Model dubokog učenja za klasifikaciju MIDI datoteka

Rod, Bartol

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 369

MODEL DUBOKOG UČENJA ZA KLASIFIKACIJU MIDI DATOTEKA

Bartol Rod

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 369

MODEL DUBOKOG UČENJA ZA KLASIFIKACIJU MIDI DATOTEKA

Bartol Rod

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 369

Pristupnik:	**Bartol Rod (0036508261)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	doc. dr. sc. Stjepan Šebek

Zadatak: **Model dubokog učenja za klasifikaciju MIDI datoteka**

Opis zadatka:

U ovom diplomskom radu zadatak je iskoristiti modele dubokog učenja u svrhu klasifikacije MIDI datoteka, koje su uobičajeni formati za glazbene zapise. Cilj je razviti model koji može efikasno razvrstati MIDI datoteke prema klasičnim glazbenicima. Koristi se duboka neuronska mreža koja se trenira na velikom skupu raznolikih MIDI zapisa iz skupa podataka MAESTRO. Evaluacija će se provesti usporedbom performansi različitih arhitektura dubokih modela i analizom njihove sposobnosti generalizacije na novim glazbenim podacima. Krajnji cilj je razvoj efikasnog sustava za automatsku klasifikaciju glazbenih datoteka koji bi imao određenu primjenu u glazbenoj industriji, istraživačkim projektima ili umjetničkoj domeni.

Rok za predaju rada: 28. lipnja 2024.

Hvala na čekanju.

Sadržaj

1. Uvod

Povijest prijenosa i zapisa glazbenih djela odražava tehnološki i kulturološki napredak ljudskog društva. Od usmene predaje, do zapisa u obliku pisma, tiskarskog stroja i digitalnim medijima glazba se prenosila i čuvala iz generacije u generaciju prateći napredak tehnologije i kulture. Svrha zapisivanja glazbe je raznolika; radi li se o edukaciji, zabavi ili umjetnosti, svaki oblik prijenosa glazbe pronađe svoj medij. Danas možemo reći da se prijenos glazbe dijeli u dvije vrste medija: oni namijenjeni slušatelju i oni namijenjeni glazbeniku. Prvi tip medija prenosi zvučne valove s ciljem što vjernijeg dostavljanja glazbe slušatelju. To uključuje različite formate kao što su CD, digitalni audio formati poput Wav-a ili MP3-a, te streaming servisi. Drugi tip medija nastoji glazbu zapisati kako bi se interpretirala i izvodila, te je namijenjen izvođačima. To uključuje notne zapise, tabulature ili digitalne formate poput MIDI, engl. *Musical Instrument Digital Interface*, protokola.

MIDI protokol nije samo medij za prijenos glazbenog zapisa, već je primarno namijenjen računalima i definira skup instrukcija kojima se određuje ponašanje glazbenog hardvera. MIDI omogućava komunikaciju između različitih elektroničkih instrumenata i računala, omogućujući kontrolu nad različitim aspektima zvuka, uključujući visinu tona, jačinu, trajanje i dinamiku. Također, omogućava i programiranje složenih glazbenih aranžmana te sekvenciranje, čineći ga ključnim alatom u modernoj glazbenoj produkciji. Melodije i ritam, tj. sve ono što obuhvaća zapis glazbe u njenom klasičnom smislu samo su dio MIDI protokola.

Ljudska potreba za strukturom i uređenjem jedan su od osnovnih problema koji čovjek pokušava riješiti. Podjela glazbe u žanrove ili neku drugu kategoriju često je predmet rasprave i nije jasno koje djelo svrstati u koju kategoriju. Klasifikacija MIDI datoteka stoga se može svesti i na klasifikaciju informacija o glazbi zapisanoj u njima. To uklju-

čuje analizu parametara kao što su tempo, skala, ključ i aranžman koji se mogu dobiti iz informacije sadržane u MIDI datoteci.

U ovom radu prvo se opisuje MIDI protokol i njegova aplikacija te MIDI datoteke i na koji način je u njima pohranjena glazbena informacija. Zatim je predstavljen skup podataka *GiantMIDI-Piano* [1] i kako će se koristiti te kriterij klasifikacije klasičnih skladatelja. Također, pokazano je i kako u programskom jeziku *Python* iz MIDI datoteke izlučiti bitne značajke glazbenog zapisa. Istraživanje je podijeljeno u dva dijela: u prvom se provodi klasifikacija korištenjem algoritama strojnog učenja, a u drugom metodama dubokog učenja. Klasifikacija strojnim učenjem istražena je korištenjem SVM-ova, engl. *Support Vector Machines* i RF klasifikatorom, engl. *Random Forest Classifier*, a zatim se problem svodi na NLP, engl. *Natural Language Processing*, kao što je i predloženo u radu [2] i uče duboke neuronske mreže, RNN-ovi, eng. *Recurrent Neural Network*. Nakon provedenih metodi rezultati su pokazani u obliku usporedbe preciznosti (engl. *accuracy*) i *F1-score*-a.

2. MIDI protokol

MIDI, engl. *Musical Instrument Digital Interface* tehnički je standard koji opisuje komunikacijski protokol, digitalno sučelje i konektore koji povezuju razne elektroničke glazbene instrumente, računala i druge povezane uređaje u svrhu stvaranja, uređivanja i reprodukcije glazbe. Dakle, MIDI sam po sebi ne prenosi zvuk, već informacije o glazbenim notama, tempu, dinamici i drugim parametrima koji su potrebni za reprodukciju glazbenog djela te definira specifične parametre za prijenos informacija. Razvijen je početkom 1980-ih kao odgovor na potrebu za standardizacijom komunikacije između različitih glazbenih instrumenata i uređaja. Prije MIDI-a, različiti proizvođači imali su svoje vlastite sustave koji nisu bili međusobno kompatibilni. Prvi put javno je predstavljen 1983. godine na godišnjem Sajmu glazbenih instrumenata, NAMM, u Anaheimu, Kalifornija i rezultat je suradnje glavnih proizvođača elektronskih instrumenata, uključujući Roland, Yamaha, Korg i druge.

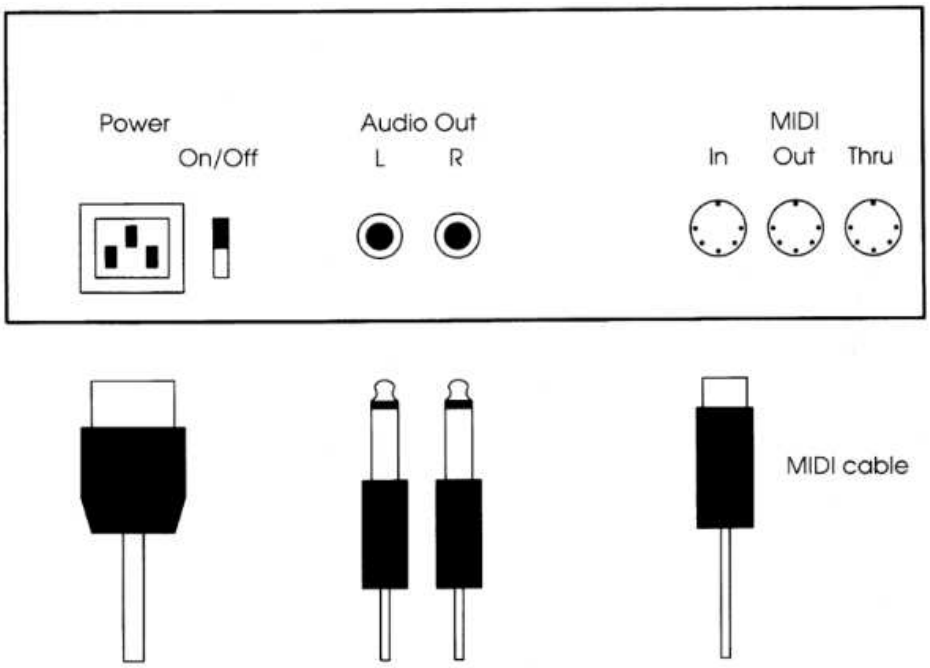
2.1. Tehničke karakteristike

Tehničke karakteristike MIDI protokola mogu se svrstati u dvije grupe zahtjeva koje proizvođač mora ispuniti [3, *The MIDI Specification*]. Prvi zahtjevi odnose se na fizičku vezu i tu spadaju MIDI Portovi i MIDI kablovi [3, *The MIDI Specification*]. Drugi zahtjevi se odnose na podatkovne strukture i njihovu uređenost u protokolu te ih možemo nazvati MIDI porukama [3, *The MIDI Specification*].

2.1.1. MIDI portovi i kablovi

MIDI port ili sučelje je mjesto preko kojeg dva inače nekompatibilna uređaja ostvaruju komunikaciju dok su kablovi fizički prijenosnici voltaže u vrijednostima od 0 ili 1 [3, *MIDI Ports*]. Specifikacija opisuje tri tipa MIDI portova, tipove kablova njihovo kori-

štenje kao što je ilustrirano na slici [3, *MIDI Ports*]. MIDI uređaji tradicionalno koriste 5-pinski DIN konektore, iako novije verzije također koriste USB ili bežične veze. Na slici 2.1. možemo vidjeti tri vrste 5-pinskih DIN konektora na pozadini nekog MIDI hardvera.



Slika 2.1. Ilustracija MIDI uređaja sa sučeljima kablovima [3, *MIDI Ports*]

U ovom radu ne dotičemo se implementacije i korištenja hardver uređaja te njihovih sučelja, stoga ćemo sada prijeći na opis MIDI poruka.

2.1.2. MIDI poruke

MIDI poruke mehanizam su serijalizacije i tumačenja informacije prenesene s jednog uređaja na drugi preko komunikacijskih kanala protokola. Neke od MIDI poruke prikazane su i opisane u tablici 2.1. O njima će biti više riječi u kasnijim poglavljima i bitan su dio obrade MIDI datoteka u svrhu klasifikacije.

MIDI poruke	
Naziv poruke	Opis
Note Off	Nota je zaustavljena
Note On	Nota je počela
Control Change	Promijeni kontrolu parametra
Program Change	Promijeni instrument
Pitch Bend	Promijeni visinu tona
...	...

Tablica 2.1. Neke MIDI poruke i njihova značenja

Kanal za prijenos MIDI poruka podržava 16 neovisnih kanala po vezi, što omogućava istovremenu kontrolu nad različitim instrumentima te se podaci mogu slati brzinom prijenosa od 31.25 kbps. MIDI poruke kakve ćemo koristiti u ovom istraživanju sadržane su u MIDI datotekama.

2.2. MIDI datoteka

MIDI datoteke su digitalni formati koji pohranjuju informacije o glazbenim izvedbama u obliku MIDI poruka. One ne sadrže stvarni audio zapis, već skup uputa koje govore elektronskim instrumentima kako reproducirati glazbu. Standardni MIDI datotečni format koristi ekstenziju .mid ili .midi. MIDI datoteke temeljni su alat u glazbenoj industriji i primjena im je široka. Od komponiranja glazbe u prilagođenim softverima gdje se note i drugi parametri mogu jednostavno mijenjati, premještati i kopirati gdje se svaki aspekt izvedbe uključujući visinu tona, trajanje note i dinamiku može precizno prilagoditi, do orkestracije i notacije glazbe gdje se MIDI datoteke koriste kao izvor u koji se prepisuje glazbeni zvuk u MIDI poruke. U tu svrhu nastao je niz softvera za obradu MIDI datoteka i upravo bi korisnici tih softvera imali značajne koristi od klasifikacije MIDI datoteka.

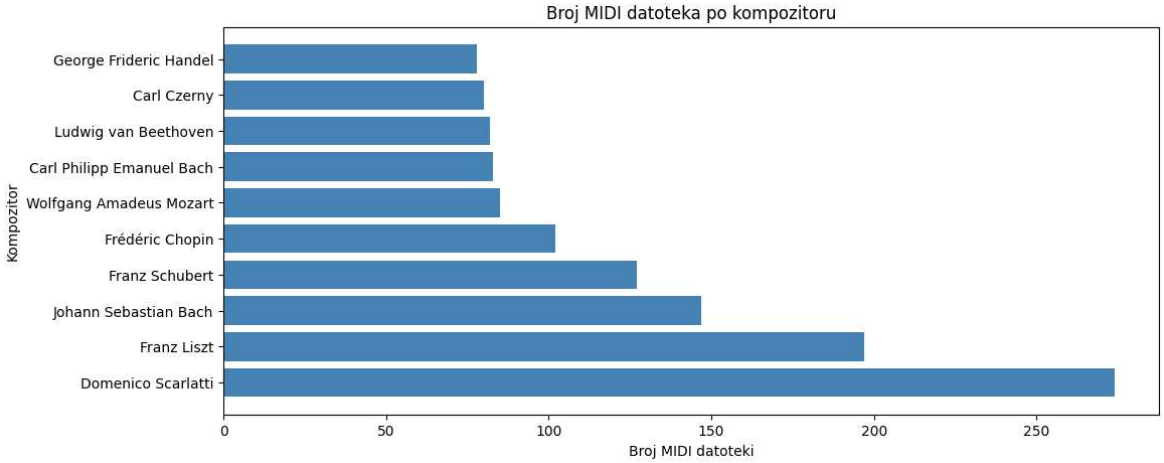
Standardna MIDI datoteka sastoji se od tri glavna dijela: *Header Chunk*-a koji sadrži osnovne informacije o datoteci, poput formata datoteke, broja traka i razlučivosti *tick*-a (najmanja jedinica mjerenja vremena u MIDI datoteci), *Track Chunk*-a gdje svaka traka (*track*) sadrži niz događaja (*events*) koji definiraju glazbenu izvedbu te *Events*-a koji specificiraju glazbene upute. Ukratko, možemo reći da svaka MIDI datoteka sadrži skup MIDI poruka i upravo će se iz informacija o glazbi iz njih dobiti korisne informacije za njihovu klasifikaciju.

