигледнији начин јесте да се престане са тренирањем чим алгоритам почне да учи буку у подацима. Код овог начина треба да се пази да се тренирање не заустави превише рано и због тога да се не стигне до оптималног модела.

За разлику од случаја када андерфитујемо модел, код оверфитовања додатни подаци би нам добро дошли. Алгоритму је дато више података које мора да уклопи и самим тим и вероватноћа да ће у пракси видети инстанцу какву пре није видео је доста мања. Овде је доста битно да се унесу подаци који су тачни и који осликавају патерн, иначе овај метод не може да помогне.

Још један од популарних начин јесте Ансамбл учење. Код овог учења алгоритам тренира више модела, и изводи предвиђање на основу комбинације предвиђања свих модела. Алгоритам насумичне шуме је један од најчешће коришћених ансамбл учења. Он комбинује предвиђања са више стабала одлучивања, и због тога овај модел је отпорнији на оверфитовање (Стабла одлучивања и алгоритам насумичне шуме).

3.3.5 Регуларизација

Регуларизација је један од најчешће коришћених начина да избегнемо изданподесност. Регуларизација представља ограничавање нашег модела како би се он учинио једноставнијим и како би се смањио ризик од оверфитовања. Начина на који је модел регулисан, зависи првенствено од модела који користимо као и од резултата које очекујемо. Различите методе регуларизације могу да дају потпуно другачија решења.

Најчешћи начина на који модел може бити регулисан јесте да се дадају додатне информације у функцију губитка. Додатне информације представљају вредности параметара модела тј. његових коефицијената. Пошто је сам циљ тренирања да се минимизује функција губитка, алгоритам ће учинити да коефицијенти буду што мањи. Количину регуларизације коју ћемо додати функцији губитка, контролишемо са хиперпараметрима модела.

Два начина за регуларизацију регресионих модела јесу гребенска регресија и ласо регресија (Ridge regression and Lasso regression).

Гребенска регресија (такође се зове Л2 норма) додаје квадратну магнитуду коефицијената на функцију губитка.

$$J = \sqrt{\frac{\sum_{i=1}^{n}((w*x+b)-y)^2}{n}} + \frac{\lambda}{2} * \sum_{i=1}^{n} w_i^{\,2}$$

Једначина 3.5 Средња квадратна грешка са Л2 нормом

λ (Ламбда) параметар контролише колико регуларизације додајемо. Уколико је параметар једнак нули, онда практично и немамо регуларизацију. Уколико је λ параметар превелик, коефицијенти ће тежити нули, и самим тим ћемо андерфитовати модел. Кључна ствар је да нађемо баланс између изнадподесноти и исподподесности.

Ласо регресија (Lasso-Least Absolute Shrinkage and Selection Operato) додаје апсолутну вредност магнитуде коефицијената на функцију губитка.

$$J = \sqrt{\frac{\sum_{i=1}^{n}((w*x+b)-y)^2}{n}} + \lambda * \sum_{i=1}^{n} |w_i|$$

Једначина 3.6 Средња квадратна грешка са Л1 нормом

Овде се λ параметар понаша исто као и у горе наведеном примеру [32].

У обе методе је веома важно да се подаци стандардизују, како би се довели сви коефицијенти на исту скалу и како би сваки коефицијент имао једнак утицај на функцију губитка. (Види Предпроцесирање података).

Главна разлика између гребенске и ласо регресије јесте то што ласо регресија сведе коефицијенте атрибута који не доприносе регресији на нулу, док гребенска регресија своди коефицијенте близу нуле, али никад тачно на нулу. Тако да ласо регресија поред тога што ради рестрикцију модела, врши и селекцију параметара тако што оне које нису битни сведе тачно на нулу. Ово је доста корисна особина када нам је потребо не само предвиђање, већ и интерпретација решења и информација о томе колико је који параметар битан. Поље на којем се ова регресија често користи јесу биолошка истраживања. Када биоложи раде предвиђање одређених процеса у човековом телу, није им у интересу само да изврше предвиђање, већ им је у интересу и да виде који су гени утицали на тај процес.

Мана ласо регресије јесте то што селекција променљивих које су битне а које не, доста зависи од података са којима радимо, стога сама селекција може да буде нестабилна.

У пракси не можемо у напред да знамо које од ове две методе регулариације ће бити боља за наш модел. Препоручује се ако имамо времена и ресурса да пробамо обе методе, и видимо која даје боље резултате, или чак можемо да комбинујемо ове две методе .

Метода регуларизација која је комбинује ове две методе се зове Еластична мрежа (Elastic net).

$$J = \sqrt{\frac{\sum_{i=1}^{n}((w*x+b)-y)^2}{n}} + \lambda(\alpha * \sum_{i=1}^{n}|w_i| + \frac{1-\alpha}{2} * \sum_{i=1}^{n} {w_i}^2)$$

Једначина 3.7 Средња квадратна грешка са пенализацијом еластичне мреже

Еластична мрежа преко (алфа) параметра контролише баланс између гребенске регресије и ласо регресије. Уколико је параметар један нули, наша регресија ће бити као грбенеска регресија, а уколико је једнака јединици, регресија ће се понашати као ласо регресија.

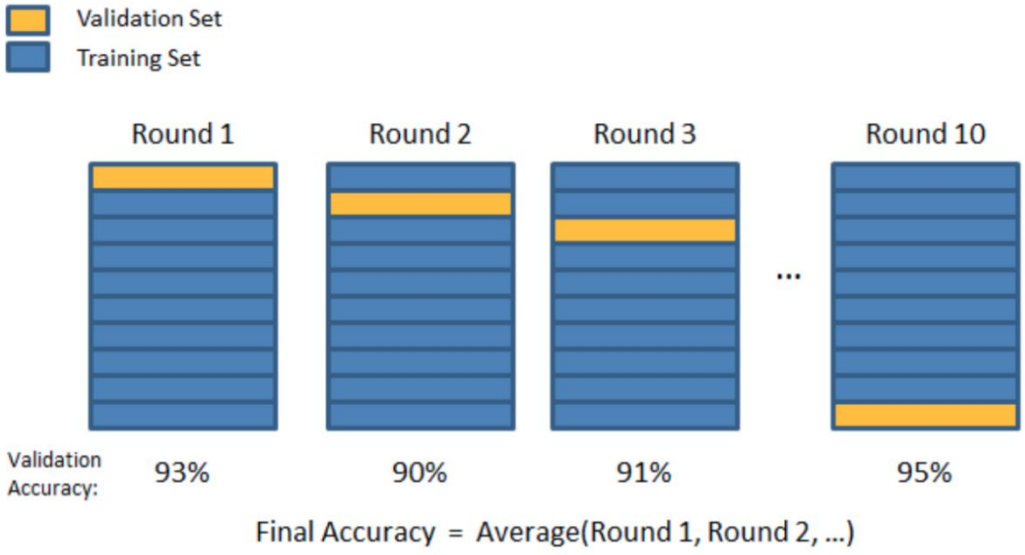
3.3.6 Укрштена валидација и селекција модела

Укрштена валидација (Cross-validation) је техника у машинском учењу која се користи за проверу како се статистички модел генерализује на независне подскупове података. Ова техника ради тако што тренира модел на неколико подскупова а тестира податке на подскупу који још није видео[33].

Традиционални начин на који се ради валидација модела, јесте да се од тренинг података одбоји одређени проценат података, и када завршимо тренирање, валидацију одрадимо на овом сету података. Међутим, резултати који добијемо тестирањем на валидационом сету, нису у потпуности репрезентативни. Резултате које добијемо, у многоме зависе од начина на који смо поделили податке, да ли смо мешали податке, колико

је валидациони подскуп сличан тренинг подскупу. Такође, циљ нам је да искористимо што више података за тренирање, а ако би морали још да одвајамо податке за валидацију, то би додатно одузело податке из нашег скупа за тренирање.

Укрштена валидација је уведена да надокнади ове недостатке традиционалног начина валидације модела. Укрштена валидација ради тако што подели све податке у К подгрупа (К је природни број, најчешће 3,5 или 10). Када је поделио податке он тренира модел на К-1- ој подруги података, а оставља једну подгрупу за тестирање. Овај поступак понавља К пута, у свакој итерацији користимо другу подгрупу за тестирање, све док свака подгрупа не буде искоришћена. Након сваке итерације бележимо резултате нашег модела и када завршимо са тренирањем свих модела, узмемо средњу вредност од наших резултата и то ће бити коначан резултат. Овај резултат можемо поуздано да користимо за процењивање тачности нашег модела.



Слика 3.11 Приказ процеса крос валидације

Када је завршена корс валидација, треба још једном истренирати модел над свим подацима. Разлог овоме јесте то што сваки модел који смо тренирали у крос валидацији користио је К-1 подгрупу, а нама је циљ да користимо модел који је трениран на максималном броју података.

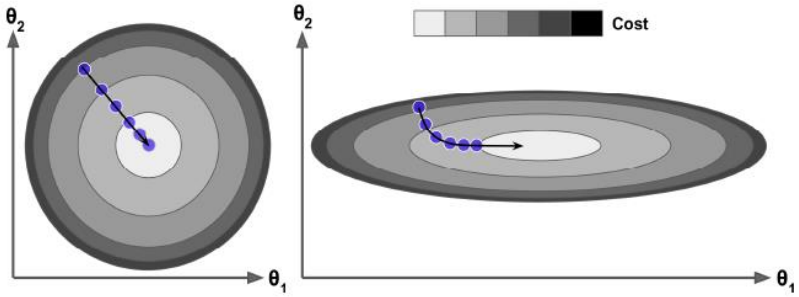
Укрштена валидација је изузетно корисна када се ради селекција параметара за наш модел. Селекцију параметара радимо тако што се изаберу параметри који су сматрани да би били погодни за наш модел. Када смо изабрали параметре тренирамо наш модел помоћу укрштене валидације. Понављамо овај поступак жељени број пута, и на крају изаберемо параметре који су дали најбоље резултате.

3.3.7 Предпроцесирање података

Скалирање података је једно од најважнијих корака у препроцесирању. Оно је посебно битно за алгоритме који рачунају раздаљину у метричком простору између података. Пошто подаци често имају другачије скале њихов утицај на саму рачуницу растојања није исти (сликовити пример је разлика између једног килограма и 1000 грама, суштински ове две вредности за нас су исте, али за алгоритам који рачуна дистанце у метричком простору нису). Стога, подаци морају бити доведени на исту скалу, да би сви атрибути имали подједнаки утицај.

Још једна битна ствар коју нам скалирање омогућава јесте да значајно убрзамо време које је потребно за тренирање података, поготово за алгоритме који користе градијенти пад. Значај скалирања Разлика је приказана на слици 3.12. Обе функције представљају функцију губитка у параметарском простору и обе зависе од параметара θ_1 и θ_2. Разлика у облику ове две функције јесте то што су подаци са леве стране скалирани, а подаци са десне нису. Атрибут који је асоциран са θ_1 је доста мањи у поређењу са атрибутом који је асоциран са θ_2. Због тога сам параметар θ_1 мора да буде већи да би атрибут асоциран са θ_1 имао утицај.

Ова разлика у облику функције губитка је разлог зашто алгоритам спорије тражи параметре глобалног минимума. Ово је поготово значајно за Неуронске мреже.



Слика 3.12 Функције губитка у параметарском простору

Два начина да одрадимо скалирање јесте да нормализујемо податке и да их стандардизујемо.

Нормализација представља када све вредности пребацимо у неки фиксни интервал, обично између нула и један тј. између минус један и један. Један од начина је да од вредности атрибута одузмемо његову најмању вредност, а затим поделимо са највећом вредношћу.

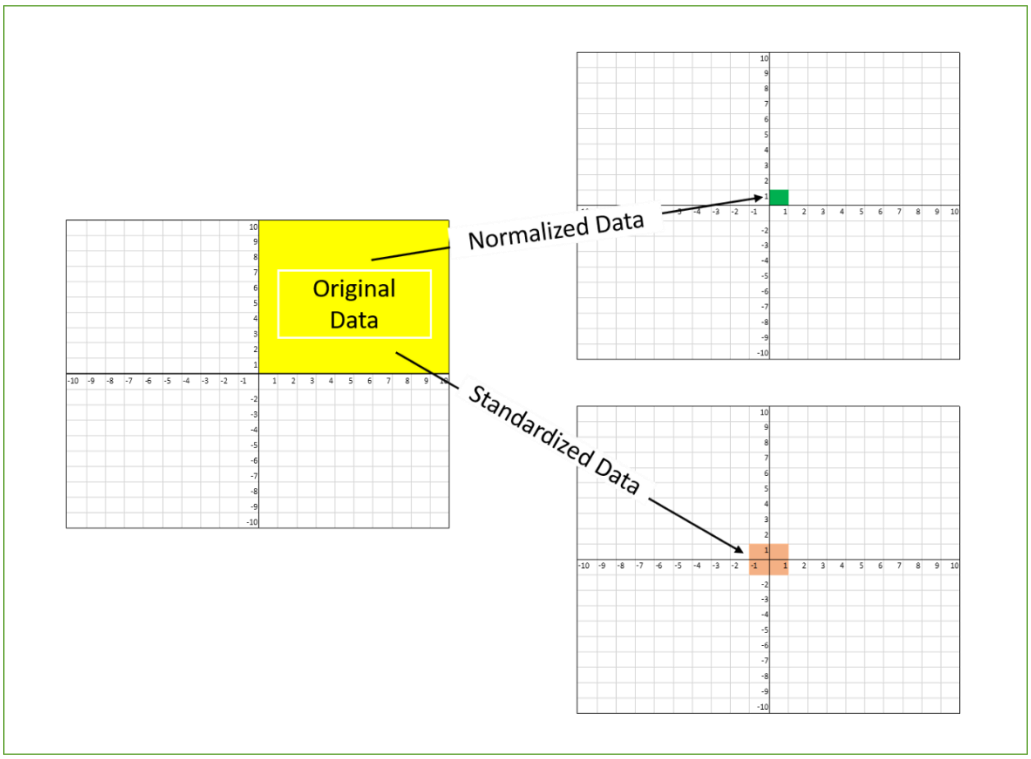
$$X_{norm} = \frac{X - X_{min}}{X_{max}}$$

Једначина 3.8 Нормализација података

Стандардизација је термин који је позајмљен из статистике. Да би се подаци стандардизовали, прво се од сваке вредности атрибута одузме средња вредност тог атрибута, а затим се подели са стандардним одступањем. Ово има ефекат да су подаци центрирани на нули и да имају стандардну девијацију која износи 1. За разлику од нормализације интервал у коме се налазе вредности није фиксан. Када вршимо стандардизацију претпостављамо да су подаци нормално распоређени.

$$Z = \frac{X - \mu}{\sigma}$$

Једначина 3.9 Стандардизација података



Слика 3.13 Разлика између нормализације и стандардизације података

3.4 Модели надгледаног учења

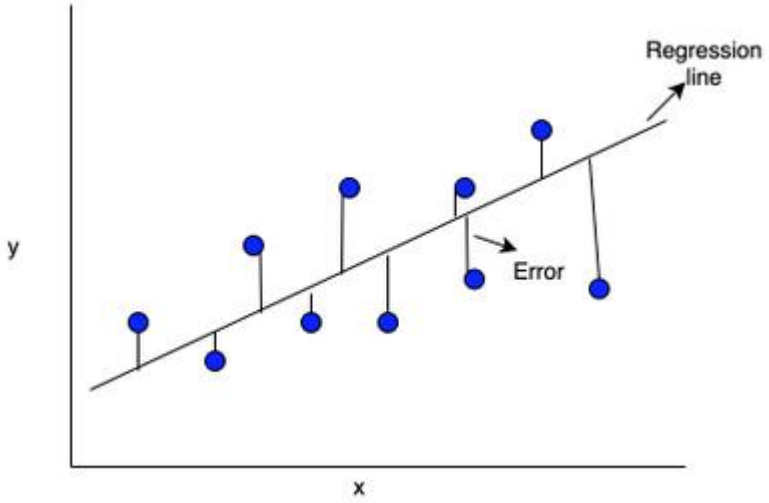
3.4.1 Линеарна регресија

Линеарна регресија је најосновнији модел надгледане регресије. Циљ линеарне регресије је да се предвиди вредност једне променљиве на основу осталих променљивих. Променљива коју желимо да предвидимо се зове зависна променљива, а променљиве на основу којих извршавамо предвиђање зову се независне променљиве [24].

Претпоставка коју линеарна регресија прави када је у питању моделовање односа између зависних и независних променљивих јесте да је однос линеаран. Такође се претпоставља да је однос статистички а не детерминистички. Разлика између ове две

претпоставке јесте да код детерминистичког односа тачно можемо да знамо зависну промењиву на основу независне. На пример уколико знамо угловну вредност у радијанима директно можемо да добијемо угловну вредност у степенима. Статистичка однос између две варијабле није егзактан тј. не може сто посто тачно да се одреди. Пример за овакав однос јесте однос између БДП-а државе и животног задовољства, где не можемо да предвидимо егзактну вредност, али можемо да предвидимо приближну вредност.

Циљ линеарне регресије јесте да нађу параметри који најбоље моделују однос између независних и зависних променљивих. Најбољи значи да је укупна грешка између предвиђених и датих вредности за све варијабле буде минимална. Грешка се рачуна тако што се одреди квадратно растојање између инстанце и регресионе линије за све инстанце и та квадратна растојања се сумирају (Средња квадратна грешка).



Слика 3.14 Приказ линеарне регресије. Линије између плавих тачака и регресионе лине представљају величину грешке за сваку тачку

Формула за линеарну регресију је приказана је у једначини 3.10

$$y = w * x + b$$

Једначина 3.10 Основна једначина линерног модела

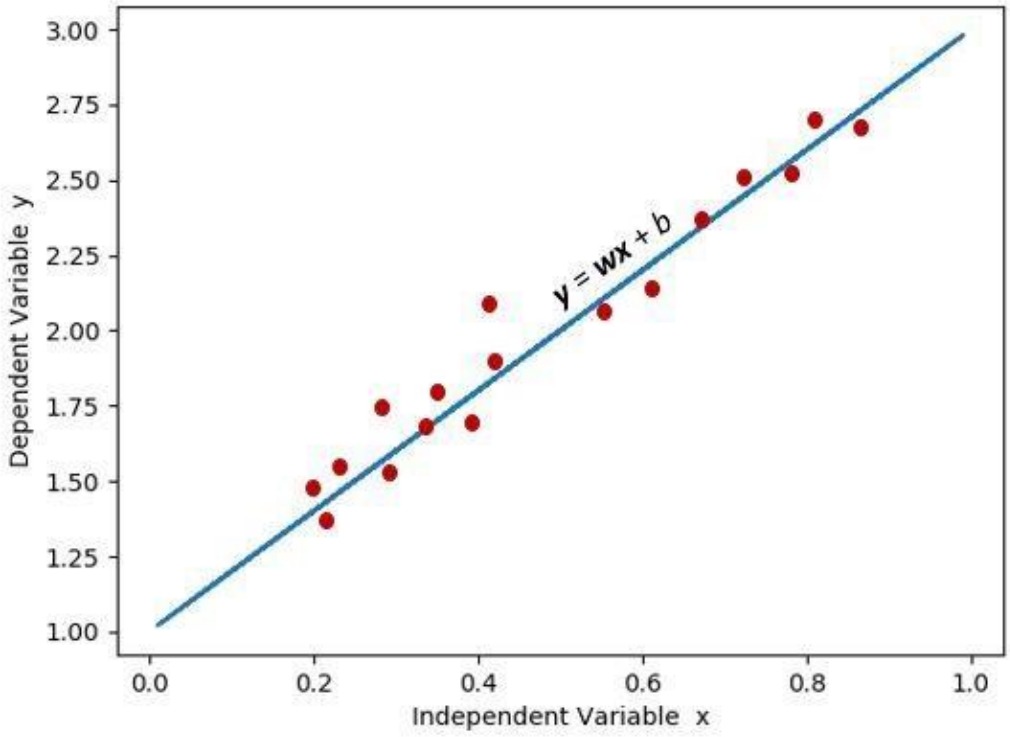
Елементи у овој једначини представљају следеће:

• Y представља израчунати излаз тј. Зависну променљиву

• W представља параметре модела, тј. тежине на основу којих алгоритам врши предвиђања

• X представља улазне параметре односно векторе карактеристика (независне променљиве)

• Грешка (*bias*) , она нам омогућава да повећамо или смањимо предвиђену вредност за фиксну вредност



Слика 3.15 Линеарни модел

Један од начина на који тражимо оптималне параметре за линеарну регресију јесте преко једначине нормале. Једначина нормале јесте алгебарски израз који када израчунамо добијамо оптималне параметре.

$$w = (X^T X)^{-1} X^T Y$$

Једначина 3.11 Нормална једначина

Мана овог приступа јесте то што је рачунски јако комплексан и дуготрајан. Када имамо велику количину тренинг података, се великом бројем атрибута, једначина нормале постаје неупотребљива. Разлог томе јесте што једначина нормале укључује инвертовање матрице коваријансе X, а инвертовање само по себи је дуготрајан процес за матрице већих димензија.

Алгоритам који се углавном користи за линеарну регресију, као и за много друге оптимизационе проблеме јесте градијентни пад, који итеративно поправља параметре линеарне регресије док се не стигне до оптималног решења.

Поред претпоставке да је однос између зависне и независних променљивих линеаран, линеарна регресија такође претпоставља да не постоји грешки у независним променљивим, да су независне променљиве заиста независне и да нису у међусобној корелацији као и да су грешке независне и да имају константну варијансу. Ове претпоставке резултују у томе да модел није најсофистицираније и не даје најбоље резултате за комплексне сетове података

Линеарна регресија је релативно прост модел и омогућава нам да лако интерпретирамо његова решења. Такође нам омогућава релативно брзо тренирање и предикцију. Линеарна регресија се користи у разним академским и пословним сферама. Користи се у биологији, психологији, економији итд [24]. Поред тога што се користи за предвиђање, линеарна регресија такође може да се користи да објасни неке феномене, као и да квантификује односе између њих [26].

3.4.2 Логистичка регресија

Упркос томе што у имену стоји регресија, логистичка регресија је заправо статистички модел који се користи за класификацију. Логистичка регресија је веома слична линеарној регресији. Главна разлика је у томе што је зависна променљива категоричка променљива а не нумеричка.

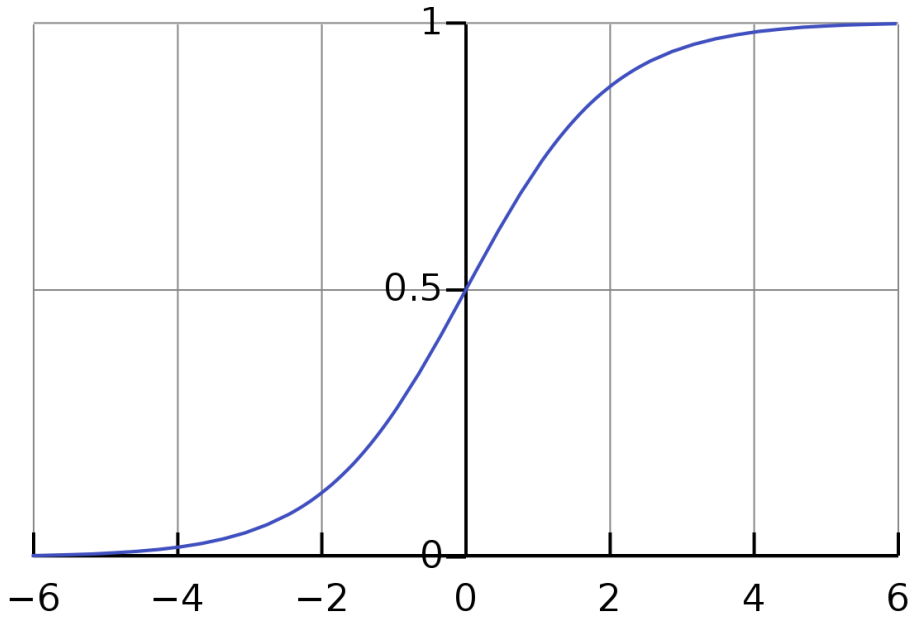
Уколико се ради о бинарној класификацији, да ли је сателитски снимак представља урбано подручје или не, могуће вредност за y су 0, сателитски снимак не представља урбано подручје, или 1 сателитски снимак представља урбано подручје. Е сада шта треба да учинимо да би алгоритам избацио нулу или јединицу? Једно од решења јесте да избацимо вероватноћу да ли дата инстанца припада једном објекту и да ако је та вероватноћа већа од неког прага, онда кажемо да та инстанца припада тој класи тј. доделимо јој вредност 1. На нашем примеру то би рекли овако: уколико је вероватноћа да снимак представља урбано подручје већа од 0.5 (најчешћи праг), сврстај дати снимак у класу која представља урбано подручје тј. додели јој вредност 1.

