SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1442

ODREĐIVANJE POPULARNOSTI PJESAMA S USLUGE SPOTIFY KORISTEĆI METODE STROJNOG UČENJA

Lara Ćorić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1442

ODREĐIVANJE POPULARNOSTI PJESAMA S USLUGE SPOTIFY KORISTEĆI METODE STROJNOG UČENJA

Lara Ćorić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1442

Pristupnica:	**Lara Ćorić (0036541284)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Određivanje popularnosti pjesama s usluge Spotify koristeći metode strojnog učenja**

Opis zadatka:

Cilj završnog rada je usporedba više algoritama strojnog učenja (npr. logistička regresija, stablo odluke, slučajna šuma, stroj s potpornim vektorima, XGBoost) na problemu određivanja popularnosti pjesme (hoće li pjesma biti popularna ili ne) s usluge Spotify. Pritom je potrebno razmotriti neki otvoreno dostupni skup podataka s prikupljenim numeričkim značajkama koje opisuju pjesme, kao što je The Spotify Hit Predictor Dataset s web sjedišta Kaggle (https://www.kaggle.com/datasets/theoverman/the-spotify-hit-predictor-dataset). Prilikom izgradnje modela strojnog učenja potrebno je detaljno razmotriti različite značajke koje opisuju pjesmu, podijeliti skup podataka na odgovarajući način na skup za učenje i skup za testiranje, provesti potrebne metode predobrade podataka i odabir značajki, izabrati najbolje hiperparametre pojedinih modela na skupu za učenje te prikazati i usporediti rezultate algoritama na skupu za testiranje. Osim točnosti, potrebno je prilikom usporedbe uzeti u obzir i mogućnost tumačenja modela kao i broj i jednostavnost određivanja značajki modela na temelju pjesama. Implementaciju je potrebno napraviti u programskom jeziku po vlastitom izboru.

Rok za predaju rada: 14. lipnja 2024.

Zahvaljujem mentoru , izv. prof. dr. sc. Alan Joviću, na pomoći pri izradi završnog rada.

Sadržaj

1. Uvod

Kroz povijest, pjesme su se uvijek držale šablone koja osigurava ugodno iskustvo, no i malu dozu iznenađenja u određenim aspektima, bilo da je riječ o vokalima, instrumentalu ili neočekivanoj strukturi pjesme. Poznato je među pjesnicima i producentima da se rijetko sa sigurnošću može reći da će neka pjesma biti hit, jer mnogi pozadinski čimbenici utječu na uspješnost pjesama [1]. Ipak, kroz godine prikupljenih podataka o popularnim pjesmama sigurno postoje uzorci i određeni atributi koji mogu skoro osigurati uspješnost pjesme na današnjim ljestvicama.

Cilj ovog rada je, koristeći različite metode strojnog učenja kao što su: logistička regresija, stablo odluke, slučajna šuma, stroj s potpornim vektorima (SVM) i XGBoost, procijeniti postoje li doista značajke koje vidljivo povećavaju potencijalnu popularnost pjesme. Analiziranjem podataka i odabirom najboljih hiperparametara pri korištenju različitih modela pokušat će se doći do zaključka.

2. Opis podataka

2.1. The Spotify Hit Predictor Dataset

The Spotify Hit Predictor Dataset je skup podataka[2] korišten u ovom radu. Ovaj skup podataka sastoji se od šest csv datoteka gdje svaka sadrži najpopularnije pjesme tog desetljeća. Desetljeća koja analiziramo su od 1960-ih sve do kraja 2010-ih. Skup podataka sadrži mnoge ključne značajke koje opisuju pjesme, kao što su:

- **Track Name**: Naziv pjesme.
- **Artist Name**: Ime izvođača.
- **uri**: Jedinstveni identifikator za pjesmu.
- **Acousticness**: Vjerojatnost da je pjesma akustična.
- **Danceability**: Koliko je pjesma pogodna za ples.
- **Energy**: Mjera intenziteta i aktivnosti pjesme.
- **Instrumentalness**: Vjerojatnost da pjesma nema vokale.
- **key**: Procijenjeni osnovni tonalitet pjesme.
- **Loudness**: Prosječna glasnoća pjesme.
- **Speechiness**: Vjerojatnost da pjesma sadrži izgovorene riječi.
- **Tempo**: Brzina pjesme (u BPM).
- **Valence**: Mjera koliko je pjesma pozitivna ili vesela.
- **target**: Ciljana varijabla za pjesmu.

te još niz glazbenih atributa, ukupno njih 19. Ove značajke omogućuju detaljnu analizu kako bi se odredilo koje vrijednosti pridonose uspjehu pjesme. Takva analiza može pomoći u predviđanju popularnosti pjesama i razumijevanju što ih čini hitovima.

Skup podataka sadrži sveukupno 41106 pjesama, koje su raspoređene po desetljećima:

- *dataset-of-60s*: sadrži 8642 pjesama
- *dataset-of-70s*: sadrži 7766 pjesama.
- *dataset-of-80s*: sadrži 6908 pjesama.
- *dataset-of-90s*: sadrži 5520 pjesama.
- *dataset-of-00s*: sadrži 5872 pjesama.
- *dataset-of-10s*: sadrži 6398 pjesama.

2.2. Obrada podataka

Za obradu podataka napravljena je Pythonova datoteka nazvana `load_and_preprocess.py`. Unutar ove datoteke nalaze se tri funkcije: `load_data`, `preprocess_data` i `explore_data`. U glavnoj datoteci `main.py` učitavaju se zadane CSV datoteke u podatkovne okvire (engl.*dataframe*) na koje se prvo primjenjuje funkcija `preprocess_data`.

Unutar funkcije `preprocess_data` uklanjaju se nepotrebni atributi te standardiziraju podatci, kao što se vidi na slici 2.1., kako bi sve vrijednosti atributa bile u brojčanom obliku unutar određenih granica. Nakon toga, u `main.py` spajaju se svi uređeni podatkovni okviri u jedan zajednički nazvan `all_data`.

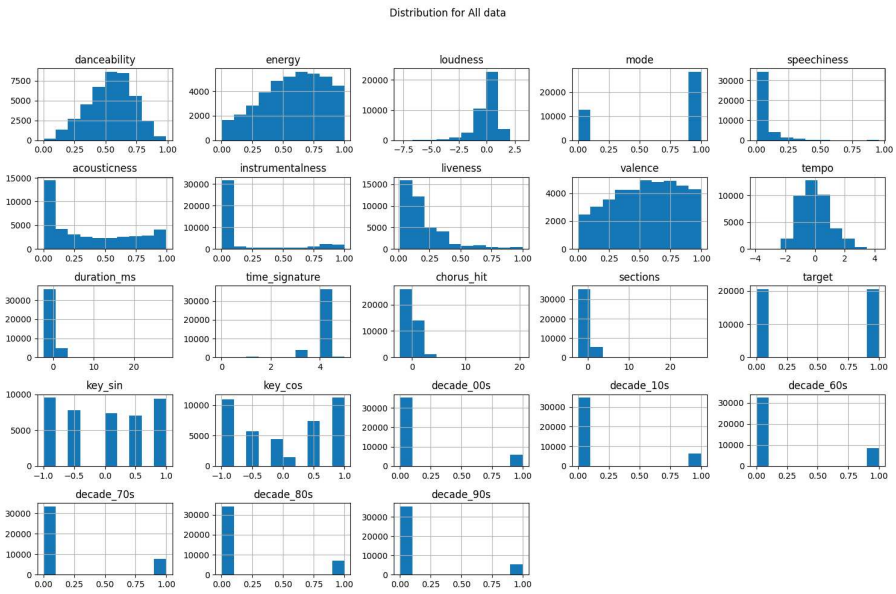
Zadnja funkcija koja se poziva je `explore_data` koja grafički prikazuje sve atribute, prvo po desetljećima i na kraju sveukupno. Prikaz podataka možemo vidjeti na slici 2.2.