3. Skup podataka GiantMIDI-Piano

Skup podataka *GiantMIDI-Piano* sastoji se od približno 39 milijuna noti, prepisanih iz audio u MIDI datoteke [1, *Abstract*]. Sadrži oko 11 tisuća djela napisanih za klavir svako pridruženo nekom od 2786 skladatelja [1, *Abstract*]. 90% MIDI datoteka u skupu stvoreno je prijepisom audio zapisa u MIDI, a ostalih 10% je upisano kao MIDI sekvenca [1, *Abstract*]. Ovaj skup podataka ćemo koristiti za učenje klasifikatora.

3.1. Zastupljenost podataka

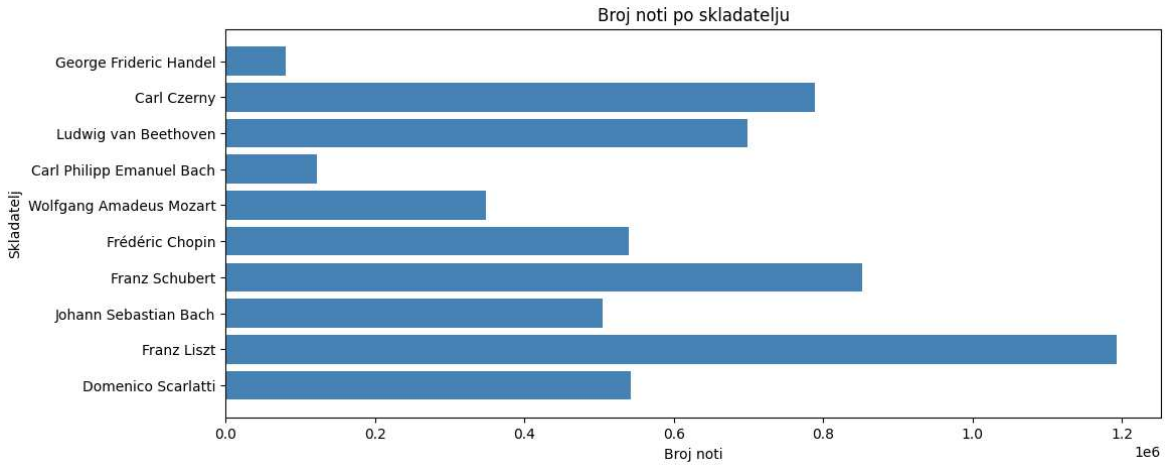
Uvjet klasifikacije bit će 10 skladatelja klasičnih djela. Za kratki pokaz klasi odabrano je prvih 10 najzastupljenijih skladatelja u skupu podataka. Na slici, 3.1., prikazan je broj djela prvih 10 skladatelja. Ovdje se primjećuje nebalansiranost primjera klasi ovakvim odabirom. Pri radu s ovim tipom podataka savršenu balansiranosst ne možemo očekivati te se nećemo previše koncentrirati na pretjerano uređivanje zasebnih klasi.



Slika 3.1. Broj MIDI datoteki po 10 najzastupljenijih skladatelja u skupu podataka

Pošto skladbe nisu jednake duljine bitno je vidjeti koliko je informacije, broj noti,

sadržano u svim MIDI datotekama pojedinog skladatelja što je prikazano na slici 3.2. Promatranjem ova dva grafa (3.1. i 3.2.) možemo zaključiti da količina informacije u klasama ne ovisi samo o količini MIDI datoteka već i broju noti. Prilikom odabira skladatelja bitna su nam dakle dva saznanja: broj zastupljenih MIDI datoteka i broj noti.



Slika 3.2. Broj noti zastupljenih u skupu podataka svakog skladatelja

3.2. Odabir skladatelja

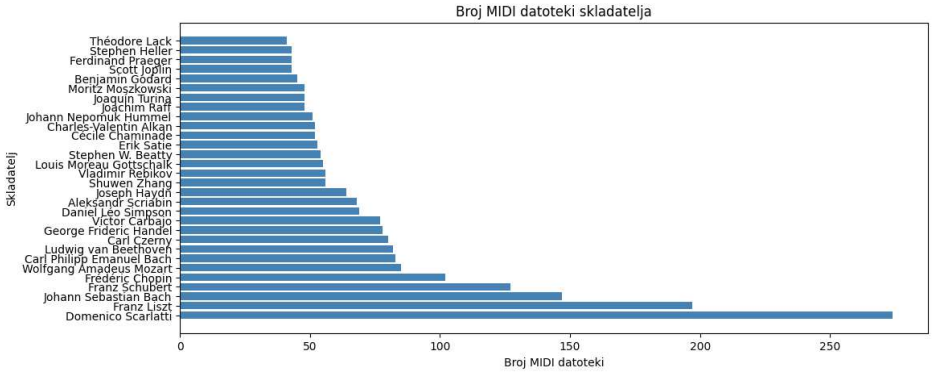
Nakon navedenih spoznaji o podatcima možemo odabrati skladatelje koje ćemo klasificirati. Dakle, u obzir uzimamo broj MIDI datoteka i broj noti zastupljenih u podatcima svakog skladatelja. Odabir je učinjen sljedećim kodom.

```
# Obrisi pri najzastupljeniji zapis
del(composers_by_notes[0])
del(composer_by_midi[0])

# Napravi presjek
targets = list(set(composers_by_notes) & set(composer_by_midi))

# Odaberi prvih 10 skladatelja
targets = targets[:10]
```

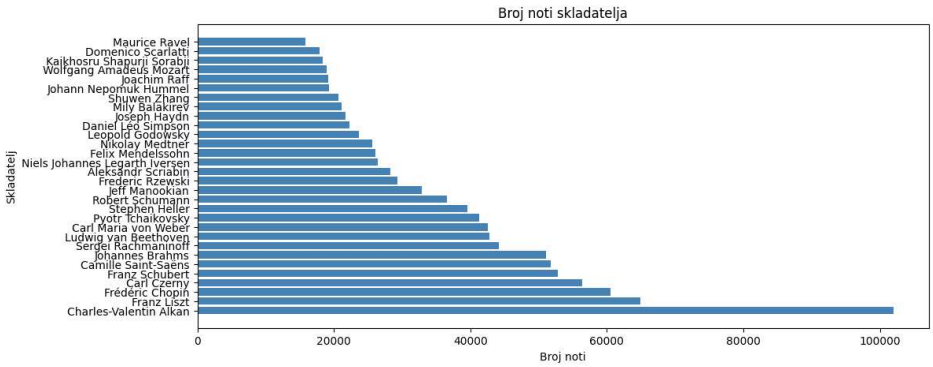
Liste *composers_by_midi* i *composers_by_notes* sadrže 30 prvih skladatelja sortirano po broju MIDI datoteka, tj. noti. Iz njih je uklonjen prvi skladatelj kako bismo postigli blagu balansiranost. Na slikama 3.3. i 3.4. vidljivo je kako ti zapisi strše. Zatim, u listu *targets*

učinjen je presjek i odabrano je prvih 10 skladatelja koji čine odabrane skladatelje za klasifikaciju.

```
targets = {
    'Franz Liszt': '0',
    'Wolfgang Amadeus Mozart': '1',
    'Stephen Heller': '2',
    'Aleksandr Scriabin': '3',
    'Daniel Leo Simpson': '4',
    'Ludwig van Beethoven': '5',
    'Johann Nepomuk Hummel': '6',
    'Carl Czerny': '7',
    'Shuwen Zhang': '8',
    'Joseph Haydn': '9'
}
```



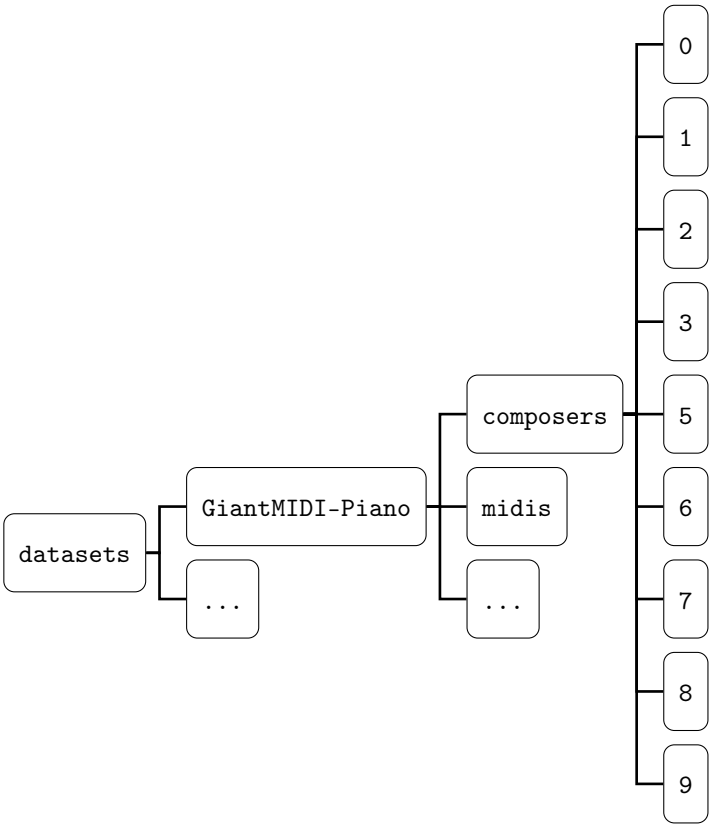
Slika 3.3. 30 najzastupljenijih skladatelja po broju MIDI datoteki



Slika 3.4. 30 najzastupljenijih skladatelja po broju noti

3.3. Podjela podataka

Prije no što počnemo s klasifikacijom preoblikovat ćemo skup podataka u novu datotečnu strukturu kako bismo iz imena lakše izlučili klasu. Unutar mape skupa podataka *GiantMIDI-Piano* stvorena je nova mapa *composers*, a u njoj 10 novih mapi nazvane rednim brojem klase skladatelja. U svaku od 10 tako kreiranih mapi iskopirane su MIDI datoteke pojedinog skladatelja. Ovakva podjela skladatelja u datoteke bit će nam od koristi u poglavlju *Klasifikacija metodama obrade prirodnog teksta* 5. Datotečna struktura prikazana je u nastavku.



4. Stroj potpornih vektora i klasifikator nasumičnih šuma

Klasifikacija podataka algoritmima stroja potpornih vektora, SVM (engl. *Support vector machine*) i klasifikatorom nasumičnih šuma, RF (engl. *Random forest classifier*) zahtijeva podatke s jednakim brojem značajki što znači da zapisi u skupu podataka moraju imati jednako dugu vektorsku reprezentaciju pridruženu pojedinoj klasi. U slučaju MIDI podataka to nije slučaj. Čak i kada bi se datoteke srezale na manje dijelove opet nismo sigurni što su naše značajke. Možemo pretpostaviti da je značajka redni broj note, a što ako je u pitanju akord, tj. više noti je pritisnuto u isto vrijeme, u tom slučaju također griješimo. Isto možemo reći ako je datoteke rascjepkamo na više njih, tada redni broj značajke nikako ne odgovara rednom broju note ili glazbene jedinice te se gubi trag melodiji. Stoga prije no što pokušamo klasificirati naše podatke moramo smisliti mehanizam ekstrakcije značajki.

4.1. Ekstrakcija značajki

Problem različite duljine skladbi pokušat će se riješiti korištenjem algoritma analize glavnih komponenata, PCA (engl. *Principal component analysis*) koji će biti opisan u ovom poglavlju. No, prije nego što iskoristimo PCA trebamo nekako iz podataka izvući informacije o notama. U tome će nam pomoći *Python* biblioteka *pretty_midi*.

4.1.1. *Pretty_midi*

Pretty_midi [4] biblioteka je razvijena u programskom jeziku Python za olakšano barataanje MIDI protokolom. Također, omogućava korisniku da stvori tok MIDI poruka iz MIDI datoteke i filtrira samo potrebne poruke.

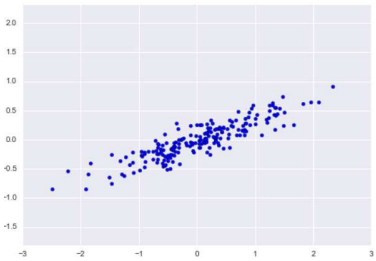
Svaka MIDI datoteka u sebi sadrži informaciju o notaciji skladbe u obliku noti. Dakle,

svaka nota ima svoju visinu tona (*pitch*), jačinu tona (*velocity*), početak (*note_start*) i kraj (*note_end*) trajanja note. Pošto su *note_start* i *note_end* značajke vremenske i rastuće ujedinit ćemo ih u novu značajku trajanja note *duration*. Dobivenom značajkom *duration* ćemo zamijeniti značajku *note_end*.

