COMP 4948 Assignment 1 Report

# Report Introduction

This project explores a dataset of songs in Spotify:

<https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify>

The music streaming platform, Spotify, analyses a song’s different musical properties. These properties help the platform manage their music inventory and customize their experience for different users. In this report we will make a predictive model that predicts a song’s genre using its properties.

The dataset contains around 42,000 rows of songs from the music streaming platform, Spotify. This dataset has 22 columns with one being our target variable ‘genre’. Several of the columns will be dropped because they cannot be relevant columns since they are metadata or unused columns such as: type, id, uri, track\_href, analysis\_url, song\_name, title, and Unnamed: 0. This leaves us with 13 feature columns where 10 are continuous numerical columns, while the other 3 are discrete or binary numerical columns.

The dataset did not need imputing, and features were selected using:

* Recursive Feature Elimination (RFE)
* Forward Feature Selection (FFS), and
* Feature Importance

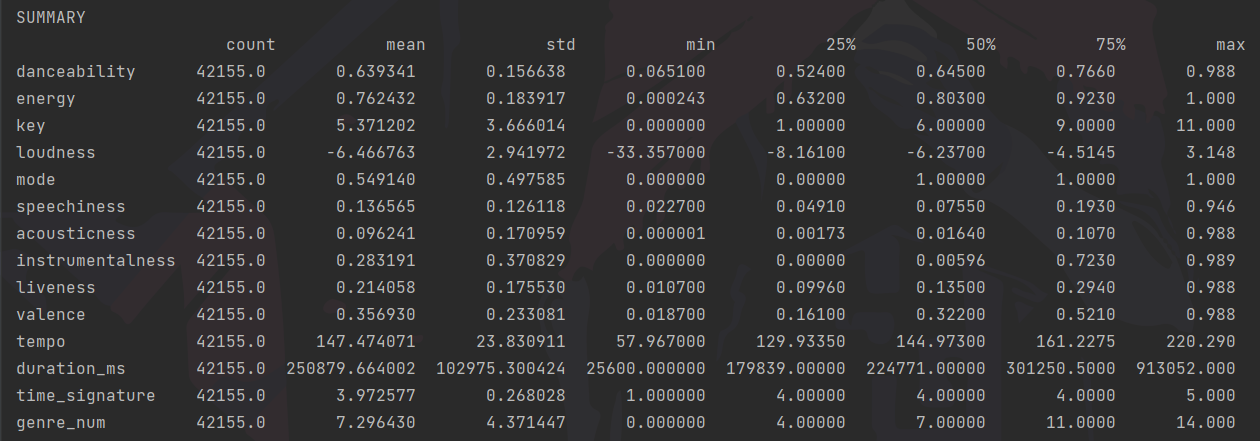
Several models were made and tested using crossfold validation against these metrics:

* Accuracy
* Recall
* Precision
* F1 Score, and
* Area Under Curve score

# Exploratory Data Analysis

## Data Dictionary

The dataset contains 13 valid feature columns, and 1 target column (genre). No imputing was needed since all important columns do not have missing values. The genre column has been converted from categorical strings into discrete integers ranging from 0 to 14.



## Target Variable

The dataset contains 13 valid feature columns, and 1 target column (genre). No imputing was needed since all important columns do not have missing values.