Posljednji korak prije implementacije algoritama je odvajanje `all_data` u `train_set` i `test_set` uz pomoć funkcije `train_test_split`, kao što je prikazano na slici 2.3. Podaci su podijeljeni tako da 80% podataka odlazi u skup za učenje i 20% u skup za testiranje. Postavljen je `random_state` na 42 i `stratify` na značajku *target* kako bi se osigurao

jednaki iznos podataka koji su hit i non-hit. Sljedeći korak je spremanje skupova u datoteke oblika csv i pickle, nakon čega se može započeti s pozivanjem funkcija za izgradnju algoritama strojnog učenja.

```
df.drop(["uri","artist","track"], axis=1, inplace=True)

#standardizacija
scaler = StandardScaler()
df['loudness'] = scaler.fit_transform(df[['loudness']])
df['tempo'] = scaler.fit_transform(df[['tempo']])
df['duration_ms'] = scaler.fit_transform(df[['duration_ms']])
df['chorus_hit'] = scaler.fit_transform(df[['chorus_hit']])
df['sections'] = scaler.fit_transform(df[['sections']])

#key pretvaranje u sin i kos komponente
df['key_sin'] = np.sin(df['key'] * (2. * np.pi / 12))
df['key_cos'] = np.cos(df['key'] * (2. * np.pi / 12))
df.drop('key', axis=1, inplace=True)
```

Slika 2.1. Poboljšavanje podataka



Slika 2.2. Prikaz podataka

```
train_set, test_set = train_test_split(all_data, test_size=0.2, random_state=42, stratify=all_data['target'])
```

Slika 2.3. Stvaranje skupova za učenje i testiranje

3. Opis i implementacija algoritama strojnog učenja

Strojno učenje postaje sve popularnije i nalazi široku primjenu. Cilj ovog poglavlja je analizirati najpoznatije modele strojnog učenja, razumjeti njihov rad, područja primjene te uspješnost u predikciji popularnosti pjesama.

3.1. Biranje hiperparametara

Pri korištenju različitih modela strojnog učenja, također je važno obratiti pažnju na hiperparametre. U strojnom učenju, hiperparametar je parametar, poput stope učenja ili izbora optimizatora, koji određuje detalje procesa učenja [3]. Kako modeli postaju složeniji, broj hiperparametara raste, što otežava ručno isprobavanje svih mogućih kombinacija.

Za optimizaciju hiperparametara u ovom radu koristi se metoda **RandomizedSearchCV**. Ova metoda nasumično isprobava unaprijed definirani broj kombinacija hiperparametara, čime se smanjuje vrijeme potrebno za pronalaženje optimalnih postavki u usporedbi s metodom grid searcha, koja pretražuje sve moguće kombinacije. Randomized Search omogućuje efikasnije istraživanje prostora hiperparametara, često pronalazeći dobre kombinacije brže nego tradicionalne metode [4].

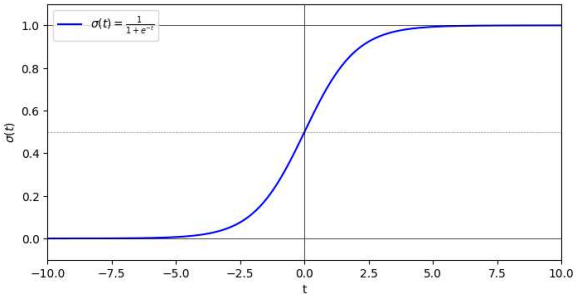
3.2. Logistička regresija

Logistička regresija (također poznata kao Logit regresija) često se koristi za procjenu vjerojatnosti da instanca pripada određenoj klasi. Model predviđa da instanca pripada toj klasi (nazvanoj pozitivna klasa, označena sa "1"), a u suprotnom predviđa da ne pripada (tj. pripada negativnoj klasi, označenoj sa "0"). Ovo čini logističku regresiju binarnim

klasifikatorom [5].

Logistička regresija izračunava težinski zbroj ulaznih značajki i koristi logističku (sigmoidnu) funkciju 3.1. za transformaciju tog rezultata u vjerojatnost između 0 i 1.

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



Slika 3.1. Logistička funkcija

Cilj učenja modela je postaviti parametar θ tako da model daje visoke vjerojatnosti za pozitivne instance i niske za negativne.

$$c(\theta) = \begin{cases} -\log(\hat{p}) & \text{ako } y = 1 \\ -\log(1 - \hat{p}) & \text{ako } y = 0 \end{cases} \tag{3.1}$$

Jednadžba (3.1) Funkcija gubitka za pojedini primjer iz skupa za treniranje

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \tag{3.2}$$

Jednadžba (3.2) Funkcija gubitka za logističku regresiju, koja mjeri koliko dobro model predviđa vjerojatnosti za pozitivne i negativne instance.

Prednosti logističke regresije

- Jednostavna za implementaciju i interpretaciju.
- Dobro radi kada su značajke linearno razdvojive.
- Koristi se kao osnovna metoda za binarnu klasifikaciju.

Nedostaci logističke regresije

- Može biti neefikasna za nelinearno razdvojive podatke bez prethodne transformacije značajki.
- Osjetljiva je na prekomjerno prilagođavanje (engl. *overfitting*) kod malih skupova podataka ili kada postoji veliki broj značajki u usporedbi s brojem uzoraka.

3.2.1. Implementacija

Implementacija logističke regresije napravljena je uz pomoć Pythonovog paketa **sklearn.linear_model**. Prva i glavna funkcija u Pythonovoj datoteci `log_reg_model.py` jest `build_log_reg` (slika 3.2.). Unutar ove funkcije učitava se `train_set` i `test_set` te se stvara instanca klase `LogisticRegression` sa zadanim parametrima. Nakon toga se poziva metoda `fit` za treniranje modela na skupu `X_train`. Zadnja funkcija koja se poziva prije evaluacije modela je `predict`, metoda uz pomoć koje se stvaraju `y_train_pred` i `y_train_test`.

