SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 418

WEB APLIKACIJA ZA RASPOZNAVANJE RASPOLOŽENJA PJESAMA S USLUGE SPOTIFY KORISTEĆI STROJNO UČENJE

Katarina Bošnjak

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 418

WEB APLIKACIJA ZA RASPOZNAVANJE RASPOLOŽENJA PJESAMA S USLUGE SPOTIFY KORISTEĆI STROJNO UČENJE

Katarina Bošnjak

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 418

Pristupnica:	**Katarina Bošnjak (0036524178)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Web aplikacija za raspoznavanje raspoloženja pjesama s usluge Spotify koristeći strojno učenje**

Opis zadatka:

Cilj završnog rada je usporedba više algoritama strojnog učenja na problemu raspoznavanja raspoloženja pjesme te izrada web aplikacije koja će omogućiti korištenje najboljeg modela raspoznavanja. Aplikacija treba omogućiti registraciju i prijavu korisnika. Prijavljeni korisnik može izabrati između prikaza raspoloženja najslušanijih pjesama sa svog računa na web usluzi Spotify, prikaza raspoloženja pjesama određenog umjetnika te prikaza raspoloženja jedne pjesme. Podatke o korisniku/umjetniku/pjesmi potrebno je dohvatiti koristeći Spotifyjev API. Umjetnika i pjesmu korisnik može unijeti preko njihovog imena i URI-ja. Za učenje modela potrebno je prikupiti odgovarajući skup podataka također putem Spotify API-ja. Skup podataka o pjesmama treba biti zasnovan na metapodacima (parametrima) pjesama dostupnima na Spotifyju kao što su plesnost, valencija, energičnost, tempo, itd. U radu je potrebno usporediti više modela algoritama strojnog učenja (npr. neuronske mreže, slučajne šume, stroj s potpornim vektorima. Implementaciju aplikacije je potrebno napraviti u programskom jeziku po vlastitom izboru.

Rok za predaju rada: 10. lipnja 2022.

Zahvaljujem roditeljima i obitelji na bezuvjetnoj podršci te zahvaljujem mentoru izv. prof. dr. sc. Alanu Joviću na pomoći pri izradi rada.

SADRŽAJ

1. Uvod

Aplikacija Spotify krajem godine za svakog korisnika objavi njegov "Spotify wrapped". To je analiza najčešće slušanih pjesama i raspoloženja tih pjesama kroz godinu.

U ovom radu prikazat ću svoje rješenje izrade aplikacije koja za korisnika Spotifyja može izvući najčešća raspoloženja najslušanijih pjesama korištenjem algoritama strojnog učenja.

Ovaj rad se sastoji od 4 dijela.

U prvom dijelu opisuje se način prikupljanja skupa podataka za učenje.

U drugom dijelu opisuju se algoritmi strojnog učenja koji su korišteni: neuronska mreža, slučajna šuma i stroj potpornih vektora.

U trećem dijelu opisani su modeli prethodnih algoritama te je napravljena usporedba tih algoritama.

U četvrtom dijelu opisana je web aplikacija.

Kroz ovaj rad se nadam više naučiti o algoritmima strojnog učenja te o izradi poslužiteljskog dijela web aplikacije. Smatram da bi najveći problem moglo biti postizanje što veće točnosti u prepoznavanju raspoloženja pjesama u modelima strojnog učenja.

2. Opis skupa podataka

Skup podataka je kreiran povlačenjem meta podataka pjesama sa Spotify API-ja. Za prikupljanje podataka korištena je biblioteka spotipy za Python.

Za pristup Spotify API-ju bilo je potrebno kreirati račun te tako dobiti CLIENTID i CLIENTSECRET koji se koristi kako bi se stvorio objekt SpotifyClientCredentials. Zatim se stvara objekt kojim će se dalje slati zahtjevi Spotify API-ju. (slika 2.1)

```
In [4]: import spotipy
        import csv


        from spotipy.oauth2 import SpotifyClientCredentials

        auth_manager = SpotifyClientCredentials(CLIENT_ID, CLIENT_SECRET)
        spotify = spotipy.Spotify(auth_manager=auth_manager)
```

Slika 2.1: Kreiranje objekata za pristup Spotify API-ju

Pronađene su liste pjesama određenog raspoloženja: sretne, tužne, smirene i energične te njihovi URI-ji.Ti URI-ji koriste se kako bi se dohvatili URI-ji pjesama u funkciji na slici 2.2. Smirenih pjesama je 393 u skupu podataka, tužnih je 100, sretnih je također 100, a energičnih je 50.

```
In [5]: def get_playlist_tracks(uri):
            results = spotify.playlist_items(uri)
            tracks = [song['track']['uri']  for song in results['items']]
            while results['next']:
                results = spotify.next(results)
                tracks.extend([song['track']['uri']  for song in results['items']])
            return tracks
```

Slika 2.2: Funkcija za dohvat pjesama s listi pjesama pomoću URI-ja

Nakon toga su dohvaćene značajke za te pjesme koristeći funkcije na slici 2.3.

Onda se iz listi miču duplikati koristeći funkciju na slici 2.4.

Podaci su označeni tako što im je dana oznaka liste pjesmi iz koje dolaze. Na primjer ako je pjesma iz liste pjesama koja je sretna, pjesma će biti označena kao 'happy' što je prikazano na slici 2.5.

```
In [6]: def get_features(song_uris):
            gen = chunks(song_uris, 100)
            features = []
            for g in gen:
                features.extend(spotify.audio_features(g))

            return features
```

```
In [7]: def chunks(lst, n):
            """Yield successive n-sized chunks from lst."""
            for i in range(0, len(lst), n):
                yield lst[i:i + n]
```

Slika 2.3: Funkcija za dohvat pjesama s listi pjesama pomoću URI-ja

```
In [8]: def remove_duplicates(list):
            seen = set()
            lst = [l for l in list if l['id']  not in seen and not seen.add(l['id'])]
            return lst
```

Slika 2.4: Funkcija za micanje duplikata

Nakon označavanja podaci su zapisani u datoteku csv formata.

Značajke pjesmi su sljedeće: plesnost, energičnost, ključ, glasnoća, mod, rječitost, akustičnost, instrumentalnost, živost, valencija, tempo, tip, id, URI, poveznica prema listi kojoj pripadaju, poveznica prema analizi, trajanje, oznaka vremena, te dodana oznaka raspoloženja. (slika 2.6)

```
In [438]: def add_mood(dict, mood):
              for d in dict:
                  d |= {'mood': mood}
```

```
In [439]: add_mood(features_calm, 'calm' )
          add_mood(features_happy, 'happy' )
          add_mood(features_sad, 'sad' )
          add_mood(features_energetic, 'energetic' )
```

Slika 2.5: Dodavanje oznake podacima

```
features = []
features.extend(features_calm)
features.extend(features_sad)
features.extend(features_happy)
features.extend(features_energetic)
```

```
keys = features[0].keys()
keys
```

```
dict_keys(['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness',
'valence', 'tempo', 'type', 'id', 'uri', 'track_href', 'analysis_url', 'duration_ms', 'time_signature', 'mood'])
```

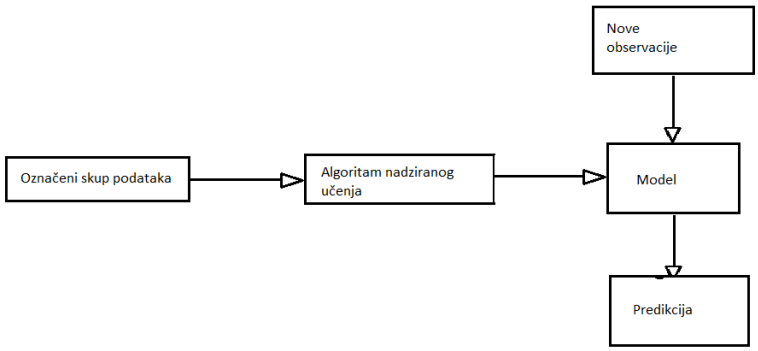
Slika 2.6: Prikaz značajki

3. Algoritmi strojnog učenja

Strojno učenje je grana umjetne inteligencije i računarske znanosti koja se fokusira na korištenje algoritama i podataka da bi se imitirao način na koji ljudi uče. (2)

Strojno učenje možemo podijeliti na tri dijela: nadzirano, nenadzirano i podržano.

