A Machine Learning Approach to Predicting Popularity of Music Records in Billboard Charts

October 2024

1 Introduction

Music industry, which globally generates billions of euros in revenue every year, revolves around very fierce competition - record companies fighting over whose artist or publication should use the biggest investments and promotion funds. Although machine learning has many different uses, it can also be used to create a business strategy. Despite all the big investments and marketing campaigns, from Tiktok to physical banners, earning real commercial popularity for music is still very uncertain.

In this project, we decided to explore the use of machine learning to address this challenge by predicting the popularity of music records. Machine learning offers many opportunities to analyze and use data to predict various models. Music history provides a large amount of data for training models, and thus we can see small parts of which music factors offer the greatest chance for success in the top charts.

Our dataset [1] consist of records that either made it to the Top 10 of the Billboard Hot 100 from 1990 to 2010 or fell short. With this project, we aim to demonstrate the benefits of machine learning by assisting record labels in making more informed decisions, also affecting financial risks. In section 2 we formulate the problem and give short description of the data. In section 3 we introduce our used method and give deeper information of the dataset. Finally, we state the results in section 4 and summarize the report.

2 Problem formulation

The primary objective of this supervised machine learning project, is to develop a machine learning model that predicts the likelihood of a music record reaching the Top 10 of the Billboard Hot 100 charts. The dataset used in this work was obtained in Kaggle [1] and contains numerical data associated with various songs. In the dataset all feature columns are numeric and the label is binary data. "Timesignature" and "key" -features are categorical and the rest are continuous variables.

3 Methods

3.1 Popularity of Music Records Dataset

Originally the dataset has been gathered from EchoNest, which is an music intelligence and information platform for developers and media companies. The data consists of 7574 rows with 39 columns. Each data instance comprises multiple feature vectors representing different properties and audio attributes, such as energy, tempo, and loudness, with values ranging from specific ranges like -4.707 to 0.853 for

energy and 91.525 to 160.512 BPM for tempo. The label column, Top10, contains values 1 and 0, indicating if the song made it to the top list.

Because the dataset was very imbalanced, we decided to undersample the majority class, ending up with a total of 2222 data points. Due to our sample size being quite large, the performance of our models should not suffer from undersampling. Column-set was also narrowed to 7 columns: duration, loudness, tempo, key, energy, length of song name and Top10 (binary variable indicating if the song made it to the top 10). These features were selected because they were clear, numeric/categorical and easy to use in the models. Some features such as "artist name" was dropped because the model we use do not handle text well. Also "pitch" was dropped, because it can not be found in Spotify Developers site, which provides information about songs in the same way as our dataset. [2]

Instead of using the song title as a feature, a new variable "songnamelength" is used as a feature because our models should handle numeric features quite well. The original dataset contained duplicate songs from different years, but since they had the same "Top10" value, we removed the duplicates. All rows with NA-values were also removed. Categorical features "timesignature" and "key" were transformed into dummy variables to make them compatible with the used models. We checked the Pearson correlations coefficients of the feature vectors and decided to drop "loudness" in our first method because it correlated strongly (0.74) with "energy", and "loudness" has lower accuracy in predicting the label than "energy".[6] Strong correlation between features hinders the performance of logistic regression. [8]

Training Set. The training set consists of a randomly selected 70% of the dataset, which is 1555 datapoints. This proportion is chosen because 70% is typically sufficient for training the model effectively. Random selection should make sure that the ratio of Top10=1 and Top10=0 is approximately the same as in the whole dataset.

Validation Set. Our validation set is 15% of the total dataset we are using which equals 333 rows. 15% should be enough to evaluate the performance of the models and provide sufficient evidence to choose the superior model.

Test Set. The test set is randomly selected 15% of the dataset, 333 rows. 15% should provide an adequate amount of data to test the performance of the final model.

3.2 Logistic Regression

Logistic regression is a statistical method used for binary classification problems, where the outcome is one of two possible classes. We decided to use logistic regression, because it handles both binary and continuous features well and predicts binary labels. In our case, we have exactly two outcomes: 1 = the song will make it to the top 10 and 0 = song will not make it to the top 10. [3]

Logistic regression works by mapping a weighted linear combination of features to a value between 0 and 1, which can be interpreted as a probability. Specifically, it estimates the probability that a given instance belongs to a particular class. If this probability exceeds 0.5, the instance is classified into class 1; otherwise, it is classified into class 0. [4]

In our scenario, the logistic regression model will use numeric input variables to compute the probability of a song making it to the top 10 chart. If the computed probability is greater than 0.5, the song is predicted to make it to the top 10. Given that our features are numeric and our label is binary, logistic regression is well-suited for this task.