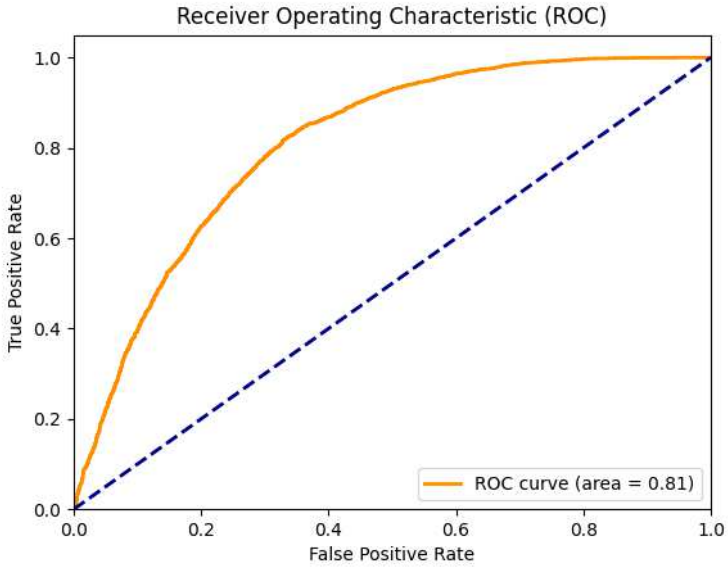
```
def build_log_reg(train_set, test_set):
    X_train = train_set.drop(columns=['target'])
    X_test = test_set.drop(columns=['target'])
    y_train = train_set['target']
    y_test = test_set['target']

    log_reg = LogisticRegression()
    log_reg.fit(X_train, y_train)


    with open('logistic_regression_model.pkl', 'wb') as model_file:
        pickle.dump(log_reg, model_file)

    y_train_pred = log_reg.predict(X_train)
    y_test_pred = log_reg.predict(X_test)
```

Slika 3.2. Izrada modela logističke regresije

Nakon izrade modela, podatci o modelu se spremaju u `log_reg_results.txt` datoteku koja sadrži sve podatke o modelu, kao što su točnost na skupu za učenje i testiranje (engl. *accuracy*), matrica konfuzije (engl. *confusion matrix*) te izvještaj klasifikacije

(engl. *classification report*). Kako bi se dodatno provjerilo da model nije prekomjerno prilagođen, poziva se funkcija `cross_val_score` koja računa točnost modela koristeći unakrsnu validaciju (engl. *cross-validation*).

Na kraju, pomoću funkcije `feature_importances` ispisuju se najvažnije značajke prema procjeni modela logističke regresije i pomoću funkcije `plot_roc_curve`,3.3., crta se krivulja ROC, koja prikazuje performanse klasifikacijskog modela.



Slika 3.3. Krivulja ROC

3.3. Stablo odluke

Stablo odluke je vrlo svestran algoritam strojnog učenja koji se može koristiti za regresiju i klasifikaciju podataka. Svaki čvor stabla predstavlja značajku skupa podataka, dok svaka grana ispod čvora predstavlja vrijednost te značajke ili skup vrijednosti. Listovi stabla predstavljaju klasifikacijske odluke ili predviđanja.

Stablo odluke se može izraditi uz pomoć različitih algoritama kao što su: *ID3*, *C4.5*, *C5.0* i *CART* [5].

scikit-learn koristi optimiranu verziju CART-a (engl. *Classification and Regression Trees*) koja služi za izradu stabala za klasifikaciju i regresiju. Za razliku od drugih algoritama, podržava numeričke varijable te stvara iskjučivo binarna stabla. Svaki čvor stabla predstavlja značajku (engl. *feature*), dok grane predstavljaju vrijednosti značajki. CART

koristi indeks Gini za klasifikaciju i metodu najmanjih kvadrata za regresiju. Algoritam dijeli podatke u čvorovima koristeći prag koji daje najveći informacijski dobitak. Prag u ovom kontekstu označava vrijednost određene značajke koja najbolje razdvaja podatke na podskupove s najmanjom mogućom nečistoćom. Kvaliteta podjele računa se pomoću funkcije nečistoće ili funkcije gubitka, ovisno o zadatku.

Za klasifikaciju skupa podataka korištenih u ovom radu koristit ćemo sljedeće jednadžbe:

$$Gini(t) = 1 - \sum_{k=1}^{K} p_k^2 \tag{3.3}$$

Jednadžba (3.3) Gini: p_k je proporcija klase k u čvoru.

$$H(t) = -\sum_{k=1}^{K} p_k \log(p_k) \tag{3.4}$$

Jednadžba (3.4) Log Loss ili Entropija

Prednosti stabla odluke:

- Jednostavna interpretacija i vizualizacija.
- Može raditi s numeričkim i kategorijskim značajkama.
- Sposobnost integracije u složenije modele.

Nedostaci stabla odluke:

- Sklonost prenaučenosti, posebno na malim skupovima podataka.
- Stabla mogu postati vrlo velika i kompleksna.

3.3.1. Implementacija

Stablo odluke je implementirano uz pomoć Pythonovog paketa `sklearn.tree`. Glavna funkcija `build_decision_tree` prima `train_set` i `test_set` kao parametre te poziva

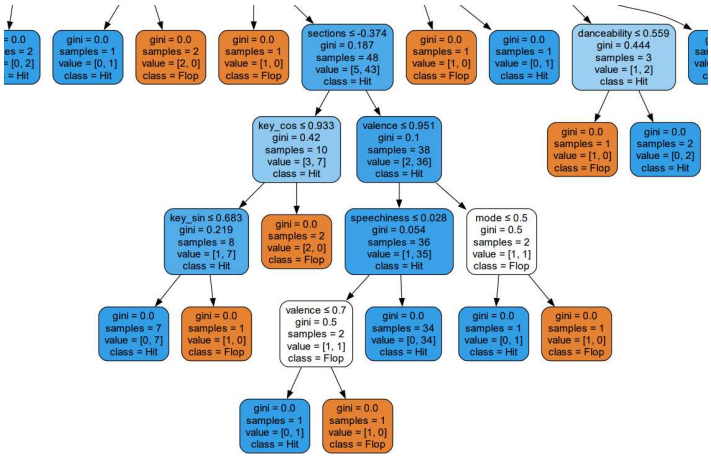
dodatnu funkciju `decision_tree_randomized_search` koja, uz pomoć *Randomized Search* pokušava pronaći najbolje hiperparametre, što se vidi na slici 3.4. Isprobava se više opcija za: `max_depth`,`min_samples_split`, `min_samples_leaf`, `criterion`.

```
def decision_tree_randomized_search(X_train, y_train):
    param_dist = {
        'max_depth': [3, None],
        'min_samples_split': [2, 10, 20],
        'min_samples_leaf': [1, 5, 10],
        'criterion': ['gini', 'entropy']
    }
    tree = DecisionTreeClassifier()
    random_search = RandomizedSearchCV(tree, param_distributions=param_dist, n_iter=36, cv=5, scoring='accuracy', random_state=42, n_jobs=-1)
    random_search.fit(X_train, y_train)
    return random_search.best_estimator_
```

Slika 3.4. Podešavanje hiperparametara

Nakon toga se stvara datoteka `decision_tree_results.txt` u kojoj se spremaju točnost skupova za učenje i testiranje, matrica konfuzije, izvještaj klasifikacije i rezultati unakrsne validacije.

Na kraju, uz pomoć funkcija `plot_feature_importances` i `tree_visualisation` vizualiziraju se rezultati novonastalog modela. Pomoću `plot_feature_importances` prikazuju se najbitniji atributi po modelu stabla odluke. S funckijom `tree_visualisation` stvara se prikaz stabla odluke (slika 3.5.).

