Sveučilište J. J. Strossmayera u Osijeku
Fakultet elektrotehnike, računarstva i informacijskih tehnologija
Diplomski sveučilišni studij Računarstvo

Stjepan Stojčević

Raspoznavanje ljudskih lica pomoću metode potpornih vektora

Seminar iz kolegija Obrada slike i računalni vid

Mentori: prof. dr. sc. Irena Galić, dr. sc. Marija Habijan

Osijek, 2023.

Raspoznavanje ljudskih lica pomoću metode potpornih vektora

Sažetak

Metoda potpornih vektora (engl. Support Vector Machine, SVM) pripada pod nadzirane metode strojnog učenja koja za dani skup podataka predviđa kojoj klasi pojedini podatak pripada. Ova metoda je učinkovitija od sličnih metoda za klasifikaciju podataka kao što je Perceptron algoritam zbog toga što metoda potpornih vektora pronalazi jedinstvenu optimalnu hiperravninu koja razdvaja podatke tako da geometrgijska margina bude najveća moguća. Metoda potpornih vektora korištena je u ovom projektu u kojem je cilj bio binarno klasificirati slike prema tome smije li se osoba na slici ili se ne smije. Korištene su ugrađene OpenCV klase za detekciju lica na slici. U ovom seminaru teorijski će se opisati algoritam metode potpornih vektora kao i ostale metode koje su korištene u samom projektu, a nakon toka bit će obješnjen svaki zasebni dio projekta.

