# Algoritmi strojnog učenja primijenjeni na zadanom skupu podataka

**Gelo, Anja**




SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# ZAVRŠNI RAD

**Anja Gelo**

Zagreb, 2023. godina.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# Algoritmi strojnog učenja primijenjeni na zadanom skupu podataka

Mentor:

Izv. prof. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Anja Gelo

Zagreb, 2023. godina.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru izv. prof. dr. sc. Tomislavu Stipančiću na pruženoj pomoći i podršci tijekom izrade ovog rada. Zahvaljujem se svojoj obitelji i prijateljima na podršci i motivaciji tijekom cijelog studija.

Anja Gelo

SVEUČILIŠTE U ZAGREBU
**FAKULTET STROJARSTVA I BRODOGRADNJE**
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

| Sveučilište u Zagrebu<br>Fakultet strojarstva i brodogradnje | |
|---|---|
| Datum | Prilog |
| Klasa: **602 – 04 / 23 – 6 / 1** | |
| Ur.broj: **15 - 1703 - 23 -** | |

# ZAVRŠNI ZADATAK

Student: **Anja Gelo** JMBAG: **0035215842**

Naslov rada na hrvatskom jeziku: **Algoritmi strojnog učenja primijenjeni na zadanom skupu podataka**

Naslov rada na engleskom jeziku: **Machine learning algorithms applied to a given data set**

Opis zadatka:

U svijetu podataka i velike količine informacija algoritmi umjetne inteligencije postaju sve značajniji. U ovisnosti o vrsti i strukturi podataka te metodi strojnog učenja algoritmi se koriste prilikom predviđanja, odlučivanja, klasifikacije, prepoznavanja i slično. Prije primjene algoritma na zadanom skupu podatke je potrebno obraditi i prilagoditi.

Na zadanom skupu podataka potrebno je:

- izvršiti funkcije prilagodbe podataka uključujući funkcije čitanja i čišćenja podataka, funkcije vizualizacije podataka, te funkcije skaliranja podataka,
- opisati i primijeniti tri različita algoritma strojnog učenja (stroj potpornih vektora – SVM, drvo odluke te klasifikator k-najbližih susjeda) koji će podatke podijeliti u klase,
- usporediti rezultate izvođenja algoritama s gledišta preciznosti te odrediti koji je najuspješniji.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:

30. 11. 2022.

Zadatak zadao:

Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:

**1. rok:** 20. 2. 2023.
**2. rok (izvanredni):** 10. 7. 2023.
**3. rok:** 18. 9. 2023.

Predviđeni datumi obrane:

**1. rok:** 27. 2. – 3. 3. 2023.
**2. rok (izvanredni):** 14. 7. 2023.
**3. rok:** 25. 9. – 29. 9. 2023.

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

# SADRŽAJ

# POPIS SLIKA

# POPIS TABLICA

# POPIS OZNAKA

| Oznaka | Opis |
|---|---|
| $K$ | Broj klasa |
| $h(x)$ | Hipoteza modela strojnog učenja |
| $x$ | Primjer u ulaznom prostoru |
| $n$ | Broj značajki |
| $\mathbf{w}$ | Vektor težine |
| $w_0$ | Težina |
| $\theta$ | Parametar |
| $y_i$ | Klasa *i* kojoj pripada jedan primjer |
| $N$ | Broj potpornih vektora |
| $\alpha_i$ | Lagrangeov multiplikator |
| d | Udaljenost dvije točke |
| $x_j^{[a]}$ | Koordinata x primjera a |
| $x_j^{[b]}$ | Koordinata x primjera b |
| $k$ | Broj susjeda |
| $I_G$ | Gini indeks |
| $t$ | Trenutni čvor |
| $p_i$ | Vjerojatnost klase u čvoru *t* |
| $m$ | Broj klasa u modelu |
| $\bar{x}$ | Aritmetička sredina |
| $u$ | Srednja vrijednost primjera |
| $s$ | Standardna devijacija primjera |
| $z$ | Skalirani primjer |
| $x\_train$ | Primjeri za učenje |
| $y\_train$ | Klase primjera za učenje |
| $x\_test$ | Primjeri za testiranje |
| $y\_test$ | Klase primjera za testiranje |
| $y\_pred$ | Predviđena klasa za $x\_test$ |

# SAŽETAK

U svijetu podataka i velike količine informacija algoritmi umjetne inteligencije postaju sve značajniji. U ovisnosti o vrsti i strukturi podataka te metodi strojnog učenja algoritmi se koriste prilikom predviđanja, odlučivanja, klasifikacije, prepoznavanja i slično. U ovom radu opisane su i objašnjene tri klasifikacijske neparametarske metode strojnog učenja koje pripadaju nadgledanom učenju – stroj potpornih vektora, algoritam k-najbližih susjeda te stablo odluke. Sva tri algoritma su primijenjena na zadanom skupu podataka u programskom jeziku Python te *open-source* web aplikaciji Jupyter Notebook. Prije same primjene algoritama na zadani skup podataka, na podatke je bilo potrebno primijeniti funkcije obrade podataka – funkcije čitanja, funkcije čišćenja, funkcije skaliranja te funkcije vizualizacije. Algoritmi su uspoređeni po validacijskom kriteriju preciznosti.

Ključne riječi: strojno učenje, algoritam, SVM, stroj potpornih vektora, stablo odluke, klasifikator k-najbližih susjeda, umjetna inteligencija, klasificiranje podataka

# SUMMARY

In the world of data and large amounts of information, artificial intelligence algorithms are becoming increasingly important. Depending on the type and structure of the data and the machine learning method, algorithms are used for prediction, decision-making, classification, recognition, and the like. In this paper, three classification non-parametric machine learning methods that belong to supervised learning are described and explained - the support vector machine, the k-nearest neighbors algorithm and the decision tree. All three algorithms were applied to the given data set in the Python programming language and the open-source web application Jupyter Notebook. Before applying the algorithms to the given data set, it was necessary to apply data processing functions to the data - reading functions, cleaning functions, scaling functions and visualization functions. Algorithms were compared according to the validation criterion of precision.

Key words: machine learning, algorithm, SVM, machine supported vectors, decision tree algorithm, K-NN, classified data, artificial intelligence

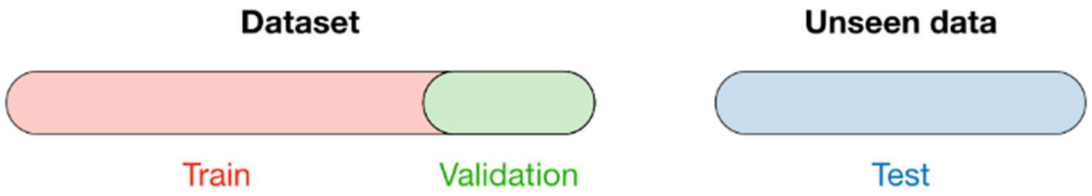
# 1. UVOD

Predmet i glavna okosnica ovog rada je strojno učenje (eng. *Machine Learning*, ML), odnosno programiranje računala tako da primjenom umjetne inteligencije strojevi mogu učiti iz iskustva i usavršavati svoje izvedbe bez pomoći ljudi. Današnja računala, vođena umjetnom inteligencijom, mogu analizirati, planirati, predviđati i donositi odluke i sve ostale uloge koje smo oduvijek smatrali da su rezervirana za ljudska bića [1]. Da bi računalo bilo u mogućnosti izvesti ovakve operacije potreban mu je algoritam, postupak ili način rješavanja nekog problema ili zadatka [2], tj. konačan niz koraka koji vodi prema rješenju nekog problema te ulazni i/ili izlazni podaci. Kada se neki postupak može nazvati algoritmom? Iz knjige *Rješavanje problema programiranjem u Pythonu, 2014.* autori daju jasne odgovore: 1. Uz svaki algoritam moraju jasno biti definirana početna stanja objekata nad kojima se obavljaju operacije i 2. Algoritam mora imati konačan broj koraka koji utvrđuju slijed operacija koje treba obaviti nad objektima kako bi se dobio rezultat [2]. U radu će se detaljnije opisati tri algoritma strojnog učenja: algoritam stroj potpornih vektora (eng. *support vector machine*, *SVM*), stablo odluke (eng. *decision tree algorithm*) te algoritam K-najbližih susjeda (eng. *K-NN algorithm*), primijeniti 3 algoritma na zadanom skupu podataka u Python-u te će se usporediti i analizirati uspješnost predikcije sva tri algoritma. Da bi razumijevanje algoritama bilo jasnije, potrebno je pobliže objasniti što je to strojno učenje, osnovne pojmove i metode učenja.

# 2. ŠTO JE STROJNO UČENJE?

Tom Mitchell u svojoj definiciji „Računalni program uči iz iskustva E s obzirom na zadatak T i neku mjeru izvedbe P, ako se njegova izvedba na T, mjerena pomoću P, poboljšava s iskustvom E.“ (1997) govori da računalni program uči ako ne ponavlja svoje greške i time pospješuje svoju izvedbu, što je potvrdio i Yifei Chai izrekom: „Dopustite UI da radi greške, jer na pogreškama će učiti“ (2019) [3] i objašnjava da su „algoritmi nešto što su napravili ljudi da stalno radi zadatak iznova, a UI ima mogućnost iskoristiti informacije kako bi nešto zaključila i za to treba vremena“.

Strojno učenje grana je umjetne inteligencije koja se bavi oblikovanjem algoritama koji svoju učinkovitost poboljšavaju na temelju empirijskih podataka. Strojno učenje jedno je od danas najaktivnijih i najuzbudljivijih područja računarske znanosti, ponajviše zbog brojnih mogućnosti primjene koje se protežu od raspoznavanja uzoraka i dubinske analize podataka do robotike, računalnog vida, bioinformatike i računalne lingvistike. [4] Primjene strojnog učenja su danas običnom čovjeku i neprimjetne, koriste se u svakodnevici i infiltrirale su se u osnovni dio čovjekovog života. Jednostavni algoritam strojnog učenja koji se zove „naive Bayes“ može odvojiti neželjenu e-mail poštu od željene [5]. Kako to uspijeva? Prilikom izvršavanja modela tj. algoritma koristimo se s dva seta podataka prikazano na [Slika 1].

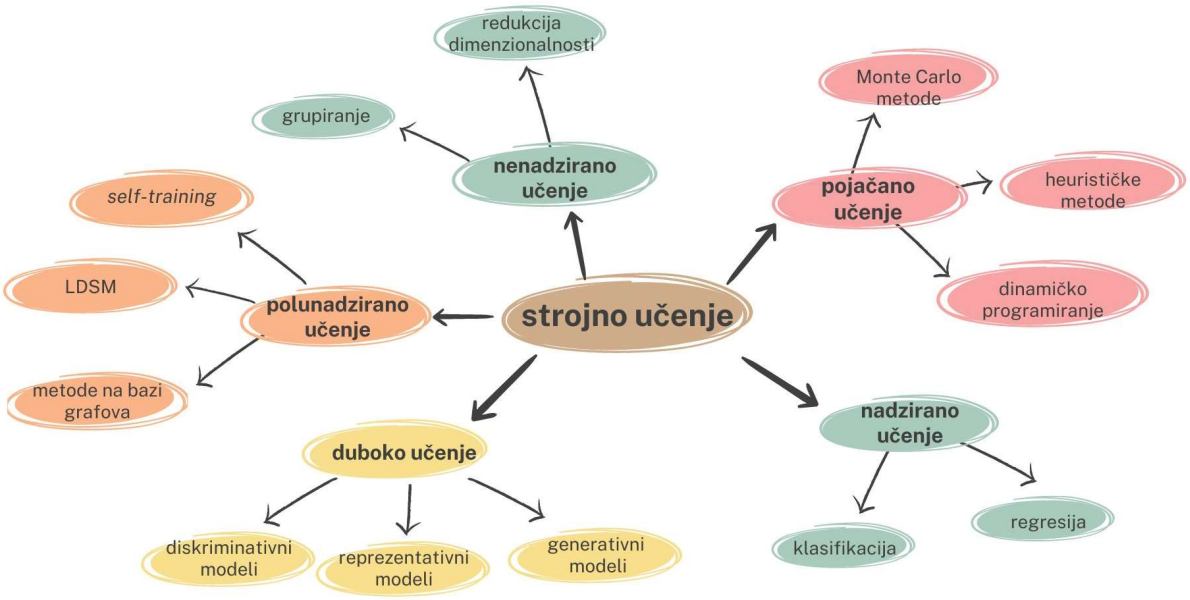



**Slika 1 Izvedba algoritma strojnog učenja**

Na prvom se setu podataka model trenira. U slučaju izdvajanja neželjene pošte, dajemo modelu odvojenu neželjenu i željenu poštu te on detektira i prolazi kroz riječi koje se nalaze u neželjenoj pošti, stvara veze između njih i uspoređuje ih s riječima iz željene pošte te uči koji je e-mail neželjeni, a koji željeni. Obično je za treniranje odvojeno 80% prvog seta podataka. Zatim slijedi validacija seta podataka u kojoj se koristi ostalih 20% prvog seta podataka. Validacija je postupak utvrđivanja valjanosti modela. Na engleskom se još naziva i *hold-out set* ili

*development set.* Nakon što je model provjeren, testira se na novom setu podataka koji do sada nije vidio i daje predviđanja koji od e-mailova pripada neželjenoj pošti, a koji željenoj na temelju naučenog. Cilj svakog algoritma strojnog učenja je dobra sposobnost predikcije ili klasifikacije na neviđenim ulaznim podacima.

Postoji pet pristupa strojnom učenju: nadzirano učenje (klasifikacija i regresija), nenadzirano učenje (grupiranje i redukcija dimenzionalnosti), polunadzirano učenje (*self-training*, LDSM i metode na bazi grafova), pojačano učenje (dinamičko programiranje, Monte Carlo metode te Heurističke metode) i duboko učenje (diskriminativni modeli, reprezentativni modeli, generativni modeli) [Slika 2.].