## Feature Segmentation

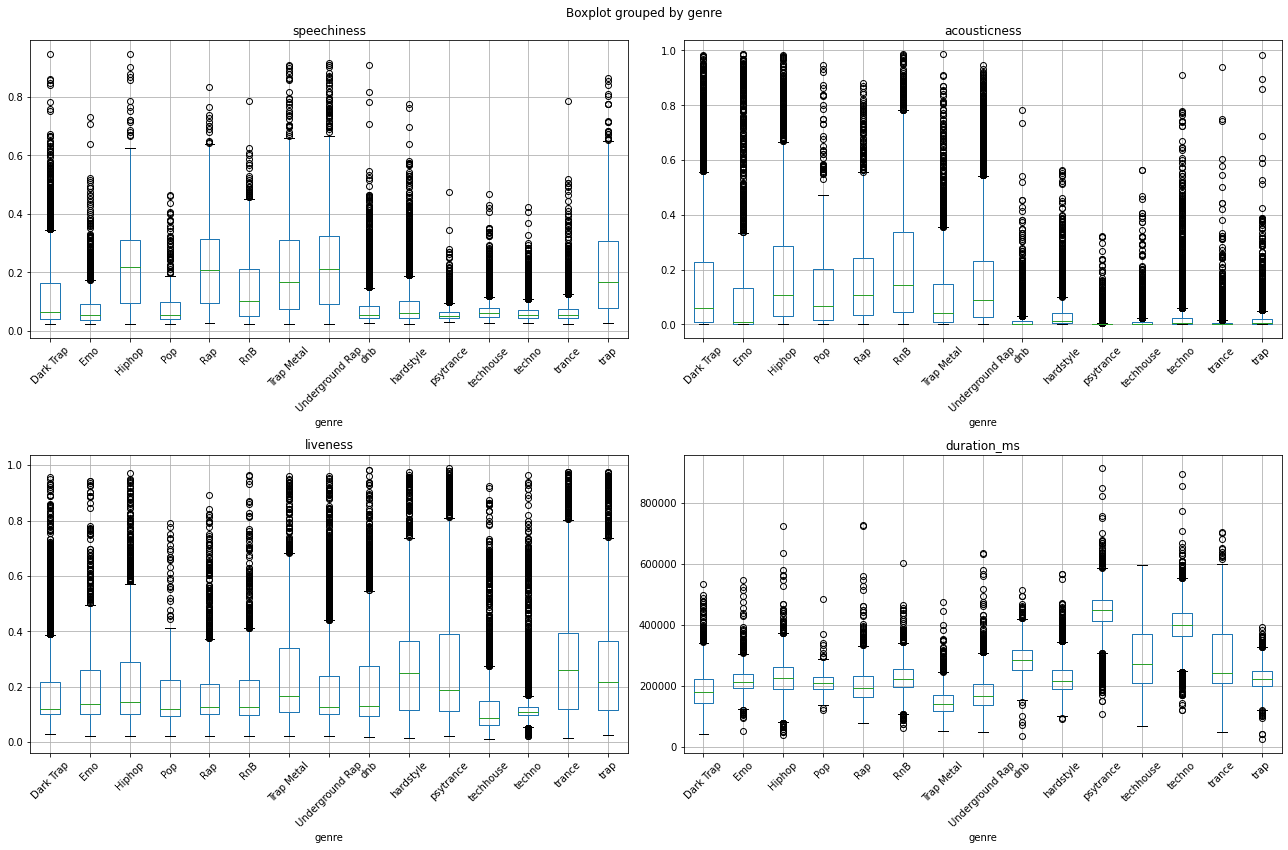
The 13 feature columns went through automated feature selection, and only 10 feature columns were significant. We also used scaling using StandardScaler which improved all created models. Binning was tried but never improved the model.

We can put these columns into three groups. All genres in the dataset have higher values for danceability, energy, and loudness as shown in the boxplot below.

Graphical user interface, application

Description automatically generated

The next group shows lower values for all genres. They consist of speechiness, acousticness, liveness, and duration\_ms. Their boxplots are shown below.



The last group consists of wildcard features where they have different values for different genres. Their boxplots are shown below.