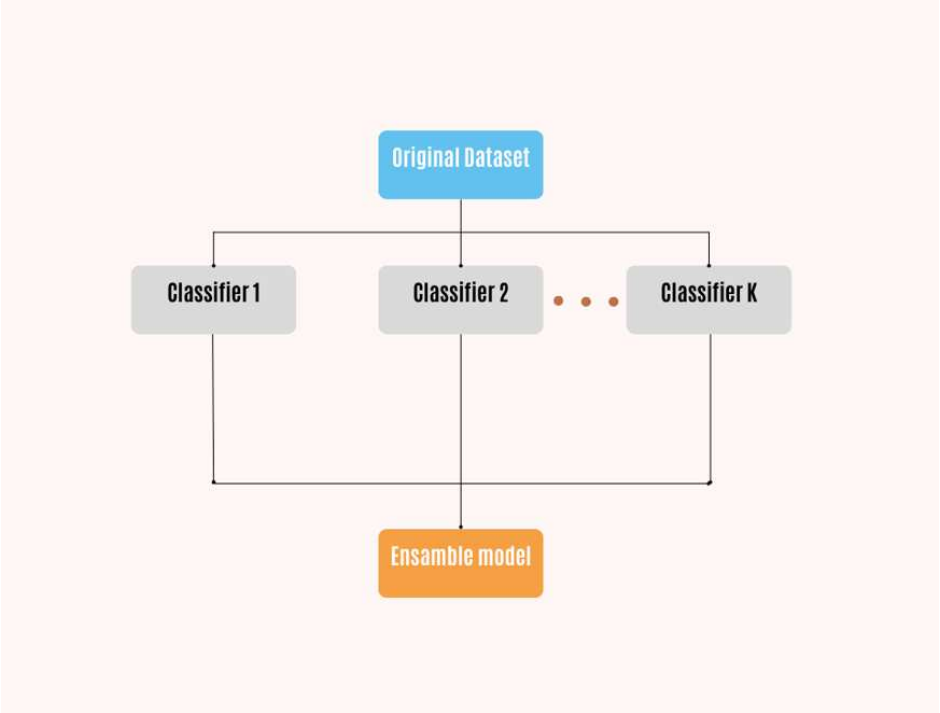


Slika 3.5. Prikaz stabla odluke

3.4. Slučajna šuma

Slučajna šuma je napredni algoritam strojnog učenja koji se oslanja na kombinaciju više stabala odluke za donošenje preciznijih predikcija. Slučajna šuma koristi tehniku zvanu *bagging (Bootstrap Aggregating)* kako bi izgradila više stabala odluke paralelno iz nasumično odabranih uzoraka našeg skupa podataka. Vizualni prikaz ovog procesa može se vidjeti na slici 3.6.

Bagging je jedna od popularnih metoda učenja ansambla (engl. *Ensemble learning*) koje također uključuju: *bagging, boosting, stacking*[6]. Jednu od ovih metoda dodatno ćemo spomenuti u sljedećim poglavljima.



Slika 3.6. Vizualizacija koncepta *bagginga*

Učenje ansambla funkcionira na ideji *"wisdom of crowds"* koja smatra da je odluka veće grupe bolja od individualne. Učenje se odnosi na grupu modela koji će zajedno dati boju predikciju nego što bi samostalno.

Umjesto da koristimo samo jedno stablo, treniramo niz stabala. Predikcije pojedinačnih stabala se zatim kombiniraju – kod klasifikacije većinskim glasovanjem, a kod regresije prosjekom rezultata [7].

Prednosti slučajne šume:

- Smanjenje prenaučenosti u usporedbi s pojedinačnim stablom odluke.
- Rad s velikim brojem značajki zbog nasumičnog odabira podskupa značajki pri svakom dijeljenju.
- Visoka točnost i otpornost na raznolike skupove podataka.

Nedostaci slučajne šume:

- Mogu biti spore i vrlo zahtjevne za obraditi ako imamo mnogo stabala ili značajki.
- Teže za interpretirati rezultate jer se predikcija temelji na rezultatu više stabala.
- Rezultati mogu dosta varirati ovisno o uzorcima korištenim za treniranje stabala.

3.4.1. Implementacija

Slučajna šuma je implementirana uz pomoć Pythonovog paketa `sklearn.ensemble`.

Prva i glavna funkcija datoteke je `build_random_forest` koja prima `train_set` i `test_set`. Unutar glavne funkcije poziva se `random_forest_randomized_search` kako bi se pronašli najbolji hiperparametri za model (slika 3.7.). S dobivenim modelom s najboljim hiperparametrima koristi se funkcija `model.predict()` za `train_set` i `test_set`.

```
def random_forest_randomized_search(X_train, y_train):
    params = {
        'n_estimators': [50, 100, 200, 300, 500],
        'max_features': [None, 'sqrt', 'log2'],
        'max_depth': [None, 10, 20, 30, 50, 70],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    rf_model = RandomForestClassifier()
    random_search = RandomizedSearchCV(rf_model, param_distributions=params, n_iter=50,scoring='accuracy', random_state=42, n_jobs=-1)
    random_search.fit(X_train, y_train)
    return random_search.best_estimator_
```

Slika 3.7. Optimizacija hiperparametara za model slučajne šume

U zadnjem koraku provodi se evaluacija i ispisuju se točnosti za `train_set` i `test_set`, matrica konfuzije, izvještaj klasifikacije te rezultati unakrsne validacije. Zadnja funkcija koja se poziva je `feature_importances`, koja, kao i za stablo odluke, crta graf najvažnijih atributa skupa podataka prema našem modelu.

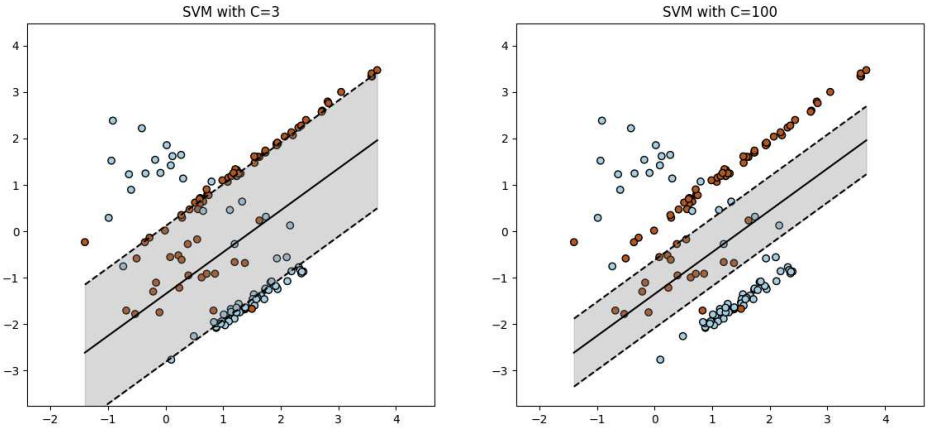
3.5. Metoda potpornih vektora (SVM)

Support Vector Machine (SVM) je jedna od najkorisnijih modela strojnog učenja danas te je široko implementiran u područjima zdravstva, obrade prirodnog jezika, obrade signala te u područjima prepoznavanja govora i slika [8].

Glavna ideja modela jest podijeliti dataset uz pomoć hiperravnine koja maksimalno razdvaja klase podataka u prostoru. SVM model koristi potporne vektore (engl. *support vectors*) čiji je posao definirati granicu između klasa.

Jedan od bitnijih hiperparametara modela jest funkcija jezgre (engl. *kernel function*). Funkcija jezgre omogućuje rad u višedimenzionalnim prostorima, čime se postiže razdvajanje početno nerazdvojivih podataka. Ova tehnika se obično naziva *"kernel trik"*. Najčešće korišteni kerneli su: *linearni, polinomski, radial basis function (RBF) i sigmoidni kernel*.

SVM također koristi regularizacijski parametar C koji kontrolira odnos između maksimalne margine i klasifikacijskih grešaka. Visoka vrijednost parametra C znači da model pokušava ispravno klasificirati sve primjere iz trening skupa, dok manja vrijednost može povećati marginu na račun toleriranja nekih grešaka (slika 3.8.).