U nadziranom strojnom učenju koristi se skup podataka koji je označen (slika 3.1). Dakle podaci su u obliku $(\vec{x}, y)$, gdje je $\vec{x}$ vektor ulaznih značajki podataka, a y je oznaka. Potrebno je naći preslikavanje: $\hat{y} = f(\vec{x})$. Ako je y kontinuirana varijabla onda ovo učenje nazivamo regresija, a ako je diskretna klasifikacija. (3)



Slika 3.1: Nadzirano strojno učenje

U nenadziranom učenju se koristi neoznačen skup podataka. Potrebno je pronaći skrivene uzorke i grupiranja podataka bez ljudske intervencije. Koristi se u eksplorativnoj analizi podataka, raspoznavanju uzoraka i slika, smanjenju značajki u modelu itd. Nenadzirano strojno učenje dijeli se na smanjenje dimenzionalnosti, grupiranje (engl. clustering) i micanje stršećih vrijednosti (engl. outliers). (4)

Podržano strojno učenje, kao i nadzirano strojno učenje koristi označeni skup podataka, ali se model ne uči koristeći taj skup podataka, već model uči tako da se nagradi

za željeno ponašanje, a kazni za neželjeno. (4)

3.1. Neuronska mreža

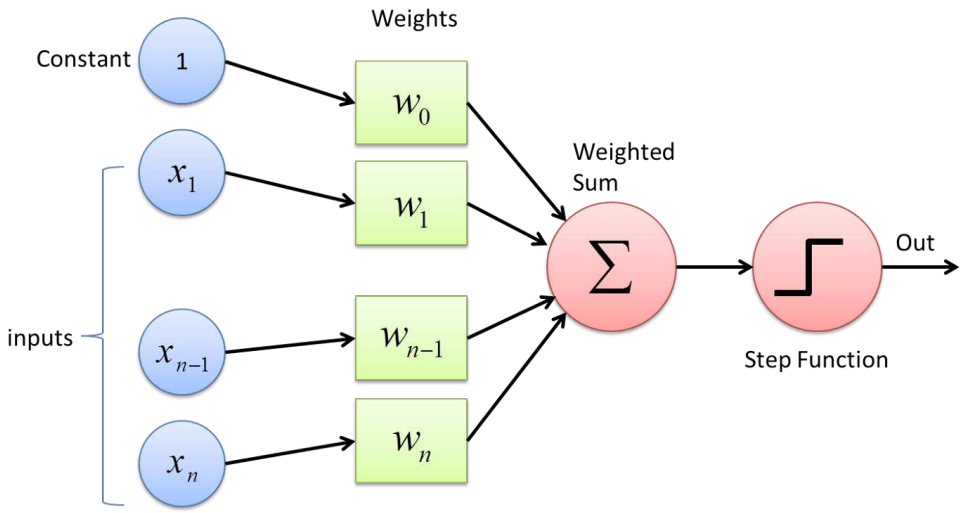
Neuronske mreže oponašaju ljudski mozak što omogućuje programima da prepoznaju uzorke i rješavaju probleme umjetne inteligencije i strojnog učenja.

Umjetni neuron je osnovna građevna jedinica neuronske mreže. Ima svoju težinu i povezan je s drugim neuronima. McCulloch-Pitts su 1943. definirali jednostavan model biološkog neurona TLU-perceptron (slika 3.2). Prvo se izračuna akumulirana vrijednost

$$net = \sum_{i=1}^{n} x_i * w_i + w_0 \tag{3.1}$$

Gdje je x_i ulaz, w_i težina tog ulaza, a w_0 je pomak. Akumulirana vrijednost se propušta kroz prijenosnu funkciju čime nastaje izlazna vrijednost. (3)

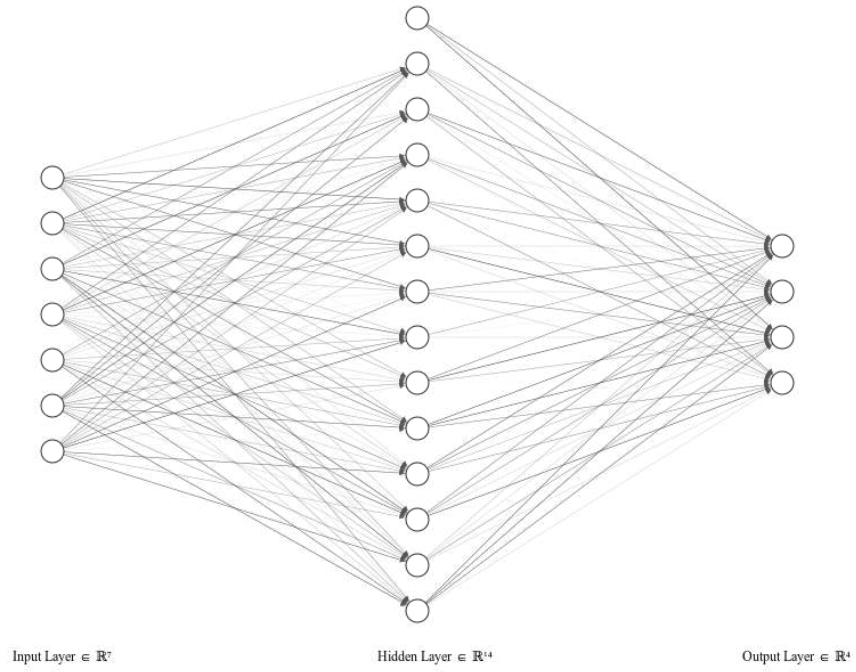
$$o = step(net)$$



Slika 3.2: TLU - perceptron (5)

Neuronska mreža definira se kao više neurona koji su međusobno povezani. Umjetne neuronske mreže sastoje se od slojeva umjetnih neurona, jedan ulazni sloj, jedan ili više skrivenih slojeva i jedan izlazni sloj (slika 3.3). Svaki sljedeći sloj kao ulaze prima izlaze prethodnog sloja. Na početku se težine neuronske mreže postavljaju na slučajne vrijednosti. Neuronske mreže uče u procesu propagiranja prema unatrag (engl. backpropagation), tako što određuju na koji način promijeniti težine i pomake. U tom

procesu neuronska mreža određuje koliko se dobiveni rezultat razlikuje od očekivanog rezultata, a onda mijenja težine i pomake po neuronima i vezama između neurona prema unatrag. (6)