**Slika 2 Podjela strojnog učenja**

Modele nadziranog učenja koristimo kada u podacima za treniranje imamo ulazne vektore zajedno s njihovim ciljnim vektorima, bilo da su to klase ili jedna ili više kontinuiranih varijabli. U drugim problemima prepoznavanja primjera, podaci za treniranje sastoje se od skupa ulaznih vektora x bez ikakvih odgovarajućih ciljnih vrijednosti. Prema tim podacima može se zaključiti da se ovdje radi o nenadziranom učenju. Cilj u takvim modelima učenja može biti otkrivanje grupa sličnih primjera unutar podataka (eng. *clustering*) ili određivanje distribucije podataka

unutar ulaznog prostora, poznato kao procjena gustoće ili projiciranje podataka iz visoko-dimenzionalnog prostora na dvije ili tri dimenzije u svrhu vizualizacije [7].

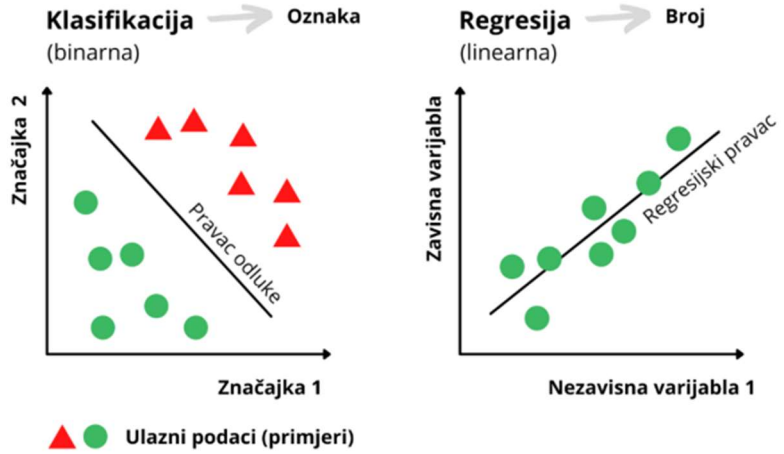
## 2.1. Osnovni pojmovi u strojnom učenju

Ulazni podatak je točka s *n* značajki u n-dimenzionalnom prostoru, što znači da je ulazni prostor modela direktno definiran brojem **značajki** (eng. *feature*) koje promatramo za svaki pojedini ulazni podatak, a taj broj definira korisnik. Ulazni podatak se još naziva i **primjer** (eng. *example, instance*), a označuje jednu točku, jedan redak u bazi podataka za koji želimo da model napravi neko predviđanje. **Značajka** (eng. feature) je neka karakteristika primjera koja je bitna i o kojoj ovisi klasifikacija primjera (kod klasifikacije) ili njegova ciljna vrijednost (kod regresije). Ulazni primjeri mogu imati različit broj značajki koje se penju do reda tisućica, ovisno o kompleksnosti problema. Svaki model nastoji „protumačiti" ulazne podatke tako da za njih „osmisli" funkciju ili **hipotezu** koja najbolje oslikava zakonitost njihovog ponašanja.

## 2.2. Nadzirano učenje

Kod nadziranog učenja, svaki primjer ima svoju **oznaku** (eng. *label*). Drugim riječima, svaki primjer ima svoje ime, dok kod nenadziranog učenja to nije takav slučaj. Oznaka će u slučaju klasifikacije biti ime klase, a kod regresije ciljna brojčana vrijednost.

Model nadziranog učenja je skup hipoteza tj. funkcija, indeksiran njihovim parametrima, koje primjere preslikavaju u oznake, a upravo se učenje modela svodi na pretraživanje i odabir najbolje hipoteze, drugim riječima optimizaciju parametara modela. Upravo je to i veza između strojnog učenja i optimizacije koji su inače vrlo tijesno povezani. Modeli strojnog učenja zapravo provode optimizacijski postupak prilikom odabira hipoteze kojim minimiziraju

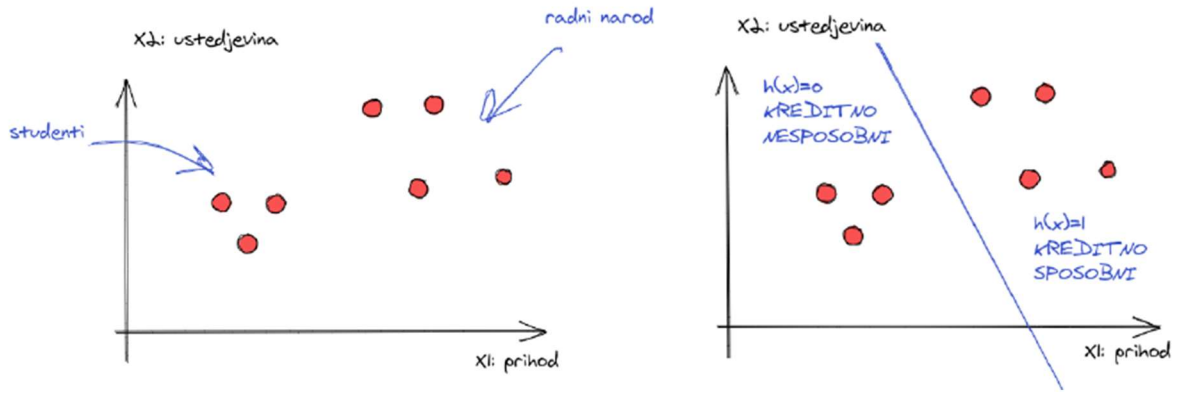



**Slika 3 Razlika regresije i klasifikacije nadziranog učenja**

empirijsku pogrešku, tj. pogrešku izvedbe svake hipoteze te biraju onu hipotezu koja ima najmanju pogrešku. Nadzirano učenje razlikuje klasifikaciju i regresiju [Slika 3.]. Osnovna razlika između klasifikacije i regresije je u izlaznoj vrijednosti algoritma, tj. modela. Izlazna vrijednost klasifikacije je klasa (oznaka), a izlazna vrijednost regresije je brojčana vrijednost.

Ako pogledamo primjere [8] nadgledanog učenja bit će jasnije razlikovati klasifikaciju od regresije.

a) Primjer 1:




**Slika 4 Primjer binarne klasifikacije**

U primjeru binarne klasifikacije bavimo se problemom klasifikacije kreditno sposobnih klijenata banke. Pretpostavka je da banke rade na temelju dvije značajke: prihod i ušteđevina. Ako imamo dvije značajke znači da je ulazni prostor dvodimenzijski. Značajke su prikazane na osima x i y tj. „$x_1$: prihod“ i „$x_2$: ušteđevina“, a svaki klijent banke je jedan dvodimenzijski vektor u tom prostoru. U ovom slučaju pravac predstavlja jednu hipotezu u dvodimenzijskom sustavu. Ta hipoteza razdjeljuje prostor na dva poluprostora. Poluprostor određen s (1)

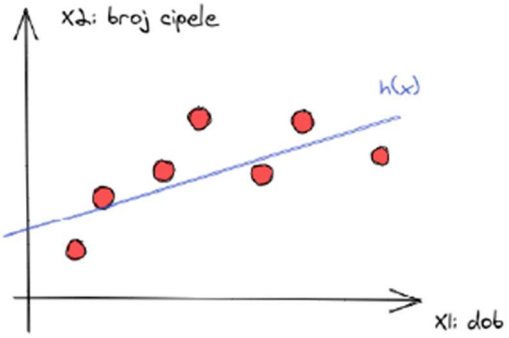
$$h(\mathrm{x}) = 0 \qquad \textbf{(1)}$$

odgovara kreditno nesposobnim klijentima (niski prihodi i niska ušteđevina) dok poluprostor određen s (2)

$$h(\mathrm{x}) = 1 \qquad \textbf{(2)}$$

odgovara kreditno sposobnim klijentima (visoki prihodi i visoka ušteđevina). Klasifikacija novih primjera (novih klijenata) će biti određena kriterijima značajki i s koje strane pravca će biti smješten novi primjer (klijent).

b) Primjer 2.




**Slika 5 Primjer regresije**

Kao primjer regresije odabran je problem predviđanja broja (veličine) cipele s obzirom na dob osobe. Primjer je regresijski jer želimo predvidjeti brojčanu vrijednost, a ne klasu. U ovom slučaju jedna, zavisna varijabla ($x_2$: broj cipele) ovisi o drugoj, nezavisnoj varijabli ($x_1$: dob). S obzirom na to da imamo samo jednu značajku (dob osobe), ulazni prostor je jednodimenzijski $n = 1$, a hipotezu onda možemo definirati pravcem (3)

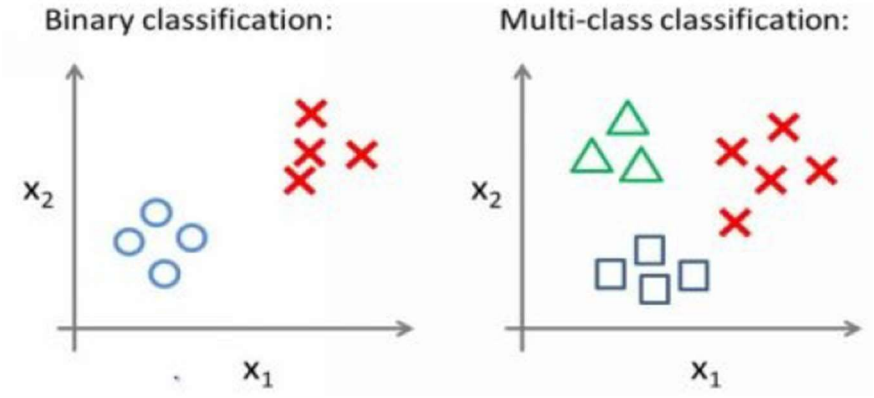
$$h(\mathrm{x}; \theta_0, \theta_1) = \theta_1 x + \theta_0 \quad \textbf{(3)}$$

jer očekujemo linearnu ovisnost između dobi i veličine cipela.

### *2.2.1. Klasifikacija*

Od ranog djetinjstva roditelji svoju djecu uče klasifikaciji. Prilikom spremanja svoje sobe ona to i primjenjuju. Na temelju naziva (npr. igračka) povezanog s materijalnom stvari (npr. lutka), djeca spremaju tu lutku u kutiju predviđenu za igračke. Pritom su djeca naučila da igračke mogu biti različite, ali sve imaju to jedno svojstvo zajedničko – predmeti koji se koriste za igranje. Takvi se klasifikacijski „problemi“ pojavljuju u stvarnom životu jako često iako toga nismo ni svjesni. Ono što čovjek radi je sada već automatizam, no računalo mora proći svaki taj korak. Pacijent dolazi u ambulantu s nekoliko simptoma koji mogu opisati i biti simptomi 3 medicinska stanja. Postavlja se pitanje koje stanje/bolest od ta tri ima pacijent? [9] Ono što će algoritam dati kao odgovor iz ova dva primjera je kvalitativne vrijednosti te se stoga klasifikacija primjera može definirati i kao predviđanje njegove kvalitativne vrijednosti s obzirom na to da tu vrijednost dodjeljuje kategoriji ili klasi [9]. Neki od najpopularnijih klasifikatora su modeli logističke regresije, analize linearne diskriminante, k-najbližih susjeda, stroj potpornih vektora te stablo odluke [10].

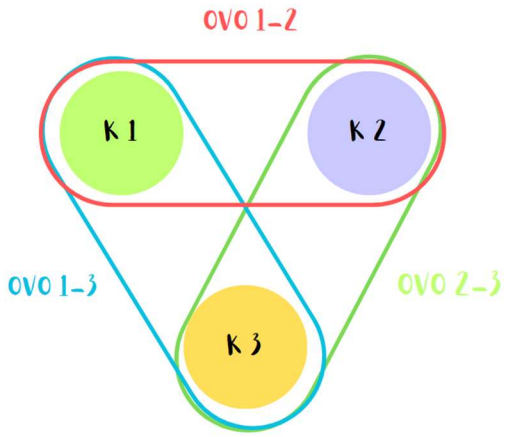
Klasifikacija kao takva, kao što je spomenuto u primjeru kreditno sposobnih klijenata banke u prethodnom poglavlju, može biti binarna (eng. *binary classification*), a može biti i višeklasna (eng. *multi-class classification*), što bi za primjer bio problem odlučivanja bolesti u pacijenta jer je potrebno analizirati 3 moguća stanja i odabrati jedan.



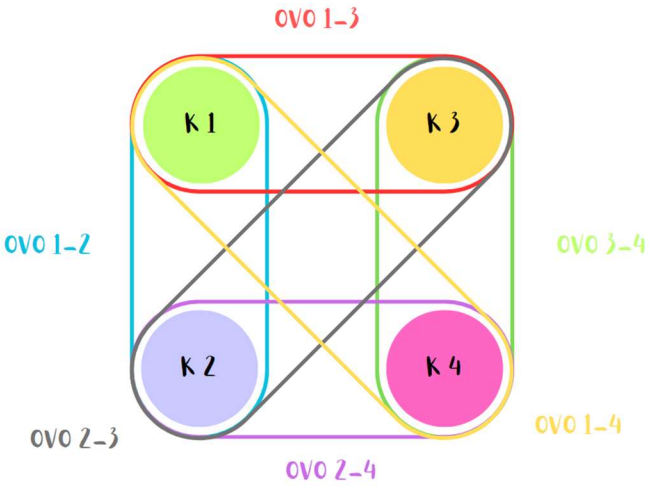

**Slika 6 Prikaz binarne i višeklasne klasifikacije**

Binarni klasifikatori se koriste kada želimo razlikovati bilo koje dvije klase i njihovi izlazi su, kao što je objašnjeno, binarni – 0 i 1. Postavlja se pitanje kakvi su izlazi kod višeklasne klasifikacije s obzirom na to da trebamo minimalno 3 izlaza koliko je i minimalno klasa u višeklasnoj klasifikaciji. Klasifikator slučajnih šuma i Bayes-ov klasifikator vrlo lako mogu riješiti taj problem, no glavna ideja je ne promijeniti model. Ako se ne mijenja model, onda se mora promijeniti problem. S obzirom na to da je binarna klasifikacija vrlo jednostavna i već poznata, najjednostavniji način je rastaviti višeklasni problem na skup binarnih klasifikacijskih problema te onda više puta primijeniti binarni klasifikator [8]. Postoje dvije opcije kako to primijeniti: shema jedan-naspram-jedan (eng. *one-versus-one*) i shema jedan-naspram-ostali (eng. *one-versus-rest*).

