# Mini Project


## Abstract

The main objective of this paper is to provide a most efficient classifier model yielding maximum accuracy of classifying the songs and predicting likeness/dis likeness of songs by Andreas when put in production. Based on the modelling process and the findings of the current study, it is evident that the performance of machine learning models is often influenced by different factors including the type of parameters used, the implementation approaches, among other factors We were expected to gain experience on different machine learning algorithms used in real world datasets and to submit reports of selected algorithms. Also, ethical aspects in providing trustworthy ML solutions were projected. Number of group members:4


## 1 Introduction

This paper showcases four methods of classifiers namely Logistic Regression, Discriminant family models, Boosting and KNN Classification for training, modelling, prediction, tuning, evaluation, and validation using the given dataset. In Supervised learning, we have a training set, and a test set. The training and test set consists of a set of examples consisting of input and output vectors, and the goal of the supervised learning algorithm is to infer a function that maps the input vector to the output vector with minimal error. For each of methods equivalent parameters are explored and projected how it can be tuned to improvise the model with respect it's accuracy, performance and other critical evaluation parameters. Finally, for all the models, its tuned performance is applied to see which method performs better when put in production. The use of distinct tuning of parameters serves as novelty in this mini project.

## 2 Logistic Regression

Logistic regression falling under the classification type of machine learning models the relationship between set of input variables and binary output. It estimates the probability of classes given the input and classifies the one having higher probability [1]. Following subsections are a set of methods applied for fitting, modelling, and evaluating the logistic regression function. Two cases of input are taken into consideration namely one the inputs **as it is**, and other inputs are **normalized**

### 2.1 Generating Logistic Regression Model

The raw data coming from the training data is split into 60 percent of training data for the purpose of fitting models and evaluating the obtained logistic regression model from the rest available 40 percent test data. Logistic Regression model determines the probability of likeness / dis likeness whose function is described in equation 1. Here LR **model before and after normalizin**g [2] inputs are generated.

LR(x)=p(y=class/x)=$e\,\frac{e^{\theta 0+\theta 1x1+\theta 1x1+\theta 2x2+\cdots\ldots\ldots\theta 13x13}}{1+e^{\theta 0+\theta 1x1+\theta 1x1+\theta 2x2+\cdots\ldots\ldots\theta 13x13}}$ --- Eq (1) where class=1(likeness),0=(dis likeness)The Coefficients for corresponding inputs stated in training data for the obtained Logistic Regression Model whose function is described in equation 1 listed in **(**Appendix A)

### 2.2 Accuracy, Confusion Matrix, ROC, AUC, and Threshold Settings

Once the LR models for both cases are generated the next step resides in predicting the model with rest of available data designated for testing and performance of the model is evaluated by computing its accuracy , Confusion matrix ,AUC for ROC Curve for predicted results with respect to test data The accuracy and confusion matrix for the above logistic function are projected below for both cases ROC Curve(Appendix A) basically computes the TPR vs FPR at various thresholds which are not only useful for tuning r but also very powerful in projecting the AUC which measures the performance of the model i.e how well it gets distinguished between the classes . The AUC when computed before and after normalizing the input was found to be 0.77 and 0.86. Also , the value of threshold should be chosen in such a way that it shouldn't be neither extreme 1 (Low TPR/FPR) nor low (high TPR/FPR).From the observation of ROC Curve (Appendix A) $0.3<r<0.4$ would be suited for both cases .

Table 1 Confusion Matrix, Accuracy and AUC

| | LR Model Before Normalising | | | LR Model After Normalising | | |
|---|---|---|---|---|---|---|
| **Confusion Matrix** | 0 | 1 | Precsion/Recall | 0 | 1 | Recall/Precision |
| **0** | 71 | 37 | 0 0.60/60 | 85 | 29 | 0 0.75/71 |
| **1** | 48 | 144 | 1 0.75/80 | 34 | 152 | 1 0.82/84 |
| **Accuracy** | **0.717** | | | **0.79** | | |
| **AUC** | **0.77** | | | **0.86** | | |

Comparing the results obtained before and after normalising showcased in Table 1 and (Appendix A) it can be inferred that LR model after noramlising the inputs performs better than without normalising inputs . Both true positives and true negative negatives increases in case 2 yielding the accuracy of the model to 0.79 which makes better in fitting the test data .Therefore, considering all the cases and parameters for evaluating the Logistic Regression model for both cases, the model with normalizing input performs better with the accuracy of 79% and yielding performance to 86%

# 3 Discriminant Analysis

## 3.1 Linear Discriminant Analysis (LDA)

LDA ("classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule" [3]). The model conducts separation by computing direction i.e. linear discriminants. Ideally, the objective of LDA is to help reduce dimensionality by using the transform method to project data into the most discriminative directions thus using fewer features to predict the class of the target attribute. LDA is implemented in this project using the Linear Discriminant Analysis () class which takes parameters such as tolerance(tol), store covariance, n_components (number of components), shrinkage, and solver parameters [4]. The solver parameter also determines the usage of other parameters such as tol(used with singular value decomposition (svd) solver only) and shrinkage. Using the grid search (Appendix B) option, the parameters optimized for the LDA model include n_components and the tolerance value i.e. tol is used to estimate the rank of a feature.

## 3.2 Quadratic Discriminant Analysis (QDA)

The QDA assumes that the target attribute has two labels and there is a multivariate gaussian distribution like LDA, however fails to assume that there is an equal covariance matrix for each of the labels [5]. As such when classifying whether a given set of observations belong to a specified class, QDA just picks the label with the minimum Euclidean distance from the supplied data X the mean of vectors that is in each label. In this study, the QDA was implemented using Quadratic Discriminant Analysis() class, which assumes parameters such as priors (priors include class proportions learned from the training set), regularization parameter i.e. reg_param which regularizes the covariance estimates by scaling the attribute of a given class, store covariance (includes covariance matrix estimated using the examples of a class) , and tol which is also used to estimate the rank of an attribute [6]. During hyperparameter tuning (Appendix B), the parameters to be optimized were set to include reg_param and tol.