Slika 3.3: Arhitektura neuronske mreže

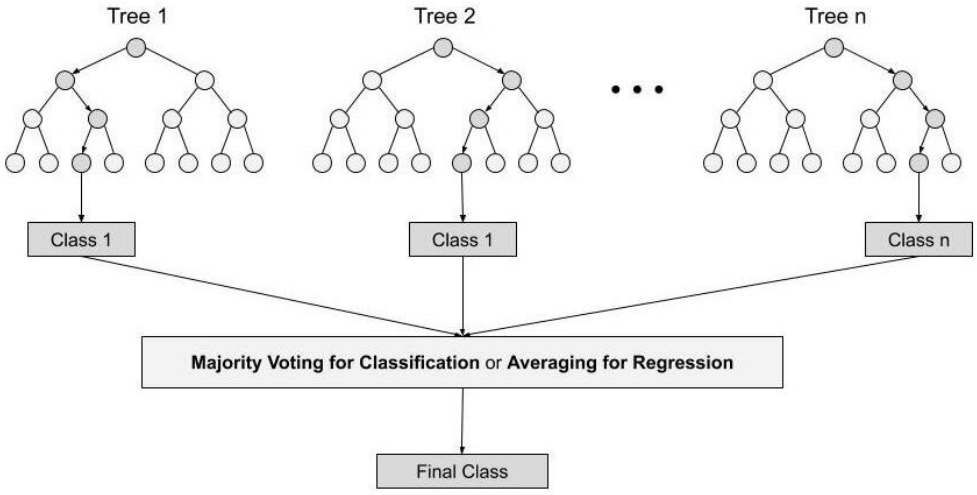
3.2. Slučajne šume

Slučajna šuma je algoritam nadziranog strojnog učenja i često se koristi u problemima klasifikacije i regresije. Gradi više stabala odluke na različitim uzorcima i uzima glas većine za klasifikaciju i prosjek za regresiju. (7)

Stablo odluke je algoritam nadziranog strojnog učenja koji gradi stablo čiji unutarnji čvorovi odgovaraju značajkama (atributima), grane ispred svakog čvora odgovaraju vrijednostima dotične značajke, a listovi su odluke. Budući da je izgradnja stabla NP-težak problem, koristi se heuristički pristup. U korijenskom čvoru potrebno je izabrati između n značajki, u idućem koraku između $n - 1$ itd. Želi se izabrati značajka koja najbolje diskriminira po oznaci među primjerima u skupu za učenje. Postoji više kriterija za odabir značajke: entropija, informacijska dobit, Gini index, Gini Ratio, smanjenje varijance, χ^2. Postoji i više algoritama za izgradnju stabla: ID3, C4.5, CART, CHAID, MARS. (8)

Velik problem stabala odluke je prenaučenost. Kako bi se spriječio problem prenaučavanja stablo se može podrezivati, tako da mu se zada maksimalna dubina ili podijeliti skup za učenje na dva dijela, skup za učenje i skup za validaciju te onda podrezivati stablo da bi se optimirala točnost na skupu za validaciju. (4)

Problem prenaučavanja stabla za odluke se može riješiti i algoritmom slučajnih šuma. Algoritam slučajnih šuma radi tako da gradi manja stabla od slučajno generiranih podskupova ulaznih podataka. Poslije kombinira ta stabla i donosi prosječnu odluku u slučaju regresije ili uzima glas većine u slučaju klasifikacije. Taj postupak se naziva Bagging. (7) (slika 3.4)



Slika 3.4: Slučajna šuma (7)

3.3. Stroj potpornih vektora

Stroj potpornih vektora je algoritam nadziranog strojnog učenja koji se može koristiti u klasifikaciji i regresiji. Algoritam radi tako da se svaku opservaciju tj. podatak iz skupa podataka smjesti u n-dimenzionalni prostor gdje je n broj značajki koje podatak ima i onda traži hiperravninu koja razdvaja te podatke. Potporni vektor su koordinate svake opservacije. (9)

SVM pokušava naći granicu između klasa tako da savršeno klasificira sve primjere iz skupa za učenje. To rezultira prenaučenim modelom. Da bi riješili ovaj problem Cortes i Vapnik su 1995. smislili su SVM s "mekom marginom" koji dozvoljava da se neke opservacije iz skupa za učenje krivo klasificiraju da se dobije bolji model. Kada se određuje hiperravnina želi se povećati udaljenost između te hiperravnine i klasa te maksimizirati broj opservacija koje se ispravno klasificiraju. Da bi se to napravilo koristi se parametar C. Parametar C je kazna za neispravno klasificiran primjer. Ako je C mali onda se izabere hiperravnina s većom udaljenosti od klasa, a ako je C velik onda SVM pokušava minimizirati broj neispravno klasificiranih primjera zbog veće kazne. Tako se dobije hiperravnina s manjom marginom. (18)

Parametar gamma određuje koliki je utjecaj jednog primjera iz skupa za učenje. Manje vrijednosti parametra gamma znače veći radijus sličnosti zbog čega se više točaka grupira skupa, dok veći parametar gamma znači da točke moraju biti bliže jedna drugoj da bi pripadale istoj grupi. (19)

Algoritmi stroja potpornih vektora koriste skup matematičkih funkcija koji se naziva jezgra. Jezgra se definira kao funkcija

$$K(\vec{x}) = \begin{cases} 1 & \text{if } \|\vec{x}\| \leq 1 \\ 0 & \text{if } \|\vec{x}\| > 1 \end{cases}$$

Vrijednost ove funkcije je 1 oko točke $\vec{x}$ unutar radijusa duljine 1, a izvan je 0. Primjeri jezgri su linearna, polinomna, RBF itd. Jezgre se koriste kako bi se hiperravnina većih dimenzija za SVM mogla odrediti uz manju cijenu izračuna. (10)

4. Modeli strojnog učenja

4.1. Priprema podataka

Korišten je skup podataka opisan u prvom poglavlju. Skup podataka ima 643 označene pjesme. Za učenje modela su uzete značajke plesnost, energičnost, akustičnost, instrumentalnost, živost, valencija i tempo. (slika 4.1)

```
In [48]: import csv

         col_features = np.array([ *df.columns[0:2], *df.columns[6:11], *df.columns[-2:len(df.columns)-2]])
         print(col_features)
         print(len(col_features))
         with open("features.csv", 'w', newline='') as output_file:
             dict_writer = csv.DictWriter(output_file, col_features)
             dict_writer.writeheader()

         ['danceability' 'energy' 'acousticness' 'instrumentalness' 'liveness'
          'valence' 'tempo']
         7
```

Slika 4.1: Uzimanje najvažnijih značajki

U idućem koraku su podaci skalirani u interval između 0 i 1 koristeći MinMaxScaler iz biblioteke sklearn. (slika 4.2)

(11)

```
In [9]: X= MinMaxScaler().fit_transform(df[col_features])
        X2 = np.array(df[col_features])
        Y = df['mood']
```

Slika 4.2: Skaliranje

Korištena je sljedeća transformacija

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

$$X_{scaled} = X_{std} * (max - min) + min$$

gdje su min i max minimalna i maksimalna vrijednost značajke u skupu.

Nakon toga je napravljeno kodiranje oznaka u brojčane vrijednosti koristeći LabelEncoder iz biblioteke sklearn (slika 4.3).

```
In [51]: encoder = LabelEncoder()
encoder.fit(Y)
encoded_y = encoder.transform(Y)
target = pd.DataFrame({'mood':df['mood'].tolist(),'encode':encoded_y}).drop_duplicates().sort_values(['encode'],ascending=True)
target.to_csv('encoder.csv',index=False)
target
```

Out[51]:

	mood	encode
0	calm	0
593	energetic	1
493	happy	2
393	sad	3

Slika 4.3: Kodiranje oznaka

Također je napravljena podjela podataka na podatke za učenje i za testiranje (slika 4.4).

```
In [11]: X_train,X_test,Y_train,Y_test =
train_test_split(X,encoded_y,test_size=0.3,random_state=15)
```

Slika 4.4: Podjela skupa podataka

4.2. Model neuronske mreže

Model neronske mreže kreiran je koristeći Sequential API iz biblioteke Keras. Napravljena je instanca klase Sequential i dodavani su slojevi.

