Sadržaj

1. UVOD

Strojno učenje (Machine Learning) je metoda u analizi podataka koja automatizira izgradnju analitičkih modela. Korištenje algoritama koji mogu iterativno učiti iz podataka računalima omogućava uvid u njihovu strukturu bez eksplicitnog programiranja. (https://www.sas.com/en_us/insights/analytics/machine-learning.html)

U današnjem svijetu postoji široka primjena strojnog učenja: počevši od robotike i medicine, preko marketinga i društvenih mreža pa sve do otkrivanja prevara i neželjene pošte – gotovo da i ne postoji područje ljudske djelatnosti koje na neki način ne bi moglo iskoristiti spoznaje dobivene ovakvom analizom podataka.
Bez obzira na primjenjivost i činjenicu da se prosječan stanovnik Zapada nekoliko puta dnevno nesvjesno susretne s nekim aspektom strojnoga učenja, ono još uvijek predstavlja svojevrsnu enigmu. Na koji to način računalo zapravo "uči"? Što mu omogućava rad s velikim količinama podataka i koji se algoritmi pri tome koriste? Kako se program prilagođava novim podatcima?

Razlikujemo četiri osnovna koncepta strojnog učenja:

- Supervised learning – nadzirano učenje
- Unsupervised learning – nenadzirano učenje
- Semisupervised learning – polunadzirano učenje
- Reinforcement learning – učenje pojačavanjem

Kod nadziranog učenja, alogritam uči na skupu podataka kod kojega su osim značajki(features) poznate i sve tražene vrijednosti. Učenje se sastoji u traženju poveznice između svake značajke i

traženih vrijednosti. Nakon toga algoritam je sposoban pridjeliti traženu vrijednost nekom novom podatku koji nije u početnom skupu. Ovisno o tipu tražene vrijednosti razlikujemo regresijske i klasifikacijske algoritme. Kod klasifikacijskh algoritama tražena vrijednost poprima jednu od vrijednosti iz ograničenog skupa, dok je kod regresijskih algoritama broj vrijednosti koju tražena vrijednost može poprimiti praktički neograničen.

S druge strane, algoritmima nenadziranog učenja nisu poznate nikakve tražene vrijednosti pa se njihov rad uglavnom svodi na traženje pravilnosti u skupu podataka.

Polunadzirano učenje je baš to – polunadzirano, što znači da su algoritmi polunadziranog učenja spoj algoritama nadziranog i nenadziranog učenja. Pokušavaju ustanoviti na koji su način povezani parametri i tražene vrijednosti, ali koriste skupove podataka kod kojih je samo za manji dio poznata tražena vrijednost.

Uzmimo primjer neželjene e-pošte. Htjeli bismo za svaku sljedeću poruku znati je li spam. Algoritmu nadziranog učenja proslijedili bismo tisuće poruka od kojih bi svaka bila označena kao spam ili ne-spam. Algoritmu nenadziranog učenja samo bismo proslijedili tisuće poruka, ali stvari bi mogle "poći po zlu" – algoritam bi mogao uočiti neku pravilnost koja nije nužno naša tražena podjela pa bismo tome pokušali doskočiti algoritmom polunadziranog učenja. Njemu bismo pak proslijedili tisuće poruka od kojih bi par stotina bilo označeno kao spam ili ne-spam (recimo da

nismo u mogućnosti označiti baš svaku od stotinjak tisuća ili više poruka) kako bismo postigli rezultate nadziranog učenja.

Učenje pojačavanjem ponajviše se koristi u robotici. Algoritmi metodom pokušaja i pogreške pokušavaju ustanoviti kojim će postupcima doći do najboljeg ishoda, a sastoje se od tri osnovne komponente: agenta – onoga koji uči i donosi odluke, okoline – svega sa čime agent ulazi u interakciju i akcija – onoga što agent može učiniti i zbog čega u konačnici biva nagrađen ili kažnjen. Cilj je maksimizirati nagradu u određenom vremenskom intervalu, na primjer: doći od točke A do točke B u najkraćem vremenu i uz najmanji broj padova.

Kroz ovaj rad osvrnut ću se na rezultate rada nekoliko najpoznatijih klasifikacijskih algoritama nadziranog učenja i to nad dvama skupovima podataka – jednom jednostavnijem i jednom složenijem. U prvom slučaju radit će se o binarnoj, a u drugom o višeklasnoj klasifikaciji.

2. KLASIFIKACIJA

Pod pojmom klasifikacije općenito podrazumijevamo podjelu živih bića ili predmeta u klase na osnovu utvrđenog kriterija. U strojnom učenju, problem klasifikacije svodi se na utvrđivanje pripadnosti novog podatka jednoj od već postojećih skupina. Algoritam uči na podatcima čija je pripadnost skupini unaprijed poznata. Razlikujemo binarnu i višeklasnu klasifikaciju.

Binarna klasifikacija je klasifikacija kod koje podatke svrstavamo u dvije skupine. Primjerice, možemo na osnovu promjera tumora raditi procjenu radi li se o malignom ili benignom tumoru. Spomenuti primjer s neželjenom poštom je također primjer binarne klasifikacije.

Kod višeklasne klasifikacije postoje tri ili više skupina u koje algoritam pokušava svrstati podatke. Višeklasna klasifikacija najčešće se svodi na izvođenje nekoliko binarnih koje se odvijaju između jedne odabrane, promatrane klase i svih ostalih klasa, koje predstavljaju drugu klasu binarne klasifikacije. Višeklasnu klasifikaciju mogli bismo koristiti kod određivanja krvne grupe. Znamo da postoje četiri krvne grupe 0, A, B i AB. Postupak bi bio sljedeći: prvo bismo krvnu grupu 0 pokušali odijeliti od ostalih binarnom klasifikacijom, gdje bi recimo 0 predstavljala pozitivne primjerke, a A, B i AB negativne, tj. dijelimo 0 od ne-0. U sljedećoj koraku uzeli bismo krvnu grupu A i na isti bismo ju način odijelili od ostalih i postupak bismo ponovili za krvne grupe B i AB. (slike postupka klasifikacije)

U uvodu je spomenuto da osim klasifikacijskih postoje i regresijski algoritmi koji se od klasifikacijskih ponajviše razlikuju u broju vrijednosti koje može poprimiti tražena vrijednost – kod regresijskih algoritama taj se broj smatra gotovo neograničenim. Iako bi se teoretski za veliki broj klasa, odnosno traženih vrijednosti moglo govoriti i o klasifikaciji i o regresiji, u praksi se najčešće koristi onaj algoritam koji daje najbolje rješenje.

3. TRANSFORMACIJA I VIZUALIZACIJA SKUPA PODATAKA

3.1 O skupovima

Prvi korak kod svakog projekta vezanog za strojno učenje je naravno odabir skupa podataka nad kojim ćemo ga primijeniti. U ovom radu prikazat će se dva skupa: složeniji skup – otto_group.csv i jednostavniji mashrooms.csv.
Otto_group.csv sadrži numeričke podatke o proizvodima koje je potrebno klasificirati. Kako bi se zaštitile poslovne tajne, nije nam dan uvid u to što svaki podatak predstavlja, već za svaki proizvod postoji 91 numerička karakteristika označena jednostavno s feature_1, feature_2 i tako dalje. Proizvode je potrebno svrstati u devet različitih kategorija za koje nam također nije rečeno što predstavljaju.
Skup mushrooms.csv sadrži karakteristike gljiva i potrebno ih je klasificirati kao jestive ili otrovne. Klasifikacija je binarna, a značajke (features) ne samo da su nam poznati, već su i svima razumljivi, što ovaj skup čini izrazito jednostavnim. Također, kada se malo razmisli o navedenim problemima, vidi se da bi klasifikacija gljiva trebala biti efikasna. Kada na temelju određenih karakteristika gljive ne bi bilo moguće odrediti je li ona otrovna ili nije, kako bi to onda mogli činiti ljudi? S druge strane, za neke proizvode nije uvijek jasno u koju kategoriju ih je potrebno svrstati – dva proizvoda mogu biti iste boje i veličine, mogu čak biti izrađeni od istog materijala, a mogu imati potpuno različite funkcionalnosti pa bi ih prema tome trebalo svrstati u različite kategorije. Iz gore navedenog možemo zaključiti da će sa skupom mashrooms.csv biti lakše raditi i da će algoritmi primijenjeni na taj skup davati bolje rezultate. Ovakva intuicija ponekad je važna da bi se na ispravan način nastavilo s projektom. U sljedećim poglavljima prikazat će se osnove klasifikacijskih algoritama na skupu mushrooms.csv, a kasnije će se obraditi skup otto_group.csv.

3.2 Općenito o transformaciji i vizualizaciji

Prije nego što na bilo koji skup primijenimo algoritme strojnog učenja, potrebno ga je proučiti i nerijetko napraviti određene transformacije. Transformacija podataka je nužna ako svi podaci s kojima raspolažemo nisu numerički. S obzirom da svi algoritmi koje ćemo u ovom radu prikazati rade isključivo s numeričkim podacima nužno je sve nenumeričke podatke pretvoriti u numeričke. Složenije transformacije podrazumijevaju i promjenu broja parametara.
Ako je broj parametara prevelik, u smislu da pojedini parametri značajno ne pridonose efikasnosti algoritma, ali ga usporavaju, onda ima smisla pokušati smanjiti njihov broj. Jedna od mogućnosti je parametre koji su usko povezani zamijeniti jednim parametrom. Ako smatramo da broj parametara nije dovoljno velik, onda je moguće na umjetan način povećati njihov broj. Novi parametar će najčešće biti nekakva funkcija početnih parametara. Kod jednostavnijih algoritama povećanje parametara na ovakav način bi moglo značajno poboljšati rezultate, ali kod najmodernijih algoritama to je sve rjeđe slučaj. U ovom radu prikazat ćemo samo jednostavnije transformacije.

3.3 Transformacija i vizualizacija koristeći programski jezik Python

Nakon što učitamo podatke koristeći metodu Data_Frame.read_csv, korisno je pozvati metodu Data_Frame.head() koja prikaže prvih 5 podataka u skupu.

```
df = pd.read_csv('C:\\Users\\viktor\\Downloads\\mushroom-classification\\mushrooms.csv')
print(df.head())
```

```
  class cap-shape cap-surface cap-color bruises odor gill-attachment  \
0     p         x           s         n       t    p               f
1     e         x           s         y       t    a               f
2     e         b           s         w       t    l               f
3     p         x           y         w       t    p               f
4     e         x           s         g       f    n               f

  gill-spacing gill-size gill-color   ...   stalk-surface-below-ring  \
0            c         n          k   ...                          s
1            c         b          k   ...                          s
2            c         b          n   ...                          s
3            c         n          n   ...                          s
4            w         b          k   ...                          s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-color  \
0                      w                      w         p          w
1                      w                      w         p          w
2                      w                      w         p          w
3                      w                      w         p          w
4                      w                      w         p          w

  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

Slika 3.1

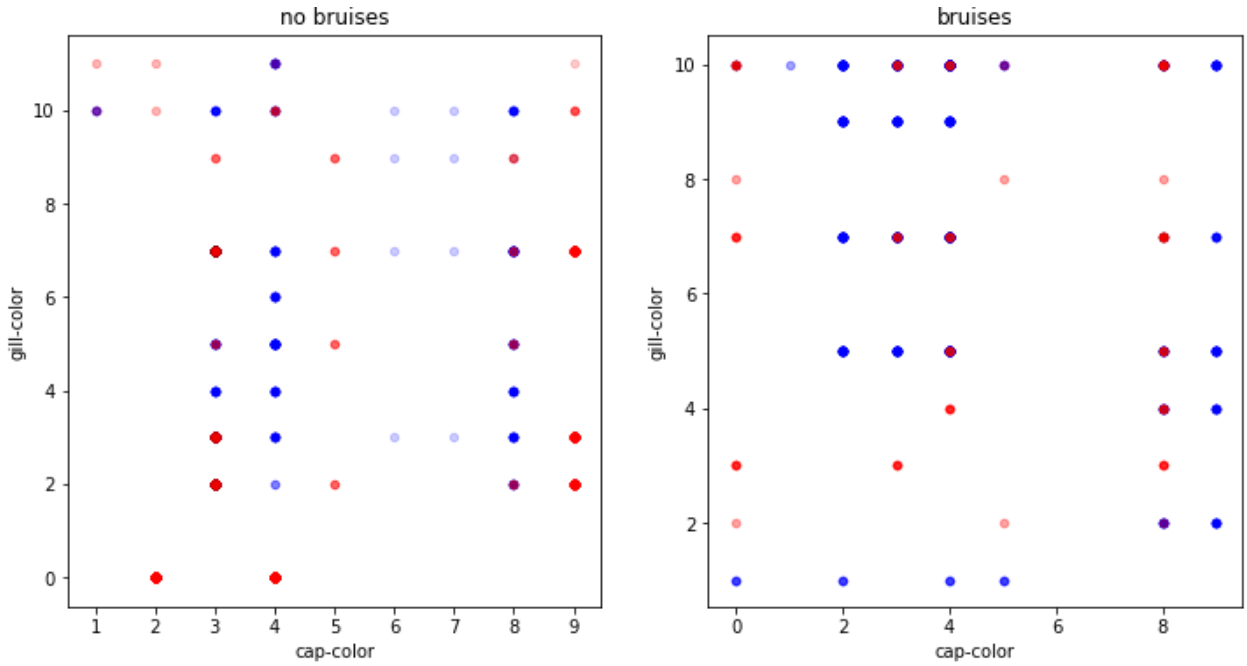
Odmah uočavamo da većina podataka nije numerička, stoga je potrebno obaviti transformacije i to na sljedeći način: svaki će znak biti zamijenjen jednim brojem. Pozovemo li funkciju Data_Frame.head() nakon izvođenja transformacija, vidjet ćemo da sada svi podatci poprimaju isključivo numeričke vrijednosti.