Slika 3.8. Usporedba performansi SVM modela s različitim vrijednostima hiperparametra *C*

Zadnji glavni pojam modela su potporni vektori koji leže na granici između klasa i time definiraju hiperravninu.

Prednosti SVM-a:

- Korištenje raznih funkcija jezgre za učinkovito modeliranje nelinearnih odnosa u podatcima.
- Učinkovit za binarne klasifikacijske probleme.

Nedostaci SVM-a:

- Izbor hiperparametara i jezgre može biti izazovno te zahtijeva puno eksperimentiranja.
- Modeli, ovisno o funkciji jezgre, mogu biti dosta kompleksni i teško interpretabilni.

3.5.1. Implementacija

Model SVM je implementiran uz pomoć Pythonovog paketa `sklearn.svm`.

Datoteka `support_vector_model.py` započinje s funkcijom `build_SVM` koja prima `train_set` i `test_set` kao parametre. Unutar funkcije poziva se `svm_randomized_search` (slika 3.9.) , uz pomoć koje se nalaze najbolji hiperparametri: (`C`, `kernel`, `gamma`).

```python
def svm_randomized_search(X_train, y_train):
    param_dist = {
        'C': [0.1, 1, 10, 100],
        'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
        'gamma': ['scale', 'auto']
    }
    svm = SVC()
    random_search = RandomizedSearchCV(svm, param_distributions=param_dist, n_iter=50, scoring='accuracy', random_state=42, n_jobs=-1)
    random_search.fit(X_train, y_train)
    return random_search.best_estimator_
```

Slika 3.9. Optimizacija hiperparametara za SVM

Nakon pronalaska najboljeg modela sljedeći korak jest evaluacija modela. Ispisuje se točnost skupova, parametre koje je funkcija `svm_randomized_search` testiranjem proglasila najboljima, matrica zabune, izvještaj klasifikacije te rezultati unakrsne validacije.

3.6. XGBoost

XGBoost (engl. *Extreme Gradient Boosting*), jest optimirana implementacija *Gradient Boostinga*. XGBoost je jedan od najkompleksnijih modela koji se gradi na idejama prijašnje

spomenutih modela poput SVM-a, stabla odluke i učenja ansambla [9].

Boosting se odnosi na ensemble metodu koja kombinira više slabijih modela u jedan snažan model. Postoji mnogo boosting metoda, a najpopularnije su *AdaBoost (Adaptive Boosting)* i *Gradient Boosting*.

Glavna ideja XGBoosta je postupno dodavanje slabih modela u ansambl, pri čemu se svaki novi model fokusira na ispravljanje pogrešaka prethodnog modela. Koristi tehniku optimizacije gradijentnim spustom (engl. *gradient descent optimization*) za pronalaženje optimalnih parametara modela i minimiziranje pogrešaka tijekom treniranja. Na taj način minimizira funkciju gubitka, koja mjeri koliko se stvarne vrijednosti razlikuju od predikcija modela [10].

Prednosti XGBoosta:

- Smanjenje prenaučenosti što poboljšava generalizaciju modela.
- Brz i efikasan zbog uporabe paralelne obrade.
- Automatski upravlja nepostojećim podatcima što ga čini robusnim.

Nedostaci XGBoosta:

- Implementacija i podešavanje modela može biti dosta zahtjevno.
- Može zahtijevati puno memorije.
- Sklon prenaučenosti na malim podatcima.

3.6.1. Implementacija

Implementacija XGBoosta započinje s funkcijom `build_xgboost` koja kao parametre prima `train_set` i `test_set`.

Unutar funkcije poziva se nova funkcija `xgboost_randomized_search` koja uz pomoć *RandomizedSearchCV* pronalazi najbolje hiperparametre za naš model, kao što je prikazano na slici 3.10.

Nakon pronalaska najboljeg modela ispisuje se evaluacija modela: točnost skupova,

matrica zabune, izvještaj klasifikacije te se na kraju zove funkcija za važnost značajki (engl. feature importances) koja grafički prikazuje najbitnije značajke stvorenog modela.

```python
def xgboost_randomized_search(X_train, y_train):
    param_dist = {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5, 7, 10],
        'subsample': [0.5, 0.7, 1.0],
        'colsample_bytree': [0.5, 0.7, 1.0]
    }
    xgb_model = xgb.XGBClassifier()
    random_search = RandomizedSearchCV(xgb_model, param_distributions=param_dist, n_iter=50, scoring='accuracy', random_state=42, n_jobs=-1)
    random_search.fit(X_train, y_train)
    return random_search.best_estimator_
```

Slika 3.10. Optimizacija hiperparametara za XGBoost

3.7. Voting Classifier

Voting Classifier je metoda ansambla koja kombinira predikcije različitih modela te uz pomoć njih donosi završnu odluku.

Postoje dvije vrste klasifikatora:

Klasifikatori s tvrdim glasanjem (engl. *Hard Voting*): Svaki model daje svoju predikciju, a klasa s najviše glasova postaje konačna predikcija. U našem slučaju imamo 5 različitih modela, ako 3 od 5 glasa da je pjesma hit, konačna odluka će biti hit.

Klasifikatori s mekim glasanjem (engl. *Soft Voting*): Svaki model daje vjerojatnost za klasu, a konačni rezultat je zbroj svih vjerojatnosti te klase. Klasa s najvećom ukupnom vrijednošću postaje konačna odluka.

3.7.1. Implementacija

Prva i glavna funkcija Python datoteke `voting_classifier.py` jest `build_voting_classifier` (slika 3.11.) koja kao parametre prima `train_set` i `test_set`. Nakon učitavanja skupova i definiranja `X_train, X_test, y_train, y_test`, svi modeli su učitani uz pomoć funkcije `pickle.load()`.

```python
def build_voting_classifier(train_set, test_set):
    X_train = train_set.drop(columns=['target'])
    X_test = test_set.drop(columns=['target'])
    y_train = train_set['target']
    y_test = test_set['target']

    with open('logistic_regression_model.pkl', 'rb') as file:
        log_reg = pickle.load(file)

    with open('decision_tree_model.pkl', 'rb') as file:
        d_tree = pickle.load(file)

    with open('random_forest_model.pkl', 'rb') as file:
        rnd_forest = pickle.load(file)

    with open('svm_model.pkl', 'rb') as file:
        svm = pickle.load(file)

    with open('xg_boost_model.pkl', 'rb') as file:
        xgb = pickle.load(file)
```

Slika 3.11. Učitavanje skupova i pickle datoteka modela

Nakon toga grade se dva tipa `VotingClassifier`. Jedan koji će implementirati *hard voting* i drugi koji će implementirati *soft voting*. (slika 3.12.)

```python
#hard voting
voting_clf_hard = VotingClassifier(
    estimators=[('lr', log_reg), ('dt', d_tree), ('rf', rnd_forest), ('svc', svm), ('xgb', xgb)],
    voting='hard'
)
voting_clf_hard.fit(X_train, y_train)

#soft voting
voting_clf_soft = VotingClassifier(
    estimators=[('lr', log_reg), ('dt', d_tree), ('rf', rnd_forest), ('svc', svm), ('xgb', xgb)],
    voting='soft'
)
voting_clf_soft.fit(X_train, y_train)
```

Slika 3.12. Izgradnja Voting classifiera s *hard* i *soft votingom*

Nakon evaluacije modela, rezultati su spremljeni u zasebne tekstualne datoteke: `hard_voting_results.txt` i `soft_voting_results.txt`. Unutar ovih datoteka nalaze se dodatne informacije o modelima kao što su točnost skupova, matrica zabune te izvještaj klasifikacije.