Из математичке вероватноће знамо да се вредности за вероватноћу крећу између 0 и 1. Да би било које у мапирали у интервал који је ограничен нулом и јединицом, користићемо сигмоид активациону функцију.

$$h = \frac{1}{1 + e^{-z}}$$

Једначина 3.12 Сигмоидна једначина

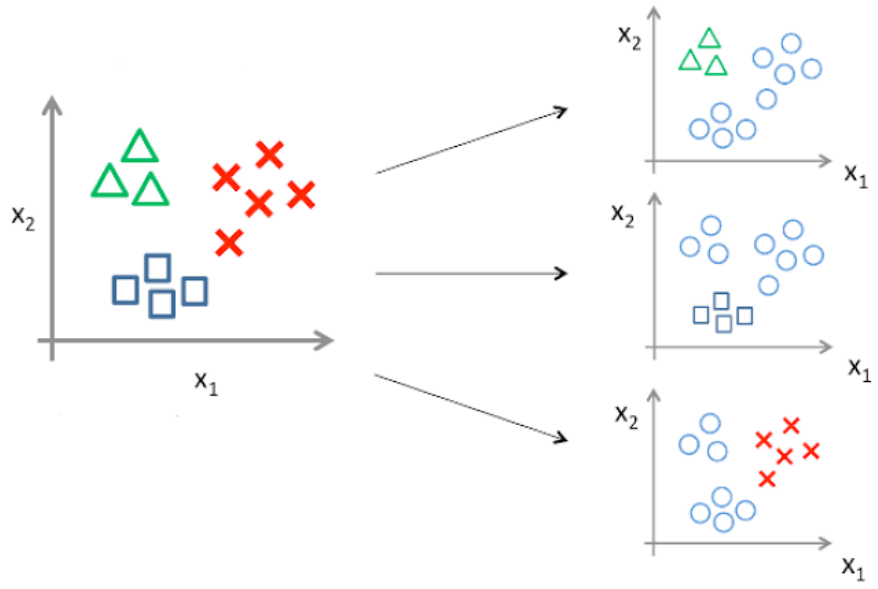
Једначина 3.12 сузбија израчунату вредност функције Z на интервал између нуле и јединице. Функција Z је иста као Основна једначина линерног модела, дакле израчунава матрични производ тежина и вредности података и додаје грешку. Уколико добијемо негативну вредност за Z, вредност сигмоидне функције ће бити мања од 0.5 и због тога ћемо имати негативно предвиђање, а уколико имамо позитивну вредност функције Z, вредност сигмоидне функције ће бити већа од 0.5 и стога ћемо имати позитивно предвиђање. Овај однос функције Z и сигмоидне функције је приказан на слици 3.16.



Слика 3.16 Сигмоид активациона функција

Логистичка регресија користи Бинарни укрштени ентрописки губитак као објективну функцију. Бинарна укрштени ентрописки губитак је погодан за логистичку регресију зато што је функција губитка конвексна и самим тим омогућава брзо и сигурна налажење глобалног оптимума.

За сада смо објаснили како функционише логистичка регресија када је у питању бинарна класификација, али како можемо да искористимо овај модел када треба да класификујемо податке у више од две класе? Тада можемо да користимо метод један против свих. Суштина метода један против свих јесте да ако имамо N класа тренирамо N различитих логистичких регресија. Поступак тренирања креће тако што узмемо једну класу j и њу издвојимо као засебну, тј. инстанцама који припадају класи j доделимо вредност 1, а свим осталим инстанцама доделимо вредност 0. Слика 3.17 илуструје овај поступак.

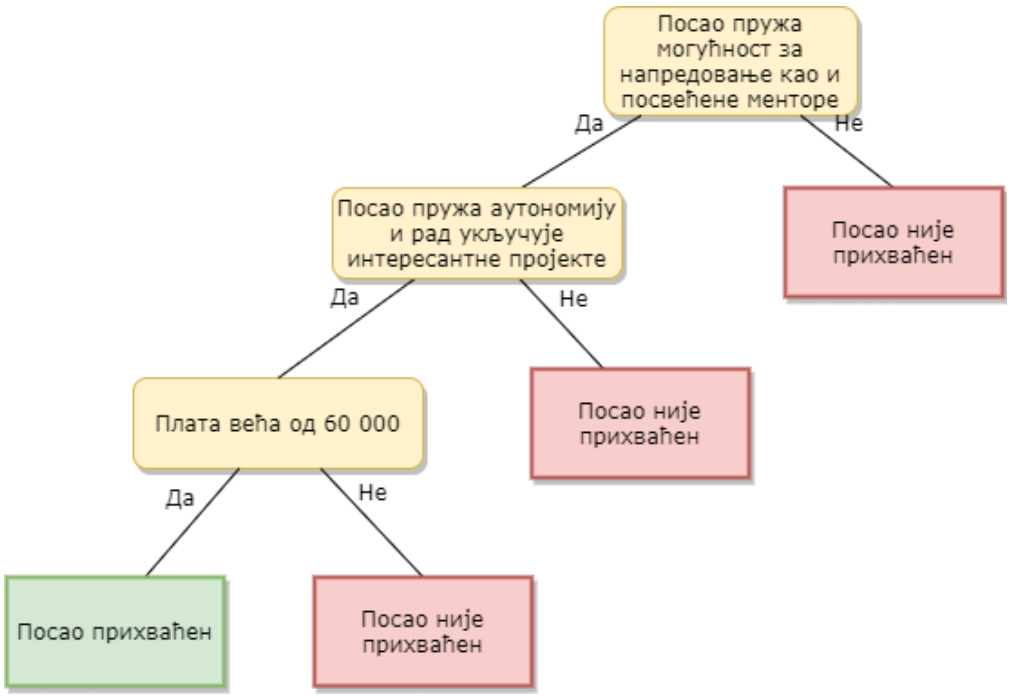


Слика 3.17 Трансформација података у поступку алгоритма један против свих

Затим тренирамо логистичку регресију да предвиди само класу *j* у односу на све остале класе. Овај поступак понављамо за свих *N* класа. Предвиђање вршимо тако што предвидимо класу помоћу *N* логистичких регресија, а затим видимо која логистичка регресија даје највећу вероватноћу припадања инстанце њеној класи, и инстанцу доделимо тој класи. Користећи пример са горње слике, уколико тренирамо три логистичке регресије да предвиде зелену, плаву или црвену и три логистичке регресије за инстанцу *i* избаце следеће резултате [0.32,0.12,0.87] инстацу *i* ћемо сврстати у класукоја представља црвене крстиће.

3.4.3 Стабла одлучивања и алгоритам насумичне шуме

Стабла одлучивања су модели који користе структуру стабла да моделује потенцијалне резултате, могућности, последиције као и у срвху класификације. Поред тога што се користе у машинском учењу, стабла одлучивања се коирсте у пословним окружењима као помоћ први одлучивању, вероватноћи, медицини и многим другим пољима. Предност овог алгоритма јесте то што је веома лак за разумевање чак и за људе који нису експерти на том пољу.



Слика 3.18 Пример стабла одлучивања

У контексту машинског учења, стабло одлучивања је нелинеарни супервизовани алгоритам. Оно је фундаментална компонента алгоритма насумичне шуме, који је међу најмоћнијим алгоритмима машинског учења [8]. Стабла одлучивања могу да се користе и за проблеме класификације и регресије [10].

Што се тиче класификације, алгоритам ради тако што покушава да подели инстанце у што хомогеније групе. Он то ради тако што гледа на основу које вредности атрибута можемо да поделимо инстанце. Стабло одлучивања креће од корена. У корену све инстанце се налазе у једној групи, а затим од корена креће дељење инстанци до одређеног критеријума заустављања.Корен се још назива коренски чвор. Његовим даљим дељењем добијамо чворове, све док не дођемо до чворова које се више не деле. Њих називамо листови.На основу листа ми датој инстанци додељујемо класу.

Разлика између стабала у машинском учењу и статистици у односу на друга стабла одлучивања јесте то што се атрибути, и њихове границе на основу којих делимо тачке, уче. Алгоритам учи тако што сваки чвор подели у два што хомогенија чвора. Он дели чворове на основу једног атрибута k и његове вредности t_k. Како алгоритам бира параметре k и t_k ? Тако тражи који пар даје најхомогеније под чворове. Хомогенсто се мери на основу *Gini* функције.

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

Једначина 3.13 *Gini* фунцкија

$p_{i,k}$ представља однос истанци класе k и осталих инстанци у чвору i.

Функција губитка коју овај алгоритам покушава да минимизује је дефинисани на следећи начин:

$$J(k, t_k) = \frac{m_d}{m} * G_d + \frac{m_l}{m} * G_l$$

Једначина 3.14 Фунцкија губита CART алгоритма

$\frac{m_d}{m}$ и G_d представљају однос инстаци односно *Gini* хомогенсто за десни добијени чвор. Други сабирак представља исте вредности само за леви добијени чвор. Овај алгоритам за креирање стабла одлучивања се зове *CART (Classification and regression tree)* алгоритам.

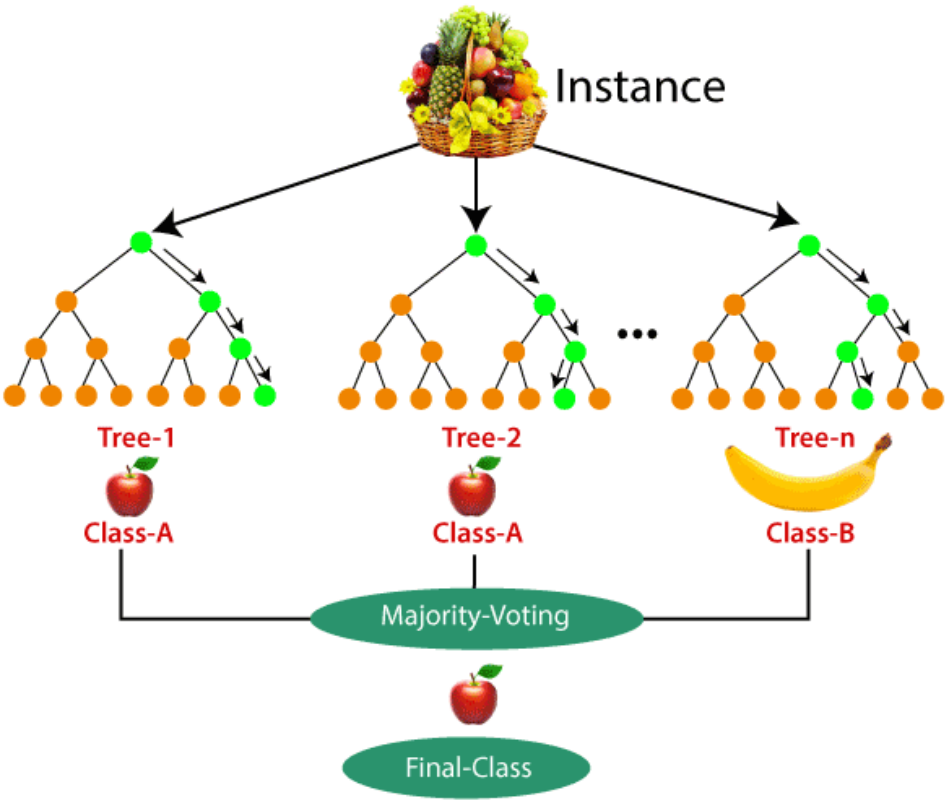
Када алгоритам успешно подели коренски чвор, он наставља рекурзивно да дели чворове све док не можемо више да поделимо податке тако да смањимо хомогеност чвора или док се не стигне до неког критеријума заустављања.

Критеријуми заустављања представљају регуларизацију за стабла одлучивања. Чести критеријуми који се користе када се тренирају стабла одлучивања јесу максимална дубина стабла одлучивања, минимални број тачака који сваки чвор мора да има пре него што је подељен, минмални број тачака за сваки лист итд.

Стабло одлучивања је непараметарски модел, што не значи да нема параметре , напротив има их много, већ да нема предефинисане параметре тренирања (као на пример линеарне регресија која има предефинисани параметар алфа). Ово резултује томе да стабла одлучивања теже да се превише прилагоде тренинг подацима и резултат је изнадподесни модел. Стога је веома битно да прикладно регулишемо модел [8].

Алгоритам насумичне шуме ради тако што тренира више насумичних стабала одлучивања, и агрегирајући њихова предвиђања он изводи своје. Обична стабла одлучивања траже најбољи атрибут и границу од свих расположивих атрибута. За разлику од њих стабла одлучивања у алгоритму насумичне шуме тражи најбољи атрибут на насумичном подсету од свих атрибута. Ово доприноси томе да стабла буду разнолика и прави компромис између грешке и варијанце што смањује изнад подесност модела.

Прво што алгоритам насумиђне ради јесте да узима узорке из података са понављањем. Када је узео узорак, он користи тај узорак да тренира стабло одлучивања користећи насумичне атрибуте. Када је истренирао више стабала одлућивања он агрегира њихова и на основу тога изводи своје предвиђања. Предности алгоритма насумичне шуме користи информацију од више стабала одлучивања и самим тим добијамо боље резултате и алгоритам је отпорнији на „оверфитовање“ [11].

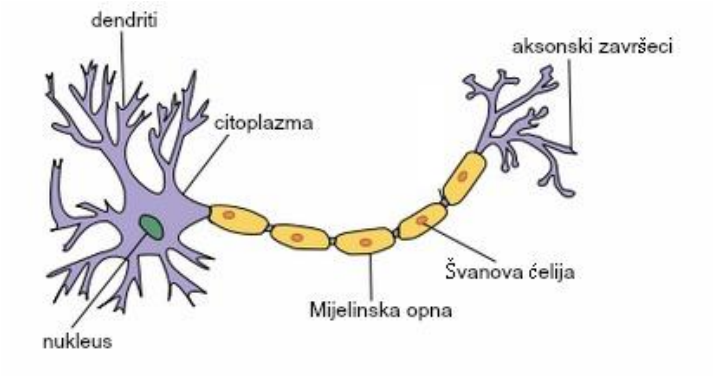


Слика 3.19 Рад алгоритма насумичне шуме

3.4.4 Вештачке неуронске мреже

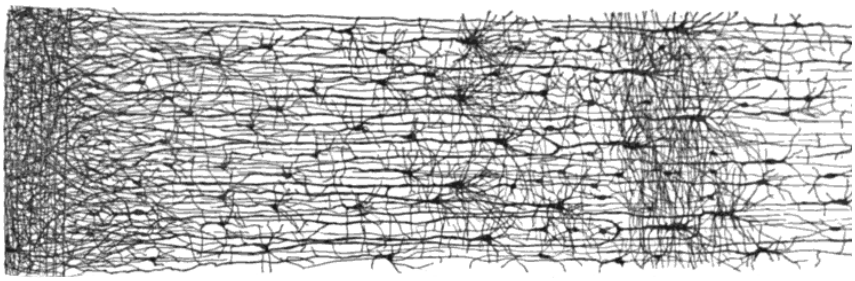
Неуронске мреже су подскуп машинског учења и представљају основу за дубоко учење. Овај алгоритми су инспирисани људским мозгом, и они копирају начин на који биолошки неурони шаљу сигнале један другом.

Пре него што објаснимо вештачке неуроне потребно је прво да објаснимо биолошки неурон. Нервна ћелија је основна јединица грађе и функције нервног система. Неуронска ћелија се састоји од цитоплазме која садржи нуклеус и још мноштво комплексних компоненти. Такође се састоји од деандрита који си кратки нервни завршеци на телу неурона, као и од аксона који је дугачак продужетак цитоплазме. Биолошки неурона прима електричне импулсе од других неурона који се зову сигнали. Сигнале прима преко синапси. Синапсе су структуре између деандрита једног неурона и аксонског завршетка другог, које омогућавају комуникацију између неурона. Када неурон прими довољан број сигнала од других неурона, он се активира и шаље свој сигнал осталим неуронима.



Слика 3.20 Неуронска ћелија и њени делови

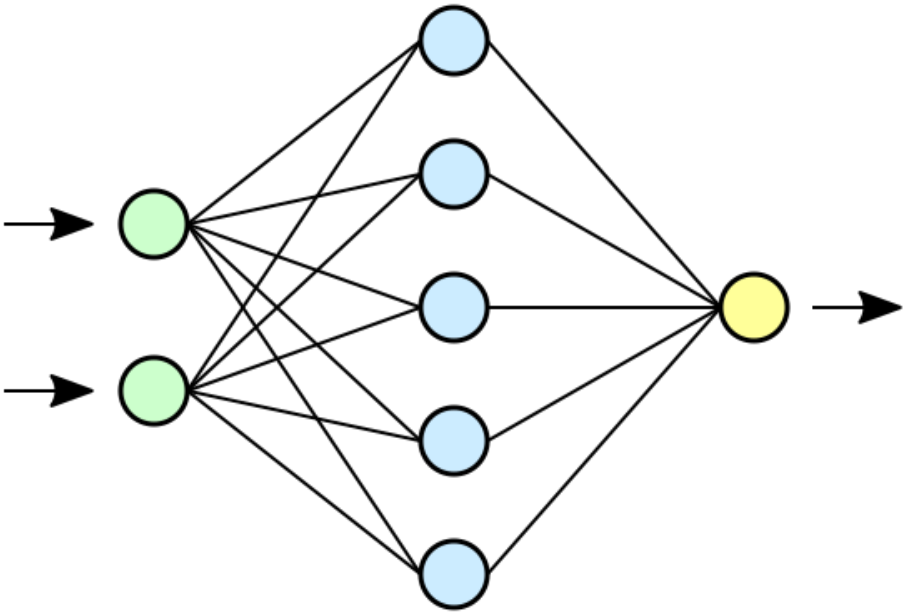
Тако да се биолошки неурони понашају у на доста прост начин, али су организовани у велике мреже од по неколико милијарди неурона, где је сваки неурон повезан са хиљадама других неурона. Архитектура биолошких неуронских мрежа још није у потпуности истражена, али неки делови који јесу истражени наговештавају да су неурони повезани у узастопне слојеве као што је приказано на слици испод. Оваква архитектура је инспирисала истраживаче да развију вештачке неуронске мреже.



Слика 3.21 Неколико слојева биолошке неуронсе мреже

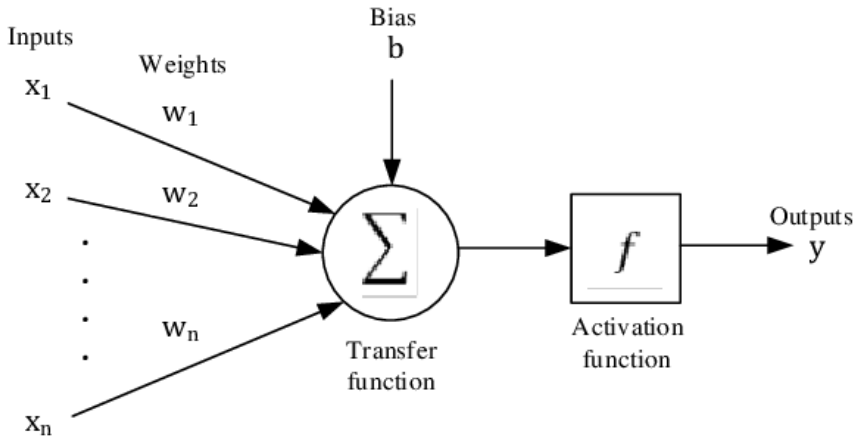
Вештачке неуронске мреже се састоје од више слојева. На улазу у сваку неуронску мрежу се налази улазни слој. Сврха улазног слоја јесте да у мрежу унесе иницијалне податке, да би се они обрадили у предстојећим слојевима. Овај слој је на почетку сваког процеса рада неуронских мрежа [12]. На излазу се налази излазни слој који представља резултат које је наша неуронска мрежа израчунала. Између улазног и излазног слоја се налазе такозвани скривени слојеви, и у овим слојевима се одвија тренирање и рачунање.

Сваки слој у неуронској мрежи се састоји од више чворова тј. неурона. Неурон у неуронској мрежи представља математичку функцију која покушава да опонаша биолошки неурон. За разлику од биолошког неурона који је или активан или није активан, активност вештачког неурона се често изражава децималним бројем између нула и један.



Слика 3.22 Приказ архитектуре неуронске мреже зеленом бојом је представљен улазни слој, плавом скривени а жутом излазни

Сваки неурон је повезан са свим неуронима из предходног слоја са такозваним тежинама. Тежине нам говоре колико активација једног неурона зависи од активације другог неурона. Технички говорећи, тежине квантификује везу између два неурона. Да би се израчунала активација сваког неурона, рачуна се матрични производ тежина са одговарајућим неуронима, додаје се грешка (*bias*), а затим се додаје још и нелинеарна функција.

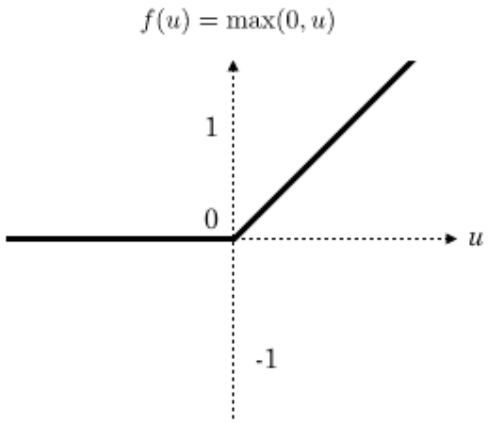


Слика 3.23 Приказ рада једног неурона у Неуронској мрежи

Нелинеарна функција се додаје да би се моделу додала додатне комплексност тј. Нелинеарност која је веома корисна за проблеме класификације и регресије, зато што решење за скоро све реалне проблеме не може да се изрази као просто сабирање и множење бројева. Такође уколико не би користили нелинеарну функцију не би били у стању да

тренирамо више од једног слоја неуронске мреже. Разлог томе зато што сукцесивне линеарне операције могу бити просто изражене као једна операција.

Нелинарна функција која се често користила у неуронским мрежама је сигмоидна функција. Међутим истраживачи су открили да *ReLU* активациона функција даје боље резултате. *ReLU (Rectified Linear Unit)* активациона функција је диференцијабилна и монотона, али за разлику од већине осталих активационих функција и њен извод је такође монотон. Функција све негативне вредности претвара у нулу, тиме говорећи да све небитне вредности су подједнако небитне. *ReLU* је тренутно најкоришћенија активациона функција, поготово у дубоким и конволуционим неуронским мрежама [45].