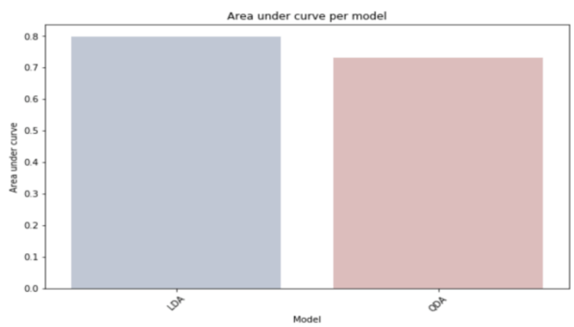
### 3.3 Results

How well LDA and QDA specified above performs as well as the criteria of how the best model was chosen. Table 2 and figure 1, below show the classification accuracy and the area under the curve for each of the models showing the best and the least performing model. Parameter tuning is done by giving the least tolerance value or the default tolerance value to different components, the accuracy varies when we are testing the model in production (Appendix B).

Table 2: Model Performance

| | Model | Classification Accuracy | Area under curve |
|---|---|---|---|
| 0 | LDA | 81.467 | 0.796857 |
| 1 | QDA | 71.067 | 0.730972 |

Figure 1: Area under the curve




From table 1 above, it is established that the LDA model has the highest classification accuracy (81.467) with the highest area under the curve (0.796857) (Appendix B) ,hence is considered the fit model for predicting whether Andreas Lindholm will like a song or not in production since the AUC provides a general performance of a machine learning model i.e. It evaluates how well predictions are ranked, instead of only their absolute values.

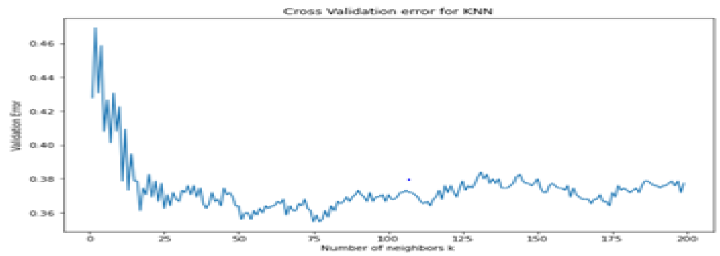
## 4 K Nearest Neighborhood

K Nearest Neighbour (KNN) is a supervised machine learning algorithm, which classifies data based on closest training neighbour data. We used the Euclidean distance that calculates the square root of the sum of all squares of the attributes' value differences. The data is classified based on the majority vote of its neighbours [9]. This model is useful when we do not have or have very little idea of the distribution of data. With respect to the current problem of classification of songs, we are taking all features except 'Label' of training data as our input X and 'Label' as result y. Following are the steps of the algorithm.

- Considering all data in training_data.csv as training data and evaluating the training error with respect to different values of k.The maximum prediction accuracy we got for training data is 0.792 and error rate as 0.208 for k=2 .When k=1 ,the error rate is 0.0 for the training sample .However error rate is different for validation data at k=1.So we are over fitting the boundaries at k=1.
- Splitting the data randomly into a training and validation set and see the validation error rate wrt different value of k. The validation error rate we got is 0.3657 for k=21.
- For the evaluation of the best value of k, where error rate is minimum, we are performing 10-fold cross validation to select the best model. The data were automatically divided into ten equally sized parts and the training and testing process was conducted ten times, with each of these parts being the test data once and the remaining parts being used for training. After training and prediction, model gives the classification of each song in test file like (1) /dislike (0).

### 4.1 Results

Figure 2 Cross Validation




Model is giving the best result when k=76 .Accuracy of the model is 0.653 and error rate is 0.354.For the extreme case k=1, bias is 0 for training data .But for validation data at k=1, the result varies a lot which means the chances of error is high. So when k increases the training error will increase but the test error will decrease which means the variance will decrease [13].

Table 3: Model Performance

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.54 | 0.50 | 0.52 | 28 |
| 1 | 0.71 | 0.74 | 0.73 | 47 |
| accuracy | | | 0.65 | 75 |
| macro avg | 0.63 | 0.62 | 0.62 | 75 |
| weighted avg | 0.65 | 0.65 | 0.65 | 75 |

After getting the optimum value of k, we evaluated the confusion matrix, precision and recall of the model. The obtained precision of LIKE class is 71% which means 71% of the songs which Andears likes are correctly identified (Appendix C) . Also, the recall / true positive rate is obtained as 74%. However, when the model is applied on production data the prediction accuracy is reduced to 0.575. This could be because of the value of k=76 which makes the model very complex and results in high bias. Which in turn results in the algorithm missing the relevant relationship between the features and target.

## 5 Boosting

AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get a high accuracy strong classifier [10]. AdaBoost uses Decision Tree Classifier as default Classifier.

Created model with default values of Classifier and taking all input parameters into consideration as features for the training model. Estimators are useful to train weak learners iteratively to make strong, created different models with different estimators

model_boost = AdaBoostClassifier()
model_boost_70_estimators = AdaBoostClassifier(n_estimators=70)
model_boost_100_estimators = AdaBoostClassifier(n_estimators=100)

AdaBoost use SAMME.R as default algorithm, I have used model with SAMME.R algorithm with default estimators 50 and created another model with SVC as base estimator

model_svc =AdaBoostClassifier(n_estimators=50, base_estimator=svc)

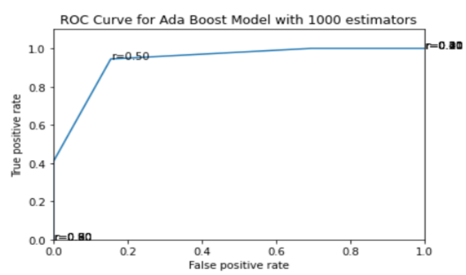
We need to tune several hyperparameters in order to develop the most accurate possible. We have to place in the grid several values for each of these. Once we set the arguments for the AdaBoostClassifier and the search grid we combine all this information.

```
ada_grid={n_estimators[100,500,1000],learning_rate:[.001,.01,.1]}
search=GridSearchCV(estimator=adaboost, param_grid=ada_grid, scoring=’accuracy’)
```

Higher AUC is higher performance of model. Confusion matrix will provide True positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) rates, each model will give different rates. Less TN and FN are better performance of model. Calculate ROC and AUC of all models which will help to select the best model to run on test data.

