SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Algoritmi strojnog učenja u Pythonu

Dario Nikolić

Voditelj: *Marin Šilić*

Zagreb, travanj, 2019

Sadržaj

1. Uvod

Strojno učenje je dio računarske znanosti koji se bavi razvijanjem algoritama koji se oslanjaju na skupovima podataka kao kolekcije uzoraka nekakvoga fenomena. Takvi uzorci mogu biti ručno rađeni od čovjeka, generirani iz drugoga algoritma ili proizlaziti iz prirode. Drugim riječima, strojno učenje definiramo kao proces rješavanja praktičnog problema prikupljanjem skupa podataka te gradnjom statističkog modela nad tim skupom podataka kojim se rješava problem.

Cilj seminarskog rada je razraditi neke od algoritama strojnog učenja radi boljeg shvaćanja njihovog unutarnjeg djelovanja.

Algoritmi se opisuju pa potom razrađuju. Prilikom razrade obrađuje se njihova matematička podloga i prikazuje algoritam napisan u Pythonu. Ukoliko algoritam ima važnu ili zanimljivu karakteristiku ona se također detaljnije obrađuje.

Algoritmi su napisani po uzoru na knjižnicu scikit-Learn. Svaki model je objekt s metodama fit (treniranje), predict (predviđanje) i score (testiranje).

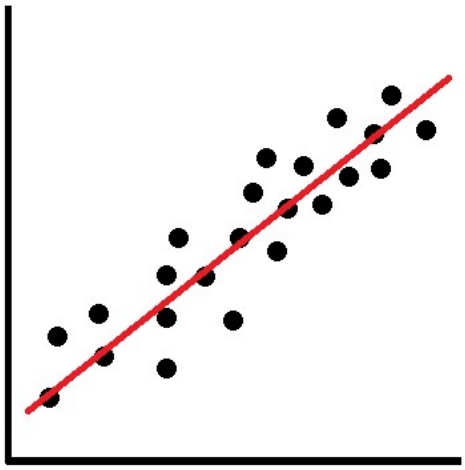
Prilikom implementacije algoritama koristi se Python 3.7.2 te pomoćne knjižnice NumPy i scikit-Learn.

2. Jednostavna linearna regresija

2.1 O algoritmu

Linearna regresija je oblik nadziranog učenja koji se koristi za predviđanje kontinuiranih vrijednosti temeljem korištenja statističkih metoda.

Ideja algoritma je pronaći najbolje odgovarajuću liniju nad skupom kontinuiranih podataka čije značajke[1] ovise o nekoj oznaci.[2] Kako se traži linearna ovisnost, kompleksnost algoritma je minimalna.



Slika 2.1: prikaz najbolje odgovarajuće linije

Iako postoje elegantniji primjeri regresije, linearna regresija je još uvijek korisna i široko korištena metoda. Mnogi kompleksniji primjeri statističkog modela učenja se mogu promatrati kao generalizacija ili proširenje linearne regresije.

Često se koristi pri analizi marketinga, predviđanje cijena dionica, analizi anketa, analize podataka vremenskih uvjeta i mnogo više.

[1] Individualno mjerljivo svojstvo ili obilježje fenomena koji se promatra.

[2] Izlaz (output) treniranog modela.

2.2 Matematička podloga

Kako se traži linearna ovisnost kao model će se koristiti obična linearna jednadžba:

$$y = mx + b$$

gdje m predstavlja nagib te b predstavlja presjek na y-osi.

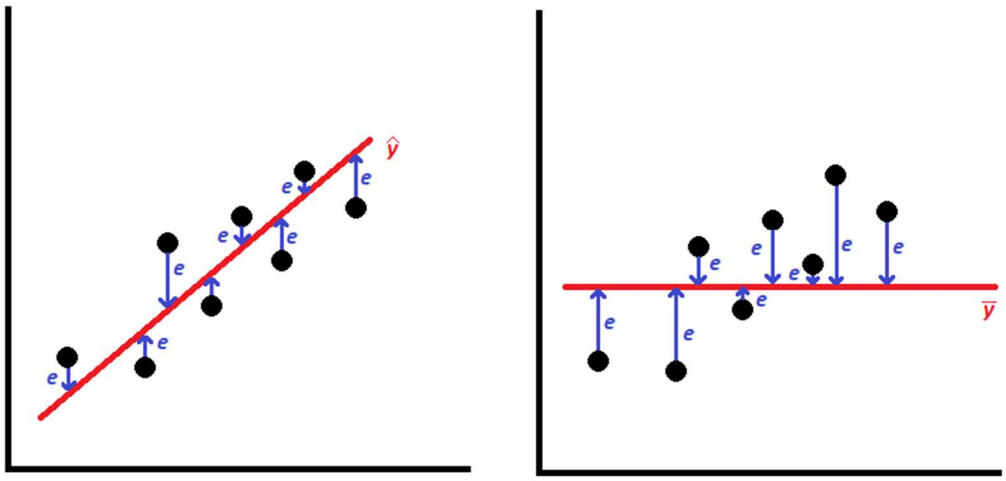
Treniranjem se dobivaju nepoznati koeficijenti m i b korištenjem gotovih formula dobivenih metodom najmanjih kvadrata:

$$m = \frac{\overline{x} \cdot \overline{y} - \overline{x \cdot y}}{(\overline{x})^2 - \overline{x^2}}$$

$$b = \overline{y} - m\overline{x}$$

Navedene jednadžbe se koriste srednjim vrijednostima podataka za treniranje prikazane kao $\overline{x}$ *i* $\overline{y}$.

Preciznost obrade algoritma će se određivati pomoću kvadratne pogreške (SE) i koeficijenta determinacije.



Slika 2.2: prikaz pogrešaka regresijske linije $\hat{y}$ i srednje vrijednosti $\overline{y}$

Formula računanja kvadratne pogreške:

$$SE = \sum_{i}^{n} (y_i - (mx_i + b))^2$$

Koeficijent determinacije (r^2) je mjera preciznosti regresijskog linije s obzirom na dane podatke. Dobiva se pomoću omjera kvadratne pogreške regresijske linije ($SE\ \hat{y}$) i kvadratne pogreške srednje vrijednosti podataka ($SE\ \overline{y}$).

$$r^2 = 1 - \frac{SE\ \hat{y}}{SE\ \overline{y}}$$

2.3 Razrada algoritma

Algoritam strogo prati svoju matematičku podlogu te se od knjižnica koriste: statistics i NumPy.

```
from statistics import mean
import numpy as np
```

Kod 2.1: uvoz knjižnica

2.3.1 Metoda fit() – Treniranje

```
def fit(self, xs, ys):
    numerator_m = (mean(xs) * mean(ys) - mean(xs * ys))
    denominator_m = (mean(xs) * mean(xs) - mean(xs*xs))
    self.m = numerator_m / denominator_m
    self.b = mean(ys) - self.m*mean(xs)
```

Kod 2.2: metoda fit

Metoda prima polja xs i ys, odnosno polje podataka za treniranje.