```
df = dataTransformation(df)
print(df.head())
```

```
   class  cap-shape  cap-surface  cap-color  bruises  odor  gill-attachment  \
0      1          5            2          4        1     6                1
1      0          5            2          9        1     0                1
2      0          0            2          8        1     3                1
3      1          5            3          8        1     6                1
4      0          5            2          3        0     5                1

   gill-spacing  gill-size  gill-color   ...     stalk-surface-below-ring  \
0             0          1           4   ...                            2
1             0          0           4   ...                            2
2             0          0           5   ...                            2
3             0          1           5   ...                            2
4             1          0           4   ...                            2

   stalk-color-above-ring stalk-color-below-ring  veil-type  veil-color  \
0                       7                       7          0           2
1                       7                       7          0           2
2                       7                       7          0           2
3                       7                       7          0           2
4                       7                       7          0           2

   ring-number  ring-type  spore-print-color  population  habitat
0            1          4                  2           3        5
1            1          4                  3           2        1
2            1          4                  3           2        3
3            1          4                  2           3        5
4            1          0                  3           0        1

[5 rows x 23 columns]
```

Slika 3.2

Pokazat ćemo jednostavan graf koji na temelju 3 parametra cap-color (boja klobuka), gill-color (boja prevjesa) i boolean vrijednosti bruises (točkice) prikaže crvenom bojom otrovne gljive, a plavom jestive. Ljubičaste točke predstavljaju situacije u kojima za dane vrijednosti ova 3 parametra postoje i otrovne i jestive gljive. Iz grafa je jasno vidljivo da broj ljubičastih točaka nije prevelik, što znači da bi se mogla obaviti klasifikacija samo na temelju ova 3 parametra i kod većine podataka bila bi ispravna. Jedino bi u situacijama u kojima za dane parametre postoje i otrovne i jestive gljive bilo pogreški. Ovakva je situacija u praksi rijetka i često čak ni na temelju svih početnih parametara nije lako obaviti klasifikaciju sa značajnom točnošću.



Slika 3.3

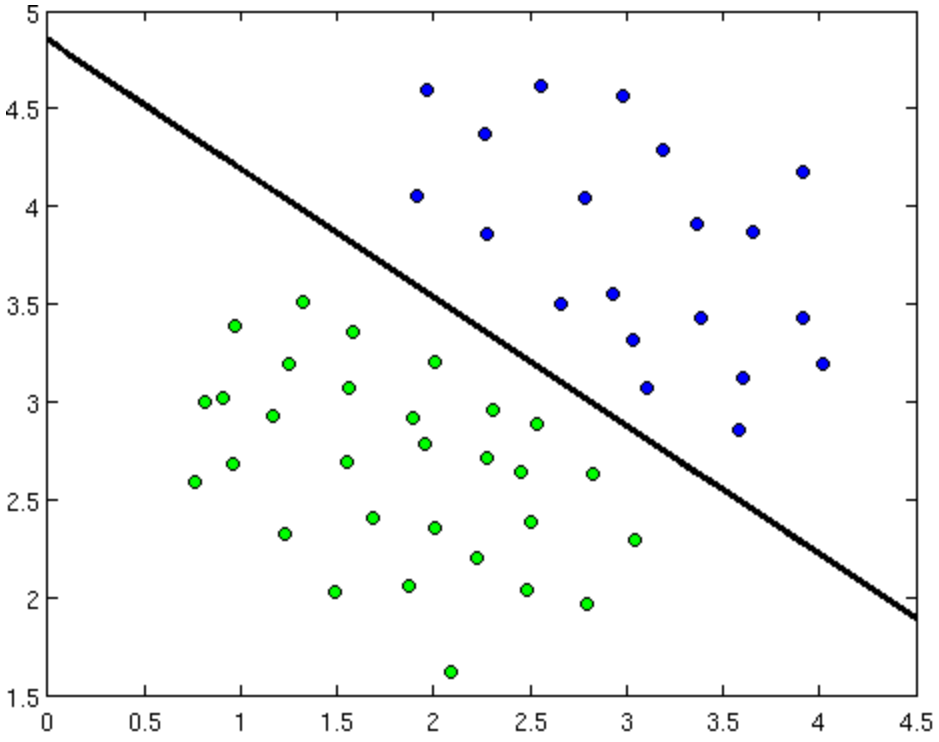
4. Logistička regresija

4.1 Općenito o logističkoj regresiji

4.1.1 Dvije značajke i dva skupa

Logistička regresija je klasifikacijski algoritam koji za svaki od zadanih skupova izračuna vjerojatnost da podatak pripada tom skupu. Podatak se najčešće zatim klasificira u onaj skup za koji je izračunata vjerojatnost da podatak pripada tom skupa najveća. Kod binarne logističke regresije algoritam izračunava dvije vjerojatnosti za svaki podatak koji je potrebno klasificirati i zbroj tih vjerojatnosti je uvijek jednak jedan. Dakle najčešće će podatak biti svrstan u onaj skup za koji je izračunata vjerojatnost da podatak pripada tom skupu veća od 0.5. Ponekad se ova granica može mijenjati. Recimo, na našem primjeru klasifikacije gljiva onu gljivu za koju je vjerojatnost da je jestiva 0.51 bi imalo smisla svrstati u skup otrovnih, iako je vjerojatnost da je jestiva veća od vjerojatnosti da je otrovna. Razlog tomu je činjenica da je u praksi puno gore svrstati otrovnu gljivu kao jestivu nego jestivu kao otrovnu. Zbog toga bi mogli granicu postaviti na 0.65 za jestive gljive (0.35 za otrovne) i time postići da u graničnim situacijama algoritam uvijek svrsta gljivu kao otrovnu. Kako je u ovom radu naglasak na analizi algoritama na primjeru dva skupa, a ne na praktičnoj primjeni, koristit ću granicu od standardnih 0.5.

Sada ću objasniti na koji način logistička regresija zapravo radi. Za to je potrebno nekoliko matematičkih koncepata. Značajke je potrebno promatrati kao vektor koji se najčešće označava s x. Recimo da je zbog jednostavnosti da vektor dvodimenzionalan sa vrijednostima x1 i x2. Kada taj vektor pomnožimo s nekim parametrima, nazovimo ih a i b, i dodamo parametar c, dobit ćemo funkciju ax1 + bx2 + c = z. Na temelju njene vrijednosti za svaki podatak možemo odrediti s kojom vjerojatnošću pripada određenom skupu.
Glavni zadatak logističke regresije je naći parametre a, b i c za koje vrijedi sljedeće: ako je za neki podatak funkcijska vrijednost z pozitivna, onda podatak pripada prvom skupu, a ako je negativna, onda podatak pripada drugom skupu. Nakon što kroz učenje dobijemo takve parametre, za svaki novi podatak potrebno je samo odrediti z, odnosno uvrstiti x1 i x2 u formulu z = ax1 + bx2 + c. Faza učenja logističke regresije sada se može promatrati kao faza u kojoj pokušavamo izračunati one parametre a, b i c koji daju najbolji rezultat. Za z jednak 0 dobijemo pravac ax1 + bx2 + c = 0. Taj pravac možemo smatrati granicom između dvaju skupova.
Za podatak koji se nalazi točno na tom pravcu vjerojatnost da pripada prvom i drugom skupu bit će jednaka pa će ga algoritam nasumično svrstati u jedan od skupova. Očito je da će se s jedne strane ovoga pravca nalaziti podatci za koje je z pozitivan, a s druge oni za koje je z negativan.



Slika 4.1

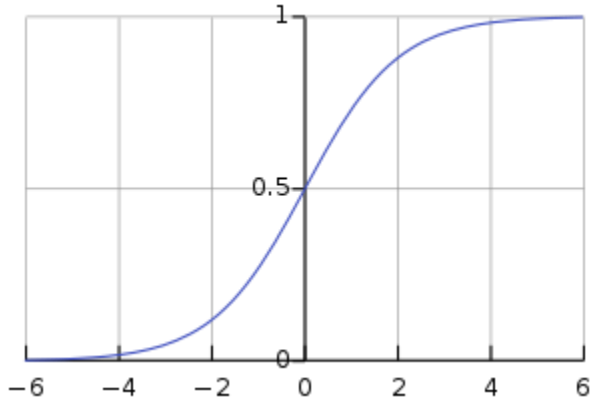
Postavljaju se dva pitanja:
Prvo, je li moguće pronaći parametre takve da nam jednostavno množenje vektora parametara s vektorom značajki podataka daje sve potrebne informacije za njihovu klasifikaciju?
I drugo, ako takvi parametri postoje, kako ih algoritam može izračunati?
Odgovor na prvo pitanje je da je ovakvu klasifikaciju moguće obaviti ako postoji pravac ax1 + bx2 + c = 0 koji podatke ispravno dijeli u dva tražena skupa. Ako ovakav pravac postoji, onda ga je samo potrebno pronaći, čime smo izravno dobili parametre a, b i c. Ako ne postoji pravac koji podatke savršeno dijeli na dva tražena skupa, onda je potrebno pronaći pravac koji radi najbolju moguću podjelu. U tom slučaju algoritam neće imati savršenu preciznost, ali je moguće da će ostvarena preciznost u praksi biti zadovoljavajuća. Kako algoritam izračunava ove parametre bit će objašnjeno u sljedećem poglavlju.

Vjerojatnost da podatak pripada prvom skupu dobije se uvrštavanjem vrijednosti funkcije z za taj podatak u sigmoidnu ili logističku funkciju (Sigmoidna i logistička funkcija su sinonimi).

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

Označimo tu vjerojatnost sa p1. Tada je vjerojatnost da podatak pripada drugom skupu p2 jednaka p2 = 1 – p1.
Ako za neki podatak izračunamo funkcijsku vrijednost z i dobijemo da je ona velik pozitivan broj, tada će taj podatak biti svrstan u prvi skup s velikom vjerojatnošću jer će e^{-z} biti blizu nule ($\lim_{z\to\infty} e^{-z} = 0$) pa će vjerojatnost da je broj svrstan u prvi skup biti blizu jedinice. Ako je z velik negativan broj, onda će nazivnik biti velik broj pa će vjerojatnost da je podatak svrstan u prvi

skup biti blizu nule ($\lim_{z\to-\infty} e^{-z} = \infty$, $\frac{1}{1+\infty} = 0$). Ako je z blizu nule onda će e^{-z} biti blizu jedinice pa će vjerojatnost da je podatak svrstan u prvi skup biti otprilike 0.5. ($\frac{1}{1+e^0} = 0.5$)
Sva tri svojstva su intuitivno logična jer ako je z blizu nule odnosno ako je podatak blizu granice između ova dva skupa vjerojatnosti da je podatak u prvom bi trebale biti slične. Ako je z veliki pozitivan broj odnosno velik negativan broj onda je podatak daleko od granice pa možemo biti relativno sigurni da je podatak u prvom odnosno u drugom skupu.



Slika 4.2

4.1.2 Više značajki i dva skupa

Sličan princip vrijedi i za podatke koji imaju više od dviju značajki. Parametre a i b možemo smatrati dvodimenzionalnim vektorom w. Drugim riječima, funkcija z je skalarni umnožak vektora značajki x i vektora parametara w zbrojenim s parametrom c koji ćemo označiti sa w_0. Ako vektor x nije dvodimenzionalan, već ima n dimenzija, vektor parametara w će također biti n-dimenzionalan. Funkcija z će sada biti jednaka
$z = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + \cdots + w_n * x_n + w_0$, gdje su elementi
$x_1, x_2, x_3, \ldots, x_n$ značajke početnih podataka, a elementi $w_0, w_1, w_2, w_3 \ \ldots, w_n$ izračunati parametri. Sada je potrebno pronaći jednadžbu
$w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + \cdots + w_n * x_n + w_0 = 0$ kojom možemo podatke podijeliti na dva skupa. Već je spomenuto da je za dvije značajke ovo jednadžba pravca. Za tri značajke dobit ćemo jednadžbu ravnine i bit će potrebno naći što bolju ravninu koja podatke dijeli na dva promatrana skupa. Za broj značajki veći od 3 skupovi su odvojeni nečim što se naziva afina hiperravnina (affine hyperplane) koju nije moguće vizualizirati, ali princip je i dalje isti.

4.2 Određivanje parametara w

4.2.1 Cost function

Kako bismo mogli razumjeti na koji način algoritam uči parametre w, prvo ćemo definirati cost function, koji predstavlja koliko je rad algoritma daleko od idealnog. Postavlja se pitanje što je za logističku regresiju idealan rad algoritma. Moguća definicija bi bila: „Idealna je ona klasifikacija koja sve podatke ispravno klasificira“. Ova definicija je nepotpuna jer logistička regresija osim klasifikacije određuje i vjerojatnost da se podatak nalazi u određenoj kategoriji. Zato se kod logističke regresije idealnim smatra ispravna klasifikacija svih podataka i procjena vjerojatnosti na stopostotnu sigurnost. Primjerice, recimo da podatak stvarno pripada kategoriji nula (y = 0). Idealna procjena algoritma bila bi da je vjerojatnost da se podatak nalazi u kategoriji nula jednaka jedan ($p_o = 1$). Iako bi algoritam ispravno klasificirao podatak i uz procjenu $p_o = 0.6$, prvu procjenu smatramo boljom jer je vjerojatnost p_o veća (algoritam je sigurniji u ovu procjenu). Ako podatak pripada kategoriji jedan (y = 1), onda je procjena $p_o = 1$ lošija od procjene $p_o = 0.6$, premda obe procjene rezultiraju pogrešnom klasifikacijom podatka.
Koliko je algoritam blizu idealnom mogli bismo matematički definirati kao p_o za y = 0 i p_1 za y = 1 (idealan je $p_o = 1$ za y = 0 i $p_1 = 1$ za y = 1). S obzirom na to da cost function označava koliko je algoritam daleko od idealnog, mogli bismo ju definirati kao $-p_o$ za y = 0 i $-p_1$ za y = 1. Bolje rezultate daje sljedeća definicija:

$$cost = \begin{cases} -\log p_0\,, & y = 0 \\ -\log p_1\,, & y = 1 \end{cases}$$

Ako je podatak ispravno klasificiran s vjerojatnošću jedan, cost za taj podatak jednak je nuli ($\log 1 = 0$); što je ta vjerojatnost manja, cost je veći, a za vjerojatnost nula cost teži u beskonačnost $-\lim_{p\to 0} \log p = \infty$.

Kako vrijedi $p_0 = 1 - p_1$ formulu za cost function možemo zapisati i ovako:

$$cost = \begin{cases} -\log(1 - p_1)\,, & y = 0 \\ -\log p_1\,, & y = 1 \end{cases}$$

Sada kada smo definirali cost za jedan podatak, prosječni cost za cijeli skup na kojem algoritam uči je jednostavno $cost = \frac{1}{m} * \sum_{i=0}^{m} cost_i$ gdje m predstavlja broj podataka u skupa, a $\sum_{i=0}^{m} cost_i$ sumu cost funkcija za sve podatke.

4.2.2 minimizacija cost function

Sve što algoritam treba napraviti je minimizirati ukupnu cost funkciju pa će time klasifikacija biti onoliko blizu idealnoj, koliko je to moguće za dani skup i algoritam.
Prisjetimo se: algoritam pri učenju određuje parametre w, dakle cilj je naći one parametre w za koje je vrijednost cost funkcije najmanja.

Za početak prikažimo kako bismo minimizirali neku jednostavniju funkciju. Neka je zadana funkcija $f(x) = (\mathrm{x} + 1)^2$. Zanima nas za koju vrijednost x je vrijednost funkcije $f(x)$ najmanja. Mogli bismo derivirati $f(x)$ i izjednačiti derivaciju s nulom pa bismo dobili $f'(x) = 2 * (x + 1) = 0$ odnosno traženi x = -1.
Kada bismo željeli odrediti za koji je w vrijednost cost funkcije minimalna, mogli bismo derivirati cost funkciju u ovisnosti o parametrima w. Problem je što w predstavlja vektor parametara pa bi funkciju trebalo derivirati po svakom parametru. Dobili bismo $\frac{\delta cost(w)}{\delta w_0} = 0$, $\frac{\delta cost(w)}{\delta w_1} = 0$, $\frac{\delta cost(w)}{\delta w_2} = 0\ldots$, odnosno, za svaki parametar dobili bismo jednu jednadžbu pa bi ovaj način minimizacije za velik broj značajki rezultirao rješavanjem prevelikog broja jednadžbi s prevelikim brojem nepoznanica, što bi u konačnici bilo prezahtjevno za izračun. Zbog toga se u praksi češće koristi metoda gradijentnog spusta.

4.2.3 Gradijentni spust

Metodu gradijentnog spusta prikazat ćemo prvo na primjeru funkcije $f(x) = (\mathrm{x} + 1)^2$.
Kod ove metode potrebno je prvo postaviti argument na neku proizvoljnu vrijednost. Recimo da x postavimo na nulu. Derivacija će tada biti jednaka $f'(0) = 2 * (x + 1) = 2$. Činjenica da je derivacija pozitivna govori nam da bi malo povećanje vrijednosti argumenta x trebalo povećati vrijednost funkcije $f(x)$. S obzirom na to da je naš cilj pronaći minimum ove funkcije, mi ćemo vrijednost argumenta x smanjiti. Recimo da ga smanjimo za vrijednost 0.5. Sada je x = - 0.5, a vrijednost derivacije $f'(-0.5) = 2 * (x + 1) = 1$. Derivacija je ponovno pozitivna pa ćemo x ponovno smanjiti za 0.5 čime dobivamo x = -1. Derivacija je sada jednaka nuli što nam pokazuje da smo dobili traženi minimum. Da smo započeli na primjer sa x = -5, derivacija bi bila negativna, što bi značilo da povećanje vrijednosti x-a smanjuje vrijednost funkcije $f(x)$. Tada bi x povećavali dok ne bismo došli do minimuma. Dakle, ako je derivacija negativna, x ćemo povećati (dodat ćemo mu pozitivnu vrijednost), a ako je derivacija pozitivna, x ćemo smanjiti (dodat ćemo mu negativnu vrijednost). Općenito, metoda gradijentnog spusta sastoji se u tome da započnemo s proizvoljnom vrijednošću argumenata i zatim u svakom koraku dodajemo vrijednost čiji je predznak suprotan predznaku derivacije. Ovaj korak ponavljamo sve dok se derivacija ne izjednači s nulom, što označava da smo dostigli traženi minimum. Kao što smo već napomenuli, w je vektor parametara pa ne govorimo o jednoj derivaciji, ali princip je vrlo sličan. S obzirom na to da imamo vektor parametara, i za svaki član vektora po jednu derivaciju, imat ćemo i vektor derivacija koji se naziva gradijent. Vrijedi:

$$\nabla = \begin{matrix} \frac{\delta cost(w)}{\delta w_o} \\ \frac{\delta cost(w)}{\delta w_1} \\ \ldots \end{matrix}$$

Ponovno ćemo započeti s proizvoljnom vrijednošću parametara w i u svakom ćemo koraku mijenjati vektor parametara w tako da mu dodamo vektor čiji je smjer suprotan smjeru gradijenta (- ∇), tj. svaki parametar mijenjamo tako da mu dodamo vrijednost čiji je predznak

suprotne vrijednosti pripadajuće parcijalne derivacije (ako je $\frac{\delta cost(w)}{\delta w_i}$ pozitivna, w_i ćemo smanjiti i obrnuto).
Prema tome, predznak derivacija određuje koje ćemo parametre smanjiti, a koje povećati. Vrijednost za koje ćemo parametre mijenjati u svakom koraku može biti konstanta ε koju ili moramo specificirati ili je algoritam samo odredi. Ta vrijednost također može biti proporcionalna gradijentu $\varepsilon * \nabla$ ili se može raditi o još složenijim funkcijama. To ovisi o vrsti i implementaciji gradijentnog spusta i nije toliko važno za daljnje razumijevanje rada pa se ovime više nećemo baviti.
Ono što je najvažnije u ovom poglavlju je osnovni koncept učenja jer je on primjenjiv na većinu algoritama strojnog učenja. Prvo se definira idealan rad algoritma, zatim se odredi cost funkcija kao mjera koliko je algoritam trenutačno daleko od idealnog i na kraju se kroz minimizaciju odrede parametri algoritma.

4.3 Implementacija logističke regresije u programskom jeziku Python

U ovom će poglavlju biti prikazana implementacija logističke regresije koristeći programski jezik Python i bilježnicu (library) scikit. Implementacije drugih algoritama bit će slične implementaciji logističke regresije pa neće biti tako detaljno prikazane.
Prvi korak je podijeliti skup podataka na dva skupa – prvi, na kojem će naš model učiti, i drugi, na kojem ćemo testirati njegovu efikasnost. Većinom skup podataka iz kojega model uči sadrži između 60 i 80 posto početnog skupa. Razlog zbog kojeg je nužno testirati model na podatcima iz kojih nije učio je činjenica da se modeli ponekad jako dobro prilagode podacima na kojima su učili, a za ostale podatke daju puno lošije rezultate. Ovaj problem se naziva prijevod (overfitting) i detaljnije će biti objašnjen u idućem poglavlju.