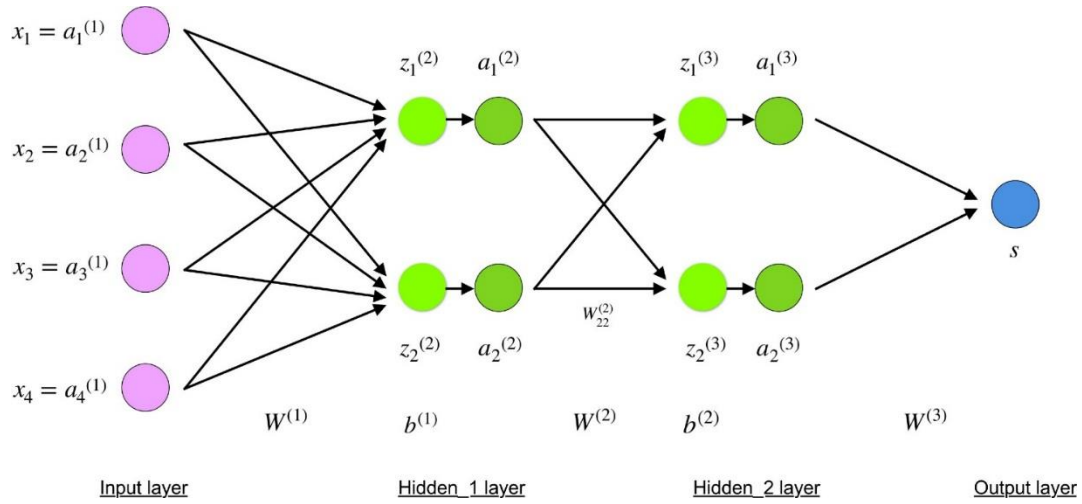
Слика 3.24 *ReLU* активациона функција

Сигмоидна функција, због својих особина да избацује вредности од 0 до 1, се користи у излазним слојевима бинарне класификације да избаци вероватноћу припадања одређеној класи. Што се тиче мултикласне класификације користи се *Softmax* функција која представља генерализацију сигмоидне функције.

$$y_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Једначина 3.15 Сигмоидна једначина

Поступак у коме неуронска мрежа на основу улазних података израчуна излаз се зове пропагација унапред. Пропагација је само низ матричних производа и додате нелинеарности. У следећим поглављима ћу објаснити математички поступак пропагација унапред.



Слика 3.25 Архитектура неуронске мреже са активацијама

Активације у улазном слоју су једнаке улазним подацима. Активације у улазном слоју ћемо означити са *X.* Активација у слоју 2 је добијена тако што смо извршили матрично множење, активације из улазног слоја, са тежинама које их спајају са слојем 2, додата је грешка а затим и нелинеарна функција.

$$Z^{(2)} = W^{(1)} * X + b^{(1)}$$
$$a^{(2)} = \sigma(Z^{(2)})$$

Једначина 3.16 Рачунање активације слоја 2 на основу улазног слоја

Овај поступак се итеративно понавља док не стигнемо до последњег слоја. Генерализација претходне формуле је

$$Z^{(i+1)} = W^{(i)} * a^{(i)} + b^{(i)}$$
$$a^{(i+1)} = \sigma(Z^{(i+1)})$$

Једначина 3.17 Рачунање активације слоја i-1 на основу слоја i

Када процес дође до задњег слоја тј. када алгоритам избаци предвиђања за дати вектор карактеристика, предвиђена вредност се пореди са тачном вредношћу и рачуна се функција губитка. На основу функције губитка и пропагације уназад, врши се само тренирање модела. Пропагација уназад итеративно мења тежине неуронске мреже, све док модел не достигне глобални минимум, или достигне неки други критеријум заустављања. Пропагација уназад рачуна колико сваки параметар утиче на грешку и на основу тога мења вредности тежина. Пропагација уназад се тако зове зато што се прво надограђују тежине у задњем скривеном слоју, па одатле иду све до првог скривеног слоја. Колико ће измена бити за дату тежину зависе од свих тежина и између ње и излаза.

Формула испод представља парцијални извод функције губитка по тежини која спаја неурон j из слоја *l-1* са неуроном *i* из слоја *l*

$$\frac{\partial \mathrm{J}}{\partial w_{ij}} = \frac{\partial \mathrm{J}}{\partial z_{ij}} * {a_k}^{l-1}$$

Једначина 3.18 Рачунање парцијалног извода функције губитка одређену тежину

Ово је само парцијални извод по једној тежини за један пример. Да би добили градијент од функције губитка неуронске мреже, неопходно је да израчунамо парцијалне изводе из свих тежина, и убацимо их у градијентни вектор, а затим узмемо средњу вредност од парцијалних извода за све тренинг инстанце.

Наравно пре него што уопште кренемо да мењамо тежине тако да модел буде што тачнији, морамо да имамо неке постојеће тежине би мењали. Процес у коме пре почетка тренирања, додељујемо тежине се зове иницијализација тежина. Иницијализација је важна зато што ако ту зезнемо наш алгоритам неће бити у стању добро учи. Ако се иницијализује све на нулу, ако је иницијализација симетрична или ако је различита варијанса између слојева, тренирање неће бити оптимално.

Тренутно најбољи начин да се изведе иницијализација је Гзавијеровом методом иницијалицазије (Xavier Initialization) . Вредности за сваку тежине се узимају из униформне дистрибуције која је ограничена са следећим вредностима

$$\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}$$

Једначина 3.19 Границе за униформну дистрибуцију из које се узимају вредности за улазне параметре

Препоруке за селекцију броја скривених чворова су следећи. Lachtermacher и Fuller [18] даје хеуристично ограничење броја скривених чворова. За случај најчешћих неуронских мрежа са једним скривеним слојем, постоји неколико практичних смерница. Оне укључују „2n+1“ ([19]; [20]), „2n“ ([21]), —„n“ [22], „n/2“ ([22]), где је n број улазних неурона односно чворова. Међутим, ниједан од ових хеуристичних избора не ради добро за све проблеме.

Tang и Fishwick [22] су истраживали утицај скривених чворова и пронашли да број скривених чворова утиче на перформансе класификације али да тај утицај није толико значајан [17]. За тренирање модела у овој студији случаја, грубо смо се држали ових правила.

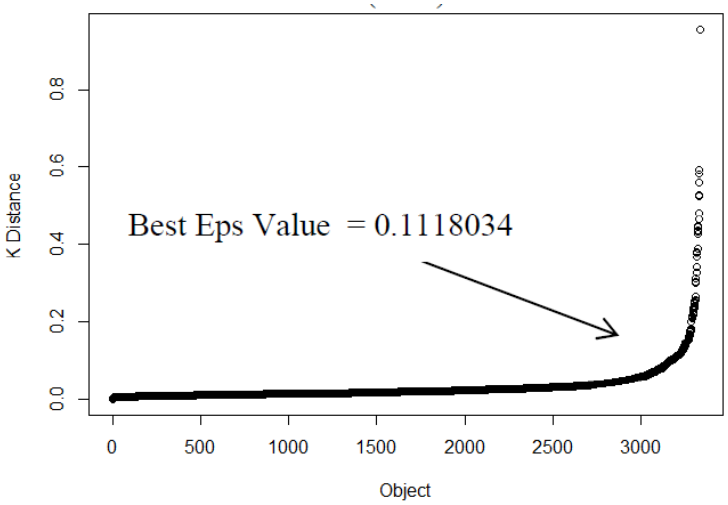
3.5 Модели ненадгледаног учења

3.5.1 DBSCAN

DBSCAN (*density-based spatial clustering of applications with noise*) је алгоритам ненадгледаног учења који се користи за сегментацију података. Кластери у овом алгоритму су дефинисани као региони са великом густином инстанци [8].

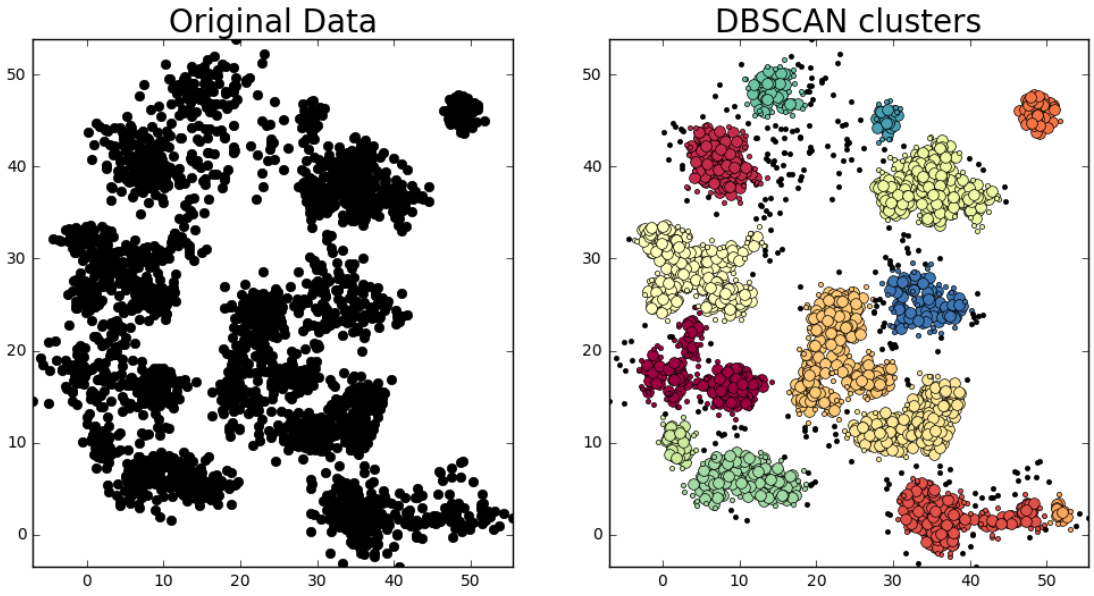
Алгоритам ради на следећи начин: За сваку инстанцу алгоритам рачуна колико других инстанци се налазе у његовом окружењу, које се овде зове епсилон окружење. Затим свака инстанца која у свом епсилон окружењу има више инстанци проглашава се за кључну. Дакле, кључне инстанце су оне које се налазе у густим регијама. Затим све инстанце које се налазе у окружењу кључне, се сврставају у један кластер. У окружење које се сврстава у један кластер могу да уђу више кључних инстанци, тако да је један кластер заправо само дуги низ повезаних инстанци. Хиперпараметри које треба да дефинишемо за овај алогритам су колико је то епсилон окружење, тј. која ја максимална удаљеност између тачки које сматрамо суседним, као и колико тачака мора бити у суседству инстанце да би се она прогласила за кључну инстанцу.

Хиперпараметре које треба да користимо се разлику од скупа податак до скупа података, те је веома важно да се параметри лако и брзо могу одредити за различите скупове. Један од начина да дођемо до оптималних хиперпараметара за *DBSCAN* алгоритам је следећи поступак. Минимални број тачака у комшилуку који су потребни да би се тачка прогласила за кључну, треба одредити на основу броја атрибута наших података. Препорука је да се за минимални број тачака изабере број између n и $2n$, где n представља број атрибута [12]. Када је у питању одабир епсилон параметра, прво треба одредити за сваку тачку растојање до њених трију најближих комшија, а потом та растојања до трећег најближег суседа сортирани у растући низ. Затим је треба да се плотује овај низ, и у подручју где се функција највише прелама, изабере вредност епсилон параметра.



Слика 3.26 Неопадајућа функција која представља раст удаљености првог комшије од тачака

DBSCAN ради добро уколико су кластери довољно густи и раздвојени су регијама мање густине. Алгоритам је једноставан, а ипак доста моћтан. Омогућава да индентификујемо различите бројеве кластере, разних величина и плус је отпоран на изузетке. Рачунарска комплексност му је скоро линеарна, што значи да може брзо да кластерује огромне количине података. Мана овог алгоритма јесте ако имамо податке чија густина веома варира,и у том случају он није у стању да их исправно кластерује.



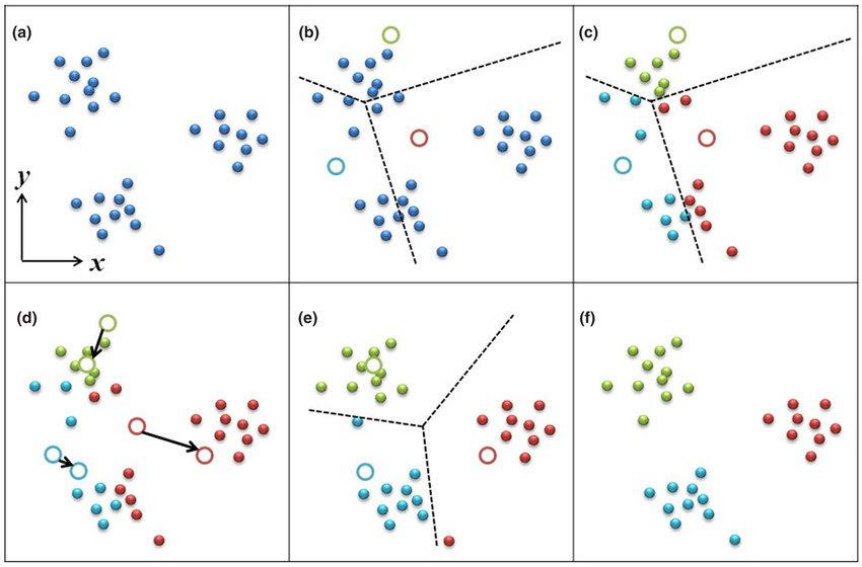
Слика 3.27 Кластеризација тачака помоћу DBSCAN-а

3.5.2 KMeans алгоритам

KMeans алгоритам је ненадгледаног учења који користи за кластеризацију података. Он је веома једноставан алгоритам који је у стању да класификује разне сетове података веома брзо и ефикасно, често у само неколико итерација. Њега је предложио Стјуарт Лојд у Бел Лабораторији 1957. Године али је ова метода објављена ван лабораторије тек 1982.

Године. До тада је Едвард Форги објавио готово исти алгоритам и овај алгоритам се некад назива и Лојд-Форгијев алгоритам [8].

Алгоритам ради на следећи начин: Прво изаберемо у колико кластера желимо да сврстамо наше податке (параметар К који стоји у имену). Када смо изабрали колико кластера желимо, алгоритам насумично изабере толико инстанци и они се прогласе за центроиде. Све инстанце које су најближе Н-том центроиду се сврставају у Н-ти кластер. Када су све инстанце сврстане у свој кластер, за сваки кластер се израчунава средња вредност и на месту те средње вредности се померају центроиди. Сада када су центроиди померени, поновно се рачуна које су инстанце најближе ком центроиду и поновно се понавља поступак доделе и тражења средине све док алгоритам не конвергира.



Слика 3.28 Приказ рада KMeans алгоритма. Од а па до ф примера су приказани кораци алгоритма

Иако алгоритам засигурно конвергира, то не значи да је кластеризација оптимална. Резултат кластеризације у многоме зависи од саме иницијализације тачака. Оно што већина имплементација овог алгоритма раде, јесте да алгоритам покрећу више пута са различитим иницијализацијама, и гледају на основу инерције кластера која се кластерзација најбоља и њу узимају као коначну. Инерција представља просечно квадратно растојање између сваке инстанце унутар кластера и њеног центроида.

Постоје такође и верзије алгоритма у коме сама иницијализација и није потпуно насумична. Дејвид Артур и Сергеј Василвиски су предложили метод иницијализације, у коме је велика вероватноћа да почетни центроиди буду удаљени једни од другог [16].**Error! Reference source not found.**

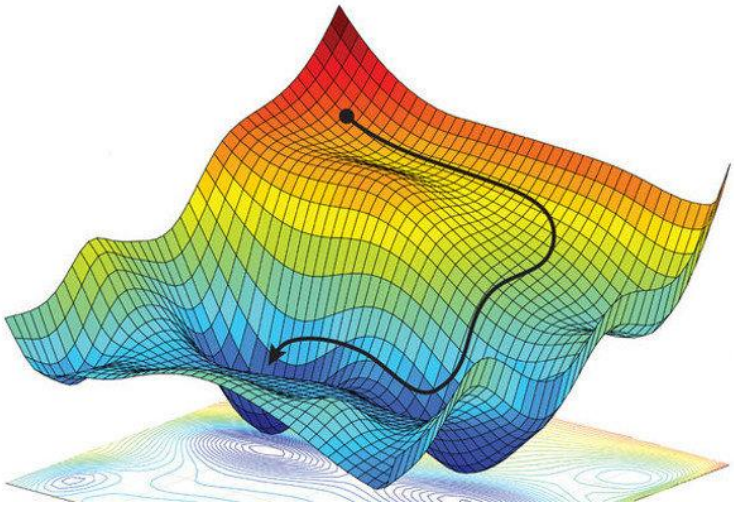
Такође, *Kmeans* се често користи за квантизацију боја Квантизација боја представља поступак у коме се боје из оригиналног пребацујемо у приказ са пажљиво селектованим

мањим бројем боја. Коришћење редукције боја нам омогућава да смањимо меморијске захтеве, као и да убрзамо сам тренинг модела [13].

3.6 Оптимизациони алгоритми

Оптимизациони алгоритам је поступак које се извршава итеративно, тако што пореди разна решења док оптимиум или задовољавајуће решење нису пронађени. Када смо дефнисали наш модел, и када смо изабрали функцију губитка, сам начин како алгоритам проналази идеалне параметре тј. учи, је уз помоћ оптимизационог алгоритма. Ефикасност оптимизационог алгоритма директно утиче на преформансе нашег модела, као и на само трајање тренирања, стога је битно да изабермо оптимизациони алгоритам који највише одговара нашем задатку.

Технички речено оптимизација се односи на минимизацији/максимизацију објективне функције *f(x)*, која као параметар прима варијаблу x. У машинском/дубинском учењу, задатак је минимизовати функцију губитка *J(w)* чији је параметар $w \in R^d$.



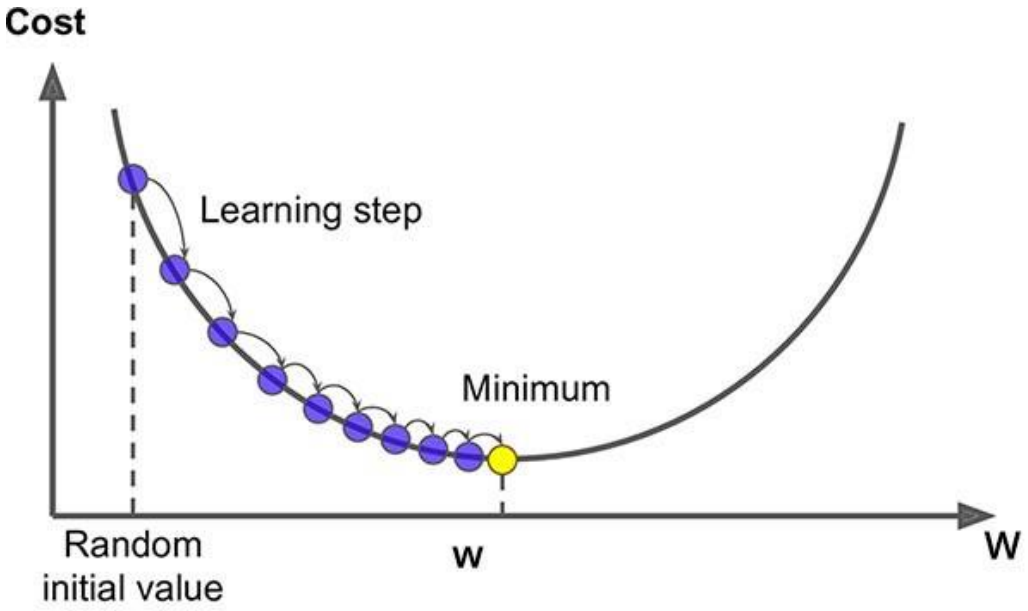
Слика 3.29 Функција губитка. Линија представља пут којим је алгоритам прошао да дође до минимума

3.6.1 Градијентни пад

Највише оптимизационих алгоритама се базирају на алгоритму градијентног пада. Алгоритам градијентног пада је алгоритам првог реда. То значи да алгоритам рачуна само изводе првог реда како би дошао до идеалних параметара. У свакој итерацији мењамо параметре модела у правцу супротном од градијента, који даје правац највећег раста функције.

Идеја алгоритма градијентног пада јесте да итеративно поправљамо параметре модела све док модел не конвергира ка минимуму. Параметре поправљамо у смеру највећег пада функције губитка, тј. у смеру супротном од градијента функције. Када је градијент једнак нули стигли смо до локалног минимума.

Један од важних параметара за алгоритам градијентног пада јесте стопа учења тј. дужина корака којим ће се алгоритам кретати у смеру највећег пада. Овај параметар је важно добро да потрефимо зато што ако му доделимо вредност која је премала, алгоритму ће требати пуно итерација да стигне до минимума а уколико поделимо вредност која је превелика, алгоритам може да дивергира. Уколико и ставимо мало већу стопу учења која неће дивергирати, са њом ће алгоритам у почетку направити значајан напредак али неће стићи скроз до оптималног решења већ ће „плесати“ око њега. Један од начина да решимо овај проблем јесте да уведемо распоред мењања стопе учења. Да у почетку ставимо велику стопу учења, да би алгоритам у почетку доста напредовао, а да онда када се приближимо оптималном решењу да смањимо стопу учења како би добили што је боље решење.



Слика 3.30 Оптимизациони алгоритам градијентног пада

Сам начин како овај алгоритам ради јесте тако што нађе парцијалне изводе у односу на сваки параметар модела засебно. Парцијални изводи нам кажу колико ће се функција губитка променити уколико ми променимо вредност датог параметра за малу вредност.

Градијенти вектор садржи парцијалне изводе за све параметре модела и он је усмерен у смеру највећег раста функције губитка. Ако узмемо инверз од градијента тј. одузмемо вредност градијента наша функција ће се кретати у смеру највећег пада.

Сада можемо да изразимо једначину за градијентни пад

$$w = w - \eta * \nabla J(w)$$

Једначина 3.20 Корак градијентног пада

Сада се поставља питање колико итерација је потребно да одрадимо да би добили идеалне параметре. Ако буде премало итерација наше решење неће конвергирати до оптималног решења, а ако будемо имали превише итерација, без потребно ћемо користи рачунарске ресурсе. Решење је да изаберемо епсилон толеранцију, и када разлика између два узастопна градијента буде мања од епсилон толеранције, ми престанемо са тренирањем.

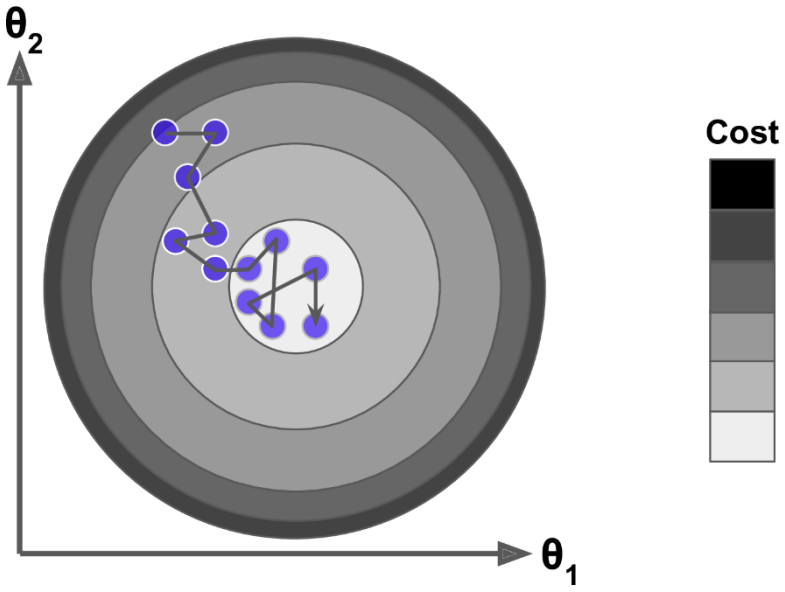
3.6.2 Стохастички и мини серијски градијентни пад

Горе споменути алгоритам представља серијско градијентно учење. Њега карактеристише то што за сваки корак градијентног пада, неопходно је израчунати функцију губитка користећи све тренинг инстанце.