For logistic regression we decided to remove feature "loudness" because it correlated strongly with feature "energy" and strong correlation tends to hinder the performance of logistic regression. GridSearchCV was used to fine tune parameter "iter_max" for the model.

3.2.1 Loss Function - Logistic Loss

The loss function is the function that the model optimizes. The logistic loss function was chosen because it allows us to leverage pre-built libraries for logistic regression. [5]

Log Loss is calculated as follows:

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

where: $\mathcal{D}$ is the dataset containing many labeled examples, which are (x_i, y_i) pairs. y_i is the label. Since this is logistic regression, every value of y_i must either be 0 or 1. p_i is model's prediction (somewhere between 0 and 1), given the set of features in x_i.

3.3 Random Forest

Another method that we use is Random forest, which is an ensemble machine learning algorithm. It is built from multiple decision trees, each using a random subset of the data and features to reach a single result. The process begins at the root node and continues splitting the data at each decision node until reaching the leaf nodes, where predictions are made. Randomness reduces overfitting and improves prediction accuracy. In classification tasks, we look for values that are either 0 or 1. This method was chosen because it works well with continuous and binary features and predicting binary labels. [7]

Again, GridSearchCV was used to select optimal number of decision trees and the function the splits the data at each node for the Random Forest Classifier.

3.3.1 Loss Function - Gini Impurity

RandomForestClassifier from scikit does not use a traditional loss function in the same way as many other classifiers (for example logistic regression). When constructing each decision tree, the Random Forest uses either Gini Impurity or Information Gain to split the data. We fine tuned the number of decision trees and which criterion function the model uses with GridSearchCV. Using this method, the best parameter for the function was Gini Impurity. [9]

Gini Impurity measures the impurity of a node, where lower values indicate purer nodes. The formula for Gini Impurity is:

$$Gini(p) = 1 - \sum(p_i^2)$$

where p_i is the proportion of class i in the node.

4 Results

Logistic regression and random forest was used to classify Top 10 songs based on musical features and accuracy, precision, recall and f1-score were used to validate the performance of the models. Average log loss was also calculated for the Logistic Regressor. Average validation log loss of 0.658 indicates an adequate performance but not exceptional (log loss ranges from 0 to ∞). Since the Random Forest Classifier does not use a loss function in a traditional sense, it was not useful to compare the average losses of the models. Instead, the other metrics were used to evaluate the performance the models and to compare the models with each other.

	Logical Regression	Random Forest Classification
Training accuracy	0.586	0.698
Training precision	0.586	0.697
Training recall score	0.572	0.695
Training f1-score	0.579	0.696
Training error	0.656	
Validation accuracy	0.646	0.673
Validation precision	0.643	0.653
Validation recall score	0.611	0.698
Validation f1-score	0.627	0.675
Validation error	0.658	
Test accuracy	0.608	0.635
Test precision	0.649	0.667
Test recall score	0.557	0.614
Test f1-score	0.599	0.639
Test error	0.656	

Table 1: Test results for logistic regression and random forest.

Accuracy tells how frequently the model correctly predicted whether a song made it to the Top 10 or not. Precision measures how many of the songs predicted by the model to be in the Top 10 actually are in the Top 10. Recall score measures the proportion of real Top 10 hits that were correctly predicted by the model. The F1 score is the harmonic mean of precision and recall.

As shown in the Table 1, training metrics seem to be slightly lower than the validation metrics for logistic regression, suggesting that the model may be underfitting during training.

For the Random Forest Classifier however, the training metrics are higher than the validation metrics, which indicates that the model might be overfitted.

Overall, the differences in test metrics between the two models are not substantial. However, the Random Forest Classifier has better validation performance than the Logistic Regressor. Thus, we confidently say that Random Forest Classifier solves our classification problem more efficiently than Logistic Regression. What's particularly interesting about these results, is that common song attributes available on Spotify can predict a song's success with an accuracy of 60-65%.

5 Conclusion

In this report, we apply machine learning to predict the likelihood of music records reaching the Top 10 of the Billboard Hot 100 charts, using logistic regression and random forest models. After preprocessing the data and selecting key features like duration, loudness, tempo, and energy, we split the dataset into training, validation and testing subsets. Then we apply logistic regression and decision tree classification.