```
msk = nr.rand(len(df)) < 0.8
dfTrain = df[msk]
dfTest = df[~msk]
```

Slika 4.3

Sljedeći je korak odabrati značajke na kojima će model učiti. To mogu biti sve značajke ili samo dio njih. U ovom primjeru nekoliko izostavljenih parametara nije poboljšavalo rezultat rada algoritama pa su zbog toga izbačeni. Nakon toga potrebno je odabrati klasifikator (classifier). Klasifikator je algoritam koji ćemo upotrijebiti za klasifikaciju. U ovom smo slučaju odabrali logističku regresiju bez ikakvih parametara. Zatim pozovemo funkciju classificationAlgorithm koja je zajednička za sve prikazane algoritme i prikažemo rezultate.

```
numOfFeaturesLogR = 16
featuresLogR = ['stalk-shape', 'bruises', 'gill-size' , 'stalk-color-above-ring', 'stalk-root',
                'cap-shape','cap-surface', 'odor', 'gill-color', 'veil-type', 'spore-print-color',
                'ring-type', 'cap-color', 'veil-color', 'habitat', 'population']

classifier = linear_model.LogisticRegression()
dfTest2 = classificationAlgoritam(dfTrain, dfTest, featuresLogR, numOfFeaturesLogR, 'class', classifier)

classificationResults(dfTest, 'class', 'odor')
```

Slika 4.4

Funkcija classificationAlgorithm (slika 4.5) s prvih pet naredbi obavi posljednje transformacije potrebne da bi algoritam mogao raditi s promatranim podatcima, a potom metoda classifier.fit(X,Y) uči iz značajki (X) i oznaka skupova u koje pripadni podatci trebaju biti svrstani. Naposljetku naredba trainedClassifier.predict(XTest) obavi klasifikaciju na temelju značajki iz skupa XTest. Bitno je primijetiti da nigdje nismo koristili informacije o tome u koje skupine podatci iz testnog skupa pripadaju.

```
def classificationAlgoritam(df, dfTest, features, numOfFeatures, targetedClass, classifier):

    nrow = df.shape[0]
    nrowTest = dfTest.shape[0]

    X = df[features].as_matrix().reshape(nrow, numOfFeatures)
    XTest = dfTest[features].as_matrix().reshape(nrowTest, numOfFeatures)

    Y = df[targetedClass].as_matrix().ravel()

    trainedClassifier = classifier.fit(X, Y)

    dfTest.predicted = trainedClassifier.predict(XTest)

    return dfTest
```

Slika 4.5

Pozovemo li nakon obavljene klasifikacije funkciju classificationResults, dobit ćemo podatke o tome koliko ju je efikasno algoritam obavio.

4.4 Prikaz rezultata

```
def classificationResults(df, targetedClass, singleFeature):

    truePos = df[((df.predicted == 0) & (df[targetedClass] == df.predicted))]
    falsePos = df[((df.predicted == 0) & (df[targetedClass] != df.predicted))]
    trueNeg = df[( (df.predicted != 0) & (df[targetedClass] != 0))]
    falseNeg = df[((df.predicted != 0) & (df[targetedClass] == 0))]

    print('number of true positive is: ' + str(truePos[singleFeature].count()))
    print('number of false positive is: ' + str(falsePos[singleFeature].count()))
    print('number of true negative is: ' + str(trueNeg[singleFeature].count()))
    print('number of false negative is: ' + str(falseNeg[singleFeature].count()))

    correctlyClassified = (truePos[singleFeature].count() + trueNeg[singleFeature].count())/df[singleFeature].count()

    print("Correctly Classified: " + str(correctlyClassified))
    print("Precision: " + str(truePos[singleFeature].count() /
                              (truePos[singleFeature].count() + falseNeg[singleFeature].count())))

    print("Recall: " + str(truePos[singleFeature].count()/
                            (truePos[singleFeature].count() + falsePos[singleFeature].count())))
```

Slika 4.6