Овај начин тренирања изискује доста времена за рачунање сваког корака, али је загарантовано да ће сваки корак бити у смеру највећег пада као и то да ће алгоритам конвергирати ка глобалном минимуму (уколико је функција губитка конвексна).

Алгоритам који је развијен да би надоместио недостатке серијског градијентног пада, пре свега време које је потребно за сваки корак јесте, стохастички градијетни пад. Идеја овог алгоритма јесте да функцију губитка израчунамо само на основу једне инстанце, и да корак градијентог пада направимо само на основу градијента овакве функције.

Стохастички градијенти пад нам омогућава да уштедимо много времена за сваки градијенти корак, али нам не гарантује да ће сваки корак бити у правом смеру, међутим гарантује нам да ћемо се у просеку спустити. Алгоритам такође не гарантује да ћемо наћи глобални минимум зато што се функција понаша нерегуларно, и стога је битније за овај алгоритам да правилно подесимо стопу учења. Још једна предност овог алгоритма у односу на градијентни пад јесте то што лакше избегне локали минимум, уколико функција није конвексна.



Слика 3.31 Стохастични градијентни пад

Дакле, имамо серијски градијентни пад, који за сваки корак рачуна функцију губитка на основу свих тренинг инстанци, и прави споре али проверене кораке, и стохастички градијентни пад који рачуна функцију губитка на основу само једне инстанце, и прави брзе али не толико тачне кораке. Алгоритам који прави компромис између ова два алгоритма зове се мини-серијски градијентни пад. (*Mini-Batch gradient descent*). Овај алгоритам рачуна функцију губитка на основу одређеног броја инстанци. Број инстанци који користимо се зове величина серије (*batch size*), и врло је важан параметар. Типична вредност овог параметра се креће од 50 до 256 [23].

На овај начин смањујемо диверзитет сваког корака, што самим тим утиче на стабилност конвергенције. Такође савремене библиотеке су имплементиране тако да максимално оптимизују сам процес рачунања градијента. Због ових оваквих карактеристика, мини-серијски градијенти пад је најкоришћенија варијанта градијентног пада [23].

3.6.3 Оптимизациони алгоритми који користе моментум

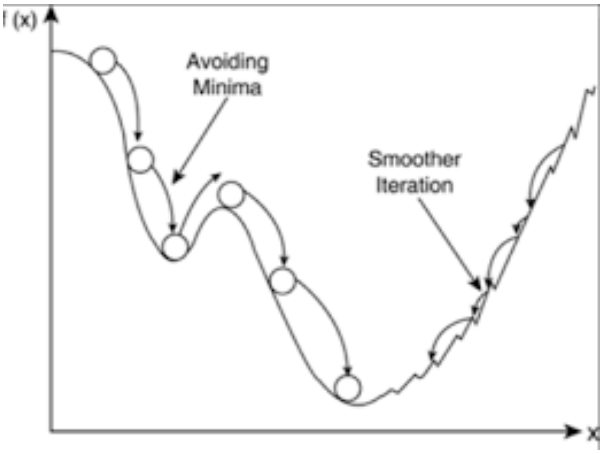
Обични алгоритми градијентног пада имају проблема да нађу глобали минимум уколико функција губитка није конвексна и ако има „јаруга“ тј. места на којима функција опада много брже у правцу једном правцу, што је типично за локални минимум. У оваквим ситуацијама ако користимо обичан градијенти пад, врло је вероватно да ће се функција заглавити у локалном минимуму и самим тим ћемо добити не оптимални модел.

Моментум је метода која нам омогућава да убрзамо градијенти пад у коректном смеру, и самим тим избегнемо локални минимум и осцилације које алгоритам прави. Ова ради тако што дода део од вектора промене од прошлог корака тренутном кораку:

$$v_t = \beta * v_{t-1} + \eta * \nabla J(w)$$
$$w = w - v_t$$

Једначина 3.21 Корак градијентног пада са моментумом

Уобичајена вредност за бета параметар је око 0.9. Члан који се односи на моменат се повећава за димензији чији градијент је усмерен у истом смеру, а смањује се за димензије чији градијент упире у супротном смеру. Као резултат тога добијамо бржу конвергенцију и и мање осцилација.



Слика 3.32 Градијентни пад са моментумом

Један од метода који користе моментум је *RMSprop*. *RMSprop* је необављени оптимизациони алгоритам, ког је први предложио Џеф Хинтно у лекцији 6 у онлајн курсу [34]. Алгоритам се базира на адаптивном учењу степена учења. Алгоритам ради тако што користи покретни просек квадратног корена од градијента да би нормализовао градијенте. Нормализација балансира моментум, смањивајући градијенти корак за огромне градијенте, а повећавајући корак за мање градијенте.

$$v_t = \beta * v_{t-1} + (1-\beta) * \nabla J(w)^2$$
$$w = w - \frac{\eta}{\sqrt{v_t}} \nabla J(w)$$

Једначина 3.22 Корак градијентног пада *RMSprop*

Просто речено *RMSprop* користи адаптивну стопу учења уместо да третира степен учења као хиперпараметар. Ово значи да се стопа учења мења са временом [35].

Адам је још један метод који користи адаптивну стопу учења за сваки параметар. Осим што чува експоненцијално опадајући просек квадратног корена као и *RMSprop* Адам

такође чува експоненцијално опадајући просек претходних градијената, што је јако слично самом моментуму. Просто речено Адам је комбинација *RMSprop* и моментума.

Прво што треба да урадимо јесте да израчунамо експоненцијално опадајући просек за градијенте и за квадриране градијенате.

$$v_t = \beta * v_{t-1} + (1-\beta) * \nabla J(w)^2$$
$$m_t = \beta * m_{t-1} + (1-\beta) * \nabla J(w)$$

Једначина 3.23 Експоненцијални опадајући просек квадрата претходних градијената и моментум

m_t и v_t су иницијализовани као нула вектори, и самим тим се крећу близу нуле, поготово у првих пар итерација. Ово доводи до тога да првих пар итерација алгоритма су практично безначајни, и морали би да чекамо пар итерацији док алгоритам не почне да ради. Да би поправили ово, користимо следеће формуле

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

Једначина 3.24 Поправак параметара из претходне једначине (Једначина 3.27)

Сада када смо поправили параметре убацујемо их у следећу једначину.

$$w = w - \frac{\eta}{\sqrt{\hat{v}_t}} \hat{m}_t$$

Једначина 3.25 Корак градијентног пада. *Адам*

Аутори су предложили да за параметре користимо 0.9 за β1 0.999 за β2. У пракси аутори су показали да алгоритам боље ради у односу на остале адаптивно базиране алгоритме [23].

3.7 Мере перформанса за класификацију

Мере перформанса је есенцијалан део сваког пројекта машинског учења. Евалуација модела нам помаже да одредимо који модел је најбољи за наш задатак, као и колики квалитет предвиђања можемо да очекујемо у будућности.

Стандардна мера перформанса јесте тачност, тј. однос тачно класификованих података и укупног броја података. Тачност даје грубу евалуацију модела, и честа је метрика коју истраживачи објављују у својим радовима. Међутим тачност нам не даје пуно слику наше класификације. На пример, уколико би хтели да класификујемо бенигне и малигне туморе, а имамо податак да је 95% тумора бенигно, и ако би имали модел који сваку инстанцу класификује као бенигну, наш модел би имао 95% тачности. Иако је наш модел очигледно погрешан, тачност му је веома добра, стога требају нам друге мере перформанса како би добили комплетну и непристрасну евалуацију нашег модела.

3.7.1 Матрица конфузије

Матрица конфузије је табела која се користи да прикажемо резултате класификације. Она омогућава интуитивну визуализацију из које лако можемо да израчунамо остале мере перформанса.

Редови у матрици конфузије представљају предвиђене класе података, а колоне представљају стварне класе (Начин репрезентације може да варира . Ако означимо матрицу конфузије C, онда је број обзервација у $C_{i,j}$ број тачака чија је стварна класа I, а класификоване су у класу j.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Слика 3.33 Матрица конфузије

Све једно је за коју класу одредимо да нам буде позитивна а која негативна, мада пракса је да класе које желимо да предвидимо буду позитивне. Вредности у матрици имају следеће значење.

$C_{1,1}$ - Стварно позитивни (*True positive - TP*) – представљају инстанце које припадају позитивној класи, а исправно им је додељена позитивна класа

$C_{1,2}$ - Лажно позитивне (*False positive - FP*) – представљају инстанце које припадају негативној класи, а погрешно су класификоване као позитивне.

$C_{2,1}$ – Лажно негативне (*False negative - FN*) – представљају инстанце које припадају позитивној класи, а лажно су класификоване као негативне

$C_{2,2}$ – Стварно негативе (*True negative - TN*) - - представљају инстанце које припадају негативној класи, а коректно су класификоване као негативне

Идеална ситуација би била да су све вредности ван дијагонале једнаку нули. У следећим пасусима ће бити наведене мере перформансе које се могу израчунати директно из матрице конфузије.

Прецизност представља однос исправно класификованих позитивних инстанци и свих класификованих позитивних инстанци

$$\text{Прецизност} = \frac{TP}{TP + FP}$$

Једначина 3.26 Прецизност

Одзив представља однос исправно класификованих позитивних инстанци и свих позитивних инстанци

$$\text{Одзив} = \frac{TP}{TP + FN}$$

Једначина 3.27 Одзив

Прецизност и одзив се често представљају преко једне вредности која се зове Ф1 резултат. Ф1 скор представља хармоничну средња вредност између прецизности и одзива.

$$\text{Ф1 резултат} = \frac{2 * \text{Прецизност} * \text{Одзив}}{(\text{Прецизност} + \text{Одзив})}$$

Једначина 3.28 Ф1 резултат

Разлог зашто не користимо обичну средњу вредност, јесте то што хармонична средња вредност додатно пенализује случаје када је једна вредност знатно мања од друге (имамо велику прецизност а јако мали одзив, или обрнуто).

Да ли нам је прецизност или одзив битнији зависи доста од самог класификационог случаја који радимо. Ако нам је битно да су све инстанце које су класификоване као позитивне заиста буду позитивне онда ћемо приоритизирати прецизност. Са друге стране, уколико нам је битно да нам не промакне ни једна позитивна класа тј. да је класификујемо као негативну, ставићемо приоритет на одзив. На пример, уколико радимо класификацију

спам имејлова, и сваки имејл који се класификује као спам се брише, битно нам је да сваки мејл који је класификован као спам заиста буде спам, да којим случајем не би избрисали неки имејл који нам је важан. У овом случају би нам прецизност била много битнија. Супротан случај би био уколико би требали да класификујемо малигност тумора код пацијената. Ту би нам било битније да сваки рак који је потенцијално малигни, идентификујемо и пошаљемо пацијента на додатне контроле. Ове је случај у којем би нам био битнији одзив.

Однос између прецизности и одзива можемо да контролишемо тако што ћемо контролисати праг класификације. Најчешћи праг класификације (праг дискриминације) је 0.5, што значи да уколико инстанца припада некој класи са вероватноћом већом од 0.5 биће класификована као позитивна, а у супротном случају као негативна. Генерално говорећи, уколико смањимо праг класификације, имаћемо више позитивних предвиђања и самим тим већи одзив а мању прецизност, а уколико ставимо већи праг, имаћемо већу прецизност а мањи одзив. Кажем генерално зато што не мора увек да значи да постоји компромис између прецизности и одзива у односу на праг, да не мора увек да се деси да ако смањимо праг да ћемо смањити и прецизност. Постоје неки прагове који су свеукупно бољи од других прагова. Начин да откријемо колики нам праг треба јесте преко криве радних карактеристика.

3.7.2 Крива радних карактеристика

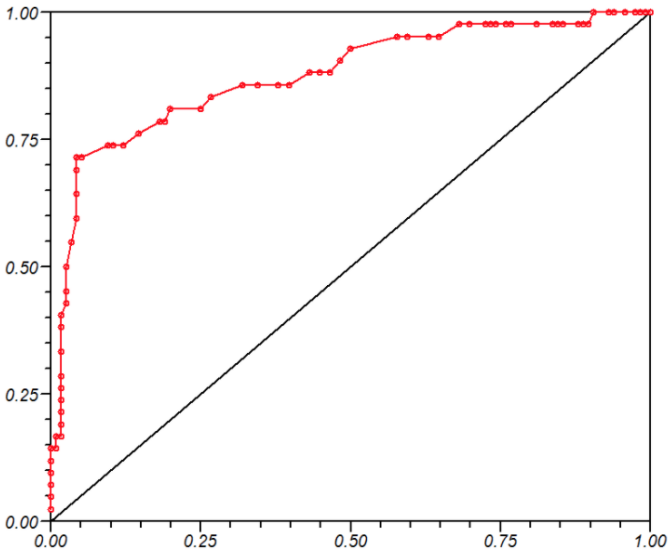
Крива радне карактеристике пријемника (РОЦ крива) је графички приказ који илуструје дијагностичку способност бинарног класификатора како варира његов праг дискриминације[39]. РОЦ крива плотује сензибилитет модела на *y* оси а специфицитет кад *x* оси.

Сензитивитет је синоним за одзив модела (Једначина 3.24) док је специфицитет проценат коректно предвиђених негативних класа у односу на све негативне класе.

$$\text{Специфицитет} = \frac{TN}{TN + FP}$$

Једначина 3.29 Специфицитет

Кирва радних карактеристика плотује сензитивитет и специфицитет над целим интервалом прага дискриминације.



Слика 3.34 Крива радних карактеристика

Област испод криве (Слика 3.33) је интеграл криве радних карактеристика. Уколико област једнак 0.5 алгоритам не први разлику између класе и није ништа бољи од модела који би насумично класификовао инстанце. Изванредни модели имају вредност области испод криве близу јединице.

3.8 Машинско учење у Пајтон програмском језику

Пајтон је интерпретирани, објектно-оријентисани виши програмски језик са динамичком семантиком. Пајтон-ова једноставна синтакса се лако учи и наглашава читљивост кода која за последицу има ниску цену одржавања [41]. Пајтон је језик отвореног кода и тренутно је најпопуларнији програмски језик на свету [42]. Његова популарност директно утиче на величину заједнице која ради на развијању библиотека отвореног кода за Пајтон програмски језик.

Програмски језик је развио Гвидо ван Росум (*Guido van Rossum*). Са радом је почео осамдесетих година прошлог века, а инспирацију за овај пројекат добио је од ABC програмског језика. Прва верзија Пајтона је изашла 1991. године. Касније верзије Пајтон 2 и Пајтон 3 су изашле 2000. и 2008. године.

3.8.1 Пајтон библиотеке за машинско учење

Као што смо већ напоменули Пајтон пружа својим корисницима огромну количину библиотека отвореног кода. Један од најпознатијих библиотека јесте.

SciPy је екосистем слободног софтвера за математику, науку и инжењерство.Његову основу чини програмски језик Пајтон и група пакета од којих су за машинско учење најзначајнији [43]

1. NumPy је написан у *C* програмском језику, и то је екстензија Пајтон модула. Дефинисан је као Пајтон пакет који се користи за разне нумеричке комутације и процесирање мулитдимензионалних и једнодимензионалних низова. Калкулацију у Пајтону су оптимизоване и много су брже него калкулације у самом пајтону. Због високо оптимизованог начина израчунавања матричног множења и начина складиштења огромних количина података, NumPy је користи као темељ за доста других библиотека (Пандас, тензорфлов...)
2. Пандас је библиотека отвореног кода која пружа перформансе високог нивоа за манипулацију података у пајтон програмском језику. Написана је на темељу NumPy програмског језика. Највише се користи за манипулацију табеларних података. Пандас употпуњује Пајтон са пет кључних опција за аналитику података, а то су учитавање, манипулација, припрема, моделовање и анализа [44].

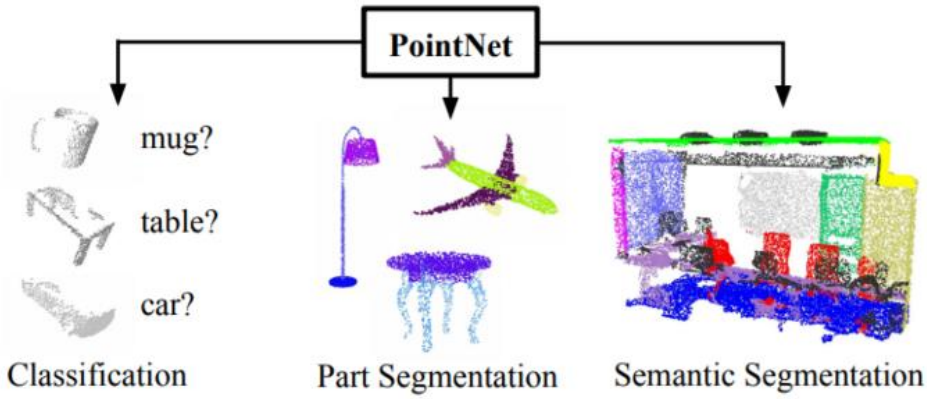
Библиотека која је такође једна од најпопуларнијих за машинско учење јесте тензорфлов (*TensorFlow*). Тезорфлов је платформа отвореног кода за машинско учења, која обухвата цео радни процес машинског учења. Онда има обиман, флексибилан екосистем алата, библиотеке и ресурса који омогућавају истраживачима да и компанијама да лако израда моделе машинског учења. Тензорфлоф су развили инжењери Гугл-а за њихову употребу унутар компаније, а објавили су га јавно 2015. године.

Библиотеку коју смо користили за визуализацију облака тачака јесте *open3d.*Open3d је библиотека отвореног кода која подржава брзо развијање софтвера која се бави са 3D подацима, између осталог бави се визуализацијом и процесирањем облака тачака.

4. Примена машинског учења за класификацију облака тачака

Машинско учење је примило доста пажње у последње време због своје применљивости у разним доменима као што су компијутерска визија, аутономна вожња и роботика. Што се тиче класификације облка тачака, машинско учење се највише примењује у класификацији облика, 3D препознавању и праћењу објеката као и сегментација објеката. Један од највећих изазова у овим задацима представља висока димензионалност, обимност и нерегуларност облака тачака [48].

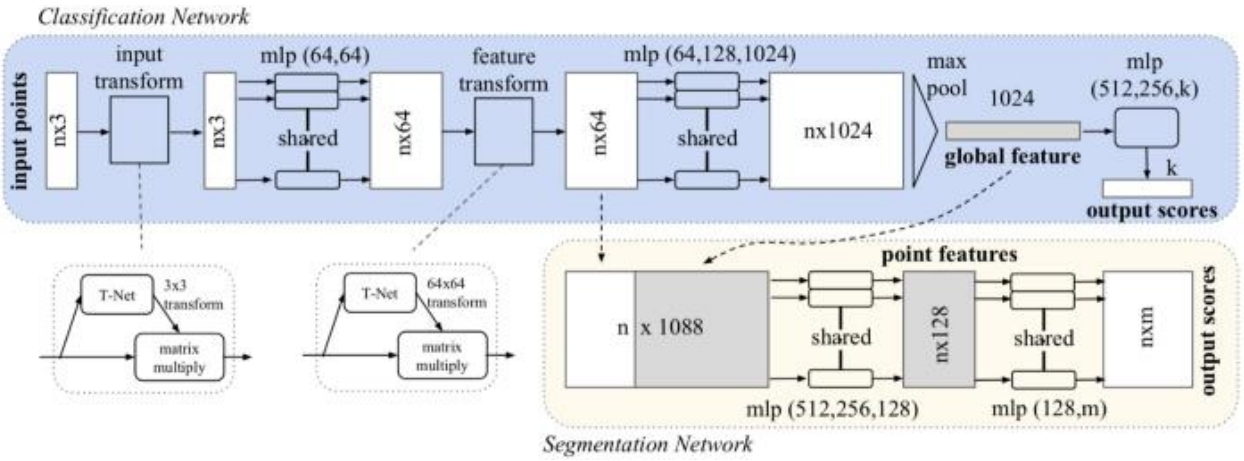
Уградња LiDAR уређаја унутар новијих телефона и аутомобила додатно је дала на значају примени машинског учења за облаке тачака. Истаживачи на MIT-у (Massachusetts institute of technology) користе у целости алгоритме машинског учења за пројекте аутоматске вожње. Да би успели да поднесе високе рачунарске захтеве које мапинско учење задаје, и толико убрзали процес да омогуће вожњу у реалном времену, дизајнирани су модели који користе облак тачака и ГПС мапе на више графичких процесора истовремено.



Слика 4.1 Примене мапинског учења на облаку тачака

Проблему употребе машинског учења на облаком тачака, истраживачи су пришли на различите начине. Решења која су се користила за овај проблем убрајају коришћење вештачких неуронских мрежа са претходном селекцином атрибута, коришћење конволуционих неуронских мрежа на креираном вокселу (3D растеру), коришћење ауто-енкодера итд.

Тренутно један од најсавременијих модела је PointNet мрежа [49][50]. Основна идеја јесте да се тачке прво мапирају у вишедимензионални простор користећи заједничке вештачке неуронске мреже за све инстанце. Свака инстанце се идентично и независно мапира у вишедимензинални простор. Затим када су тачке трансформисане, групишу се више тачака у један мали кластер, који је онда компресован у једну тачку. У зависности да ли се врши класификација целокупног облака тачака (да ли облак тачака представља зграду или не), или се врши класификације сваке појединачне тачке облака, користе се различите архитектуре *PointNet* мреже.



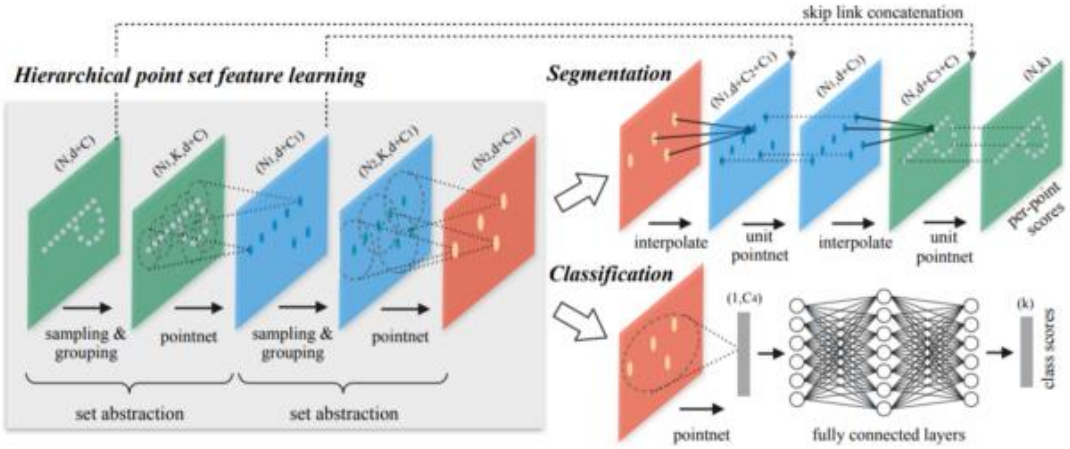
Слика 4.2 Архитектура PointNet -а

Проблем неструктурисаности података и њихове осетљивости на пермутације, *PointNet* користи симетричне функције. Симетричне функције су оне које дају исто решење за параметре , без обзира на њихов редослед. Функција која је показала најбоље резултате јесте max pooling.