After calculating confusion matrix, ROC and AUC for above training models (Appendix D), opting AdaBoost with 1000 estimators and learning rate 0.1 to run on production data for good prediction results. It has given 79.5 per accuracy on production data.

Fig 3: ROC of Adaboost

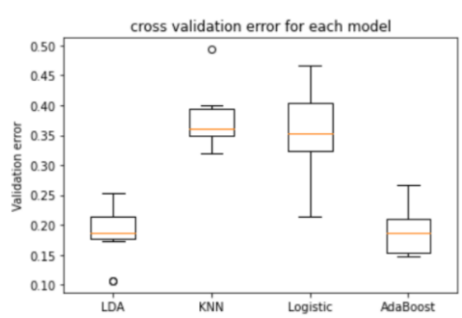



## 6 Model Selection

Table 4 Accuracy Comparison

| Methods | LR after Normalisation | LDA | KNN | Boosting |
|---|---|---|---|---|
| A) Accuracy on Training Data | 79 | 82 | 65.3 | 82 |
| B) Accuracy on Production | 76 | 81 | 57.5 | 79.5 |

Figure 4 Accuracy Comparison




Three cases are considered for selection of better classifier thinking from a production point of view. First one being the accuracy of all tuned classifiers are taken into consideration. Secondly evaluation done with respect to 10-fold cross validation on training set of data. Lastly performance when put in production for all of models is taken into consideration. It is evident that from considering all the cases LDA performed well in all of the above cases yielding to accuracy of 82% on training data , accuracy of 81 % obtained from production evaluation and error rate of 18 percent yielded from 10 fold cross validation method

## 7 Ethical Reflection

Whereas there are different sources of bias in research. One thing is sure i.e. bias can lead to distorted results and misinformed conclusions [12]. Since machine learning models work with data, the predicted outcome will be based on how diverse the used data is. For instance, if the past insurance data involves members of a certain gender or race or income group predominantly acting in a given way, the prediction of how a future client will behave may be biased towards the experience gained from the data by the algorithms. As there is a good chance for the system to inherit several biases like cultural bias, gender bias by training data, results predicted by models can be biased. Popular examples are like "Amazon sexist model for recruitment", it was rejecting many females resumes for technical positions as the model was trained on previous data which included more male resumes. Unfortunately, in the past many positions were filled by male candidates due to several reasons.

**7.1 View Yes:**

1. It is the responsibility of a Data Scientist to inform appropriate higher officials if the model is giving biased results. If the client is not informed of the result, it's going to harm the reputation of the company and the company will lose the trust of the client.

2. Also by not informing the client about the issue, will hamper the contractual situation with the client where client might have high expectation on the systems performance. To hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public [11].

**7.2 View No:**

- As a ML developer for a developing organisation, we are not directly responsible to the insurance company for real world bias results. The Insurance company is responsible to implement the system and in turn responsible for the biases introduced in the system. The implementing company should test for quality of the product for appropriate results without any bias towards some people. It is always difficult to assess the accuracy of the models with AI systems, especially when it comes to advanced models in neural networks. That is what these systems are considered as black-box systems as it is difficult to understand the result. In the AI systems, the traditional model of responsibility fails because nobody has enough control over the machine's actions.

- As we disclose the risk and limitations of the system ,the insurance company might opt to do business with other firms that are not "biased" .Which might not be true for them except that they are not open with their clients about the underlying risks of using algorithms. While doing implementation we may not have required data due to many possible reasons, if any unwanted results were coming while testing company should consult for improvement of system to give better predictions

**7.3 Group Opinion on the Ethical Aspect:**

It is our responsibility to avoid real conflicts of interest whenever possible, and to disclose them to affected parties when they exist. To improve our technical aspects of the system we can implement restrictions to the system so that corrupt data cannot be introduced to the system which will in turn induce unwanted bias in the system. We should strive to ensure the code is upheld, and to not retaliate against individuals reporting a violation.

# 8 References


1. Brownlee, J., 2020. *Logistic Regression For Machine Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/logistic-regression-for-machine-learning/> [Accessed 4 December 2020].

2. Medium. 2020. *Logistic Regression Model Tuning With Scikit-Learn—Part 1*. [online] Available at: <https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5> [Accessed 4 December 2020].

3. L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, r. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, Alex, r. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and Ga, "Design for machine learning software: experiences from the scikit-learn project," in ECML PKDD Workshop: Languages for Data Mining and Machine Learning, sklearn_api, 2013, pp. 108-122.
4. Scikit-learn.org. 2020. *Sklearn.Discriminant_Analysis.Lineardiscriminantanalysis — Scikit-Learn 0.23.2 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html> [Accessed 4 December 2020].

5. Medium. 2020. *LDA& QDA*. [online] Available at: <https://medium.com/@xkagjaeo05/review-the-machine-learning-2-c2dc0872a23a> [Accessed 4 December 2020].

6. Scikit-learn.org. 2020. *Sklearn.Qda.QDA — Scikit-Learn 0.16.1 Documentation*. [online] Available at: <https://scikit-learn.org/0.16/modules/generated/sklearn.qda.QDA.html> [Accessed 4 December 2020].

7. Becker, M., Binder, M., Bischl, B., Lang, M., Pfisterer, F., Reich, N., Richter, J., Schratz, P. and Sonabend, R., 2020. *3.1 Hyperparameter Tuning | Mlr3 Book*. [online] Mlr3book.mlr-org.com. Available at: <https://mlr3book.mlr-org.com/tuning.html> [Accessed 4 December 2020].

8. Kaggle.com. 2020. *KNN For Classification Using Scikit-Learn*. [online] Available at: <https://www.kaggle.com/amolbhivarkar/knn-for-classification-using-scikit-learn> [Accessed 4 December 2020].

9. Scikit-learn.org. 2020. *Sklearn.Neighbors.Kneighborsclassifier — Scikit-Learn 0.23.2 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html?highlight=kneighborsclassifier#sklearn.neighbors.KNeighborsClassifier> [Accessed 4 December 2020].

10. Medium. 2020. *Boosting And Adaboost Clearly Explained*. [online] Available at: <https://towardsdatascience.com/boosting-and-adaboost-clearly-explained-856e21152d3e> [Accessed 4 December 2020].