Prvi sloj je ulazni sloj koji se sastoji od $n = 7$ neurona zato što toliko ima početnih značajki.

Nakon toga dolazi skriveni sloj koji ima $n * 2 = 14$ neurona.(12) Taj sloj je tipa Dense i računa net koristeći formulu 3.1 te onda koristi aktivacijsku funkciju ReLU: (13)

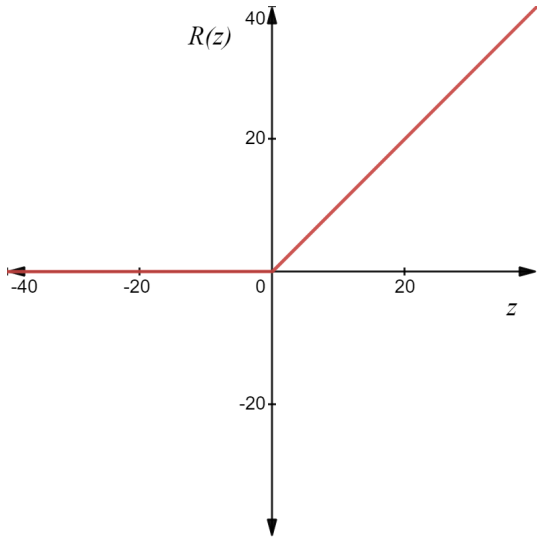
$$R(z) = max(z, 0)$$

Graf funkcije je prikazan na slici 4.5.

Poslije skrivenog sloja dodan je sloj Dropout koji pomaže u sprječavanju prenaučenosti. Ulazne vrijednosti ovog sloja se postavljaju na 0 s frekvencijom $rate$, u ovom slučaju $rate = 0.1$. Svi ostali ulazi se skaliraju faktorom $\frac{1}{1-rate}$ kako bi zbroj svih ulaza ostao isti. (15)

Idući sloj je izlazni sloj koji ima 4 neurona zato što imaju 4 klase. Aktivacijska funkcija ovog sloja je softmax. Softmax funkcija skalira vektor od K realnih vrijednosti u vektor K vrijednosti koje u zbroju daju 1. Te vrijednosti su vrijednosti između 0 i 1 pa se mogu protumačiti kao vjerojatnosti svake klase. (14)

$$\sigma(\vec{z})_i = \frac{e^z_i}{\sum_j^K e^z_j}$$



Slika 4.5: Graf funkcije ReLU

Model koristi rijetku kategoričku unakrsnu entropiju kao funkciju gubitka. Funkcija gubitka (engl. loss function) procjenjuje koliko je model prigodan za zadani skup podataka.

$$Loss = -\sum_{i=1}^{n} y_i * \log \hat{y}_i$$

Gdje je n veličina izlaza, y_i je očekivana vrijednost izlaza, a $\hat{y}_i$ je dobivena vrijednost izlaza. (16)

Tijekom svake epohe učenja potrebno je promijeniti težine u neuronskoj mreži kako bi se bolje prilagodile skupu podataka. U ovom modelu je za to korišten optimizator ADAM (engl. ADAptive Moment estimation - procjena adaptivnog momenta).

Kod modela neuronske mreže prikazan je na slici 4.6.

```
In [34]: n = len(col_features)

In [44]: def base_model(hiddenLayerOne=n*2, dropout=0.1, learnRate=0.01,
                       activation_function1 = 'relu'):

             model = Sequential()
             model.add(tf.keras.Input(shape=(n,)))
             model.add(Dense(hiddenLayerOne,input_dim=n,activation=activation_function1))
             model.add(Dropout(dropout))

             model.add(Dense(4,'softmax'))


             model.compile(loss='sparse_categorical_crossentropy',optimizer=Adam(learning_rate=learnRate),
                           metrics=['accuracy'])
             return model
```

Slika 4.6: Model neuronske mreže

Nadalje je napravljen objekt KerasClassificator gdje je definiran broj epoha i broj uzoraka kojima se uči mreža. Ovdje je $batchsize = 32$ što znači da će se prvo uzeti

32 uzorka kojima će se učiti mreža (prilagođavati težine), pa onda opet 32, i tako kroz 300 epoha. (slika 4.7)

```
In [55]: estimator = KerasClassifier(build_fn=base_model,epochs=300,batch_size=32,verbose=1, callbacks = None)
```

Slika 4.7: Klasifikator

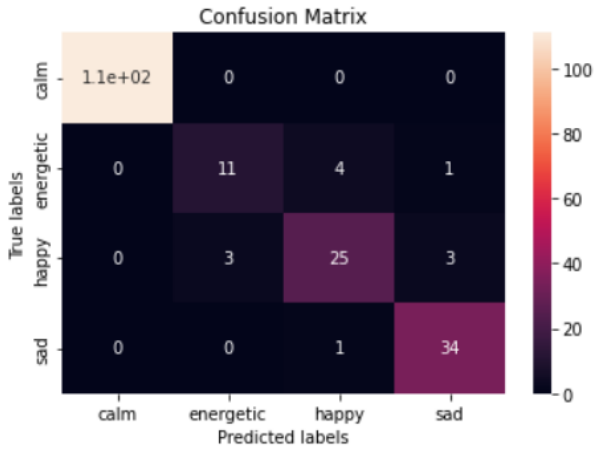
Nakon učenja modela dobivena je točnost od 0.9778 i loss od 0.0656. (slika 4.8)

```
n [56]: estimator.fit(X_train,Y_train)
        y_preds = estimator.predict(X_test)
15/15 [==============================] - 0s 999us/step - loss: 0.0691 - accuracy: 0.9689
Epoch 292/300
15/15 [==============================] - 0s 996us/step - loss: 0.0703 - accuracy: 0.9733
Epoch 293/300
15/15 [==============================] - 0s 927us/step - loss: 0.0602 - accuracy: 0.9733
Epoch 294/300
15/15 [==============================] - 0s 1ms/step - loss: 0.0691 - accuracy: 0.9733
Epoch 295/300
15/15 [==============================] - 0s 997us/step - loss: 0.0646 - accuracy: 0.9778
Epoch 296/300
15/15 [==============================] - 0s 1ms/step - loss: 0.0710 - accuracy: 0.9711
Epoch 297/300
15/15 [==============================] - 0s 1ms/step - loss: 0.1028 - accuracy: 0.9622
Epoch 298/300
15/15 [==============================] - 0s 1ms/step - loss: 0.0655 - accuracy: 0.9756
Epoch 299/300
15/15 [==============================] - 0s 1ms/step - loss: 0.0613 - accuracy: 0.9822
Epoch 300/300
15/15 [==============================] - 0s 1ms/step - loss: 0.0656 - accuracy: 0.9778
```

Slika 4.8: Učenje modela

Budući da broj uzoraka svake klase nije jednak napravljena je matrica konfuzije (slika 4.9). Za svaku klasu se računa koliko primjera iz te klase je model točno klasificirao, a koliko je tih primjera zamijenio s drugim klasama. Ovaj model najčešće miješa energične i sretne pjesme, vjerojatno zato što energične i sretne pjesme imaju sličnosti i postoje preklapanja između njihovih značajki.

```
In [58]: cm = confusion_matrix(Y_test,y_preds)
         ax = plt.subplot()
         sns.heatmap(cm,annot=True,ax=ax)

         labels = target['mood']
         ax.set_xlabel('Predicted labels')
         ax.set_ylabel('True labels')
         ax.set_title('Confusion Matrix')
         ax.xaxis.set_ticklabels(labels)
         ax.yaxis.set_ticklabels(labels)
         plt.show()

         print("Accuracy Score",accuracy_score(Y_test,y_preds))
```




```
Accuracy Score 0.9378238341968912
```