PointNet такође импелемтира T мрежу. Она се користи да своју неуронску мрежу да научи које параметре матрице треба да користи за афину трансформацију. Због увођења додатних параметара повећавамо шансу да оверфитујемо наш модел, стога морамо да извршимо и додатне регуларизације када. Идеја T мреже јесте да наш модел не буде осетљив на ротације облака тачака.

Међутим, *PointNet* не успева да препозна локалне структуре унутар метричког простора у коме се налазе тачке, и самим тим његова могућност да препозна ситне патерне и да се генерализује на комплексне сцене је лимитирана. Да би решили овај проблем, Charles

R. Qi и његове колеге су увели хијерархиску неуронску мрежу која користи PointNet рекурзивно на угњежденим партицијама улазног облака тачака. Такође, имплементирани су додатни слојеви мреже који адаптивно комбинију податке који се налазе у различитим пољима густине. Овај унапређени модел је назван *PointNet++*.



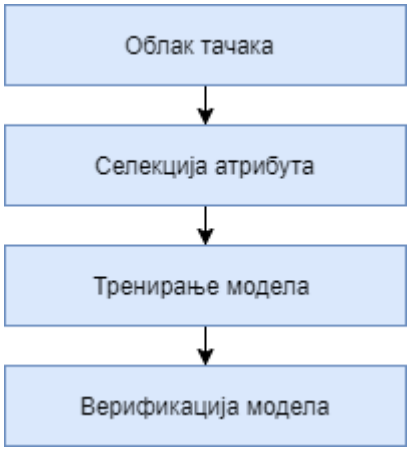
Слика 4.3 Архитектура PointNet++.

Архитектура *PointNet++* се састоји од неколико компоненти који сабирају локалне информације и прослеђују их следећем слоју. Она има неколико фаза али свака фаза је прецизно одређена. Почињући од целог облака тачака, модел кластерује групу тачака на основу њихове сличности у метричком простору. Затим, када су тачке груписане користимо *PointNet* мрежу да научи параметре за апстрактно представљање сваке групе тачака. На овај начин добили смо тачке које поред просторних информација носе и информације које представљају групу у којој се тачка налази. Опет у зависности од тога да ли радимо класификацију сваке тачке или целог облака тачака користимо другачији приступ. Уколико радимо класификацију сваке тачке, неопходно је да обрнемо процес груписања тачака, и одрадимо њихову интерполацију како би добили оригиниални приказ облака тачака.

Конволуционе неуронске мреже су такође доста успешне у раду са облаком тачака. Успешност се огледа у томе што могу да препознају просторно-локале корелације на сликама које су рендероване из пресека облака тачака. Такође, урађени су многи радови користећи 3D вокселе. Међутим, и улазни воксел и конволуциони кернели се налазе у високодимензионом простору, и количина меморије и рачунарске моћи која је неопходна за рад са њима брзо постаје превелика [51].

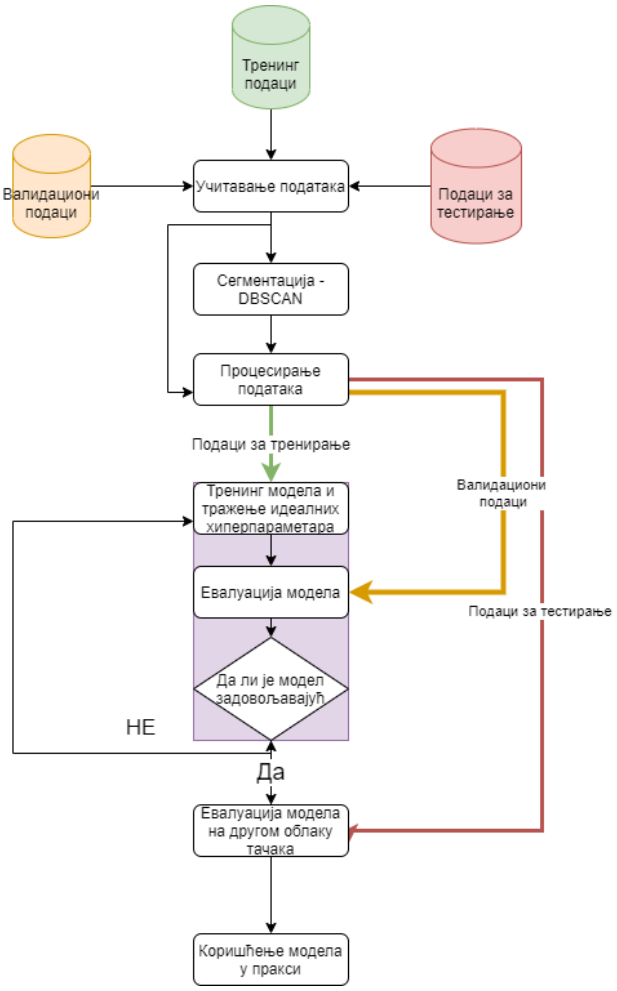
Још један правац у коме се развија примене машинског учења јесте коришћење вештачких неуронских мрежа са претходном селекцијом атрибута. Селецкија атрибута

подразумева креирање нових атрибута на оносву односа и интеракције тачке са тачкама из њеног суседства. Сегментација и рачунање нормала су често применљиве у овом случају.



Слика 4.4 Процес рада тренирања модела машинског учења

Модел обраде који ми предлажемо за овај задатак налази се на следећој слици



Слика 4.5 Модел обраде

Као улазне податке битно је да користимо три различта облака тачака. Један за тренирање, други за валидацију а трећи за верификацију. Основи принцип машинског учења јесте да за верификацију модела користмо податке које он пре није видео. Међутим, уколико би се ти подаци које модел није пре видео, налазили у истом облаку тачака као и подаци са којима смо тренирали, то би довело до тога на имамо претерано оптимистичну верификацију. Уосталом, наме је циљ да направимо модел који ће класификовати облак тачака који није пре видео, а не његове делове који није пре видео.

Затим, зарад тестирања учинковитости сегментација, модели су тренирани на сегментисаним и несегментисаним подацима. Након тога извршава се се прецесирање података како би они били спремни за тренинг.

Тражење оптималног модела и његових хиперпараметара се врши тако што прво тренирамо модел на тренинг подацима, а затим га верификујемо на основу другог облака тачака. Овај процес се понавља одређени број пута. Колико пута се понавља процес зависи од тога колико имамо рачунарских ресурса и колико тачност смо постигли.

Када завршимо овај процес, бирамо модел који је показао најбоље резултате на валидационом облаку тачака. Тај модел потом верификујемо над тест облаком тачака, и након тога наш моделје спреман на употребу.

5. Студија случаја

Задатак ове студије случаја је класификација облака тачака на подручју Новога Сада, користећи алгоритме машинског учења. Конкретно, подручје од интереса приказано је на следећој слици (слика 5.1).

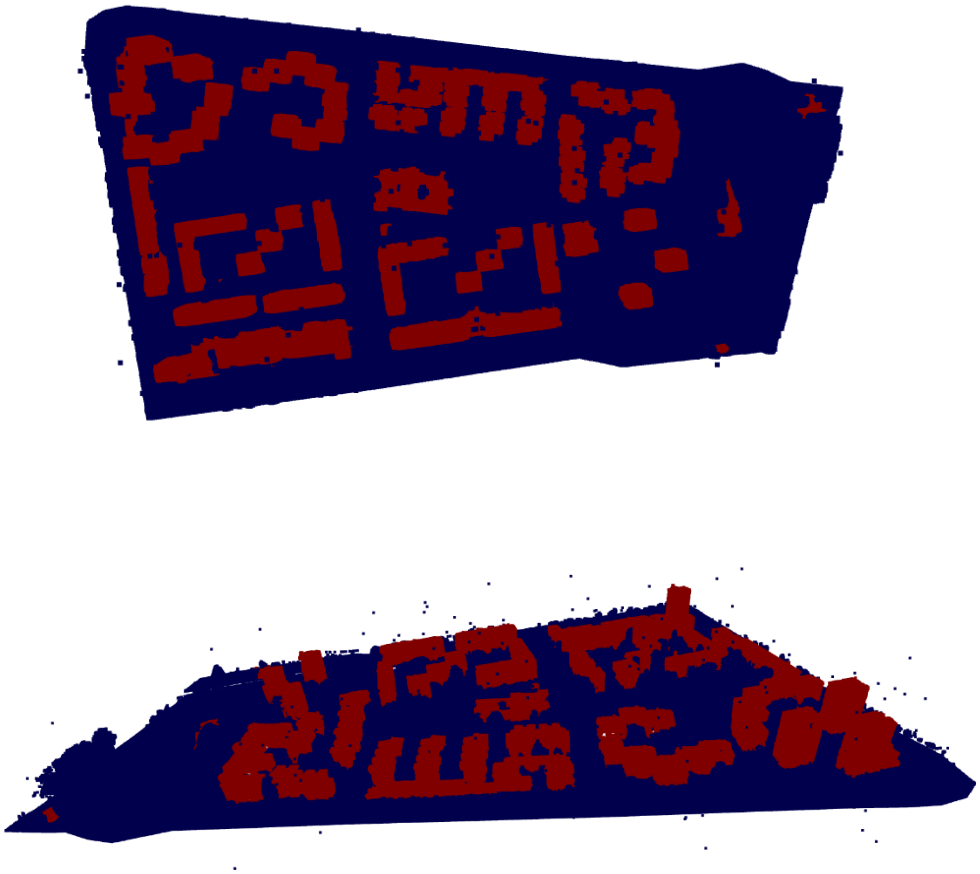
Слика 5.1 Подручје од интереса у Новом Саду

Подручје је ограничено Булеваром Михајла Пупина, Стражиловском и Фрушкогорском улицом као и Дунавом. Облак тачака се састоји од 46 милиона тачака. Подаци су класификовани од стране студената којима је то био дипломски задатак. Они су класификовани у преко 40 класа.

Подаци са којима је овај пројекат рађен су .las формата који поред тога што имају податке о координатама и интензитету одбитка, такође садрже и боју за сваки пиксел изражену у црвеној, плавој и зеленој нијанси. Претпоставка је да би подаци о боји додатно могли да побољшају класификациони модел, а само додавање боје је лако изводљиво помоћу ортофото снимака.

У овој студији случаја вршићемо класификацију за различите класе од интереса. Прво ће бити трениран бинарни класификатор чији је циљ да нађе разлику између тачака које представљају зграде и тачака које представљају остале појаве. Затим ће бити тренирани алгоритми којима је циљ да класификује више класа (тло, пут, зграде,вегетација...).

Подаци који су коришћени за класификацију зграда су приказани на следећим сликама.



Слика 5.2 Подаци коришћени за класфикацију зграда. Црвеном бојом су означено зграде, плавом све остало

Подаци који су коришћени за мултикласну класификацију су приказани на следећим сликама. Наранџастом су означене зграде, зеленом вегетација, плавом су означени тло и шикара а црвеном бојом пут.

Слика 5.3

Слика 5.4

Слика 5.5

Оно што се на овим сликама примећује јесте да су тачке које су класфиковане као тло и тачке које су класификоване као пут доста сличне и да се на доста места ове класе

преплићу. Овакви улазни подаци нису најбољи за тренирање класификатора, уколико желимо да наши алгоритми праве разлику између пута и тла.

Такође, на слици 5.5 примећујемо да има доста тачака које се налазе изнад тачака које представљају терен, и очигледно су настале због сметњи током снимања. Међутим, оне нису уклоњене у процесу постобраде.

Још један проблем у самом тренирању може се јавити у случају када се облаци тачака доста разликују, и нису све класе довољно заступљене у појединим облацима тачака. Тако видимо да се облаку тачака који је приказан на слици 5.5 не налази довољно вегетације. Овакве незаступљености одређених класа у облацима тачака, може довести до тога да као решење добијемо неоптималан модел.

5.1 Препроцесирање података

Пре него што се ишта почне радити у Пајтон програмском окружењу, неопдоходно је учитати податке у формату у коме њему то одговара. Да би податке учитали, користли смо библиотеку за рад са ласерким подацима "*laspy*". Затим смо из учитаног облака тачака пребацили податке (x,y,z координате, интезитет одбитка као и боје за сваки одбитак) у *Pandas DataFrame*. Ово је урађено зато што је Pandas DataFrame погоднији формат за тренирање алгоритама машинског учења. У њему је лакше издвојити тачке различитих класа које нас интересују и са овим издвојеним тачкама да се ради тренирање модела.

Метода коју је овај рад покушао да испита је и улога сегментације у самом процесу рада класификације ласерских података. Идеја је да пре саме класификације буде искоришћен и неки алгоритам сегментације тј. кластеризације. Тако би били добијени додатни атрибути који могу бити употребљени у класификацији.

У овом случају коришћен је алгоритам *DBSCAN*. Разлог због чега је употребљен, јесте то што код овог алгоритма није потребно да сами специфицирамо број кластера као код *KMenas-a*. Такође, овај алгоритам кластерује засебно тачке које су у пољу веће густине, па је претпоставка била да ће алгоритам моћи да кластерује засебно зграде, дрвеће, тло ...

Из разлога што сама вредности коју *DBSCAN* додељује сваком кластеру није много информативна јер је различита из итерације у итерацију, било је потребно да на другачији начин буду представљени резултати сегментације. Предлог је да за сваки кластер буде израчуната средња вредност за висину и за интензитет, као и варијанса за оба ова атрибута. На тај начин, поред информације о сопственој висини и интензитету, свака тачка би, такође, имала информације о средњој вредности и варијанси за висину, и интензитет кластера у ком се та тачка налази.

У току самог рада, одлучено је да се испроба још неколико мера које могу да служе за представљање кластера. Одлучено је да се испроба представљање кластере и помоћу средње вредности висине, црвене и зелене боје.

За проналажење идеалних хиперпараметара коришћен је поступак који је описан у поглављу *DBSCAN*. Међутим, овакав начин тражења параметра не даје увек оптималне резултате, те се добијени параметри често морају кориговати у њиховом окружењу.

Када су пронађени оптимални параметри за *DBSCAN* исвршени су тренинг и кластеризацију на основу координата тачака и за сваку групу је одређена средња вредност и варијанса висине и интензитета.

Следећи корак је био квантизацију боја користећи *KMeans* алгоритам. Међутим, у почетним фазама тренирања, сама квантизација боја није повећала тачност нити је убрзала тренирање, те ова метода даље није коришћена. Када је све ово било урађено, преостало је још да подаци буду скалирани (*Предпроцесирање података).*

Овај процес је одрађен за три различита облака тачака. Један је коришћен за само тренирање модела, други за његову валидацију, а трећи облак тачака је коришћен за коначно тестирање модела. Разлог због чега је ово одрађено на овакав начин, јесте да би се добила боља генерализација модела, тј. његова способност да квалитетно класификује облак тачака који никад раније није видео. Уколико би за био коришћен један облак тачака који би био подељен на сетове за тренинг, валидацију и тестирање, модел би у фази валидације и тестирања видео податке које никад раније није видео. Међутим, ти подаци би били веома слични подацима који су коришћени за тренирање, и самим тим би била добијена претерано оптимистична слика квалитета нашег модела, и било би доведено до тога да наш модел буде изнадподесан.

5.2 Избор, тренирање и подешавање параметара модела

Након што су подаци били предпроцесирани, следећи корак је избор, његово тренирање и подешавање хиперпараметара модела.

Због своје популарности, ефикасности, јачине и могућности да се користе за велике количине података за ову студију случаја, изабрани су алгоритми насумичне шуме , вештачке неуронске мреже и логистичке регресије. За сваки од класификационих проблема у овој студији случаја, биће одрађена класификација коришћењем сва три алгоритма, а након тога направљено поређење.

Када је у питању само тренирања и проналажења оптималних параметара биће употребљене различите методе у зависности од библиотека које су коришћене. Библиотека SKlearn нам пружа погодну функцију *sklearn.model_selection.GridSearchCV* [14]. Ова

функција је намењена за претрагу оптималних параметара међу комбинацијама њиховим комбинацијама.Функција *GridSearchCV* ради тако што тренира дати модел за сваку комбинацију параметара користећи крос валидацију и као резултат нам даје модел са најбољим параметрима за дати проблем.

Поред функције *sklearn.model_selection.GridSearchCV*, употребљена је и функција *sklearn.model_selection.RandomizedSearchCV*. Код ове функције су дефинисане дистрибуције података за вредности хиперпараметара. Из ове дистрибуције, функција *RandomizedSearchCV*, узима вредности и са овим вредностима тренира модел. Предност овог модела јесу случајеви када је опсег простора параметара оргоман.

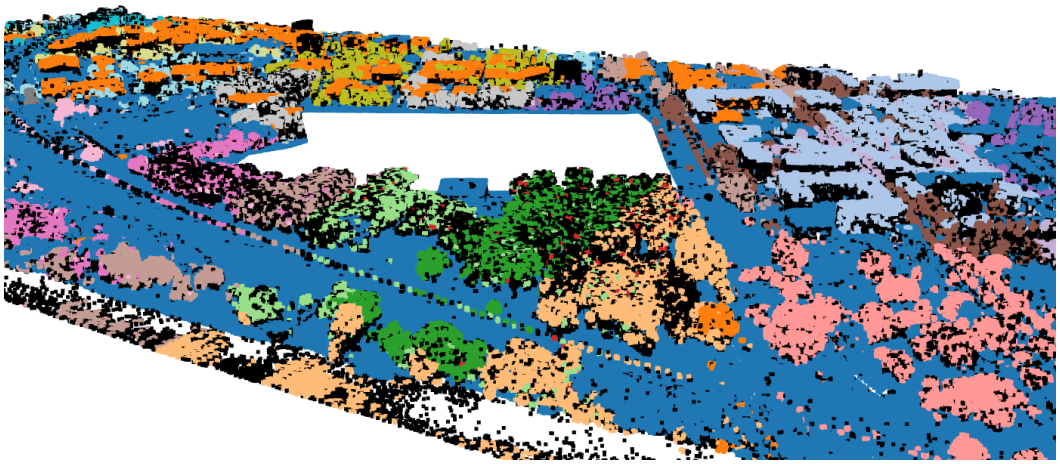
Што се тиче проналажења идеалних параметара за вештачке неуронске мреже, праћене су одређене препоруке, али је процес углавном текао поступком покушаја и грешке (*trial and error*).

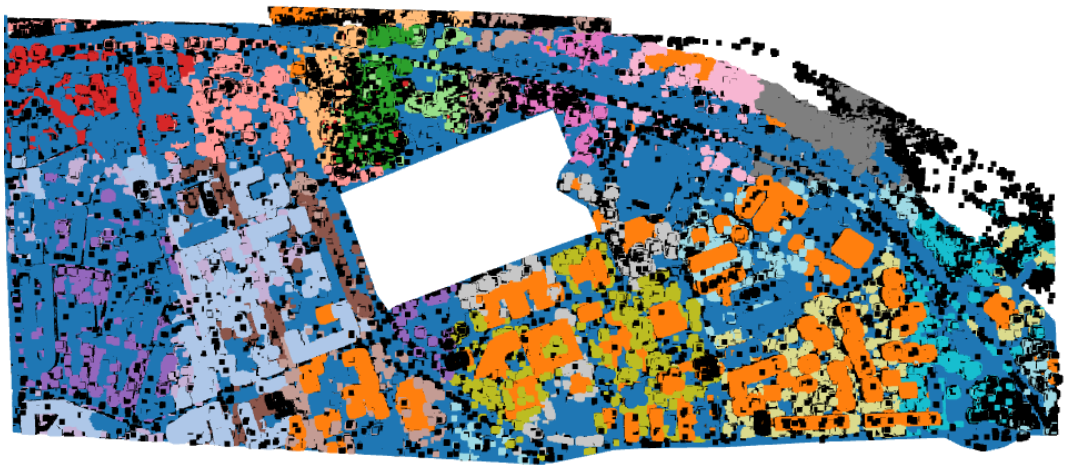
Пошто је циљ овог рада такође провера учинковитости сегментација као корака предпроцесирања у класификацији, сваки од ових модела ће бити подвргнути тренингу на сегментисаним и несегментисаним подацима. Из сегментисаних података су извучена два скупа метрика. Један садржи просечну висину и интезитет као и варијансу, а други скуп метрика садржи просечну висину,црвену и зелену боју. Модели добијени на свим скуповима ће бити упоређени.

5.3 Резултати истраживања

5.3.1 Резултати сегментација

Резултати сегментације облака тачака приказани су на следећој фотографији.





Слика 5.6 Приказ кластеризација податак који су корштени за бинарну класификацију

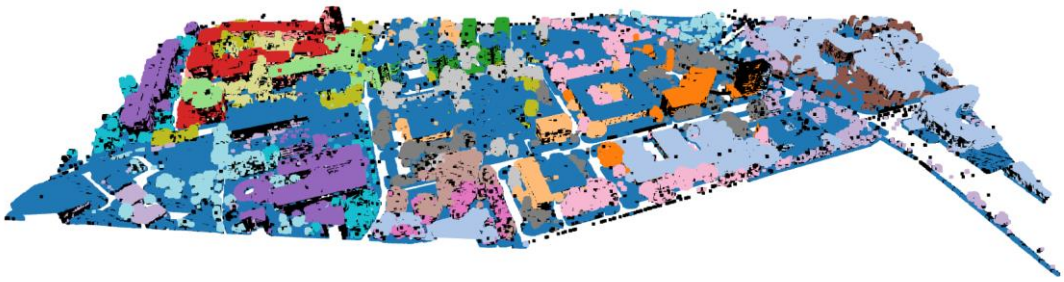
Са слика се може закључити да је алгоритам издвојио плави кластер као највећи. Претпоставка је да тај кластер у највећем делу представља тло и ниску вегетацију, али такође, у овај кластер су сврстане и неке тачке које представљају зграде и дрвеће (са леве стране на горњој слици примећује се да се већина објеката означена плавом бојом међу којима је Извршно веће).

Још једна која се може уведети код кластеровања јесте појава црних тачака. На овој слици оне представљају тачке које DBSCAN није уопште класификовао и оне немају своју групу. Те тачке се третирају као аномалије. Потом су тачке груписане и на основу ових група су извучене одређене метрике. Потенцијалан проблем је то што некластеризоване тачке могу бити примећене код зидова високих зграда, као и на врху дрвећа и осталим пределима са малом густином тачака.

Поред овога може се приметити да је алгоритам доста добро издвојио одређене објекте. Наранџастом бојом су издвојени објекти око студентским домова (доњи десни део и центар слике), и што је можда још важније већина дрвећа које стоји уз објекте, засебно је кластеровано. Такође, може се увидети да су зграде око Стражиловске и Радничке улице(део лево од празнине где нема тачака) исто добро издвојене светло плавом бојом.

Сегментација се веома слично понашала и на осталим облацима тачака. У већини случајева, постојао је један кластер који је обухватао велики облак тачака, углавном тло као и поједино дрвеће и зграде. Поред тога доста дрвећа и објеката је правилно извдојено у сваком облаку тачака након сегментације.

Испод су приказане сегментације на различитим облацима тачака.