Graphical user interface, application, Excel

Description automatically generated

## Target Variable

The target variable is the genre column which we have converted into numerical values put into the genre\_num column. The table below shows the mapping of genre to genre\_num.

|  |  |  |  |
| --- | --- | --- | --- |
| Genre\_num | Genre | Genre\_num | Genre |
| 0 | Dark Trap | **8** | Dnb |
| 1 | Emo | **9** | Hardstyle |
| 2 | Hiphop | **10** | Psytrance |
| 3 | Pop | **11** | Techhouse |
| 4 | Rap | **12** | Techno |
| 5 | RnB | **13** | Trance |
| 6 | Trap Metal | **14** | Trap |
| 7 | Underground Rap |  |  |

The dataset is not imbalanced and has a clear bias towards Underground Rap (7), while having significantly less data for Pop (3). We have used SMOTE to address this imbalance. The graph below shows side-by-side bar plots for before and after SMOTE.

A picture containing icon

Description automatically generated

## Feature Selection

Using automated feature selection methods we can identify the most important features in the dataset. Feature selection suggests that the other 3 columns: key, mode, and time\_signature were not significant predictors. The table below shows the selected features between RFE, FFS, and Feature Importance using a RandomForest model.

|  |  |
| --- | --- |
| Selector | Selected Features |
| RFE | ['danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'duration\_ms'] |
| FFS | ['danceability', 'energy', 'loudness', 'instrumentalness', 'duration\_ms'] |
| Feature Importance | ['danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'valence', 'liveness','tempo', 'duration\_ms'] |

Feature importance also ranks the features according to the most significant. The order is the same as in the table. All the features scored above 0.05 in the feature importance. A decision was made to not include features below this threshold.

# Model Development

Five different models were considered to predict the genre of songs in the dataset, these were:

* LogisticRegression
* RandomForestClassifier
* BaggingClassifier
* EnsembleVoteClassifier
* Stacked model

For RandomForest, 200 estimators were used after performing a grid search. Trial and error were used to determine the models incorporated into the BaggingClassifier, EnsembleVoteClassifier, and the Stacked model. BaggingClassifier bags the KNeighborsClassifier model. EnsembleVoteClassifier uses the models: RandomForestClassifier, XGBoostClassifier, KNeighborsClassifier, ExtraTreesClassifier. The Stacked model uses XGBoostClassifier, ExtraTreesClassifier, RandomForest, KNeighborsClassifier, and uses LogisticRegression as the meta-classifier.

We used 5-fold cross fold validation to test all models mentioned (and some not mentioned). The metrics used to measure each fold are accuracy, precision, recall, f1, and area under curve score. We then take the average the results for each fold. The table below shows the means and the standard deviations from each model.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Mean accuracy | Mean precision | Mean recall | Mean f1 score | Mean area under the curve |
| LogisticRegression | 0.5947801418439717 | 0.5871027565633732 | 0.5948294904195863 | 0.589103436156387 | 0.933955307495944 |
| Randomforest | 0.7787574468085106 | 0.7715400107780207 | 0.7787935014376305 | 0.7741317378165344 | 0.9697596053840867 |
| baggingclassifier | 0.7032510638297873 | 0.6901503513927583 | 0.7032582829706343 | 0.6926823176643204 | 0.9427206999391347 |
| ensemblevote | 0.774104964539007 | 0.7666160359050085 | 0.7740925299737027 | 0.7679890098815448 | 0.97515560974634 |
| stacked | 0.37251475261007716 | 0.295002277443362 | 0.37357383651805975 | 0.3008633115088687 | 0.8669027308954937 |

Their respective standard deviations are also showed in the following table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | accuracy STD deviation | precision STD deviation | recall STD deviation | f1 score STD deviation | area under the curve STD deviation |
| LogisticRegression | 0.004111028864167207 | 0.0032905143671257426 | 0.002953175983026523 | 0.0031838860490832456 | 0.0009847470756601763 |
| Randomforest | 0.0033404420953949024 | 0.004758833113570287 | 0.004842098959404523 | 0.004619149148904746 | 0.0005696385587728012 |
| baggingclassifier | 0.0031563136375975464 | 0.002849450550563026 | 0.002071429878452561 | 0.002434144033021757 | 0.0008945662045182712 |
| ensemblevote | 0.003490998866349721 | 0.002355268279439228 | 0.00207201678565309 | 0.0022812073670014244 | 0.0003491899309456657 |
| stacked | 0.04233766252249676 | 0.04245207780099364 | 0.03924328077967014 | 0.046058939559975066 | 0.0008642937988886642 |

# Model Selection

The best model between our five developed models is the RandomForestClassifier. It has the best values on all metrics. It is competitive with BaggingClassifier and the EnsembleVote, but it is a much simpler model to make.