Slika 4.9: Matrica konfuzije

4.3. Model slučajne šume

Model slučajne šume napravljen je s predefiniranim parametrima. Nakon toga je napravljena unakrsna validacija podataka. Unakrsna validacija je ovdje napravljena s RepeatedStratifiedKFold, a to je podjela skupa za učenje na k dijelova te je onda $k-1$ podskupova korišteno za učenje, a zadnji podskup se koristi za izračun točnosti modela. Pri podjelama je zadržana razdioba slučajno izabranih primjeraka po klasama. To se ponavlja više puta i ukupna točnost modela se izračunava kao sredina svih izračunatih vrijednosti (slika 4.10). (17)

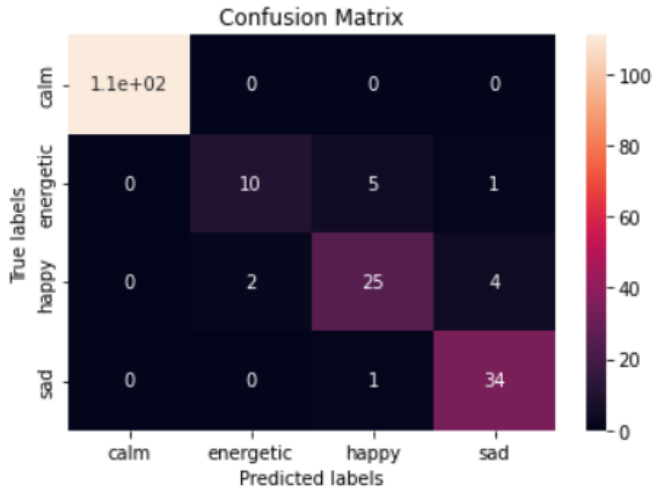
Nakon toga je napravljena matrica konfuzije te je postignuta točnost 0.93. Model griješi između sretnih i energičnih pjesama. (slika 4.11)

Iako su i s predefiniranim hiperparametrima postignuti dobri rezultati, u idućim koracima su promijenjeni hiperparametri modela. Na slici 4.12 prikazan je model koji ima postavljene hiperparametre te njegova matrica konfuzija. Postavljeni su broj izgrađenih stabala, $n_{estimators} = 50$ i maksimalan broj značajki, $max_{features} = 4$ te

```
In [33]: from sklearn.ensemble import RandomForestClassifier
         model = RandomForestClassifier()
         model.fit(X_train, Y_train)
         y_preds = model.predict(X_test)

         cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
         n_scores = cross_val_score(model, X_train, Y_train, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
         print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

         Accuracy: 0.947 (0.030)
```

Slika 4.10: Učenje modela

```
In [36]: cm = confusion_matrix(Y_test,y_preds)
         ax = plt.subplot()
         sns.heatmap(cm,annot=True,ax=ax)

         labels = target['mood']
         ax.set_xlabel('Predicted labels')
         ax.set_ylabel('True labels')
         ax.set_title('Confusion Matrix')
         ax.xaxis.set_ticklabels(labels)
         ax.yaxis.set_ticklabels(labels)
         plt.show()

         print("Accuracy Score",accuracy_score(Y_test,y_preds))
```




```
Accuracy Score 0.9326424870466321
```

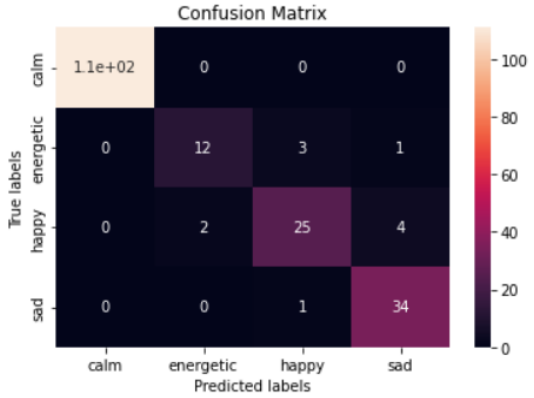
Slika 4.11: Matrica konfuzije

je dobivena malo veća točnost, koja iznosi 0.94 te bolje razlikovanje između sretnih i energičnih pjesama.

I na ovom modelu je provedena unakrsna validacija te je dobivena malo bolja toč-

nost uz manju standardnu devijaciju. (slika 4.13)

```
[40]: from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=50,max_features=4)
model.fit(X_train, Y_train)
y_preds = model.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(Y_test,y_preds)
ax = plt.subplot()
sns.heatmap(cm,annot=True,ax=ax)
labels = target['mood']
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
plt.show()
print("Accuracy Score",accuracy_score(Y_test,y_preds))
```




```
Accuracy Score 0.9430051813471503
```

Slika 4.12: Matrica konfuzije

```
In [41]: cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X_train, Y_train, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.950 (0.024)
```

Slika 4.13: Unakrsna validacija

4.4. Model stroja potpornih vektora

Skup podataka je podijeljen na skup za učenje i skup za testiranje. (slika 4.14)

```
X_train,X_test,Y_train,Y_test = train_test_split(X,encoded_y,test_size=0.3,random_state=15)
```

Slika 4.14: Podjela skupa podataka

Da bi se dobio model s najboljim hiperparametrima korišten je GridSearchCV koji prima rječnik koji definira vrijednosti hiperparametara. Skup za učenje se dijeli na skup za učenje i skup za validaciju. Tada se rade klasifikatori s različitim vrijednostima parametara te se provjerava njihova točnost na skupu za validaciju. Pronalazi se model s najvećom točnosti na skupu za validaciju i onda se pravi novi model s tim hiperparametrima koji se uči koristeći cijeli skup za učenje. (slika 4.15) (20)

```
In [51]: from sklearn.model_selection import GridSearchCV
         param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['linear','rbf', 'poly', 'sigmoid']}
         grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 1)
         grid.fit(X_train, Y_train)
```

Slika 4.15: Traženje modela s najboljim hiperparametrima

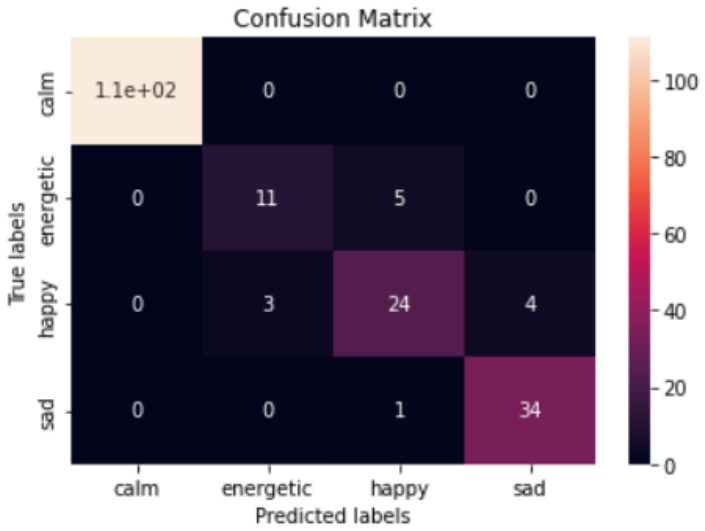
Najbolji model je model s hiperparametrima prikazanim na slici 4.16.

```
In [50]: print(grid.best_params_)
         print(grid.best_estimator_)

         {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
         SVC(C=10, gamma=1)
```