Слика 5.7 Сегментисани тренинг облак тачака

Слика 5.8 Сегментисани валидациони облак тачака

Слика 5.9 Сегментисани тестинг облак тачака

5.3.2 Резултати бинарне класификације зграда

Класификација зграда представља бинарну класификацију стога, се од ове класификације очекује највећа тачност. Модел који је дао најбоље резултате за овај случај јесте модел вештачке неуронске мреже. Модел је достигао тачност од 97.11%.

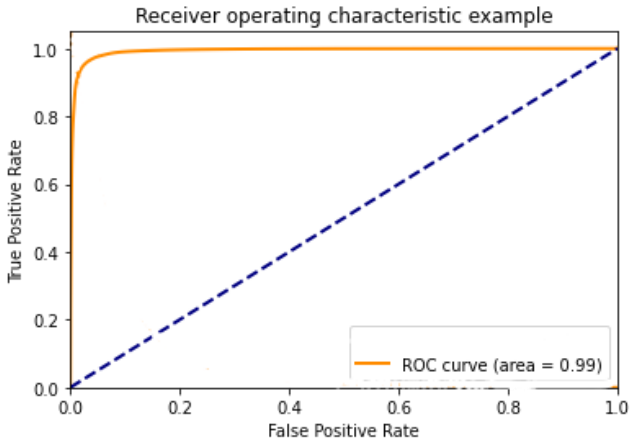
Матрица конфузије је за овај проблем приказана на следећој табели

	Зграде	Незграде
Зграде	1177298	91306
Незграде	78323	5187367

Табела 5.1 Матрица конфузије- класификација зграда

Из матрице конфузије можемо видети да је већина грешака типа 2, тј. Зграде које нису зграде а класификоване као оне. Ипак грешке су доста сличне што показује и чињеница да и прецизност и одзив износе 98%.

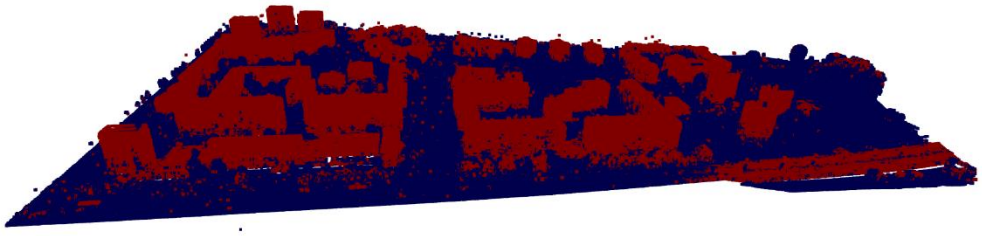
Уколико би постојала жеља да се промене однос прецизности и одзива, било би могуће да променимо праг дискриминације користећи криву радних карактеристика (На пример уколико би желели да модел не пропусти да класификује ни једну тачку зграде).



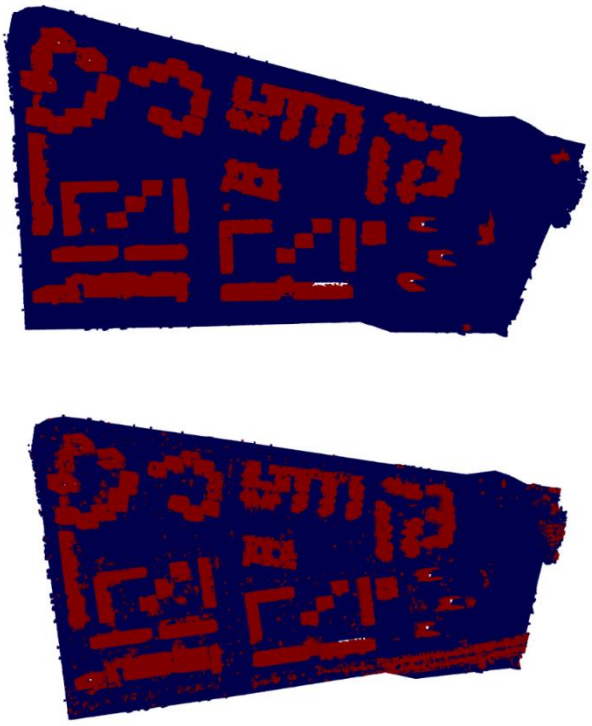
Слика 5.10 Крива радних карактеристика

Модел вештачке неуронске мреже који је дао најбоље резултате има један скривени слој са 15 неурона, и користи регуларизатор типа Л2 и *ReLU* активациону функцију. Модели са више слојева, више неурона или без регуларизације су оверфитовали тренинг податке и нису били у стању да се генерализују на нове податке.

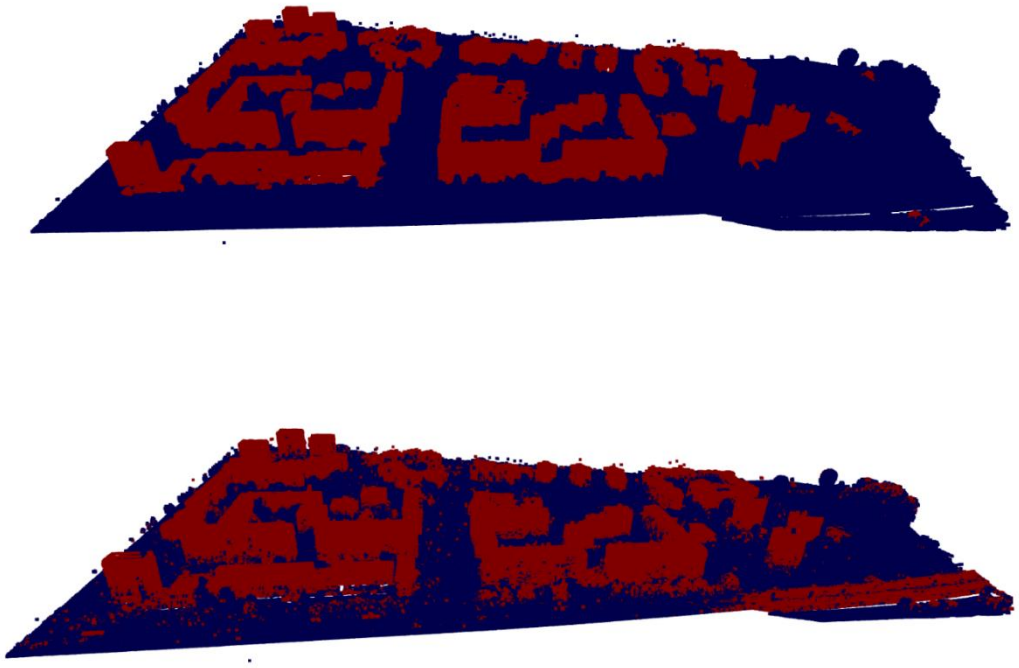
Резултати класификације облака тачака су приказане на следећим сликама. Такође, приказано је поређење између класификованог облака тачака и референтног облака тачака. Тачке које су класификоване као зграде приказане су црвеном, а оне које су класификоване као остало плавом бојом.



Слика 5.11 Класификован тестинг облак тачака



Слика 5.12 Поређење референтног и класификованог облака тачака



Слика 5.13 Поређење референтног и класификованог облака тачака

На сликама се види да су класификовани облаци тачака, на одређеним местима неправилнији у поређењу са референтним. Класификовани облаци тачака немају хомогене групе тачака. Између одређених тачака које представљају зграде, налазе се и оне које то не представљају. Такође, доста је заступљен и обрнути случај, где се тачке које представљају зграде налазе раштркане по оним који представљају остале објекте. Примећујемо и то да је пут са доње десне стране погрешно класификован у класу зграда.

Упркос манама модела, модел је издвојио сваку зграду, и даје верну репрезентацију терена.

5.3.3 Класификација тла, зграда и вегетације

Због своје сличности, у овој класификацији коришћене су тачке које су класфиковане као тло и шикара под једном класом- тло. У овом случају, поред зграда, циљ је био и класификација вегетације.Модел који је најбоље класификовао ове тачке јесте модел насумичне шуме. Модел насумичне шуме је достигао прецизност од 95.56% сегментираним подацима.

	Тло	Зграде	Вегетација
Тло	1767807	60450	10234
Зграде	11035	2057516	123856
Вегетација	4359	81066	2435544

Табела 5.2 Матрица конфузије класификације тла, зграда и вегетације

У табели 5.2 примећује се да је највећа конфузија између класа зграда и вегетације. Доста тачака које припадају класи зграде су класификовани као вегетација, и обрнуто. То је донекле и очекивано зато што сваки алгоритам даје доста битности самој висини, а висока вегетација и ниже зграде су исте висине.

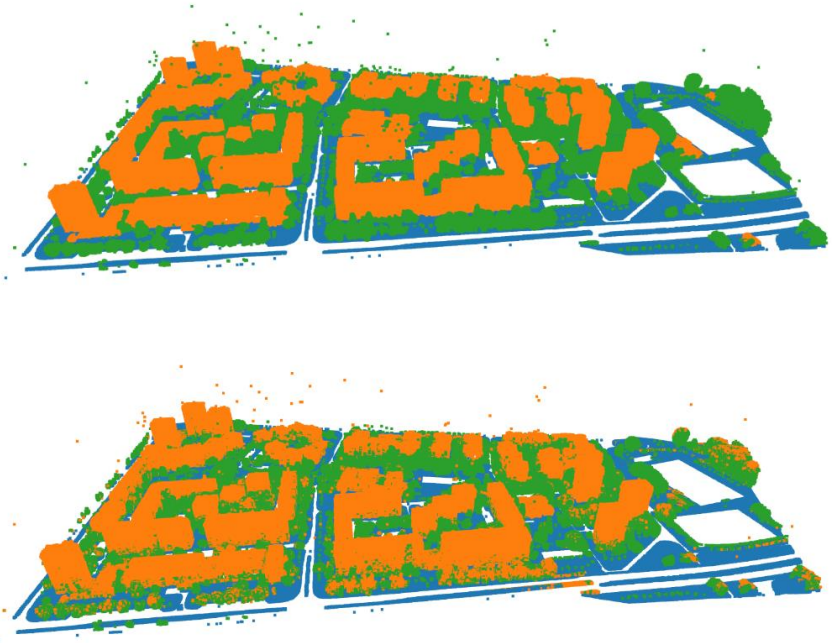
Модел насумичне шуме који је коришћен за ову класификацију се састоји од 50 стабала одлучивања, и сваки лист у сваком стаблу мора да садржи макар 0.1% укупног броја тачака.

Модел који је имао другу највећу тачност јесте модел вештачке неуронске мреже, који је имао тачност од 95.18%. Овај модел је трениран на подацима који нису сегментирани.

Резултати класификације облака тачака су приказни на сликама испод. Такође је приказано поређење између класификованог облака тачака и референтног облака

Слика 5.14 Класификовани тест облак тачака

Слика 5.15 Поређење референтног и класификованог облака тачака



Слика 5.16 Поређење референтног и класификованог облака тачака

5.3.4 Класификација тла, зграда, вегетација у пута

У овој класификацији су додати пут и паркинг да би се видело да ли алгоритми могу да диференцирају асфалтни пут и паркинг у односу тло и шикару. Резултати које су добијени показују да и најбољи модел класификује само 64.50% пута као пут, док је 35.25% погрешно класификује као тло, што је и очекивано с обзиром на сличност између ове две класе у висинском смислу.

Најбољи модели за овај тип класификације јесу модел насумичне шуме као и модел вештачке неуронске мреже. Оба ова модела су тренирани на несегментираним облаком тачака, што нам указује на то да сегментација није добро када имамо тачке различитих класе које теже да буду кластероване у исти групу.

Модел насумичне шуме је користио исте хиперпараметре као и претходни (из класификације тла зграда и вегетације), а што се тиче неуронске мреже, она се састојала из једног скривеног слоја са 13 неурона, *ReLU* активационе функције , а излазни слој је имао *Softmax* функцију.

Модел насумичне шуме је имао тачност од 85.45%, док је модел неуронске мреже имао 85.16%. Разлика између одва два модела је да модел насумичне шуме Разлика између ова два модела јесте то што је моделанасумичне шуме прави више грешака другог, док модел вештачке неуронске мреже прави више грешака првог типа.

	Тло	**Зграде**	**Вегетација**	**Пут**
Тло	4503209	30813	41216	453130
Зграде	67117	2364457	184812	4544
Вегетација	70366	169682	2691599	950
Пут	640854	3465	978	1172686

Табела 5.3 Матрица конфузије- класификација тла, зграда, вегетације и пута помоћу алгоритма насумичне шум

	Тло	**Зграде**	**Вегетација**	**Пут**
Тло	3259730	13039	54169	649001
Зграде	23785	2105299	130946	6663
Вегетација	11366	139197	2367739	2745
Пут	494594	13807	793	1110118

Табела 5.4 Матрица конфузије- класификација тла, зграда, вегетације и пута помоћу алгоритма вештачке неуронске мреже

5.3.5 Поређење класификације са и без сегментације

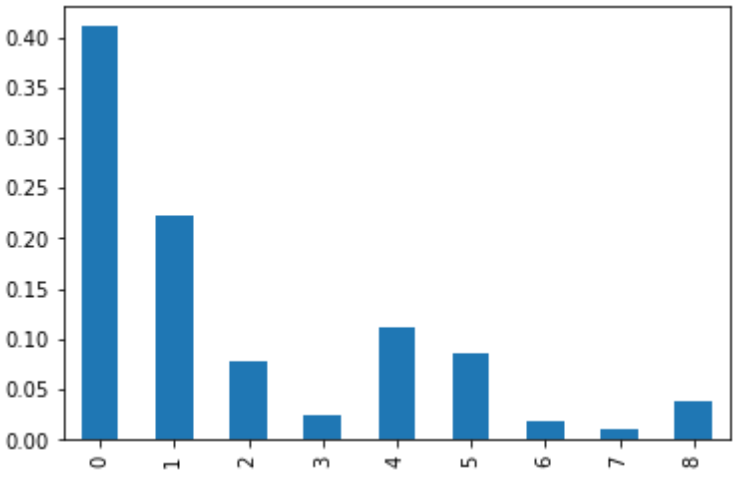
Сегментација облака тачака коришћењем методе *DBSCAN*-а, показала је различите резултате. За одређене моделе на одређеним облацима тачака, показало се да сегментација побољшава тачност, док се на другим моделима испоставило чак и да боље раде они у којима није примењена сегментација. Резултати класификација са и без сегментације на различитим задацима класификације су приказане на следећој табели

Гледајући табелу можеме се закључити да се модели који су користили сегментисане податке дали доста боље резултате у односу на оне који су тренирани на несегментисаним подацима, у случају када смо класификовали зграде. Могући разлог за то јесте могућност *DBSCAN-а* да успешно кластерује тачке зграда у засебне групе.

	Алгоритам насумичне шуме			Логистичка регресија			Вештачка неуронса мрежа		
	сегментација	сегментација боја	без сегментације	сегментација	сегментација боја	без сегментације	сегментација	сегментација боја	без сегментације
Бинарна класификација зграда	96.55%	95.56%	95.86%	95.25%	90.21%	93.98%	97.11%	95.96%	96.72%
Класификација 3 класе - тла зграда и вегетације	95.56%	94.40%	94.57%	92.85%	94.18%	94.35%	91.64%	94.37%	95.18%
Класификације 4 класе- тла зграда, вегетације и пута	77.05%	79.83%	85.45%	81.72%	82.12%	80.77%	79.42%	85.45%	85.16%

Табела 5.5 Поређење модела на различитим задацима са и без сегментације

Тренирањем алгоритма насумичне шуме добили смо меру колико је сваки атрибут допринео класификацији



Слика 5.17 Битност атрибута

На слици видимо да је атрибут број 5 који представља средњу висину кластера којем тачка припада, четврти најбитнији у самој класификацији. То је један од објашњења зашто сегментација ради боље, када је у питање класификација зграда у односу на остале класификације. Кластери који представљају зграде имају затно већу просечну висину о односу на остале кластере и стога сегментацији помаже алгоритму да диференцира зграда од осталих класа.

Што се тиче класификације тла, зграда и вегетације, ту не примећујемо претерано разлику између модела који су тренирани на сегментисаном у поређењу са онима који су тренирани на несегментисаном облаку тачака. Разлике у резултатима могу да се припишу и случајном процесу.

Класификација тла, зграда, вегетације и пута исто не показује разлике и даје резултате који нису статистички значајни. За логистичку регресију и вештачку неуронску мрежу се испоставило да је сегментацији помогла, док је за насумичну шуму сегментација одмогла моделу. Разлог сметње код ове врсте класификације јесте то што су тачке тла и пута на многим местима заједно кластероване а представљају различите класе.

6. Закључак

Тема овог рада била је да била је да се истражи примена алгоритама машинског учења за класификацију облака тачака добијених LIDAR методом премера, као и поређење успешности различитих алгоритама машинског учења на сегментираним и не сегментираним подацима.

Из ове студије закључено је да коришћење алгоритама машинског учења је обећавајући метод када је у питању класификација облака тачака. Са нашим лимитираним средствима, и за кратак временски период усели смо да истренирамо модел који је у стању да екстракује из облака тачака скоро све објекте и да да верну репрезентацију класификованог облака тачака.

Сегментација као корак у предпроцесирању није дала резултате онолико добре колико се од ње то очекивало. Сегментација је успела да повећа тачност у случају бинарне класификације, али није имала толико успеха у милтикласној класификацији. Мора се узети у обзир и то што сама сегментација додаје доста рачунарске комплексности и изискује додатно време и ресурсе.

Наравно, уколико се кани да се направи модел који би могао да класификује било који облак тачака, мора се тренирати алгоритам над огромном количином података, на којима су све класа које можемо да сусретнемо довољно заступљене. Примера ради наше подручје од интереса није укључивало воду. Уколико би задатак био и класификације воде, а ми нисмо имало облака тачака са довољном заступљеношћу воде, нам алгоритам се не би добро понашао када би видео воду.

Уз употребу јачих рачунара, обимнијих, разноврснијих и квалитетнијих података, као и оптималнијих модела класификације и сегментације, потенцијално би се могао направити модел који прави минималне грешке и задовољава све индустријске стандарде. Овакав

модел би потенцијално уштедео доста времена, рада и новца и уз минималну људску корекцију омогућио скоро па моменталну класификацију облака тачака.

Да би ово успели, поред бољих података и рачунарских ресурса, неопходно је да се одради још истраживања на пољима сегментације и екстракције атрибута из сегментираних података као и на пољима добијања тачнијим модела класификације.

7. Литература

[1] E. Alpaydin, Introduction to Machine Learning, The MIT Press Cambridge, Massachusetts London, England, 2004.

[2] Миро Говедарциа, Материјали са предавања, Ласерско скенирање терена и објеката

[3] Inertial Navigation Systems and Its Practical Applications By Aleksander Nawrat, Karol Jędrasiak, Krzysztof Daniec and Roman Koteras

[4] GNSS DATA PROCESSING Volume I: Fundamentals and Algorithms

[5] An Introduction to GNSS GPS, GLONASS, BeiDou, Galileo and other Global Navigation Satellite Systems NovAtel Inc

[6] Tropospheric Correction for InSAR Using Interpolated ECMWF Data and GPS Zenith Total Delay Measurements From the Southern California Integrated GPS Network

[7] Hardware Accelerated Compression of LIDAR Data Using FPGA Devices

[8] Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems Aurélien Géron

[9] Where Deep Learning Meets GIS, Rohit Singh ArcWatch

[10] Fürnkranz J. (2011) Decision Tree. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_204

[11] „Decision Trees in Machine Learning“, Prashant Gupta

[12] Techopedia What Does Input Layer Mean? https://www.techopedia.com/definition/33262/input-layer-neural-networks

[13] Nadia Rahmah and Imas Sukaesih Sitanggang 2016 IOP Conf. Ser.: Earth Environ. Sci. 31 012012

[14] Henryk Palus ,On Color Image Quantization by the K-Means Algorithm

[15] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[16] Arthur, David and Vassilvitskii, Sergei (2006) k-means++: The Advantages of Careful Seeding. Technical Report. Stanford.

[17] Александар Андрејевић, Класификација ALS података методама машинског учења 2018

[18] Lachtermacher, G., Fuller, J.D., 1995. Backpropagation in timeseries forecasting. Journal of Forecasting 14, 381–393.

[19] Lippmann, R.P., 1987. An introduction to computing with neural nets, IEEE ASSP Magazine, April, 4–22.

[20] Hecht-Nielsen, R., 1990. Neurocomputing. Addison-Wesley, Menlo Park, CA

[21] Wong, F.S., 1991. Time series forecasting using backpropagation neural networks.Neurocomputing 2, 147–159.

[22] Tang, Z., Fishwick, P.A., 1993. Feedforward neural nets as models for time series forecasting. ORSA Journal on Computing 5 (4), 374–385.

[23] Sebastian Ruder,“An overview of gradient descent optimization algorithms” https://ruder.io/optimizing-gradient-descent/

[24] IBM, What is linear regression? https://www.ibm.com/topics/linear-regression

[25] Saishruthi Swaminathan ,”Linear Regression — Detailed View” https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86

[26] Peter Bruce, Andrew Bruce,” Practical Statistics for Data Scientists”

[27] IBM, What is logistic regression? https://www.ibm.com/topics/logistic-regression

[28] IBM, „Overfitting“ https://www.ibm.com/cloud/learn/overfitting

[29] Prashant Gupta,Regularization in Machine Learning Prashant Gupta https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a

[30] IBM, „underfitting“ https://www.ibm.com/cloud/learn/underfitting

[31] Seema Singh, "Understanding the Bias-Variance Tradeoff", https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

[32] Anuja Nagpal "L1 and L2 Regularization Methods ", https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

[33] Amitrajit Bose,"Cross Validation — Why & How " https://towardsdatascience.com/cross-validation-430d9a5fee22

[34] Geoffrey Hinton Neural Networks for machine learning nline course. https://www.coursera.org/learn/neural-networks/home/welcome

[35] DeepAI "RMSprop definition" https://deepai.org/machine-learning-glossary-and-terms/rmsprop

[36] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations

[37] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[38] Sarang Narkhede, "Understanding AUC - ROC Curve " https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[39] Receiver operating characteristic (23.09.2021) in Wikipedia https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[40] Michael Nielsen "Neural networks and Deep learning" ,Determination press 2015

[41] What is python? executive summary. https://www.python.org/doc/essays/ blurb/.

[42] Top Computer Languages ,https://statisticstimes.com/tech/top-computer-languages.php

[43] Scipy.org. https://www.scipy.org/.

[44] Pandas vs. NumPy https://www.javatpoint.com/pandas-vs-numpy

[45] Sagar Sharma,"Activation Functions in Neural Networks", https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[46] Dejan Vasić, "Model geodetskog premera savremenim akvizicionim tehnologijama"

[47] TerraScan User Guide, https://terrasolid.com/guides/tscan/crground.html

[48] F. Patricia Medina, Randy Paffenroth,"Machine Learning in LiDAR 3D point clouds"

[49] Qi, Charles R and Su, Hao and Mo, Kaichun and Guibas, Leonidas J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. arXiv preprint arXiv:1612.00593, 2016.

[50] Qi, Charles R and Yi, Li and Su, Hao and Guibas, Leonidas J. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. arXiv preprint arXiv:1706.02413, 2017.