Shema jedan-naspram-jedan (eng. *one-vs-one*, OVO) rješava višeklasni problem tako da problem rastavi na $\binom{K}{2}$ nezavisnih binarnih klasifikacijskih problema, po jedan za svaki par klasa. Ako radimo s $K = 3$ klase, shema će postaviti 3 problema. Ako su u pitanju $K = 4$ klase, shema će problem rastaviti na 6 binarnih klasifikacijskih problema.

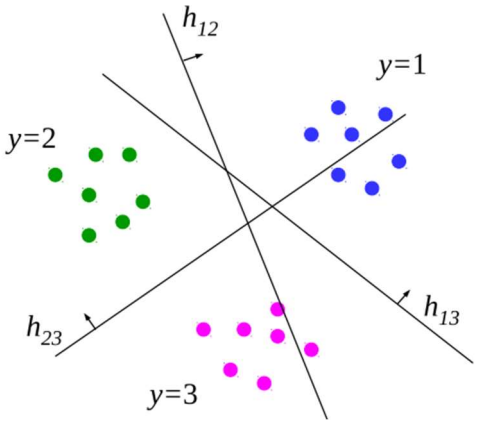



**Slika 8 Višeklasni problem s 3 klase rastavljen na 3 problema one-vs-one (OVO 1-3, OVO 1-2, OVO 2-3)**




**Slika 7 Višeklasni problem s 4 klase rastavljen na 6 binarnih klasifikacijskih problema one-vs-one (OVO 1-2, OVO 1-3, OVO 3-4, OVO 2-4, OVO 2-3, OVO 1-4)**

Na konkretnom primjeru za $K = 3$ klase, u dvodimenzijskome ulaznom prostoru to bi izgledalo ovako [Slika 9.]:




**Slika 9 Shema jedan-naspram-jedan, klase odvojene trima klasifikatorima, tj. hipotezama koje su označene kao $h_{12}$, $h_{23}$ i $h_{13}$**

Zadatak je treniranje modela $h_{ij}$ tako da nauči razdvojiti primjere iz klase $y = i$ od primjera iz klase $y = j$. Na [Slika 9.] strelice pokazuju u smjeru pozitivne orijentacije hiperravnine, tj. su smjeru u kojem $h_{ij}$ kao pozitivne primjere klasificira one iz klase $y = i$. Tako će hiperravnina $h_{12}$ pozitivno klasificirati one primjere koji pripadaju klasi $y = 1$, a negativno one koji se nalaze u klasi $y = 2$, dok u isto vrijeme potpuno zanemaruje primjere iz klase $y = 3$. Odluka o tome u koju klasu se svrstava koji primjer donosi se većinskim glasanjem koje glasi ovako (4)

$$h(\mathrm{x}) = \underset{i}{\operatorname{argmax}} \sum_{i \neq j} sgn\left(h_{ij}(\mathrm{x})\right) \tag{4}$$

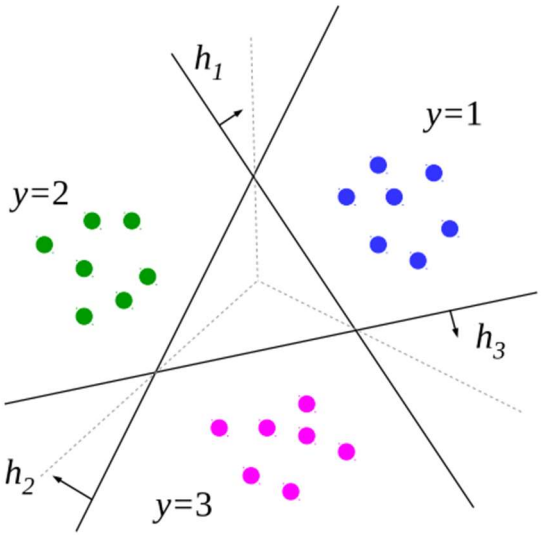
pri čemu (5)

$$h_{ij}(\mathrm{x}) = -h_{ji}(\mathrm{x}) \tag{5}$$

Druga shema jedan-naspram-ostali (eng. one-vs-rest, OVR) problem višeklasne klasifikacije rastavlja na $K$ nezavisnih binarnih klasifikatora, po jedan za svaku klasu. Svaki klasifikator $h_i$ je naučen da za određeni primjer prepozna radi li se o primjeru iz klase $y = i$ ili ne. Prethodni primjer s 3 klase će jako dobro prikazati razliku između ove dvije sheme. U shemi OVR će biti opet 3 binarna klasifikatora, no za razliku od sheme OVO, oni će raditi potpuno drugačiji zadatak. Višeklasni model u shemi OVR odluku donosi na temelju pouzdanosti pojedinačnih klasifikatora tako da primjer svrstava u onu klasu za koju je klasifikacija najpouzdanija [8], a definira se ovako (6)

$$h(\mathrm{x}) = \underset{j}{\operatorname{argmax}}\, h_j(\mathrm{x}) \tag{6}$$

a grafički je shema jedan-naspram-ostali prikazana na [Slika 10.].




**Slika 10 Shema jedan-naspram-ostali, klase odvojene trima klasifikatorima tj. hipotezama koje su označene kao $h_1, h_2$ te $h_3$**

Još jedna važna podjela klasifikacijskih modela je na parametarske i neparametarske metode. Parametarske metode uvijek imaju fiksan broj parametara prilikom učenja modela. Bez obzira koliko se podataka unese u model, on ne mijenja svoju odluku o tome koliko mu je potrebno da napravi predikciju. Na velikim skupovima podataka predikcija može biti znatno lošija od

očekivanog. S druge strane, neparametarski modeli su algoritmi koji ne stvaraju snažne pretpostavke o distribuciji podataka i slobodni su naučiti bilo koji funkcionalni oblik iz podataka za učenje. U Tablici 1. prikazana je osnovna usporedba ove dvije metode.

**Tablica 1. Usporedba parametarskih i neparametarskih metoda**

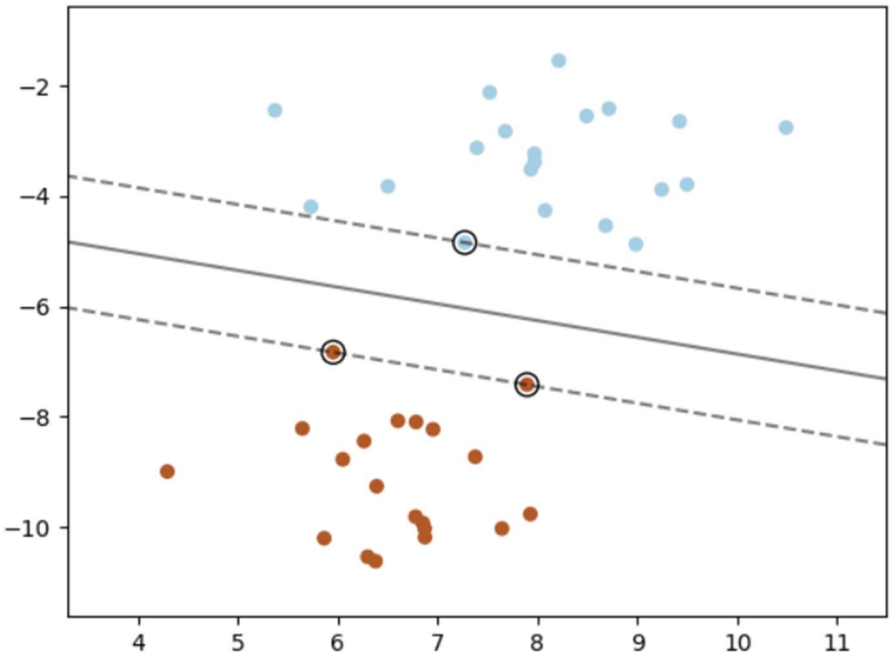
| **Parametarske metode** | **Neparametarske metode** |
|---|---|
| Hipoteza je definirana do na parametre $\boldsymbol{\theta}$ | Hipoteza nije eksplicitno definirana |
| Broj parametara modela n (složenost modela) ne ovisi o broju primjera N | Broj parametara ovisi o broju primjera |
| Pretpostavljaju da se podaci ravnaju po nekom modelu (distribuciji) | Ne pretpostavljaju model (distribuciju) podataka |
| Primjeri imaju globalan utjecaj na izgled hipoteze | Lokalna aproksimacija hipoteze u okolici pohranjenih primjera |
| Malo podataka i/ili poznat model/distribucija | Mnogo podataka i nepoznat model/distribucija |
| Skloni su podnaučenosti modela (eng. underfitting) | Skloni su prenaučenosti modela (eng. *overfitting*) |

#### *2.2.1.1. Stroj potpornih vektora*

Algoritam stroj potpornih vektora (eng. *support vector machine)* je model nadziranog učenja koji se koristi za klasifikaciju i regresiju osmišljen 1963. godine od strane dva ruska matematičara V. N. Vapnik i A. Y. Chervonenkis. Algoritam je originalno bio razvijen za klasifikaciju (SVC), 1992. godine su ga nadogradili s tzv. jezgrenim funkcijama koje su omogućile primjenu SVM-a na nelinearne probleme, daljnje proširenje je bila formulacija SVM-a mekom marginom 1995. godine, a 1996. godine je predložena regresija SVM-a (SVR) od strane V. N. Vapnik, H. Drucker, C. J. C. Burges, L. Kaufman i A. J. Smola.

SVC (eng. *support vektor classifier*) ili klasifikator potpornih vektora je linearni binarni klasifikator koji može rješavati nelinearne probleme upravo pomoću jezgrenog trika (B. E. Boser, I. M. Guyon i V. N. Vapnik, 1992). Kao i svaki klasifikacijski model, cilj SVM-a je odrediti kojoj klasi određeni primjer pripada. Algoritam SVC zadatak rješava problemom maksimalne margine. Klasifikacija pomoću algoritma stroja potpornih vektora ima mnoštvo primjena poput klasifikacije slika, kategorizacije teksta, primjene u bioinformatici, medicini,

fizici i mnogim drugim područjima [11]. Na području bioinformatike SVC postiže najbolje rezultate u rješavanju jednog važnog problema naziva „*protein remote homology detection*“.

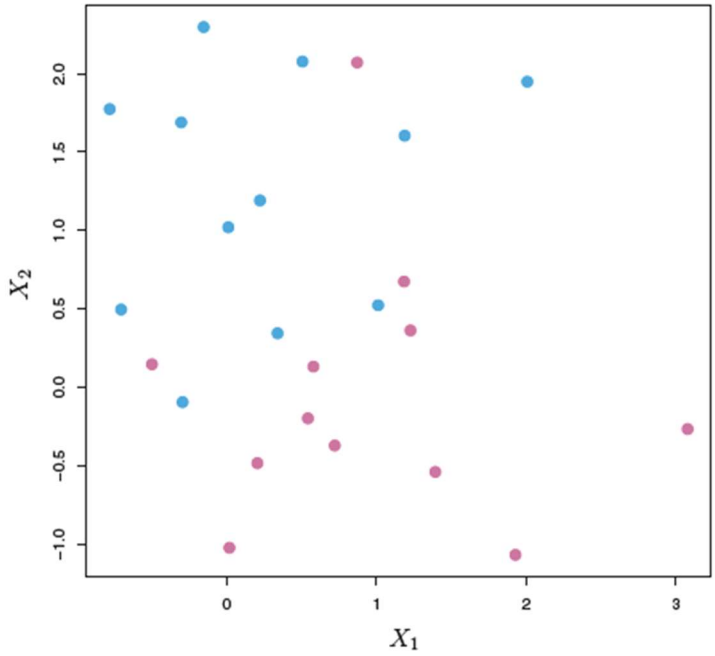



**Slika 11 Hiperravnina SVM**

Ideja algoritma je postaviti hiperravninu (granicu između klasa $h(\mathrm{x}) = 0$) tako da bude najviše udaljena od primjera iz dviju klasa. Može biti $(n-1)$ dimenzionalnih hiperravnina između dvije klase. Primjeri iz dviju klasa na maksimalnim marginama se nazivaju potporni vektori po čemu je i sam algoritam dobio ime, a na [Slika 11.] to su dva zaokružena crvena kruga i jedan zaokruženi plavi krug koji se nalaze na iscrtkanim linijama (marginama).

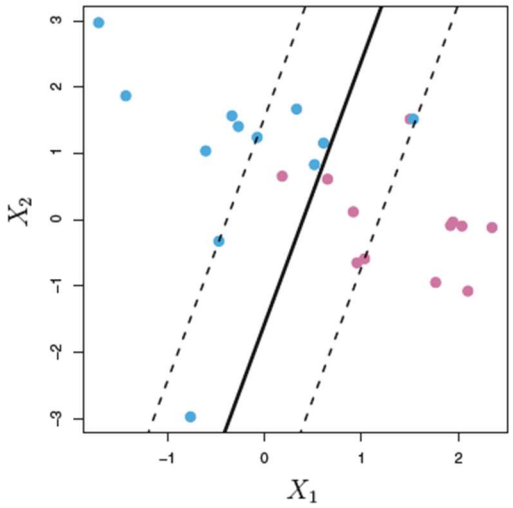
Konačno, predikcija algoritma stroja potpornih vektora se radi na temelju usporedbe ulaznog primjera i odabranih označenih primjera, tj. potpornih vektora. Ova metoda je poznata i kao metoda tvrde margine. Hiperravnina je linija paralelna s marginama i nalazi se točno na jednakoj udaljenosti od obje margine. Pronalaženje hiperravnine koja maksimizira udaljenost te iste hiperravnine do najbližeg primjera, tj. maksimizira marginu naziva se optimizacijski postupak te on omogućuje jako dobru generalizaciju.

No, postoje slučajevi kada nije tako jednostavno odrediti hiperravninu koja dijeli područje na dvije klase kao što to prikazuje [Slika 12.].