Slika 4.16: Model s najboljim hiperparametrima

Model postiže točnost od 0.9326, ali miješa sretne i energične pjesme što se vidi na matrici konfuzije na slici 4.17.

```
In [52]: clf = grid.best_estimator_
         clf.fit(X_train, Y_train)
         y_preds = clf.predict(X_test)
         cm = confusion_matrix(Y_test,y_preds)
         ax = plt.subplot()
         sns.heatmap(cm,annot=True,ax=ax)
         labels = target['mood']
         ax.set_xlabel('Predicted labels')
         ax.set_ylabel('True labels')
         ax.set_title('Confusion Matrix')
         ax.xaxis.set_ticklabels(labels)
         ax.yaxis.set_ticklabels(labels)
         plt.show()
         print("Accuracy Score",accuracy_score(Y_test,y_preds))
```




```
Accuracy Score 0.9326424870466321
```

Slika 4.17: Matrica konfuzije SVM modela

4.5. Usporedba modela

Model neuronske mreže, model slučajne šume s $n_{estimator} = 50$ i $max_{features} = 4$ i model SVM s RBF jezgrom daju slične rezultate. Svi modela nekad pogriješe u klasifikaciji sretnih i energičnih pjesama.

Budući da model neuronske mreže postiže malo bolje rezultate od ostalih modela, odlučeno je u aplikaciji koristiti taj model.

Spremanje je napravljeno pomoću biblioteka joblib i pipeline iz sklearn. (slike 4.18, 4.19, 4.20)

```
In [30]: import joblib
         import pickle
         from sklearn.pipeline import Pipeline

         pip = Pipeline([('minmaxscaler',MinMaxScaler()),('keras', estimator)])
         pip.fit(X2,encoded_y)
```

Slika 4.18: Kreiranje pipeline objekta

```
In [35]: pip.named_steps['keras'].model.save('keras_model_final01.h5')
```

Slika 4.19: Spremanje modela

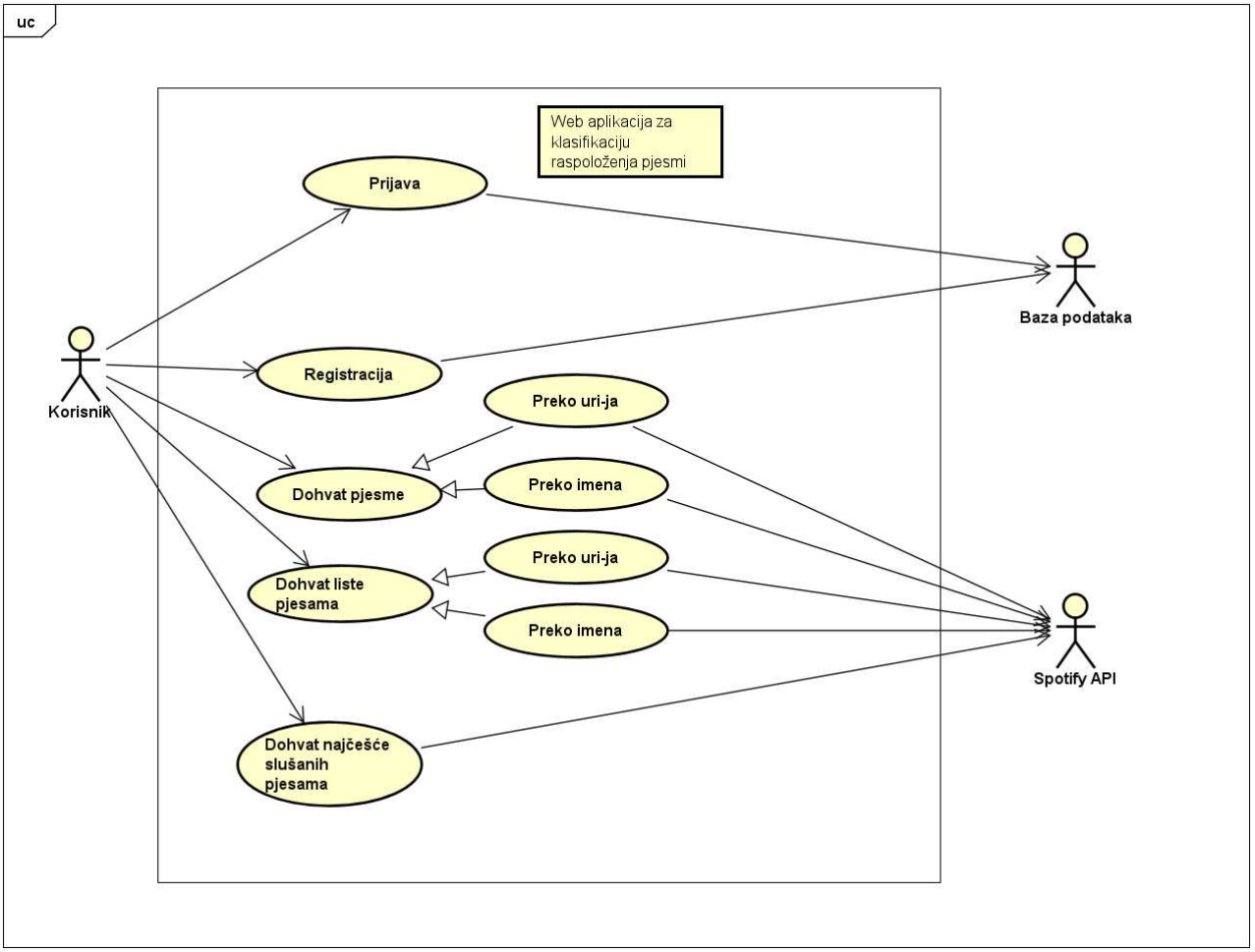
```
In [37]: joblib.dump(pip, 'sklearn_pipeline.pkl')
```

Slika 4.20: Spremanje pipeline objekta

5. Web aplikacija

5.1. Funkcionalni zahtjevi

Korisnik ima mogućnost prijave i registracije u sustav. Nakon registracije korisnik dobije email od aplikacije koji treba potvrditi te se nakon toga može prijaviti u sustav. Nakon prijave korisnik može dohvaćati pjesme i liste pjesama od određenog umjetnika pomoću URI-ja i imena te svoje najslušanije pjesme i dobiti raspoloženja pjesama. Ako je izabrao jednu pjesmu, nakon obrade će dobiti raspoloženje te pjesme. Ako je izabrao umjetnika ili svoje najslušanije pjesme, nakon obrade dobit će odgovor u vidu broja pjesama po raspoloženjima. (slika 5.1)



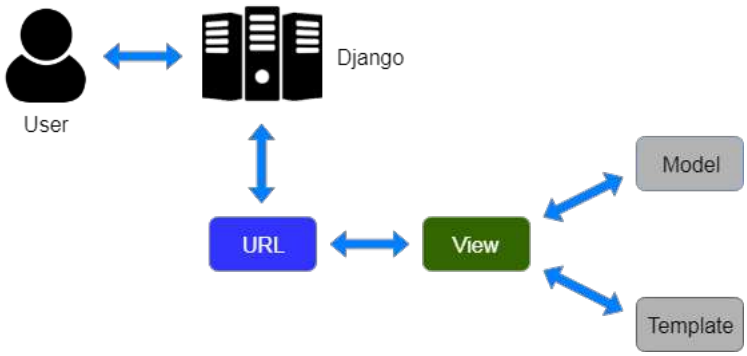
Slika 5.1: Dijagram obrazaca uporabe za ovaj sustav

5.2. Tehnologije i alati

Za izgradnju web aplikacije korišten je programski jezik Python i radni okvir za web Django. Za prikaz podataka korisniku korišteni su HTML, CSS i JavaScript. Za izgradnju modela strojnog učenja korišten je programski jezik Python. Korištena je baza podataka PostgreSQL te platforma za upravljanje bazom pgAdmin.