[51] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, Baoquan Chen,"PointCNN: Convolution On X-Transformed Points"

8. Програмски код

8.1 Импортовање библиотека

```
import open3d as o3d
import laspy as lp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import joblib


from scipy import stats
from sklearn.cluster import DBSCAN
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV,Pred
efinedSplit
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.neighbors import NearestNeighbors
from sklearn.utils import shuffle

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,BatchNormalization
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping


from sklearn.metrics import accuracy_score,confusion_matrix,silhouette_score,precision_score,r
ecall_score,f1_score
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

8.2 Дефинисање функција за процесирање података

```
'''Funkcija na osonvu zscore vrednosti koji utklanja tacke bazirano na tome da li visina tacke se nalazi mnogo
iznad ostalih  tacaka. Ideja je da funkcija otlkloni grube greske ali u zavisnoti od parametra Z moze i da otkloni tacke
koje su na visokom objektu (dom za asistente)'''

def remove_outliners(points,zscore):
    z = np.abs(stats.zscore(points['z']))
    return points[z<zscore].copy()

'''Funkcija splitvuje podatke na X matricu koja sadrzi vektore karakteristika koji ne ukljucuje X,Y koordinate, zato sto nam
one nisu pogodne za samo treniranje a pogodone su nam za segmentaciju, stoga ovu funkciju pozivamo tek kada izvrsimo
segmentaciju '''

def splitovanje(podaci):
    podaci_train = podaci.drop(['x','y'],axis=1).dropna()
    X = podaci_train.drop('klase',axis=1)
    y = podaci_train['klase'].copy()
    return shuffle(X,y)


"""Funkcija koja plotuje sortirane distance svake tacke do svog treceg komsije """

def kurvature(data,tacka = None):
    points = data[['x','y','z']].copy()
    min_samples = 2*len(points.columns)
    neighbor = NearestNeighbors(n_neighbors = min_samples)
    neighbor.fit(points)
    distance, instance = neighbor.kneighbors(points)
    distance = np.sort(distance, axis=0)
    distances = distance[:,2]
    tangente =  np.gradient(distances)
    tangente_2=np.gradient(tangente)
    srednja_tangenta = tangente.mean()
    a = tangente[np.argmin(np.abs(tangente - srednja_tangenta))]
    c = instance[np.argmin(np.abs(tangente-a))]
    epsilon =(distances[c][0])
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(211)
    ax.plot(distances)
    ax.scatter (c,distances[c])
    ax.scatter(c[0],epsilon,color = 'red')
    ax = plt.subplot(212)
    ax.plot(distances)
    ax.scatter (c,distances[c])
    ax.scatter(c[0],epsilon,color = 'red')
    ax.axis([np.min(c)-2e5,np.max(c)+5e5,0,epsilon+5])
    print(distances[c])
```

```
    print('Uzeta vrednost za epsilon je:',epsilon)
    plt.show()

    if tacka is not None:
        plt.figure(figsize=(12, 4))
        plt.plot(distances)
        plt.scatter(tacka,distances[tacka],color = 'green')
        plt.axis([tacka-2e5,tacka+5e5,0,distances[tacka]+5])
        plt.show()
        print('Vrednost odabrane tacke je:',distances[tacka])
        return epsilon,min_samples,tacka

    return epsilon,min_samples


'''Funkcija grupneStatistike kao ulaz prima podatke koju za svaku instancu imaju grupu kojoj pri
padaju. Zatim grupuiše podatke
na osnovu te grupu i izračuna prosečnu visinu i itezitet kao i standardnu devijaciju za svaku tačk
u, zatim te izračunate
vrednosti merge-uje sa originalnim podacima preko klastera kojim pripadaju i pošto nam sam po
datak o grupi više ne treba
uklolnima ga '''

def grupneStatistike(podaci):
    z_mean = podaci.groupby('grupa').z.mean()
    red_mean = podaci.groupby('grupa').red.mean()
    green_mean = podaci.groupby('grupa').green.mean()
    g = pd.concat([z_mean,red_mean,green_mean],axis=1)
    return pd.merge(podaci,g,left_on ='grupa',right_index =True).drop('grupa',axis=1)


'''def grupneStatistike(podaci):
    z_stat = podaci.groupby('grupa').z.agg(['mean','std'])
    inten_stat = podaci.groupby('grupa').intensity.agg(['mean','std'])
    first = pd.merge(podaci,z_stat,left_on = 'grupa',right_index = True)
    return pd.merge(first,inten_stat,left_on ='grupa',right_index =True).drop('grupa',axis=1)'''

'''Funkcija segmentacija vrši segmentaciju tačaka pomoću DBSCAN-a i svakoj tački dodeljuje se
gment kome pripada
Kada je segmentacija završena funkcija takodje izbacuju vizualizaciju te naše segmentacije, i na
kraju poziva prethodne
dve funkcije (grupneStatistsike i splitovanje) i to funckija vraća'''

def segmentacija(data,epsilon,min_samples):
    points = data[['x','y','z']].copy()
    colors = data[['red','green','blue']].copy()

    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(points.values)
    dbscan = DBSCAN(eps= epsilon, min_samples= min_samples)# do ovih parametara sam dos
ao u drugom Notebook-u koristići motodu koju sam u wordu opisao
```

```
    dbscan.fit(points)
    labels = dbscan.labels_
    max_labels = labels.max()
    colors_ = plt.get_cmap('tab20')(labels/max_labels if max_labels > 0 else 1)
    colors_[labels ==-1] = 0
    pcd.colors = o3d.utility.Vector3dVector(colors_[:,:3])
    o3d.visualization.draw_geometries([pcd])
    data['grupa'] = labels.reshape(-1,1)

    return splitovanje(grupneStatistike(data)).copy()

"""Ideja funkcije redukcija_boja je da se odradi kvantizacija boja na osnovu KMeans- algoritma
"""


def redukcija_boja(kmeans,podaci):
    colors = podaci[['red','green','blue']].copy()
    boje_klasteri = kmeans.predict(colors)
    podaci[['red','green','blue']] = kmeans.cluster_centers_[boje_klasteri]
    return podaci

"""Funkcija koja spaja prethodne funkcije u jednu"""

def preprocess_data(data,zscore,epsilon,min_samples):
    data1 = remove_outliners(data,zscore)
    return segmentacija(data1,epsilon,min_samples)

def preprocess_data_boje(data,kmeans,zscore,epsilon,min_samples):
    data1 = remove_outliners(data,zscore)
    data2 = redukcija_boja(kmeans,data1)
    return segmentacija(data2,epsilon,min_samples)

"""Funkciaj koja kao izlaz daje preciznost modela i matricu konfuzije nad trening i test podacim
a, kao i bitnost feature-a
ukoliko je model RandomForest"""

def izvestaj_statistike(y,y_pred,train = True):
    if train is True:
        print('Tacnost na trening podacima je:\n',accuracy_score(y,y_pred),'\n')
        print('Matrica konfuzije za trening podatke:\n')
        trening_konfuzija=pd.DataFrame(confusion_matrix(y,y_pred),columns = klase,index = klas
e)
        display(trening_konfuzija)
        print('Preciznost nad trening podacima:',precision_score(y,y_pred,average = 'macro'))
        print('Odziv nad trening podacima je:',recall_score(y,y_pred,average = 'macro'))
        print('F1 vrednost nad trening podacima je:',f1_score(y,y_pred,average = 'macro'))
        trening = trening_konfuzija.values
        np.fill_diagonal(trening,0)
        plt.imshow(trening,cmap = plt.cm.gist_heat_r)
        plt.show()
    if train is False:
        print('Tacnost na test podacima je:\n',accuracy_score(y,y_pred),'\n')
        print('Matrica konfuzije za test podatke:\n')
```

```
        testing_konfuzija=pd.DataFrame(confusion_matrix(y,y_pred),columns = klase,index = klas
e)
        display(testing_konfuzija)
        print('Preciznost nad testing podacima:',precision_score(y,y_pred,average = 'macro'))
        print('Odziv nad testing podacima je:',recall_score(y,y_pred,average = 'macro'))
        print('F1 vrednost nad testing podacima je:',f1_score(y,y_pred,average = 'macro'))
        testing = testing_konfuzija.values
        np.fill_diagonal(testing,0)
        plt.imshow(testing,cmap = plt.cm.gist_heat_r)
        plt.show()


def izvestaj(model,X_train,X_test,y_train,y_test,klase=None,columns = None):

    if str(type(model)) =="<class 'tensorflow.python.keras.engine.sequential.Sequential'>":


        if len(np.unique(y_train))!=4:
            y_predict = model.predict(X_train).argmax(axis=1)
            izvestaj_statistike(y_train,y_predict)

            y_predict_test = model.predict(X_test).argmax(axis=1)
            izvestaj_statistike(y_test,y_predict_test,train= False)
        if len(np.unique(y_train))==2:
            y_predict = model.predict(X_train)>0.5
            izvestaj_statistike(y_train,y_predict)

            y_predict_test = model.predict(X_test)>0.5
            izvestaj_statistike(y_test,y_predict_test,train= False)
    else:

            y_predict = model.predict(X_train)
            izvestaj_statistike(y_train,y_predict)



            y_predict_test = model.predict(X_test)
            izvestaj_statistike(y_test,y_predict_test,train= False)



    try:
        if str(type(model))=="<class 'sklearn.ensemble._forest.RandomForestClassifier'>":
            importance = pd.Series(data = model.feature_importances_,index = columns)
            importance.plot(kind = 'bar')
            plt.show()
    except:
        print()
    try:
        if str(type(model.best_estimator_))=="<class 'sklearn.ensemble._forest.RandomForestClass
```

```
ifier'>":
            importance = pd.Series(data = model.best_estimator_.feature_importances_,index = colu
mns)
            importance.plot(kind = 'bar')
            plt.show()
    except:
        print()

# Funkcije za vizualizaciju podataka
def vizualizacija(X,y,podaci):
    koordinate =podaci.loc[X.index,['x','y','z']]
    broj_klase=len(np.unique(y))
    if broj_klase==2:
        colors = plt.get_cmap('seismic_r')(y/1)
    else:
        colors = plt.get_cmap('tab10')(y)

    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(koordinate.values)
    pcd.colors = o3d.utility.Vector3dVector(colors[:,:3])

    o3d.visualization.draw_geometries([pcd])

def vizualizacija_proba(X,y,podaci):
    koordinate =podaci.loc[X.index,['x','y','z']]
    broj_klase=len(np.unique(y))
    if broj_klase==2:
        colors = plt.get_cmap('seismic_r')(y/1)
    else:
        colors = y.map({0:[0.97647059, 0.65490196, 0.24313725],1:[0.15294118, 0.70196078, 0.4
627451 ]
                        ,2:[0.74901961, 0.12941176, 0.18431373],3:[0., 0.43529412, 0.23529412]})

    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(koordinate.values)
    pcd.colors = o3d.utility.Vector3dVector(colors)

    o3d.visualization.draw_geometries([pcd])
```

8.3 Мултикласна класификација

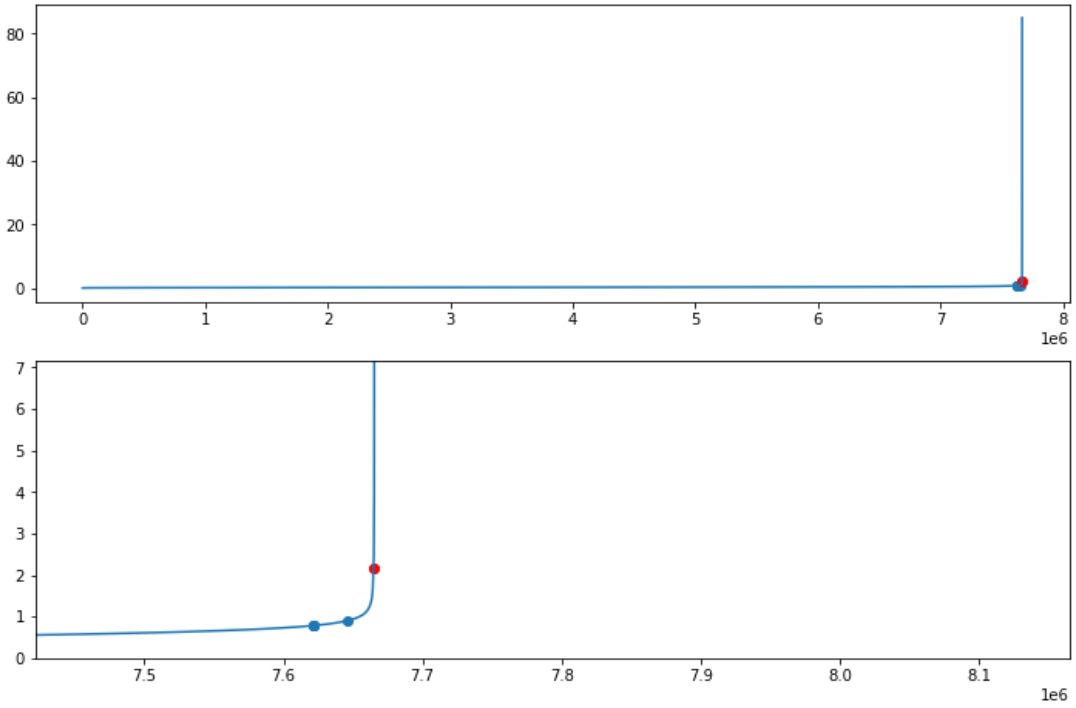
```
data_valid = pd.read_csv(r"C:\Users\ilija\Fax\Diplomski\Seminarski Septembar\Novi podaci\Tri
Klase_gornji.csv")
data_train = pd.read_csv(r"C:\Users\ilija\Fax\Diplomski\Seminarski Septembar\Novi podaci\Tri
Klase_donji.csv")

klase = ['Tlo','Zgrade','Vegetacija']
broj_klasa = len(klase)

data_train.head()
```

	x	y	z	red	green	blue	intensity	klase
0	409750.97	5011004.76	78.22	42496.0	41984.0	42496.0	64.0	0

```
1  409750.83  5011005.15  78.12  43264.0  42752.0  43264.0     188.0     0
2  409751.12  5011005.07  78.24  43776.0  44032.0  44288.0     114.0     0
3  409751.09  5011005.66  78.10  43776.0  43520.0  43520.0     167.0     0
4  409751.44  5011005.56  78.12  43520.0  43520.0  43520.0     135.0     0
```

```
epsilon,min_samples = kurvature(data_train)
```

```
[2.15269598 0.78364533 0.90376988 0.78370913 0.78351771 0.78364533]
Uzeta vrednost za epsilon je: 2.1526959840750397
```




```
epsilon = 0.66
min_samples = 6
```

```
X,y = preprocess_data(data_train,3.5,epsilon,min_samples)
```

```
vizualizacija(X,y,data_train)
```

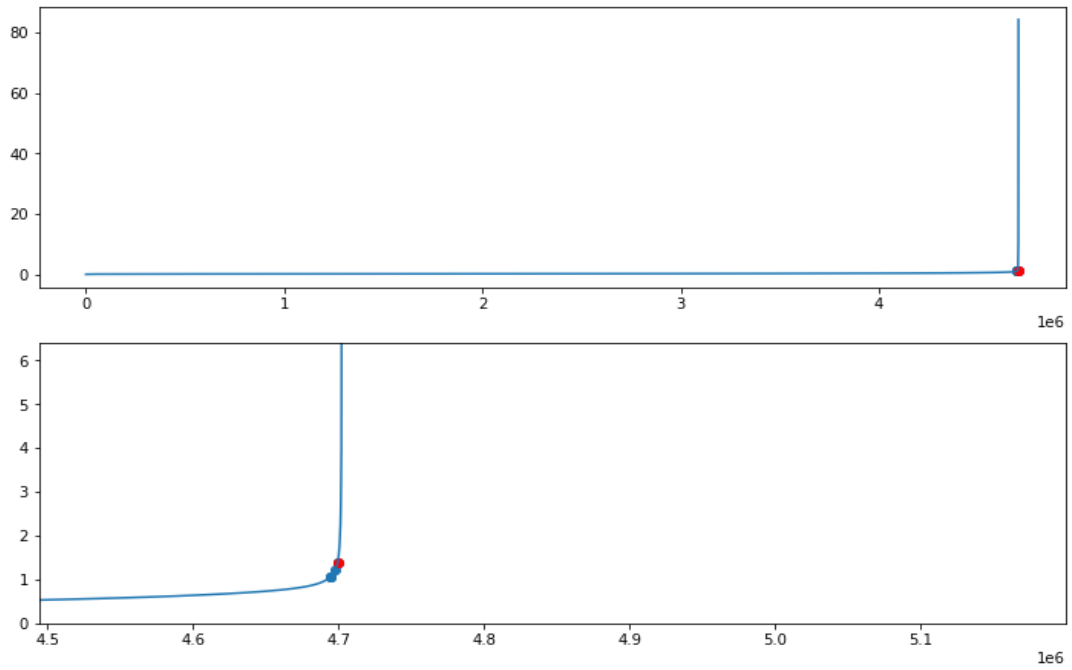
```
[Open3D WARNING] GLFW Error: WGL: Failed to make context current: The requested transf
ormation operation is not supported.
```

```
transformer = ColumnTransformer([
    ('scaler',StandardScaler(),['z_x', 'red_x', 'green_x', 'blue', 'intensity', 'z_y', 'red_y',
       'green_y'])
],remainder = 'passthrough')
```

```
X_scaled = transformer.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled,columns = X.columns,index= X.index)
```

```
epsilon_valid,min_samples_valid = kurvature(data_valid)
```

```
[1.39405882 1.39405882 1.2292274  1.05479856 1.05494076 1.05489336]
Uzeta vrednost za epsilon je: 1.3940588223477308
```




```
epsilon_valid = 0.64
min_samples_valid = 6

X_valid,y_valid = preprocess_data(data_valid,3.6,epsilon_valid,min_samples_valid)

X_valid_scaled = transformer.transform(X_valid)
X_valid_scaled_df = pd.DataFrame(X_valid_scaled,columns = X_valid.columns,index = X_vali
d.index)

vizualizacija(X_valid,y_valid,data_valid)
```

[Open3D WARNING] GLFW Error: WGL: Failed to make context current: The requested transf
ormation operation is not supported.

```
X_train = pd.concat([X_scaled_df,X_valid_scaled_df])
y_train = pd.concat([y,y_valid])

X
```

	z_x	red_x	green_x	blue	intensity	z_y	red_y	\
5765341	79.64	29184.0	33536.0	24064.0		10.0	81.186597	32805.308527
4304474	78.12	32512.0	29952.0	30976.0		43.0	78.900789	51080.982456
3538340	85.81	49664.0	52224.0	52736.0		165.0	87.821099	32126.333827
2295887	86.91	50944.0	42240.0	38656.0		161.0	86.577993	36382.903871
7041040	80.06	25088.0	26880.0	24832.0		11.0	81.555626	31212.598495
...	...	...	...	...	...	...	...	...
224128	76.87	39936.0	40192.0	35072.0		63.0	76.599339	33161.197532
2618946	94.95	35840.0	31488.0	31488.0		45.0	98.134718	45890.620435
208406	77.51	30464.0	31488.0	28160.0		100.0	77.350307	38826.143149
6827026	83.46	32256.0	35584.0	26368.0		8.0	83.442074	31320.316109
6154813	92.79	17664.0	20992.0	22784.0		77.0	93.395727	21096.023101

	green_y
5765341	33610.021705
4304474	47602.526316

```
3538340  34600.537375
2295887  33104.443758
7041040  31022.292561
...              ...
224128   34876.257807
2618946  40704.866100
208406   39298.006817
6827026  31971.793313
6154813  22675.702197

[7638787 rows x 8 columns]
```

8.4 Класификација помоћу стохастичког градијентног пада

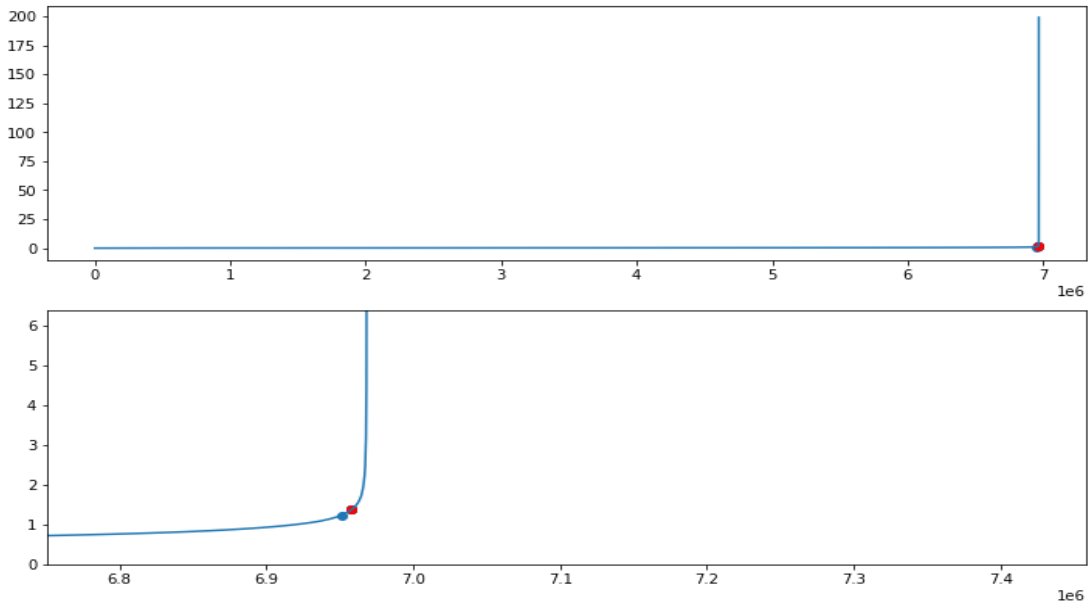
```
split_index = np.full(len(X_train),-1)
split_index[len(X):]=0

fold = np.array(split_index)

pds = PredefinedSplit(test_fold = fold)

clf = SGDClassifier( class_weight='balanced', penalty='elasticnet', random_state=42)
param_dist = {'l1_ratio': stats.uniform(0, 1),
          'alpha': stats.loguniform(1e-4, 1e0)}


n_iter = 10
rand_search = RandomizedSearchCV(clf,param_dist,n_iter=n_iter,n_jobs = -1,cv=pds)

rand_search.fit(X_train,y_train)

RandomizedSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ...,  0,  0])),
                   estimator=SGDClassifier(class_weight='balanced',
                                           penalty='elasticnet',
                                           random_state=42),
                   n_jobs=-1,
                   param_distributions={'alpha': <scipy.stats._distn_infrastructure.rv_frozen object at 0
x000001E70A4D21C0>,
                                        'l1_ratio': <scipy.stats._distn_infrastructure.rv_frozen object at 0x00000
1E7BAC348E0>})

pd.DataFrame(rand_search.best_estimator_.coef_,columns = X.columns,index = klase)

                z_x     red_x   green_x      blue  intensity       z_y  \
Tlo       -8.219135  0.332584  0.047891 -0.562193   0.264959  0.124363
Zgrade     1.189708  0.507808 -3.329219  3.374173   1.204456  0.032626
Vegetacija  0.220010  0.567645  0.643357 -1.655696  -4.001631 -0.072210


              red_y   green_y
Tlo       -0.095649  0.246723
Zgrade     0.938549 -0.755237
Vegetacija -1.099130  0.841048

data_test = pd.read_csv(r"C:\Users\ilija\Fax\Diplomski\Seminarski Septembar\Novi podaci\TriKlase_novi")
```

data_test

```
                x          y       z    red    green     blue  intensity  \
0       409166.67  5010719.89   77.92  39424.0  38656.0  35584.0      152.0
1       409166.35  5010719.93   77.92  38656.0  37888.0  34048.0      140.0
2       409166.37  5010719.19   77.83  48896.0  48896.0  48640.0      168.0
3       409165.37  5010718.76   77.82  48384.0  48640.0  48384.0      135.0
4       409165.04  5010718.80   77.87  48384.0  48640.0  48128.0       96.0
...           ...         ...    ...      ...      ...      ...        ...
6626229  409282.67  5010068.12   82.91  21504.0  26880.0  23040.0       13.0
6626230  409282.78  5010067.45   80.74  17152.0  24064.0  21760.0        8.0
6626231  409282.76  5010067.80   80.61  17664.0  23040.0  20480.0        9.0
6626232  409282.75  5010068.11   80.31  19456.0  24832.0  22272.0       11.0
6626233  409282.44  5010068.10   80.41  22016.0  27648.0  22272.0       25.0

         klase
0            0
1            0
2            0
3            0
4            0
...        ...
6626229      2
6626230      2
6626231      2
6626232      2
6626233      2