# Appendix

from imblearn.over\_sampling import SMOTE  
from sklearn.preprocessing import StandardScaler  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.feature\_selection import RFE, f\_classif, SelectKBest  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, \  
 ConfusionMatrixDisplay, roc\_auc\_score  
from sklearn.model\_selection import KFold, train\_test\_split  
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, ExtraTreesClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear\_model import LogisticRegression  
from xgboost import XGBClassifier  
from mlxtend.classifier import EnsembleVoteClassifier  
import warnings  
  
warnings.filterwarnings('ignore')  
pd.set\_option('display.max\_rows', 500)  
pd.set\_option('display.max\_columns', 500)  
pd.set\_option('display.width', 1000)  
  
FILEPATH = './genres\_v2.csv'  
df = pd.read\_csv(FILEPATH)  
  
# Join columns title and song\_name  
titles = df[["song\_name", "title"]]  
titles = titles["song\_name"].combine\_first(titles['title'])  
df['song\_name'] = titles  
  
# Create a new col with a num counterpart of genre  
num\_genres = range(15)  
GENRES = list(df.copy().groupby('genre').count().index) # number to genre  
genre\_to\_num = dict(zip(GENRES, num\_genres))  
tmp = df['genre'].copy(deep=True).replace(genre\_to\_num)  
df['genre\_num'] = tmp  
  
# Fill n/a values of song\_name into unnamed  
df["song\_name"] = df["song\_name"].fillna("unnamed")  
  
# Remove irrelevant columns from the dataset (remove metadata)  
to\_remove = ["type", "id", "uri", "track\_href", "analysis\_url", "title", "Unnamed: 0"]  
for rm in to\_remove:  
 del df[rm]  
  
print(df)  
print("\nDtypes\n", df.dtypes)  
print("\nSUMMARY\n", df.describe(include='number').T, end='\n\n')  
print("genre\_num to genre mapping")  
for i, genre in enumerate(GENRES):  
 print(i, genre)  
print()  
  
# exit()  
# \*\* SPLIT INTO X AND Y \*\*  
# Split data into X and y  
X = df.copy()  
del X['genre\_num']  
del X['genre']  
  
y = df['genre\_num']  
  
X = X.select\_dtypes(include='number')  
print(f"Features ({len(X.columns)}):", list(X.columns))  
print("Target column:", y.name)  
  
# \*\* SMOTE \*\*  
X, y = SMOTE().fit\_resample(X, y)  
  
# \*\* SCALING \*\*  
X\_cpy = X.copy()  
sc\_features = X\_cpy[  
 ['danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence',  
 'tempo', 'duration\_ms']]  
unscaled\_features = X\_cpy[['key', 'mode', 'time\_signature']]  
sc\_x = StandardScaler().fit(sc\_features.values)  
scaled = sc\_x.transform(sc\_features.values)  
sc\_x\_features = pd.DataFrame(scaled, index=X.index, columns=sc\_features.columns)  
X = pd.concat([sc\_x\_features, unscaled\_features], axis='columns')  
print("\nUnscaled feature columns:", list(unscaled\_features.columns.values))  
print("Scaled feature columns:", list(sc\_features.columns.values))  
print("\nScaled X:", X)  
print("\nSUMMARY of transformed features:\n", X.describe(include='number').T, end='\n\n')  
  