Prema formuli se računaju nagib m i presjek b koji se spremaju kao vrijednosti objekta za daljnje korištenje prilikom predviđanja. Brojnik i nazivnik (numerator_m i denominator_m) su raspodijeljeni radi preglednosti.

Za određivanje srednje vrijednosti koristi se metoda mean() iz knjižnice statistics.

2.3.2 Metoda predict() – Predviđanje

```
def predict(self, predict_x):
    return (self.m * predict_x) + self.b
```

Kod 2.3: metoda predict

Metoda predict() prima vrijednost predict_x čija se vrijednost želi predvidjeti. Predviđanje se vrši uvrštavanjem vrijednosti u linearnu jednadžbu čiji su koeficijenti dobiveni treniranjem modela.

2.3.3 Metoda score() – Testiranje

```
def squared_error(self, ys_orig, ys_line):
        return sum((ys_line - ys_orig)**2)

    #coefficient_of_determination
    def score(self, xs_test, ys_test):
        y_test_line = [self.predict(x) for x in xs_test]
        y_mean_line = [mean(ys_test) for y in ys_test]
        squared_error_regr = self.squared_error(ys_test, y_test_line)
        squared_error_y_mean = self.squared_error(ys_test, y_mean_line)
        return 1 - squared_error_regr/squared_error_y_mean
```

Kod 2.4: metoda score

Metodom score() se provjerava preciznost rada modela. Ona se računa kao koeficijent determinacije.

Metoda prima polje testnih podataka xs_test i ys_test. Prave se dvije nove liste y_test_line te y_mean_line koje predstavljaju predviđene vrijednosti i srednje vrijednosti. Nad listama se poziva metoda za računanje kvadratne pogreške squared_error() s kojom se dobiva suma razlike originalne vrijednosti te linijske vrijednosti na kvadrat.

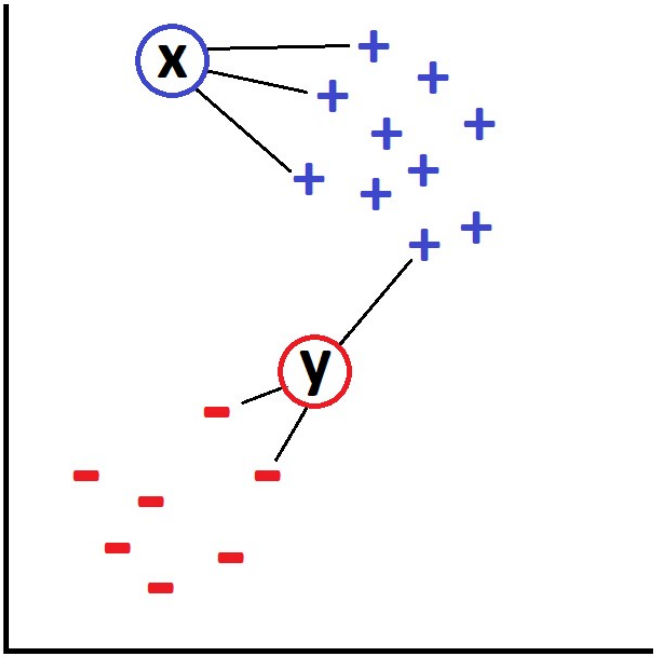
Score() naposljetku vraća preciznost algoritma nad skupom testnih podataka.

3. K-najbližih susjeda

3.1 O algoritmu

K-najbližih susjeda je vrsta nadziranog učenja koja se najčešće koristi za klasifikaciju. Osim klasifikacije uz preradu algoritma se može koristiti i za regresiju.

Algoritam za dani podatak traži koji su mu K najbližih susjeda te pregledava njihove kategorije. Ovisno o učestalosti kategorija susjeda, dani podatak pridružuje najučestalijoj kategoriji. Varijablu K je potrebno odabrati skladno sa podacima za treniranje zbog mogućnosti podijeljenih glasova kada se kategorija ne može odrediti.



Slika 3.1: Odabir kategorije za k=3

Algoritam pripada „lijenom učenju“ odnosno učenju bazirano na instanci. Posljedica toga je da ne postoji treniranje modela nego se klasifikacija odvija na razini jedne instance. Ovakav pristup omogućava laku izmjenu podataka za treniranje i brzo izvođenje, ovisno o količini primjera za treniranje te njihovoj dimenziji.

Performanse opadaju za jako višedimenzionalne podatke te jako velike skupove podataka jer se udaljenost mora izračunati za svaku točku posebno. Mane algoritma se mogu popraviti korištenjem višedretvenosti i restrikcijom odabira točaka koje su udaljene od zadane točke za određen radijus.

Algoritam je često korišten u sustavima preporuke. Ako je poznato što je kupac kupio ili pogledao, lagano se mogu preporučiti slični proizvodi.

3.2 Matematička podloga

Podloga iza algoritma K-najbližih susjeda je vrlo jednostavna. Za točku kojoj se želi odrediti kategorija određuje se euklidska udaljenost za sve ostale točke u prostoru.

$$d = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

U navedenoj jednadžbi n predstavlja dimenziju točke dok su q_i i p_i koordinate trenutne dimenzije.

Za dvodimenzionalni prostor kojemu je prva dimenzija x i druga y zadani se izraz može pokazati kao:

$$\sqrt{\sum_{i=1}^{2}(q_i - p_i)^2} = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2}$$

3.3 Razrada algoritma

Algoritam će predviđati kategoriju zadanog podatka na osnovu već spomenutih načela K-najbližih susjeda.

Koristit će se knjižnice NumPy i collections.

```
from collections import Counter
import numpy as np
```

Kod 3.1: uvoz knjižnica

3.3.1 Metoda fit() – Treniranje

```
def fit(self, data, k=5):
    self.k = k
    self.data = data
```

Kod 3.2: metoda fit

Treniranje kao takvo ne postoji u ovom algoritmu jer se predviđanje odvija nad instancom. Funkcija metode fit je interno spremanje varijable k koja predstavlja broj susjeda. Zadana vrijednost varijable k je 5.

Metoda prima rječnik data čiji su ključevi kategorije, a vrijednosti liste n-dimenzionalnih lista tj. točaka.

3.3.2 Metoda predict() – Predviđanje

```
def predict(self, predict_value):
    distances = []
    for group in self.data:
        for features in self.data[group]:
            difference = np.array(features)-np.array(predict_value)
            euclidean_distance = np.linalg.norm(difference)
            distances.append([euclidean_distance, group])
    votes = [i[1] for i in sorted(distances)[:self.k]]
    vote_result = Counter(votes).most_common(1)[0][0]
    return vote_result
```

Kod 3.3: metoda predict

Metoda predict() prima točku u n-dimenzionalnom prostoru čija se kategorija želi predvidjeti.

Računanje udaljenosti se vrši kroz dvije petlje. Prva iterira po kategorijama, odnosno ključevima rječnika. Druga for petlja iterira po točkama unutar prostora za treniranje.

Pomoću polja pravi se matrica razlike trenutno promatrane točke prostora za treniranje i točke koju se želi odrediti. Euklidska udaljenost računa se kao norma matrice pomoću metode linalg.norm() knjižnice NumPy. Računanje norme odnosno euklidske udaljenosti je matematički istovjetno. Izračunate udaljenosti se dodaju u listu distances kao lista udaljenosti i kategorije.