**Slika 12 Postoje dvije klase promatranja, prikazane plavom i ljubičastom bojom. U ovom slučaju, dvije klase se ne mogu odvojiti hiperravninom, pa se klasifikator maksimalne margine ne može koristiti**

Klasifikator potpornih vektora, često nazivan i klasifikator meke ravnine, se upravo bavi ovim problemima. Umjesto da se traži maksimalna moguća margina tako da svaki primjer nije samo na ispravnoj strani hiperravnine, već i na ispravnoj strani margine, kao što to radi klasifikator tvrde margine koji je objašnjen gore, klasifikator meke ravnine dopušta da neki primjeri budu na netočnoj strani margine ili čak netočnoj strani hiperravnine. Upravo se zato i naziva meka margina jer ju mogu „preskočiti“ neki od promatranih primjera [13].




**Slika 13 Postavljanje hiperravnine i meke margine klasifikatorom potpornih vektora**

Meka margina radi u korist dobre klasifikacije većine primjera. Omogućava pogrešno klasificiranje nekih primjera u korist ostale većine [Slika 13.]

Matematički se SVM može opisati kao linearan model (7)

$$h(\mathbf{x}; \mathbf{w}, w_0) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \quad \textbf{(7)}$$

a traži se hiperravnina čija je jednadžba dana kao (8)

$$h(\mathbf{x}; \mathbf{w}, w_0) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 = 0 \quad \textbf{(8)}$$

gdje su **w** i $w_0$ zasad neodređeni parametri, a **w** je ujedno i vektor normale na hiperravninu.

$y_i$ predstavlja klasu kojoj podatak pripada te poprima vrijednost 1 za klasu na pozitivnoj strani pravca, odnosno strani u smjeru vektora normale na hiperravninu **w** i vrijednost $-1$ za klasu na drugoj strani hiperravnine. Formulacija problema maksimalne margine se matematički može definirati ovako (9)

$$\underset{\mathbf{w},w_0}{\operatorname{argmax}} \left\{ \frac{1}{\|w\|} \min_i \left\{ y^{(i)}(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0) \right\} \right\} \quad \textbf{(9)}$$

Nakon što je definiran problem maksimalne margine, potrebno ga je preformulirati u problem optimizacije uz ograničenja. Ograničenja postavljamo tako da je udaljenost hiperravnine do najbližih primjera jednaka točno 1 pa za sve primjere vrijedi (10)

$$y^{(i)}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} + w_0\right) \geq 1, \qquad i = 1, \dots, N \quad \textbf{(10)}$$

Problem se stoga svodi na (11)

$$\underset{\mathbf{w},w_0}{\operatorname{argmax}} \left\{ \frac{1}{\|w\|} \right\} \quad \textbf{(11)}$$

Odnosno (12)

$$\underset{\mathbf{w},w_0}{\operatorname{argmin}} \left\{ \frac{1}{2} \|w\|^2 \right\} \quad \textbf{(12)}$$

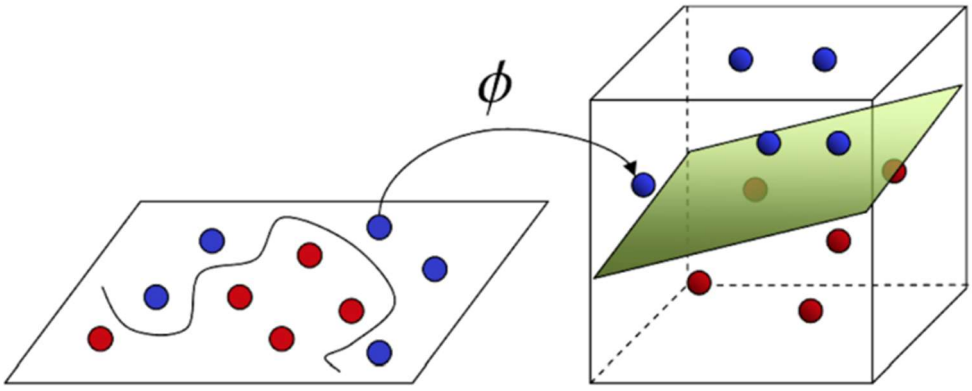
Ovim postupkom se problem maksimalne margine sveo na kvadratno programiranje koje se rješava metodom Lagrangeovim multiplikatorima i dovodi u dualni model stroj potpornih vektora koji predikciju novih primjera ne radi na temelju težina značajki, nego na temelju sličnosti primjera s potpornim vektorima dviju klasa. Pritom je primarna formulacija (kvadratno programiranje) parametarski model, a dualna formulacija (maksimalna margina)

neparametarski model. Jednadžba hiperravnine koja predstavlja granicu između dvije klase u dualnoj formulaciji izgledat će ovako (13)

$$h(\mathbf{x};\mathbf{w},w_0)=\mathbf{w}^{\mathrm{T}}\mathbf{x}+w_0=\sum_{i=1}^{N}\alpha_i y^{(i)}x^{T}x^{(i)}+w_0=0 \qquad \textbf{(13)}$$

Gdje se za svaki $x^{(i)}$ (primjer) treba pohraniti njegov $\alpha_i$ tj, Lagrangeov multiplikator i $y^{(i)}$ oznaka tj. klasa primjera. U dualnoj formulaciji moguće je imati najviše *N* potpornih vektora koji ovise o broju primjera (broju ulaznih podataka), dok je broj parametara proporcionalan s brojem potpornih vektora te se zbog toga naziva neparametarska formulacija. U klasifikaciji dualnog modela sudjeluju samo oni primjeri za koje vrijedi $\alpha_i > 0$ i kod klasifikacije dani primjer treba usporediti sa svakim potpornim vektorom. Činjenica da se pravilo odlučivanja klasifikatora potpornih vektora temelji na malom podskupu koji uključuje potporne vektore znači da je algoritam prilično robustan na primjerima koji su daleko od hiperravnine.

Budući da se klasifikator potpornih vektora definira kao linearni binarni klasifikator, postavlja se pitanje kako on rješava nelinearne probleme i probleme s više klasa. Ranije je spomenuto da se nelinearni problemi ovim algoritmom mogu riješiti preko jezgrenog trika. Jezgreni trik je osmišljen za preslikavanje $\Phi(x):\ \mathbb{R}^n \rightarrow \mathbb{R}^m$ u višedimenzionalan prostor u kojem bi primjeri mogli biti linearno odvojivi [14]




**Slika 14 Prikaz preslikavanja podataka u višedimenzionalan prostor radi postizanja linearne odvojivosti**

Jezgreni trik računa sličnost pomoću određene jezgrine funkcije koja će implicitno odrediti i preslikavanje u višedimenzionalan prostor kao (14)

$$K(x, x') = \Phi(x) \cdot \Phi(x') \quad \textbf{(14)}$$

U poglavlju 2.2.1 su objašnjene sheme problema s više klasa – shema jedan-naspram-jedan i shema jedan-naspram-ostali. Algoritam SVM ne radi ništa drugačije od te dvije sheme. No kod obje sheme postoje i određeni problemi. Shema jedan-naspram-jedan ima veći broj klasifikatora i može se dogoditi da postoji područje koje ne pripada ni jednoj klasi, dok shema jedan-naspram-ostali ima manji broj klasifikatora, no može uzrokovati neuravnoteženost klasa te se također može naći područje koje ne pripada niti jednoj klasi. Ipak, postoji i način kako da se izbjegne to područje. To se može napraviti ako se umjesto predznaka za određivanje klase koriste kontinuirane vrijednosti naučenih funkcija. Na primjer, ako jedna klasa ima značajno manji broj primjera od ostalih, klasifikatoru koji odgovara toj klasi bit će isplativije uvijek reći da primjer pripada ostalima jer će time imati jako veliku točnost.

Algoritam stroj potpornih vektora se pokazao dobar u raznim postavkama i često se smatra jedan od najboljih „out of the box" klasifikatora [16].

#### *2.2.1.2. Klasifikator k-najbližih susjeda*

Algoritam k-najbližih susjeda (eng. *k-nearest neighbours*, **KNN**) je jedan od osnovnih algoritama, no ujedno i jedan od najbitnijih klasifikacijskih algoritama u strojnom učenju. To je neparametarski klasifikacijski algoritam koji vrši predikciju na temelju većinske oznake *k* najbližih susjeda (eng. *nearest neighbours*) (15)

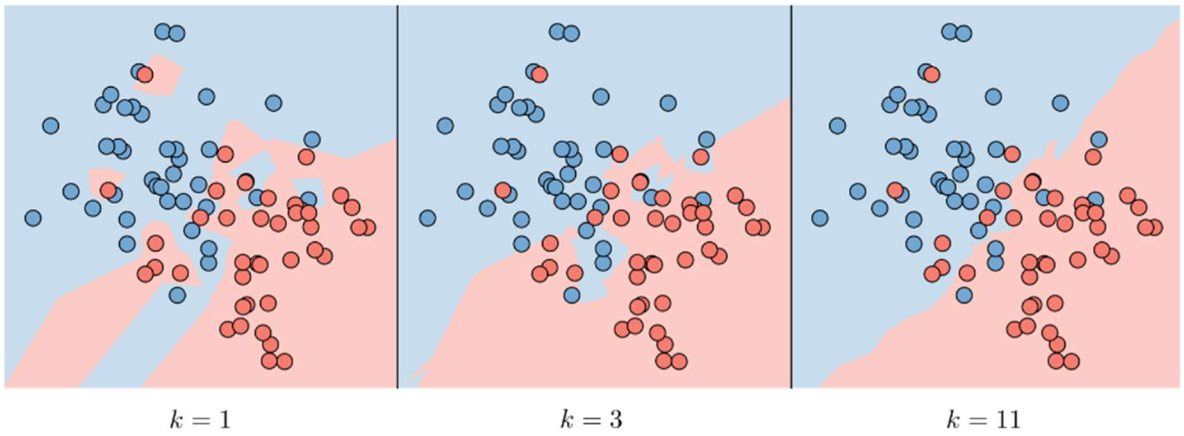
$$h(\mathrm{x}) = \underset{j\in\{o,\ldots,K-1\}}{\operatorname{argmax}} \sum_{(x^{(i)},y^{(i)})\in \mathrm{NN}_k(x)} \mathbf{1}\{y^{(i)} = j\} \quad \textbf{(15)}$$

KNN pripada domeni nadziranog učenja te ima mnoštvo primjena u prepoznavanju uzoraka (eng. *pattern recognition*), rudarenju podataka (eng. *data mining*) i otkrivanju upada (eng. *intrusion detection*) [17]. Ovaj algoritam uzima u obzir različite težišne točke i koristi Euklidsku funkciju za usporedbu udaljenosti dvije točke. (16)

$$d\left(x^{[a]}, x^{[b]}\right) = \sqrt{\sum_{j=1} (\boldsymbol{x_j^{[a]}} - \boldsymbol{x_j^{[b]}})^2} \quad \textbf{(16)}$$

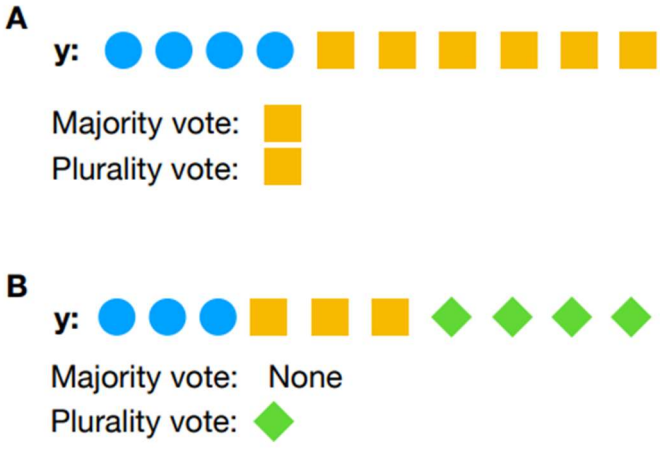
Zatim analizira rezultate i klasificira svaki bod u skupinu kako bi ga optimizirao na mjesto sa svim najbližim bodovima. Nove primjere u ulaznom prostoru klasificira koristeći većinu glasova od svojih *k* susjeda. Kako klasifikacija ovisi o broju susjeda *k* prikazuje [Slika 19.]. *k*

je hipermetar algoritma - što je veći parametar $k$, veća je pristranost, a što je niži parametar $k$, veća je varijanca i model je složeniji.




**Slika 15 Prikaz klasifikacije pomoću k-najbližih susjeda te kako klasifikacija ovisi o broju susjeda koji se uzimaju u obzir**

Da bi algoritam mogao odlučiti kojoj klasi dodijeliti pojedini primjer, primjer mora ispuniti tzv. „majority voting“ tj. „glasovanje većine“. Termin „majority voting“ je pomalo nespretan jer se obično odnosi na referentnu vrijednost od >50% za donošenje odluke. Npr. u zadanom skupu podataka algoritam je definirao dvije klase: crvenu i plavu. Neka je zadan $K = 3$ najbližih susjeda koji će se uzeti u obzir prilikom odabira klase. Novi primjer ulazi u ulazni prostor te se smješta blizu 2 crvena primjera i 1 plavog primjera. Novi primjer će se klasificirati kao crvena klasa jer ispunjava uvjete glasovanja većine te algoritam prepoznaje kako je on 66% za crvenu klasu, a 33% za plavu klasu. Kod binarne klasifikacije će uvijek biti ili većina ili izjednačeno. No prilikom klasificiranja primjera u više od dvije klase, većina nije potrebna za donošenje odluka. Na primjer, u klasifikaciji s 3 klase učestalost od $> \frac{1}{3}$ (približno 33%) je dovoljna za određivanje klase. Slikovito je ova situacija prikazana na [Slika 16.].