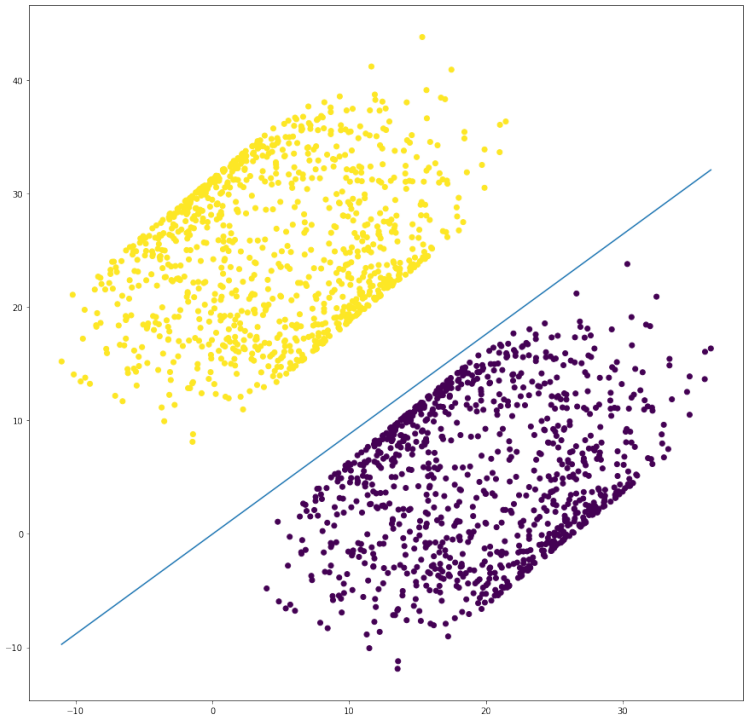
Ključne riječi

Metoda potpornih vektora, binarna klasifikacija, detekcija lica

Sadržaj

1 Metoda Potpornih Vektora

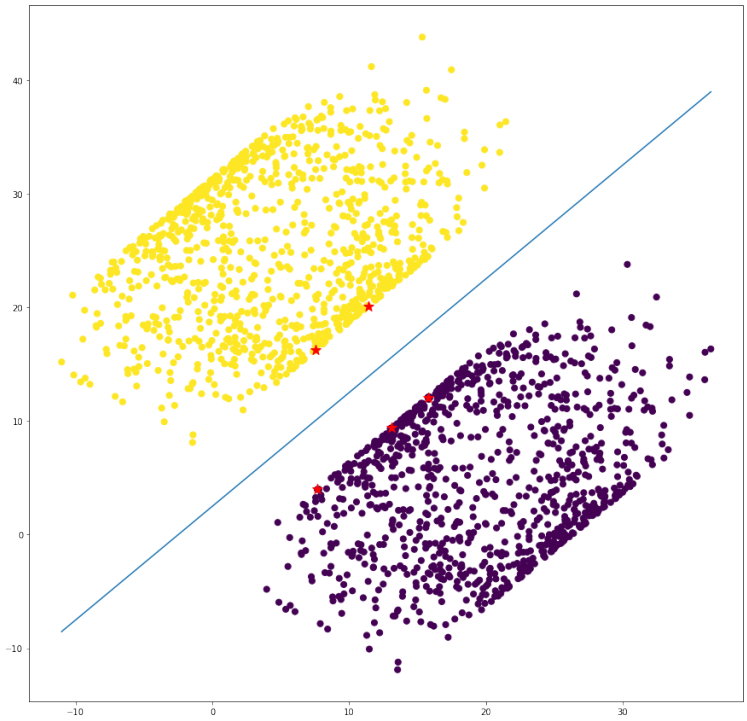
Metoda potpornih vektora je model nadziranog učenja koji se može koristiti za klasifikaciju i regresiju. Osmislili su ga V. N. Vapnik i A. Y. Chervonenkis 1963. godine. Originalno je metoda potpornih vektora bila razvijena za klasifikaciju kao linearni binarni klasifikator koji može rješavati nelinearne probleme te se može proširiti na višeklasne probleme na standardan način. Regresija pomoću metode potpornuh vektora predložena je 1996. godine (V. N. Vapnik, H. Drucker, C. J. C. Burges, L. Kaufman and A. J. Smola.). U usporedbi metode potpornih vektora sa sličnim algoritmima za klasifikaciju podataka kao što je na primjer Perceptron algoritam, metoda potpornih vektora je u velikoj prednosi zbog toga što nam daje hiperravninu koja separira naš skup podatka na način da je geometrijska margina maksimizirana, gdje pod pojmom geometrijska margina podrazumijevamo udaljenost hiperravnine od najbližeg primjerka iz skupa podataka koji je potrebno klasificirati. S druge strane Perceptron algoritam može raditi isključivo na linearno separabilnim podatcima i kao rezultat daje nam hiperravninu bez razmišljanja o tome kolika će geometrijska margina biti. Sljedeća slika prikazuje rezultat Perceptron algoritma na jednom skupu linearno separabilnih podataka. Na tom istom skupu podataka pokazat ćemo rezultat metode potpornih vektora i uočiti razlike. Na slici 1 vidimo podatke iz dvije klase, a to nam govori različita boja točaka. Plavi pravac je hiperravnina koja separira podatke, ali vidljivo je da su podatci iz ljubičaste klase bliže našoj hiperravnini u odnosu na podatke iz žute klase. To nam daje skeptičnost u vezi rezultata na testnom skupu podataka budući da postoji vjerojatnost da neki podatak iz ljubičaste klase bude pogrešno klasificiran odnosno da završi na strani poluravnine na kojoj su podatci iz žute klase.



Slika 1 : Rezultat Perceptron algoritma

Optimalna hiperravnina je jedinstvena odnosno nije moguće pronaći više od jedne hiperravnine tako da geometrijska margina bude maksimizirana. Velika geometrijska margina nam povećava vjerojatnost da će testni podatci koje još nismo vidjeli biti točno klasificirani i upravo zbog toga metoda potpornih vektora ima veću uspješnost u klasifikaciji

podataka. Sljedeća slika prikazuje nam rezultate metode potpornih vektora na istom skupu podataka. Vidljivo je golim okom da je hiperravnina takva da je geometrijska margina maksimizirana. Crvenim zvjezdicama označeni su potporni vektori odnosno točke koje leže točno na geometrijskoj margini i uz pomoć kojih je hiperravnina pronađena. Da se zaključiti da je veća vjerojatnost da će budući testni podatci biti točno klasificirani.



Slika 2 : Rezultat metode potpornih vektora

Za pronalazak optimalne hiperravnine potrebno je postaviti odgovarajući optimizacijski problem. Ako su podaci linearno separabilni hiperravninom kroz ishodište, onda postoje $\Theta \in \mathbb{R}^n$ i $\gamma > 0$ takvi da $y^{(i)}\Theta^T x^{(i)} \geq \gamma,\ i = 1, 2, \ldots, m$ gdje je m broj podatka. Optimizacijski problem možemo postaviti kao

$$\text{argmax}_{\gamma,\Theta} \frac{\gamma}{||\Theta||}$$

$$\text{uz uvjet } y^{(i)}\Theta^T x^{(i)} \geq \gamma,\ i = 1, 2, \ldots, m,$$

što se može svesti na

$$\text{argmin}_{\gamma,\Theta} \frac{1}{2}\left(\frac{||\Theta||}{\gamma}\right)^2$$

$$\text{uz uvjet } y^{(i)}\Theta^T x^{(i)} \geq \gamma,\ i = 1, 2, \ldots, m,$$

odnosno

$$\text{argmin}_{\Theta} \frac{1}{2}||\Theta||^2$$

$$\text{uz uvjet } y^{(i)}\Theta^T x^{(i)} \geq 1,\ i = 1, 2, \ldots, m,$$

Problem ovakve binarne klasifikacije može se generalizirati na hiperravninu koja ne prolazi ishodištem

$$\text{argmin}_{\Theta,\theta_0} \frac{1}{2}||\Theta||^2$$

$$\text{uz uvjet } y^{(i)}(\Theta^T x^{(i)} + \theta_0) \geq 1,\ i = 1, 2, \ldots, m,$$