Varijabla votes je lista prvih k kategorija dobivene iz uzlazno sortirane liste udaljenosti. Pomoću metode Counter(votes).most_common(1) iz liste votes se uzima prva najučestalija oznaka kao lista tuplea od dva elementa (kategorija, učestalost).

Metoda vraća predviđenu kategoriju za danu ulaznu točku.

3.3.3 Metoda score() – Testiranje

```
def score(self, test_data):
    correct = 0
    count = 0
    for group in test_data:
        for features in test_data[group]:
            result = self.predict(features)
            if result == group:
                correct += 1
            count += 1
    return correct/count
```

Kod 3.4: metoda score

Metoda score prima rječnik testnih podataka čiji su ključevi oznake kategorije te vrijednosti točke u n-dimenzionalnom prostoru.

Iteracijom po grupama pa po točkama predviđa se kategorija trenutne točke i uspoređuje s istinitom kategorijom.

Metoda vraća preciznost kao omjer točnih i ukupnih rezultata.

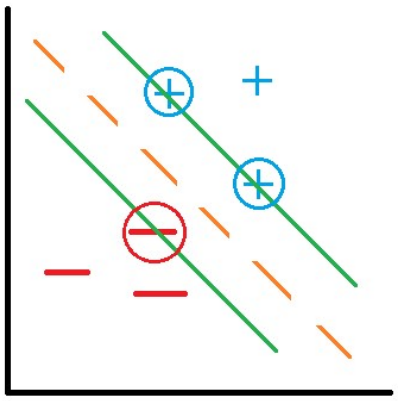
4. Stroj potpornih vektora – SVM

4.1 O algoritmu

Stroj potpornih vektora (eng. Support vector machine) je oblik nadziranog učenja. Nastao u 60-im godinama kao znanstveni rad Vladimira Vapnika, ali se u praksi koristi od ranih 90-ih godina kada je pokazano da SVM bolje prepoznaje ručno pisane brojeve od neuronskih mreža te time postaje jedan od najpopularnijih algoritama.

Algoritam je binarni klasifikator[3], no ovisno o implementaciji može se koristiti i kao regresor.

Cilj SVM je naći najbolju dijeleću hiperravninu tako da je udaljenost hiperravnine i podataka što veća. Kako bi se to postiglo uvode se dvije hiperravnine koje prolaze kroz ekstreme klasa koji se nazivaju potporni vektori. Klase se dijele u dvije oznake: 1 i -1. Često se koristi motivacija široke ulice gdje je želja ostvariti najširu moguću ulicu, tako da s jedne strane budu pozitivni, a s druge strane negativne vrijednosti.



Slika 4.1: princip rada SVM

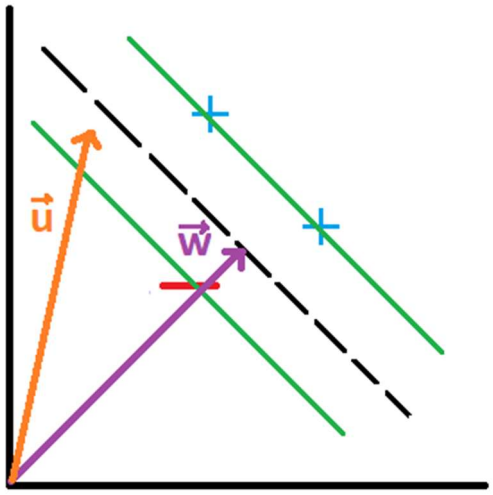
Na slici 4.1 potporni vektori su zaokruženi te su kroz njih povućene pozitivna i negativna hiperravnina. Isprekidana crta predstavlja srednju vrijednost pozitivne i negativne hiperravnine.

Proces treniranja SVM-a može potrajati ovisno o količini podataka za treniranje, no predviđanja i testiranja su značajno brža.

[3] U jednom trenutku može kategorizirati dvije klase

4.2 Matematička podloga

Kao što je navedeno u opisu algoritma, traži se najširа moguća „ulica“ među podacima.



Slika 4.2: prikaz vektora $\vec{u}$ i $\vec{w}$

Pojavljuje se vektor $\vec{u}$ kojem je potrebno odrediti klasu. Vektor $\vec{w}$ je vektor okomit na dijeleću hiperravninu. Želja je projicirati vektor $\vec{u}$ na vektor $\vec{w}$ kako bi dobili vektor s proporcionalnom vrijednošću na vektor $\vec{w}$. Tako se ovisno o duljini projekcije određuje pripada li novi uzorak desnoj ili lijevoj strani ulice.

$$\vec{w} \cdot \vec{u} \geq c$$

Gdje je c vrijednost iznad koje uzorak pripada pozitivnoj skupini. Ako je $c = -b$, dobiva se pravilo odlučivanja:

$\vec{w} \cdot \vec{u} + b \geq 0$ za pozitivnu klasu

Trenutno nije poznato koji vektor $\vec{w}$ niti koju konstantu b koristiti. Zbog toga je potrebno uvesti više uvjeta.

$$\vec{w} \cdot \overrightarrow{x_+} + b \geq 1$$
$$\vec{w} \cdot \overrightarrow{x_-} + b \leq -1$$

Uzorak je sigurno pozitivan ili negativan, ako se nalazi iznad ili ispod potpornih vektora. U uvjetima $\overrightarrow{x_+}$ i $\overrightarrow{x_-}$ predstavljaju pozitivan i negativan uzorak.

Također radi olakšanog računanja uvodi se dodatna varijabla y_i:

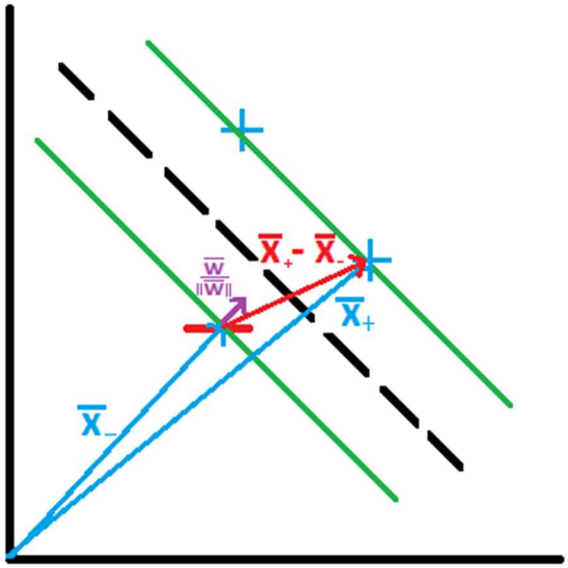
$$y_i = \begin{cases} +1, & za\ poz.uzorke \\ -1, & za\ neg.uzorke \end{cases}$$

Uvrsti li se y_i u uvjete proizlazi:

$$y_+(\vec{w} \cdot \overrightarrow{x_+} + b) \geq 1$$
$$y_-(\vec{w} \cdot \overrightarrow{x_-} + b) \geq 1$$