**Slika 16 Razlika između „majority vote“ i „plurality vote“**

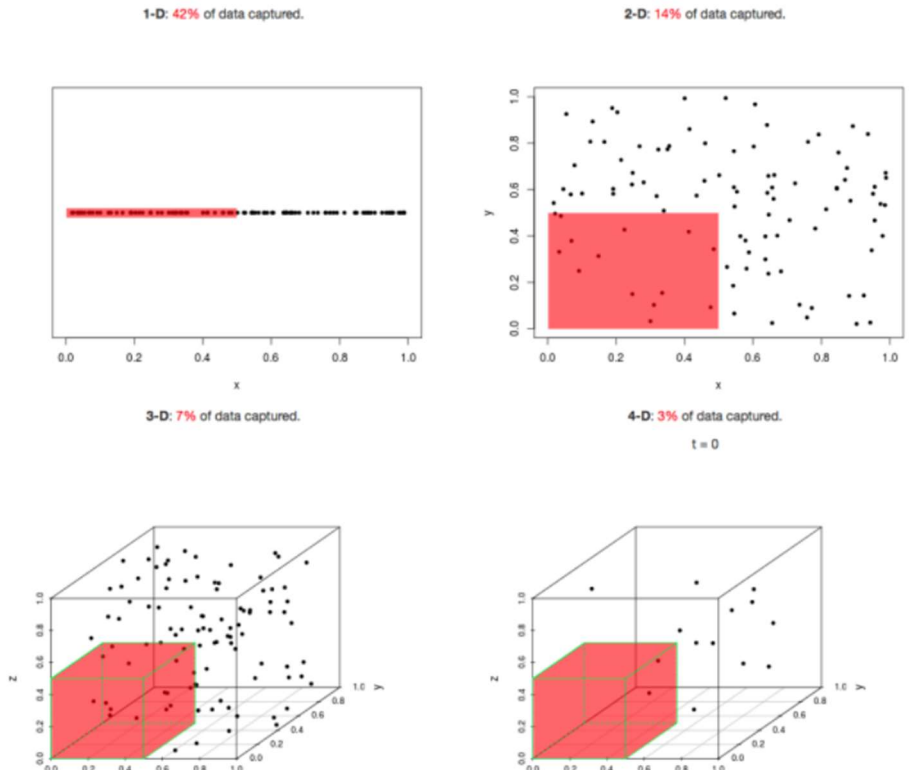
Algoritam KNN se naziva i „lazy learning“ model. On pohranjuje samo skup podataka za treniranje u memoriju te ne zahtijeva nikakvu obuku. To također znači da se svi proračuni događaju u samom trenutku kada se vrši klasifikacija ili predviđanje. S obzirom na to da se uvelike oslanja na memoriju za pohranjivanje svih podataka o treniranju, naziva se i „*istance-based*“ i „*memory-based*“ model učenja.

Budući da algoritam predviđa klasu novih primjera identificirajući klase primjera koji su mu najbliži, skala varijabli je jako bitna. Svi primjeri koji su na velikoj skali imat će mnogo veći utjecaj na udaljenost primjera, a time i na KNN klasifikator, nego primjeri koji su na maloj skali. Značajka reda veličine $10^3$ će imati puno veći utjecaj na klasifikaciju od značajke reda veličine $10^1$. Zato je posebno kod algoritma najbližih susjeda bitna standardizacija podataka tako da sve standardizirane varijable imaju srednju vrijednost 0 i standardnu devijaciju 1. Tako će sve varijable biti na usporedivoj ljestvici [9].

Težinski utjecaj primjera (kernel) koji ovisi o udaljenosti/sličnosti se matematički može zapisati ovako (17)

$$h(\mathrm{x}) = \underset{j\in\{o,\ldots,K-1\}}{\operatorname{argmax}} \sum_{(x^{(i)},y^{(i)})\in \mathrm{NN}_k(x)} \boldsymbol{\kappa}(\boldsymbol{x}^{(i)},\boldsymbol{x})\mathbf{1}\{y^{(i)}=j\} \qquad \textbf{(17)}$$

Međutim, porastom skupa podataka za klasificiranje, algoritam postaje sve neučinkovitiji ugrožavajući ukupnu izvedbu modela. Tu se javljaju alternative iscrpnom pretraživanju koje se



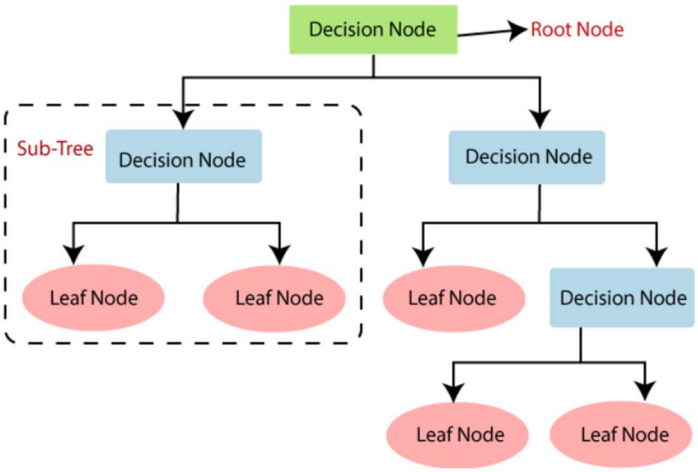

**Slika 17 Porastom broja dimenzija, udaljenost između susjeda raste**

dijele na egzaktne metode (npr. eng. *ball tree*) i aproksimativne metode (eng. *localy sensitive hashing*, LSH). Isto tako, vrlo često ga se veže i uz prokletstvo dimenzionalnosti (eng. *curse of dimensionality*). U strojnom učenju prokletstvo dimenzionalnosti odnosi se na scenarije s fiksnom veličinom primjera za treniranje, ali sve većim brojem dimenzija i rasponom vrijednosti značajki u svakoj dimenziji u visokodimenzionalnom prostoru značajki. U algoritmu KNN veći broj dimenzija [Slika 17.] postaje problematičniji jer što se više dimenzija doda, to veći volumen u hiperprostoru treba biti da bi se uhvatio isti broj susjeda. Kako obujam postaje sve veći i veći, susjedi postaju sve manje slični primjeru u ulaznom prostoru za kojeg je potrebno odrediti klasu [19].

#### *2.2.1.3. Stablo odluke*

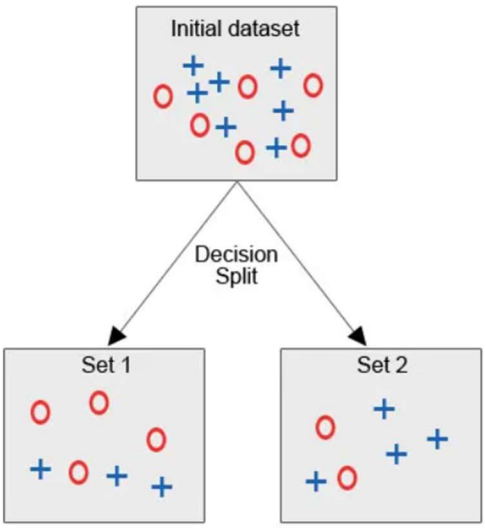
Algoritam stabla odlučivanja jedan je od najpopularnijih algoritama koji se koristi za izgradnju klasifikacijskih i regresijskih modela [20]. Djelomično je tome razlog jer se ta metoda provlači kroz mnoga znanstvena i životna područja. Osim u strojnom učenju, u znanosti se provlači u analizi odlučivanja (eng. *Decision analysis*) gdje se koristi kako bi se vizualno (grafički) predstavio način donošenja odluke te se polazi od ručnog kreiranja stabla. Metoda odlučivanja se bazira na jednostavnoj Booleovoj logici te oponaša ljudsku sposobnost razmišljanja dok donosi odluke te ih je stoga i vrlo lako za razumjeti. Stablo odlučivanja jednostavno postavlja pitanje i na temelju odgovora (DA/NE) dalje dijeli stablo na podstabla. Što je stablo dublje, to su pravila složenija i model prikladniji. Huntov algoritam, koji je razvijen 1960-ih za modeliranje ljudskog učenja u psihologiji, čini temelj mnogih popularnih algoritama stabla odlučivanja kao što su ID3, C4.5 i CART [21].

Stablo odlučivanja (DT) je neparametarska nadzirana metoda učenja koja se koristi za klasifikaciju i regresiju. Za izgradnju stabla se koristi CAST algoritam (eng. *Classification and Regression Tree Algorithm*). Stabla odlučivanja su prediktivni modeli koji na temelju podataka izvode njihove veze u cilju dobivanja izlaznih vrijednosti (klasa ili vrijednost ciljne varijable). Kao takvi se koriste najčešće u rudarenju podataka (eng. *Data mining*) tj. traženju skrivenih veza među podacima. [Slika 18] prikazuje općenitu formu stabla s određenim pojmovima u svakoj od etapa tog stabla. Stablo započinje s korijenskim čvorom (eng. *Root Node*) – na slici obojeno zeleno, koji predstavlja cijelu populaciju, odnosno skup primjera, a on se dalje dijeli na dva ili više homogenih skupova. Kao što je i prikazano, korijenski čvor je i čvor odluke.




**Slika 18 Osnovni pojmovi stabla odlučivanja**

Čvor odluke je kada se čvor dijeli na daljnje podčvorove. Ovdje ključni faktor igra svima dobro znana entropija. Entropija u kontekstu strojnog učenja i baza podataka označava neuređenost podataka. Koristi se u stablu odlučivanja jer je krajnji cilj tog algoritma grupiranje sličnih skupina podataka u slične klase, tj. uređivanje podataka. Listovi stabla su čvorovi koji se ne dijele dalje na list ili završni čvor te oni ujedno predstavljaju vrijednost ciljne (izlazne) varijable tj. klase.




**Slika 19 Entropija u algoritmu stabla odlučivanja**

Stablo se dobiva „učenjem“ na podacima tako da se vrši grananje (eng. *splitting*) izvornog skupa podataka u podskupove na temelju testiranja vrijednosti varijabli. Proces se ponavlja na

svakom izvedenom podskupu na rekurzivni način (eng. *recursive partitioning*). Rekurzija je završena kada podskup određenog čvora ima sve iste vrijednosti izlazne varijable ili kada daljnje grananje više ne doprinosi poboljšanju rezultata (Witten, Frank, 2000) [23].

CAST algoritam na temelju raspoloživih podataka o ulaznim i izlaznim varijablama kreira binarno stablo grananjem slogova u svakom čvoru prema funkciji određenoj za svaku ulaznu varijablu. Evaluacijska funkcija korištena za odluku je Gini indeks (eng. *Gini impurity*, IG) definiran prema formuli (18) (Apte, 1997.)

$$I_G(t) = 1 - \sum_{i=1}^{m} p_i^2 \tag{18}$$

gdje je $t$ trenutni čvor, $p_i$ je vjerojatnost klase $i$ u čvoru t, a $m$ je broj klasa u modelu. Gini indeks je vjerojatnost netočnog klasificiranja nasumične podatkovne točke u skupu podataka ako je označena na temelju distribucije klase skupa podataka. U slučaju da postoji više klasa s istom i najvećom vjerojatnošću, klasifikator će predvidjeti klasu s najnižim indeksom među tim klasama. S obzirom na to da svako sljedeće grananje ima na raspolaganju manje reprezentativnu populaciju, potrebno je smanjivati stablo (eng. *pruning*), kako bi se dobila točnija klasifikacija.

Koraci u kreiranju stabla odlučivanja:

i) Priprema podataka, priprema ulaznih i izlaznih varijabli te podjela primjera na primjere za treniranje i primjere za testiranje stabla

ii) Izbor algoritma i parametara stabla

iii) Generiranje grafičkog stabla odlučivanja (eng. *tree plot*) i numeričke strukture stabla (eng. *tree structure*)

iv) Tumačenje rezultata stabla odlučivanja – računanje greške

v) Upotreba stabla odlučivanja u praksi

Stablo odlučivanja ima mnoge prednosti. Prva je da zahtjeva malo pripreme podataka. Druge metode, poput ranije spomenutih SVM i KNN, često zahtijevaju normalizaciju podataka, izradu lažnih varijabli te uklanjanje praznih vrijednosti. Može baratati podacima koji su i numerički i kategorički, ima sposobnost rješavanja problema s više izlaza i lako ga je potvrditi korištenjem

statističkih testova. Ima dobre rezultate čak i ako su njegove pretpostavke donekle narušene pravim modelom iz kojeg su podaci generirani.

S druge strane, koliko god je izvrsno i lako shvatljivo korištenje jednostavne Booleove logike i binarnog klasifikatora koji daju izlaz DA ili NE i 0 ili 1, postoje koncepti koje stabla odlučivanja ne izražavaju lako, kao što su XOR, problemi pariteta ili multipleksera. Također, mogu biti nestabilna jer male varijacije u podacima mogu rezultirati generiranjem potpuno drugačijeg stabla, a imaju i sklonost prenapučenosti (eng. *to overfit*) i ne generaliziraju dobro na nove podatke. Ovakav se scenarij može izbjeći s prethodnim smanjivanjem stabla ili naknadnim smanjivanjem stabla (eng. *pre-pruning or post-pruning*). Prethodno smanjivanje stabla zaustavlja rast stabla kada nema dovoljno podataka, dok naknadno smanjivanje stabla uklanja podstabla s neadekvatnim podacima nakon izgradnje stabla.

# 3. IZVEDBA ZADATKA U PYTHON-U

## 3.1. Programski jezik Python

Python je besplatan, *open-source*, interpretacijski programski jezik opće namjene i visoke razine. Dopušta korištenje više stilova programiranja – objektno, strukturno i aspektno orijentirano programiranje, a zbog raznolikosti stilova programiranja, intuitivnosti kod korištenja i jednostavnosti jedan je od najpopularnijih programskih jezika među mlađim i starijim programerima. Osmišljen je 1989. godine od strane Nizozemca Guido van Rossum koji ga je nazvao po popularnoj grupi Monty Python iz TV-emisije Monty Python's Flying Circus. Većina programera početnika upravo koristi Python jer omogućava brzo svladavanje sintakse u mjeri potrebnoj za rješavanje problema, a služi i kao dobra podloga i dobar prvi jezik za buduće programere u jezicima *Java*, *C++* ili *C#*.