[6626234 rows x 8 columns]
```

epsilon_test, min_samples = kurvature(data_test)

[1.36832014 1.36707717 1.23032516 1.37237021 1.36707717 1.36835668]
Uzeta vrednost za epsilon je: 1.3683201379911283



```
epsilon_test = 0.56
min_samples = 6

X_test,y_test = preprocess_data(data_test,3.5,epsilon_test,min_samples)

X_test_scaled = transformer.transform(X_test)

vizualizacija(X_test,y_test,data_test)
```

8.4.1 Приказ резултата

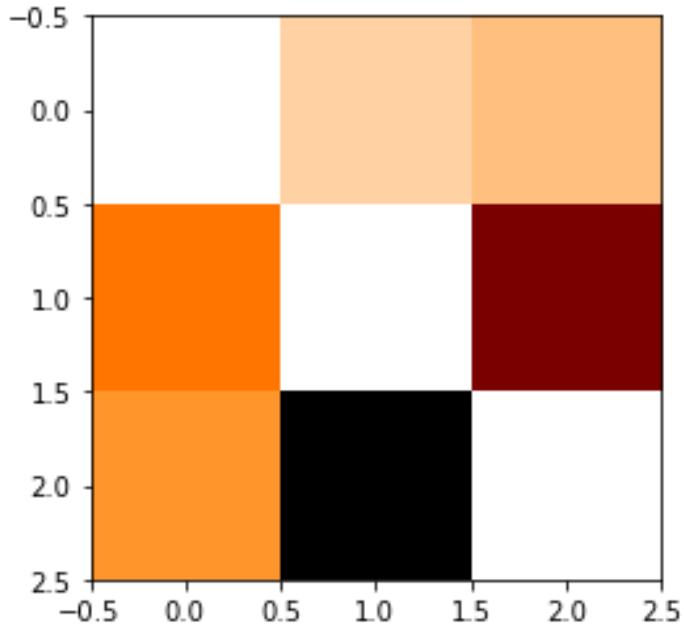
```
izvestaj(rand_search,X_scaled,X_test_scaled,y,y_test,klase = klase)
```

Tacnost na trening podacima je:
0.9293340683540463

Matrica konfuzije za trening podatke:

	Tlo	Zgrade	Vegetacija
Tlo	2141023	21231	29202
Zgrade	61310	2378862	154104
Vegetacija	47068	226887	2579100

Preciznost nad trening podacima: 0.9303373004908345
Odziv nad trening podacima je: 0.9326435194814401
F1 vrednost nad trening podacima je: 0.9313439510989056

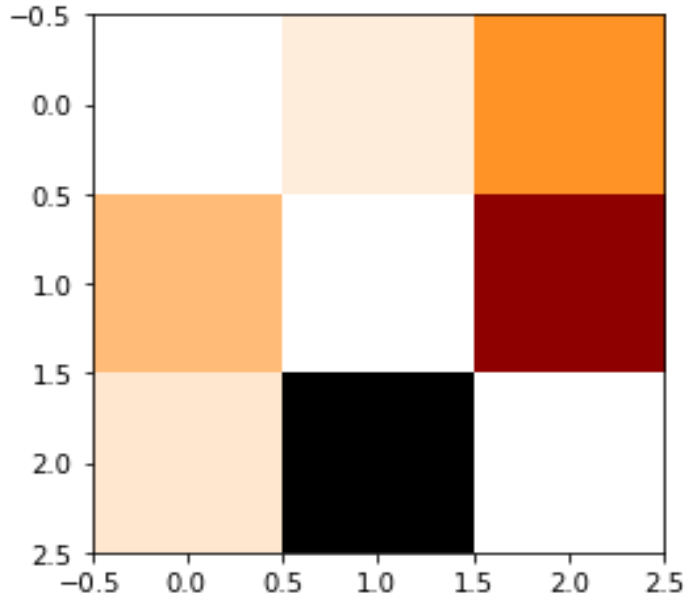


Tacnost na test podacima je:
0.9418418463183084

Matrica konfuzije za test podatke:

	Tlo	Zgrade	Vegetacija
Tlo	1792030	7075	39389

```
Zgrade       24999  2051879     115544
Vegetacija    9202   184837    2326938

Preciznost nad testing podacima: 0.9444387409826387
Odziv nad testing podacima je: 0.944551129250785
F1 vrednost nad testing podacima je: 0.9444309864006785
```




```
vizualizacija(X_test,rand_search.predict(X_test_scaled),data_test)
```

8.5 Мултикласна класификација применом алгоритма неуронске мреже

```
model = Sequential([
    Dense(13,activation='relu',input_shape = [8],kernel_regularizer = l2(0.001)),
    Dense(broj_klasa,activation = 'softmax')
])

model.compile(loss='sparse_categorical_crossentropy',optimizer = 'adam',metrics = ['accuracy'])
model.fit(X_scaled,y,validation_data = (X_valid_scaled,y_valid), epochs = 100, callbacks = Earl
yStopping(patience= 3,restore_best_weights=True))

Epoch 1/100
238713/238713 [==============================] - 203s 848us/step - loss: 0.1897 - ac
curacy: 0.9385 - val_loss: 0.3463 - val_accuracy: 0.8757
Epoch 2/100
238713/238713 [==============================] - 236s 988us/step - loss: 0.1632 - ac
curacy: 0.9454 - val_loss: 0.3316 - val_accuracy: 0.8788
Epoch 3/100
238713/238713 [==============================] - 318s 1ms/step - loss: 0.1620 - accu
racy: 0.9454 - val_loss: 0.3330 - val_accuracy: 0.8837
Epoch 4/100
238713/238713 [==============================] - 270s 1ms/step - loss: 0.1612 - accu
racy: 0.9454 - val_loss: 0.3840 - val_accuracy: 0.8700
Epoch 5/100
 54794/238713 [=====>.....................] - ETA: 2:10 - loss: 0.1604 - accuracy: 0.9460
```

izvestaj(model,X_scaled,X_test_scaled,y,y_test,klase = klase)

Tacnost na trening podacima je:
0.9463677937347906

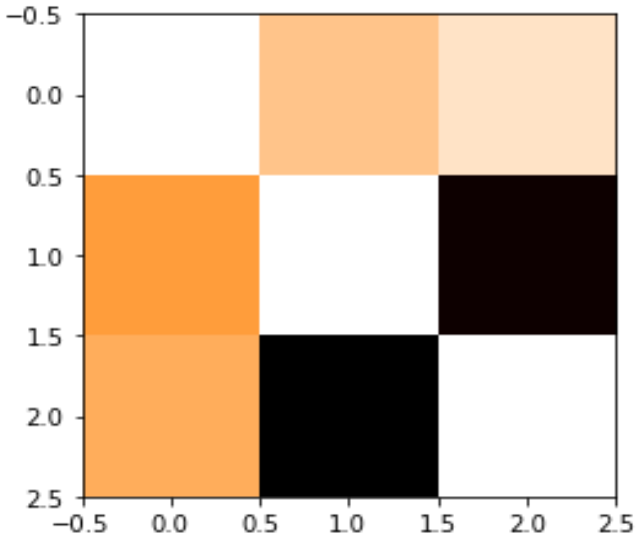
Matrica konfuzije za trening podatke:

	Tlo	Zgrade	Vegetacija
Tlo	2163626	18780	9050
Zgrade	31647	2404153	158476
Vegetacija	26939	164793	2661323

Preciznost nad trening podacima: 0.9478252452088792

Odziv nad trening podacima je: 0.9489375863082516

F1 vrednost nad trening podacima je: 0.9483593844273982

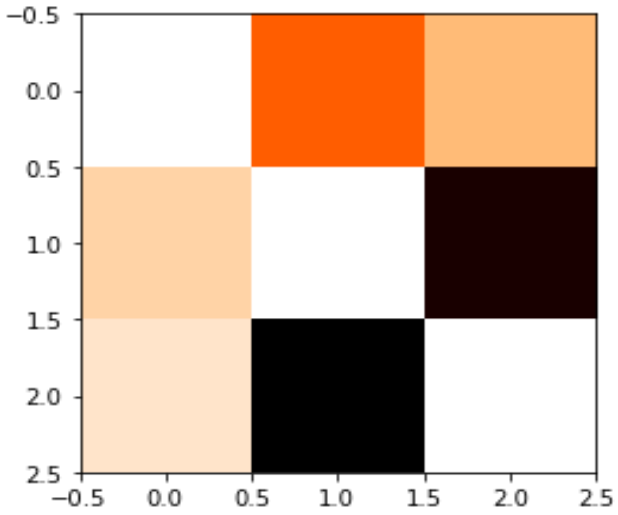


Tacnost na test podacima je:
0.9436816199531952

Matrica konfuzije za test podatke:

	Tlo	Zgrade	Vegetacija
Tlo	1772245	46811	19438
Zgrade	12957	2043347	136118
Vegetacija	7506	146162	2367309

Preciznost nad testing podacima: 0.9468788432611054
Odziv nad testing podacima je: 0.945004768400945
F1 vrednost nad testing podacima je: 0.9458597999645822

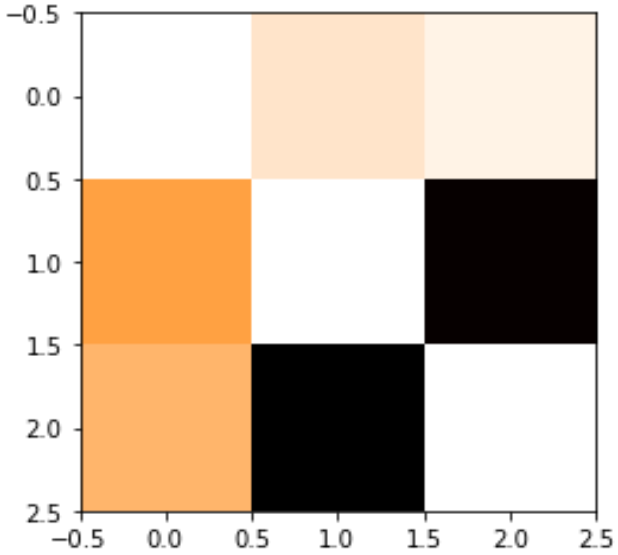



```
random = RandomForestClassifier(min_samples_split=0.001, n_estimators=50,
                 random_state=42)
random.fit(X_scaled,y)
```

```
RandomForestClassifier(min_samples_split=0.001, n_estimators=50,
                 random_state=42)
```

```
izvestaj(random,X_scaled,X_test_scaled,y,y_test,klase = klase)
```

```
Tacnost na trening podacima je:
 0.9565565841801846


Matrica konfuzije za trening podatke:
                Tlo   Zgrade  Vegetacija
Tlo         2180358     7375        3723
Zgrade        25679  2432638      135959
Vegetacija    20513   138606     2693936

Preciznost nad trening podacima: 0.9577824467671187
Odziv nad trening podacima je: 0.9589529040699715
F1 vrednost nad trening podacima je: 0.9583403525064198
```

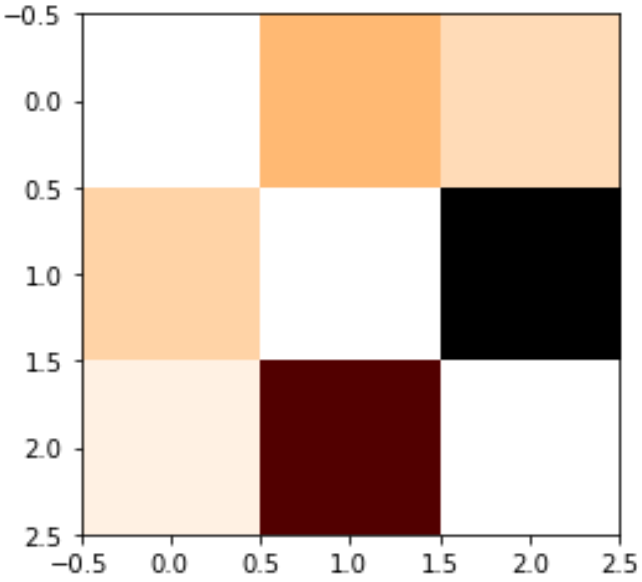


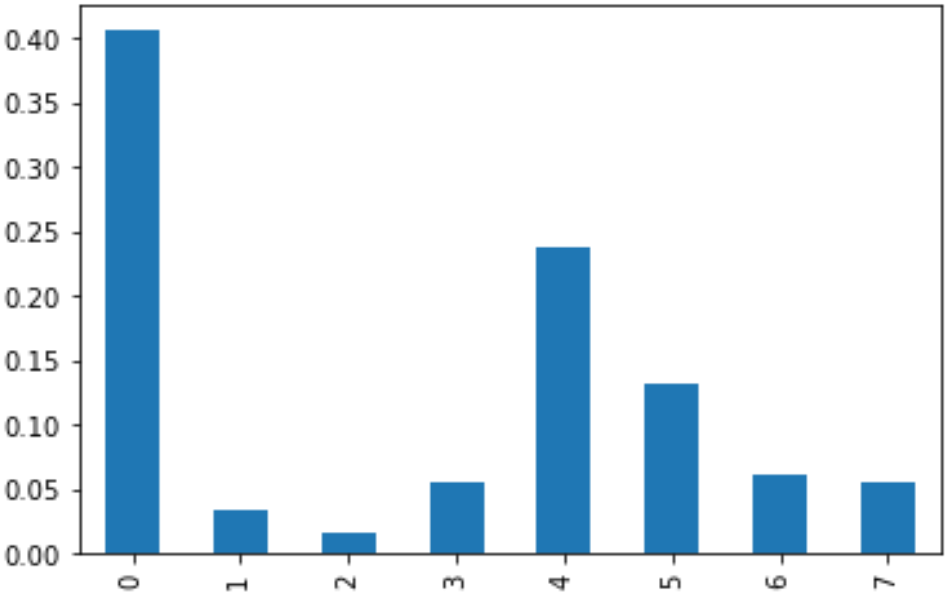
Tacnost na test podacima je:
0.9440390738981849

Matrica konfuzije za test podatke:

	Tlo	Zgrade	Vegetacija
Tlo	1801497	24380	12617
Zgrade	15474	2003373	173575
Vegetacija	4891	135713	2380373

Preciznost nad testing podacima: 0.9474260916046657
Odziv nad testing podacima je: 0.9459581560458964
F1 vrednost nad testing podacima je: 0.9466467519385472





joblib.dump(random,'random_forest_3klase6oje.h5')

```
['random_forest_3klase.h5']

import numpy as np
import laspy as lp
import pandas as pd
import matplotlib.pyplot as plt
import open3d as o3d

Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.

las_podaci = lp.read(r"C:\Users\ilija\Fax\Diplomski\Seminarski Septembar\Novi oblak tacaka\klasifikovani_podaci_novo.las")

# Funkcija koja pretvara ucitane podatke iz las fajlsa u odgovarajuci DataFrame
def las2df(las):
    x_dim = las.X
    x_scale = las.header.scales[0]
    x_offset = las.header.offset[0]
    x = ((x_dim*x_scale)+x_offset).reshape(-1,1)
    y_dim = las.Y
    y_scale = las.header.scales[1]
    y_offset = las.header.offset[1]
    y = ((y_dim*y_scale)+y_offset).reshape(-1,1)
    z_dim = las.Z
    z_scale = las.header.scales[1]
    z_offset = las.header.offset[1]
    z = ((z_dim*z_scale)+z_offset).reshape(-1,1)
    intensity = las.intensity.reshape(-1,1)
    klase = np.array(las.raw_classification,dtype = np.int32).reshape(-1,1)
    try:
        red = las.red.reshape(-1,1)
        green = las.green.reshape(-1,1)
        blue = las.blue.reshape(-1,1)
        xyzrgb = np.hstack([x,y,z,red,green,blue,intensity,klase])
        return pd.DataFrame(xyzrgb,columns = ['x','y','z','red','green','blue','intensity','klase'])
    except:
        xyz = np.hstack([x,y,z,intensity,klase])
        return pd.DataFrame(xyz,columns = ['x','y','z','intensity','klasa'])

#Funkcija za vizualizaciju

def viz(df,color = False):
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(df[['x','y','z']].values)
    if color is True:
        pcd.colors = o3d.utility.Vector3dVector(df[['red','green','blue']].values)

    o3d.visualization.draw_geometries([pcd])

'''Funkcije koje dele oblaka tacaka na dva dela na osnovu njihovih koordinata'''
def donji(df):
    return df.loc[(df.x <409903.75)&(df.y < 5011239.46)&(df.y >5010298.51)].copy()
```

```
def gornji(df):
    return df.loc[(df.y>5011368.56)&(df.y<5011859.53)].copy()

from sklearn.utils import shuffle

def podaci_zgrade(podaci):

    ne_zgrade = podaci.loc[~podaci.klase.isin([75,77,79,80])].copy()
    zgrade = podaci.loc[podaci.klase.isin([75,77,79,80])].copy()
    ne_zgrade.klase = 1
    zgrade.klase = 0
    return pd.concat([zgrade,ne_zgrade])

def podaci_4klase(podaci):
    tlo = podaci.loc[podaci.klase == 2].copy()
    sikara = podaci.loc[podaci.klase ==23].copy()
    zgrade = podaci.loc[podaci.klase.isin([75,77,79,80])].copy()
    vegetacija = podaci.loc[podaci.klase ==82].copy()
    tlo.klase = 0
    sikara.klase = 1
    zgrade.klase = 2
    vegetacija.klase = 3
    return pd.concat([tlo,sikara,zgrade,vegetacija])

def podaci_3klase(podaci):
    tlo = podaci.loc[podaci.klase == 2].copy()
    zgrade = podaci.loc[podaci.klase.isin([75,77,79,80])].copy()
    vegetacija = podaci.loc[podaci.klase ==82].copy()
    tlo.klase = 0
    zgrade.klase = 1
    vegetacija.klase =2
    return pd.concat([tlo,zgrade,vegetacija])

def podaci_putParking(podaci):
    tlo = podaci.loc[podaci.klase.isin([2,23])].copy()
    zgrade = podaci.loc[podaci.klase.isin([75,77,79,80])].copy()
    vegetacija = podaci.loc[podaci.klase.isin([82,87])].copy()
    put = podaci.loc[podaci.klase.isin([41,50])].copy()
    tlo.klase = 0
    zgrade.klase = 1
    vegetacija.klase =2
    put.klase =3
    return pd.concat([tlo,zgrade,vegetacija,put])

def podaci_Rasveta(podaci):
    tlo = podaci.loc[podaci.klase.isin([2,23])].copy()
    zgrade = podaci.loc[podaci.klase.isin([75,77,79,80])].copy()
    vegetacija = podaci.loc[podaci.klase.isin([82,87])].copy()
    put = podaci.loc[podaci.klase.isin([41,50])].copy()
    rasveta = podaci.loc[podaci.klase.isin([28,38])].copy()
```

```
    tlo.klase = 0
    zgrade.klase = 1
    vegetacija.klase =2
    put.klase =3
    rasveta.klase = 4
    return pd.concat([tlo,zgrade,vegetacija,put,rasveta])

podaci = las2df(las_podaci)
podaci
```

```
             x         y     z     red    green     blue  intensity  \
0       409250.07  5010500.16  77.26  49920.0  50176.0  50688.0       93.0
1       409163.81  5010728.73  77.67  40960.0  41472.0  42752.0       51.0
2       409163.96  5010728.62  77.68  40960.0  41728.0  42496.0       75.0
3       409163.65  5010728.67  77.68  40704.0  41216.0  42240.0       86.0
4       409163.93  5010728.40  77.68  40960.0  41728.0  43264.0       64.0
...           ...         ...    ...      ...      ...      ...        ...
11738479  409281.67  5010073.73  76.47  41472.0  41472.0  42752.0       49.0
11738480  409281.00  5010072.27  76.45  40192.0  39680.0  40704.0      100.0
11738481  409280.98  5010072.66  76.45  38656.0  38656.0  39424.0       74.0
11738482  409280.95  5010073.08  76.46  35328.0  36096.0  38144.0       65.0
11738483  409280.93  5010073.47  76.46  36864.0  38144.0  40192.0       51.0

         klase
0         41.0
1         41.0
2         41.0
3         41.0
4         41.0
...        ...
11738479   41.0
11738480    2.0
11738481   41.0
11738482   23.0
11738483   41.0

[11738484 rows x 8 columns]
```

```
zgrade = podaci_zgrade(podaci)
zgrade
```

```
             x         y     z     red    green     blue  intensity  \
26261     409137.06  5010684.11  82.33  13568.0  19200.0  23552.0      260.0
26696     409137.20  5010683.70  82.34  13568.0  19968.0  23552.0      185.0
26698     409136.87  5010683.74  82.32  14848.0  20736.0  24576.0      947.0
26699     409136.54  5010683.79  82.32  16384.0  21760.0  26112.0      306.0
26701     409136.24  5010683.83  82.28  15360.0  20480.0  24576.0        8.0
...           ...         ...    ...      ...      ...      ...        ...
11738479  409281.67  5010073.73  76.47  41472.0  41472.0  42752.0       49.0
11738480  409281.00  5010072.27  76.45  40192.0  39680.0  40704.0      100.0
11738481  409280.98  5010072.66  76.45  38656.0  38656.0  39424.0       74.0
11738482  409280.95  5010073.08  76.46  35328.0  36096.0  38144.0       65.0
11738483  409280.93  5010073.47  76.46  36864.0  38144.0  40192.0       51.0
```

```
         klase
26261        0
26696        0
26698        0
26699        0
26701        0
...        ...
11738479     1
11738480     1
11738481     1
11738482     1
11738483     1

[11738484 rows x 8 columns]
```

```
zgrade.to_csv(r'C:\Users\ilija\Fax\Diplomski\Seminarski Septembar\Novi podaci\zgrade_novi.csv',index = False)
```

```
podaci_4klase
```

```
             x          y     z      red    green     blue  intensity  \
21229    410214.27  5011331.73  80.10  31488.0  27136.0  22528.0      103.0
21381    410215.93  5011331.28  80.09  35072.0  28672.0  25600.0      114.0
21382    410215.55  5011331.34  80.13  34560.0  29184.0  27136.0       94.0
21384    410214.81  5011331.44  80.10  34304.0  28928.0  27136.0      100.0
22235    410215.35  5011329.74  80.02  19456.0  22784.0  21504.0       95.0
...            ...        ...   ...      ...      ...      ...        ...
46102072  409500.06  5010975.06  84.19  34048.0  34816.0  32256.0        9.0
46102073  409500.07  5010974.83  83.12  34048.0  36608.0  31488.0       19.0
46102074  409500.08  5010974.63  82.15  34304.0  36608.0  29440.0       11.0
46102076  409500.07  5010975.42  84.24  30976.0  30976.0  30720.0       18.0
46102078  409500.09  5010975.43  82.65  30976.0  30976.0  30720.0       29.0

         klase
21229        0
21381        0
21382        0
21384        0
22235        0
...        ...
46102072     3
46102073     3
46102074     3
46102076     3
46102078     3

[32291526 rows x 8 columns]
```

```
podaci_3klase = podaci_3klase(podaci)
```

```
klase3_donji = donji(podaci_3klase)
```