Music Classification Lab1

Azalea Alothmani

September 2021

1 Introduction

The following methods were implemented and evaluated in order to find the best model to use in production for classifying new unseen song as Like and Dislike.

- Logistic regression
- K-Nearest Neighbors (KNN)
- Random Forests, Bagging
- Boosting
- Support Vector Machines (SVMs)

2 Problem

The task is to classify a test dataset of 200 songs which contains a high-level audio features extracted using Spotify web-API. These features describe characteristics for classifying new songs such energy, danceability, valence, tempo etc. First we start with trying out different classification methods in order to find the best algorithm with highest accuracy.

Tools:

- Training data set with 750 songs classified as like (1) and dislike (0).
- Seaborn, matplotlib, pandas and numpy for data visualization and analyzing
- Sklearn to build the machine learning models

3 Exploratory Data Analysis

First step to obtain the objective described above was to do an exploratory analysis of the provided features. In addition, finding out the correlations between the 13 features is important in order to make the algorithms as efficient as possible.

3.1 Visualising the Data

The data was explored by visualization the variability of the 13 features. This was achieved by plotting the normal distributions for both like and dislike categories as shown below:

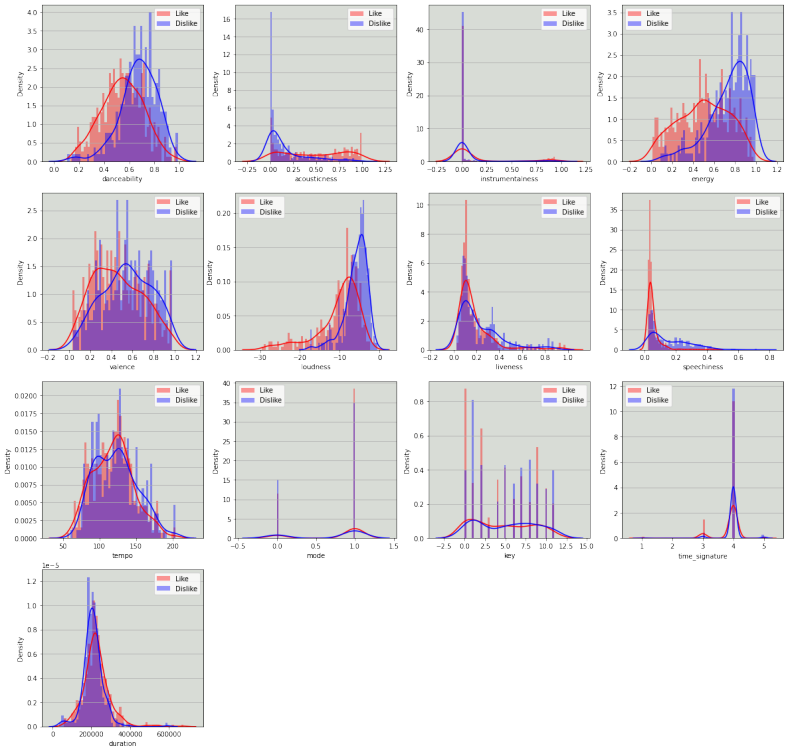


Figure 1: Frequency Distribution visualization of each of the 13 features.

As it could be noticed there is a significant overlapping between some features' distributions. Some features such as duration has a large weight which means that this feature will have a higher influence on the model than the other features. There are a clear distinctions between the two classes Liked and Disliked, specially in *danceability*, *loudness*, *acusticness*, *energy*, *liveness*, *tempo* and *speechiness*. However, there is a large overlap in some features such as *intrumentalness*, *duration* and even in *tempo*. As it also could be noticed there is a large overlap in the categorical features; Key, *mode* and $time_signature$, thus these variables might not have any impact on the model performance.