U ovom radu korišten je Python 3.9.7, inačica Pythona koja sadrži većinu potrebnih biblioteka koje su korištene prilikom implementacije funkcija čitanja, čišćenja, skaliranja i vizualizacije podataka te izvedbe zadanih algoritama strojnog učenja. Biblioteke koje nisu dio ove verzije Pythona instalirane su pomoću „pip install“ komande u Jupyter notebook-u.

### *3.1.1. Python biblioteke i paketi*

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
        from sklearn.metrics import classification_report
```

**Slika 20 Pregled potrebnih biblioteka i paketa**

Za primjenu tri različita algoritma strojnog učenja te izvršavanja funkcija pripreme podataka potrebno je bilo implementirati potrebne biblioteke i pakete što je prikazano na slici [Slika 20.]

#### *3.1.1.1. Pandas*

Pandas je *open source* biblioteka stvorena za Python programski jezik koja se koristi za obradu i analizu podataka. Pruža strukture podataka i opreacije za manipulaciju numeričkim tablicama. Naziv „Pandas“ je izveden iz izraza „Panel dana“. Pandas se uglavnom koristi za analizu, spajanje ili preoblikovanje podataka te se podaci mogu uvesti iz raznih izvoda i tipova datoteka.

Neke od značajke Pandas biblioteke su: „DataFrame“ tj. tablica za manipulaciju podacima s integriranim indeksiranjem, usklađivanje podataka i integrirano rukovanje podacima koji nedostaju, preoblikovanje i uređivanje skupova podataka, spajanje i združivanje setova podataka te filtriranje podataka.

#### *3.1.1.2. NumPy (Matplotlib, seaborn)*

NumPy (eng. *Numerical Python*) je biblioteka za vektorsku i matričnu manipulaciju. (eng. *array*), razne izvedene objekte te asortiman rutina za brze operacije na nizovima uključujući matematičke operacije, logičke, manipulacije oblikom, sortiranje, odabir, I/O, diskretne Fourtierove transformacije, osnovnu linearnu algebru te još mnogo toga.

Matplotlib je biblioteka implementirana u obliku matematičkog proširenja NumPy biblioteke. Koristi se za stvaranje statičnih, animiranih i interaktivnih vizualizacija u Pythonu te njihovo ugrađivanje u programske aplikacije.

Seaborn je biblioteka za vizualizaciju statističkih podataka te također dolazi kao nadogradnja i matematiško proširenje NumPy biblioteke.

#### *3.1.1.3. Scikit-learn*

Scikit-learn pruža standardno Python sučelje za niz nenadziranih i nadziranih metoda učenja. Nudi razne alate za prilagođavanje modela, pretprocesiranje podataka, odabir modela, procjenu modela i mnoge druge alate.

## 3.2. Funkcije prilagodbe podataka

### *3.2.1. Funkcije čitanja podataka*

Funkcija pd.read_cvs() iz biblioteke Pandas omogućava čitanje podataka iz datoteke .cvs (zarezom odvojene vrijednosti). Svaki zarez u retku prepoznaje kao odvajanje kolona te shodno tome ubacuje podatke u df = DataFrame tj. nama čitljivu tablicu [Slika 21.]

```
In [2]: df = pd.read_csv("C:\\Users\\anjag\\Documents\\FSB\\ZAVRSNI\\Classified_data.csv",index_col=0)
        df.head()
```

Out[2]:

| | WTT | PTI | EQW | SBI | LQE | QWG | FDJ | PJF | HQE | NXJ | TARGET CLASS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.913917 | 1.162073 | 0.567946 | 0.755464 | 0.780862 | 0.352608 | 0.759697 | 0.643798 | 0.879422 | 1.231409 | 1 |
| **1** | 0.635632 | 1.003722 | 0.535342 | 0.825645 | 0.924109 | 0.648450 | 0.675334 | 1.013546 | 0.621552 | 1.492702 | 0 |
| **2** | 0.721360 | 1.201493 | 0.921990 | 0.855595 | 1.526629 | 0.720781 | 1.626351 | 1.154483 | 0.957877 | 1.285597 | 0 |
| **3** | 1.234204 | 1.386726 | 0.653046 | 0.825624 | 1.142504 | 0.875128 | 1.409708 | 1.380003 | 1.522692 | 1.153093 | 1 |
| **4** | 1.279491 | 0.949750 | 0.627280 | 0.668976 | 1.232537 | 0.703727 | 1.115596 | 0.646691 | 1.463812 | 1.419167 | 1 |

**Slika 21 Otvaranje datoteke .csv u Jupyter Notebook-u**

Funkcijom df.head(*n*) ispisujemo prvih *n* redaka, a ako nije naznačeno koliko – ispisuje se prvih 5 redaka što je dovoljno za sami pregled osnovnih informacija u tablici. Obično ovakvi dokumenti s bazama podataka imaju jako puno redova i jako puno primjera te nikome nije u interesu čitati i analizirati 1000+ linija podataka kada to ionako radi računalo. Ono što nas zanima je informacija koliko sveukupno ima redova i stupaca, a to vrlo lako pokazuje funkcija df.shape [Slika 22.]

```
In [137]: df.shape
Out[137]: (1000, 9)
```

**Slika 22 Dimenzije tablice zadanog skupa podataka**

Funkcijom df.info() dobivamo sažetak zadanog skupa podataka [Slika 9.]. Funkcija pokazuje tip podataka u samoj datoteci, u ovom slučaju to su float i integer. Kada znamo tipove podataka, znamo i kako se s njima ponašati. Non-Null Count jednostavno govori jesu li svi podaci valjani u tom skupu podataka i brzo dolazimo informacije ako skup podataka treba dodatno urediti.

```
In [4]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   WTT           1000 non-null   float64
 1   PTI           1000 non-null   float64
 2   EQW           1000 non-null   float64
 3   SBI           1000 non-null   float64
 4   LQE           1000 non-null   float64
 5   QWG           1000 non-null   float64
 6   FDJ           1000 non-null   float64
 7   PJF           1000 non-null   float64
 8   HQE           1000 non-null   float64
 9   NXJ           1000 non-null   float64
 10  TARGET CLASS  1000 non-null   int64
dtypes: float64(10), int64(1)
memory usage: 93.8 KB
```

**Slika 23 Sažetak zadanog skupa podataka**

Funkcijom df.describe() dobivamo deskriptivnu statistiku podataka. Deskriptivna statistika uključuje srednje vrijednosti podataka, standardnu devijaciju, minimalnu i maksimalnu vrijednost podataka te disperziju podataka.

Standardna devijacija je prosječno srednje kvadratno odstupanje numeričkih vrijednosti neke veličine $x_1, x_2, \dots, x_N$ od njihove aritmetičke sredine $\bar{x}$ [13]. Standardnu devijaciju i srednju vrijednost ćemo kasnije koristiti u skaliranju podataka.

```
In [5]: df.describe()
```

Out[5]:

| | WTT | PTI | EQW | SBI | LQE | QWG | FDJ | PJF | HQE | NXJ | TARGET CLASS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.00000 |
| mean | 0.949682 | 1.114303 | 0.834127 | 0.682099 | 1.032336 | 0.943534 | 0.963422 | 1.071960 | 1.158251 | 1.362725 | 0.50000 |
| std | 0.289635 | 0.257085 | 0.291554 | 0.229645 | 0.243413 | 0.256121 | 0.255118 | 0.288982 | 0.293738 | 0.204225 | 0.50025 |
| min | 0.174412 | 0.441398 | 0.170924 | 0.045027 | 0.315307 | 0.262389 | 0.295228 | 0.299476 | 0.365157 | 0.639693 | 0.00000 |
| 25% | 0.742358 | 0.942071 | 0.615451 | 0.515010 | 0.870855 | 0.761064 | 0.784407 | 0.866306 | 0.934340 | 1.222623 | 0.00000 |
| 50% | 0.940475 | 1.118486 | 0.813264 | 0.676835 | 1.035824 | 0.941502 | 0.945333 | 1.065500 | 1.165556 | 1.375368 | 0.50000 |
| 75% | 1.163295 | 1.307904 | 1.028340 | 0.834317 | 1.198270 | 1.123060 | 1.134852 | 1.283156 | 1.383173 | 1.504832 | 1.00000 |
| max | 1.721779 | 1.833757 | 1.722725 | 1.634884 | 1.650050 | 1.666902 | 1.713342 | 1.785420 | 1.885690 | 1.893950 | 1.00000 |

**Slika 24 Statistika zadanog skupa podataka**

Kao što je funkcija df.info() izbacila sve valjane podatke, funkcijom df.isnull() pronalazimo one elemente/čelije koji nemaju u sebi nikakav podatak ili taj podatak nije čitljiv. Funkcija vraća tip bool brojčane vrijednosti te možemo samo vidjeti koliko je tih elemenata, ne i gdje se točno oni nalaze.

```
In [6]: df.isnull().sum()
Out[6]: WTT             0
        PTI             0
        EQW             0
        SBI             0
        LQE             0
        QWG             0
        FDJ             0
        PJF             0
        HQE             0
        NXJ             0
        TARGET CLASS    0
        dtype: int64
```

**Slika 25 Nevaljani podaci**

Kao što je i ranije zaključeno, svi su podaci valjani u skupu podataka.

Funkcijom df.iloc() želimo ispisati sve varijable tipa integer na nekoj lokaciji. S obzirom na to

```
In [8]: # target variable
        df.iloc[:,-1].values

Out[8]: array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
               0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
               0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1,
               1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1,
               1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
               0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
               0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
               1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
               0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
               0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
               1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0,
               0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
               0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1,
               1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0,
               1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
               0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
```

**Slika 26 Vrijednosti kolone TARGET CLASS**

da je funkcija df.info() pokazala da se podaci tipa integer nalaze u zadnjem stupcu naziva „Target class“, ispisat ćemo zadnji stupac preko naredbe [:,-1]. Dvotočka kazuje da funkcija ispiše cijeli stupac, a -1 da ispiše zadnji stupac.

```
In [9]: df.corr()
```

Out[9]:

| | WTT | PTI | EQW | SBI | LQE | QWG | FDJ | PJF | HQE | NXJ | TARGET CLASS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **WTT** | 1.000000 | -0.235255 | -0.301018 | -0.101517 | -0.083401 | -0.183628 | -0.148100 | -0.330977 | 0.324981 | -0.073889 | 0.551394 |
| **PTI** | -0.235255 | 1.000000 | 0.271908 | 0.001580 | 0.097322 | 0.173701 | 0.095060 | 0.246387 | -0.266242 | 0.035834 | -0.468748 |
| **EQW** | -0.301018 | 0.271908 | 1.000000 | -0.040291 | 0.144539 | 0.182021 | 0.059533 | 0.323857 | -0.324656 | 0.126143 | -0.598120 |
| **SBI** | -0.101517 | 0.001580 | -0.040291 | 1.000000 | -0.007983 | 0.022026 | -0.007666 | 0.037767 | 0.078032 | 0.145045 | 0.028874 |
| **LQE** | -0.083401 | 0.097322 | 0.144539 | -0.007983 | 1.000000 | 0.049944 | 0.035750 | 0.083734 | -0.143929 | 0.044496 | -0.205043 |
| **QWG** | -0.183628 | 0.173701 | 0.182021 | 0.022026 | 0.049944 | 1.000000 | 0.098062 | 0.199189 | -0.181809 | 0.036078 | -0.327664 |
| **FDJ** | -0.148100 | 0.095060 | 0.059533 | -0.007666 | 0.035750 | 0.098062 | 1.000000 | 0.122888 | -0.116969 | -0.012923 | -0.214885 |
| **PJF** | -0.330977 | 0.246387 | 0.323857 | 0.037767 | 0.083734 | 0.199189 | 0.122888 | 1.000000 | -0.363736 | 0.028012 | -0.571483 |
| **HQE** | 0.324981 | -0.266242 | -0.324656 | 0.078032 | -0.143929 | -0.181809 | -0.116969 | -0.363736 | 1.000000 | -0.129283 | 0.643989 |
| **NXJ** | -0.073889 | 0.035834 | 0.126143 | 0.145045 | 0.044496 | 0.036078 | -0.012923 | 0.028012 | -0.129283 | 1.000000 | -0.111730 |
| **TARGET CLASS** | 0.551394 | -0.468748 | -0.598120 | 0.028874 | -0.205043 | -0.327664 | -0.214885 | -0.571483 | 0.643989 | -0.111730 | 1.000000 |

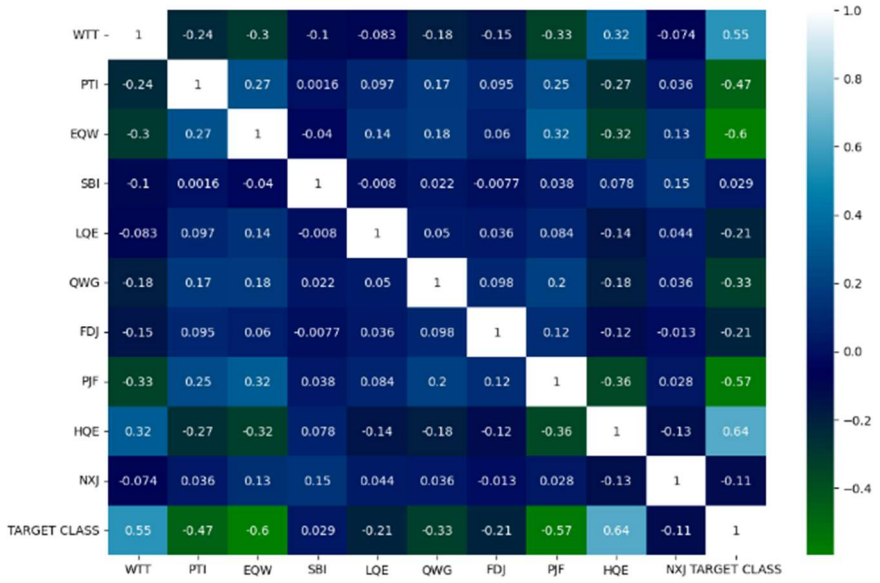
**Slika 27 Tablica korelacije podataka**

U korelacijskoj matrici vrijednosti su u rangu [1, -1] te ona postavlja ovisnost jedne varijable od drugoj. Potpuna korelacija postignuta je kada je koeficijent korelacije jednak 1 ili -1 tj. kada je vjerojatnost da određenoj vrijednosti jedne varijable odgovara određena vrijednost druge varijable jednaka 1 [25]. Kao što je prikazano i u tablici, korelacija WTT naspram WTT je jednaka 1, isto vrijedi po cijeloj dijagonali. Općenito, kad god postoji korelacija blizu -1 ili 1 između varijable x i varijable y, to znači da se y može objasniti lineranom jednadžbom koristeći x kao varijablu. Npr. (19)

$$Y = AX + B \quad \textbf{(19)}$$

Funkcija sns.heatmap je samo način da pomoću boja prikažemo koliko su jake korelacije. U ovom slučaju bijela boja pokazuje pozitivnu korelaciju blizu ili jednaku broj 1.