The logistic regression model performed satisfactorily but was outperformed by the random forest model in most metrics, particularly with higher accuracy, precision, recall, and F1-scores on the validation set. The use of a random forest classifier offered better generalization, though there were signs of overfitting in the training data. Despite that, both models were able to capture meaningful patterns in the data, however suggesting that random forest could be slightly more suitable for this classification task than logistic regression.

In conclusion, while our models show some promise, there is room for improvement. Future work

could involve incorporating additional features, balancing the dataset more effectively, and experimenting with other machine learning algorithms to further enhance prediction accuracy. Nonetheless, this project demonstrates the feasibility of using machine learning to provide valuable insights into the dynamics of music popularity.

References

[1] www.kaggle.com. (n.d.). Popularity of Music Records. [online] Available at: https://www.kaggle.com/datasets/econdata/popularity-of-music-records.

[2] Spotify (n.d.). Web API Reference — Spotify for Developers. [online] developer.spotify.com. Available at: https://developer.spotify.com/documentation/web-api/reference/get-audio-features.

[3] Wikipedia Contributors (2019). Logistic regression. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Logistic_regression.

[4] GeeksforGeeks (2017). Understanding Logistic Regression. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/understanding-logistic-regression/.

[5] Reddy, S. (2021). Understanding the log loss function. [online] Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/understanding-the-loss-function-of-logistic-regression-ac1eec2838ce.

[6] Calkins, K. (2019). Correlation Coefficients. [online] Andrews.edu. Available at: https://www.andrews.edu/ calkins/math/edrm611/edrm05.htm.

[7] IBM (2023). What Is Random Forest? — IBM. [online] www.ibm.com. Available at: https://www.ibm.com/topics/random-forest.

[8] Tutorialspoint.com. (2023). Logistic Regression with Two Highly Correlated Predictors. [online] Available at: https://www.tutorialspoint.com/logistic-regression-with-two-highly-correlated-predictors

[9] Karabiber, F. (n.d.). Gini Impurity. [online] www.learndatasci.com. Available at: https://www.learndatasci.com/glossary/gini-impurity/.

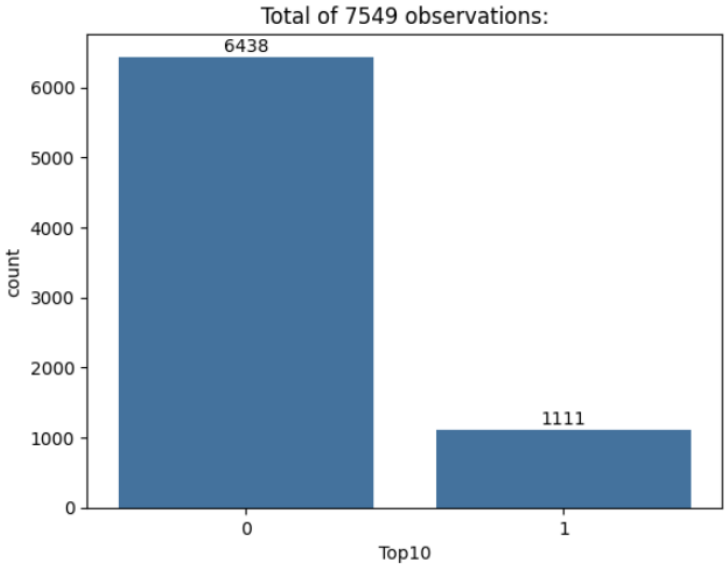
6 Appendix

The Python code and visualizations for this project are provided in the appendix at the end of the report for further reference.