```
number of true positive is: 780
number of false positive is: 77
number of true negative is: 698
number of false negative is: 40
Correctly Classified: 0.926645768025
Precision: 0.951219512195
Recall: 0.910151691949
```

Slika 4.7

Funkcija classificationResults (slika 4.6) prima set za testiranje algoritma koji sada za svaki podatak ima i oznaku klase kojoj taj podatak stvarno pripada i oznaku klase u koju ga je naš algoritam svrstao. Obično za klasu koja sadrži manje podataka kažemo da je ona pozitivna (u daljnjem tekstu „prva“ klasa).
Tada za svaki podatak postoje četiri mogućnosti:

- Podatak pripada prvoj klasi i u nju je svrstan pa je on prijevod (true positive)
- Podatak pripada prvoj klasi, ali svrstan je u drugu, stoga je on prijevod (false negative)
- Podatak pripada drugoj klasi, ali svrstan je u prvu, takav je podatak prijevod (false positive)
- Podatak pripada drugoj klasi i u nju je svrstan – podatak je prijevod (true negative)

Prve četiri naredbe funkcije classificationResults podjele početni skup na gore navedene kategorije. Zatim se ispiše koliko podataka pripada pojedinoj kategoriji. Ispravno klasificiran (correctly classified) udio je jednostavno broj ispravno klasificiranih podataka podijeljen s ukupnim brojem podataka. Mjere precision i recall nisu toliko intuitivne, ali su neizostavne zato što udio ispravno svrstanih podataka ponekad nije relevantan i to je najjednostavnije prikazati na primjeru. Recimo da je 99% gljiva jestivo, a

samo 1% otrovno. Glavni cilj algoritma bio bi ustanoviti koje podatke svrstati u tih 1%. Tada bi neki algoritam jednostavno sve gljive mogao proglašavati jestivima i bio bi u pravu u 99% slučajeva! Jasno je da takav algoritam ne ispunjava svoju svrhu i da je praktički beskoristan, bez obzira na iznimno visok udio ispravno klasificiranih podataka. Općenito, ako većina podataka pripada jednome skupu, nije moguće koristiti udio ispravno klasificiranih podataka kao mjeru efikasnosti algoritma. U tim slučajevima mogu se koristiti precision i recall.

Precision predstavlja omjer broja podataka koji su ispravno označeni kao pozitivni i ukupnog broja pozitivno označenih podataka.

Precision = $\frac{\#truePositive}{\#truePositive + \#falsePostivie}$

Algoritam iz našeg primjera koji sve podatke svrstava u negativnu kategoriju (sve gljive klasificira kao jestive) imao bi nedefiniran precision – brojnik i nazivnik bili bi nula, zato što niti jedna gljiva ne bi bila označena kao pozitivna, tj. otrovna.

Recall se izračunava kao omjer podataka koji su ispravno klasificirani kao pozitivni i svih podataka koji bi trebali biti označeni kao pozitivni.

Recall = $\frac{\#truePositive}{\#truePositive + \#falseNegative}$

Očito je da bi za algoritam iz našeg primjera recall imao vrijednost nula – niti jedna gljiva ne bi bila označena kao otrovna, a postoji određeni broj otrovnih gljiva.

S obzirom na to da dio nazivnika kod precisiona čini broj lažno pozitivnih, a kod recalla lažno negativnih podataka, u praksi se nerijetko koristi njihova kombinacija kako bi se jednaka važnost pridala jednoj i drugoj vrsti pogreške. Najčešće se koristi F1 score koji se izračunava kao:

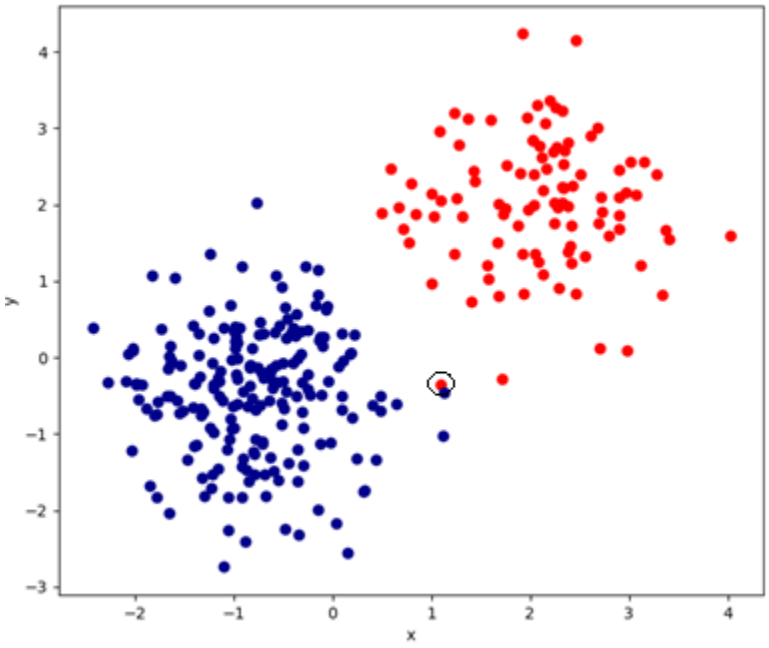
$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

5. C PARAMETAR I OVERFITTING

5.1 Uvod u overfitting

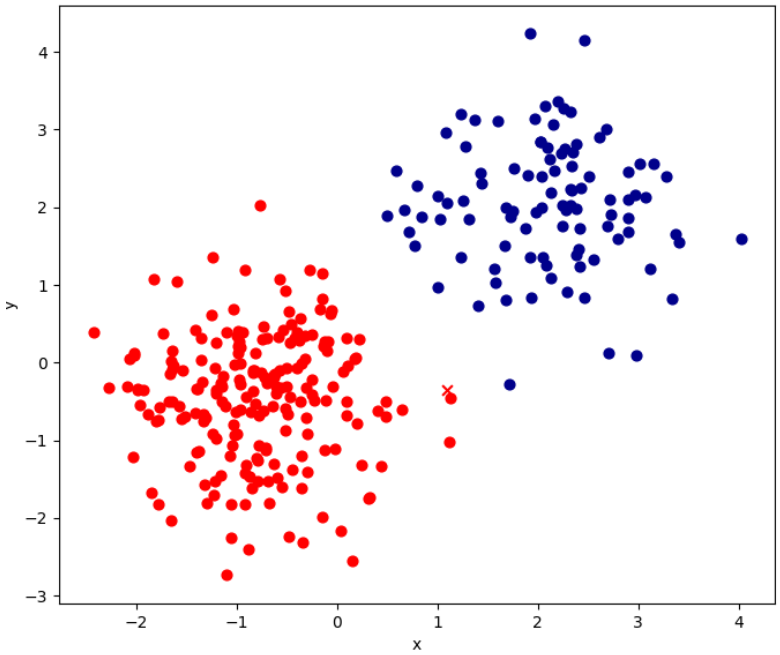
Prije osvrta na problem overfittinga, pokazat ću dvije moguće klasifikacije niže prikazanog skupa.

Posebnu pažnju treba posvetiti zaokruženom podatku koji se, iako crven, nalazi puno bliže plavim podatcima.



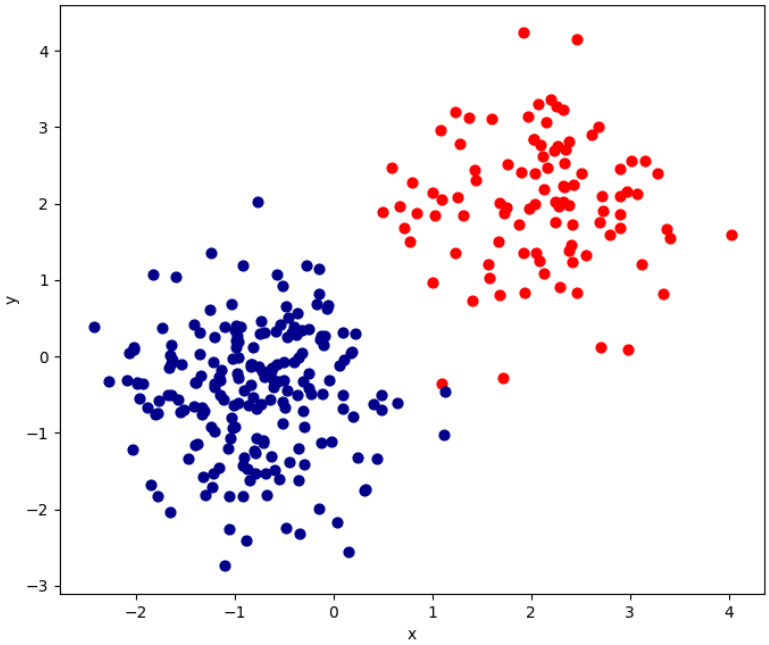
Slika 5.1

Recimo smo odlučili upotrijebiti algoritam logističke regresije koristeći tri značajke: jedna značajka će biti x, druga y, a treća će odgovarati njihovom umnošku x*y. Iako se možda čini neobičnim da je jedna značajka funkcija preostalih dviju, u praksi je često jedna značajka određena ostalima, samo što ova povezanost rijetko bude toliko očita.
Ispravno svrstani podatci označeni su krugom, a neispravno klasificirani znakom x. Uočavamo da je algoritam uspio ispravno klasificirati sve podatke osim onog promatranog.



Slika 5.2

Sada na istom skupu upotrijebimo isti algoritam, ali ovoga puta koristeći veći broj značajki. Recimo da su značajke sada x, y , x*y, x*x, y*y, x/y i y/x. Koristeći ovih 7 značajki algoritam je uspio ispravno klasificirati sve podatke.
Je li ova klasifikacija bolja od prethodne i možemo li povećati uspješnost algoritma jednostavno povećavajući broj značajki?



Slika 5.3

5.2 Overfitting

Problem overfittinga može se usporediti s ljudskim problemom predrasuda. Zamislite da ste došli na godišnji odmor u neku državu koju dosada niste posjetili. Ugodno ste putovali, sletjeli i sada tražite najlakši način da od aerodroma dođete do svoga hotela. Kako to obično biva, aerodrom vrvi taksijima. Bez puno razmišljanja sjedate u prvi taksi i navodite adresu odredišta. Kada ste stigli, taksist za uslugu prijevoza traži neku ogromnu svotu novca. Taksimetar ne vidite, jasno vam je kako vas pokušava prevariti, ali nemate velikog izbora i plaćate mu onoliko koliko je zatražio. Iz ovog neugodnog iskustva izvodite zaključak da su svi taksisti u ovoj državi lopovi. Na temelju jednog „primjerka“ presudili ste o svima i gotovo je sigurno da ta pretpostavka nije točna i da je ovaj taksist iznimka. Kako biste se razuvjerili, mogli biste se još puno puta dok ste tamo voziti taksijem, ali to bi bilo vremenski zahtjevno i svakako bi puno koštalo. S druge strane, mogli biste odustati od svoje pretpostavke da su svi taksisti lopovi i zamijeniti je jednostavnijom i realnijom pretpostavkom poput one da svugdje ima poštenih i nepoštenih ljudi, pa tako i među taksistima.
Kada uče, računala često tako razmišljaju – u setu na kojem algoritam uči pojavi se iznimka i tada na osnovu nje algoritam stvara prilično specifično pravilo pa će kasnije sve podatke slične ovoj iznimci loše klasificirati. Iako bi najbolja opcija bila prikupiti puno podataka, kao i u priči s taksistom, možda nemamo dovoljno vremena i novca. Ako je to slučaj, mogli bismo „reći“ računalu da odustane od specifičnog pravila i zamijeni ga jednostavnijim modelom, slično kao što bismo pretpostavku da su svi taksisti lopovi zamijenili pretpostavkom da svugdje ima poštenih i nepoštenih ljudi.

U primjeru sa skupovima dvije su mogućnosti: možda je zaokruženi crveni podatak jednostavno iznimka pa onda nije loše da ga algoritam krivo klasificira, a možda je ipak nešto na osnovu čega treba modificirati pravilo i podatke slične njemu stvarno svrstati u crveni skup. U praksi, kada se pojavi jedan izoliran podatak, puno je realnije da je on uistinu iznimka nego neki novi slučaj koji treba uzeti u obzir i zato bi prvi algoritam s manje značajki koji ga neispravno svrstava vjerojatno radio bolje od algoritma koji sve podatke klasificira točno i pritom koristi više značajki. Postavlja se pitanje kako u praksi odrediti koji algoritam treba upotrijebiti. U ovome slučaju imamo samo jednu iznimku i značajke koje su funkcija dvije osnovne značajke pa je jednostavno odrediti da je prvi algoritam bolji. Ako u praksi budemo imali stotine značajki i milijune podataka možda neće biti moguće odrediti koji su podatci iznimke, a koje značajke nepotrebne. Na bilo koji način prisiljavati model da bude dovoljno jednostavan kako ne bi patio od overfittinga, a opet dovoljno složen da podatke koji nisu iznimke ispravno klasificira, u ovakvim slučajevima nije tako lako. Zato je gotovo neizbježno korištenje regularizacije.

5.3 Regularizacija

U poglavlju o gradijentnom spustu rekli smo da algoritam uči tako da pokušava minimizirati cost funcion koja predstavlja koliko je algoritam daleko od idealnog. Idealan algoritam smo definirali kao onaj algoritam koji sve podatke klasificira ispravno sa stopostotnom vjerojatnošću. Ova definicija favorizira složenije algoritme, a kažnjava algoritme koji bilo koji podatak (pa makar i iznimku) pogrešno klasificiraju. Zato je tu definiciju potrebno promijeniti. Najintuitivnije bi bilo proglasiti idealnim onaj algoritam koji ima najveću točnost, ali koristi najmanje značajki, tj. radi najbolji kompromis između točnosti i složenosti. Uvodimo novi cost function koja sada glasi :

$$cost(w) = -\frac{1}{m} * \sum_{i=1}^{m}\left(y^i * \log_e logistic\left(w * x^i\right) - (1 - y^i) * \log_e(1 - logistic\left(w * x^i\right))\right) + \lambda * \mathrm{k}.$$

Gdje je k broj parametara w različitih od nula, a λ konstanta.

Kada bi za neku značajku x_k wk bio jednak nuli, onda bi umnožak x_k*w_k također iznosio nula, što je ekvivalentno nekorištenju značajke x_k. Vrijednost parametra λ sada nosi informaciju o tome koliko nam je važno imati mali broj značajki u odnosu na to koliko nam je važna točnost samog algoritma. Za malenu vrijednost λ algoritam se ponaša slično kao i bez regularizacije, dok za veliku vrijednost λ algoritam je prisiljen koristiti manji broj značajki, bez obzira na potencijalno narušavanje točnosti. λ ponekad nazivamo hiperparametrom kako bi ju razlikovali od parametara w koje algoritam određuje pri fazi učenja.

Za apsolutnu vrijednost parametara w_k možemo reći da određuje važnost značajke x_k. Ako je $|w_k|$ velik onda će umnožak w_k*x_k najčešće biti velik pozitivan ili velik negativan broj što znači da će imati velik utjecaj na sumu $z = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + \cdots + w_n * x_n + w_0$. Ako je $|w_k|$ blizu nule onda će umnožak w_k*x_k također najčešće biti blizu nule pa neće imati velik utjecaj na ovu sumu. Umjesto da nastojimo pojednostaviti algoritam izbacivanjem dijela značajki postavljajući određene parametara na nulu, to možemo učiniti i na manje intuitivan način. Ako smanjimo vrijednost određenog parametra umjesto da ju postavimo na nulu, nećemo potpuno izbaciti utjecaj odgovarajuće značajke, ali ćemo ga svakako smanjiti. Na ovaj način također dobivamo jednostavniji algoritam. Ako bismo to učinili za sve parametre, tada bismo nastojali minimizirati sumu apsolutnih vrijednosti parametara w. Cost funcion bi sada izgledao ovako:

$$cost(w) = -\frac{1}{m} * \sum_{i=1}^{m}\left(y^i * \log_e logistic\left(w * x^i\right) - (1 - y^i) * \log_e(1 - logistic\left(w * x^i\right))\right) + \sum_{i=1}^{m} |w|_j.$$

Ova vrsta regularizacije naziva se l1 regularizacija.

Drugi način na koji možemo ovo promatrati je sljedeći: tražimo kompromis između točnosti i važnosti svake značajke. Ako bismo trebali znatno povećati utjecaj nekih značajki kako bismo ispravno klasificirali samo jedan podatak, time bi se značajno povećala suma apsolutnih vrijednosti parametara w, a točnost bi se neznatno promijenila. Dakle, na ovaj način algoritam bi mijenjao svoj rad samo ako bi time značajno povećao točnost ili ako bi time postajao jednostavniji. Budući da ispravno klasificiranje jednog dodatnog podatka ne utječe značajno na točnost, a dodavanje pravila vezanog isključivo uz neki podatak povećava složenost, očito je da bi algoritam sada bio manje osjetljiv na iznimke. U praksi se osim sume apsolutnih vrijednosti koristi i suma kvadrata, između ostaloga zato što je s kvadratima jednostavnije „raditi“. Tada govorimo o l2 regularizaciji.

$$cost(w) = -\frac{1}{m} * \sum_{i=1}^{m}\left(y^i * \log_e logistic\left(w * x^i\right) - (1 - y^i) * \log_e(1 - logistic\left(w * x^i\right))\right) + \sum_{i=1}^{m} w_j^2.$$

5.4 C parametar i implementacija regularizacije

Kod scikit implementacije logističke regresije automatski se koristi l2 regularizacija. Umjesto hiperparametra λ koristi se hiperparametar C za čiju vrijednost možemo reći da je jednaka $C = \frac{1}{\lambda}$. Kao što je manja vrijednost λ odgovarala manjem utjecaju regularizacije, tako veća vrijednost hiperparametra C odgovara manjem utjecaju regularizacije. Pri klasifikaciji gljiva vrijednost hiperparametra C nije bila precizirana, pa ju je algoritam automatski postavio na C = 1. S obzirom na to da mi isprva ne možemo znati koliko je naš algoritam osjetljiv na iznimke i koliko iznimki uopće skup sadrži, ne možemo ni znati kolika je vrijednost hiperparametra C potrebna.
Ako ga postavimo na preveliku vrijednost, utjecaj regularizacije bit će preslab i pojavit će se problem overfittinga. Ako je njegova vrijednost pak premala, utjecaj regularizacije bit će prevelik i model neće moći biti dovoljno složen za obavljanje potrebne klasifikacije. Očito nam je potreban način da pronađemo „idealnu“ vrijednost C-a i za početak ćemo koristiti najočitiju metodu – jednostavno ćemo isprobati velik broj vrijednost i uzeti onu koja daje najbolji rezultat.