11. Ieee.org. 2020. *IEEE Code Of Ethics*. [online] Available at: <https://www.ieee.org/about/corporate/governance/p7-8.html> [Accessed 4 December 2020].

12. Biochemia-medica.com. 2020. [online] Available at: https://www.biochemia-medica.com/assets/images/upload/xml_tif/Simundic_AM_-Bias_in_research.pdf

13. Smlbook.org. 2020. [online] Available at: <http://smlbook.org/book/sml-book-draft-

latest.pdf> [Accessed 4 December 2020]. [Accessed 4 December 2020].

# 9 Appendix

## A Logistic Regression

```
# # Fetch the Songs Training Data Given for Training and Predicting the Logistic Regression Model

# In[4]:

trainingdata = pd.read_csv('training_data.csv', na_values='?', dtype={'ID': str}).dropna().reset_index()
trainingdata.describe()
testproduction = pd.read_csv('songs_to_classify.csv', na_values='?', dtype={'ID': str}).dropna().reset_index()


# # Feed all the inputs to x and given output to Y

# In[5]:


Yraw = trainingdata[['label']]
Xraw =trainingdata.drop(columns=['label'])
Yraw = Yraw.values.ravel()


# # Split the test and train data with 40% of raw data to test data

# In[6]:


xtrain, xtest, ytrain, ytest = train_test_split(Xraw, Yraw, test_size = 0.4, random_state = 1234, stratify=Yraw)


# # Normalise the input in the range(0,1)

# In[7]:
from sklearn.preprocessing import MinMaxScaler
normalisedscaler = MinMaxScaler(feature_range = (0,1))
normalisedscaler.fit(xtrain)
X_train_normalised = normalisedscaler.transform(xtrain)
X_test_normalised = normalisedscaler.transform(xtest)
X_test_normalised
xtestproduction=normalisedscaler.transform(testproduction)
```

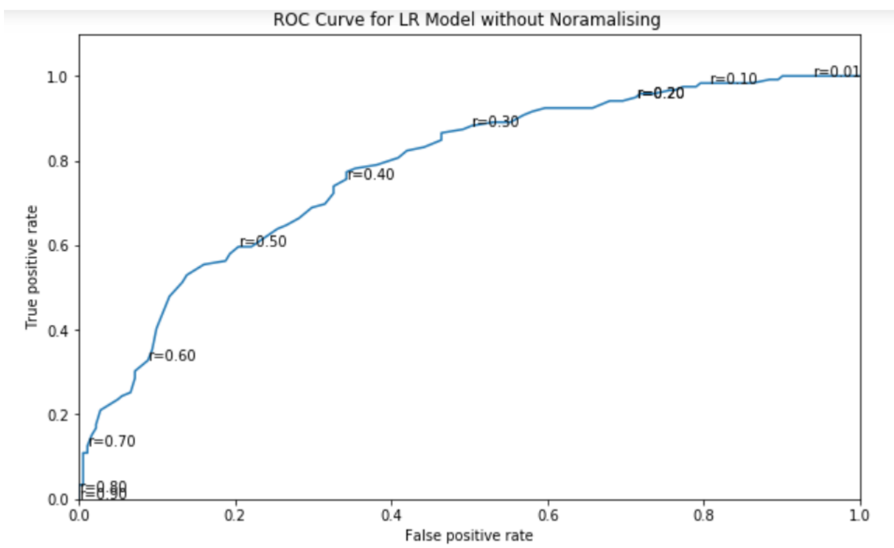
```
# # Fit a Logistic Regression Model to available 60 percent of test data without normalising

# In[8]:
#from sklearn.linear_model import LogisticRegression
lrmodel = LogisticRegression()
lrmodel.fit(xtrain,ytrain)
print(lrmodel)
print(lrmodel.classes_)
# # Obtain the coeffcients for Obtained Logistic Regression Model without normalising
# In[9]:
lrmodel_coefficientsfortheta = lrmodel.coef_
lrmodel_coefficientsfortheta
# # Compute Accuracy of the model By Generating Confusion Matrix without normalising
# In[10]:
y_prediction=lrmodel.predict(xtest)
print('confusion matrix before normalising :\n')
print(pd.crosstab(y_prediction,ytest),'\n')
print(f"Accuracy: {np.mean(y_prediction == ytest):,.3f}")
# # Fit the model ,obtain coefficients and compute accuracy  after normalising input
# In[11]:
lrmodel1 = LogisticRegression()
lrmodel1.fit(X_train_normalised,ytrain)
y_prediction1=lrmodel1.predict(X_test_normalised)
lrmodel_coefficientsfortheta1 = lrmodel1.coef_
print(lrmodel_coefficientsfortheta1)
#y_prediction1=lrmodel1.predict(X_test)
print('confusion matrix:\n')
print(pd.crosstab(y_prediction1,ytest),'\n')
print(f"Accuracy: {np.mean(y_prediction1 == ytest):,.3f}")
err_rate=np.mean(y_prediction1!=ytest)
err_rate
# In[12]:
y_predproduction=lrmodel1.predict(xtestproduction)
y_productionjoin = ''.join([str(elem) for elem in list(y_predproduction)])
y_productionjoin
```