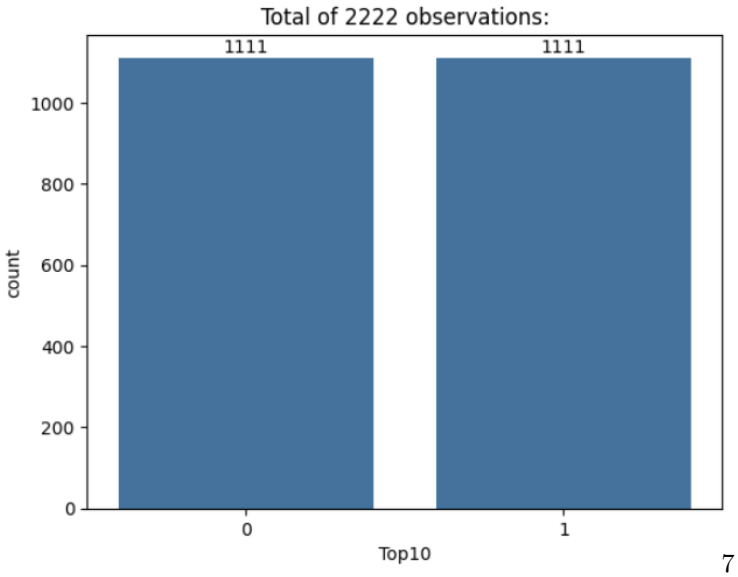
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, precision_score, f1_score, recall_score, log_loss
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE


## CLEANING THE DATA
df = pd.read_csv("C:/Users/User1/mlprojekti/songs.csv", encoding='ISO-8859-1') # Read the data

timbres = [elem for elem in df.columns if "timbre" in elem or "confidence" in elem] # Create a list of all column names that contain "timbre"

df.drop(timbres, axis=1, inplace = True) # Drop all columns names that contain "timbre" or "confidence"
df.drop(["pitch"], axis=1, inplace = True) # Drop column "pitch"


df1 = df.groupby("songID").filter(lambda x: (1 in x["Top10"].values) and (0 in x["Top10"].values))  # Check how many songs have at least 2 obs
# len(df1)) == 0. 0 songs have been on the top 10 chart in some year but have not been on the chart in some other year.
# This means that it is enough to simply remove duplicate songs

df = df.drop_duplicates(subset=["songID"], keep="first") # Remove duplicate songs, keep the first instance
df.drop(["artistname", "artistID", "year", "songID"], axis=1, inplace = True) # Drop more columns that will not be useful

df["songnamelength"] = df["songtitle"].str.len() # Create new column that states the number of characters in the title of the song
df.drop(["songtitle"], axis=1, inplace = True) # Remove column for the song title as it is not needed anymore

df.dropna(axis=0, inplace=True) # Remove rows with NA values

# Create dummy variables (0 or 1) for all categories of categorical variables
df = pd.get_dummies(df, columns=["key", "timesignature"], drop_first=True)