Ako podaci nisu linearno separabilni, dozvoljavamo kršenje uvjeta, ali uz penalizaciju koju je potrebno minimizirati

$$\text{argmin}_{\Theta,\theta_0,\xi_i} \frac{1}{2}||\Theta||^2 + C\sum_{i=1}^{m} \xi_i$$

$$\text{uz uvjet } y^{(i)}(\Theta^T x^{(i)} + \theta_0) \geq 1 - \xi_i \text{ i } \xi_i \geq 0,\ i = 1, 2, \ldots, m,$$

Dostupne su implementacije nekoliko algoritama: SVC, LinearSVC, NuSVC.

2 Dokumentacija

Ovaj projekt napisan je u Jupyter Notebooku odnosno u programskom jeziku Python. U nastavku će biti objašnjena svaka ćelija koda. Na početku potrebno je učitati sve potrebne biblioteke koje će nam koristiti u izradi projekta. Tu je biblioteka math za račun s matricama, za dobivanje iracionalnih brojeva i za ostale algebarske potrebe. Numpy biblioteka koristi nam za izradu polja i rad s poljima, pandas biblioteka služi za otvranje i rad s skupom podataka koji je u ovom slučaju u .csv formatu. Učitali smo SVC model metode potpornih vektora iz sklearn biblioteke koji će nam odraditi posao nakon završne obrade podataka. Uz pomoć seaborn biblioteke izradit ćemo matricu zabune na samom kraju kako bi vizualizirali uspješnost projekta. Iz biblioteke matplotlib učitali smo pyplot koji će nam koristiti pri svakoj vizualizaciji slika ili matrice zabune. Podatke ćemo skalirati uz pomoć StandardScaler-a iz biblioteke sklearn, a podijelit ćemo ih na trening i testne podatke uz pomoć traintestsplit funkcijom iz biblioteke sklearn. Biblioteka cv2 koristit će nam za korištenje alata OpenCV-a što ćemo koristit za detekciju lica na slici i obradi slike. Za kraj iz biblioteke sklearn učitali smo confusionmatrix što će nam dati matricu zabune na samom kraju projekta.

```
import math
import numpy as np
import pandas as pd
from sklearn.svm import SVC
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import cv2
from sklearn.metrics import confusion_matrix
```

Kod 1. Učitavanje biblioteka

Uz pomoć koda 2 učitali smo "dataset.csv" skup podataka koji je u sebi ima stupac koji govori smije li se osoba ili ne i stupac koji sadrži polja koje kao elemente ima vrijednosti pixela od 0 do 255 što nam predstavlja sliku. Sa zadnjom linijom koda 2 prikazano je kako izgledaju prvih par podataka nakon čisćenja.

```
data = pd.read_csv('dataset.csv')
data.head()
```

Kod 2. Učitavanje skupa podataka

Zbog ilustracije i kasnijeg lakšeg razumijevanja dobro je prikazati kakvo to naši podatci točno izgledaju. Označili smo s riječi "smile" svaki podataka na kojemu se osoba smije i s "no smile" svaki podatak na kojemu se osoba ne smije. Uz pomoć sljedeće for petlje uspjeli smo od svake emocije koju imamo prikazati samo jedan primjerak, a sudeći da mi imamo samo dvije klase koristili smo brojač c koji prati koliko različitih klasa smo prošli.

```
emotion_label_to_text={0:'smile',1:'no smile'}
fig = plt.figure(1, (14,14))
k = c = 0
for i in data["emotion"]:
    if i == 0 or i == 1:
        c += 1
    for j in range(1):
        k += 1
        px = data[data.emotion==i].pixels.iloc[k]
        px = np.array(px.split(' ')).reshape(48, 48).astype('float32')
        ax = plt.subplot(7,7, k)
        ax.imshow(px , cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(emotion_label_to_text[i])
        plt.tight_layout()
    if c == 2:
        break
```

Kod 3. Prikaz skupa podataka

Rezultat koda 3 izgleda kao na sljedećoj slici 3. Osoba na lijevom djelu slike se ne smije i ima oznaku "no smile" kako i treba biti dok se osoba na desnom dijelu slike smije i ima oznaku "smile".