3.2 Correlation Map

Another interesting aspect is to study the correlation between the features as shown below

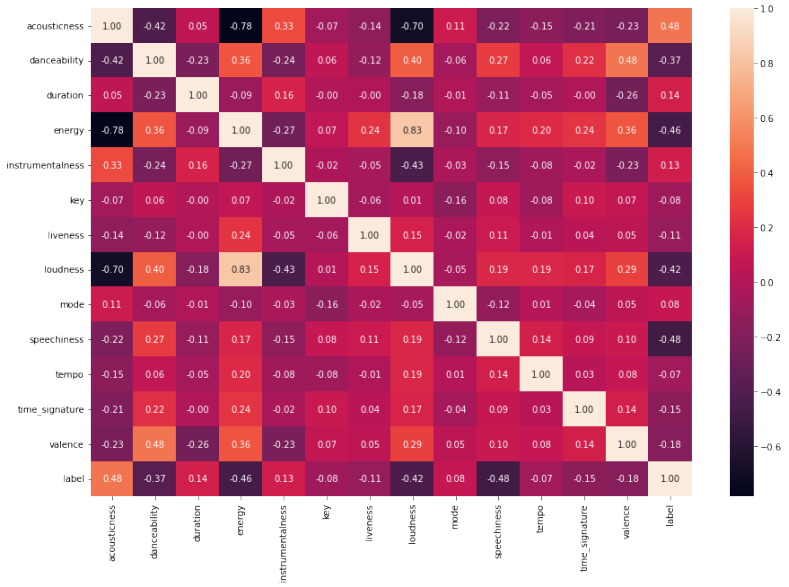


Figure 2: Correlation Map for all 13 features; numerical and categorical.

By studying correlation map in Fig.2 a couple of insights can be derived:

- Loudness and Energy are highly positively correlated features.
- Energy and Acusticness, Loudness and Acustincness are highly negatively correlated features.
- Valence and Danceability, Loudness and Danceability are positivily correlated features

In general, highly correlated features do not improve the performance of the models. For linear models such as logistic regression this multicolinearity can give solutions that are quiet unstable. For Random forests algorithms the highly correlated features can mask the ability of the algorithms to detect interactions between features. This is a special case of *Occam's Razor* where models with fewer features are preferable.

Considering that, it might be a good idea to not include Loudness together with Energy and Acusticness. Therefore, the algorithms will be evaluated based on two criteria:

- only consider including the numerical features
- Including all the 13 features, numerical and categorical
- Dropping some features, such as Loudness, Energy, Duration or Time-signature

4 Data Pre-processing

Furthermore, as it could be noticed in Fig.3 the standard derivation of the different features is not 1. Features' distribution should be centred around 0, and with standard deviation around 1.
Therefore and before algorithms implementation the data should be standardised to make different variables follow the same scale. This step is important to avoid

biasing the model towards variable/variables that have a larger weight than the other variables. There are different tools available for this purpose. For normalization StandardScaler() from Scikit-Learn is used in order to make the data more suitable before starting implementing the algorithms.

	acousticness	danceability	duration	energy	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	time_signature	valence	label
count	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000
mean	0.357394	0.596439	220112.733333	0.594188	0.100245	4.829333	0.203376	-8.509339	0.741333	0.098966	120.405761	3.902667	0.497321	0.602667
std	0.338405	0.172036	65587.690483	0.253301	0.259921	3.636001	0.177609	5.039488	0.438194	0.104715	28.378116	0.400091	0.239615	0.489673
min	0.000001	0.107000	33840.000000	0.009250	0.000000	0.000000	0.024000	-29.601000	0.000000	0.023400	55.747000	1.000000	0.033200	0.000000
25%	0.037150	0.480000	185490.250000	0.423250	0.000000	1.000000	0.094550	-10.173500	0.000000	0.035900	98.998000	4.000000	0.297000	0.000000
50%	0.244500	0.606000	215108.500000	0.631500	0.000010	5.000000	0.129000	-7.270000	1.000000	0.048750	120.104500	4.000000	0.483000	1.000000
75%	0.678500	0.715750	244236.750000	0.804750	0.002245	8.000000	0.264750	-5.097750	1.000000	0.113000	138.074750	4.000000	0.684500	1.000000
max	0.994000	0.986000	675360.000000	0.995000	0.967000	11.000000	0.979000	-0.533000	1.000000	0.721000	204.162000	5.000000	0.975000	1.000000

Figure 3: Illustrating the mean, standard derivation, max and min value for the 13 features.