Nakon što znamo što tražimo, iz svake MIDI datoteke uz pomoć biblioteke *pretty_midi* izvlačimo četiri glavne značajke iz svake note: *pitch*, *velocity*, *note_start* i *duration*. Tako dobivene vrijednosti za svaku notu spremaju se u listu. Dakle, za skup podataka MIDI datoteka veličine (N) sada smo dobili skup podataka dimenzija (N, n_i, 4), gdje je n_i proizvoljan za svaku MIDI datoteku. Ovdje uočavamo problem varirajuće dimenzije n_i svakog MIDI zapisa. Upravo taj problem ćemo riješiti algoritmom PCA.

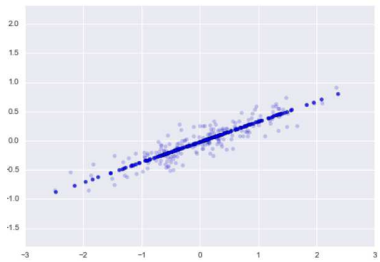
4.1.2. Analiza glavnih komponenata

Analiza glavnih komponenata ili PCA, engl. *Principal component analysis* jednostavna je i brza nenadzirana metoda strojnog učenja redukcije dimenzionalnosti podataka [5, *Introducing Principal Component Analysis*]. Ovdje je bitno napomenuti kako se radi o metodi koja smanjuje dimenzionalnost podataka stoga imamo ograničenje minimalnom duljinom skladbe u skupu podataka. Algoritam ovdje neće biti pokazan već će ga se koristiti kao *crnu kutiju*. Na slikama 4.1., 4.2. i 4.3. slikovito je prikazano kako algoritam u suštini djeluje. U podatcima 4.1. prepoznaju se glavne komponente 4.2. te se ona manje važnije svodi na nulu 4.3. i time se reduciraju podatci.



Slika 4.1. Skup podataka prije primjene PCA metode [5, *Introducing Principal Component Analysis*]

Slika 4.2. Glavne komponente [5, *Introducing Principal Component Analysis*]



Slika 4.3. Skup podataka nakon obrade PCA metodom [5, *Introducing Principal Component Analysis*]

Dakle, kako bismo uspješno saželi podatke dobivene iz MIDI datoteka potrebno je učiniti više koraka PCA algoritma u više dimenzija. PCA je izvršen po uzoru na odlomak

iz knjige [5, *Introducing Principal Component Analysis*] i pseudokod za sažimanje MIDI informacije pokazan je u nastavku:

```
reduce_features(notes: list, num_features: int) -> list:
    pca_1 = PCA(n_components=1)
    features = pca_1.fit_transform(notes)

    chunks = floor(len(features) / num_features)
    features = features[:int(chunks * num_features)]
    features = features.reshape(num_features, chunks)

    pca_2 = PCA(n_components=1)
    features = pca_2.fit_transform(features)

    return featuress
```

Početno polje *notes* dimenzije je *dim(n, 4)*, sadrži n note, a svaka nota sadrži 4 vrijednosti. Prvo korištenje PCA algoritma reducira 4 vrijednosti svake note u jednu komponentu *n_components* i dobivamo novo polje *features* dimenziji *dim(n, 1)*. Nakon tog koraka grupiramo vrijednosti polje **features** u skupine, *chunks* te tako dobivamo broj skupini ekvivalentan broju željenih značajki *num_features*. U posljednjem koraku još jednim korakom PCA algoritma reduciramo skupine na jednu vrijednost te dobivamo polje *features* veličine *num_features*.

Ovakvim korištenjem PCA algoritma reducirali smo dimenzije MIDI podatka na sljedeći način:

$$dim(n_i, 4) \rightarrow dim(n_i, 1) \rightarrow dim(n, 1),$$

gdje je n_i količina noti u i-tom MIDI zapisu, a n količina značajki nakon provođenja algoritma.

Bitno je napomenuti da ovakvim pristupom ne dobivamo smislene informacije o okolini neke note već pokušavamo naprosto pogoditi vrijednosti koje će biti svojstvene okolini određene note. Također, pošto je glazba slijed noti, ne možemo nikako znati dali je veličina *chunk*-a smislena za određenu notu, tj. jesmo li veličinom *chunk*-a pogodili

neku melodiju ili akord. S ovim činjenicama možemo pokušati naučiti naše klasifikatore stroja potpornih vektora i slučajnih šuma.

4.2. Učenje klasifikatora

Nakon podjele skupa podataka u klase u datotečnu strukturu kao što je prikazao na grafu u potpoglavlju *Podjela podataka* 3.3. jednostavno možemo učitati puteve do datoteka u jednu listu. Kada smo tako podijelili podatke sada ih možemo razvrstati u skup za učenje i skup za treniranje u omjeru 80/20. Također, oznaku MIDI datoteke sada lako možemo izlučiti iz puta do datoteke uz pomoć neke arbitrarne funkcije. To se može učiniti na sljedeći način:

```
midi_paths = list(Path(dataset_path, "by_composer")
midi_paths = midi_path.glob("**/*.mid"))
random.shuffle(midi_paths)

train_files = midi_paths[:int(len(midi_paths) * 0.8)]
test_files = midi_paths[
    int(len(midi_paths) * 0.8):int(len(midi_paths) * 1.0)
]
```

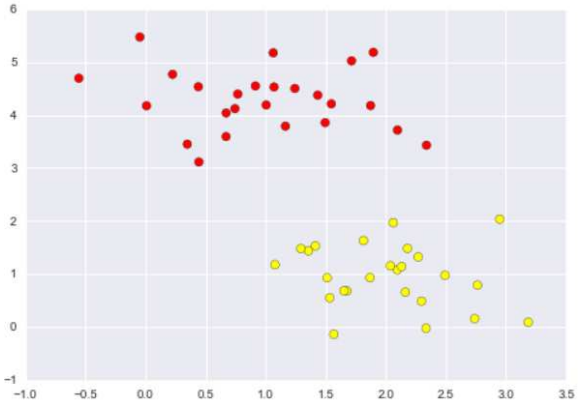
Prije no što na učitanim podatcima primijenimo PCA potreban nam je još jedan parametar, a to je minimalan broj noti u MIDI datoteci. Tako ćemo, pošto činimo redukciju podataka, neke datoteke jednostavno zanemariti.

```
number_of_features = n
min_notes = number_of_features

train_midis, train_labels = load_files(train_files, min_notes)
test_midis, test_labels = load_files(test_files, min_notes)

X_train = extract_features_batch(train_midis, number_of_features
    )
X_test = extract_features_batch(test_midis, number_of_features)
```

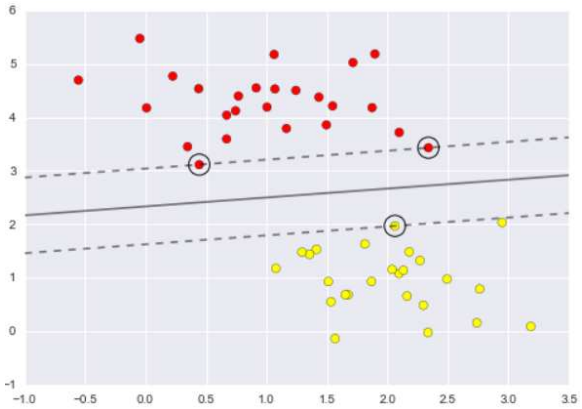
U odsječku koda funkcija *load_files* uz pomoć biblioteke *pretty_midi* učitava podatke kao što je predstavljeno u poglavlju 4.1.1. te iz putanje do datoteke izluči oznaku klase dok funkcija *extract_features* vrši PCA algoritam predstavljen u poglavlju 4.1.2. U klasifikaciji će se probati više vrijednosti varijable *num_features*: 50, 75, 150, 250 i 500 te za svaku vrijednost naučit oba klasifikatora, SVM i RF.

4.2.1. Stroj potpornih vektora

Stroj potpornih vektora, skraćeno SVM, engl. *Support Vector Machine*, iznimno je moćan i fleksibilan algoritam nadziranog učenja korišten za klasifikaciju ili regresiju [5, *In-Depth: Support Vector Machines*]. Algoritam u suštini funkcionira tako da između dvije odvojene klase podataka nastoji povući pravac koji maksimizira minimalnu marginu između najbližih podataka u pojedinim klasama.



Slika 4.4. Skup podatak koji sadrži dvije klase [5, *In-Depth: Support Vector Machines*]



Slika 4.5. Podatci u skupu odvojeni marginom [5, *In-Depth: Support Vector Machines*]

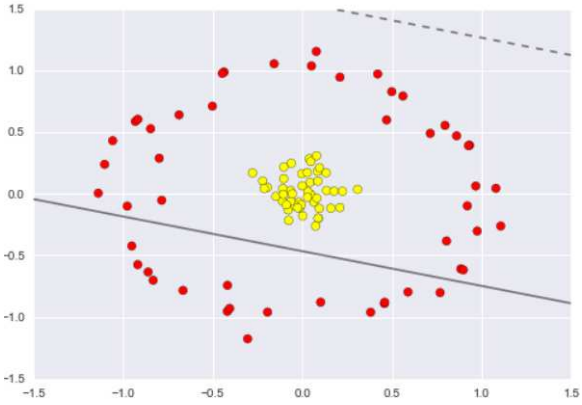
Na slikama 4.4. i 4.5. pokazan je skup podataka odvojen pravcem koji maksimizira minimalnu marginu između najbližih podataka dvije klase. Vektori koji se nalaze točno na margini, zaokruženi na slici, nazivaju se potporni vektori. U slučaju naše klasifikacije MIDI datoteka postoji 10 klasi, no taj problem je rješiv algoritmima OvO (*One-vs-One*) ili OvA (*One-vs-All*) koji su već implementirani u sklopu SVM algoritma iz biblioteke *skit-learn* [6].

SVM algoritam moguće je konfigurirati parametrima. Parametre kojima će SVM klasifikator biti konfiguriran su: *kernel*, *C* i *gamma*. Za svaku konfiguraciju parametara u parametarskom *grid*-u naučit će se model za pojedinu vrijednost varijable *num_features*

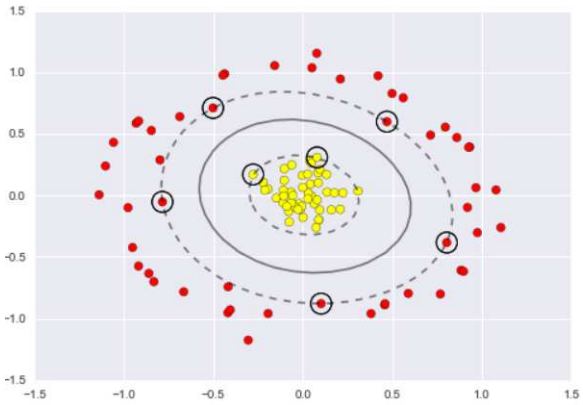
predstavljene u poglavlju 4.2. Značenje pojedinog parametar i na što njihova promjena utječe bit će predstavljeno u nastavku. Također, matematika SVM-ova neće biti predstavljena, već samo kako ih koristiti u sklopu problema klasifikacije korištenjem već implementiranog algoritma iz biblioteke *skit-learn* [6].

```
param_grid = {
    'kernel': [rbf]
    'C': [1, 5, 10, 50],
    'gamma': [0.0001, 0.0005, 0.001, 0.005]
}
```

Kernel (ili jezgra) parameter omogućava nam projekciju podataka u višedimenzionalni prostor definiran polinomima ili Gaussovim baznim funkcijama i time definira nelinearnu granicu između podataka [5, *In-Depth: Support Vector Machines, Beyond linear boundaries: Kernel SVM*]. Na slikama 4.6. i 4.7. slikovito je prikazano što *rbf* jezgra čini. Upravo tu jezgrenu funkciju ćemo koristiti u konfiguraciji klasifikatora.