```
from sklearn.model_selection import train_test_split

dfTrain2, dfValidation = train_test_split(dfTrain, train_size = 0.25)

bestScore = 0
bestCParam = -1

for i in range(1,100):
        classifier = linear_model.LogisticRegression(C = 0.1*i)
        dfValidation = classificationAlgorithm(dfTrain, dfValidation,
                    featuresLogR, nFeaturesLogR, 'class', classifier)
        score = classificationScore(dfValidation, 'class', 'odor')
        if score > bestScore:
            bestScore = score
            bestCParam = 0.1 * i

print(bestScore)
print(bestCParam)
```

```
0.925993883792
4.1000000000000005
```

Slika 5.4

Sada je skup podijeljen na tri dijela:

- Dio na kojem algoritam uči
- Dio na kojem testiramo efikasnost algoritma
- Dio na kojem testiramo efikasnost algoritma za određeni hiperparametar C

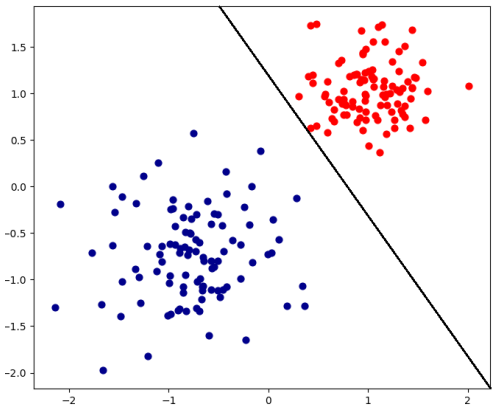
Uočimo da se koriste dva različita podskupa za pronalaženje najboljeg hiperparametra C i za konačno testiranje algoritma. Razlog tomu je činjenica da konačno testiranje treba pokazati efikasnost algoritma

na još neviđenim podatcima, kojima se nije ima priliku prilagoditi, kako bismo dobili objektivnu procjenu efikasnosti koju će algoritam imati u praksi. Isto kao što ne možemo za testiranje koristiti podatke na kojem je algoritam učio parametre w, jer je moguće da su ti parametri više prilagođeni skupu na kojem su trenirani nego bilo kojem drugom skupu, tako ne možemo za testiranje koristiti ni podatke na kojim je određen hiperparametar C. Moguće je da je vrijednost hiperparametra C malo više prilagođena skupu na kojem je određena nego nepoznatom skupu, pa će na njemu algoritam davati malo bolji rezultat nego što će davati u praksi. Razlika u ovom slučaju vjerojatno ne bi bila velika, jer se određuje samo jedan relativno jednostavan hiperparametar, ali ako se broj hiperparametara poveća, ova razlika može postati značajna.
Nakon što smo odabrali vrijednost hiperparametra C koja daje najbolji rezultat, rezultat klasifikacije se popravio, ali ne za preveliku vrijednost. S obzirom na to da se u praksi najviše vremena utroši na dobivanje posljednjih 5% efikasnosti algoritma, ovakva poboljšanja ipak nisu zanemariva.

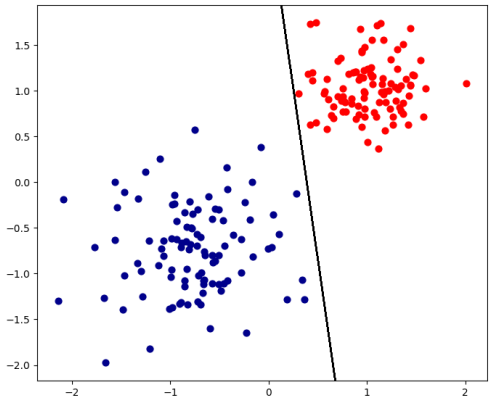
6. SUPPORT VECTOR MACHINE

6.1 Linearni support vector machine

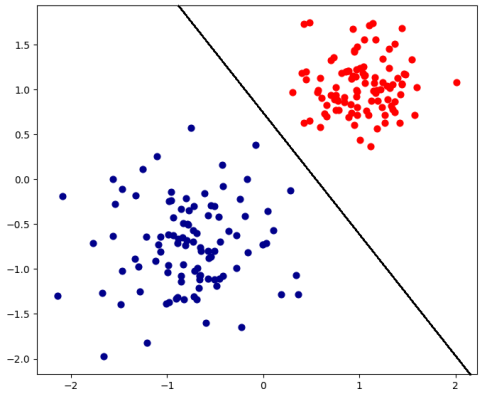
Support vector machine je algoritam koji kao i logistička regresija služi za nadziranu klasifikaciju. Za razliku od logističke regresije, algoritam ne izračunava izravno vjerojatnost da svaki od podataka pripada određenom skupu, već samo određuje kojem skupu on pripada.
Za početak ćemo govoriti o Support vector machine algoritmu koji ne koristi kernel. Što je kernel, čemu služi i kako pridonosi Support vector machine algoritmu, objasnit će se u jednom od sljedećih poglavlja. Također, zbog jednostavnosti pretpostavit ćemo da je skup podataka iz kojih algoritam uči moguće odvojiti pravcem, ako se radi o dvije dimenzije (dvije značajke), ravninom u tri dimenzije i afinom hiperravninom u četiri i više dimenzija. Jednadžba granice je kao i kod logističke regresije
$w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + \cdots + w_n * x_n + w_0 = 0$ odnosno $x * w + w_0 = 0$, gdje x predstavlja vektor značajki, w vektor parametara, a w_0 konstantu. Algoritam logističke regresije u fazi učenja pokušava pronaći vektor w i konstantu w_0 takvu da granica dobivena ovom jednadžbom dijeli podatke na dva tražena skupa. Ipak takvih granica može biti više (u praksi najčešće ili ne postoji granica ili postoji beskonačno mnogo granica).



Slika 6.1



Slika 6.2



Slika 6.3

Uči li algoritam na nekom dobro razdijeljenom setu podataka, poput onog na gornjim slikama, postojat će više granica koje na tom setu daju jednako dobre rezultate. Uvođenjem testnog seta mogu se pojaviti podatci koji se nalaze „između“ onih dosada viđenih pa za testni set nije svejedno koju granicu odabiremo. Primjerice, algoritam sa slike (prve od tri) bi podatak koji bi se nalazio slijeva granici, svrstao u plavi skup, čak i ako mu je crveni skup „bliže“. S druge strane, kako na slici trećoj od tri granica prolazi točno sredinom između dvaju skupova, algoritam bi podatak svrstao u onu grupu do koje mu je manja udaljenost pa bismo mogli reći da je ta granica bolja od ostalih.

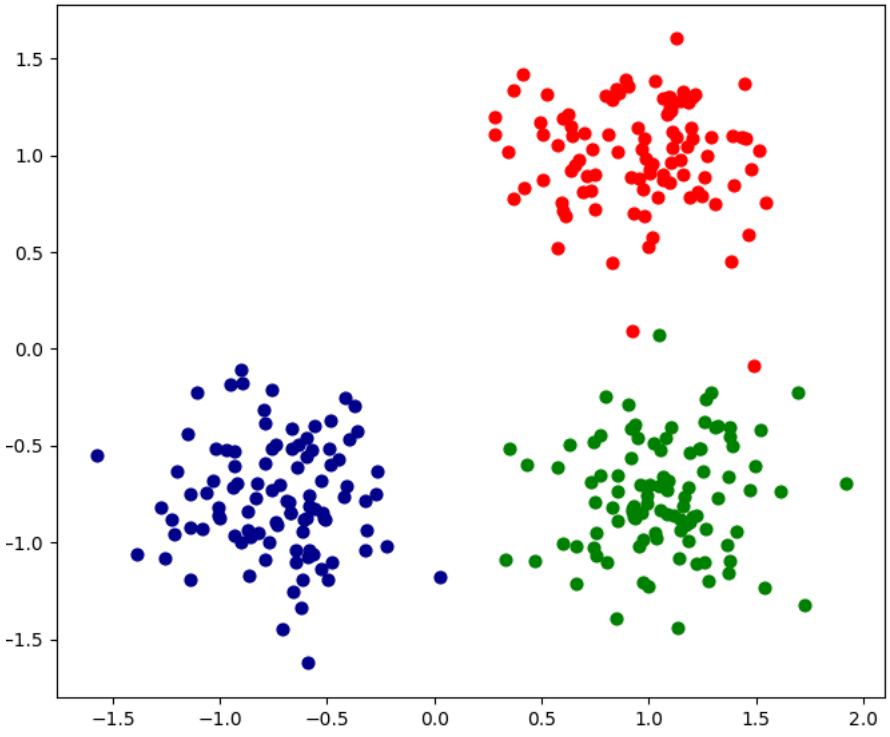
Najbolja se granica definira kao ona granica za koju je udaljenost između nje same i podatka koji je toj granici najbliži maksimalna.

Logistička regresija odabrala bi jednu od granica, ne nužno najbolju. SVM pak, traži upravo najbolju granicu i to je prva njegova prednost u odnosu na logističku regresiju.

Zbog ovakvog načina rada za SVM kažemo da je maximum margin classifier. Druga velika prednost je mogućnost korištenja različitih kernela.

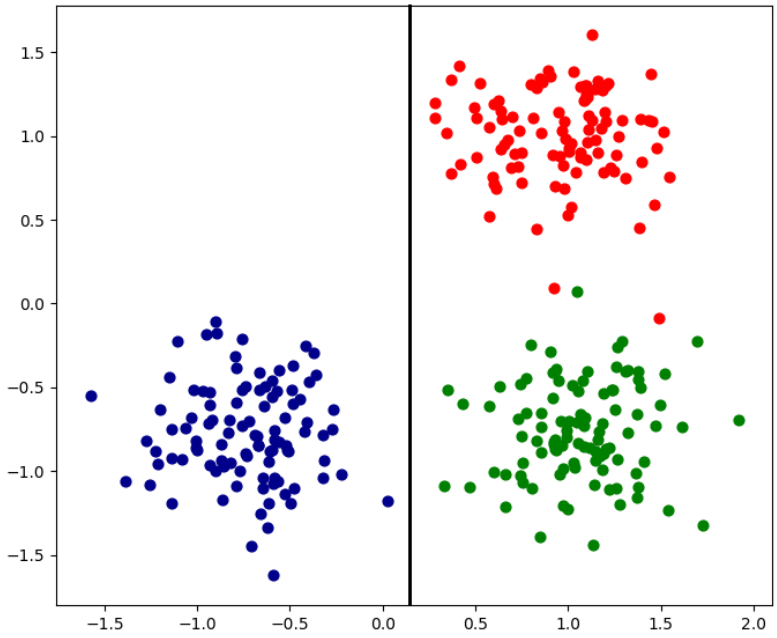
7. STABLO ODLUKE (DECISION TREE)

Stablo odluke dosta je drugačije od dosada spomenutih algoritama. Za razliku od logističke regresije i SVM, koji kod binarne klasifikacije odmah pokušavaju podijeliti skup u 2 tražena dijela, stablo odluke će početni skup podijeliti u puno više od dvije skupine. Ako se radi o binarnom stablu, početni će se skup u prvoj iteraciji podijeliti na dva podskupa, ali onda će se i svaki od tih podskupova ponovno podijeliti na manje podskupove. Ti podskupovi će se dalje dijeliti sve dok ne dobijemo takozvane listove stabla. Listovi stabla su podskupovi početnog skupa nastali nizom uzastopnih podjela i njih više ne dijelimo na manje podskupove. Ideja algoritma je da pokušamo dobiti list koji će sadržavati podatke samo iz jedne klase. Tada bi mogli jednostavno sve podatke tog lista svrstati u tu klasu. Kada bismo ovo postigli za sve listove, algoritam bi imao savršenu preciznost. Ako u nekom podskupu postoji velik broj podataka koji pripadaju različitim klasama, algoritam će najčešće nastaviti dijeliti taj podskup. S obzirom na to da stablo odluke obavlja klasifikaciju na temelju više podjela, svaka podjela može biti puno jednostavnija nego u prošlim algoritmima. Podjela se u svakom koraku najčešće radi samo na temelju jednog parametra. Ako neki podatak ima promatrani parametar veći od neke granične vrijednosti, svrstat će ga se u jedan podskup, a ako je promatrani parametar podatka manji od te granične vrijednosti, bit će svrstan u drugi podskup. Prikažimo rad algoritma na primjeru skupa sa slike.



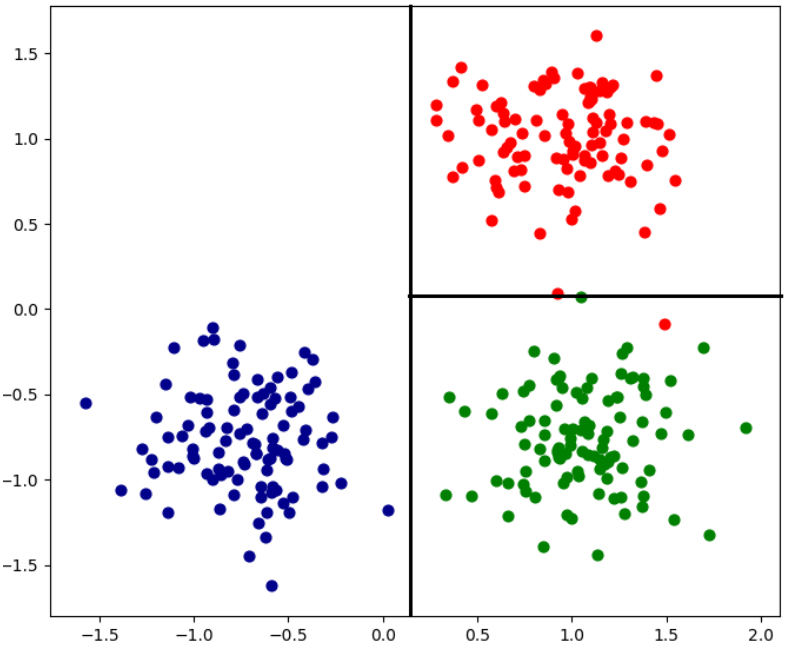
Slika 7.1

Algoritam će prvo podijeliti skup prema parametru x_1.



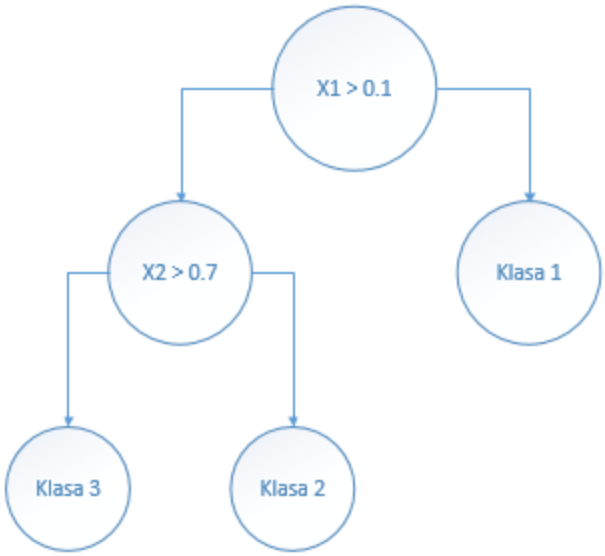
Slika 7.2

Uočimo da jedan podskup već sadrži elemente samo jedne skupine pa daljnja podjela tog podskupa nije potrebna. Drugi skup ćemo podijeliti prema parametru x_2.



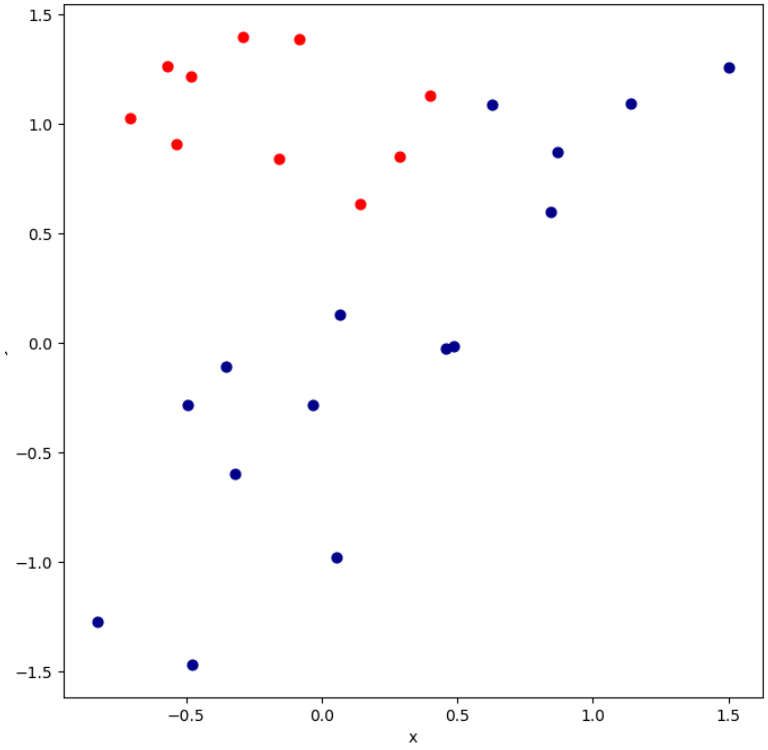
Slika 7.3

Time smo podijelili skup u tri podskupa. U jednom se nalaze podatci prve klase, a u dva podatci druge klase. Rad stabla možemo prikazati i na sljedeći način:



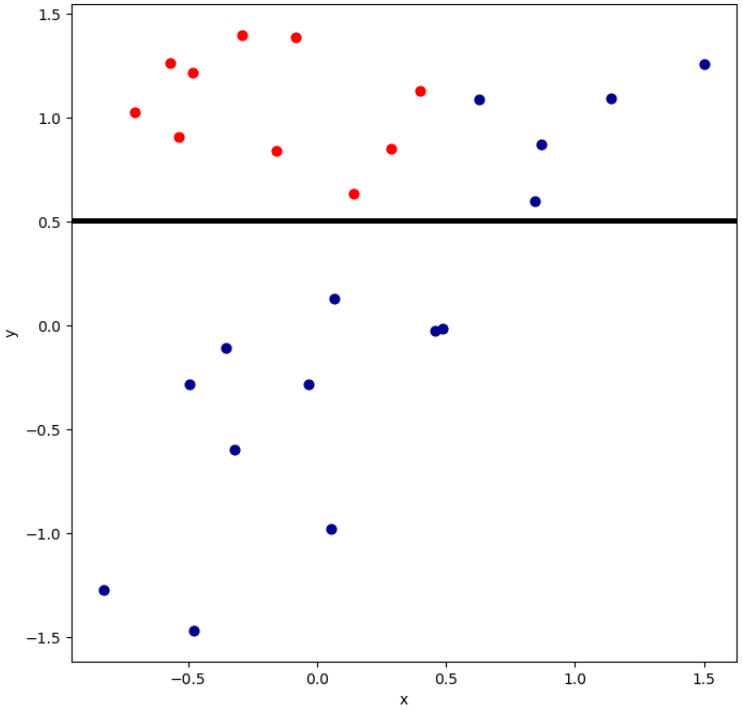
Slika 7.4

Ako je poznato koje podjele treba učiniti, rad algoritma je trivijalan. Pitanje je kako algoritam određuje na temelju koje će se značajke obaviti podjela i koju će graničnu vrijednost odabrati za promatranu značajku. Odgovor je korištenje pohlepnog algoritma koji u svakome koraku pokušava smanjiti broj pogrešno klasificiranih podataka. Promotrimo primjer na sljedećoj slici.