5.3. Poslužitelj

Poslužitelj je implementiran u radnom okviru Django koji koristi obrazac MVT - Model, Template, View. Sloj modela je sloj pristupa podacima. Olakšava komunikaciju s bazom. Template (predložak) je prezentacijski sloj. View radi poslovnu logiku, razgovara s modelom i renderira predložak. Predložak odgovara Viewu iz MVC (Model - View - Controller), View odgovara Controlleru iz MVC. (slika 5.2) (1)



Slika 5.2: Django MVT (1)

5.3.1. Model

```
class User(AbstractUser):
    is_email_verified = models.BooleanField(default = False)
    def __str__(self):
        return self.email
```

Slika 5.3: Model korisnika

Veza prema bazi definira se u datoteci settings.py u atributu DATABASES. Naredbom python manage.py makemigrations i python manage.py migrate se automatski stvore tablice u bazi koje su Djangu potrebne. Uz to Django stvara i tablice za modele. Za model korisnika (slika 5.3) stvara tablicu 5.3.1 u kojoj se za aplikaciju koriste samo id, username, password, email i is email verified.

User		
id	BIGINT	Jedinstvena oznaka, primarni ključ
username	VARCHAR	Korisničko ime
password	VARCHAR	Lozinka
email	VARCHAR	Email adresa
is email verified	BOOLEAN	Je li email verificiran
last login	TIMESTAMP	Zadnja prijava
is superuser	BOOLEAN	Je li korisnik super korisnik
first name	VARCHAR	Ime
last name	VARCHAR	Prezime
is staff	BOOLEAN	Je li korisnik dio osoblja
is active	BOOLEAN	Je li korisnik aktivan
date joined	TIMESTAMP	Datum registracije

5.3.2. View

Django definira funkcije view kao funkcije koje primaju HTTP zahtjev i vraćaju HTTP odgovor. Odgovor može biti HTML stranica, slika, greška, preusmjeravanje, bilo što preglednik može generirati. U aplikaciji postoje views za authentication i spotify.

Authentication views sadrže logiku za registraciju, prijavu i odjavu korisnika te verifikaciju emailom.

Funkcija za odjavu koristi standardnu Djangovu funkciju za odjavu korisnika i vraća početnu stranicu.

Funkcija za prijavu dohvaća korisnika s korisničkim imenom i lozinkom i provjerava je li korisnik verificiriran. Funkcija za registraciju registrira korisnika i šalje mu mail na zadani email. Obje funkcije provjeravaju unesene vrijednosti (provjeravaju je li korisničko ime bilo prazno, postoji li već korisnik s tim korisničkim imenom u slučaju registracije i slično) te vraćaju poruke ako uneseni podaci nisu bili dobri.

Slanje maila radi tako da se stvori novi objekt EmailMessage koji šalje mail s maila posebno stvorenog za web aplikaciju na mail korisnika. Ta poruka sadržava poveznicu koji ima uid koji je kriptirani primarni ključ i token koji je generiran posebno za svakog korisnika (slika 5.5). Slanje maila se odvija na zasebnoj dretvi. (slika 5.4)

Nakon klika na tu poveznicu korisnik šalje zahtjev GET na server s uid-om i tokenom te se u views u funkciji `activateuser` dekodira taj uid i provjerava generirani token.

```python
def send_email(request, user):
    current_site = get_current_site(request)
    email_subject = 'Activate account'
    email_body = render_to_string('authentication/activate-account.html',
    {
        'user' : user,
        'domain' : current_site,
        'uid'  : urlsafe_base64_encode(force_bytes(user.pk)),
        'token' : generate_token.make_token(user)
    })
    email = EmailMessage(subject=email_subject, body=email_body, from_email = settings.EMAIL_FROM_USER,
    to = [user.email], )

    EmailThread(email).start()
```

Slika 5.4: Slanje maila

```python
from django.contrib.auth.tokens import PasswordResetTokenGenerator
import six

class TokenGenerator(PasswordResetTokenGenerator):
    def _make_hash_value(self, user, timestamp):
        return six.text_type(user.pk) + six.text_type(timestamp) + six.text_type(user.is_email_verified)


generate_token = TokenGenerator()
```

Slika 5.5: Generiranje tokena

Korisnik ima mogućnost dohvaćanja raspoloženja za jednu pjesmu, za umjetnika i raspoloženja svojih najčešće slušanih pjesama.

Za dohvat pjesme može se unijeti ime i URI. Nakon dohvata pjesme toj pjesmi se odredi raspoloženje i prikaže se korisniku.

Za umjetnika je također moguće unijeti ime i URI. Nakon dohvata pjesama, ispisuje se koliko je pjesama imalo koje raspoloženje.