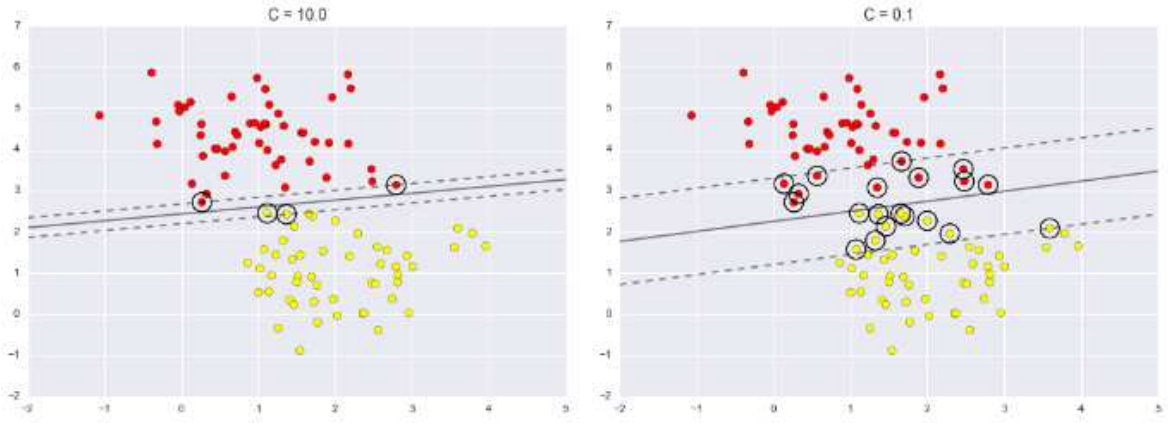
Slika 4.6. Skup podatak koji ima nelinearnu granicu između klasi [5, *In-Depth: Support Vector Machines*]



Slika 4.7. Nelinearnost modelirana *rbf* jezgrom[5, *In-Depth: Support Vector Machines*]

C parametar kontrolira takozvanu mekoću margine. U stvarnom svijetu, našem primjeru klasifikacije, granica između klasi nije jasna linija, tj. podaci jedne klase ulaze u prostor druge pošto su često marginalni primjeri sličnih vrijednosti. Iz navedenog razloga nekim podacima moramo pustiti da uđu unutar margine, tj. u prostor između potpornih vektora naših klasa. Taj pristup naziva se omekšavanje margine i kontroliran je parametrom C. Parametar C poprima vrijednosti između 0 i $+\infty$ i što je parametar manji

margina je mekša, tj. više primjera puštamo u marginu, a što je parametar veći margina je tvrđa i manje primjera se nalazi u njoj. Može se reći da ovaj parametar regularizira naš model, tj. smanjenjem parametra C model postaje manje sklon prenaučenosti skupu podataka za učenje. Na slici 4.8. prikazan je utjecaj veličine parametra C na marginu.



Slika 4.8. Utjecaj pametra **C** na mekoću margine [5, *In-Depth: Support Vector Machines*]

Parametar gamma koristan je samo ako je za jezgru SVM modela odabrana rbf jezgra. Taj parametar kontrolira zakrivljenost linije naše jezgre [5, *In-Depth: Support Vector Machines*]. Što je gamma veći krivulja je zakrivljenija i prilagođenija na podacima za učenje. Stoga smanjenjem ovog parametra također utječemo na regularizaciju modela.

Predstavljenim parametrima SVM klasifikatora sada možemo naučiti model na sljedeći način:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

estimator = SVC(kernel='rbf', class_weight='balanced')
grid = {
    'C': [10e-3, 10e-2, 10e-1, 0.1, 0.5, 1, 5, 10, 50, 100, 500,
        1000],
    'gamma': [10e-3, 10e-2, 10e-1, 0.1, 0.5, 1, 5, 10, 50, 100,
       500, 1000]
}

grid_search = GridSearchCV(estimator, grid)
```

```
grid_search.fit(X_train, y_train)
y_pred = grid_search.predict(X_test)
```

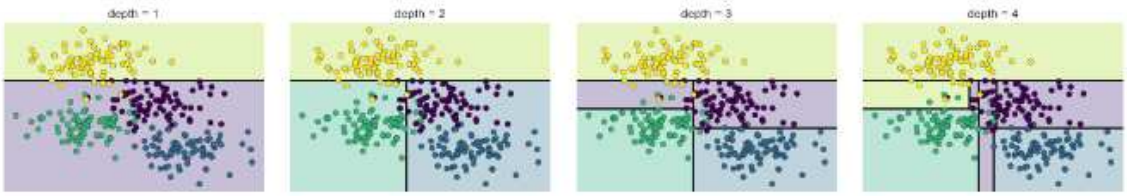
Varijabla *estimator* stvara objekt klase SVC koji u sebi sadrži algoritme za računanje potpornih vektora. Klasa *GridSearchCV* namjesti estimator i *grid* koje smo zadali, a metodom *fit grid*-om i skupovima za treniranje *X_train* i *y_train* optimira se *estimator*. Metoda *predict* predviđa oznake za neoznačeni skup *X_test* i dobivamo oznake *y_pred*.

Rezultati u obliku točnosti, engl. *accuracy* i F1-vrijednosti prikazani su u potpoglavlju 4.3.

4.2.2. Klasifikator slučajnih šuma

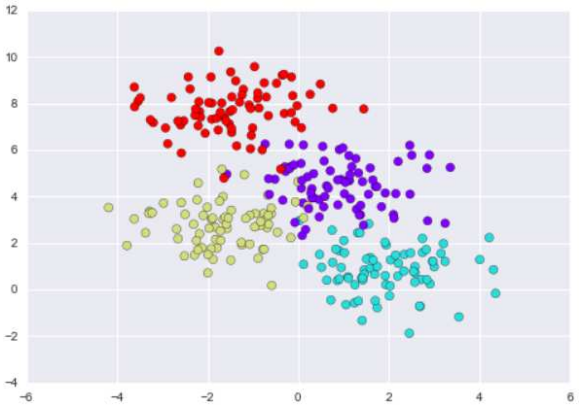
Klasifikator slučajnih šumi ili skraćeno RF, engl. Random forest, za razliku od SVM-a neparametarski je klasifikator [5, *In-Depth: Decision Trees and Random Forests*]. Također, RF klasifikator je ansambl metoda što znači da se oslanja na skupljene podatke skupa jednostavnijih klasifikatora [5, *In-Depth: Decision Trees and Random Forests*]. Logika iza ansambla je ta da je *zbroj veći od pojedinačnih dijelova*, dakle glasanje među više klasifikatora donosi bolju odluku od svakog klasifikatora pojedinačno [5, *In-Depth: Decision Trees and Random Forests*]. RF primjer je ansambla skupa klasifikatora zvanih stablima odluke.

Stabla odluke veoma su intuitivan algoritam klasifikacije podataka [5, *In-Depth: Decision Trees and Random Forests*]. Možemo ih zamisliti kao skup pitanja kojima dolazimo do zaključku o objektu klasifikacije. Takva binarna podjela odluke ako su pitanja dobro koncipirana čini jako efikasnu klasifikaciju. U strojnom učenju pitanja generalno poprimaju oblik podjele podataka po nekoj osi u prostoru parametara. Dakle za svaku dubinu stabla podaci su podijeljeni nekom hiperravninom u prostoru podataka i tako su stvorene zone u kojima pripada pojedinačna klasa kao što je prikazano na slici 4.9.

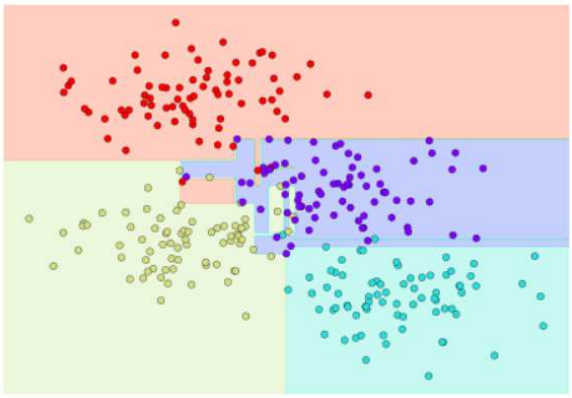


Slika 4.9. Promjena granica klasifikacije stablom odluke dubinom stabla *depth* [5, *In-Depth: Decision Trees and Random Forests*]

Na slikama 4.10. i 4.11. kao što je to pokazano u knjizi [5, *In-Depth: Decision Trees and Random Forests*] vidi se kako stablo odluke razvrstava klase u svoja područja unutar prostora značajki. Također, vidljivo je kako ta podjela nije savršena pošto su se neki primjeri zavukli i uzrokovali svoja područja unutar drugih klasi i tu možemo reći da se klasifikator prenauči. Takva prenaučenost jedna je od značajki stabla odluke pošto je jako lako ući preduboko u stablo i izazvati rupe kao na slici 4.11. [5, *In-Depth: Decision Trees and Random Forests*]. Taj problem prenaučenosti rješava se korištenjem algoritma nasumičnih šumi [5, *In-Depth: Decision Trees and Random Forests*].



Slika 4.10. Nepodijeljen skup podataka [5, *In-Depth: Decision Trees and Random Forests*]



Slika 4.11. Skup podijeljen stablom odluke [5, *In-Depth: Decision Trees and Random Forests*]

Ideja da više prenaučenih klasifikatora može biti iskorišteno kao jedan kako bi se smanjila prenaučenost i povećala generalizacija naziva se *bagging*. U praksi stabla odluke najbolje funkcioniraju kada su nasumična, unošenjem stohastičnosti u podjelu podataka hiperravninom. [5, *In-Depth: Decision Trees and Random Forests*] U biblioteci *skit-learn* [6] stohastičnost je implementirana klasom *RandomForestClassifier* koji takvu nasumičnost automatizira; sve što je potrebno učiniti jest dati broj klasifikatora varijablom *n_estimators* [5, *In-Depth: Decision Trees and Random Forests*]. U nastavku je is-

ječkom koda prikazano kako iskoristiti algoritam nasumičnih šuma koristeći biblioteku *skit-learn*. Također, kao i u prijašnjem poglavlju 4.2.1. namješten je grid i prikazan je cijeli postupak učenja klasifikatora.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

estimator = RandomForestClassifier()
grid = {
    'n_estimators': [50, 100, 250, 500, 750, 1000]
}

grid_search = GridSearchCV(estimator, grid)
grid_search.fit(X_train, y_train)
y_pred = grid_search.predict(X_test)
```

Rezultati ovog algoritma u obliku točnosti, engl. *accuracy* i F1-vrijednosti također su prikazani sljedećem potpoglavlju 4.3.

4.3. Rezultati

Rezultati su prikazani u tablicama *Accuracy* 4.1. i *F1-macro* vrijednostima 4.2. te matricama zabune u potpoglavlju *Matrice zabune* 4.3.1. Prije nego što pokažemo rezultate objasnit ćemo ove pojmove i što oni znače.

Accuracy (točnost) je udio ispravno klasificiranih primjera među svim uzorcima. Daje nam uvid u to koliko je puta model pogodio točnu klasu. *Accuracy* nas može lako zavarati jer model može imati veliku točnost predviđajući samo najzastupljeniju klasu, a to je često tako pri treniranju modela nebalansiranim skupom podataka. Računa se na idući način:

$$Accuracy = \frac{n_{ispravno\_klasificirani\_uzorci}}{n_{upupan\_broj\_uzoraka}}.$$

F1-macro vrijednost je srednja vrijednost *F1-score* vrijednosti za svaku klasu. *F1-score*

za pojedinu klasu računa se kao harmonijska sredina između preciznosti (*Precision*) i osjetljivosti (*Recall*) te klase. *F1-macro* je bolji u situacijama gdje postoji neuravnoteženost među klasama, jer uzima u obzir preciznost i osjetljivost svake pojedine klase. Računa se na sljedeći način:

$$F1\_score_i = 2 \cdot \frac{Precision_i \cdot Recall_i}{Precision_i + Recall_i}$$

$$F1\_macro = \frac{1}{n} \cdot \sum_{i}^{n} F1\_score_i.$$

Precision (preciznost) klase i odnosi se na udio točno pozitivnih predikcija klase i (TP) u odnosu na sve predikcije koje je model klasificirao točno ($TP + FP$), gdje su FP svi negativni primjeri koje je model klasificirao točno. Preciznost se računa na sljedeći način:

$$Precision = \frac{TP}{TP + FP}.$$

Recall (osjetljivost) odnosi se na udio ispravno klasificiranih pozitivnih uzoraka u odnosu na sve ispravno klasificirane uzorke ($TP + TN$), gdje su TN svi negativni uzorci koje je model klasificirao negativno. Računa se ovako:

$$Recall = \frac{TP}{TP + TN}.$$

Iz tablica *Accuracy* 4.1. i *F1-macro* možemo očitati najviši *accuracy* za modele s 250 parametara i iznose 0.42 i 0.40. Također najviše *F1-macro* vrijednosti očitavaju se za iste modele te iznose 0.31 i 0.33. Ovakva razlika u ovim vrijednostima nalaže nam da skup podataka nije dobro balansiran i vjerojatno se model prilagodio jednoj klasi. Kako bismo vjernije prikazali rezultate treniranih modela, također smo rezultate pokazali i u obliku matrica zabune gdje se nebalansiranost podataka može lakše vidjeti.