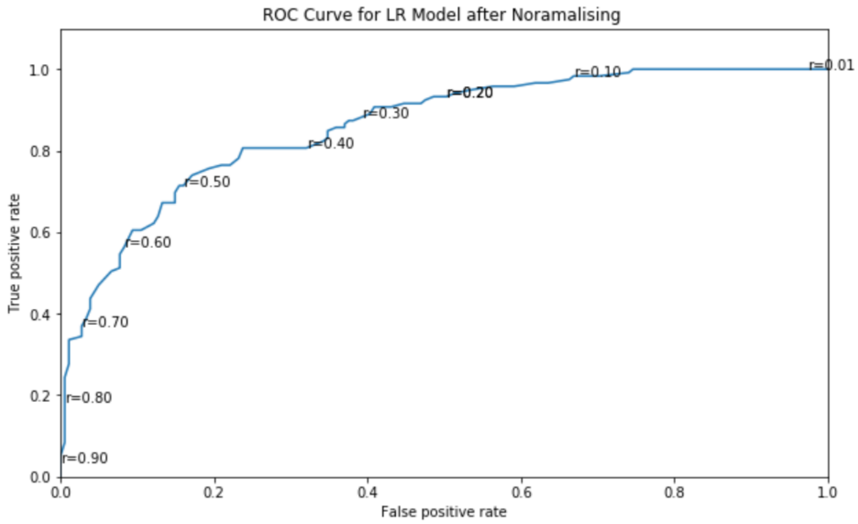
```
#Cross Vaidaltion After  Normalising Inputs
n_fold = 10
models = []
models.append(skl_lm.LogisticRegression(solver='liblinear'))
#models.append(skl_da.LinearDiscriminantAnalysis())
#models.append(skl_da.QuadraticDiscriminantAnalysis())
#models.append(skl_nb.KNeighborsClassifier(n_neighbors=2))
misclassification = np.zeros((n_fold, len(models)))
cv = skl_ms.KFold(n_splits=n_fold, random_state=1, shuffle=True)
for i, (train_index, val_index) in enumerate(cv.split(X)):


    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]
    X_ntrain = normalisedscaler.transform( X_train)
    X_nval=normalisedscaler.transform( X_val)
    for m in range(np.shape(models)[0]):


            # try different models
        model = models[m]
        model.fit(X_ntrain, y_train)
        prediction = model.predict(X_nval)
        misclassification[i, m] = np.mean(prediction != y_val)
plt.boxplot(misclassification)
plt.title('cross validation error for different methods')
plt.xticks(np.arange(4)+1, ('logReg', 'LDA', 'QDA', 'kNN'))
plt.ylabel('validation error')
plt.show()
```

## Results not included in Main Report

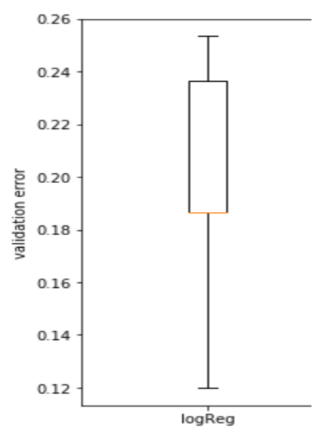
| Coefficients for LR f(x)without normalizing input | | Coefficients for LR f(x) after normalizing input |
|---|---|---|
| [[ 5.03631927e-04, 2.28199520e-02, -8.81999404e-03, 4.55927895e-07, -1.64662161e-02, 3.66111237e-03, -2.71103431e-02, -4.14511664e-03, -2.50911971e-01, 7.23757812e-03, -8.40551509e-03, -1.24701809e-02, -1.78442892e-02, -5.74870451e-03]]) | | 0.8141249 2.22185634 -1.64023375 1.3633774 -1.49369883 -0.17585658 0.25678163 -0.56222435 -1.24711821 0.24294319 -4.16192273 0.16199766 -0.60122711 0.20204136]] |

## ROC Curves before and after Normalizing






**Cross Validation Error After Normalizing**



**B Discriminant Analysis.**

**LDA and QDA**

```
# #### Import data

# In[26]:


train = pd.read_csv('/Data/training_data.csv')
test = pd.read_csv('/Data/songs_to_classify.csv')


# ##### Information regading the data

# In[27]:


train.info()


# In[28]:


print(train.shape)


# ##### Split the train data into features and target

# In[29]:


y = train.label
X = train.drop('label', axis = 1)
print('Data shape, features: ',X.shape,"Target:", y.shape)


# ###  Discriminant analysis

# #### LDA

# In[30]:


import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb
import sklearn.preprocessing as skl_pre
from sklearn.model_selection import RepeatedStratifiedKFold
cv = KFold(n_splits=3, random_state=1000, shuffle=True)


# In[31]:

```

```python
# In[31]:


model = skl_da.LinearDiscriminantAnalysis() #Instantiate the model
params = {
    'n_components': [0,1,2,3,4,5, 6, 8, 10, 12, 15, 17, 20],
    'tol': [0.0000000001, 0.000000001,0.00000001, 0.0000001,0.000001,0.00001,0.0001,0.001,0.01,0.1,1, 10, 100, 1000, 10000 ]
} #define parameters for optimization

model_search = GridSearchCV(model,
                         param_grid=params,cv = cv,
                         return_train_score=False)#Conduct a grid search to determine the optimal parameters
model_search.fit(X,y)


# In[32]:


print(list(model_search.best_params_.keys()))
print(list(model_search.best_params_.values()))


# In[33]:


model = skl_da.LinearDiscriminantAnalysis(n_components=list(model_search.best_params_.values())[0], tol = list(model_search.best_params_.values())[1])
#Predict
predicted = cross_val_predict(model, X, y, cv=cv)
# report performance
model_s = round(np.mean(predicted == y)*100,3)
model_s_a = roc_auc_score(y, predicted)
# Confusion matrix
print("Confusion matrix:")
print(pd.crosstab(y, predicted),'\n')
# Accuracy
print(f"Accuracy: {model_s }")


# #### QDA

# In[34]:


from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
clf = QuadraticDiscriminantAnalysis() #Instantiate the model
params = {
    'reg_param': [0,1,2,3,4,5,6,7,8,9,10,11,12,13],
    'tol': [0.0000001,0.000001,0.00001,0.0001,0.001,0.01,0.1,1, 10, 100, 1000, 10000 ]
} #define parameters
```

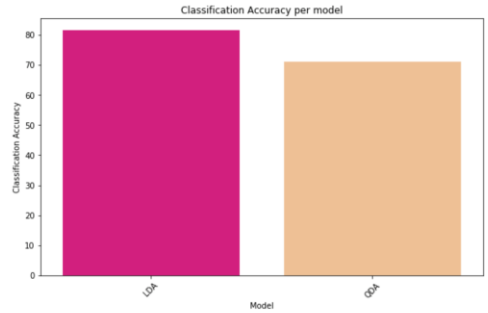
# parameter selected in  the grid search (LDA)

```
['n_components', 'tol']
[0, 1e-10]
```