#Print info of the data set
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7549 entries, 0 to 7573
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   loudness         7549 non-null   float64
 1   tempo            7549 non-null   float64
 2   energy           7549 non-null   float64
 3   Top10            7549 non-null   int64
 4   songnamelength   7549 non-null   int64
 5   key_1            7549 non-null   bool
 6   key_2            7549 non-null   bool
 7   key_3            7549 non-null   bool
 8   key_4            7549 non-null   bool
 9   key_5            7549 non-null   bool
 10  key_6            7549 non-null   bool
 11  key_7            7549 non-null   bool
 12  key_8            7549 non-null   bool
 13  key_9            7549 non-null   bool
 14  key_10           7549 non-null   bool
 15  key_11           7549 non-null   bool
 16  timesignature_1  7549 non-null   bool
 17  timesignature_3  7549 non-null   bool
 18  timesignature_4  7549 non-null   bool
 19  timesignature_5  7549 non-null   bool
 20  timesignature_7  7549 non-null   bool
dtypes: bool(16), float64(3), int64(2)
memory usage: 471.8 KB
```

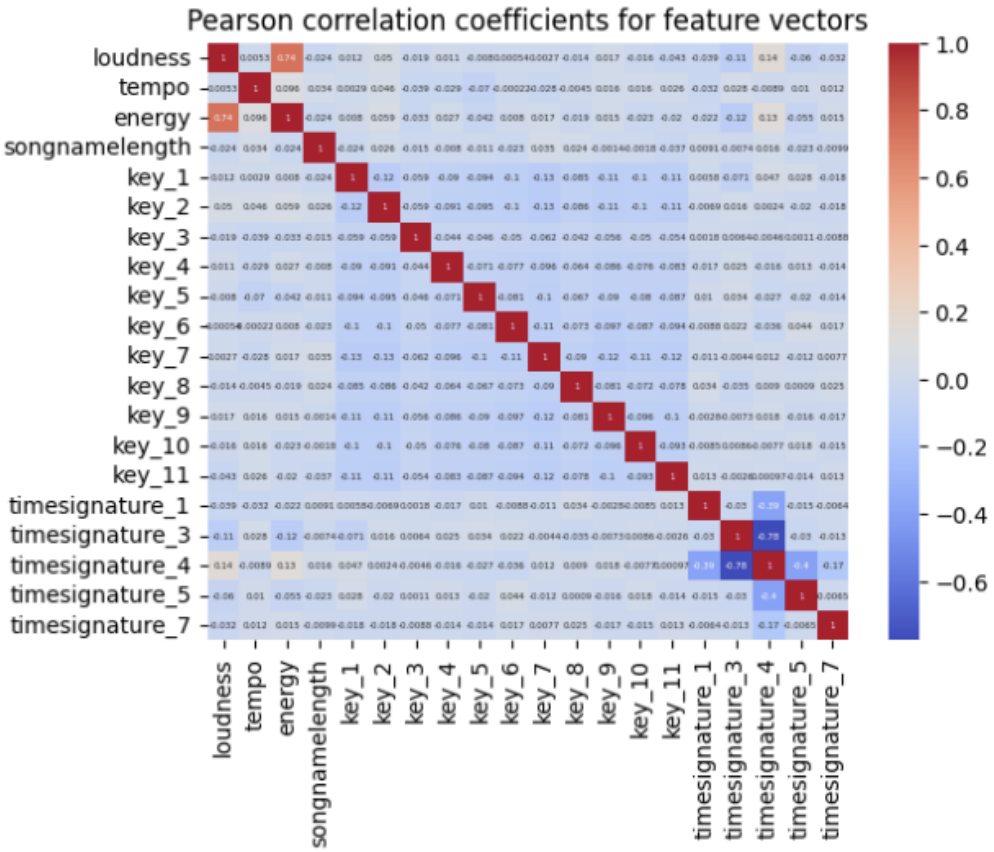
```
#PLOTTING
ax = sns.countplot(x="Top10", data=df)
plt.title(f"Total of {len(df)} observations:")
for p in ax.patches:
    ax.annotate(f"{int(p.get_height())}", (p.get_x()+p.get_width()/2, p.get_height()), ha = 'center', va = 'baseline',
                fontsize=10, color='black', xytext=(0, 3),
                textcoords='offset points')
plt.show()
```




```
df_label_1 = df[df["Top10"] == 1]  # Separate classes
df_label_0 = df[df["Top10"] == 0]

# Undersample the minority class to match the n of majority class to avoid problems of imbalanced data sets.
df_label_0_new = resample(df_label_0, replace=False, n_samples=len(df_label_1), random_state=42)

df_balanced = pd.concat([df_label_1, df_label_0_new])

y = df_balanced["Top10"] # Label vector
X = df_balanced.drop(["Top10"], axis=1) # Feature vectors

#PLOTTING
ax = sns.countplot(x="Top10", data=df_balanced)
plt.title(f"Total of {len(df_balanced)} observations:")
for p in ax.patches:
    ax.annotate(f"{int(p.get_height())}", (p.get_x()+p.get_width()/2, p.get_height()), ha = 'center', va = 'baseline',
                fontsize=10, color='black', xytext=(0, 3),
                textcoords='offset points')
plt.show()
```



```
correlation_matrix = X.corr()
ax = plt.subplot()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', annot_kws={"size": 4})
ax.set_title("Pearson correlation coefficients for feature vectors")
plt.show()
```




```
#Energy and loudness correlate too much (between 0.7-0.9). One has to be removed. Check which predicts the label with higher accuracy.

for feature in df.columns:
    if feature == "energy" or feature == "loudness":
        X_train1, X_test1, y_train1, y_test1 = train_test_split(df[[feature]], df["Top10"], test_size=0.2, random_state=42)
        clf1 = LogisticRegression()
        clf1.fit(X_train1, y_train1)
        y_pred1 = clf1.predict(X_test1)
        accuracy1 = accuracy_score(y_test1, y_pred1)

        print(f"Accuracy of classification based on {feature}, Accuracy: {accuracy1}")
```