Uvjeti se sažimaju u jedan te se 1 prebacuje na lijevu stranu što daje:

$$y_i(\vec{w} \cdot \overrightarrow{x_i} + b) - 1 \geq 0$$



Slika 4.3: prikaz razlike vektora i jediničnog vektora

Kako bi se dobila širina između potpornih vektora, oduzima se jedan pozitivan i jedan negativan uzorak te skalarno množi s jediničnim vektorom okomitim na dijeleću hiperravninu.

$$širina = (\overrightarrow{x_+} - \overrightarrow{x_-}) \cdot \frac{\vec{w}}{||\vec{w}||}$$

Kada se uvjet izjednači s nulom, dobiva se izraz samih potpornih vektora tj. vektora kroz koji prolazi potporna hiperravnina.

$$y_i(\vec{w} \cdot \overrightarrow{x_i} + b) - 1 = 0$$
$$\vec{w} \cdot \overrightarrow{x_+} = 1 - b$$
$$\vec{w} \cdot \overrightarrow{x_-} = 1 + b$$

Uvrste li se dobiveni skalarni produkti za pozitivne i negativne uzorke u jednadžbu širine dobiva se izraz:

$$(\overrightarrow{x_+} - \overrightarrow{x_-}) \cdot \frac{\vec{w}}{||\vec{w}||} = \frac{2}{||\vec{w}||}$$

Iz toga slijedi da će širina ulice biti veća poveća li se $\frac{2}{||\vec{w}||}$ odnosno smanji $||\vec{w}||$ do najmanje točke. Radi lakšeg računanja smanjivat će se $\frac{1}{2}||\vec{w}||^2$ što je pri određivanju

minimuma istovjetno s $\left|\left|\vec{w}\right|\right|$. Trenutni cilj je pronaći ekstrem funkcije uz uvjet pa se zbog toga uvodi Lagrangeov multiplikator.

$$L = \frac{1}{2}\|\vec{w}\|^2 - \sum \alpha_i[y_i(\vec{w}\cdot\vec{x}_i + b) - 1]$$

Parcijalno se derivira L po $\vec{w}$ i izjednačava s nulom kako bi se došlo do izraza za $\vec{w}$.

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum \alpha_i y_i \vec{x}_i = 0$$

$$\vec{w} = \sum \alpha_i y_i \vec{x}_i$$

Također parcijalno se derivira L i po b.

$$\frac{\partial L}{\partial b} = -\Sigma \alpha_i y_i = 0$$

Izraz za $\vec{w}$ se uvrštava u Lagrangeovu funkciju što daje:

$$L = \sum \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j \, y_i y_j \vec{x}_i \cdot \vec{x}_j$$

Te se isto uvrsti u pravilo za odlučivanje:

$$\sum \alpha_i y_i \vec{x}_i \cdot \vec{u} + b \ \geq 0 \text{ za pozitivnu klasu}$$

Iz navedenih jednakosti vidljivo je da optimizacija hiperravnine ovisi samo o skalarnom umnošku vektora uzoraka u Lagrangreovoj funkciji te skalarnom umnošku uzorka i nepoznatog uzorka.

Matematički se može dokazati da optimizacija nikada neće zapeti u lokalnom minimumu tj. da je SVM konveksan optimizacijski problem.

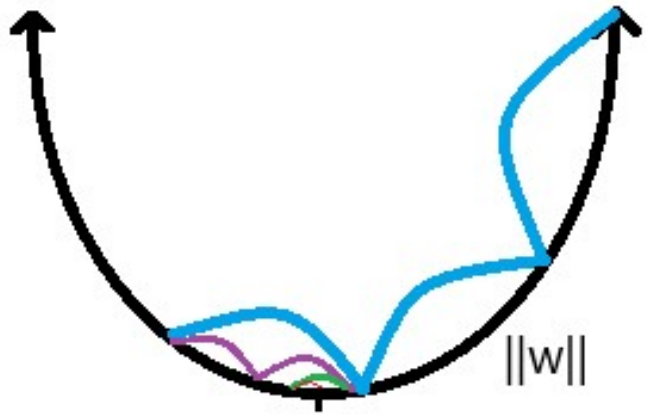
4.3 Razrada algoritma

SVM se svodi na rješavanje problema kvadratnog programiranja[4].

Ideja je implementirati:

$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1$$

tako da se $\|\vec{w}\|$ minimizira, a maksimizira b. Uzima se neki vektor $\vec{w}$ koji će služiti za optimizaciju. Cilj je smanjiti njegovu duljinu pritom pazeći kako je vektor orijentiran tj. odgovara li modelu.



Slika 4.4: proces optimizacije

Radi povećanja brzine izvođenja prvo se smanjuje vrijednost $\vec{w}$ u većim koracima dok ne pređe minimalnu točku, nakon čega se preciznije s manjim smanjivanjima $\|\vec{w}\|$ dovodi do minimuma. Ovakav pristup je moguć zbog konveksnosti problema.

Postoje gotove knjižnice koje obrađuju probleme optimizacije kao što su CVXOPT i LibSVM (koristi ga SciKit-learn), no kako bi se pokazao mehanizam SVM u nastavku se ne koriste knjižnice za optimiziranje.

U razradi se koristi jedino knjižnica NumPy.

```
import numpy as np
```

Kod 4.1: uvođenje knjižnica

[4] Vrsta matematičke optimizacije problema gdje se optimizira (minimizira ili maksimizira) kvadratna funkcija sa nekoliko promjenjivih varijabli uz postojanje linearnih ograničenja nad promjenjivim varijablama.

4.3.1 Metoda fit() – Treniranje

```
def fit(self, data):
        self.data = data
        opt_dict = {}
        transforms = [[1,1], [-1,1], [-1,-1], [1,-1]]

        all_data = []
        for yi in self.data:
            for featureset in self.data[yi]:
                for feature in featureset:
                    all_data.append(feature)
        self.max_feature_value = max(all_data)
        self.min_feature_value = min(all_data)
        all_data = None

        step_sizes = [self.max_feature_value * 0.1,
                      self.max_feature_value * 0.01,
                      self.max_feature_value * 0.001]
        b_range_multiple = 5
        b_multiple = 5
        latest_optimum = self.max_feature_value*10
```

Kod 4.2: definiranje varijabli

Metoda prima rječnik čiji su ključevi -1 i 1 te vrijednosti polja vektora za treniranje.

Priređuje se rječnik (opt_dict) koji će se puniti po ključu koji odgovara duljini vektora te za vrijednosti imati listu u kojoj se nalaze vektor i konstanta b { ||w||: [w, b] }

Transforms je lista po kojoj će se provjeravati orijentiranost vektora.

U sljedećem bloku određuje se minimalna te maksimalna vrijednost uzorka koji se kasnije koriste za skaliranje koraka.

Step_sizes je lista koraka kroz koje će se iterirati. B_range_multiple te b_multiple se koristi radi povećanja brzine izvođenja. S njima se skaliraju koraci za konstantu b jer njena preciznost ne utječe značajno na algoritam.