4. Rezultati i rasprava

Nakon detaljne analize više modela, potrebno ih je usporediti kako bi se zaključilo koji modeli su najefiksasniji i najtočniji u predikciji popularnosti.

4.1. Usporedba performansi modela

Logistička regresija bio je prvi model koji je treniran, no rezultati su pokazali da je ovaj model jedan od najslabijih. Kao što se može vidjeti iz priložene slike 4.1., točnost modela iznosi oko 0.7435. Iako to nije loš rezultat, u usporedbi s ostalim modelima, pokazuje se slabijim. Glavna značajka koja se istaknula u ovom modelu bila je *danceability*, što je prikazano na slici 4.2.

```
LOGISTIC REGRESSION RESULTS

Training Accuracy: 0.7425495681790536
Test Accuracy: 0.7434930673801995
Confusion Matrix:
[[2749 1362]
 [ 747 3364]]
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.67      0.72      4111
           1       0.71      0.82      0.76      4111

    accuracy                           0.74      8222
   macro avg       0.75      0.74      0.74      8222
weighted avg       0.75      0.74      0.74      8222

Cross-Validation Scores: [0.73924282 0.73194466 0.74714916 0.74273985 0.74756691]
Average Cross-Validation Score: 0.7417286777226691
```

Slika 4.1. Rezultati modela logističke regresije

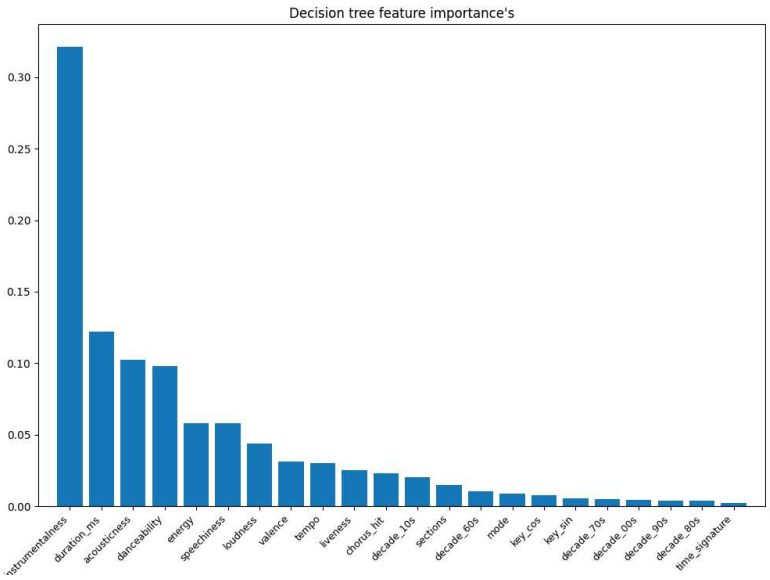
```
                   importance
danceability         4.081820
loudness             0.642991
decade_60s           0.363153
mode                 0.358541
time_signature       0.151236
tempo                0.083254
decade_70s           0.027900
key_cos             -0.004585
sections            -0.015509
duration_ms         -0.028641
chorus_hit          -0.037503
key_sin             -0.054440
decade_00s          -0.136096
decade_90s          -0.255548
decade_10s          -0.273620
decade_80s          -0.306175
valence             -0.309015
liveness            -0.347326
energy              -1.745511
acousticness        -1.801953
speechiness         -2.824026
instrumentalness    -3.446596
```

Slika 4.2. Prikaz značajki modela

Stablo odluke je drugi model koji je treniran. Uz pomoć *RandomizedSearchCV* točnost je povećana za oko 0.02, no sam model i dalje daje niske rezultate u usporedbi s naprednijim modelima. Točnost modela iznosi oko 0.7432, što se može vidjeti iz priložene slike 4.3. Uz pomoć funkcije `plot_feature_importances` mogu se vizualno prikazati najvažnije značajke, od kojih je glavna bila *instrumentalness*, što je prikazano na slici 4.4.

```
DECISION TREE WITH RANDOMIZED SEARCH RESULTS

Training Accuracy: 0.8670173944775574
Test Accuracy: 0.7431281926538555
Best Parameters Found:
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 10, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'monotonic_cst': None, 'random_state': None, 'splitter': 'best'}

Confusion Matrix:
[[3083 1028]
 [1084 3027]]
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.75      0.74      4111
           1       0.75      0.74      0.74      4111

    accuracy                           0.74      8222
   macro avg       0.74      0.74      0.74      8222
weighted avg       0.74      0.74      0.74      8222

Cross-Validation Scores: [0.7412194  0.73802646 0.73833055 0.74000304 0.73783455]
Average Cross-Validation Score: 0.7390827986787252
```

Slika 4.3. Rezultati modela stabla odluke

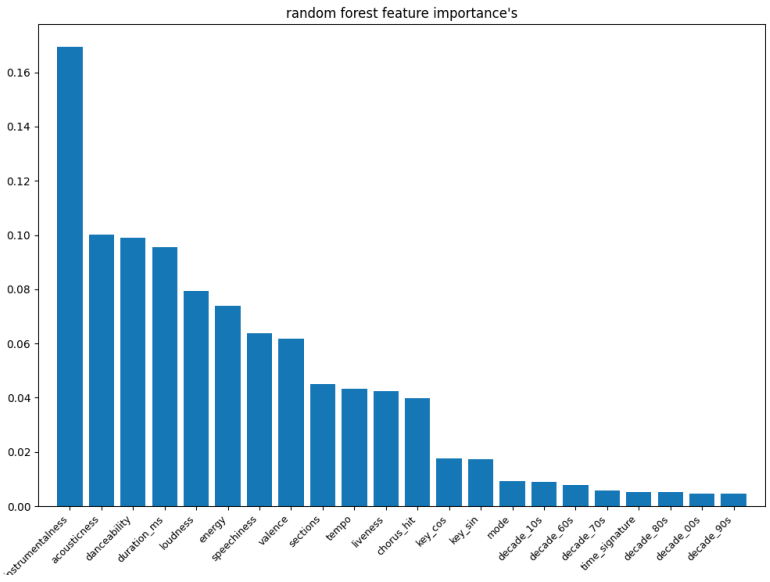


Slika 4.4. Prikaz značajki modela stablo odluke

Slučajna šuma bio je treći model koji je implementiran. Uz pomoć *RandomizedSearchCV* prilagođeni su hiperparametri modela i postignuta je,zasad, najbolja točnost od 0.8069, što je vidljivo na slici 4.5. Uz pomoć funkcije `plot_feature_importances` također je vizualno prikazana važnost određenih značajki. Graf prikazuje da je najvažnija značajka opet *instrumentalness*, što je prikazano na slici 4.6., no vidljivo je da su u ovom modelu ostale značajke imale veću važnost u usporedbi sa stablom odluke.

```
RANDOM FOREST WITH RANDOMIZED SEARCH RESULTS

Training Accuracy: 0.9996654908162024
Test Accuracy: 0.8068596448552664
Best Parameters Found:
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
 'max_depth': 70, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None,
 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0, 'n_estimators': 500, 'n_jobs': None, 'oob_score': False,
 'random_state': None, 'verbose': 0, 'warm_start': False}

Confusion Matrix:
[[3114  997]
 [ 591 3520]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.76      0.80      4111
           1       0.78      0.86      0.82      4111

    accuracy                           0.81      8222
   macro avg       0.81      0.81      0.81      8222
weighted avg       0.81      0.81      0.81      8222

Cross-Validation Scores: [0.80720693 0.80416603 0.80051695 0.8097917  0.81569343]
Average Cross-Validation Score: 0.8074750096831582
```