```
Accuracy of classification based on loudness, Accuracy: 0.8622516556291391
Accuracy of classification based on energy, Accuracy: 0.8629139072847682
```

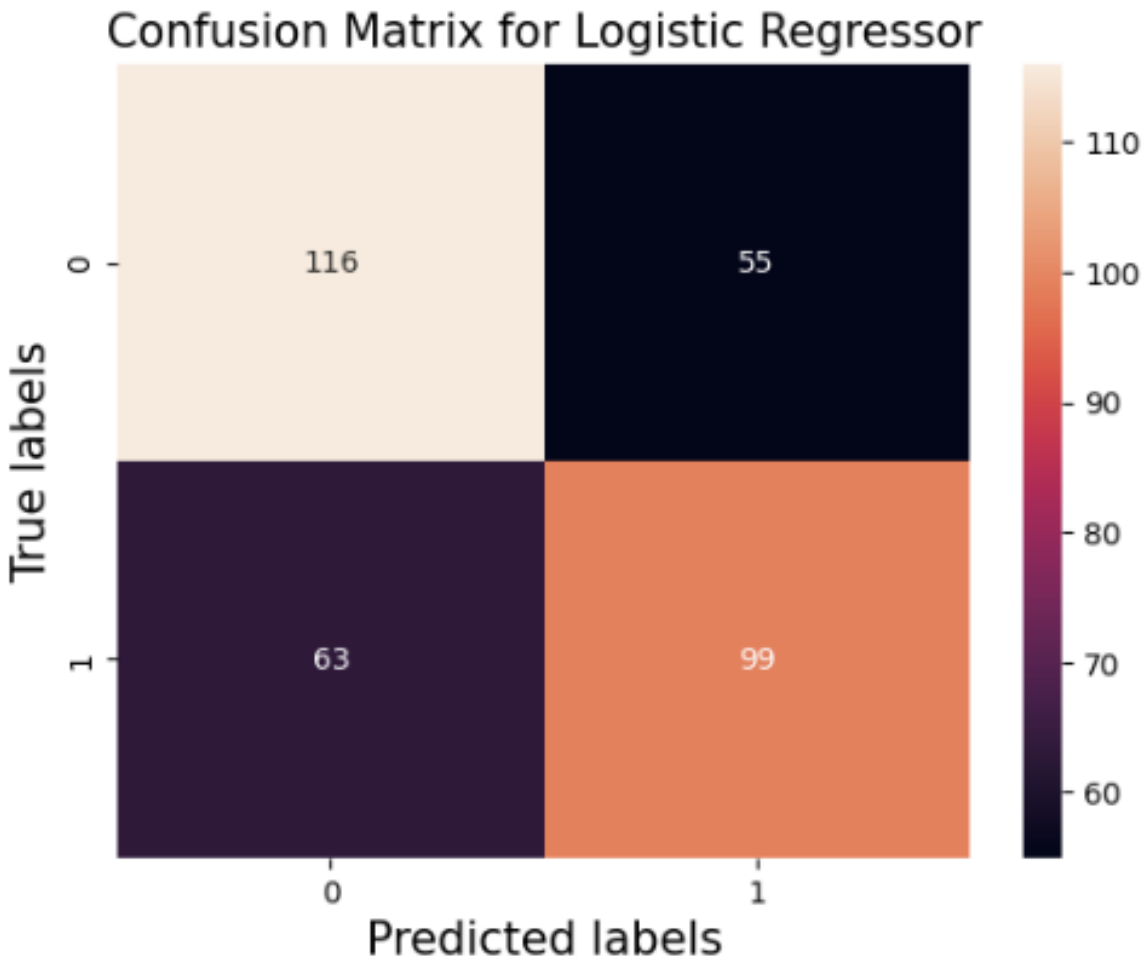
```
# loudness seems to predict the label slightly worse so let's remove energy
X.drop(["loudness"], axis=1, inplace=True)
```

```
## LOGISTIC REGRESSOR

# Divide the data to training and testing data
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size = 0.3, random_state = 42) # Split data into training and testing data
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size = 0.5, random_state = 42)
param_grid = {'max_iter': [50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]}
clf = LogisticRegression(solver='liblinear') # Our dataset is quite small so liblinear should be a viable solver
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")

best_clf = LogisticRegression(max_iter=best_params["max_iter"], solver='liblinear')
best_clf.fit(X_train, y_train) # Train the model

y_pred = best_clf.predict(X_val) # Predict y values based on testing feature data
y_train_pred = best_clf.predict(X_train) # Predict y values based on training feature data for training error
y_test_pred = best_clf.predict(X_test)


#Validation metrics
y_val_proba = best_clf.predict_proba(X_val)
val_error = log_loss(y_val, y_val_proba)
accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)
recall = recall_score(y_val, y_pred)