Furthermore, some features such as *key*, *mode* and $time_signature$ are categorical variables as they contain label values rather than numeric values. Categorical features are non-numerical type of data that need to be pre-processed before building a machine learning models. This step is essential since many of machine learning models such regression and SVM, are algebraic meaning that their input has to be numerical accordingly.

As many machine learning algorithms cannot work wit categorical variables and to avoid poor performance one should consider converting the categories into numbers. OneHotEncoding from scikit-learn was used for this purpose. The basic strategy of One-Hot encoding is to convert each category value into a new column assigning 1 or 0 value to the column.

4.1 Normalization

The features were divided into two categories; one with the numerical audio features and another with only the categorical features; *key, mode, time-signature*. The second step was to use StandardScaler on numerical values and OneHotEncoder on the categorical to normalize the features. Finally, the dataset was split by 80 % for training and 20 % for testing/valdiation. The following procedure was done:

```
df = pd.read_csv('training_data.csv')
X = df.drop(columns=['label'])
y = df.loc[:,'label'].values

features = ['danceability','acousticness', 'instrumentalness', 'energy', 'valence',
            'loudness', 'liveness', 'speechiness', 'tempo', 'duration']
features_cat = ['time_signature', 'key', 'mode']

# Scaling the dataset to ensure more accurate results
from sklearn.preprocessing import StandardScaler
numerical_preprocessor = StandardScaler()
categorical_preprocessor = OneHotEncoder(handle_unknown="ignore")
preprocessor = ColumnTransformer([
    ('one-hot-encoder', categorical_preprocessor, features_cat),
    ('standard-scaler', numerical_preprocessor, features)])
```

```
model = make_pipeline(preprocessor, LogisticRegression(max_iter=1000,C=5,tol=0.01))
```

```
# Split into dataset into training and test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model.fit(X_train, np.ravel(y_train))
```

4.2 Evaluation

The training dataset is only slightly imbalanced among the two classes like (0) and dislike (1), according to Fig.4.

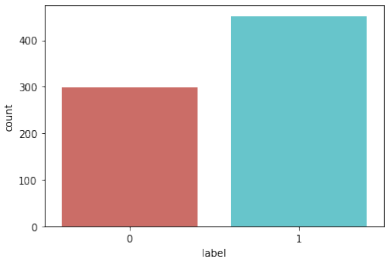


Figure 4: The two classes and their counts.

It is therefore important to use different methods to tackle classification problems and detect *naivebehaviours*. Different evaluation methods should be used. One example is Confusion Matrix from where some matrices to evaluate the accuracy of the model can be derived. The following evaluation metrics will applied:

- AUC: To study the relation between true-positive rate and false-positive rate
- Precision/specificity and recall/sensitivity

In addition, K-fold cross validation is another important step to evaluate the machine learning models. To evaluate the performance of the implemented algorithms hyperparameter tuning was performed using cross validation. Overfitting is a situation which occurs when the learning algorithm would have a perfect score on training data but fail to predict unseen data. To avoid overfitting k-fold cross-validation technique can be performed by first splitting the dataset into 2 parts, training and test data. The training data for training the model, while test data is hold for final prediction. The training dataset is split into k smaller sets and the model is trained using $k-1$ of the folds of the training data. The model is then validated on the test dataset.

Having a good accuracy means that the model has not overfitted. The process of splitting the dataset continues for k times and the average of these iteration is calculated, referred to as accuracy.

To show the performance of the classification model at various classification threshold settings ROC curve (receiver operating characteristic curve) can be used. ROC is a probability curve and AUC (area under curve) is a measure of separability. AUC near to 1 means that we have an excellent model. Higher AUC means that the model is good at distinguishing between the two classes Like and Dislike.