  
## \*\* FEATURE SELECT \*\*  
def select\_features(selector, selector\_name, X, y):  
 selector.fit(X, y)  
 selected = list(selector.get\_feature\_names\_out())  
 selected\_features = {}  
 selected\_features[selector\_name] = selected\_features  
 return selected  
  
  
rfe = RFE(RandomForestClassifier(n\_estimators=100), step=5, n\_features\_to\_select=10)  
print("RFE:", select\_features(rfe, 'rfe', X, y))  
  
ffs = SelectKBest(score\_func=f\_classif, k=5)  
print("FFS", select\_features(ffs, 'ffs', X, y))  
  
def showFeatureImportances(clf\_model, X, y):  
 print(f"\*\* {clf\_model.\_\_class\_\_.\_\_name\_\_} feature importance \*\*")  
 clf\_model.fit(X, y)  
 importances = list(clf\_model.feature\_importances\_)  
  
 dfImportance = pd.DataFrame()  
 selected = []  
 for i in range(0, len(importances)):  
 dfImportance = dfImportance.append({"importance": importances[i], "feature": X.columns[i]}, ignore\_index=True)  
 if importances[i] > 0.05:  
 selected.append(X.columns[i])  
 dfImportance = dfImportance.sort\_values(by=['importance'], ascending=False)  
 print(dfImportance)  
 print("SELECTED by feature importance > 0.05:", selected)  
  
  
showFeatureImportances(RandomForestClassifier(n\_estimators=200), X, y)  
  
# Declare the best features  
best\_features = ['danceability', 'energy', 'loudness', 'speechiness', 'acousticness', 'instrumentalness', 'valence',  
 'liveness', 'tempo', 'duration\_ms']  
X = X[best\_features]  
print("Selected Features:", list(X.columns))  
  
  
## \*\* MODEL TRAINING (Xfold validation) \*\*  
def evaluate(model, X\_test, y\_test):  
 y\_pred = model.predict(X\_test)  
 y\_pred\_probs = model.predict\_proba(X\_test)  
  
 evals = {}  
  
 evals['cm'] = confusion\_matrix(y\_test, y\_pred)  
 evals["accuracy"] = accuracy\_score(y\_test, y\_pred)  
 evals["precision"] = precision\_score(y\_test, y\_pred, average='macro')  
 evals["recall"] = recall\_score(y\_test, y\_pred, average='macro')  
 evals["f1"] = f1\_score(y\_test, y\_pred, average='macro')  
 evals["AUC"] = roc\_auc\_score(y\_test, y\_pred\_probs, multi\_class='ovr')  
  
 return evals, y\_pred  
  
  
k = 5  
kfold = KFold(k, shuffle=True)  
results = pd.DataFrame(columns=('cm', 'accuracy', 'precision', 'recall', 'f1', 'AUC'))  
  
clfs = [  
 LogisticRegression(),  
 RandomForestClassifier(n\_estimators=200),  
 BaggingClassifier(KNeighborsClassifier(), n\_estimators=10),  
 EnsembleVoteClassifier(clfs=[XGBClassifier(), RandomForestClassifier(n\_estimators=100), KNeighborsClassifier()],  
 voting='hard')  
]  
for clf in clfs:  
 i = 0  
 print(f"\*\* Training {clf.\_\_class\_\_.\_\_name\_\_} \*\*")  
 for train, test in kfold.split(X, y):  
 print(f"\nTrain size: {len(train)}", f"Test size: {len(test)}")  
 train\_x, test\_x = X.iloc[train], X.iloc[test]  
 train\_y, test\_y = y.iloc[train], y.iloc[test]  
  