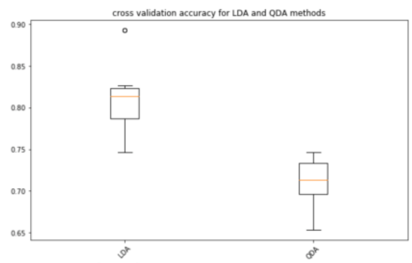
# Parameter selection in the grid search(QDA)

```
Optimal Parameters
['reg_param', 'tol']
[0, 1e-07]
```

#Classification accuracy of LDA and QDA




#cross validation accuracy for LDA and QDA

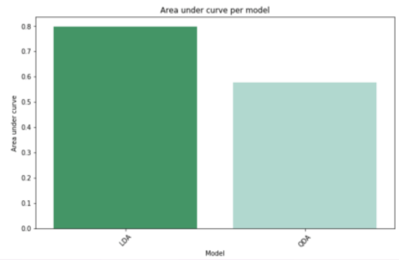



```
params = {
    'reg_param': [1,2,3,4,5,6,7,8,9,10,11,12,13],
    'tol': [0.0000001,0.000001,0.00001,0.0001,0.001,0.01,0.1,1, 10, 100, 1000, 10000 ]
} #define parameters

clf_search = GridSearchCV(clf,
                          param_grid=params,
                          cv=3,
                          return_train_score=False) #Conduct a grid search to determine the optimal parameters
clf_search.fit(X,y)


# In[58]:


print('Optimal Parameters ')
print(list(clf_search.best_params_.keys()))
print(list(clf_search.best_params_.values()))


# In[59]:


clf = QuadraticDiscriminantAnalysis(reg_param=list(clf_search.best_params_.values())[0], tol = list(clf_search.best_params_.values())[1])
#Predict
predicted = cross_val_predict(clf, X, y, cv=cv)
# report performance
clf_s = round(np.mean(predicted == y)*100,3)
clf_s_a = roc_auc_score(y, predicted)
# Confusion matrix
print("Confusion matrix:")
print(pd.crosstab(y, predicted),'\n')
# Accuracy
print(f"Accuracy: {clf_s }")
```

```
Optimal Parameters
['reg_param', 'tol']
[1, 1e-07]
 Confusion matrix:
 col_0    0    1
 label
 0      180  118
 1      207  245

 Accuracy: 56.667
```

#Above refers to the selection of specific parameter  in QDA.

#Area Under the curve.




```
clf_search = GridSearchCV(clf,
                          param_grid=params,
                          cv=3,
                          return_train_score=False) #Conduct a grid search to determine the optimal parameters
clf_search.fit(X,y)


# In[35]:


print('Optimal Parameters ')
print(list(clf_search.best_params_.keys()))
print(list(clf_search.best_params_.values()))


# In[36]:


clf = QuadraticDiscriminantAnalysis(reg_param=list(clf_search.best_params_.values())[0], tol = list(clf_search.best_params_.values())[1])
#Predict
predicted = cross_val_predict(clf, X, y, cv=cv)
# report performance
clf_s = round(np.mean(predicted == y)*100,3)
clf_s_a = roc_auc_score(y, predicted)
# Confusion matrix
print("Confusion matrix:")
print(pd.crosstab(y, predicted),'\n')
# Accuracy
print(f"Accuracy: {clf_s }")


# # Cross validation for  the models and compute the classification accuracy for each of the models

# # Add models defined above using the optimized parameters

# In[37]:


import sklearn.model_selection as skl_ms
n_fold = 10
models = []
"""LDA using parameter optimization and cross-validation"""
models.append(skl_da.LinearDiscriminantAnalysis(n_components=list(model_search.best_params_.values())[0], tol = list(model_search.best_params_.values())[1]))
"""QDA using parameter optimization and cross-validation"""
models.append(skl_da.QuadraticDiscriminantAnalysis(reg_param=list(clf_search.best_params_.values())[0], tol = list(clf_search.best_params_.values())[1]))#Using optimal
    parameters


# ######  Conduct cross validation for each of the models defined above
```

#Accuracy of the selected parameter(LDA)

```
Confusion matrix:
col_0     0    1
label
0       207   91
1        52  400

Accuracy: 80.933
```

#Accuracy of the selected parameter(QDA)

```
Confusion matrix:
col_0     0    1
label
0       246   52
1       156  296

Accuracy: 72.267
```

```
# In[38]:


"""Cross-validation"""
classification_acc = np.zeros((n_fold, len(models)))
rocss = np.zeros((n_fold, len(models)))
cv = skl_ms.KFold(n_splits=n_fold, random_state=1, shuffle=True)
for i, (train_index, val_index) in enumerate(cv.split(X)):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]
    for m in range(np.shape(models)[0]): # try different models
        model = models[m]
        model.fit(X_train, y_train)
        prediction = model.predict(X_val)
        classification_acc[i, m] = np.mean(prediction == y_val)
        rocss[i, m] = roc_auc_score(y_val, prediction)
plt.boxplot(classification_acc)
plt.title('cross validation accuracy for different methods')
plt.xticks(np.arange(2)+1, ( 'LDA', 'QDA'))
plt.ylabel('validation error')
plt.xticks(rotation=45);
plt.show()


# ### Compare performance of the models

# In[39]:


#Compute mean classification accuracy and AUC
perf_accuracy = pd.DataFrame(classification_acc)
perf = pd.DataFrame(rocss)
perf_accuracy.columns = ['LDA', 'QDA']
perf.columns = ['LDA', 'QDA']
#LDA
lda_s_a = perf.LDA.mean()
lda_s = round(perf_accuracy.LDA.mean()*100,3)#Mean cross validation score
#QDA performance
qda_s_a = perf.QDA.mean()
qda_s = round(perf_accuracy.QDA.mean()*100,3)




models = ['LDA', 'QDA']
scoress = [ lda_s, qda_s]
aucss = [lda_s_a, qda_s_a]
perf = pd.DataFrame()
perf['Model'] = models
```

```python
perf['Classification Accuracy'] = scoress
perf['Area under curve'] = aucss
perf.sort_values(by='Area under curve', ascending=False)


# In[40]:


import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure(figsize = (10, 6))
ax = sns.barplot(x="Model", y="Classification Accuracy", data=perf,palette="Accent_r");
plt.xlabel("Model")
plt.ylabel("Classification Accuracy")
plt.title("Classification Accuracy per model")
plt.xticks(rotation=45);


# In[41]:


fig = plt.figure(figsize = (10, 6))
ax = sns.barplot(x="Model", y="Area under curve", data=perf, palette = 'vlag');#BuGn_r
plt.xlabel("Model")
plt.ylabel("Area under curve")
plt.title("Area under curve per model")
plt.xticks(rotation=45);


# ### Predicting class of songs

# In[42]:


mod = skl_da.LinearDiscriminantAnalysis(n_components=list(model_search.best_params_.values())[0], tol = list(model_search.best_params_.values())[1])
mod.fit(X,y)
preds = mod.predict(test)


# In[43]:


out_preds = ''.join([str(elem) for elem in list(preds)])
out_preds
```