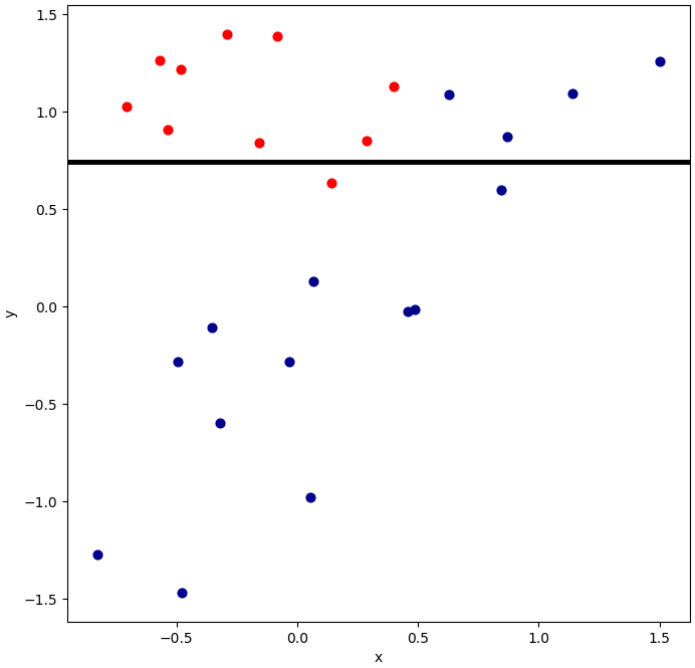
Slika 7.5

Recimo da plave točke predstavljaju kategoriju nula, a crvene kategoriju jedan. Bez ijedne podjele sve što algoritam može učiniti je sve točke svrstati u istu kategoriju (recimo u kategoriju nula). Za svih 15 točaka koje uistinu pripadaju nultoj kategoriji, ova će klasifikacija biti ispravna, ali za preostalih 10 točaka koje pripadaju prvoj kategoriji, klasifikacija će biti pogrešna. U sljedećem koraku algoritam će skup točaka podijeliti na onaj način koji će rezultirati najmanjim brojem pogrešno klasificiranih točaka. Postoji više podjela koje rezultiraju sa samo 5 pogrešno klasificiranih točaka (a niti jedna koja rezultira s manje od 5 pogrešno klasificiranih točaka) i algoritam će proizvoljno odabrati jednu od njih. Recimo da odabere podjelu prikazanu na NAZIV SLJEDEĆE SLIKE. Točke koje se nalaze ispod prikazane linije bit će svrstane u kategoriju nula, a one koje se nalaze iznad nje u kategoriju jedan. Kod ovakve podjele sve su crvene točke ispravno klasificirane, ali 5 plavih točaka sada je smješteno u pogrešnu kategoriju.



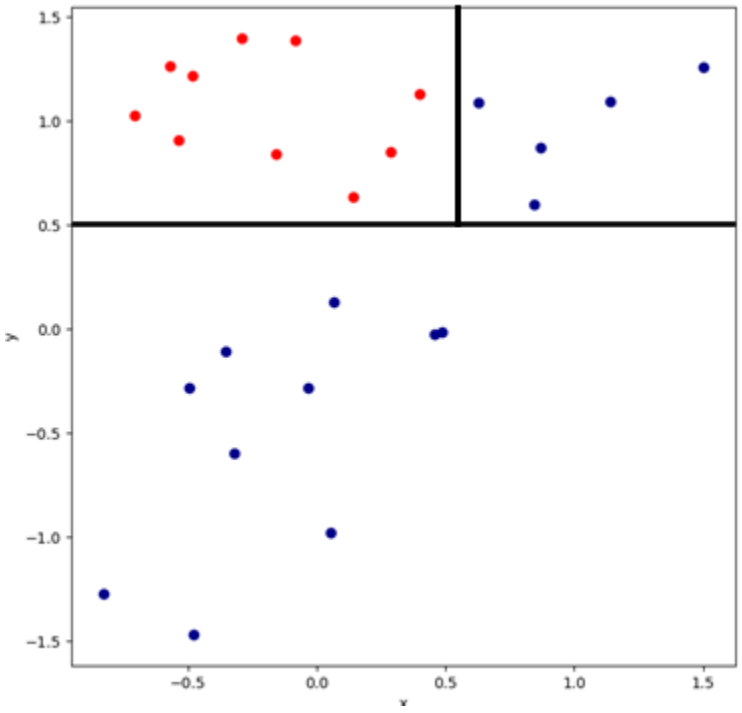
Slika 7.6

Druga moguća podjela je prikazana 7.7. Kod nje će algoritam pogrešno svrstati četiri plave točke i jednu crvenu točku, ali ona bi rezultirala složenijim radom algoritma. Naime, još jedna dodatna podjela skupa koji se nalazi iznad pravca dodatnim vertikalnim pravcem na gornjoj slici rezultirala bi podjelom bez pogrešaka, dok bi, učinimo li istu podjelu na donjoj slici, ostala jedna pogrešno svrstana crvena točka, pa zato podjelu s donje slike nećemo dalje razmatrati.



Slika 7.7

Na sljedećoj slici nalazi se gore opisan posljednji korak algoritma.

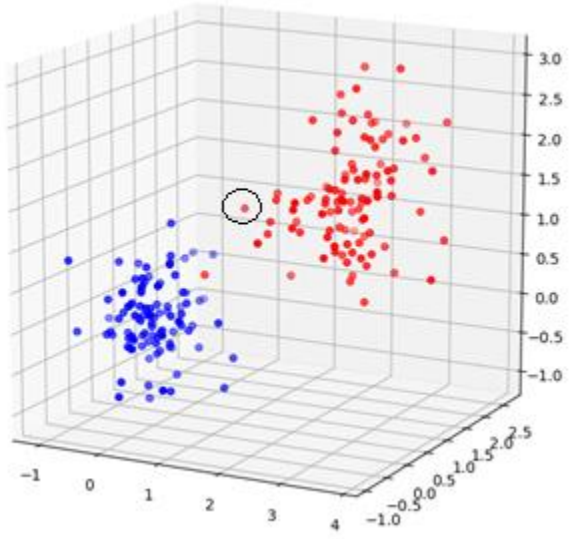


Slika 7.8

Algoritam je brz, jer će pronaći najbolju podjelu relativno jednostavno s obzirom na to da se podjela najčešće radi samo na temelju jedne značajke i da se pokušava pronaći podjela koja daje najbolji rezultatu u sljedećem koraku, bez da se vodi računa o tome hoće li takva podjela biti dugoročno idealna. Također, ako nas osim pitanja gdje je svrstana određena točka zanima i zašto je ona tako svrstana, odgovori koje bi nam mogli dati logistička regresija i SVM nisu previše korisni. Kod najednostavnije logističke regresije mogli bismo saznati da točka pripada određenoj kategoriji, zato što se nalazi s određene strane neke hiperravnine. Kod logističke regresije koja koristi regularizaciju ili kod SVM, odgovor je još manje jasan. Odgovor koji daje stablo odluke je s druge strane jasan, intuitivan i vrlo često koristan. U ovom slučaju odgovor zašto su crvene točke svrstane u kategoriju nula je sljedeći: vrijednost značajke y im je veća od 0.5 (nalaze se iznad horizontalnog pravca) i vrijednost značajke x im je manja od 0.6 (nalaze se lijevo od vertikalnog pravca). Na skupu gljiva bi odgovor na pitanje zašto je određena gljiva svrstana kao otrovna mogao biti: „Zato što je smeđe boje i ima crne točke.“ Mislim da bi većina ljudi preferirala ovakav odgovor u odnosu na: „Zato što njene značajke promatranu gljivu stavljaju u onaj dio ravnine koji hiperravnina s jednadžbom…“. Stablo odluke je na ovom skupu čak imalo i stopostotnu točnost.
Ipak, kao što ćemo kasnije vidjeti, na zahtjevnim skupovima algoritam uglavnom ne daje toliko dobre rezultate.

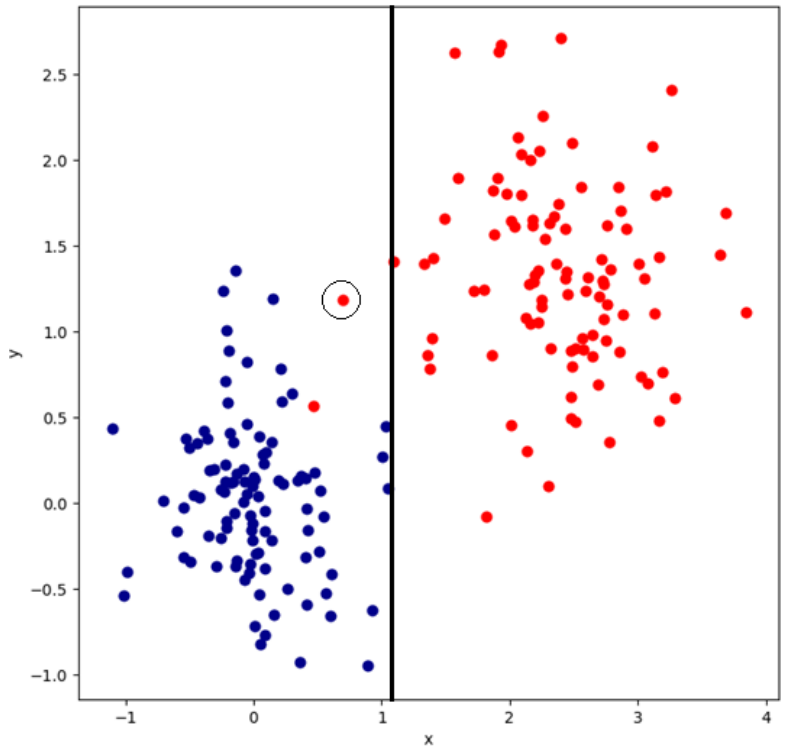
8. SLUČAJNA ŠUMA (RANDOM FOREST)

Osnovna ideja slučajnih šuma vrlo je jednostavna. Umjesto da koristimo jedno stablo odluke, koristit ćemo više različitih stabala odluke i na kraju sumirati njihove rezultate. Kod najjednostavnije verzije slučajne šume svako stablo dobije samo dio značajki i na temelju njih mora obaviti klasifikaciju. Svako će stablo dobiti različit podskup značajki i svako će stablo raditi klasifikaciju neovisno o ostalima. Na ovaj smo način prisilili svako stablo da razvije vlastitu logiku prema kojoj će raditi klasifikaciju. Nakon što su sva stabla obavila klasifikaciju, za svaki podatak pogledamo u koju ga je klasu svrstalo pojedino stablo odluke. Na kraju ćemo podatak jednostavno svrstati u onu klasu u koju ga je svrstalo najviše stabala odluke. Pokažimo rad algoritma na primjeru skupa sa značajkama x, y, t prikazanog na slici IMESLIKE



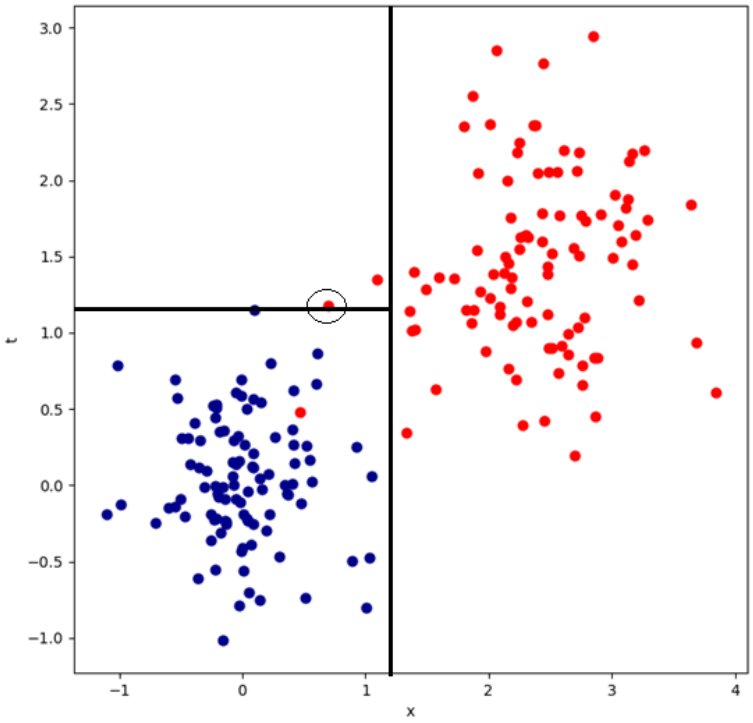
Slika 8.1

Recimo da algoritam koristi samo 3 stabla od kojih će jedno imati pristup značajkama x i y, drugo značajkama x i t, a treće značajkama y i t. Posebnu pozornost posvetimo zaokruženom podatku, koji je, iako to možda ne izgleda tako, isti na sve četiri slike (gornjoj i sljedeće tri).
Prvo će stablo odluke napraviti samo jednu podjelu i na temelju nje klasificirati podatke. Iako ta podjela radi ispravno za gotovo sve podatke, zaokruženi podatak će biti pogrešno klasificiran.



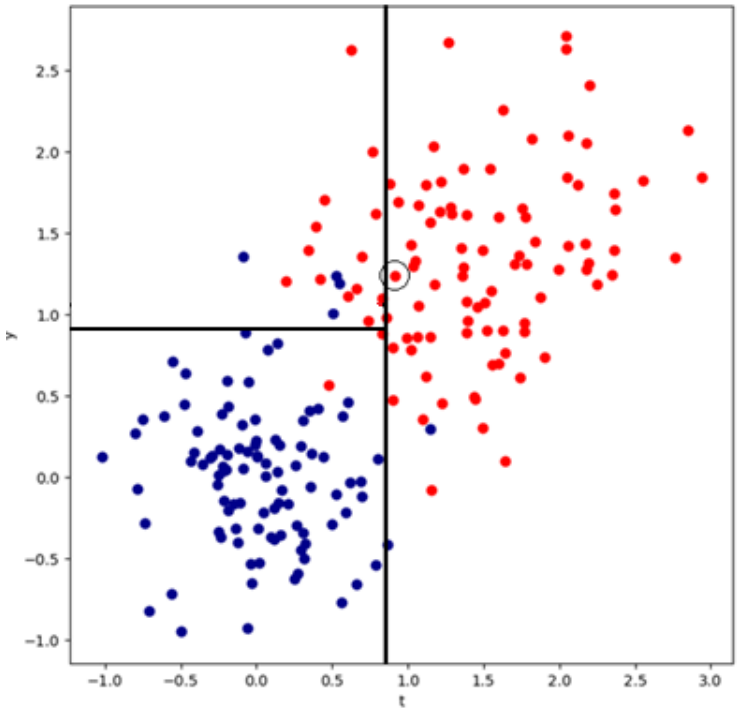
Slika 8.2

Drugo će stablo odluke dvjema podjelama ispravno svrstati sve osim jednog podatka, koji je iznimka lako uočljiv već na prvoj slici (slika 8.1).



Slika 8.3

Treće stablo odluke također ispravno klasificira zaokruženi podatak.



Slika 8.4

Na kraju će algoritam za svaki podatak pogledati u koju su ga kategoriju svrstala ova tri stabla odluke i podatak će biti svrstan u onu kategoriju u koju su ga svrstala barem dva stabla. Iako je prvo stablo odluke naš promatrani podatak pogrešno klasificiralo, preostala dva stabla su ga svrstala u ispravnu kategoriju i on će u konačnici biti ispravno klasificiran.
S obzirom na to da svako stablo odluke radi samo s podskupom značajki, klasifikacija svakog pojedinog stabla najčešće ima manju točnost od prije spomenutih algoritama. Ipak, kada se sumiraju rezultati većeg broja stabala i pogleda njihov ukupni rezultat, točnost je iznenađujuće velika.
Slučajna šuma jedan je od najefikasnijih algoritama pa ne iznenađuje da je i na skupu gljiva imao stopostotnu točnost. Ovaj algoritam također je relativno brz, jer stabla koja koristi najčešće nisu prevelika, a njihov broj rijetko prelazi nekoliko desetaka (defaultna vrijednost je 10). Najveća mana ovog algoritma u odnosu na korištenje jednog stabla odluke je što se gubi intuitivnost koju je prošli algoritam imao – ovdje umjesto jednog jednostavnog odgovora imamo desetak potencijalno različitih koji se sumiraju, odnosno, podatak se nalazi u određenoj kategoriji zato što ga je najveći broj stabala odluke u nju svrstao. U praksi je algoritam najčešće malo složeniji no što je to ovdje prikazano pa su i odgovori koje bi teoretski mogli dobiti još složeniji. Upravo zato je u slučajevima kada nas zanima zašto je podatak svrstan u određenu kategoriju pogodniji algoritam stabla odluke.

9. PRIKAZ SLOŽENIJEG SKUPA PODATAKA

Do sada je prikazan red sva četiri klasifikacijska algoritma, ali implementacija je detaljnije analizirana samo kod logističke regresije. Na skupu mushroom.csv tri su preostala algoritma bila brza i imali stopostotnu točnost pa nije postojala prava potreba za njihovim prilagođavanjem. Zbog toga će implementacija ostalih algoritama, prilagođavanje vrijednosti njihovih hiperparametara, kao i analiza rezultata, biti prikazani u sljedećim poglavljima na skupu otto.csv

Ponovno ćemo započeti učitavanjem skupa i ispisivanjem prvih pet podataka.