5 Creating The Models

The following models were implemented and evaluated:

- Logistic Regression
- K-nearest Neighbor
- Tree-based algorithms: Random Forests and bagging to improve performance

- Support Vector Machine (SVM)
- Boosting: Gradient Boosting

All the methods were evaluated using cross-validation, evaluation metrics and ROC curve.

5.1 Logistic Regression

In this section the Logistic regression model was first implemented using all the available features, that include both numerical and categorical variables. Best model performance was achieved by using all the features, hence only the evaluation of this model would be reviewed here. This algorithm can be seen as a first step to evaluate how the choice of features could influence the prediction accuracy.
¨ In order to build a model based on Logistic Regression, the training dataset was pre-processed and the model was evaluated. Best result was achieved using all the numerical and categorical features. Some of the important parameters one should consider are:

- Penalty: By default it is equal to 'L2' in Sklearn LogisticRegression(). It is called *Rigde regression*. This term allows the algorithm to be used with many features by shrinking the coefficients of the variables with less contribution close to zero. Lasso regression is another penalty term in which the coefficients of less contributes variables are forced to be exactly zero.
- γ is the regularization strength term that determines the flexibility of the model. Large γ decreases overfitting which mean more regularization. C is the inverse of regularization strength.

In order to find the best hyperparameters cross validation method was used. Best algorithm performance was achieved using the following hyperparameters:

- max-iter/number of iterations: 1000
- C-parameter: 0.1
- tol/threshold: 0.01

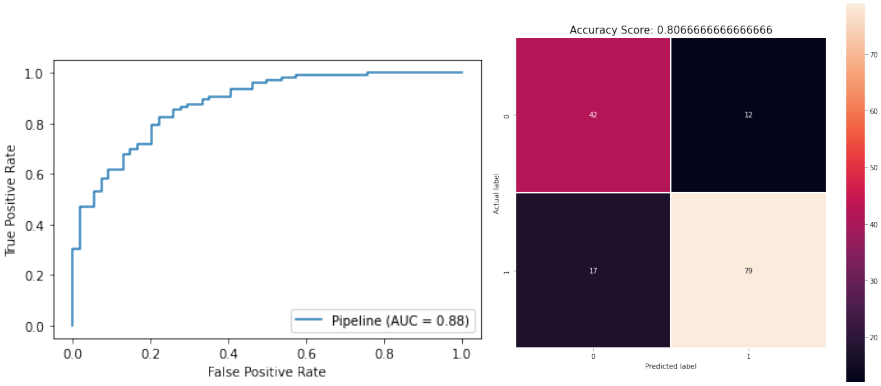
5.1.1 Logistic Regression Model Evaluation

The model is evaluated by estimating training accuracy and test accuracy on the training dataset on test dataset respectively using cross validation.

- Model evaluation: 81.33% accuracy with a standard deviation of 0.07
- Test score (validation = 80.667 %

Prediction:
The resulting confusion matrix for the prediction made on test dataset is compared with the true classes; Like and Dislike,

- Accuracy score = 80.667 %
- AUC = 0.88
- Precision = 0.79 and Recall = 0.80 (Calculated using classification report)

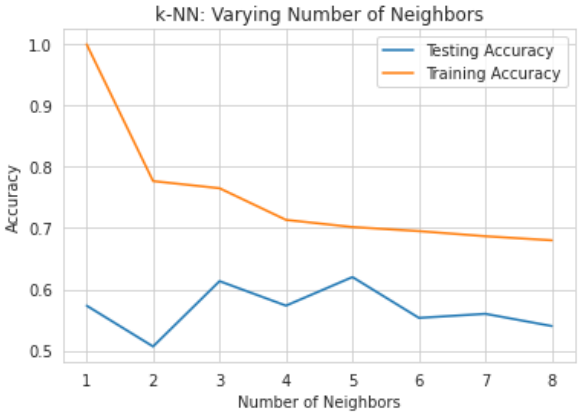


5.2 K-nearest Neighbors (KNN)

KNN is based on the assumption that similar data points exit in close proximity. The algorithm is performed by calculating the Euclidean distance of the K number of neighbors to estimate the number of data points among these k neighbors in each category.