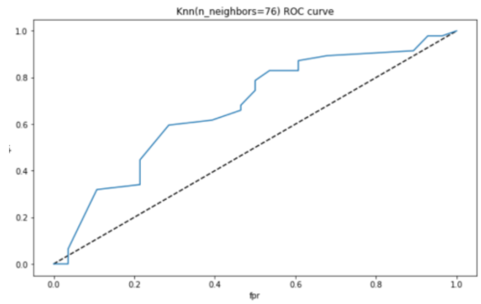
## C KNN

```python
traindata = pd.read_csv('data/training_data.csv')
traindata.describe()


#training error varies with different values of  k .Consider all data in training_data.csv as training data.
np.random.seed(1)
X=traindata.drop(columns=['label'])
y=traindata['label']
model=skl_nb.KNeighborsClassifier(n_neighbors=2)
model.fit(X,y)
prediction=model.predict(X)
accuracy=np.mean(prediction==y)
err_rate=np.mean(prediction!=y)
print('Accuracy for KNN is :'+str(accuracy))
print('Error Rate for KNN is :'+str(err_rate))


# Split the data randomly into a training and validation set and see the validation error rate wrt diff value of k


#Consider value of k between 0-100 and see the best performing value of k.

np.random.seed(1)
N=len(X)
M=np.ceil(0.70*N).astype(int)
idx=np.random.permutation(N)
X_train,X_val =X.iloc[idx[M:]],X.iloc[idx[:M]]
y_train,y_val =y.iloc[idx[M:]],y.iloc[idx[:M]]
misclassification=[]
K=np.arange(1,100)
for k in K:
    model=skl_nb.KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train,y_train)
    prediction=model.predict(X_val)
    misclassification.append(np.mean(prediction!=y_val))
plt.plot(K,misclassification)
plt.title('Validation error for KNN')
plt.xlabel('Number of neighbors k')
plt.ylabel('Validation Error')
plt.show()
#min error rate for k=21 is 0.3657
```

```python
misclassification=np.zeros(len(K))

for train_index,val_index in cv.split(X):
    X_train, X_val=X.iloc[train_index] ,X.iloc[val_index]
    y_train, y_val=y.iloc[train_index] ,y.iloc[val_index]

    for j,k in enumerate(K):
        model=skl_nb.KNeighborsClassifier(n_neighbors=k)
        model.fit(X_train,y_train)
        prediction=model.predict(X_val)
        misclassification[j] += np.mean(prediction!=y_val)


misclassification /= n_fold
plt.plot(K,misclassification)
plt.title('Cross Validation error for KNN')
plt.xlabel('Number of neighbors k')
plt.ylabel('Validation Error')
plt.show()


model=skl_nb.KNeighborsClassifier(n_neighbors=76)
model.fit(X_train,y_train)
model.score(X_val,y_val)


# In[57]:


from sklearn.metrics import confusion_matrix
prediction = model.predict(X_val)
pd.crosstab(y_val, prediction, rownames=['True'], colnames=['Predicted'], margins=True)


# In[ ]:
```

```python
#Classification report for model


# In[58]:


from sklearn.metrics import classification_report
print(classification_report(y_val,prediction))


# In[ ]:


#ROC for the selected model


# In[59]:


pred_probability = model.predict_proba(X_val)[:,1]
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_val, pred_probability)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=76) ROC curve')
plt.show()


# In[ ]:

#Area under ROC curve for selected model

# In[60]:

from sklearn.metrics import roc_auc_score
roc_auc_score(y_val,pred_probability)
```