```
df = pd.read_csv('C:\\Users\\viktor\\Downloads\\ottoTest\\train.csv')
print(df.head())
```

```
   id  feat_1  feat_2  feat_3  feat_4  feat_5  feat_6  feat_7  feat_8  feat_9
0   1       1       0       0       0       0       0       0       0       0
1   2       0       0       0       0       0       0       0       1       0
2   3       0       0       0       0       0       0       0       1       0
3   4       1       0       0       1       6       1       5       0       0
4   5       0       0       0       0       0       0       0       0       0

     ...     feat_85  feat_86  feat_87  feat_88  feat_89  feat_90  feat_91  \
0    ...           1        0        0        0        0        0        0
1    ...           0        0        0        0        0        0        0
2    ...           0        0        0        0        0        0        0
3    ...           0        1        2        0        0        0        0
4    ...           1        0        0        0        0        1        0

   feat_92  feat_93   target
0        0        0  Class_1
1        0        0  Class_1
2        0        0  Class_1
3        0        0  Class_1
4        0        0  Class_1

[5 rows x 95 columns]
```

Slika 9.1

Značajke su označene samo kao feat_1, feat_2,…, feat_93 pa ne znamo što svaka od značajki predstavlja, a tražena klasa je za ove podatke označena kao Class_1 pa nemamo ni informaciju o tome što ona točno predstavlja. Iako to nije odmah vidljivo iz ovih pet podataka, postoji devet različitih kategorija u koje podatci mogu biti klasificirani.
Sve značajke su numeričke pa nema potrebe za njihovim transformacijama, a skup ćemo kao i prije podijeliti u dio iz kojeg algoritmi uče i dio na temelju kojeg ćemo testirati njihovu efikasnost, a kada to učinimo, možemo krenuti s klasifikacijom.

10. LOGISTIČKA REGRESIJA

Klasifikaciju smo ovaj put podijelili u tri funkcije od kojih prva vraća klasifikator, druga ga trenira i treća na temelju istreniranog klasifikatora obavlja klasifikaciju. Hiperparametre klasifikacijskog algoritma možemo poslati kao argumente prve funkcije ili će ih ona sama odrediti. U ovom slučaju funkcija logisticRegression prima samo hiperparametar C, koji je objašnjen u poglavlju 5.4. U slučaju da nikakva vrijednost hiperparametra C nije poslana, algoritam će isprobati sto različitih vrijednosti i odabrati onu koja daje najbolji rezultat. Jedina razlika u odnosu na prošli program je što je tamo traženje najbolje vrijednosti implementirano, dok se ovdje koristi već gotova funkcija GridSearchCV. Prosječna točnost klasifikacije dobivena logističkom regresijom je gotovo 75% što je jako dobar rezultat za algoritam koji je na prošlom skupu bio daleko najlošiji.

```
classifier = logisticRegression(dfTrain, 'target', cParameter = 8.5)
trainedClassifier = train(dfTrain, features, nFeatures, 'target', classifier)

dfTest = test(dfTest, features, nFeatures, 'target', trainedClassifier)
```

```
Score:  0.748661697271
```

Slika 9.2

S obzirom na to da logistička regresija podjelu na klase ostvaruje podjelom skupa hiperravninama, možemo zaključiti da je ovaj skup strukturiran tako da ga se može hiperravninama relativno dobro podijeliti. Ovo svojstvo se naziva linearna odvojivost (linear separability) i karakteristična je za skupove čije je podatke moguće ispravno razdijeliti najjednostavnijim funkcijama i za skupove koji imaju dovoljno velik broj značajki. Intuitivno objašnjenje zašto velik broj značajki ima utjecaj je sljedeće: linearna podjela skupa s dvije značajke se odnosi na podjelu pravcem, a pravac možemo smatrati jako jednostavnom funkcijom. Podjela skupa s tri značajke odnosit će se na podjelu skupa ravninom, a ravnina je složenija od pravca. Sa četiri značajke imali bismo još složeniju četverodimenzionalnu hiperravninu, a sa 100 značajki još složeniju stotinudimenzionalnu hiperravninu. Za dovoljno velik broj značajki možemo reći da podjela skupa hiperravninom postaje dovoljno složena za obavljanje jako kompleksnih klasifikacija. Više u ovoj intuiciji u Dodatku.

11. SUPPORT VECTOR MACHINE

Support vector machine algoritam ima dva hiperparametra koja ću prikazati, a to su C i kernel. Kao i kod logističke regresije, C se odnosi na regularizaciju – veći C značit će manji naglasak na regularizaciji, što će potencijalno dovesti do overfittinga, dok će manji C značiti manji naglasak na samoj točnosti algoritma. Drugi hiperparametar kernel označava kakvu će funkciju SVM koristiti za razdvajanje podataka. Ako se radi o binarnoj klasifikaciji podataka s dvjema značajkama, x i y, linearni kernel razdvojit će podatke pravcem $w_1 * x + w_2 * y + w_0 = 0$, slično kao i logistička regresija. Kernel „poly“ razdvojit će podatke korištenjem složenije polinomne funkcije oblika:
$w_0 + w_1 * x + w_2 * y + w_3 * x^2 + w_4 * x * y + w_5 * y^2 + w_6 * x^3 + \cdots = 0.$
Rbf ili Gaussov kernel pretpostavlja da je svaka klasa podataka generirana iz jedne Gaussove distribucije i pokušava naći Gaussove distribucije koje najbolje opisuju skup na kojem je algoritam treniran. Kernel općenito možemo smatrati naprednim načinom transformacije podataka koji nam umjesto jednostavnog pravca (hiperravnine za veći broj značajki) omogućuje klasifikaciju podataka složenijim funkcijama. Za linearni kernel nije nam potrebna nikakva transformacija podatka, pa su linerni kernel i ne korištenje kernela sinonimi. Funkcionalnost poly kernela mogli bismo ostvariti tako da za svaku značajku svakog podatka izračunamo sve potrebne članove polinoma i sve te nove izračunate značajke predamo algoritmu da iz njih uči. Problem je što već za polinom drugog stupnja potrebno izračunati sve moguće umnoške značajki i njihove kvadrate: na primjer za značajke x, y i t bi bilo potrebno izračunati: $x * x$, $y * y$, $t * t$, $x * y$, $x * t$, i $y * t$. Za velik broj značajki i veći stupanj polinoma ovaj je izračun prezahtjevan, ali srećom, kernel koristi matematički trik koji ostvaruje ovu funkcionalnost bez potrebe za obavljanjem transformacija. Rbf kernel još je složeniji i nije ga čak ni teoretski moguće ostvariti transformacijom podataka. Na prikazanom skupu najbolji rezultat daje linearni kernel, što se slaže s pretpostavkom iz prošlog poglavlja da je ovaj skup linearno odvojiv, odnosno da se klasifikacija može obaviti dijeljenjem skupa hiperravninama.

```
classifier = SVMClassification(dfTrain, 'target', cParameter = 0.0113, kernelParameter = "linear")
trainedClassifier = train(dfTrain, features, nFeatures, 'target', classifier)

dfTest = test(dfTest, features, nFeatures, 'target', trainedClassifier)
```

```
Score:  0.756257196534
```

Iako algoritam najveću točnost ostvaruje s linernim kernelom, razlika između njega i rbf kernela nije velika.