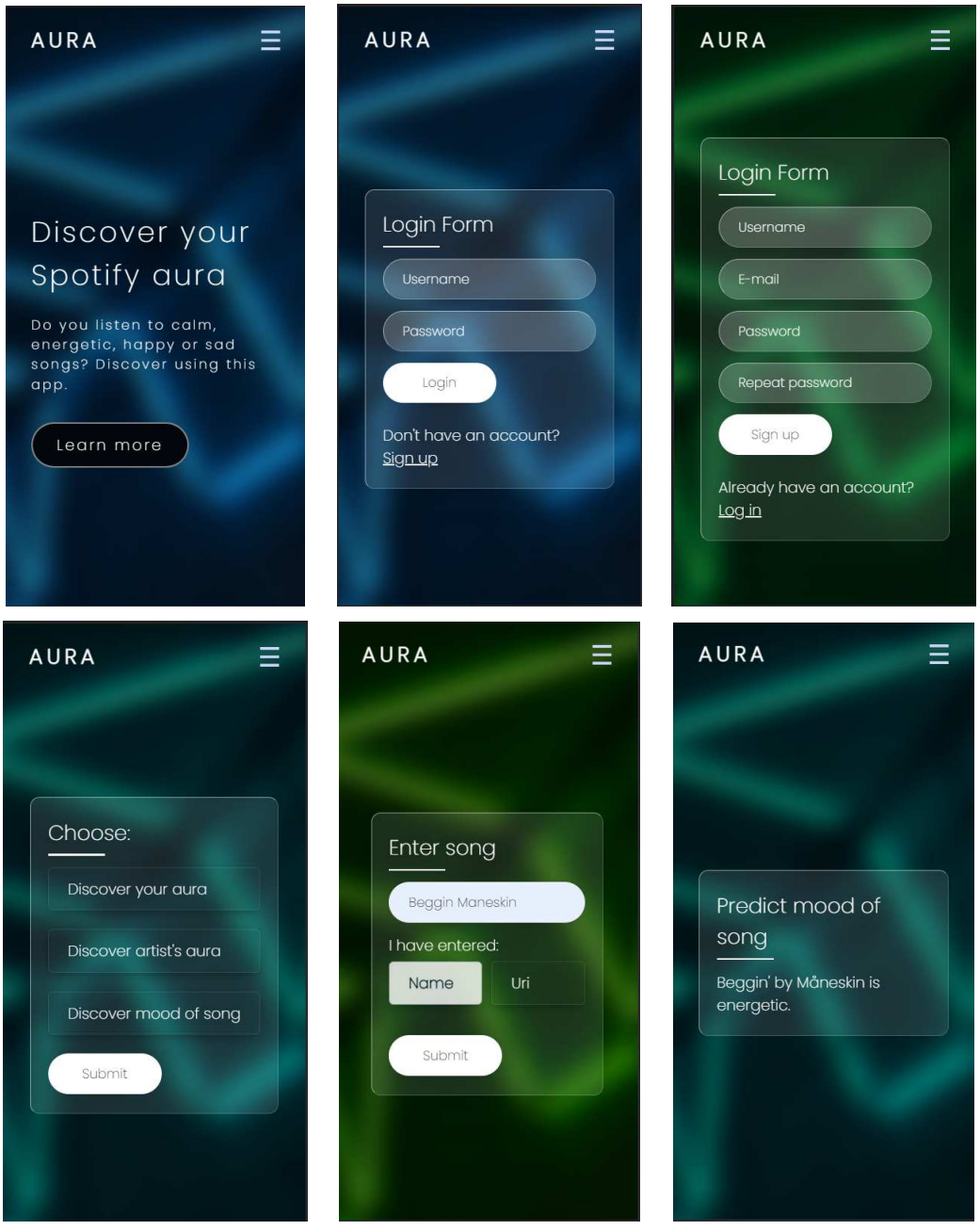
Za korisnika je prvo potrebno dohvatiti njegove podatke sa Spotifyja. Prvo se provjerava je li za trenutnog korisnika u sesiji postoji token za pristup Spotifyju, tj. je li ovaj korisnik već pristupao Spotifyju nedavno. Ako nema, generira se poveznica koja se šalje u odgovoru na zahtjev. Tu poveznicu vraća Spotify API kako bi se korisnik prijavio. Tada se korisniku otvara novi prozor za prijavu. Kada se prijavi dobije kod koji se onda šalje opet na poslužitelj koristeći skriptu na slici 5.6. Taj kod se koristi kako bi se dobio token za pristup na Spotify API i dohvatile korisnikove najslušanije pjesme. Nakon toga se određuje raspoloženje pjesama i to se prikazuje korisniku.

```
<script>
    let form = document.getElementById("form-choose")
    form.addEventListener("submit", (event) => {
        event.preventDefault()
        let btn = document.getElementById("auth")
        let url = btn.attributes['data-link'].value
        let popup = window.open(url, 'Login with Spotify', 'width=400,height=500')
        var timer = setInterval(function() {
            code = localStorage.getItem("spotify_access_code")
            localStorage.removeItem("spotify_access_code")
            if (code) {
                clearInterval(timer);
                document.getElementById("spotify_access_code").value = code
                form.submit();
            }
        }, 500);
    })

    let url = window.location.search
    if (url) {
        let params = url.split("&")
        let code = params[0].substring(params[0].indexOf("=") + 1)
        console.log(code);
        localStorage.setItem("spotify_access_code", code)

        window.close()
    }
</script>
```

Slika 5.6: Skripta za generiranje koda

5.3.3. Predložak

Predlošci opisuje ono što se prikazuje korisniku. Django dinamički generira HTML za prikaz korisniku. Izgled stranice je uređen koristeći CSS i JavaScript. Na tablici 5.1 je prikazan izgled stranice.



Tablica 5.1: Izgled stranice

6. Zaključak

U ovom radu je prikazan način upotrebe tri različita algoritma strojnog učenja na problemu klasifikacije raspoloženja pjesama.

Cilj rada je napraviti analizu ta tri algoritma i najbolji ukomponirati u web aplikaciju koja je jednostavna za korištenje. Model bi se mogao poboljšati kada bi skup podataka za učenje bio veći i bolji. Također je moguće da bi se model poboljšao kada bi se dodavali dodatni slojevi u neuronsku mrežu ili optimirali hiperparametri SVM-a i slučajne šume.

Pišući ovaj rad sam naučila kako algoritmi strojnog učenja rade te kako mijenjati njihove hiperparametre da bi se dobili bolji rezultati. Također, naučila sam kako koristiti Spotify API te kako napraviti poslužiteljski dio aplikacije u Pythonu.

LITERATURA

[1] Java Point, Django MVT, 31.5.2022., https://www.javatpoint.com/django-mvt

[2] IBM, Machine Learning, 25.5.2022., https://www.ibm.com/cloud/learn/machine-learning

[3] Bojana Dalbelo Bašić, Marko Čupić, Jan Šnajder, Uvod u umjetnu inteligenciju, Materijali s predavanja, 11. Umjetne neuronske mreže, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 25.5.2022.

[4] Bojana Dalbelo Bašić, Marko Čupić, Jan Šnajder, Uvod u umjetnu inteligenciju, Materijali s predavanja, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 10. Strojno učenje, 25.5.2022.

[5] SAGAR SHARMA, What the Hell is Perceptron?, 25.5.2022., https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

[6] Michael Skirpan, How do Neural Networks Learn?, 25.5.2022., https://www.kdnuggets.com/2015/12/how-do-neural-networks-learn.html

[7] Sruthi E R, Understanding Random Forest, 25.5.2022., https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[8] Nagesh Singh Chauhan, Decision Tree Algorithm, Explained, 25.5.2022., https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

[9] Rohith Gandhi, Support Vector Machine — Introduction to Machine Learning Algorithms, 26.5.2022., https://towardsdatascien

ce.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[10] Data Flair, Kernel Functions-Introduction to SVM Kernel Examples, 26.5.2022., https://data-flair.training/blogs/svm-kernel-functions/

[11] Scikit Learn, sklearn.preprocessing.MinMaxScaler, 10.3.2022., https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[12] Shaun Enslin, The Complete Guide to Neural Network multi-class Classification from scratch, 20.3.2022., https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[13] Precious Chima, Activation Functions: ReLU Softmax, 20.3.2022., https://medium.com/@preshchima/activation-functions-relu-softmax-87145bf39288

[14] Thomas Wood, What is the Softmax Function?, 20.3.2022., https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

[15] Keras, Dropout layer, 20.3.2022., https://keras.io/api/layers/regularization_layers/dropout/

[16] Pelatrion, Categorical crossentropy, 28.3.2022., ttps://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

[17] Jason Brownlee, A Gentle Introduction to k-fold Cross-Validation, 28.3.2022https://machinelearningmastery.com/k-fold-cross-validation/

[18] Soner Yıldırım, Hyperparameter Tuning for Support Vector Machines — C and Gamma Parameters, 28.5.2022.,https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167

[19] Bhavesh Bhatt, svm-c-gamma-hyperparameter, 28.5.2022., https://deepnote.com/@bhavesh-bhatt/svm-c-gamma-hyperparameter-ec7cdd4f-b499-4b4d-a320-f483e8099691

[20] GeeksForGeeks, SVM Hyperparameter Tuning using GridSearchCV | ML, 28.5.2022., https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/

Web aplikacija za prepoznavanje raspoloženja pjesama s usluge Spotify koristeći strojno učenje

Sažetak

U radu je opisana implementacija tri različita algoritma strojnog učenja: neuronska mreža, slučajna šuma i stroj potpornih vektora. Uspoređene su njihove performanse. Izabrana je neuronska mreža koja je ukomponirana u web aplikaciju te korisnicima otkriva raspoloženja odabranih pjesama.

Ključne riječi: Neuronska mreža, slučajna šuma, stroj potpornih vektora, strojno učenje

Web Application for Predicting the Mood of songs from Spotify Service

Abstract

The paper describes the implementation of three different machine learning algorithms: neural network, random forest and support vector machine. Their performances were compared. A neural network was chosen, which was then integrated into the web application that calculates the moods of the selected songs and presents them to the user.

Keywords: Neural network, random forest, support vector machine, machine learning