Latest_optimum predstavlja vrijednosti vektora $\vec{w}$. Iako u stvarnoj primjeni ne mora biti slučaj da sve vrijednosti matrice vektora jednake, ovime se štedi na vremenu izvođenja bez značajnih gubitaka preciznosti.

```
#nastavak koda 4.2
for step in step_sizes:
        w = np.array([latest_optimum, latest_optimum])
        optimized = False
        while not optimized:
            for b in np.arange(-1*(self.max_feature_value*b_range_multiple),
                               self.max_feature_value*b_range_multiple,
                               step*b_multiple):
                for transformation in transforms:
                    w_t = w*transformation
                    found_option = True
                    for yi in self.data:
                        for xi in self.data[yi]:
                            if not yi*(np.dot(w_t,xi)+b) >= 1:
                                found_option = False
                    if found_option:
                        opt_dict[np.linalg.norm(w_t)] = [w_t,b]

            if w[0] < 0:
                optimized = True
            else:
                w = w - step

        norms = sorted([n for n in opt_dict])
        opt_choice = opt_dict[norms[0]]
        self.w = opt_choice[0]
        self.b = opt_choice[1]
        latest_optimum = opt_choice[0][0]+step*2
```

Kod 4.3: određivanje optimalnih vrijednosti

Iterira se po koracima oduzimanja vektora $\vec{w}$ sve dok se ne dođe do minimalne vrijednosti duljine za taj korak.

Za svaki korak konstante b određuje se vektor $\vec{w}$ s odgovarajućom usmjerenošću. Ispituje se pravilo za odlučivanje za trenutni $\vec{w}$ i b te ukoliko odgovara vrijednost se upisuje u rječnik. To se ponavlja sve dok se vrijednosti vektora $\vec{w}$ ne spuste ispod nule.

Nakon prolaska kroz sve korake iteracije, sortira se rječnik te uzima najmanja duljina $\vec{w}$ kao optimalan izbor za daljnje iteriranje po sljedećem koraku, odnosno kraj treniranja.

4.3.2 Metoda predict() – Predviđanje

Nakon što je SVM istreniran, metodom predict predviđa se klasa novoga uzorka.

```
def predict(self,features):
    classification = np.sign(np.dot(np.array(features),self.w)+self.b)
    return classification
```

Kod 4.4: metoda predict

Predict prima vektor koji je potrebno klasificirati te vraća oznaku klase.

Predviđanje se radi uvrštavanjem novog vektora u jednadžbu:

$$classification = sgn(\vec{w} \cdot \vec{x_i} + b)$$

4.3.3 Metoda score() – Testiranje

Preciznost istreniranog SVM provjerava se pomoću metode za testiranje.

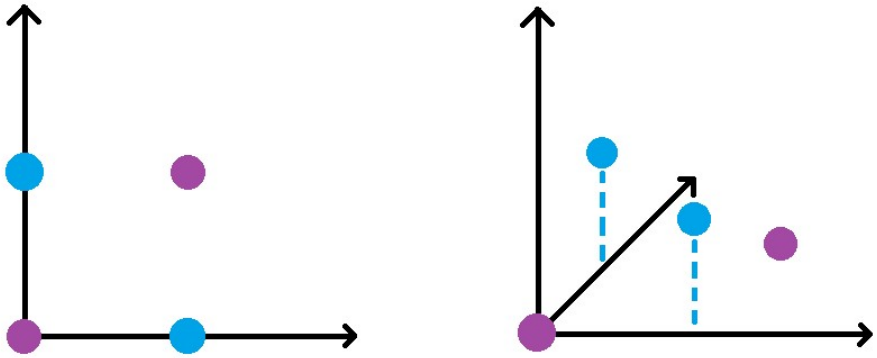
```
def score(self, data):
        self.data_test = data
        count = 0
        correct = 0

        for yi in self.data_test:
            for featureset in self.data_test[yi]:
                count += 1
                if(self.predict(featureset) == yi):
                    correct += 1

        accuracy = correct/count
        return accuracy
```

Kod 4.5: metoda za testiranje

Metoda prima rječnik čiji su ključevi 1 i -1, a vrijednosti polja vektora te vraća vrijednost točnih kroz ukupni broj testova. Za provjeru točnosti uspoređuje se predviđena vrijednost metodom predict i poznata klasa uzorka.

4.4 Jezgrene funkcije (Kernel functions)

Do sada se pretpostavljalo da su uzorci samo linearno razdjeljivi podaci. Problem nelinearno razdjeljivih podataka je moguće riješiti transformacijom u viši prostor.



Slika 4.5: transformacija u viši prostor

Čar SVM-a leži u pokazanoj ovisnosti o skalarnom produktu dva uzorka. Nije potrebno raditi transformaciju cijeloga prostora nego samo naći maksimum transformacije skalarnog umnožaka uzoraka: $\Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$ odnosno $\Phi(\vec{x}_i) \cdot \Phi(\vec{u})$.

Zbog toga se uvode jezgrene funkcije (eng. Kernel functions) kao funkcije koje vraćaju skalarni produkt vektor u drugom prostoru.

$$K(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$$

Jezgrene funkcije smanjuju kompleksnost algoritma jer je potrebno znati samo funkciju, a ne i transformaciju.

Neki od najčešćih korištenih funkcija su:

$K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j)^2$; Polinomijalna jezgrena funkcija

$K(\vec{x}_i, \vec{x}_j) = e^{-\frac{-1}{2\sigma^2}(\vec{x}_i - \vec{x}_j)^2}$; Gaussova jezgrena funkcija

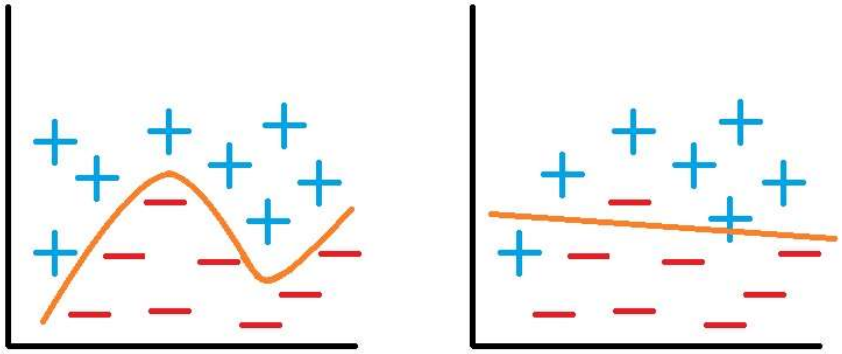
$K(\vec{x}_i, \vec{x}_j) = e^{-\gamma(\vec{x}_i - \vec{x}_j)^2}$; Radijalna osnovna funkcija

$K(\vec{x}_i, \vec{x}_j) = \tanh(\eta \vec{x}_i \cdot \vec{x}_j + \nu)$; Sigmoidna jezgrena funkcija

Bilo koja funkcija koja zadovoljava Mercerovom teoremu može biti jezgrena funkcija. Prema Mercerovom teoremu, ako je jezgrena matrica (Gram matrica) pozitivno semidefinitna za svaki skup primjera za učenja, onda je moguće funkciju rastaviti na skalarni produkt vektora.