The result from implementation of KNN algorithm: The algorithm was first created with all features included and it has been shown that the model performance was better than not included the categorical features. Cross validation and GridSerach was used to set the parameter n-neighbors for best model performance.

After running the evaluation test, it has been found that n-neighbors = 3 gives the best accuracy. Using 5 neighbors or more seems to result in a simple model that underfits the data.



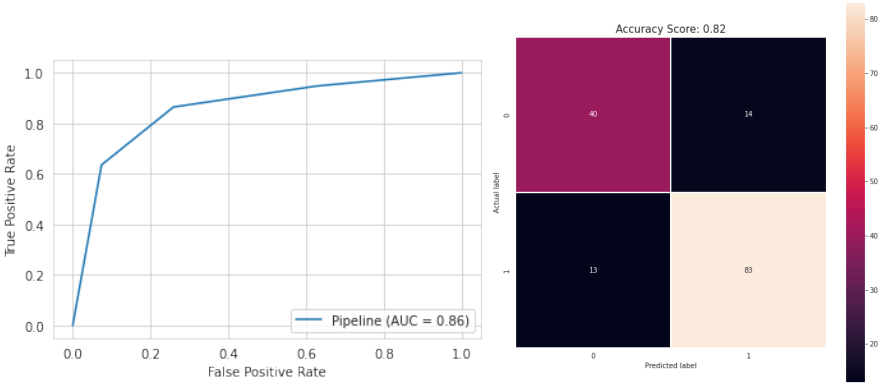
5.2.1 K-NN Model Evaluation

- Model evaluation: 81.07 % accuracy with a standard deviation of 0.07, cross validation with CV = 25

- Test score = 82.00 %.

The resulting confusion matrix for the prediction accuracy on test data is compared with the true classes; Like and Dislike. The result is shown below together with the resulting ROC.

- Accuracy score = 82.00 %
- AUC = 0.86, Recall = 0.80 , Precision = 0.81



5.3 Random Forests

A Random Forests model was implemented with all feature included. The dataset has been pre-processed by following the steps described in section.4.

Random Forest algorithm is implemented by using the ensemble algorithm called *Bagging* to create a random subsets of the training data. In fact, a bunch of tress are created, be trained on different subsets of the data. Essentially, booststrapping which is a part of bagging is used to sample data multiple times. Using Bagging could be significant in reducing variance, but not bias. That will influence the performance of unstable classifiers such as decision trees.

A RandomForestClassifier was first created with n-etimator = 100 and max-depth = 3. Training and test accuracy was then determined. This waas the first evaluation step before starting with tuning of hyperparameters to improve the performance.

```
[20]  1 # Create a random forest classifier, train it on training data and check its score on test data
      2 clf = RandomForestClassifier(n_estimators=100,max_depth=3)
      3 clf.fit(X_trainn,np.ravel(y_train))
      4 print('Train Accuracy: ', clf.score(X_trainn,np.ravel(y_train)))
      5 print('Test Accuracy: ', clf.score(X_testn,np.ravel(y_test)))

    Train Accuracy:  0.8733333333333333
    Test Accuracy:  0.8266666666666667
```

In order to find the best hyperparameters *GridSearch* is used where all possible combinations of parameters are evaluated with cross validation score as follows

```
clf = RandomForestClassifier()

param_grid = {
    'n_estimators': [100, 1000],
    'max_depth': [2, 3, 4],
    'max_features': [0.1, 0.3],
    'min_samples_leaf': [0.1, 0.3, 1, 2, 3]
    }

search = GridSearchCV(clf, param_grid, cv=4, verbose=1,n_jobs=-1)

search.fit(X_trainn, np.ravel(y_train))
score = search.score(X_testn, np.ravel(y_test))
print("Best CV score: {} using {}".format(search.best_score_, search.best_params_))
print("Test accuracy: {}".format(score))
```

```
Fitting 4 folds for each of 60 candidates, totalling 240 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   30.6s
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done 240 out of 240 | elapsed:  2.8min finished
Best CV score: 0.830000000000001 using {'max_depth': 4, 'max_features': 0.3, 'min_samples_leaf': 1, 'n_estimators': 1000}
Test accuracy: 0.8466666666666667
```