Accuracy					
Param.	50	75	150	250	500
SVM	0.35	0.31	0.38	0.42	0.37
RF	0.35	0.33	0.37	0.40	0.31

Tablica 4.1. *Accuracy* ovisno o broju parametara i klasifikatoru

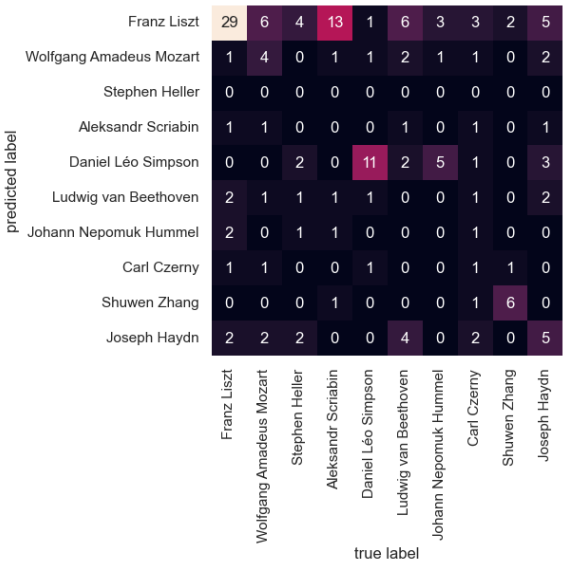
F1-macro					
Param.	50	75	150	250	500
SVM	0.25	0.22	0.30	0.31	0.25
RF	0.22	0.24	0.31	0.33	0.23

Tablica 4.2. *F1-macro* ovisno o broju parametara i klasifikatoru

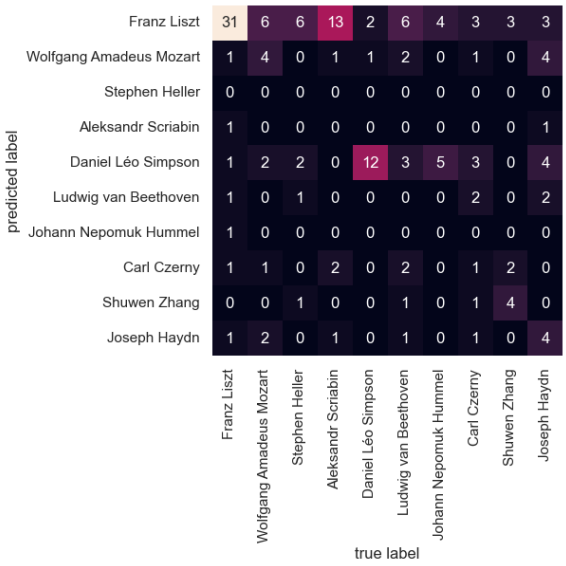
4.3.1. Matrice zabune

Matrica zabune (*confusion matrix*) je matrica iz koje možemo ocijeniti performanse modela. Ona prikazuje stvarne i predviđene klase i time omogućava uvid u to koliko je primjera završilo u točnoj, tj. krivoj klasi. U našim matricama zabune koje slijede točne klase nalaze se na stupcima, a njihove predikcije u redcima. Broj koji se nalazi na dijagonali matrice označava broj točno klasificiranih primjera pojedine klase. Što je više primjera zastupljeno na dijagonali matrice zabune to model bolje generalizira.

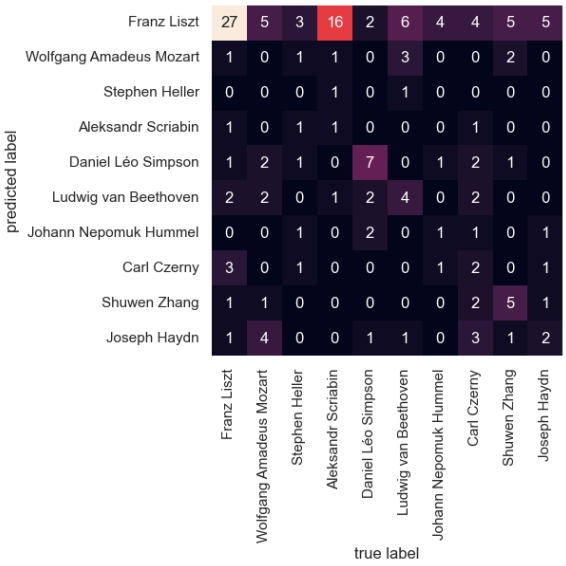
U sljedećim matricama zabune uočavamo kako je najviše primjera krivo klasificirano u klase *Franz Liszt* i *Daniel Leo Simpson*. To se može uočiti u skoro svim matricama zabune. U tablicama *Accuracy* i *F1-macro* modela SVM i RF s 250 parametara ne uočavamo znatno različite vrijednosti, no u njihovim matricama zabune možemo vidjeti kako ne klasificiraju primjere jednoliko. Dok je SVM većinu primjera smjestio u klasu *Franz Liszt* i jedino dobro prepoznao klasu *Daniel Leo Simpson* model RF donekle je popunio dijagonalu i tu možemo reći da RF s 250 parametara bolje generalizira od SVM-a s 250 parametara.



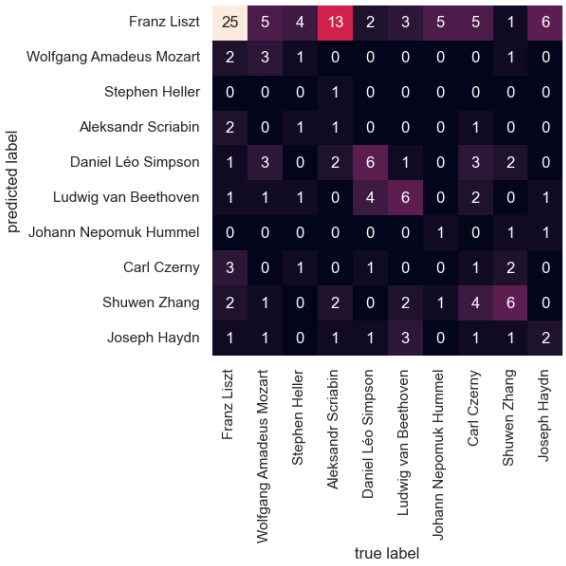
Slika 4.12. Matrica zabune SVM modela s 50 parametara



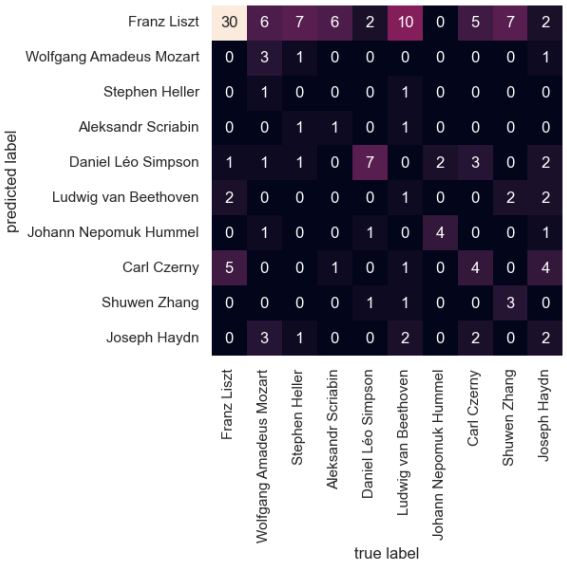
Slika 4.13. Matrica zabune RF modela s 50 parametara



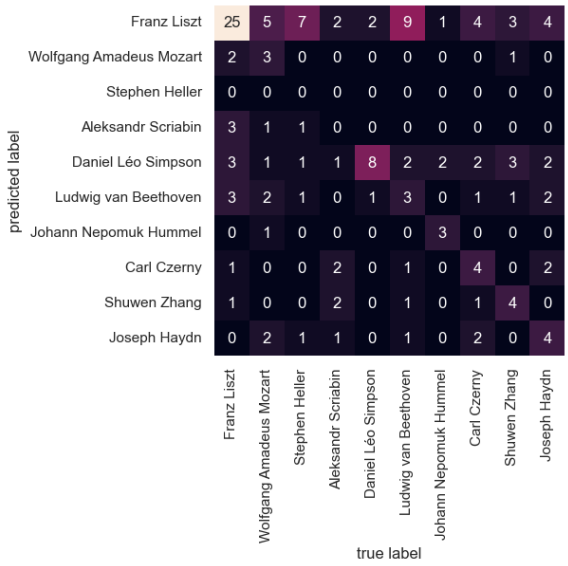
Slika 4.14. Matrica zabune SVM modela s 75 parametara



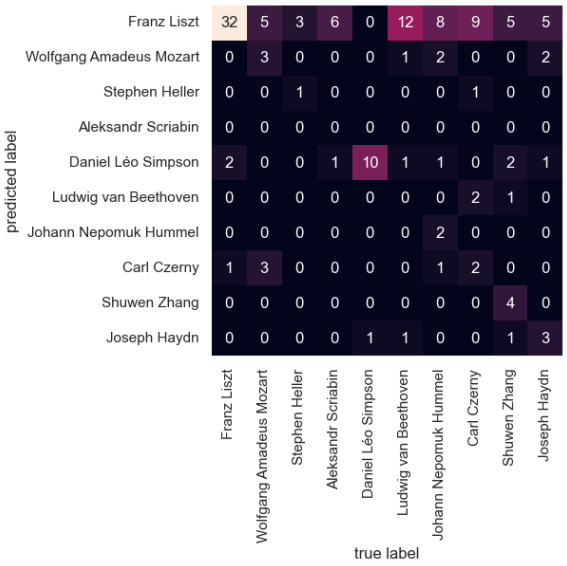
Slika 4.15. Matrica zabune RF modela s 75 parametara



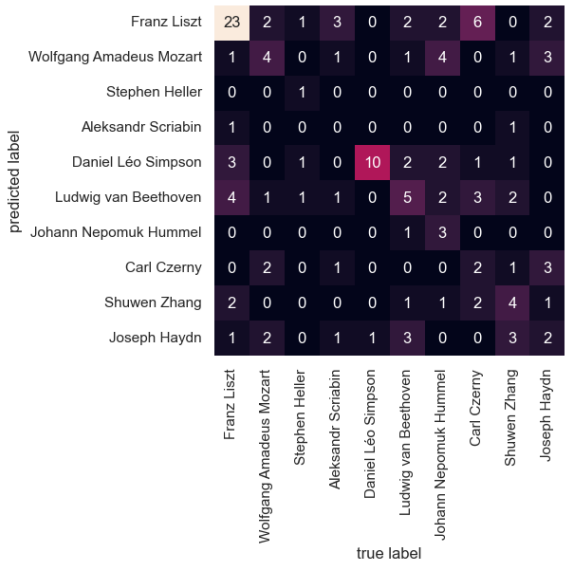
Slika 4.16. Matrica zabune SVM modela s 150 parametara



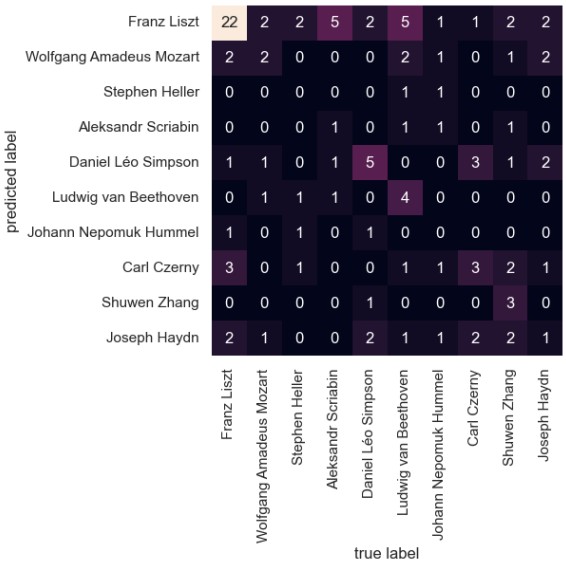
Slika 4.17. Matrica zabune RF modela s 150 parametara



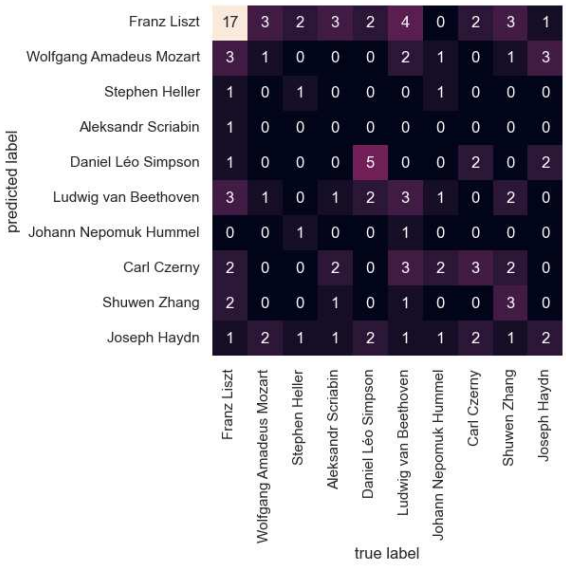
Slika 4.18. Matrica zabune SVM modela s 250 parametara



Slika 4.19. Matrica zabune RF modela s 250 parametara



Slika 4.20. Matrica zabune SVM modela s 500 parametara



Slika 4.21. Matrica zabune RF modela s 500 parametara

5. Klasifikacija metodama obrade prirodnog teksta

MIDI datoteke skup su instrukcija koje se kao što smo objasnili u poglavlju 2. mogu svesti na skup poruka. Među tim porukama lako se filtriraju one svojstvene za glazbeni zapis samog djela, tablica 2.1. Također, ništa nas ne sprječava da te poruke zapišemo u tekstualnom zapisu u neku datoteku i tako dobijemo tekst nastao iz MIDI poruka. Sada taj tekst možemo dovesti na ulaz nekog dubokog modela i naučiti ga. Dakle, ovim pristupom uklonili smo potrebu za dobivanjem i svođenjem MIDI datoteka na skup parametara, kao u poglavlju 4. te ih pretvorili u sekvence znakova i time sačuvali bitnu vremensku komponentu glazbe.