```
classifier = SVMClassification(dfTrain, 'target', cParameter = 2.2, kernelParameter = "rbf")
trainedClassifier = train(dfTrain, features, nFeatures, 'target', classifier)

dfTest = test(dfTest, features, nFeatures, 'target', trainedClassifier)
```

```
Score:  0.755933983799
```

Zanimljiva razlika je u hiperparametru C, u prvome slučaju on iznosi 0,0113, a u drugome 2,2. Kako je vrijednost od 2,2 vrlo bliska defaultnoj (koja iznosi 1), mogli bismo reći da ona nije

neočekivana. S druge strane, vrijednost od 0,0113 služi nam kao indikator za to da je devedesetdeveterodimenzionalna hiperravnina presložena pa je potrebna značajnija regularizacija (prisjetimo se: manji C, veći utjecaj regularizacije).

12. STABLO ODLUKE

Jedini hiperparametar koji smo prilagođavali kod ovog algoritma je max depth koji određuje maksimalnu dubinu stabla. Ovaj parametar također koristimo za kontrolu overfittinga. Svaka nova podjela koju stablo radi uvodi novo pravilo nad promatranim skupom podataka. Pretpostavka je da će prve podjele biti istinski vezane uz klasifikacijski problem i da je te podjele nužno učiniti. Kasnije podjele će dijeliti manje podskupove podataka, što znači da će se u određenom trenutku potencijalno uvoditi nova pravila za samo nekoliko pogrešno klasificiranih podataka. S obzirom na to da ti podatci mogu vrlo lako biti iznimke, ovakve je podjele bolje izbjegavati. Ograničavajući maksimalnu dubinu stabla ograničavamo maksimalan broj podjela kroz koji jedan podatak može „proći“. Drugi način na koji možemo izbjeći nepotrebne podjele je postavljajući hiperparametar max leaf nodes koji ograničava maksimalan broj konačnih podskupova. Treći način bio bi postavljanje hiperparametra min samples leaf koji ograničava minimalan broj podataka koji konačan podskup mora imati, što znači da algoritam neće moći napraviti novu podjelu, ako je broj podataka zbog koji se podjela radi manji od ovog ograničenja. Ovime osiguravamo da algoritam neće raditi podjelu za podatke koji su najvjerojatnije iznimke. Glavni razlog zbog kojeg smo mijenjali samo jedan hiperparametar ovog algoritma je loš rezultat koji on ostvaruje i činjenica da ga mijenjanje ostalih hiperparametara ne može značajno poboljšati. S obzirom na to da gotovo sve hiperparametre ovog algoritma ima i slučajna šuma (s tim da ih slučajna šuma posjeduje još nekoliko) implementacija ovih hiperparametara bit će prikazana u sljedećem poglavlju.

```
classifier = treeClassification(dfTrain, 'target', maxDepthParameter = 11)
trainedClassifier = train(dfTrain, features, nFeatures, 'target', classifier)

dfTest = test(dfTest, features, nFeatures, 'target', trainedClassifier)
```

```
Score:  0.640870250288
```

13. SLUČAJNA ŠUMA

Kod ovog algoritma prilagođavali smo četiri hiperparametra, tri objašnjena u prethodnom poglavlju (max depth, max leaf nodes i min samples leaf) i n_estimators koji jednostavno predstavlja broj stabala odluke koje algoritam koristi kako bi obavio konačnu klasifikaciju. Bitno je naglasiti da svaki od ovih hiperparametara utječe i na brzinu rada algoritma. Ako ograničimo broj stabala i broj podjela koje svako stablo radi, možemo značajno ubrzati rad algoritma, ali kako algoritam već ima zadovoljavajuću brzinu, u ovom ćemo se radu baviti isključivo njegovom točnošću.

```
classifier = forestClassification(dfTrain, 'target', nEstimatorsParameter = 40,
                        maxLeafNodesParameter = 16384, maxDepthParametar = 26
                        , minSamplesLeafParametar = 2)

trainedClassifier = train(dfTrain, features, nFeatures, 'target', classifier)
dfTest = test(dfTest, features, nFeatures, 'target', trainedClassifier)
```

```
Score:  0.7692665091
```

Iako je stablo odluke imalo jako loš rezultat, slučajna šuma na ovom skupu daje najbolje rezultate! Dakle, bez obzira na to što svako pojedinačno stablo odluke nije moglo imati dobar rezultat, kada se oni sumiraju, točnost se značajno poboljšava. Takav rezultat pokazuje kvalitetu ovog algoritma, ali i snagu ideje koja stoji iza njega. Ako kombinacija rezultata dobivenih radom većeg broja slabijih algoritama u konačnici predstavlja izvrstan rezultat, možda stabla odluke nisu jedini algoritam čije rezultate možemo kombinirati. U praksi danas uistinu najbolje rezultate često daju kombinacije rezultata više algoritama, bilo da se kombiniraju potpuno različiti algoritmi, bilo da se (kao u ovom slučaju) upotrebljava isti algoritam s različitim značajkama ili čak različitim podatcima.

Zaključak

U ovom radu prikazan je rad četiriju klasifikacijskih algoritama. Svaki od njih ima svoje prednosti i mane. Stablo odluke jednostavan je, brz i intuitivan algoritam. S obzirom na ove prednosti i činjenicu da je na prvom skupu imao stopostotnu točnost, to je možda algoritam za koji bi se na tom skupu trebalo odlučiti. Logistička regresija također je jednostavan algoritam koji osim klasifikacije izračunava i vjerojatnosti da se određeni podatak nalazi u određenoj kategoriji (i ostali algoritmi mogu izračunati ovu vjerojatnost, ali je logistička regresija za ovo specijalizirana). Točnost ovog algoritma usko je vezana uz svojstvo linearne odvojivosti i ako to svojstvo nije zadovoljeno, logistička regresija u pravilu nije algoritam koji bi trebalo upotrebljavati. Tek kada smo sigurni da je to svojstvo zadovoljeno, odnosno da rezultat koji logistička regresija ostvaruje ne zaostaje za rezultatom koji ostvaruju drugi algoritmi, mogli bismo se odlučiti za njenu upotrebu. Na skupu otto.csv logistička regresija daje solidan rezultat pa ako su nam vjerojatnosti važnije od klasifikacijske točnosti, mogli bismo se odlučiti za ovaj algoritam. SVM ostvaruje izvrstan rezultat na oba skupa i to nije slučajno. On gotovo uvijek ostvaruje dobar rezultat i u većini klasifikacijskih problema trebalo bi mu dati šansu. Mane su mu što nije intuitivan i što je malo sporiji od ostalih algoritama. Za kraj, ostao nam je algoritam slučajne šume koji na prvom skupu ima stopostotnu točnost, a na drugom ostvaruje najbolji rezultat. Ovaj algoritam danas ubrajamo u najefikasnije algoritme i za velik broj klasifikacijskih problema je to prvi algoritam s kojim treba pokušati. Unatoč vrhunskim rezultatima koje ovaj algoritam ostvaruje, on nije uvijek najbolji i prije no što se odlučimo za njega potrebno je isprobati barem još nekoliko algoritama. Za kraj htio bih naglasiti da ovo nisu jedina četiri klasifikacijska algoritma koja postoje. Izabrao sam ih zato što su poznati, reprezentativni i imaju implementaciju u scikit-u. Vrhunske klasifikacijske rezultate ostvaruju još i napredniji algoritmi temeljeni na stablima odluke (na primjer C4.5 algoritam), kao i neuronske mreže, ali njihova bi detaljnija analiza uvelike prekoračila okvire ovoga rada.

Dodatak 1 – Linearna odvojivost

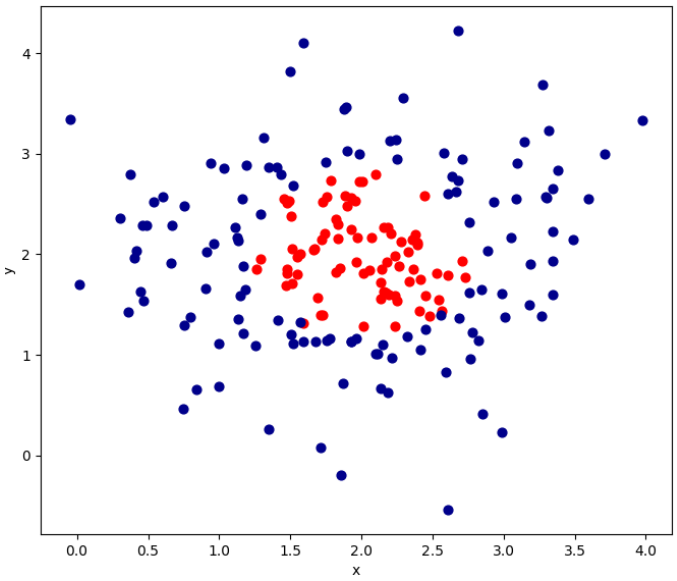
Uzmimo za početak pravac s jednadžbom $w_1 * x + w2 * y + w0 = 0$, gdje su x i y varijable, a w_0, w_1 i w_2 konstante. Taj pravac možemo standardnije zapisati kao $y = a * x + b$ ($w_2 * y$ prebacimo na drugu stranu i podijelimo s $-w_2$).
Sada uzmimo jednadžbu $y = a * x + b * x^2 + c$. Ovo je očito jednadžba parabole.
Uvedemo li sada supstituciju $t = x^2$ dobivamo $y = a * x + b * t + c$. Ovu jednadžbu možemo smatrati jednadžbom triju varijabli, x, y i t, tj. ovu jednadžbu možemo promatrati kao jednadžbu ravnine u 3 dimenzije.
Bit ovog primjera je pokazati da istu jednadžbu možemo smatrati jednadžbom parabole u dvije dimenzije i jednadžbom ravnine u tri dimenzije. Preciznije govoreći istu klasifikaciju možemo obaviti koristeći parabolu određenu značajkama x i y kao i koristeći ravninu određenu značajkama x, y i t (gdje vrijedi $t = x^2$).
Općenito govoreći, istu klasifikaciju možemo obaviti koristeći složeni polinom u manjem broju dimenzija i jednostavnu linearnu funkciju u većem broju dimenzija. Ista logika vrijedi i u drugom smjeru, jednostavna linearna funkcija sa dovoljnim brojem dimenzija može obaviti jednako složenu klasifikaciju kao i složena funkcija pri manjem broju dimenzija. Kako broj značajki odgovara broju dimenzija možemo zaključiti da ukoliko određeni skup podataka ima veći broj značajki, linearne će funkcije biti sposobne obavljati složenije klasifikacije.
Mana koju imaju i rješenja s većim brojem značajki i sa složenijim funkcijama je ta da je njihovo treniranje dosta zahtjevnije u smislu da je potreban veći broj podataka da bi se treniranje kvalitetno obavi.
Iako bi se problemi mogli pokušati rješavati tako da se na umjetan način poveća broj značajki i da se zatim upotrebe linearne funkcije, to u praksi ne daje dobre rezultate*, tako da s ovom logikom ne treba ići predaleko. Ako je skup podataka izvorno linearno odvojiv, onda su linearne funkcije pogodne, ali najčešće ne bi trebalo pokušavati učiniti skup linearno odvojivim proizvoljno dodajući dodatne značajke.

* ovdje se ne misli na algoritme čiji se način rada može promatrati kao povećanje broja značajki, već samo na najednostavnije povećanje broja značajki prije upotrebljavanja algoritma.

Dodatak 2 -kernel (maknit ili doradit)

Transformacije

Ponekad linearna funkcija može savršeno ili dovoljno dobro podijeliti skup na dvije klase. U drugim slučajevima to nije moguće učiniti.



Gornja slika je primjer u kojem podjela pravcem nije moguća. Jedno moguće rješenje je transformacija ulaznih podataka. Ako bismo značajke x1 i x2 zamijenili s $l_0 = x_1^2$, $l_0 = x_2^2$ i $l_0 = x_1 * x_2$ dobili bi tri značajke koje možemo vizulizirati slikom3DSlika. Ove podatke je sada moguće podijeliti ravninom na dvije klase. Dakle transformacijom podataka smo umjesto klasifikacijskog problema kojeg se ne može riješiti linearnom podjelom (pravcem) dobili problem u tri dimenzije koji se može riješiti korištenjem ravnine definirane sa $w_1 * l_1 + w_2 * l_2 + w_3 * l_3 + w_0 = 0$.

Karnel

Za objašnjenje karnela trebat će nam nekoliko matematičkih formula. Kada algoritam uči iz podataka svaki podatak ima svoj doprinos koji ovisi o značajkama koje možemo označiti s x_i i traženoj klasi koju ćemo označiti s y_i. Možemo reći da je doprinos svakog podatka jednak nekoj konstanti koju možemo nazvati a_i (konstanta ne mora biti jednaka za svaki podatak) pomnoženoj s x_i i y_i. Ukupni rezultat učenja bi tada bio suma $\sum a_i * x_i * y_i$. Može se pokazati da je ova suma jednaka vektoru w. Sada recimo da želimo klasificirati neki novi podatak koji ćemo nazvati x_j.. Klasifikacija tog podatka ovisi o značajkama tog podatka i o podacima iz kojih je algoritam učio. Klasifikacija podatka će biti u potpunosti određena umnoškom vektora x_j i sume $\sum a_i * x_i * y_i$. Odnosno ovisit će o umnošku vektora značajki x_j i vektoru parametara w, a ako zanemarimo konstantu w0 onda bi bit algoritma logističke regresije bio utvrđivanje predznaka umnoška ovih vektora tako da ovaj zaključak ustvari nije ništa novo. Na kraju dobijemo da

klasifikacija ovisi o $\sum a_i * y_i * x_i * x_j$. Sada recimo da linearna podjela nije dovoljno dobra već da nam je potrebna nekakva transformacija početnih značajki. Označimo li tu transformaciju s ð možemo pisati da su nove značajke ð (x). Zadnju formulu sada možemo pisati kao $\sum a_i * y_i *$ð $(x_i)*$ ð (x_j). Problem je što je ponekad vrlo teško naći jednostavnu transformaciju podataka koja nam omogućava da podatke razdvojimo na dva skupa ako to nije bilo moguće učiniti linearnom funkcijom osobito ako radimo s podacima koji imaju više od 3 značajke pa ih nije moguće vizualizirati. Primitivno rješenje bi bilo napraviti kompleksnu transformaciju koja će pokriti sve slučajeve kojih se uspijemo sjetiti. Problem kod toga je što takva transformacija za veće skupove može biti memorijski zahtjevna ili zahtjevna za obradu.
Karnel možemo definirati kao funkciju koja za svaku kombinaciju podataka x_i, x_j računa umnožak ð $(x_i)*$ ð (x_j). Dakle karnel možemo izračunati tako da za dva podatka x_i i x_j obavimo transformaciju fi dobijemo ð (x_i) i ð (x_j). i izračunamo njihov umnožak. Prednost koju pruža karnel i koja rješava naš problem je u tome što ga je moguće izračunati bez obavljanja transformacije ð. Odnosno karnel koji predstavlja umnožak transformiranih značajki x_i, i x_j je moguće izračunati bez da se stvarno obavi transformacija. Iako su karneli dosta matematički kompleksni postoje gotovi karneli koje možemo koristiti i koji daju jako dobre rezultate. Najpoznatiji karnel se zove gaussian ili rbf i ne bi ga bilo čak ni teoretski moguće ostvariti običnom transformacijom podataka. Ukratko karnel omogućava da klasificiramo podatke koje nije moguće odvojiti linearnom funkcijom bez da moramo koristiti složene transformacije.

SVM sa defultnim parametrima koristi rbf karnel i daje jako dobre razultate. Kada bi na našem skupu podataka upotrijebili isti kod, ali ovaj put pozvali SVM umjesto logističke regresije dobili bi 100% točnost klasifikacije.
Slika