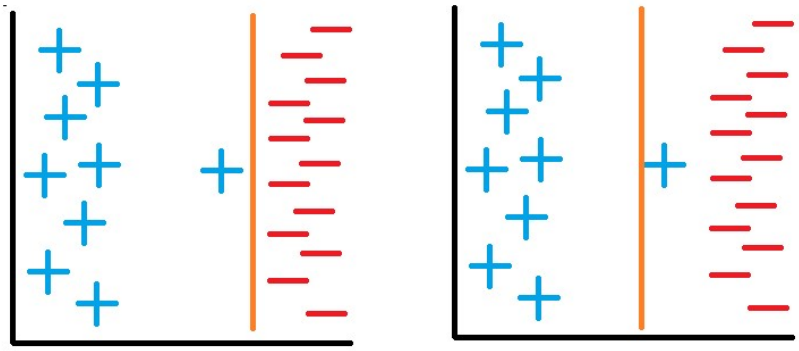
4.5 Stroj potpornih vektora s mekim marginama

SVM s mekim marginama uvodi se kao rješenje problema optimizacije modela dopuštajući greške prilikom određivanja hiperravnine. Problemi mogu nastati korištenjem jezgrenih funkcija koje dovode do prenaučenosti[5] ili samim skupovima podataka koji sadrže vrijednosti koje odskaču.



Slika 4.6: Rješavanje problema prenaučenosti

Ukoliko se na priloženom nelinearno razdjeljivom skupu podataka (slika 4.6) koristi radijalno osnovna jezgrena funkcija model se dovodi do problema prenaučenosti. Većina vektora uzoraka služe kao potporni vektori. Uvede li se linearan model mekih margina smanjuje se omjer potpornih vektora nad ukupnim brojem vektora, odnosno prenaučenost modela.

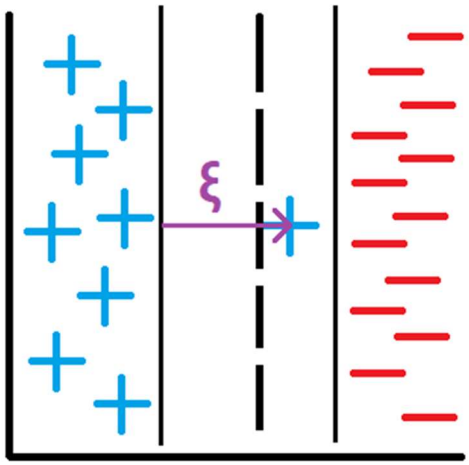


Slika 4.7: Rješavanje problema vrijednosti kojih odskaču

[5] Pojava u kojoj se model prilagođava šumu te time kvari preciznost modela

Pojava vrijednosti kojih odskaču također može našteti preciznosti rada algoritma. Ukoliko se dopusti pogreška prilikom određivanja hiperravnine dobiva se model čija je udaljenost između klasa veća nego korištenjem strogih margina (slika 4.7).Time se u teoriji dobiva optimalniji SVM.

Pogreške prilikom određivanja hiperravnine je potrebno kažnjavati. Zbog toga se uvodi nova varijabla ξ (rezervna varijabla) koja pripada svakom uzorku. Ukoliko se uzorak nalazi sa dobre strane granice $\xi = 0$. Ako se uzorak nalazi između margina rezervna varijabla poprima vrijednost $0 < \xi \leq 1$. Konačno, nalazi li se uzorak na pogrešnoj strani granice bit će kažnjen s $\xi > 1$.



Slika 4.8: Prikaz rezervne varijable

Uvođenjem kažnjavanja potrebno je preformulirati pravilo odlučivanja:

$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 - \xi_i$$

Time se dozvoljava da lijeva strana uvjetne jednadžbe bude manja od jedinice.

Potrebno je maksimizirati granice novog pravila odlučivanja, odnosno minimizirati izraz:

$$\frac{1}{2} \left|\left|\vec{w}\right|\right|^2 + C \sum_i \xi_i$$

Parametar C određuje kompromis između margina i ukupne kazne. Veći c dovodi do većeg kažnjavanja pogrešne klasifikacije, što će za posljedicu imati složenije modele.

5. K-srednje vrijednosti

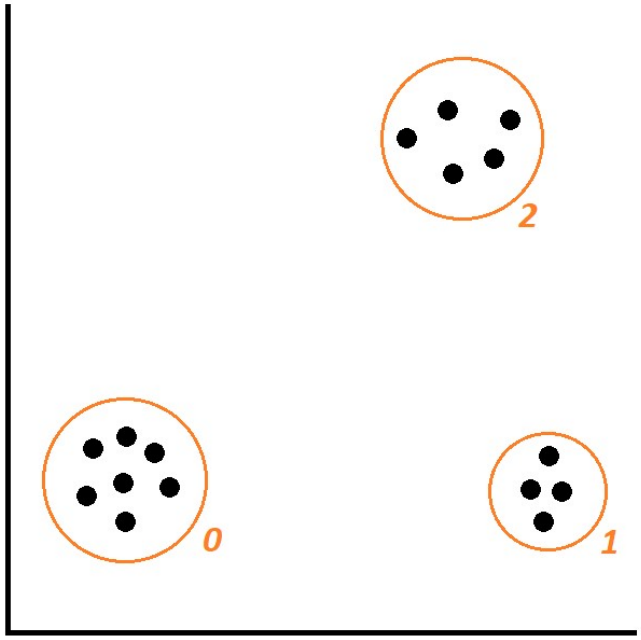
5.1 O algoritmu

Algoritam K-srednjih vrijednosti pripada grupi nenadziranog učenja, preciznije algoritmima grupiranja.

Cilj algoritma je odrediti K grupa unutar neoznačenog skupa podataka odnosno prema značajkama podataka pronaći prirodnu sličnost podataka te ih grupirati u K grupa.

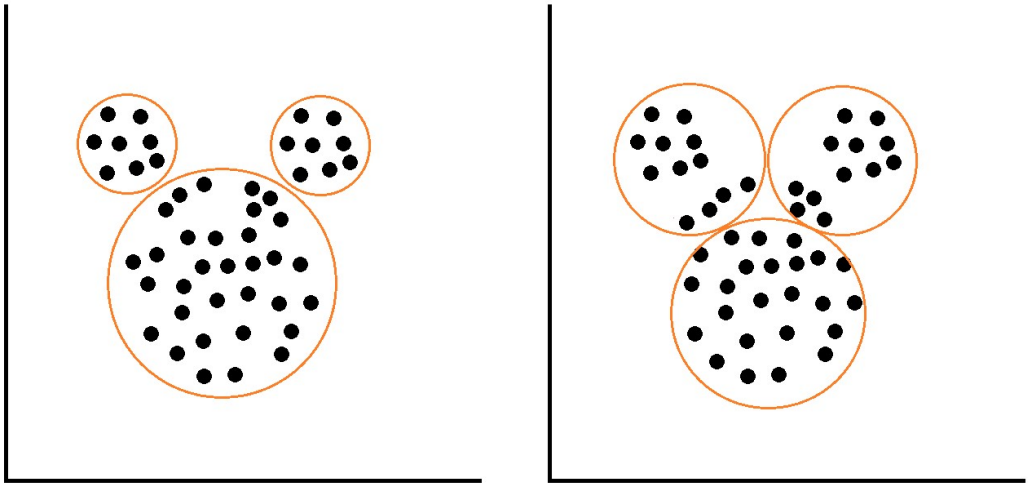
Grupe se određuju nasumičnim odabirom K točaka koje služe kao središta početnih grupa. Za svaki podatak unutar podataka za treniranje se na osnovu udaljenosti izabire najbliži središnji podatak tj. centroid. Vrijednost K je unaprijed određen broj od strane korisnika.