#testing meterics
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
y_test_proba = best_clf.predict_proba(X_test)
test_error = log_loss(y_test, y_test_proba)

#training metrics
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_f1 = f1_score(y_train, y_train_pred)
train_recall = recall_score(y_train, y_train_pred)
y_train_proba = best_clf.predict_proba(X_test)
train_error = log_loss(y_test, y_train_proba)
```

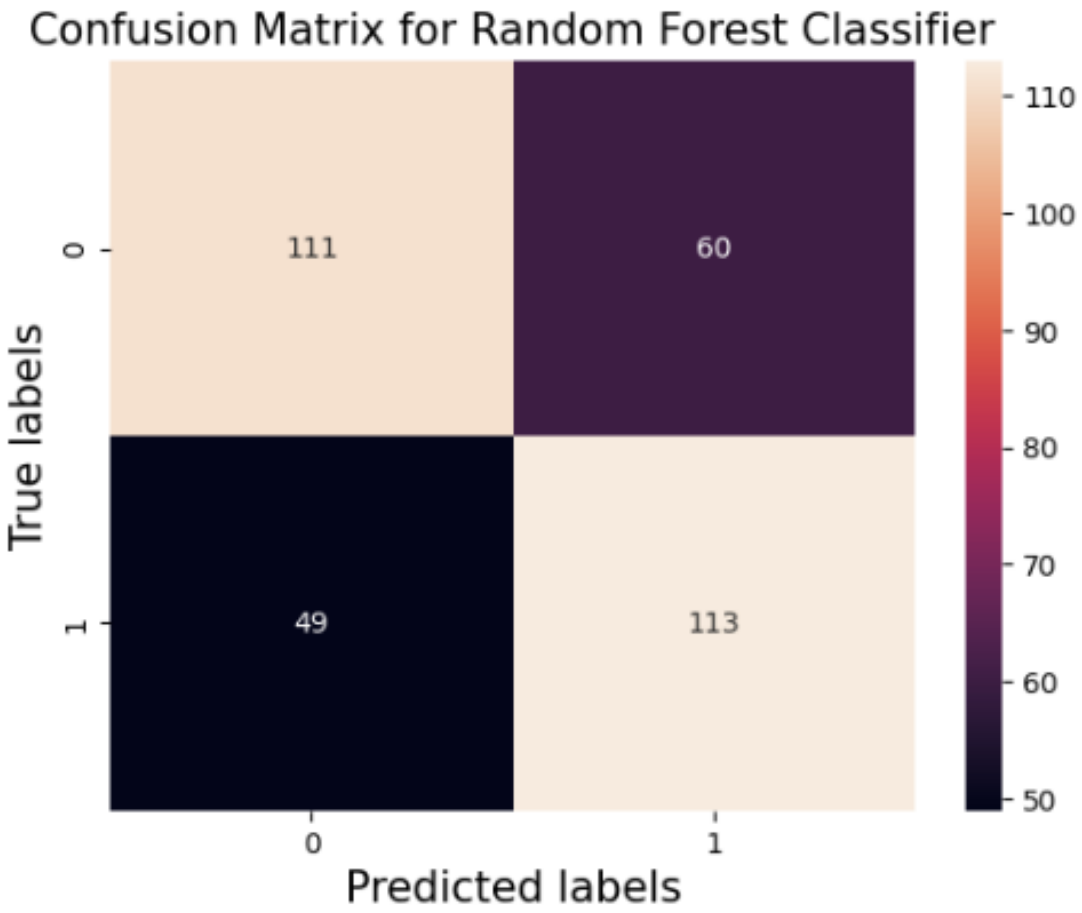
```
Best parameters: {'max_iter': 50}
```

```
# RANDOM FOREST CLASSIFIER
y = df_balanced["Top10"] # Label vector
X = df_balanced.drop(["Top10"], axis=1) # Feature vectors

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size = 0.3, random_state = 42) # Split data into training and testing data
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size = 0.5, random_state = 42)
param_grid = {'n_estimators': [100, 150, 200, 250],
              'criterion': ['gini', 'entropy', 'log_loss']
              }
rf = RandomForestClassifier(random_state=42) # Initialize random forest classifier
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
#print(f"Best parameters: {best_params}")

best_rf = RandomForestClassifier(n_estimators=best_params['n_estimators'], criterion=best_params["criterion"], max_depth=5, random_state=42)
best_rf.fit(X_train, y_train)

y_val_pred_rf = best_rf.predict(X_val)
y_test_pred_rf = best_rf.predict(X_test)
y_train_pred_rf = best_rf.predict(X_train) # Predict y values based on training feature data for training error