The best hyperparameters are:

- n-estimator = 1000
- max-depth = 4
- min-samples-leaf = 3
- max-features = 0.3

5.3.1 Random Forests Model Evaluation

The model was this created accordingly. The accuracy of the classification was evaluated by computing the *Confusion Matrix*. Accuracy score of 83.748% was achieved.

K-fold cross validation (K = 20)	82.27%
Train accuracy	89.5%
Test accuracy	83.33%
Precision	81.0%

Prediction accuracy on test data:
The resulting confusion matrix for the prediction made on test data is compared with the true classes; Like and Dislike, is shown below together with the resulting ROC.

- Accuracy score of 83.748 %
- AUC = 0.88, Precision = 0.81 and Recall = 0.80

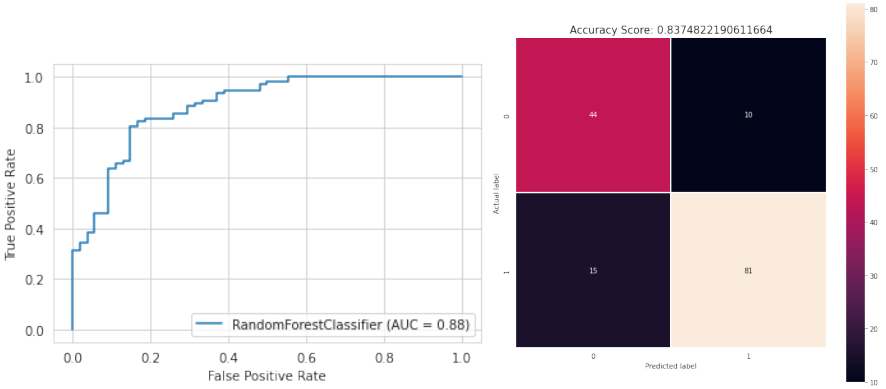
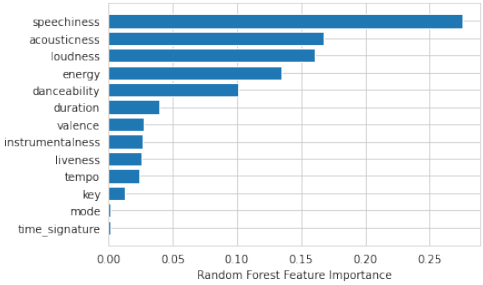


Figure 5: The resulting ROC and Confusion Matrix .

5.3.2 Feature Importance

The last step was to evaluate how changing some features will influence the performance. As it has been described in section.3.2, Energy is highly correlated with features Loudness and Acustincness, thus Energy was dropped to evaluate if model performance could be improved. According to random forest feature importance method, Energy is less important than Loudness and Acustincness, therefore dropping Energy was considered as a good choice. However, dropping Energy showed no accuracy improvement. Other features e.g. Valence and Duration were also dropped without resulting in any accuracy improvement.



The model was then evaluated in the same way as described before. After rerunning the model with different choices of features it was concluded that best model performance could be achieved with all the numerical and categorical features included.

5.4 Boosting

Algorithm based on Gradient Boosting, which is an iterative and sequential approach, has been implemented. Gradient boosting is robust to over-fitting and it is an algorithm in which the loss function is minimized by iteratively choosing a function that will results in convex optimization curve. This algorithm is built on the idea of adding trees "weak learner" one step at a time.