Nakon grupiranja računa se novi centroid kao srednja vrijednost svih točaka pojedine grupe. Ovaj postupak se ponavlja sve dok pozicija centroida ne bude nepromijenjena.



Slika 5.1: prikaz grupiranja pomoću algoritma za k = 3

K-srednjih vrijednosti je jednostavan i brz algoritam grupiranja. Najbolje se koristi ukoliko je broj grupa vidljiv iz definicije podataka.



Slika 5.2: grupe u stvarnosti i grupe dobivena K-srednjim vrijednostima

Mana algoritam je što zbog oslanjanja na udaljenost točaka od središta često grupira podatke u grupe sličnih veličina iako prirodno nije tako.

Iako cilj algoritma nije regresija niti klasifikacija nego grupiranje, nad istreniranim se modelom može predviđati kojoj bi grupi novi podatak mogao pripadati. Ukoliko su neki od podataka označeni, neoznačeni elementi grupe se mogu označiti pomoću označenih. Ovakav način korištenja modela k-srednjih vrijednosti naziva se polunadziranim učenjem.

5.2 Matematička podloga

Rješavanje problema započinje nasumičnim odabirom K podataka iz skupa podataka za treniranje veličine N koji će služiti kao centroidi pojedine grupe μ_k.
Računanjem udaljenosti ostalih podataka od centroida dobiva se određena pogreška koju je potrebno minimizirati. Pogreška je izražena ciljnom funkcijom[6]:

$$J = \sum_{i=1}^{N}\sum_{k=1}^{K} b_{ik}||x_i - \mu_k||^2$$

Ciljna funkcija je jednaka sumi kvadrata euklidskih udaljenosti $||x_i - \mu_k||^2$, odnosno zbroju kvadratnih pogrešaka.
Kako bi se odredila pripadnost podatka x_i grupi k uvedena je funkcija b_{ik} kao:

$$b_{ik} = \begin{cases} 1, & za\ k\ =\ argmin\_j\ ||x_i - \mu_j|| \\ 0, & inače \end{cases}$$

Ukoliko je euklidska udaljenost minimalna za j-ti centroid tj. k-ti centroid ciljne funkcije, kvadratna pogreška se množi s 1. Ukoliko podatak x_i ne pripada grupi k, množi se s 0. Time se određuje pripadnost podatka x_i pojedinoj grupi.
Nakon što su svi podaci pridruženi potrebno je odabrati novi centroid čiji ćemo izraz dobiti minimizacijom kriterijske funkcije J

$$\frac{dJ}{d\mu_k} = 2\sum_{i=1}^{N} b_{ik}(x_i - \mu_k) = 0$$

što daje

$$\mu_k = \frac{\sum_{i=1}^{N} b_{ik}x_i}{\sum_{i=1}^{N} b_{ik}}$$

Tako se dobiva da je vrijednost centroida μ_k jednaka srednjoj vrijednosti podataka unutar grupe k.
Ovaj postupak se ponavlja dok μ_k ne postane nepromijenjen tj. stacionaran.

[6] Funkcija koju je želja maksimizirati ili minimizirati

5.3 Razrada algoritma

Algoritam prati svoju matematičku podlogu te se od knjižnica uvodi NumPy .

```
import numpy as np
```

Kod 5.1: uvoz knjižnica

Konstruktor objekta modela prima broj grupa: k (predefinirano 2), toleranciju pomaka centroida: tol (predefinirano 0.0001) i maksimalan broj iteracija: max_iter (predefinirano 200) koji će prekinuti program upadne li u beskonačnu petlju prilikom potrage centroida.

```
class KMeans:
    def __init__(self, k = 2, tol = 0.0001, max_iter = 300):
        self.k = k
        self.tol = tol
        self.max_iter = max_iter
```

Kod 5.2: konstruktor objekta KMeans

5.3.1 Metoda fit() – Treniranje

```
def fit(self,data):
        self.centroids = {}
        for i in range(self.k):
            self.centroids[i] = data[i]

        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.k):
                self.classifications[i] = []

            for featureset in data:
                distances = [np.linalg.norm(featureset-self.centroids[centroid])
                             for centroid in self.centroids]
                classification = distances.index(min(distances))
                self.classifications[classification].append(featureset)

            prev_centroids = dict(self.centroids)
            for classification in self.classifications:
                self.centroids[classification] = np.average(
                                                 self.classifications[classification], axis=0)
```

Kod 5.3: prvi dio metode fit

Metoda fit prima polje točaka kao podatke za treniranje.

Definira se rječnik centroids čiji su ključevi nazivi grupa (od 0 do k) te vrijednosti prvih k točaka iz podataka za treniranje koje će služiti kao početni centroidi.

Otvara se for petlja za dozvoljen broj iteracija max_iter kako prilikom optimizacije algoritam ne bi upao u beskonačnu petlju.

Unutar petlje definira se rječnik classifications koja za ključeve ima nazive grupa te za vrijednosti liste točaka koje pripadaju toj grupi.

Iteracijom kroz podatke za treniranje za svaku točku mjeri se udaljenost od centroida te se sve udaljenosti spremaju u listu distances. Kako su udaljenosti od centroida spremane redom po nazivima grupa (od 0 do k) traženjem indeksa najmanje duljine dobiva se grupa kojoj podatak pripada.

Nakon pronalazaka grupa, trenutni centroidi se spremaju u rječnik prev_centroids te se traže novi centroidi kao srednje vrijednosti članova grupe.

```
#nastavak koda 5.3
        optimized = True
        for c in self.centroids:
            original_centroid = prev_centroids[c]
            current_centroid = self.centroids[c]
            if  self.tol < np.sum(
                (current_centroid-original_centroid)/original_centroid) :
                optimized = False
        if optimized:
            break
```

Kod 5.4: drugi dio metode fit

Za svaki centroid provjerava se promjena s obzirom na prethodni centroid grupe. Ukoliko se promjena ne nalazi u granici tolerancije, zastavica optimized se postavlja u False te se optimiziranje nastavlja. Ako se promjena centroide nalazi u granici tolerancije, zastavica se ne mijenja te se treniranje zaustavlja.

5.3.2 Metoda predict() – Predviđanje

```
def predict(self,data):
    distances = [np.linalg.norm(data-self.centroids[centroid])
            for centroid in self.centroids]
    classification = distances.index(min(distances))
    return classification
```

Kod 5.5: metoda predict

Metoda predict prima podatak data kojemu se želi predvidjeti grupa.