```
In [10]: plt.figure(figsize=(12,8))
         sns.heatmap(df.corr(),annot = True,cmap='ocean')
Out[10]: <AxesSubplot:>
```




**Slika 28 Heatmap tablice korelacija na [Slika 27]**

Iz matrice korelacija ćemo izvući listu minimalnih vrijednosti korelacije podataka preko funkcije df.corr().min() [Slika 29.].

```
In [11]: df.corr().min()
Out[11]: WTT            -0.330977
         PTI            -0.468748
         EQW            -0.598120
         SBI            -0.101517
         LQE            -0.205043
         QWG            -0.327664
         FDJ            -0.214885
         PJF            -0.571483
         HQE            -0.363736
         NXJ            -0.129283
         TARGET CLASS   -0.598120
         dtype: float64
```

**Slika 29 Minimalne vrijednosti korelacije podataka**

Isto tako i maksimalne vrijednosti korelacija preko funkcije df.corr().max() [Slika 30.].

```
In [12]: df.corr().max()

Out[12]: WTT             1.0
         PTI             1.0
         EQW             1.0
         SBI             1.0
         LQE             1.0
         QWG             1.0
         FDJ             1.0
         PJF             1.0
         HQE             1.0
         NXJ             1.0
         TARGET CLASS    1.0
         dtype: float64
```

**Slika 30 Maksimalne vrijednosti korelacije podataka**

Ako malo bolje proučimo i usporedimo ove dvije liste sa [Slika 28.], možemo primijetiti da je funkcija koja je izvlačila minimalne vrijednosti izvukla najzeleniju vrijednost pojedine značajke, a funkcija koja je izvlačila maksimume uzela sva bijela polja.

Prema ovim podacima možemo uočiti značajke koje najviše odstupaju od korelacije te one mogu znatno narušiti točnost učenja modela. Stoga stupce minimalnih vrijednosti korelacija koji najviše odskaču treba ukloniti ili izbrisati. To radimo funkcijama čišćenja podataka.

### *3.2.2. Funkcije čišćenja podataka*

Funkcijom **del** df[] jednostavno obrišemo kolone koje odstupaju – ovdje su to [EQW] i [PJF] [Slika 18.].

```
In [13]: del df['NXJ']
         del df['SBI']
         df.head()
```

Out[13]:

| | WTT | PTI | EQW | LQE | QWG | FDJ | PJF | HQE | TARGET CLASS |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.913917 | 1.162073 | 0.567946 | 0.780862 | 0.352608 | 0.759697 | 0.643798 | 0.879422 | 1 |
| 1 | 0.635632 | 1.003722 | 0.535342 | 0.924109 | 0.648450 | 0.675334 | 1.013546 | 0.621552 | 0 |
| 2 | 0.721360 | 1.201493 | 0.921990 | 1.526629 | 0.720781 | 1.626351 | 1.154483 | 0.957877 | 0 |
| 3 | 1.234204 | 1.386726 | 0.653046 | 1.142504 | 0.875128 | 1.409708 | 1.380003 | 1.522692 | 1 |
| 4 | 1.279491 | 0.949750 | 0.627280 | 1.232537 | 0.703727 | 1.115596 | 0.646691 | 1.463812 | 1 |

**Slika 31 Brisanje stupaca srednje korelacije**

Ponovno ćemo ispisati tablicu korelacija jer ćemo nju koristiti u daljnjim koracima [Slika 32.].

```
In [14]: df.corr()
```

Out[14]:

| | WTT | PTI | EQW | LQE | QWG | FDJ | PJF | HQE | TARGET CLASS |
|---|---|---|---|---|---|---|---|---|---|
| **WTT** | 1.000000 | -0.235255 | -0.301018 | -0.083401 | -0.183628 | -0.148100 | -0.330977 | 0.324981 | 0.551394 |
| **PTI** | -0.235255 | 1.000000 | 0.271908 | 0.097322 | 0.173701 | 0.095060 | 0.246387 | -0.266242 | -0.468748 |
| **EQW** | -0.301018 | 0.271908 | 1.000000 | 0.144539 | 0.182021 | 0.059533 | 0.323857 | -0.324656 | -0.598120 |
| **LQE** | -0.083401 | 0.097322 | 0.144539 | 1.000000 | 0.049944 | 0.035750 | 0.083734 | -0.143929 | -0.205043 |
| **QWG** | -0.183628 | 0.173701 | 0.182021 | 0.049944 | 1.000000 | 0.098062 | 0.199189 | -0.181809 | -0.327664 |
| **FDJ** | -0.148100 | 0.095060 | 0.059533 | 0.035750 | 0.098062 | 1.000000 | 0.122888 | -0.116969 | -0.214885 |
| **PJF** | -0.330977 | 0.246387 | 0.323857 | 0.083734 | 0.199189 | 0.122888 | 1.000000 | -0.363736 | -0.571483 |
| **HQE** | 0.324981 | -0.266242 | -0.324656 | -0.143929 | -0.181809 | -0.116969 | -0.363736 | 1.000000 | 0.643989 |
| **TARGET CLASS** | 0.551394 | -0.468748 | -0.598120 | -0.205043 | -0.327664 | -0.214885 | -0.571483 | 0.643989 | 1.000000 |

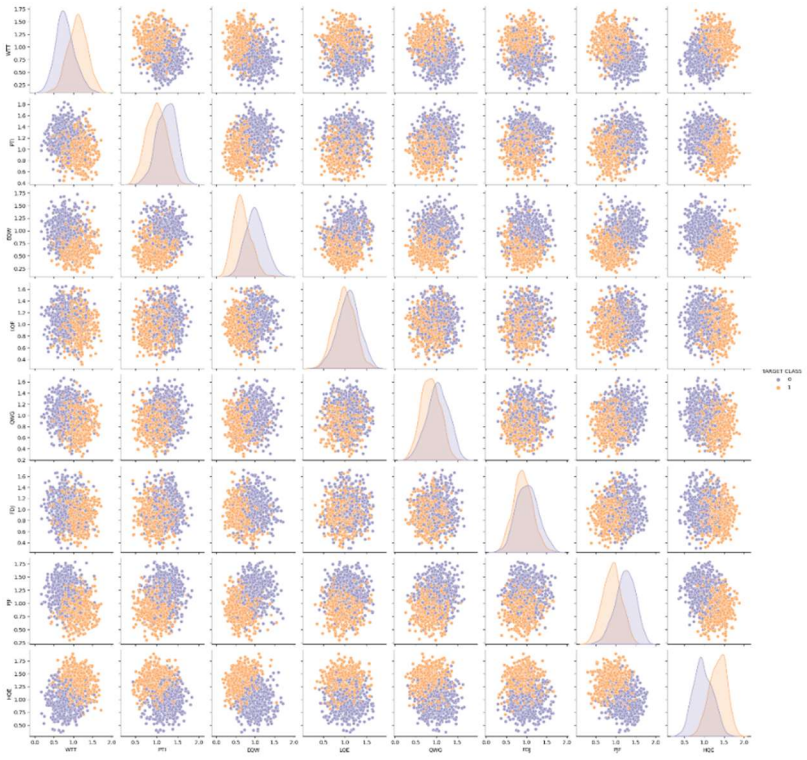
**Slika 32 Tablica korelacije podataka nakon uklanjanja stupaca minimalne srednje korelacije**

### *3.2.3. Funkcije vizualizacije podataka*

Funkcije vizualizacije podataka obično koriste radi lakše analize i lakšeg razumijevanja. Ponekad je potrebno potrošiti sate i sate na analizu velikih skupova podataka, dok vizualizacije omogućuju brzo i učinkovito razumijevanje i tumačenje podataka.

```
In [15]: sns.pairplot(hue='TARGET CLASS',data=df,palette='tab20c_r')
```

```
Out[15]: <seaborn.axisgrid.PairGrid at 0x1d67b546ac0>
```



**Slika 33 Plot međusobne korelacije podataka iz tablice na [Slika 32]**

Dijagram parova na slici gore [Slika 33.] je vizualizacija podataka koja iscrtava odnose parova između svih varijabli skupa podataka. Dijagonalna linija pokazuje distribuciju vrijednosti u toj značajki pomoću histograma. Svaka druga ćelija pokazuje korelaciju kao dijagram raspršenosti između dvije značajke na njihovom sjecištu. Svaka značajka je ispisana u redovima i stupcima pokazujući međusobnu korelaciju. Obojane točkice (plavo i narančasto) označuju svaki pojedini primjer koji se nalazi u ulaznom prostoru primjera, a boje plava i narančasta označuju klase 'TARGET CLASS' u kojoj je plava broj klasa $h(x) = 0$, a narančasta $h(x) = 1$.

### *3.2.4. Funkcije skaliranja podataka*

Funkcija skaliranja podataka uklanja srednju vrijednost i skalira podatke na jediničnu varijancu [Slika 34.]. Cilj skaliranja podataka je unificirati podatke tako da niti jedan od njih (koji možda ima varijancu za redove veličina veću od ostalih) ne dominira funkcijom cilja i učini model nesposobnim učiti iz drugih značajki kako se očekuje [8]. Drugim riječima, skaliranje osigurava

```
In [25]: from sklearn.preprocessing import StandardScaler

In [26]: sc = StandardScaler()
         scale = sc.fit(df.drop(['TARGET CLASS'],axis = 1))
         scaled_features = scale.transform(df.drop('TARGET CLASS',axis = 1))
         scaled_features

Out[26]: array([[-0.12354188,  0.18590747, -0.91343069, ..., -0.79895135,
                  -1.48236813, -0.9497194 ],
                 [-1.08483602, -0.43034845, -1.02531333, ..., -1.12979749,
                  -0.20224031, -1.82805088],
                 [-0.78870217,  0.33931821,  0.30151137, ...,  2.59981844,
                   0.28570652, -0.68249379],
                 ...,
                 [ 0.64177714, -0.51308341, -0.17920486, ..., -2.26133896,
                  -2.36249443, -0.81426092],
                 [ 0.46707241, -0.98278576, -1.46519359, ..., -0.42204066,
                  -0.03677699,  0.40602453],
                 [-0.38765353, -0.59589427, -1.4313981 , ..., -0.7262528 ,
                  -0.56778932,  0.3369971 ]])
```

**Slika 34 Skaliranje podataka**

da sve vrijednosti leže u zajedničkom rasponu te da nema ekstremnih vrijednosti među njima.

Standardno skaliranje primjera x se izračunava prema ovoj formuli (20)

$$z = \frac{x - u}{s} \tag{20}$$

gdje je *u* = srednja vrijednost primjera ili 0 ako je with_mean = False, a *s* = standardna devijacija primjera ili 1, ako je witd_std = False.

## 3.3. Algoritmi strojnog učenja u Python-u

Iz biblioteke sklearn.model.selection potrebno je učitati paket train_test_split koji dijeli podatke slučajnim odabirom na podatke za treniranje i podatke za testiranje [Slika 35.].

```
In [28]: X = scaled_features
         y = df['TARGET CLASS']

         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 1)
```

**Slika 35 Podjela podataka na skupine za treniranje i testiranje**

U kojem omjeru to radi određuje kriterij test_size koji je postavljen na 0.30. To znači da će 70% podataka biti odvojeno za treniranje, a preostalih 30% za testiranje. Prve dvije linije koda predstavljaju ulazne i izlazne podatke na kojima će se model učiti. Ulazni podaci, tj. primjeri su X = scaled features, a izlazni podaci stupac 'TARGET CLASS'.

### *3.3.1. Algoritam stroj potpornih vektora – SVM*

Za izvršavanje algoritma stroja potpornih vektora u Pythonu koristimo se bibliotekom scikit-learn i modulima za stoj potpornih vektora (from sklearn.svm). Modul SVC je ništa drugo doli stroj potpornih vektora klasifikacije koji primjenjuje one-to-one model.

Funkcijom svm.fit() se vrši učenje modela kroz sljedeća dva niza: *x_train* koji sadrži primjere za učenje i *y_train* koji sadrži oznake klase primjera za učenje [Slika 36.].

```
In [29]: from sklearn.svm import SVC
         svm = SVC()
         svm.fit(X_train,y_train)
         y_pred = svm.predict(X_test)
         print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.94      0.96       161
           1       0.94      0.96      0.95       139

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300
```

**Slika 36 Rezultati izvođenja klasifikacijskog algoritma stroja potpornih vektora**

Funkcijom svm.predict(*x_test*) se klasificiraju primjeri (*x_test*) koji su odvojeni za testiranje algoritma te se rezultati pohranjuju pod varijablom *y_pred*. Ovom funkcijom je model predvidio 'TARGET CLASS' za do sada neviđene primjere. Za ocjenjivanje modela koristi se funkcija classification_report koja uspoređuje stvarne klase primjera odvojenih za testiranje (*y_test*) i predviđene klase modelom odgovarajućih ulaznih primjera (*x_test*) [Slika 36.].

### *3.3.2. Algoritam klasifikator k-najbližih susjeda*

Kao i kod algoritma stroja potpornih vektora, za izvršavanje algoritma klasifikatora k-najbližih susjeda ponovno posežemo za scikit-learn bibliotekom, no ovoga puta za modulom najbližih susjeda koji jednostavno implementira algoritam k-najbližih susjeda [Slika 37.].