#Validation metrics
accuracy_rf = accuracy_score(y_val, y_val_pred_rf)
precision_rf = precision_score(y_val, y_val_pred_rf)
f1_rf = f1_score(y_val, y_val_pred_rf)
recall_rf = recall_score(y_val, y_val_pred_rf)

#Testing metrics
test_accuracy_rf = accuracy_score(y_test, y_test_pred_rf)
test_precision_rf = precision_score(y_test, y_test_pred_rf)
test_f1_rf = f1_score(y_test, y_test_pred_rf)
test_recall_rf = recall_score(y_test, y_test_pred_rf)

#Training metrics
train_accuracy_rf = accuracy_score(y_train, y_train_pred_rf)
train_precision_rf = precision_score(y_train, y_train_pred_rf)
train_f1_rf = f1_score(y_train, y_train_pred_rf)
train_recall_rf = recall_score(y_train, y_train_pred_rf)
```

```
conf_mat = confusion_matrix(y_val, y_pred)
ax= plt.subplot()

sns.heatmap(conf_mat, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title('Confusion Matrix for Logistic Regressor',fontsize=15)
plt.show()

print("Train metrics for logistic regressor:")
print(f"Training error (log loss): {train_error}")
print(f"Accuracy: {train_accuracy}")
print(f"Precision: {train_precision}")
print(f"F1-score: {train_f1}")
print(f"Recall score: {train_recall}\n")


print("Validation metrics for logistic regressor:")
print(f"Validation error (log loss): {val_error}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print(f"Recall score: {recall}\n")

print("Testing metrics for logistic regressor:")
print(f"Testing error (log loss): {test_error}")
print(f"Accuracy: {test_accuracy}")
print(f"Precision: {test_precision}")
print(f"F1 score: {test_f1}")
print(f"Recall score: {test_recall}")

print("\n"*5)


conf_mat_rf = confusion_matrix(y_val, y_val_pred_rf)
ax= plt.subplot()

sns.heatmap(conf_mat_rf, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title('Confusion Matrix for Random Forest Classifier',fontsize=15)
plt.show()

print("Train metrics for random forest classifier:")
print(f"Accuracy: {train_accuracy_rf}")
print(f"Precision: {train_precision_rf}")
print(f"F1 score: {train_f1_rf}")
print(f"Recall score: {train_recall_rf}\n")

print("Validation metrics for random forest classifier:")
print(f"Accuracy: {accuracy_rf}")
print(f"Precision: {precision_rf}")
print(f"F1-score: {f1_rf}")
print(f"Recall score: {recall_rf}\n")

print("Testing metrics for random forest classifier:")
print(f"Accuracy: {test_accuracy_rf}")
print(f"Precision: {test_precision_rf}")
print(f"F1 score: {test_f1_rf}")
print(f"Recall score: {test_recall_rf}")
```




```
Train metrics for logistic regressor:
Training error (log loss): 0.656478582129806
Accuracy: 0.5864951768488746
Precision: 0.5862068965517241
F1-score: 0.5789129011132941
Recall score: 0.5717981888745148

Validation metrics for logistic regressor:
Validation error (log loss): 0.6576922299618534
Accuracy: 0.6456456456456456
Precision: 0.6428571428571429
F1-score: 0.6265822784810127
Recall score: 0.6111111111111112

Testing metrics for logistic regressor:
Testing error (log loss): 0.656478582129806
Accuracy: 0.6077844311377245
Precision: 0.6490066225165563
F1 score: 0.599388379204893
Recall score: 0.5568181818181818
```




```
Train metrics for random forest classifier:
Accuracy: 0.6983922829581993
Precision: 0.6974025974025974
F1 score: 0.696046662346079
Recall score: 0.6946959896507116

Validation metrics for random forest classifier:
Accuracy: 0.6726726726726727
Precision: 0.653179190751445
F1-score: 0.6746268656716418
Recall score: 0.6975308641975309

Testing metrics for random forest classifier:
Accuracy: 0.6347305389221557
Precision: 0.6666666666666666
F1 score: 0.6390532544378699
Recall score: 0.6136363636363636
```