U listu distances spremaju se udaljenosti od podatka do centroida. Kako su centroidi poredani po vrijednostima grupa potrebno je pronaći samo indeks minimalne udaljenosti koji se sprema u classification.

Predict vraća predviđenu grupu podatka dana.

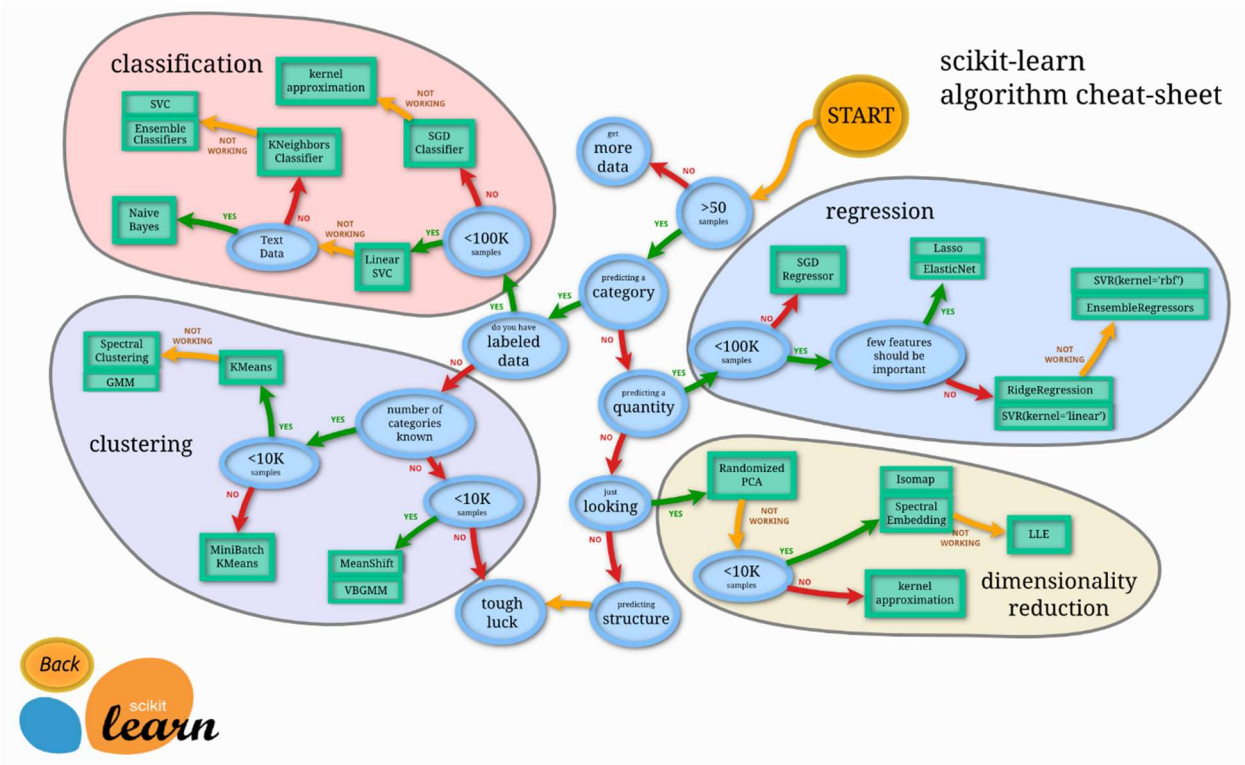
6. Zaključak

U radu su obrađena četiri važnija algoritma između mnoštva algoritama strojnog učenja te se postavlja pitanje kako odabrati najbolji.

Odabir algoritma ponajviše ovisi o potrebi. Ukoliko je potrebno predviđanje vrijednosti kontinuiranih podataka, koristit će se regresija. Ako je želja na osnovu nominalnih podataka predvidjeti kategoriju novog podatka, koristit će se klasifikacija. Za podatke čije kategorije ne poznajemo, može se poslužiti algoritmima nenadziranog učenja koji će sami grupirati podatke s obzirom na značajke podataka.

Naravno, korištenje pravog tipa algoritma za rješavanje je jedna odluka. Druga je odrediti koji algoritam tj. model najbolje odgovara skupu podataka kojim se rukuje.

Moguće je da kompleksniji model da lošije rezultate od jednostavnijeg modela. Ponekad je potrebno iste podatke provesti kroz različite modele kako bi se odabrao najbolje odgovarajući. Dobra referenca za odabir modela je scikit-learnov šalabahter algoritama (slika 6.1).



Slika 6.1: scikit-learnov šalabahter algoritama

To dovodi do zaključka da ne postoji jedan najbolji algoritam koji će riješiti svaki problem koji se nađe pred korisnikom nego je potrebno poznavati princip rada algoritma kako bi ga se moglo primijeniti nad problemom.

7. Literatura

[1] Jan Šnajder, Bojana Dalbelo Bašić: Strojno učenje; Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2014

[2] Andriy Burkov: The hundred-page machine learning book; 2019

[3] Andreas C. Müller, Sarah Guido: Introduction to Machine Learning with Python; O'Reilly, 2017

[4] Artificial Intelligence kolegij na MIT; MIT opencourseware, 16.4.2019

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/

[5] Practical Machine Learning tutorial with Python; pythonprogramming.net, 16.4.2019

https://pythonprogramming.net/machine-learning-tutorial-python-introduction/

8. Sažetak

Rad obrađuje četiri algoritma strojnog učenja s izvornim kodovima glavnih metoda (treniranje, predviđanje i testiranje) rađene u Pythonu i matematičkim ishodima algoritama.

Cjelina: Jednostavna linearna regresija

Algoritam nadziranog učenja korišten za predviđanje vrijednosti s obzirom na pružene kontinuirane podatke. Uz različite implementacije može se koristiti i za klasifikaciju.

Temelji se na linearnom modelu najbolje odgovarajuće linije nad skupom podataka.

Cjelina: K-najbližih susjeda

Algoritam nadziranog učenja korišten za predviđanje kategorije nad nominalnim skupom podataka.

Temelji se na određivanju euklidske udaljenosti podatka od podataka za treniranje. Kategorija se određuje s obzirom na najučestaliju kategoriju od K najbližih podatka.

Cjelina: stroj potpornih vektora

Algoritam nadzirnog učenja korišten za klasifikaciju kao binarni klasifikator (vrši klasifikaciju za dvije grupe). Moguća je i regresija uz drugačiju implementaciju algoritma.

Temelji se na problemu optimizacije najvećeg mogućeg prostora između podataka. Često se koristi pojam dijeljenja podataka prema „najširoj ulici“ gdje je pojedina kategorija na jednoj strani ulice.

Omogućava efikasno transformiranje nelinearno razdjeljivih podataka u viši prostor korištenjem jezgrenih funkcija.

Cjelina: K-srednje vrijednosti

Algoritam nenadziranog učenja korišten za grupiranje neoznačenih podataka.

Temelji se na podijeli podataka za treniranje s obzirom na najmanju udaljenost od centroida grupa nakon čega se novi centroidi računaju kao srednja vrijednost podataka svake trenutne grupe. Proces se ponavlja dok centroidi ne postanu nepromijenjeni.