In order to improve the performance the following hyperparameters were considered:

- n-estimators: The number of boosting stages, larger number results in a better performance
- learning-rate: Shrinks each tree contribution
- max-features: Number of features that will result in the best split
- max-depth: The depth of each individual regression estimators/trees

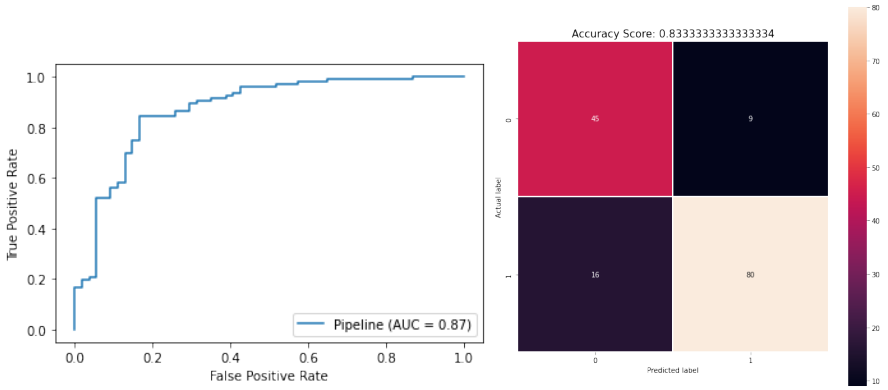
After pre-processing the data, as described previously, the Gradient Boost Algorithm was trained and the accuracy at different learning rates ranged from 0 to 1 was checked. In addition, the algorithm was trained at different max-features ranged from 5 to 13, and also at different n-estimators ranged form 5 to 30.

In conclusion, highest accuracy score on training data set and validation dataset was achieved with:

- n-estimators: 20
- learning-rate: 0.75
- max-features: 13
- max-depth: 2

5.4.1 Gradient Boosting Evaluation

The Gradient boost algorithm was then trained based on the best hyperparameters.



- Model evaluation on dataset: 80.93 % accuracy with a standard deviation of 0.06, cross validation with CV = 25
- Test score (validation) = 83.33 %.

The resulting confusion matrix for the prediction made on test data is compared with the true classes; Like and Dislike, is shown below together with the resulting ROC.

- Accuracy score = 83.33 %
- AUC = 0.87, Precision = 0.82 , Recall = 0.83

This algorithm was believed to gain a higher accuracy than random forests but it was not the case.. This might depends on the fact that Gradient Boosting overemphasized outliers since the algorithm tends to continue improving in order to minimize the errors. This results in high model flexibility and therefore a larger grid search should be preformed.

5.5 Support Vector Machines (SVM)

SVMs are supervised machine learning models and are more commonly used for binary classification problems, (SVC - Support Vector Classification). The aim of SVC is to find the best hyperplane/decision boundary that best separates the dataset into two classes. The decision boundary is called hyperplane if the number for features are more than two. The goal is to find a hyperplane that has the greatest possible margin to any support vector/data points.

Mathematically, the hyperplane is defined as:

$$\omega_1 x_1 + ... + \omega_d x_d + \beta_0 = 0$$

d is the number of features, x_d is a feature, ω_d represents weights and β_0 is a bias term.

SVM aims to solve the optimization problem by increasing the distance of decision boundary to support vectors and maximizing the number of correctly classified data points. This trade-off is controlled by the hyperparameter C. Small C-parameter results in more regularization by increasing the boundary margin, meaning increase number of misclassificed data points. Vice verse applies for large C-parameter.

In most cases there is no clear linearly separation between data points, and it is when **Kernal Trick** can be useful. Basically, the data points belonging to different classes are allocated to different higher dimensional space based on some **Kernal functions** that include linear, polynomial and radial basis function (RBF).
Kernal SVM was implemented using Scikit-Learn. To train the algorithm same *svc* class from the *svm* library is used, where the value of hyperparameters are changed for each of the three tested kernels; linear, Gaussian/RBF and polynomial.
Best result was achieved with polynomial kernel with only numerical features included, thus only the implementation of this model will be reviewed here.