Pristup obradi MIDI datoteka prirodnim tekstom podijeljen je na dva glavna dijela: tokenizacija MIDI datoteka i model klasifikatora tokena. Tokenizacijom skupa podataka dobivamo *rječnik* tokena, engl. *vocab*, kojima razlučujemo smislene cjeline u našem tekstu MIDI poruka. Za MIDI datoteke algoritmi koji tokeniziraju MIDI poruke već postoje i jedan od njih je REMI. REMI tokenizator [7] poslužit će nam za tokenizaciju melodijskih, ritmičnih i harmonijskih dijelova MIDI poruka. Zatim će se u sklopu tokenizacije predstaviti i biblioteka *MidiTok* [8] napisanan u programskom jeziku *Python*. *MidiTok* pruža nam pripremu MIDI datoteka za tokenizaciju te već enkapsulira upotrebu i učenje *REMI* klasifikatora algoritmom po izboru. Također, *MidiTok*-om podatci se mogu lako pripremiti i pretvoriti u sekvence za obradu.

Nakon učenja tokena iz skupa za učenje i pretvorbe podataka u sekvence, njima možemo naučiti duboke mreže. U ovom radu istražit će se povratne neuronske mreže pritom koristeći GRU, engl. *Gated recurrent unit* i LSTM, engl. *Long short-term memory* ćelije. Također, u njih će se i ugraditi mehanizam pozornosti, *attention*. Naučeni modeli bit će tablično uspoređeni i opisani kao i u poglavlju 4.3.

5.1. Tokenizacija

Tokenizacija skupa podataka je pronalazak smislenih leksičkih jedinica, tokena, svojstvenih za obradu podatka u kojima se pojavljuju i neophodan je dio obrade prirodnog teksta. Pošto MIDI datoteke možemo pretvoriti u tekstualni zapis te sadrže glazbene uzorke MIDI poruka, u njima također možemo prepoznati tokene. No, kako bismo uopće stvorili rječnik tokena, potrebno je prvo odrediti početne tokene na temelju kojih ćemo izgraditi cijeli rječnik. Odabir početnih tokena i učenje novih odradit će se u dva koraka korištenjem tokenizatora REMI i biblioteke koja enkapsulira REMI i daje nam na izbor više tokenizacijskih algoritama *MidiTok* kao što su BPE ili Unigram.

5.1.1. REMI

Predstavljen u radu *Pop Music Transformer (Huang and Yang)* [7] REMI, *REvamped-MIDI*, model je transformera za prepoznavanje melodijskih i ritmičkih strukturi u MIDI porukama. REMI nastoji iz previše zrnate uređenosti MIDI poruka prepoznati kompleksnije strukture i njima ih zamijeniti ne narušavajući time ritam i harmoniju zapisane glazbe. Neki učestali REMI tokeni prikazani su u tablici 5.1. Broj REMI tokena jedne MIDI datoteke ne odgovara njenom broju poruka.

REMI token	
Naziv tokena	Opis
Bar	Početak nove glazbene jednine
Position_<n>	Pozicija note
Pitch_<n>	Visina tona
Velocity_<n>	Dinamika tona
Duration_<n>	Trajanje note
Rest_<n>	Glazbena pauza
Chord_<type>	Tip akorda
...	...

Tablica 5.1. česti REMI tokeni i njihova značenja

Na primjer, neki niz REMI tokena može izgledati ovako:

```
[..., 'Rest_7.0.2', 'Rest_0.3.8', 'Position_27', 'Pitch_62', '
   Velocity_79', 'Duration_0.2.8', 'Position_29', 'Pitch_64', '
   Velocity_79', 'Duration_0.3.8', 'Position_31', 'Pitch_65', '
   Velocity_79', 'Duration_0.3.8', ...]
```

5.1.2. MidiTok

REMI tokenizator neće se koristiti izravno, već u sklopu *Python* paketa *MidiTok* [8]. *MidiTok* tokenizira MIDI datoteke enkapsulirajući pritom najpoznatije MIDI tokenizatore među kojima je i REMI. Pruža i algoritme poput BPE, *Unigram* i *WordPiece* za učenje tokenizatora. U nastavku ćemo pokazati kako se paket koristi, odabrat ćemo REMI tokenizator i BPE algoritam učenja tokenizacije. Također, pokazat ćemo i kako se podatci mogu pripremiti za učenje modela.

Prije nego što pokažemo kako korištenjem *MidiTok* paketa možemo naučiti tokenizator objasnit ćemo BPE algoritam.

BPE, engl. *Byte-Pair Encoding* originalno je osmišljen za kompresiju teksta, a nakon toga korišten u mnogim današnjim modelima transformera kao što su *GPT*, *GPT-2*, *BART* itd. BPE tokenizacija podijeljena je u sljedećih 5 koraka:

1. Određivanje inicijalnih tokena: U ovom koraku tekst se razbije na najosnovnije jedinice, tokene, najčešće znakove tj. slova ili skupove slova te je svakoj jedinici dodijeljen je jedan bajt. U slučaju obrade MIDI datoteka one su podijeljene na REMI tokene, 5.1. te je svakom tokenu dodijeljen identifikacijski broj, *id*, to jest preko *id*-a jedan bajt.

2. Brojanje frekvenciji tokena: Nakon definiranja osnovnih tokena algoritam uparuje sve tokene u tekstu (u slučaju MIDI datoteka to je skup datoteka za učenje) te ih broji.

3. Spajanje najčešćeg para: Nakon brojanja parova najučestaliji par se spaja u novi token. U slučaju REMI tokenizatora to se učini na način da se bajtovi dva tokena spoje u novu riječ, a njoj se dodijeli novi *id*.

4. Ponavljanje koraka 2. i 3.: Sada s novom riječi u rječniku koraci 2. i 3. se ponavljaju dok nije postignuta željena veličina rječnika, tj. *vocab*-a.

5. Spremanje rječnika: Nakon što smo dobili zadovoljavajuću veličinu rječnika ili je algoritam prošao korak 4. bez promjene generiran je rječnik (*vocab*) kojim možemo tokenizirati novi tekst.

Učenje rječnika, skupa tokena, *vocab* korištenjem *MidiTok* paketa prikazano je u odsječku koda koji slijedi:

```
from miditok import REMI, TokenizerConfig

tokenizer_type = "BPE"
vocab_size = 10000

config_params = {
    "pitch_range": (0, 127),
    "num_velocities": 16,
    "special_tokens": ["PAD", "BOS", "EOS"],
    "use_chords":True,
    "use_rests": True,
    "use_tempos": False,
    "use_time_signatures": False,
    "use_programs": False
}

config = TokenizerConfig(**config_params)
tokenizer = REMI(config)
tokenizer.train(vocab_size=vocab_size, model=tokenizer_type,
   files_paths=train_files)
```

Iz paketa *MidiTok* prvo učitamo REMI i *TokenizerConfig*. U varijable *tokenizer_type* i *vocab_size* pohranimo ime tokenizatora, u našem slučaju BPE, i broj riječi u rječniku. *Config_params* definira parametre kojima ćemo inicijalizirati objekt *TokenizerConfig* tj. njime ćemo zadati željene početne REMI tokene. *Pitch_range* će definirati *Pitch* tokene, *num_velocities velocity* tokene, itd. iz tablice 5.1. Također, ovdje vidimo i specijalne tokene *PAD*, *BOS* i *EOS*, oni nam pomažu pri učenju modela kako bi dobili zadovoljavajuće veličine ulaznog niza, no o njima ćemo pričati u poglavlju *Povratne neuronske mreže*, 5.2. Konfiguracija se zatim proslijedi u objekt *REMI* koji stvara tokenizator *tokenizer*. Tokenizator se metodom *train* trenira na skupu podataka za učenje *train_files* korištenjem BPE algoritma te dobivamo rječnik (*vocab*) veličine *vocab_size*.

U nastavku ćemo još pokazati kako paketom *MidiTok* korištenjem naučenog tokenizatora možemo pripremiti MIDI datoteke za učenje. Dakle, cilj je svaku MIDI datoteku rascjepkati u manje jednake dijelove kako bismo ih lakše doveli na ulaz modela koji učimo. Isječak koda koji slijedi pokazuje upravo to:

```
from miditok.pytorch_data import
    DatasetMIDI,
    split_files_for_training
from src.util.dataset import get_label_from_file_path

max_seq_len = 512
chunks_path = Path(dataset_path, "midi_chunks")

train_chunks = split_files_for_training(
    train_files,
    tokenizer,
    Path(chunks_path, "train"),
    max_seq_len)

train_dataset = DatasetMIDI(
    files_paths=train_chunks,
    tokenizer=tokenizer,
    max_seq_len=max_seq_len,
    bos_token_id=tokenizer["BOS_None"],
    eos_token_id=tokenizer["EOS_None"],
    func_to_get_labels=get_label_from_file_path
)
```

Iz paketa *MidiTok* učitane su klasa *DatasetMIDI* i metoda *split_file_for_training*. U varijable *max_seq_len* pohranjen je veličina sekvenci u koje MIDI datoteke dijelimo, a u varijablu *chunks_path* put do direktorija u koji ćemo spremiti novonastale sekvence. Metodom *split_file_for_training* podijelili smo datoteke, te klasom *DatasetMIDI* stvorili skup podataka za treniranje. Bitno je primjetiti kako sad koristimo tokene *BOS*, engl. *Beginning of Sequence* i *EOS*, engl. *End of Sequence* koji označuju kada nova sekvenca započinje, odnosno završava. Također, *DatasetMIDI* prima i funkciju za izlučи-

vanje oznake klase MIDI datoteke *func_to_get_labels*, ona je implementirana u funkciji *get_label_from_file_path* koja iz putanje do sekvence izluči oznaku klase. Oznaka klase u programskom jeziku *Python* jednostavno izluči iz strukture direktorija pokazane u potpoglavlju Podjela podataka, 3.3.

Nakon predstavljanja bitnih alata za stvaranje rječnika tokena iz MIDI datoteka možemo krenuti učiti duboke modele.

5.2. Povratne neuronske mreže

Povratne neuronske mreže ili RNN, engl. *Recurrent Neural Networks* tip su neuronskih mreža za rad sa sekvencijalnim podacima. Pogodne su za zadatke gdje je redoslijed informacije bitan. Obrada prirodnog jezika, prepoznavanje govora, generiranje teksta ili obrada vremenskih serija neki su od zadataka koje RNN može riješiti.

RNN dakle pretpostavlja međusobnu zavisnost o podacima koji mu dolaze na ulaz, u sebi sadrži mehanizam pamćenja prijašnjih vrijednosti te prilikom računanja izlaza ga uzima u obzir. Taj mehanizam RNN-a možemo nazvati skrivenim stanjem. Matematički, skriveno stanje računa se na idući način:

$$h_t = \sigma(W_h \cdot h_{t-1} + W_x \cdot x_t + b),$$

gdje su:

- h_t: skriveno stanje u trenutku t
- h_{t-1}: skriveno stanje u trenutku *t-1*
- x_t: ulazne vrijednosti u trenutku t
- W_h i W_x: težinske matrice
- b: slobodni parametar (*bias*)
- σ: aktivacijska funkcija, npr. tanh ili ReLU

Unatoč svojim korisnim svojstvima RNN-ovi nailaze nekoliko problema. Jedan od

tih problema su eksplodirajući ili nestajući gradijenti. Tijekom treniranja modela, dovođenjem sve više primjeraka sekvenci učenje gradijenata metodom propagacije unatrag zbog učestalog matričnog množenje neke gradijente svede na veoma mali broj (nestajanje gradijenata) dok neke na vrlo velik broj (rast gradijenata). Time je treniranje otežano. Također, zbog ovih problema mreža ne može *pamtiti* preduge sekvence. RNN-ovi su stoga zamijenjeni ili nadopunjeni naprednijim arhitekturama kao što su LSTM, GRU ili u današnje vrijeme transformeri. U nastavku ćemo malo jasnije objasniti LSTM i GRU.

5.2.1. LSTM i GRU

LSTM, engl. *Long Short-Term Memory* mreža razvijena je kako bi riješila problem eksplodirajućih ili nestajućih gradijenata. Kao što naziv govori LSTM ćelija ima sposobnost pamtiti dugoročne ovisnosti u podacima. Sastoji se od četiri ključnih mehanizama koji kontroliraju informaciju:

1. Stanje ćelije C_t: Vrijednost koja pamti dugoročne informacije o ulaznim sekvencama. Može se zamisliti kao neka vrsta memorije koja se prenosi iz koraka u korak. Ovaj mehanizam opisan je sljedećom jednadžbom:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C_t^{'},$$

   gdje su f_t zaboravna vrata, i_t ulazna vrata, C_{t-1} prethodno stanje ćelije, a $C_t^{'}$ doprinos stanju ćelije.

2. Ulazna vrata i_t i doprinos stanju ćelije $C_t^{'}$: Ulazna vrata odlučuju koje informacije će biti dodane u ćelijsko stanje. Ove vrijednosti računaju se sljedećim jednadžbama:

$$i_t = \sigma(W_{ihh} \cdot h_{i-1} + W_{ixh} \cdot x_t + b_f)$$

$$C_t^{'} = \tanh(W_{chh} \cdot h_{t-1} + W_{cxh} \cdot x_t + b_{ch}),$$

   gdje su W_* težinske matrice, b_* slobodni parametri, h_{t-1} skriveno stanje iz prethodnog koraka, a x_t trenutni ulaz.