Slika 3 : Prikaz slika iz skupa podataka

U nastavku slijedu pretvaranje pandas forme naših podataka u formu koju možemo koristiti za naš problem. Pretvorit ćemo stupce koje smo do sada imali u matricu X i polje Y. U matrici X bit će polja koja određuju svaku pojedinu sliku s brojevima od 0 d 255 kao pixelima. Polje Y sprema podatak smije li se osoba na pojedinoj slici ili ne.

```
X = data.pixels.apply(lambda x: np.array(x.split(' ')).astype('float32'))
X = np.stack(X, axis=0)
Y = np.array(data['emotion'])
```

Kod 4. Pretvorba podataka

Na sljedećem kodu prikazana je matrica X.

```
array([[  4.,   0.,   0., ...,  30.,  29.,  30.],
       [ 77.,  78.,  79., ..., 125.,  67.,  68.],
       [ 85.,  84.,  90., ...,  58.,  73.,  84.],
       ...,
       [ 50.,  36.,  17., ..., 223., 221., 216.],
       [178., 174., 172., ...,   0.,   0.,   0.],
       [ 30.,  28.,  28., ...,  35.,  30.,  28.]], dtype=float32)
```

Kod 5. Prikaz matrice koja definira svaku pojedinu sliku

U kodu broj 6 učitavamo ugrađene OpenCV klase za detekciju lica na slici. Ove klase ćemo koristiti u sljedećem kodu u kojemu ćemo obrađivati lica na slikama. Bez ovih klasa za pronalaz lica i očiju na slici ne bismo mogli dovršiti ovaj projekt.

```
eyeCascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_eye.xml')
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

Kod 6. Ugrađene OpenCV klase za detekciju lica na slici

Slijedi nam funkcija koju koristimo za obradu lica. Funkcija prima sliku koju će obraditi.Prvo spremimo rezoluciju slike odnosno njenu širinu i visinu i izračunamo gdje je središte slike što će nam kasnije biti potrebno zbog rotacije. Onda od očiju na svakoj slici stvaramo varijable uz pomoć ugrađene OpenCV klase kako bi ih mogli kasnije koristiti. Nakon toga slijedi proces razdvajanja lijevog i desnog oka u dvije zasebne varijable. Provjeravamo koje je oko lijevo, a koje desno i onda pronalazimo središta oba oka. Izračunamo kut između očiju kako bi mogli definirati matricu uz pomoć koje ćemo rotirati sliku za taj kut oko centra same slike. Nakon toga s drugom ugrađenom klasom detektiramo lice i vratimo sliku kao rezultat funkije.

```
def faceAlignment(image):
  size = image.shape
  height, width = image.shape[:2]
  center = (width // 2, height // 2)
  eyes = eyeCascade.detectMultiScale(image, 2, 10)
  if len(eyes) == 2 :
    i=0
    for (e1 , e2,  e3,  e4) in eyes:
      if i == 0:
        eye1 = (e1, e2, e3, e4)
      else:
```

```
            eye2 = (e1, e2, e3, e4)
        i += 1
    if eye1[0] > eye2[0]:
        leftEye = eye2
        rightEye = eye1
    else:
        leftEye = eye1
        rightEye = eye2

    rightEyeCenter = (int(rightEye[0] + (rightEye[2]/2)),
    int(rightEye[1] + (rightEye[3]/2)))
    rightEyeX = rightEyeCenter[0]
    rightEyeY = rightEyeCenter[1]

    leftEyeCenter = (int(leftEye[0] + (leftEye[2] / 2)),
    int(leftEye[1] + (leftEye[3] / 2)))
    leftEyeX = leftEyeCenter[0]
    leftEyeY = leftEyeCenter[1]

    deltaX = rightEyeX - leftEyeX
    deltaY = rightEyeY - leftEyeY
    angle = np.arctan(deltaY/deltaX)
    angle = (angle * 180) / np.pi

    M = cv2.getRotationMatrix2D(center, (angle), 1.0)
    image = cv2.warpAffine(image, M, (width, height))

    faces = faceCascade.detectMultiScale(image, 2, 10)
    if len(faces) > 0 :
        (x,y,width,height) = faces[0]
        image = image[y:y+height, x:x+width]
image = cv2.resize(image,size)
return image
```

Kod 7. Funkcija za obradu lica

Slijedi sama primjena prijašnje funkije na svaku od slika iz našeg skupa podataka odnosno na svaki redak matrice X.