First, we start training out model by using SVC() function without tuning of the hyperparameters. Next step was to use the meta-estimator GridSearch to find the best parameters that give the highest accuracy.

```
model = svm.SVC()
model.fit(X_trainn, y_train)

# print prediction results
predictions = model.predict(X_testn)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.81      0.79        54
           1       0.89      0.85      0.87        96

    accuracy                           0.84       150
   macro avg       0.82      0.83      0.83       150
weighted avg       0.84      0.84      0.84       150
```

```
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.01, 0.1, 1, 5, 10, 100, 1000],
              'gamma': [1, 0.1, 0.7, 0.01, 0.001, 0.0001],
              'kernel': ['poly']}

grid = GridSearchCV(svm.SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_trainn, y_train)
```

The model was then fitted on the best parameters and the prediction was evaluated with the classification report and confusion matrix on this grid.

In order to improve the performance the following hyperparameters were considered:

- degree: Higher degrees Kernel yields a higher dimensional hyperplane
- gamma: larger γ leads to a greater curvature of the decision boundary
- C: small C-parameter results in more regularization by increasing the boundary margin

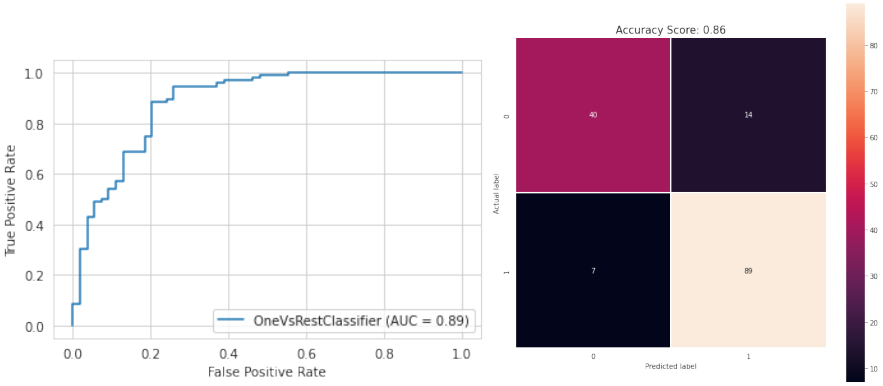
5.5.1 Polynomial Kernel Evaluation

In conclusion, highest accuracy score on training data set and validation dataset was achieved with:

- Kernel: 'poly'
- degree = 3
- gamma = 1.0
- C = 0.01

The resulting confusion matrix for the prediction made on test data is compared with the true classes; Like and Dislike, is shown below together with the resulting ROC.

- Accuracy score = 86.00 %
- AUC = 0.89, Precision = 0.86 and Recall = 0.83



The accuracy score achieved with SVM using polynominal Kernal is better than what could be achieved with Random Forests, the best evaluated algorithm so far.

6 Conclusion

To summarize the accuracy result from each implemented algorithm is shown below

	ACCURACY	PRECISION	RECALL
KNN	82.0%	81.0%	80.0%
RandomForests	83.748%	81.0%	80.0%
Logistic Regression	80.667%	79.0%	80.0%
SVM	86.0%	86.0%	83.0%
Boosting	83.33%	82.0%	83.0%

The implemented models were good at classifying the two classes Like/0 and Dislike/1. The best accuracy was achieved with SVM using Polynomial Kernel. However, the accuracy of this model on the unseen 200 songs was not evaluated with the Competition: Music Taste Prediction - Leaderboard. It was due to the fact that Random Forests algorithm was believed to be the best model, with 80.0 % performance accuracy on the Leaderboard. The approach was to keep improving this model by testing different hyperparameters and feature choices. After three submissions the accuracy of Random Forests algorithm was not showing any further improvement. Based on that, a decision was made to improve another more robust algorithms. It was found that SVM Polynomial Kernal achieved an accuracy score of 86 %, around 4 % better than Random Forests.

In most classification problems, it is assumed that the dataset points are linearly separable. By introducing kernel the input data can be converted into a higher dimensional space by finding the hyperplane that classify the data correctly.

However, random forests yields better performance than the other algorithms. This is due to the additional randomness added to the model. By using booststapping a random subsets are created and best split is considered by searching for the best feature among the randomly created subset of features.