Slika 4.5. Rezultati modela slučajna šuma



Slika 4.6. Prikaz značajki modela slučajna šuma

Model SVM bio je četvrti model koji je implementiran. Uz pomoć *RandomizedSearchCV* točnost je povećana s 0.777 na 0.7954, što predstavlja jedno od najboljih poboljšanja točnosti za testni skup uz *RandomizedSearchCV*, kako je prikazano na slici 4.7. Najprimjetljiviji nedostatak modela jest vrijeme treniranja, koje je u prosjeku trajalo oko 5 sati, u usporedbi s ostalim modelima kojima je trebalo 3-20 minuta.

```
SVM WITH RANDOMIZED SEARCH RESULTS

Training Accuracy: 0.8239265296192677
Test Accuracy: 0.7981026514230114
Best Parameters Found:
{'verbose': 4, 'output_type': 'input', 'C': 100, 'kernel': 'rbf', 'degree': 3,
'gamma': 'auto', 'coef0': 0.0, 'tol': 0.001, 'cache_size': 1024.0, 'max_iter': -1,
'nochange_steps': 1000, 'probability': True, 'random_state': None, 'class_weight': None,
 'multiclass_strategy': 'ovo'}

Confusion Matrix:
[[2996 1115]
 [ 545 3566]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.73      0.78      4111
           1       0.76      0.87      0.81      4111

    accuracy                           0.80      8222
   macro avg       0.80      0.80      0.80      8222
weighted avg       0.80      0.80      0.80      8222

Cross-Validation Scores: [0.80006082 0.79048198 0.7959556  0.79169834 0.80444039]
Average Cross-Validation Score: 0.7965274271062579
```

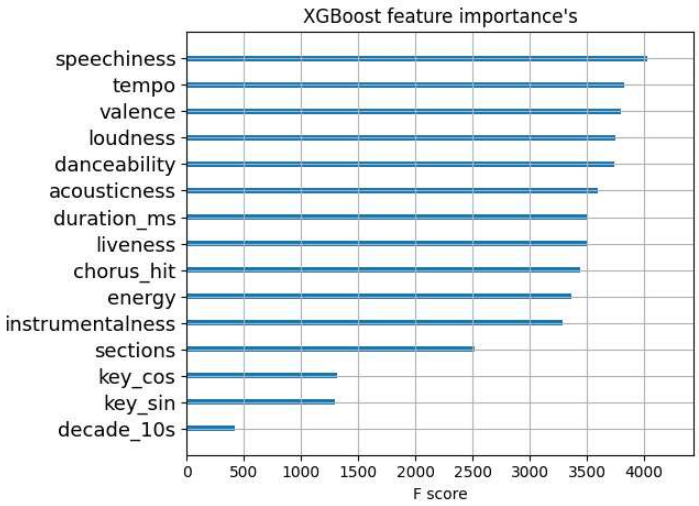
Slika 4.7. Rezultati modela SVM

XGBoost bio je zadnji model koji je implementiran. Uz pomoć *RandomizedSearchCV* postignuti su rezultati s točnošću od 0.8065 i F1-scoreom od 0.80, što je prikazano na slici

4.8. Najvažnija značajka prema ovom modelu je *speechiness*, iako iz priložene slike 4.9. može se vidjeti da su i ostale značajke vrlo bitne. Razlika u važnosti značajki je mala, te uz *speechiness* mogu se istaknuti i *tempo, valence, loudness i danceability*.

```
XGBOOST RANDOMIZED SEARCH RESULTS

Training Accuracy: 0.9794124802335482
Test Accuracy: 0.8064947701289225
Best Parameters Found:
{'objective': 'binary:logistic', 'base_score': None, 'booster': None, 'callbacks': None,
 'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': 0.7,
 'device': None, 'early_stopping_rounds': None, 'enable_categorical': False,
 'eval_metric': None, 'feature_types': None, 'gamma': None, 'grow_policy': None,
 'importance_type': None, 'interaction_constraints': None, 'learning_rate': 0.1,
 'max_bin': None, 'max_cat_threshold': None, 'max_cat_to_onehot': None,
 'max_delta_step': None, 'max_depth': 10, 'max_leaves': None, 'min_child_weight': None,
 'missing': nan, 'monotone_constraints': None, 'multi_strategy': None, 'n_estimators': 200,
 'n_jobs': None, 'num_parallel_tree': None, 'random_state': None, 'reg_alpha': None,
 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None, 'subsample': 1.0,
 'tree_method': None, 'validate_parameters': None, 'verbosity': None}

Confusion Matrix:
[[3150  961]
 [ 630 3481]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.77      0.80      4111
           1       0.78      0.85      0.81      4111

    accuracy                           0.81      8222
   macro avg       0.81      0.81      0.81      8222
weighted avg       0.81      0.81      0.81      8222

Cross-Validation Scores: [0.81313669 0.80492626 0.80599057 0.81146419 0.82223236]
Average Cross-Validation Score: 0.8115500146680887
```

Slika 4.8. Rezultati modela XGBoost



Slika 4.9. Prikaz značajki modela XGBoost

Na kraju je implementiran **Voting Classifier** koji služi da se iz svih modela dobije najbolji rezultat. Rezultati su odvojeni, jedan za *hard voting* i drugi za *soft voting*.

```
Hard_Voting_Results.Txt RESULTS

Training Accuracy: 0.9346186595304707
Test Accuracy: 0.8060082704937971
Confusion Matrix:
[[3066 1045]
 [ 550 3561]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.75      0.79      4111
           1       0.77      0.87      0.82      4111

    accuracy                           0.81      8222
   macro avg       0.81      0.81      0.81      8222
weighted avg       0.81      0.81      0.81      8222

Cross-Validation Scores: [0.80523035 0.80234149 0.80249354 0.81100806 0.81569343]
Average Cross-Validation Score: 0.8073533736789009
```

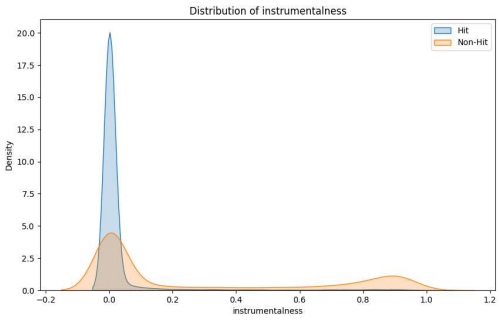
Slika 4.10. Rezultati modela *Voting Classifier*: *hard voting*

```
Soft_Voting_Results.Txt RESULTS

Training Accuracy: 0.9295402019219073
Test Accuracy: 0.8075893943079543
Confusion Matrix:
[[3091 1020]
 [ 562 3549]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.75      0.80      4111
           1       0.78      0.86      0.82      4111

    accuracy                           0.81      8222
   macro avg       0.81      0.81      0.81      8222
weighted avg       0.81      0.81      0.81      8222

Cross-Validation Scores: [0.81100806 0.80294967 0.80355785 0.80568648 0.81538929]
Average Cross-Validation Score: 0.807718272443193
```