```
XNew = []
for i in range(X.shape[0]):
image = np.array(X[i]).reshape(48,48)
image = np.array(image, dtype='uint8')
image = faceAlignment(image)
XNew.append(np.array(image).reshape(48*48))
X = np.array(XNew)
```

Kod 8. Primjena funkcije za obradu lica

Pomoću ugrađene funkcije traintestsplit podijelit ćemo podatke na podatke na trening i na podatke za testiranje. 75% podatka spremit ćemo u trening podatke, a 25% spremit ćemo u skup podataka za testiranje. Nakon toga skalirat ćemo podatke radi boljih rezultata uz pomoć ugrađene funkcije StandardScaler.

```
Xtrain, Xtest, ytrain, ytest = traintestsplit(X, Y, testsize=0.25, randomstate=0)
scaler = StandardScaler()
Xtrain = scaler.fittransform(Xtrain.astype(np.float32))
Xtest = scaler.transform(Xtest.astype(np.float32))
```

Kod 9. Podjela podataka na trening i test podatke i skaliranje

Izradimo model klasifikacije i naučimo ga da radi na trening podatcima. Nakon toga provjerimo kolika mu je uspješnost na testnim podatcima. Rezultat je oko 77% što je zadovoljavajuće.

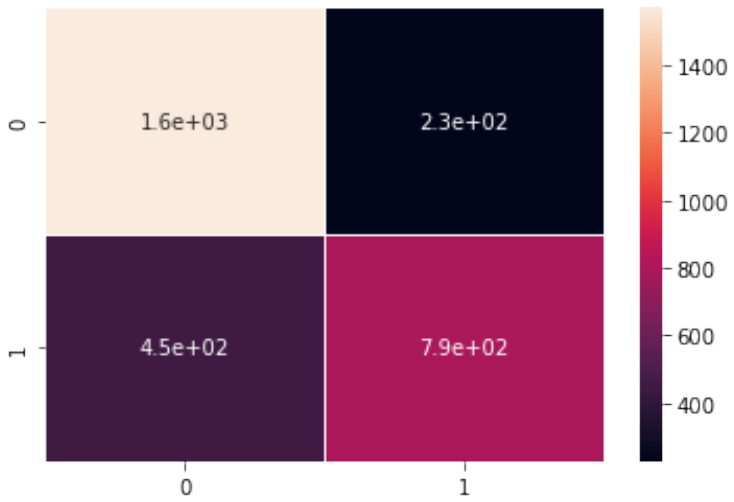
```
clf = SVC().fit(Xtrain,ytrain)
clf.score(Xtest , ytest)
```

Kod 10. Izrada modela klasifikacije

Sljedećim kodom prikazat ćemo matricu zabune prema kojoj možemo vizualizirati uspješnost dobivenih rezultata. Na slici 3 koja prikazuje matricu zabune vidimo da najviše elemenata ima na glavnoj dijagonali što nam govori da su podatci dobro klasificirani.

```
yhat = clf.predict(Xtest)
cfmatrix = confusionmatrix(ytest, yhat)
sns.heatmap(cfmatrix, linewidths=1, annot=True)
plt.show()
```

Kod 11. Matrica zabune



Slika 4 : Matrica zabune

3 Zaključak

Metoda potpornih vektora primjenjiva je metoda u klasifikaciji i daje zadovoljavajuće rezultate što smo i pokazali u ovom projektu. Uz sređen skup podataka u kojemu su vrijednosti skalirane s nekim od dostupnih ugrađenih modela metode potpornih vektora mogu se očekivati dobri rezultati bez obzira o čemu taj skup podataka zapravo govori. U ovom projektu ima mjesta za napredak kojim bi se dobili još bolji rezultati na način da su slike veće rezolucije, da su slike u boji, da smo u obradi lica koristili i usta, a ne samo oči, kao mjerilo smije li se osoba ili ne i tako dalje. Uz seminar slijedi i python bilježnica u kojoj je projekt napisan i dodatno pojašnjen.

Literatura

[1] Facial Expression Recognition Using Support Vector Machines, dostupno na: (https://www.cs.cmu.edu/~pmichel/publications/Michel-FacExpRecSVMAbstract.pdf)

[2] Support Vector Machines - Scikit-learn, dostupno na: (https://scikit-learn.org/stable/modules/svm.html)

[3] Sveučilište J. J. Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija, Obrada slike i računalni vid - Predavanja i vježbe

[4] Sveučilište J. J. Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija, Računarstvo usluga i analiza podataka - Predavanja i vježbe