3. Zaboravna vrata f_t: Ova vrata odlučuju koje informacije treba zaboraviti iz ćelij-

skog stanja. Računaju se sljedećom jednadžbom:

$$f_t = \sigma(W_{fhh} \cdot h_{t-1} + W_{fxh} \cdot x_t + b_{fn}).$$

4. Izlazna vrata o_t: Izlazna vrata odlučuju koje informacije će biti generirane na izlaz u trenutnom vremenskom koraku sekvence. Računa se sljedećim jednadžbama:

$$o_t = \sigma(W_{ohh} \cdot ht - 1 + W_{oxh} \cdot x_t + b_{oh})$$

Sada kada smo izlistali mehanizme LSTM ćelije kojima se rješava problem eksplodirajućih ili nestajućih gradijenata vidimo koliko je LSTM ćelija kompleksnija od klasične RNN ćelije. Također, ove jednadžbe nećemo objašnjavati kako i zašto rješavaju navedeni problem, one su navedene samo da nam predoče koliko LSTM ćelija ima više parametara W_* za naučiti i koliko je složena. LSTM ćelija može biti zamijenjena svojom jednostavnijom inačicom s dvojim vratima - GRU.

GRU, engl. *Gated Recurrent Unit* kao što i LSTM, varijanta je RNN ćelije te također rješava problem dugoročne ovisnosti. Jednostavnijeg je dizajna od LSTM-a zadržavajući pritom sličnu učinkovitost. Sastoji se od sljedećih vrata:

1. Vrata ažuriranja u_t: Vrata ažuriranja određuju koliko prethodnog skrivenog stanja h_{t-1} treba prenijeti u novo skriveno stanje h_t. Ova vrata omogućavaju GRU da dugoročno zadrži informacije:

$$u_t = \sigma(W_{uhh} \cdot h_{t-1} + W_{uxh} \cdot x_t + b_{uh}).$$

2. Vrata resetiranja r_t: Odlučuju koliko prethodnog skrivenog sloja ćelija treba zaboraviti u računanju nove vrijednosti skrivenog sloja:

$$r_t = \sigma(W_{rhh} \cdot h_{t-1} + W_{rxh} \cdot x_t + b_{rh}).$$

3. Povratno stanje h_t sada se računa na sljedeći način:

$$h_t = u_t \cdot h_{t-1} + (1 - u_t)h_t^{'}$$

$$h_t^{'} = \sigma(W_{hh} \cdot (r_t \cdot h_{t-1}) + W_{xh} \cdot x_t + b_h).$$

5.2.2. Pozornost

Mehanizam pozornosti, engl. *attention*, može se koristiti u zadacima klasifikacije povratnim neuronskim mrežama (LSTM, GRU) kako bi se poboljšala sposobnost modela da prepozna ključne dijelove ulazne sekvence bitne za klasifikaciju. Pozornost dakle omogućava modelu da se usredotoči na određene dijelove sekvence. U kontekstu klasifikacije podataka korištenjem RNN-ova mehanizam pozornosti funkcionira na sljedeći način:

1. Generiranje skrivenih stanja: Kao što znamo RNN (LSTM ili GRU) ćelije iz skupa ulaznih podataka X = [x_1, x_2, x_3, ..., x_t] generiraju skup skrivenih stanja H = [h_1, h_2, h_3, ..., h_t] do trenutka t.

2. Računanje težina pozornosti: Za svaki t iz H izračuna se vektor pozornosti e ili *score* koji mjeri važnost pojedinog skrivenog stanja h_t:

$$e_t = v^T \cdot \tanh(W_h h_t + b_h),$$

   gdje su W_h i b_h parametri koji se uče, v vektor za ocjenu važnosti.

3. Normalizacija *score*-ova: Normalizacijom *score*-ova e_t korištenjem funkcije *softmax* dobivamo težine pozornosti α_t:

$$\alpha_t = softmax(e_t).$$

4. Računanje kontekstnog vektora c:

$$c = \sum_t (\alpha_t \cdot h_t).$$

5. Klasifikacija: Tako izračunat kontekstni vektor c sada se koristi kao ulaz u završni klasifikacijski sloj modela. Na primjer izlaz modela y mogao bi računati ovako:

$$y = softmax(W_c \cdot c + b_c).$$

5.3. Učenje modela

Nakon što smo u poglavlju *Tokenizacija* 5.1. pokazali kako se MIDI datoteke mogu korištenjem tehnike obrade prirodnog teksta pretvoriti u smislene tokene te u poglavlju *Povratne neuronske mreže* 5.2. objasnili kako funkcioniraju LSTM i GRU modeli s ugrađenom pozornošću možemo pripremiti podatke i naučiti duboke modele.

U nastavku je pokazan isječak koda u kojem se određuju hiperparametri treniranja modela, učitavanje sekvenci, odabir modela i učenje na skupu za učenje. Također, nakon svake epohe model će se evaluirati na neviđenom skupu za evaluaciju. Nakon treniranja odabran je model koji najbolje generalizira (ima najmanji gubitak na skupu za evaluaciju) te je njime testiran skup za testiranje i pokazane su performanse tog modela.

Hiperparametri modela:

```
epoches = 20
batch_size = 64

learning_rate = 0.0005
max_norm = 1.5

vocab_size = vocab_size
embedding_dim = 128
model_name = "lstm_custom"
layer_type = "lstm"
layer_dims = [512, 256, 128]
dropout = 0.3
num_classes = 10

model = CustomRNN(
```

```
    vocab_size,
    embedding_dim,
    layer_types,
    layer_dims,
    dropout,
    num_classes,
    attention=True
)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

U odsječku koda iznad pokazano je kako odabrati hiperparametre, inicijalizirati model i optimizator modela. Prvo je određen broj epohi u varijabli *epoches* i taj broj će biti 20 u svim modelima koje ćemo učiti, također, varijable *batch_size* uvijek će biti 64. *Batch_size*-om određujemo koliko ćemo sekvenci u jednom potezu provući kroz model bez evaluiranja, dakle u svakom koraku optimizacije modela kroz njega je provučeno *batch_size* sekvenci veličine *max_seq_len* (potpoglavlje *MidiTok*) 5.1.2. *Learning_rate* i *max_norm* parametri su optimizatora i njima je varijablom *learning_rate* određena brzina učenja tj. varijablom *max_norm* najveća vrijednost norme vektora gradijenta čime dodatno sprječavamo eksplodirajuće gradijente. Nakon toga definirani su parametri svojstveni za model koji treniramo, a to su:

1. *Vocab_size*: Veličina rječnika tokena, ova veličina je objašnjena u poglavlju *MidiTok* 5.1.2.

2. *Embedding_dim*: dimenzije vektorske reprezentacije svakog tokena. Dakle, prilikom učenja svaki token dobiva svoj vektor nasumičnih vrijednosti koji se kroz korake učenja također uči. Nakon nekog vremena učenja tokeni koji se nalaze u međusobnoj blizini imat će bliže vrijednosti u svojoj vektorskoj reprezentaciji. Ove vrijednosti su u našem slučaju nasumične, no one se također mogu pametnije namijesiti ili naučiti, ali to nije tema ovog rada.

3. *Model_name*: Ime modela koji treniramo, ovaj parametar poslužit će nam za nazivanje i kategorizaciju modela kako bismo ih lakše usporedili.

4. *Layers_type*: Tip RNN ćelije koje će model koristiti. Ova varijabla poprima vrijednosti *LSTM* ili *GRU*.

5. *Layers_dims*: Dimenzije pojedine RNN ćelije.

6. *Dropout*: Vrijednost kojom se regularizira model. Određuje koliki postotka težini će se isključiti tijekom učenja.

7. *Num_classes*: Broj klasi koje klasificiramo. Određen je u poglavlju *Odabir skladatelja* 3.2. i iznosi 10.

Funkcijama *CustomRNN*, *nn.CrossEntropzLoss* i *optim.Adam* kreirani su mehanizmi objekti kojima omogućavamo duboko učenje. *Model*-om je stvorena sama mreža, *criterion* računa gubitak mreže, a *optimizer* strategiju optimiranja parametara.

Kreiranje objekata za učitavanje podataka:

```
collator = DataCollator(
    tokenizer.pad_token_id,
    copy_inputs_as_labels=True
)
train_loader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True,
    collate_fn=collator
)
valid_loader = DataLoader(
    valid_dataset,
    batch_size=batch_size,
    shuffle=True,
    collate_fn=collator
)
```

Collator nam omogućava da između svake sekvence stavi *pad_token* kako bi se najavila pojava nove sekvence. *Train_loader* i *valid_loader* učitavaju skupove podataka u obliku tokena.

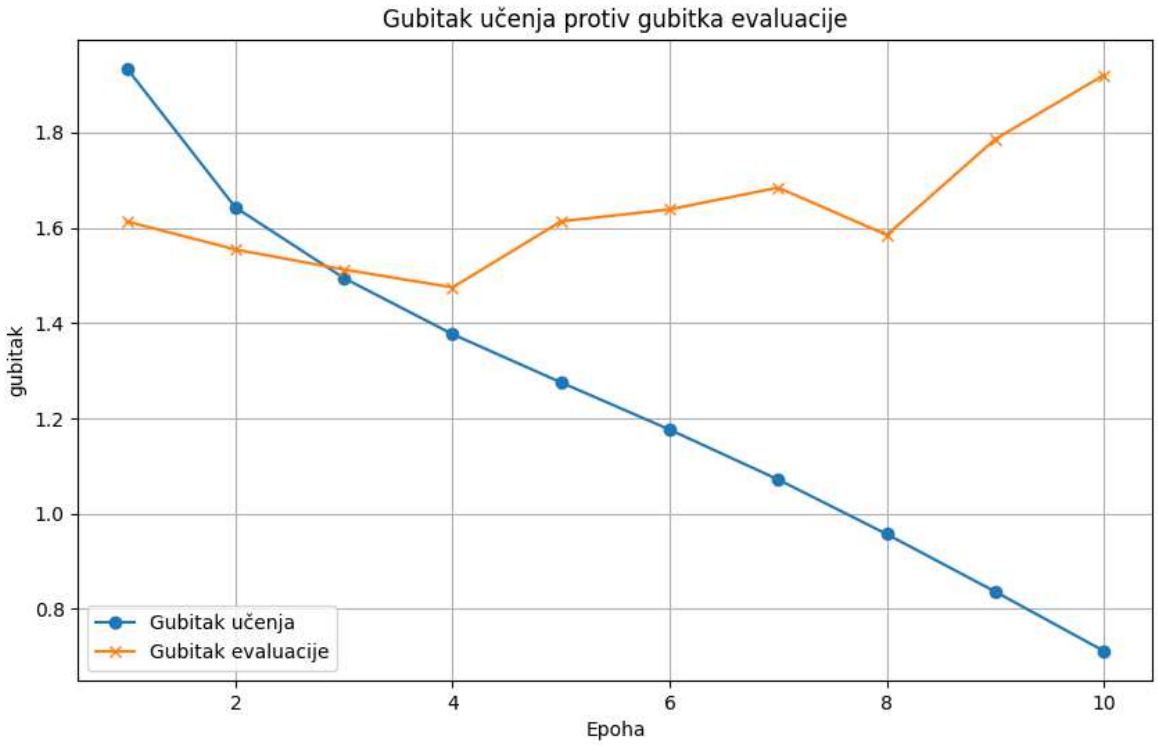
Učenje modela:

```
for epoch in range(epoches):
    print(f"Epoch: {epoch + 1}")
    train(
        model,
        train_loader,
        optimizer,
        criterion,
        max_norm
    )
    results = evaluate(
        model,
        valid_loader,
        criterion,
        num_classes
    )

    model_path = Path(
        f"models/{model_name}/{tokenizer_name}/{epoch + 1}"
    )
    os.makedirs(model_path, exist_ok=True)
    save_checkpoint(
        epoch + 1,
        model,
        optimizer,
        results,
        model_path
    )
```

Učenje modela odvija se u epohama, u svakoj epohi modelu je pokazan cijeli skup podataka. Dakle, za učenje u 10 epohi kroz model je skup podataka prošao 10 puta. Funkcijom *train* uče se parametri modela na skupu za učenje. Implementacija ove funkcije nije pokazana. Funkcijom *evaluate* na skupu za evaluaciju provjeravaju se performanse modela. Računa se točnost i F1-vrijednost te gubitak modela. Nakon svake epohe model

je spremljen na putanju pohranjenu u varijablu *model_path* kako bi se rezultati mogli usporediti.

Na slici 5.1. prikazana je promjena gubitna na skupu za učenje i skupu za validaciju modela postupcima opisanim u prethodnim koracima. Na slici možemo jasno vidjeti trenutak kada je model na skupu za evaluaciju imao najmanji gubitak i možemo reći da u tom trenu model najbolje generalizira, a taj trenutak je nakon 4. epohe učenja.



Slika 5.1. Kretanje gubitka na skupu za učenje i skupu za evaluaciju po epohama

Sada taj model možemo učitati, evaluirati ga na skupu za testiranje i pokazati njegove performanse. Na ovaj način naučit ćemo nekoliko modela i prikazati njihove performanse u poglavlju koje slijedi.

5.4. Rezultati

Rezultate treniranja ćemo prikazati u 3 skupine modela po veličini i 2 po tipu ćelije, dakle usporedit ćemo 6 modela. Za svaki tip ćelije odabrat ćemo tri veličine modela: *small*, *medium* i *large*. Kao mjere uspješnosti modela odabrat ćemo *accuracy* i *F1-macro* te matrice zabune. Veličine slojeva pojedinog modela vidimo u nastavku:

```
{
    "model": {
        "type": "small",
        "embeddings": 32,
        "layers": [128]
    }
}
{
    "model": {
        "type": "medium",
        "embeddings": 64,
        "layers": [256, 128]
    }
}
{
    "model": {
        "type": "large",
        "embeddings": 128,
        "layers": [256, 128, 128]
    }
}
```

U tablicama *Accuracy* 5.2. i *F1-score* 5.3. prikazani su rezultati treniranih dubokih modela. Možemo odmah uočiti znatnu razliku u mjerenjima plitkih modela pokazanih u poglavlju *Stroj potpornih vektora* 4. Možemo reći da su *accuracy* i *F1-score* za otprilike 10% viši što nam ukazuje kako je ova metoda efikasnija. Povećanjem slojeva mreže ne možemo reći kako se mjerenja povećavaju, no to je možda zato što smo za pojedini model povećali i veličinu rječnika ovim slijedom: 5000, 7500 i 15000. Kada bismo uzeli jednak rječnik za svaku veličinu možda bismo dobili drugačije vrijednosti. Razlika između tipa ćelije također nije velika i možemo opaziti kako GRU funkcionira malo bolje od LSTM-a.

Accuracy			
Type.	Small	Medium	Large
GRU	0.49	0.52	0.51
LSTM	0.48	0.50	0.50

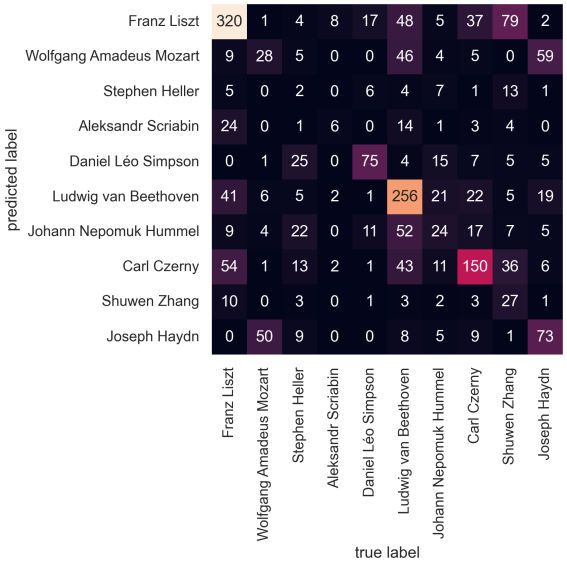
Tablica 5.2. *Accuracy* ovisno o broju parametara i tipu ćelije

F1-score			
Type.	Small	Medium	Large
GRU	0.37	0.45	0.44
LSTM	0.35	0.40	0.42

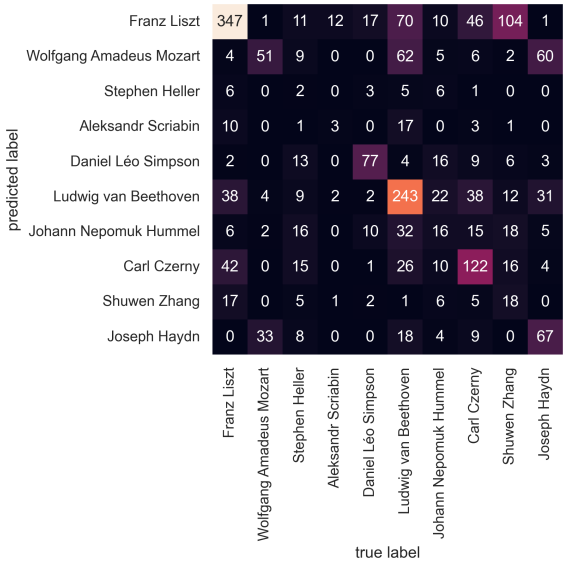
Tablica 5.3. *F1-score* ovisno o broju parametara i tipu ćelije

5.4.1. Matrice zabune

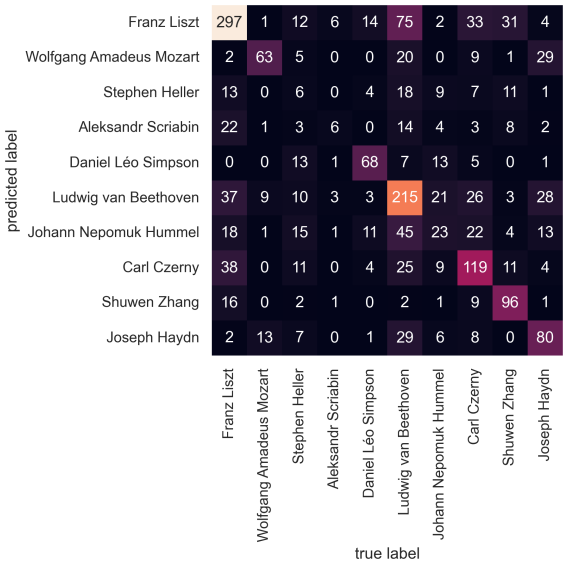
U matricama zabune modela GRU i LSTM uočavamo kako duboki modeli bolje generaliziraju od plitkih. Matrice potvrđuju i manju razliku u vrijednosti iz tablici *Accuracy* i *F1-macro*. Porastom parametara modela ne uočavamo znatnu razliku, no to se može objasniti kako smo za svaki model povećali i vokabular tokena. Uočavamo kako je model sposobniji uz *Franz Liszta* prepoznati i ostale skladatelje kao što su *Ludwig van Beethoven*, *Carl Czerny*, *Shuwen Zhang* i *Joseph Hayden* što je za ovu vrstu problema impresivno.



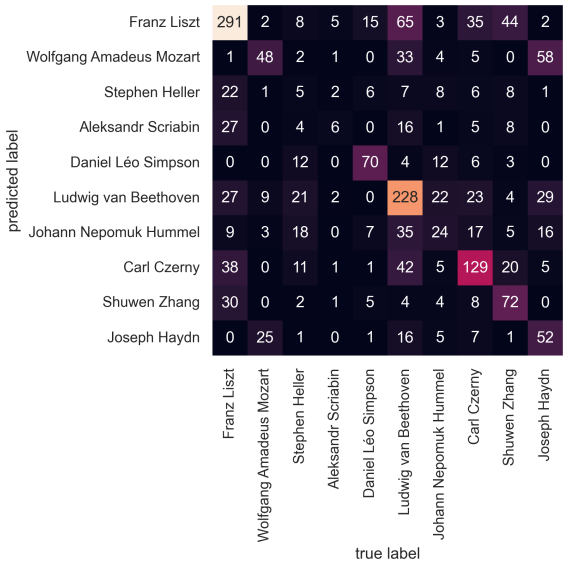
Slika 5.2. Matrica zabune GRU *small* modela



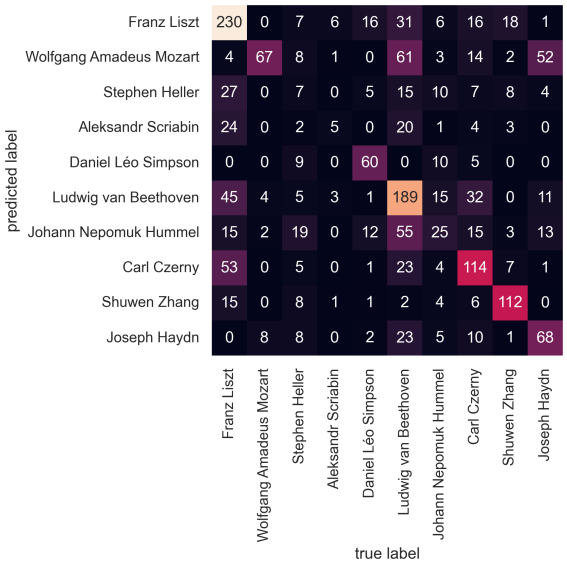
Slika 5.3. Matrica zabune LSTM *small* modela



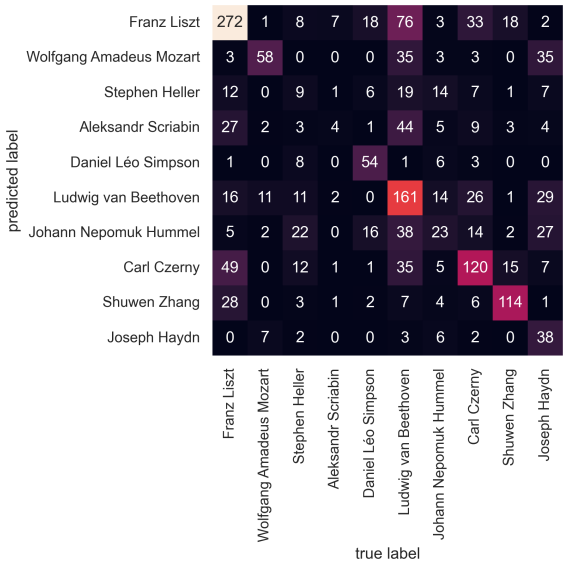
Slika 5.4. Matrica zabune GRU *medium* modela



Slika 5.5. Matrica zabune LSTM *medium* modela



Slika 5.6. Matrica zabune GRU *large* modela



Slika 5.7. Matrica zabune LSTM *large* modela

6. Zaključak

Ovaj rad predstavlja problem klasifikacije MIDI datoteke računalima. Predstavljen je MIDI protokol i objašnjena je njegova namjena: od MIDI portova u fizičkom sloju pa sve do MIDI poruka i MIDI datoteka koje se koriste u računalnim programima. Objašnjene su neke MIDI poruke i što one znače u sklopu MIDI protokola. Za zadatak klasifikacije odabran je skup podataka GiantMIDI-Piano te je pokazano kako su odabrane klase za problem klasifikacije. Nadalje, pokazan je način pristupanja podatcima u MIDI datotekama korištenjem programskog jezika *Python* i ekstrakcija bitnih značajki za klasifikaciju melodije. Zatim smo naučili plitke klasifikatore SVM i RF *naivnim* pokušajem pronalaženja cjelini u podatcima korištenjem PCA algoritma. Nakon toga iskoristili smo sofisticiranije metode obrade prirodnog jezika i MIDI datoteke smo pretvorili u tokene koje smo zatim pohranili u duboke modele te pokazali kako su uspješniji od plitkih modela.

Obrada MIDI datoteka metodama prirodnog jezika pokazala se obećavajućim putem u istraživanju klasifikacije ili čak generiranja MIDI datoteka. U daljnjim istraživanjima za klasifikaciju mogli bi se i primijeniti modeli transformatora koji su se u proteklih par godina pokazali superiornijim pristupom od LSTM i GRU ćelija.

Literatura

[1] Q. Kong, B. Li, J. Chen, i Y. Wang, "Giantmidi-piano: A large-scale midi dataset for classical piano music", *arXiv preprint arXiv:2010.07061*, 2020.

[2] S. Deepaisarn, S. Chokphantavee, S. Chokphantavee, P. Prathipasen, S. Buaruk, i V. Sornlertlamvanich, "Nlp-based music processing for composer classification", *Scientific Reports*, sv. 13, br. 1, str. 13228, 2023.

[3] J. Rothstein, *MIDI: A comprehensive introduction*. AR Editions, Inc., 1995., sv. 7.

[4] C. Raffel i D. P. Ellis, "Data with pretty_midi".

[5] J. VanderPlas, *Python data science handbook: Essential tools for working with data*. " O'Reilly Media, Inc.", 2016.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python", *the Journal of machine Learning research*, sv. 12, str. 2825–2830, 2011.

[7] Y.-S. Huang i Y.-H. Yang, "Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions", u *Proceedings of the 28th ACM international conference on multimedia*, 2020., str. 1180–1188.

[8] N. Fradet, J.-P. Briot, F. Chhel, A. E. F. Seghrouchni, i N. Gutowski, "Miditok: A python package for midi file tokenization", *arXiv preprint arXiv:2310.17202*, 2023.

Sažetak

Model dubokog učenja za klasifikaciju MIDI datoteka

Bartol Rod

Klasifikacija glazbe, točnije MIDI datoteka, zanimljiv je pristup istraživanja metodi obrade prirodnog jezika. U ovom radu pokazuje se mogućnost obrade i klasifikacije MIDI datoteka u tu svrhu. Pokazan je način pretvorbe MIDI poruka u tokene te njihovo korištenje u dubokim modelima. Time je jasno postavljena osnova za daljnja istraživanja ovog pristupa.

Ključne riječi: MIDI datoteke, duboko učenje, klasifikacija

Abstract

Deep Learning Model for MIDI Classification

Bartol Rod

Music classification, specifically MIDI file classification, is an interesting approach to exploring natural language processing methods. This paper demonstrates the possibility of processing and classifying MIDI files for this purpose. The method of converting MIDI messages into tokens and their use in deep models is shown. This clearly establishes a foundation for further research in this approach.

Keywords: MIDI files, deep learning, classification