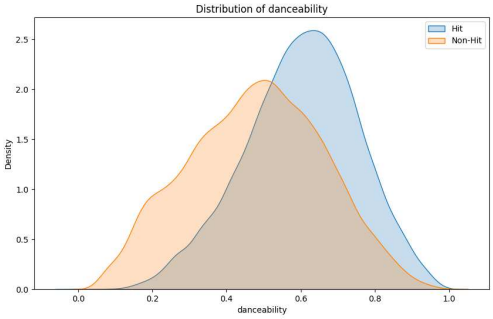
Slika 4.11. Prikaz značajki *Voting Classifier: soft voting*

Kao što se vidi iz slika 4.11. i 4.10., oba tipa postižu visoke rezultate: točnost od otprilike 0.81 i F1-score od 0.80. *Soft voting* ipak malo nadmašuje *hard voting* s razlikom od 0.0016 u točnosti. Rezultat *soft voting* također je najbolji od svih modela s točnošću od 0.8076.

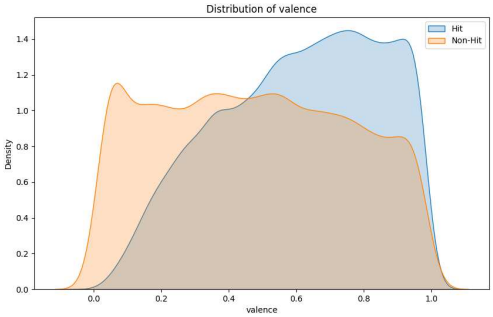
Zadnja i finalna funkcija koja se poziva nakon obrade svih modela jest `analyze_feature_values` koja vizualno prikazuje zastupljenost vrijednosti značajki za hit i non-hit. Rezultati funckije se mogu vidjeti na slikama 4.12., 4.13., 4.14. Prikazane značajke su one koje su se većinom smatrale najbitnijima.



Slika 4.12. Distribucija značajke *instrumentalness* za hit i non-hit pjesme



Slika 4.13. Distribucija značajke *danceability* za hit i non-hit pjesme



Slika 4.14. Distribucija značajke *valence* za hit i non-hit pjesme

5. Zaključak

Iako su svi modeli većinom davali dobre rezultate, najistaknutiji su bili: slučajna šuma, SVM, XGBoost te Voting Classifier koji su svi imali točnost i F1-score oko 80%. Ipak, zbog dugog vremena izvršavanja, SVM, iako daje dobre rezultate, nije najbolji model za ovaj problem.

Također, glavne značajke koje su imale najveću ulogu kroz sve modele su: *speechiness, instrumentalness, loudness, danceability, tempo*. Ovo ima smisla kada uzmemo u obzir što ljude najviše privlači ili odbija u pjesmi. Većina ljudi pridaje veliku važnost karakteristikama koje mogu odmah primijetiti, a upravo su to značajke koje su se pokazale kao najbitnije.

Još jedan par bitnih značajki bile su *valence* (pozitivnost) i *energy* (energija), uz pomoć kojih možemo vidjeti da su ugođaj i emocija koju pjesma izaziva vrlo bitni za njezinu potencijalnu popularnost među javnosti.

Za kraj možemo zaključiti da su najbolji modeli bili oni koji su radili na ideji *"wisdom of crowds"*, odnosno da će predikcija većine biti bliža točnoj vrijednosti nego predikcija pojedinca. Korištenjem više modela možemo izgraditi najbolji model s izvrsnom predikcijom popularnosti.

Ovaj rad može biti koristan za izradu stranice ili aplikacije koja će omogućiti producentima i glazbenicima da, uz pomoć metoda strojnog učenja, uspješno naprave pjesme koje će rezonirati s javnošću.

Literatura

[1] Songtown, “What really makes a song a hit?” https://songtown.com/on-songwriting/really-makes-a-song-a-hit/, [mrežno; stranica posjećena: svibanj 2024.].

[2] Kaggle, “The spotify hit predictor dataset”, https://www.kaggle.com/datasets/theoverman/the-spotify-hit-predictor-dataset, [mrežno; stranica posjećena: lipanj 2024.].

[3] Wikipedia, “Hyperparameter (machine learning)”, https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning), [mrežno; stranica posjećena: svibanj 2024.].

[4] scikit-learn, “Randomizedsearchcv”, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html, [mrežno; stranica posjećena: svibanj 2024.].

[5] ——, “Decision trees”, https://scikit-learn.org/stable/modules/tree.html, [mrežno; stranica posjećena: lipanj 2024.].

[6] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O’Reilly Media, 2019.

[7] IBM, “Random forest”, https://www.ibm.com/topics/random-forest, [mrežno; stranica posjećena: lipanj 2024.].

[8] “Support vector machine (svm)”, https://www.spiceworks.com/tech/big-data/articles/what-is-support-vector-machine/, [mrežno; stranica posjećena: lipanj 2024.].

[9] Nvidia, “Xgboost”, https://www.nvidia.com/en-us/glossary/xgboost/, [mrežno; stranica posjećena: lipanj 2024.].

[10] Analytics Vidhya, “An end-to-end guide to understand the math behind xgboost”, https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/, [mrežno; stranica posjećena: lipanj 2024.].

Sažetak

Rad se bavio određivanjem popularnosti pjesama koristeći metode strojnog učenja poput logističke regresije, stabla odluke, slučajne šume, SVM-a (Support Vector Machine), XGBoosta i Voting Classifiera. Nakon uređivanja podataka, svaka metoda je detaljno razrađena, model je izgrađen te su rezultati uspoređivani. Metode koje su izradile najbolje modele za predikciju popularnosti pjesama su: XGBoost, slučajna šuma, SVM i Voting Classifier, svaka s nešto preko 80% točnosti u određivanju hoće li pjesma biti hit ili ne. Najbitnije značajke koje su utjecale na popularnost pjesama bile su *speechiness*, *instrumentalness*, *loudness*, *danceability*, *valence*, *energy* i *tempo*. Ove značajke su odabrane zbog njihove važnosti u percepciji i privlačnosti pjesama prema slušateljima, odnosno zbog značajke *target* koja je predstavljala je li pjesma bila hit ili non-hit. Ovaj rad može potencijalno olakšati izradu i preporučivanje pjesama te praćenje trendova u glazbi.

Ključne riječi: strojno učenje, logistička regresija, stablo odluke, slučajna šuma, SVM, XGBoost, klasifikator, popularnost pjesama, učenje ansambla

Abstract

The study focused on determining the popularity of songs using machine learning methods such as logistic regression, decision tree, random forest, SVM (Support Vector Machine), XGBoost, and Voting Classifier. After preprocessing the data, each method was analysed, models were built, and results were compared. The methods that produced the best models for predicting song popularity were: XGBoost, random forest, SVM and Voting Classifier. The most important features influencing song popularity were *speechiness*, *instrumentalness*, *loudness*, *danceability*, and *tempo*. These features were chosen due to their significance in the perception and appeal of songs to listeners, specifically the *target* feature representing whether a song was a hit or a flop. This study can potentially help with the creation and recommendation of songs as well as tracking trends in music.

Keywords: machine learning, logistic regression, decision tree, random forest, SVM, XGBoost, classifier, song popularity, ensemble learning