 # Create model  
 model = clf.fit(train\_x, train\_y)  
 print(f"Model {i} fitting done")  
  
 # Evaluate metrics  
 evals, preds = evaluate(model, test\_x, test\_y)  
  
 results.loc[f"Model {i}"] = evals  
 print(f"Model {i} eval done")  
 i += 1  
  
 # Show metrics  
 print(results)  
 print()  
 averages = {}  
 for col in results.columns:  
 if col not in ['cm']:  
 key = f"Average {col}"  
 averages[key] = results[col].mean()  
 print(key + ":", averages[key])  
 print(f"Std dev {col}:", results[col].std())  
  
  
# \*\* Stacked Model \*\*  
print("\*\* STACKED MODEL \*\*")  
def fitBaseModels(X\_train, y\_train, X\_test, models):  
 dfPredictions = pd.DataFrame()  
 # Fit base model and store its predictions in dataframe.  
 for i in range(0, len(models)):  
 models[i].fit(X\_train, y\_train)  
 predictions = models[i].predict(X\_test)  
 colName = str(i)  
 # Add base model predictions to column of data frame.  
 dfPredictions[colName] = predictions  
 return dfPredictions, models  
  
  
def fitStackedModel(X, y):  
 model = LogisticRegression(solver='liblinear')  
 model.fit(X, y)  
 return model  
  
  
def evaluate\_print(y\_true, y\_pred):  
 evals = {}  
  
 # evals['cm'] = confusion\_matrix(y\_true, y\_pred)  
 evals["accuracy"] = accuracy\_score(y\_true, y\_pred)  
 evals["precision"] = precision\_score(y\_true, y\_pred, average='macro')  
 evals["recall"] = recall\_score(y\_true, y\_pred, average='macro')  
 evals["f1"] = f1\_score(y\_true, y\_pred, average='macro')  
  
 for x, y in evals.items():  
 print(f"\t{x}", y)  
 return evals  
  
  
# Split data into train, test and validation sets.  
k = 5  
kfold = KFold(k, shuffle=True)  
j = 0  
results = pd.DataFrame(columns=('cm', 'accuracy', 'precision', 'recall', 'f1', 'AUC'))  
for train, test in kfold.split(X, y):  
 X\_train, X\_temp = X.iloc[train], X.iloc[test]  
 y\_train, y\_temp = y.iloc[train], y.iloc[test]  
  
 X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_temp, y\_temp, test\_size=0.50)  
  
 # Fit base and stacked models.  
 model\_stack = [  
 XGBClassifier(),  
 ExtraTreesClassifier(n\_estimators=100),  
 RandomForestClassifier(n\_estimators=100),  
 KNeighborsClassifier()  
 ]  
 dfPredictions, models = fitBaseModels(X\_train, y\_train, X\_val, model\_stack)  
 stackedModel = fitStackedModel(dfPredictions, y\_val)  
  
 # Evaluate base models with validation data.  
 print(f"\n\*\* Evaluate Base Models {j} \*\*")  
 dfValidationPredictions = pd.DataFrame()  
 for i in range(0, len(models)):  
 predictions = models[i].predict(X\_test)  
 colName = str(i)  
 dfValidationPredictions[colName] = predictions  
 print(models[i].\_\_class\_\_.\_\_name\_\_)  
 base\_evals, base\_pred = evaluate(models[i], X\_test, y\_test)  
 for eval\_key, eval\_val in base\_evals.items():  
 if eval\_key not in ['cm']:  
 print(f"\t{eval\_key}", eval\_val)  
 print()  
  
 # Evaluate stacked model with validation data.  
 stackedPredictions = stackedModel.predict(dfValidationPredictions)  
 print(f"\n\*\* Evaluate Stacked Model {j} \*\*")  
 evals, \_ = evaluate(stackedModel, dfValidationPredictions, y\_test)  
 results.loc[f"Model {j}"] = evals  
 j += 1  
# Show metrics  
print(results)  
print()  
averages = {}  
for col in results.columns:  
 if col not in ['cm']:  
 key = f"Average {col}"  
 averages[key] = results[col].mean()  
 print(key + ":", averages[key])  
 print(f"Std dev {col}:", results[col].std())