```
In [35]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier()
         knn.fit(X_train, y_train)
         y_pred = knn.predict(X_test)
```

**Slika 37 Učenje i predikcija modela k-najbližih susjeda**

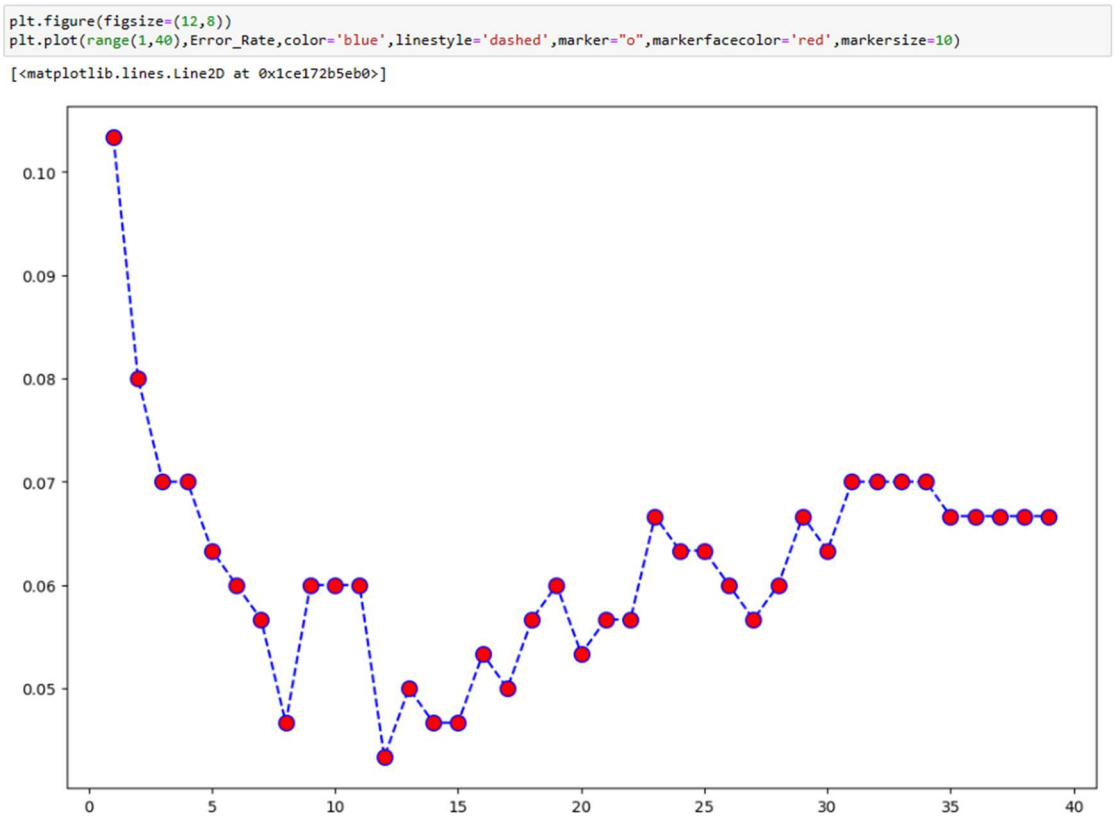
Proces je isti kao i za algoritam stroja potpornih vektora – potrebno je izvršiti učenje modela funkcijom knn.fit() te nakon toga klasificirati primjere odvojene za testiranje (*x_test*). No, za najbolje rezultate potrebno je odrediti i broj susjeda za koji će pogreška biti najmanja [Slika 38.].

```
In [37]: # Best K value
         Error_Rate = []
         for k in range(1,40):
             knn = KNeighborsClassifier(n_neighbors=k)
             knn.fit(X_train,y_train)
             y_pred = knn.predict(X_test)
             Error_Rate.append(np.mean(y_pred!=y_test))
```

**Slika 38 Određivanje broja susjeda**

[Slika 39.] prikazuje srednju vrijednost pogreške klasifikacije za svaki pojedini broj najbližih susjeda koji algoritam koristi u procesu učenja i predikcije ciljne vrijednosti. U interesu je odabrati najbolji model koji može predvidjeti klasu novog ulaznog primjera, a to se postiže sa

najnižom pogreškom. Na slici se vidi da je najmanja srednja vrijednost pogreške oko 0.02 u slučaju kada je $k = 12$, tj. model procjenjuje klasu na temelju 12 najbližih susjeda.




**Slika 39 Prikaz iznosa pogreške klasifikacije (y os) u ovisnosti o broju najbližih susjeda (x os)**

Sada ćemo ponovno pokrenuti klasifikacijski model najbližih susjeda, iskoristiti funkciju učenja te predikcije i ispisati kratki sažetak uspješnosti modela [Slika 40.]

```
In [38]: knn = KNeighborsClassifier(n_neighbors=12)
         knn.fit(X_train,y_train)
         y_pred = knn.predict(X_test)
         print(classification_report(y_test,y_pred))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.95 | 0.96 | 161 |
| 1 | 0.94 | 0.96 | 0.95 | 139 |
| accuracy | | | 0.96 | 300 |
| macro avg | 0.96 | 0.96 | 0.96 | 300 |
| weighted avg | 0.96 | 0.96 | 0.96 | 300 |

**Slika 40 Rezultati izvođenja klasifikacijskog algoritma najbližih susjeda s $k = 12$**

### *3.3.3. Algoritam stablo odluke*

Kao i kod prethodna dva algoritma, za izvršavanje algoritma stabla odluke ponovno posežemo za scikit-learn bibliotekom, no ovoga puta za modulom stabla odluke koji jednostavno implementira traženi algoritam [Slika 41.]. Koraci treniranja i testiranja su isti kao i ranije.

```
In [30]: from sklearn.tree import DecisionTreeClassifier
         dtc = DecisionTreeClassifier()
         dtc.fit(X_train,y_train)
         y_pred = dtc.predict(X_test)
         print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.89       161
           1       0.86      0.90      0.88       139

    accuracy                           0.89       300
   macro avg       0.89      0.89      0.89       300
weighted avg       0.89      0.89      0.89       300
```

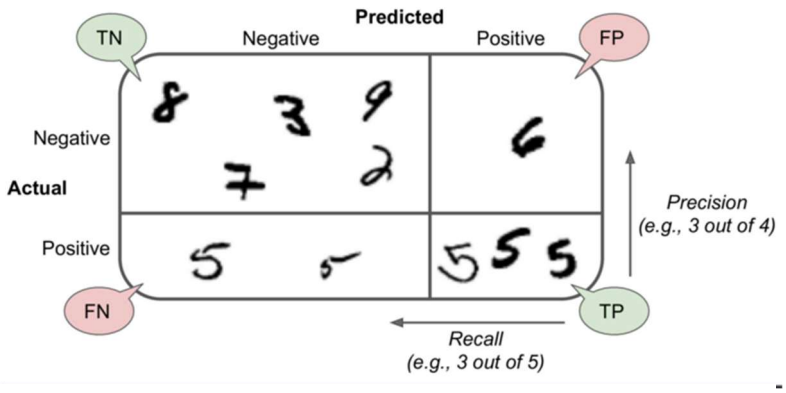
**Slika 41 Rezultati izvođenja algoritma stabla odluke**

## 3.4. Analiza preciznosti algoritama

Preciznost (eng. *precision*) je udio točno pozitivno klasificiranih primjera u skupu pozitivno klasificiranih primjera (21)

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{21}$$

gdje su: TP – true positives (točno pozitivni), FP – false positives (netočno pozitivni), TN – true negatives (točno negativni). Uz ove kategorije klasificiranih primjera postoji još jedna koja je bitna za cjelokupno razumijevanje validacije algoritama, a to je FN – false negatives (netočno negativni). Svi ti klasificirani primjeri se određuju preko matrice zabune (eng. *confusion matrix*). Matrica zabune uspoređuje stvarne oznake i predikcije modela [Slika 29.]. Predviđene klase primjera $x\_test$ su $y\_pred$, a stvarne klase su $y\_test$.




**Slika 42 Matrica zabune**

U [Tablica 2] izvučene su vrijednosti preciznosti sva tri algoritma dobivene iz izvješća o klasifikaciji, vrijednosti preciznosti za sve točno označene primjere, odnosno klase 1.

**Tablica 2 Usporedba preciznosti tri klasifikacijska algoritma**

| | **SVM** | **k-NN** | **Stablo odluke** |
|---|---|---|---|
| **Preciznost** | 0.94 | 0.94 | 0.86 |

Uočavamo kako algoritmi stroj potpornih vektora te k-najbližih susjeda imaju jednaku preciznost prilikom izvođenja algoritma dok je algoritam stablo odluke znatno lošije izveo klasifikaciju primjera. Algoritmi stroj potpornih vektora i k-najbližih susjeda imaju 94% postotnu uspješnost u klasificiranju točno pozitivnih primjera. Ta dva algoritma su također zahtijevala standardizaciju podataka kako bi pospješila svoju izvedbu, dok za algoritam stablo odluke to nije bilo potrebno. Algoritam stablo odluke je sklon prenaučenosti podataka što na kraju dovodi do smanjene generalizacije i na kraju – preciznosti.

# 4. ZAKLJUČAK

Opisani algoritmi strojnog učenja glase kao najjednostavniji klasifikacijski modeli nadgledanog učenja. Lako su shvatljivi, velika je primjena i u stvarnom svijetu te nisu potpuna nepoznanica. Jednostavno ih je primijeniti na nekom binarnom problemu s malo podataka, no takve probleme može čovjek i sam riješiti. Najveća prednost ovih modela je prilagođavanje određenom skupu podataka i treniranje i učenje na tom skupu podataka. Proširenja i nadogradnja algoritma SVM sa mekom marginom i dualnom formulacijom je omogućila bolju generalizaciju što je dovelo do visoke preciznosti algoritma. Algoritam k-najbližih susjeda se pridružuje dobroj generalizaciji i treniranju modela „u trenutku“. On ima veliku fleksibilnost s obzirom da se može i pronaći optimalan broj susjeda koji daje najbolje rezultate. Algoritam stablo odluke je sklon prenaučenosti, odnosno prevelikoj prilagodbi podacima što je na kraju dovelo do znatno manje preciznosti u odnosu na druga dva algoritma, a upravo iz razloga što prenaučenost vodi do smanjene generalizacije podataka. Svojim ulogama u računalnom svijetu omogućili su puno bržu obradu velikih baza podataka i donošenje odluka koje prije nije bilo moguće obraditi u jednom danu, a kamo li tek u nekoliko sekundi. Pojavom raznih programskih biblioteka, dubine i kompleksnosti ovih algoritama postaju pristupačnije široj publici što doprinosi njihovoj popularizaciji u rješavanju problema na području znanosti, istraživanja i poslovanja te tako postaju nezaobilazan alat gotovo u svakom području.

# LITERATURA


[1] Rješavanje problema programiranjem u pythonu, L. Budin, P. Brođanac, Z- Markučić, S. Perić, ELEMENT D.o.o, 2014., pristupljeno 04.01.2023.

[2] 01_uvod_u_strojno_ucenje.pdf, prezentacija kolegija Umjetna inteligencija, doc. dr. sc. Tomislav Stupančić, FSB, 2021., pristupljeno 04.01.2023.

[3] Supervised Machine Learning Classification: A Guide | Built In, pristupljeno 10.01.2023.

[4] Strojno učenje 1 (unizg.hr), pristupljeno 22.01.2023.

[5] Deep learning, Ian Goodfellos and Yoshua Bengio and Aaron Courville, MIT Press, 2016, pristupljeno 22.01.2023.

[6] CS 229 - Machine Learning Tips and Tricks Cheatsheet (stanford.edu), pristupljeno 03.02.2023.

[7] Pattern recognition and machine learning, Bishop, 2006., pristupljeno 22.01.2023.

[8] Strojno učenje (fer.hr), pristupljeno 03.02.2023.

[9] An Introduction to Statistical Learning: with Applications in R, G. James et al., © Springer Science+Business Media New York, 2013, pristupljleno 22.01.2023.

[10] Machine Learning, Step-by-Step Guide To Implement Machine Learning Algorithms with Python, Rudolph Russel, 2018, pristupljeno 22.01.2023.

[11] Real-Life Applications of SVM (Support Vector Machines) - DataFlair (data-flair.training), pristupljeno 18.02.2023.

[12] SVM: Maximum margin separating hyperplane — scikit-learn 1.2.1 documentation, pristupljeno 10.02.2023.

[13] sklearn.preprocessing.StandardScaler — scikit-learn 1.2.1 documentation, pristupljeno 10.01.2023.

[14] *1009206.Final_0036477124_56.pdf (irb.hr), pristupljeno 15.02.2023.

[15] Kernel trick explanation. https:// datascience.stackexchange.com/questions/17536/ kernel-trick-explanation, pristupljeno 18.02.2023.

[16] Support Vector Machines Tutorial - Learn to implement SVM in Python - DataFlair (data-flair.training), pristupljeno 18.02.2023.

[17] Machine Learning Classification - 8 Algorithms for Data Science Aspirants - DataFlair (data-flair.training), pristupljeno 18.02.2023.

[18] CS 229 - Supervised Learning Cheatsheet (stanford.edu), pristupljeno 03.02.2023.

[19] L01: Intro to Machine Learning (02_knn_notes.pdf (sebastianraschka.com)), pristupljeno 18.02.2023.

[20] Algoritam stabla odluke | Objašnjenje i uloga entropije u stablu odluke (education-wiki.com), pristupljeno 19.02.2023.

[21] What is a Decision Tree | IBM, pristupljeno 19.02.2023.

[22] Decision Tree Algorithm in Machine Learning - Javatpoint, pristupljeno 19.02.2023.

[23] PowerPoint Presentation (unios.hr), pristupljeno 19.02.2023.

[24] standardna devijacija | Hrvatska enciklopedija, pristupljeno 09.02.2023.

[25] korelacija | Hrvatska enciklopedija, pristupljeno 09.02.2023.

[26] Confusion Matrix in Machine Learning - GeeksforGeeks, pristupljeno 13.02.2023.