### Area under the curve for k=76




## D)Boosting

```
"""Loading training data file to train models."""

train_df = pd.read_csv('training_data.csv')
train_df.info()

"""Read first ten rows of data. """

train_df.head(10)

"""Read random data from traing data frame to """

random_indexes = np.random.choice(len(train_df), size=10, replace=False)
train_df.iloc[random_indexes]

"""Findout how songs have been performed live. it can be findout by liveness > 0.8"""

train_df[train_df['liveness'] > 0.8]

"""Every column has numeric data, before we proceed for further mathematical operation check for any NA values."""

train_df[train_df.isna().any(axis=1)]

"""We don't have any unwanted data which gives computational errors.

Now read test data to verify we have the same columns and data type except label before we train our models on training data.
"""

classify_songs_df = pd.read_csv('songs_to_classify.csv')
classify_songs_df.info()

"""We have the same columns and data type in both training and test data. We can start training our models based on training data.

As supervised learning states we train our models on labeled data. Here our desired output is either 0(DISLIKE) or 1(LIKE).
For training we remove label column for computaion.
"""

X_train_df = train_df.copy().drop(columns=['label'])
y_train_df = train_df['label']
```

```
"""Train AdaBoost boost model on training data to predict on test data

Train with default values
"""

model_boost = AdaBoostClassifier()
model_boost.fit(X=X_train_df, y=y_train_df)
print('test error rate for ada boost classifier is %.3f' %np.mean(model_boost.predict(X_train_df) != y_train_df))

model_boost_samme_alg = AdaBoostClassifier(algorithm='SAMME')
model_boost_samme_alg.fit(X=X_train_df, y=y_train_df)
print('test error rate for ada boost classifier for SAMME algoritham is %.3f' %np.mean(model_boost_samme_alg.predict(X_train_df) != y_train_df))

model_boost_70_estimators = AdaBoostClassifier(n_estimators=70)
model_boost_70_estimators.fit(X=X_train_df, y=y_train_df)
print('test error rate for ada boost classifier for 70 estimators is %.3f' %np.mean(model_boost_70_estimators.predict(X_train_df) != y_train_df))

model_boost_100_estimators = AdaBoostClassifier(n_estimators=100)
model_boost_100_estimators.fit(X=X_train_df, y=y_train_df)
print('test error rate for ada boost classifier for 100 estimators is %.3f' %np.mean(model_boost_100_estimators.predict(X_train_df) != y_train_df))

X = train_df.copy().drop(columns=['label'])
y = train_df['label']

svc=SVC(probability=True, kernel='linear')
model_boost_svc =AdaBoostClassifier(n_estimators=50, base_estimator=svc,learning_rate=1)
# Train Adaboost Classifer
model_boost_svc.fit(X_train_df, y_train_df)

#Predict the response for test dataset
print('test error rate for ada boost classifier for svc is %.3f' %np.mean(model_boost_svc.predict(X_train_df) != y_train_df))
```

```
"""Cross validation of models using k fold to select the best trained model to run on test data."""
n_fold = 10
models = []
models.append(model_boost)
models.append(model_boost_samme_alg)
models.append(model_boost_70_estimators)
models.append(model_boost_100_estimators)
misclassification = np.zeros((n_fold, len(models)))
cv = skl_ms.KFold(n_splits=n_fold, random_state=1, shuffle=True)

for i, (train_index, val_index) in enumerate(cv.split(X)):
  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]

  for m in range(np.shape(models)[0]):
    model = models[m]
    model.fit(X_train, y_train)
    prediction = model.predict(X_val)
    misclassification[i, m] = np.mean(prediction != y_val)

plt.boxplot(misclassification)
plt.title('cross validation error for different trained ada boost models')
plt.xticks(np.arange(len(models))+1, ('Default', 'SAMME','70Est', '100Est'))
plt.ylabel('Validation error')
plt.show()

Hyperparameter tuning

"""Hyperparameter tuning of AdaBoost model"""

crossvalidation=skl_ms.KFold(n_splits=10,shuffle=True,random_state=1)
ada=AdaBoostClassifier()
ada_grid={'n_estimators':[70,100,500,1000],'learning_rate':[.001,0.01,.1]}
search=GridSearchCV(estimator=ada,param_grid=ada_grid,scoring='accuracy',n_jobs=1,cv=crossvalidation)

search.fit(X=X_train_df,y=y_train_df)
```

```
"""Find best parameter values of AdaBoost Mode."""

search.best_params_

model_boost_1000_estimators = AdaBoostClassifier(n_estimators=1000, learning_rate=0.1)
model_boost_1000_estimators.fit(X=X_train_df, y=y_train_df)
print('test error rate for ada boost classifier for 1000 estimators is %.3f' %np.mean(model_boost_1000_estimators.predict(X_train_df) != y_train_df))

score=np.mean(cross_val_score(ada,X_train_df,y_train_df,scoring='accuracy',cv=crossvalidation,n_jobs=1))
score
```

```
"""Confusion matrix for above trained models """
prediction_def = model_boost.predict(X)
print('Confusion matrix \n')
print(pd.crosstab(y, prediction_def))
prediction_hyp = model_boost_1000_estimators.predict(X)
print('Confusion matrix \n')
print(pd.crosstab(y, prediction_hyp))
prediction_70_est = model_boost_70_estimators.predict(X)
print('Confusion matrix \n')
print(pd.crosstab(y, prediction_70_est))
prediction_100_est = model_boost_100_estimators.predict(X)
print('Confusion matrix \n')
print(pd.crosstab(y, prediction_100_est))
prediction_dep = model_boost_samme_alg.predict(X)
print('Confusion matrix \n')
print(pd.crosstab(y, prediction_dep))
true_postive_rate = []
false_positive_rate = []
positive_class = 1
negative_class = 0
P = np.sum(y == positive_class)
N = np.sum(y == negative_class)
threshold = np.linspace(0.00, 1, 101)
models = []
models.append(model_boost_1000_estimators)
for m in range(len(models)):
  model = models[m]
  model.fit(X, y)
  predict_prob = model.predict_proba(X)
  positive_class_index = np.argwhere(model.classes_ == positive_class).squeeze()
  for r in threshold:
    prediction = np.where(predict_prob[:, positive_class_index] > r,positive_class,negative_class)
    FP = np.sum((prediction==positive_class)&(y==negative_class))
    TP = np.sum((prediction==positive_class)&(y==positive_class))
    false_positive_rate.append(FP/N)
    true_postive_rate.append(TP/P)
  plt.plot(false_positive_rate, true_postive_rate)
  for idx in [ 1, 10,20,20,30,40,50,60,70,80,90, ]:
    plt.text(false_positive_rate[idx], true_postive_rate[idx], f"r={threshold[idx]:.2f}")
    plt.xlim([0,1])
    plt.ylim([0,1.1]);
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title("ROC Curve for Ada Boost Model with 1000 estimators")

"""Calculate AUC (Area under Curve) for all above trained models"""

y_prob = model_boost.predict_proba(X)[:,1]
auc = round(roc_auc_score(y, y_prob)*100,2)
auc

y_prob_100 = model_boost_100_estimators.predict_proba(X)[:,1]
auc_100 = round(roc_auc_score(y, y_prob_100)*100,2)
auc_100

y_prob_70 = model_boost_70_estimators.predict_proba(X)[:,1]
auc_70 = round(roc_auc_score(y, y_prob_70)*100,2)
auc_70

y_prob_1000 = model_boost_1000_estimators.predict_proba(X)[:,1]
auc_1000 = round(roc_auc_score(y, y_prob_1000)*100,2)
auc_1000

y_prob_alg = model_boost_samme_alg.predict_proba(X)[:,1]
auc_alg = round(roc_auc_score(y, y_prob_alg)*100,2)
auc_alg

"""After calculating confusion matrix, ROC and AUC for above training models,
opting Adaboost with 1000 estimators and 0.1 learning rate to run on production data for good prediction results."""

X_test = classify_songs_df.copy();
prediction = model_boost_1000_estimators.predict(X=X_test)
out_preds = ''.join([str(elem) for elem in list(prediction)])
out_preds
```