Group S653 (4 members):Mini Project Report

Anonymous Author(s)
Affiliation
Address
email

Abstract

This document contains various classification methods used in solving the Mini Project problem for the course Statistical Machine Learning, 1RT700. The goal is to classify a test set of 200 songs and predict which songs Andreas Lindholm is going to like. In this report, we analyze four different classification methods with each member implementing one method that was taught in the course and finally implement the best method in production.

1 Introduction

The idea of the project is to train a classification model using the training dataset containing 750 songs alongwith their features extracted from the spotify web-API and predict the songs that Anreas would like. We achieve this by tuning the parameters and implementing the best model in practice.

2 Classification Methods

This section contains the analysis and testing of four different classification methods. The cross-validation method used for all the four different methods is the K-10 fold cross validation method.

2.1 K - Nearest Neighbours Classifier

For this project we have, among other methods, tried the K-NN approach to classify the songs as either liked or disliked. It's an abbreviation for the k-nearest neighbors and works in the following way: given a new unlabeled data point that we want to classify, we observe how the k-nearest neighbors, for k being a positive integer, have been classified and based on a majority vote among them we would classify this new data point. Of course, the majority vote will result in two probabilities, one for like and the other dislike. We choose ourselves a decision boundary, i.e. if the probability is equal to or greater than that boundary, then the song is classified as like, otherwise dislike. K-NN is a non-parametric method and is therefore different from the parametric methods, that have a fixed number of parameters which are estimated using the training data at hand. The more the training data, the better the estimations will be, but lots of training data is not always available, why choosing non-parametric methods may be a wise idea. For this kind of projects, where the task is to classify songs, K-NN is generally considered as being a good method to choose [4], since there is a high probability that you like a song close to what you earlier have liked.

We have for different values of k, plotted the misclassification error versus k. For a plot for k = 1 to 30 using all the features as input see figure 2.

But sometimes one does not need to include all the features. For example, by plotting the different features against each other and observing the scatter-plot matrix or by studying the co-variance matrix between two features, we can see that some features does indeed have a relationship. Take the scatter-plot matrix between valence and danceability as an example, featured in figure 1.

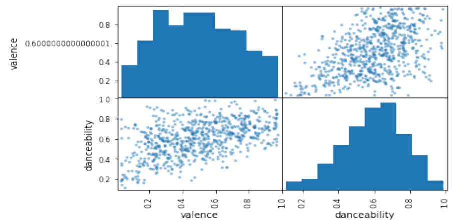

Figure 1: Correlation matrix

This plot is obtained by the following python code: $pd.plotting.scatter\text{-}matrix(a)$ where a is a vector containing the features valence and danceability from the training set. pd is a reference to the python library pandas, and plotting.scatter-matrix is simply making a scatter matrix plot of some input data, in this case the vector a.
From this plot we can read that the more a song is danceable, the higher the valence is of that song. Thinking intuitively, this is correct because hos probable is it that someone would want to dance to a depressive song? Therefore, by excluding the valence from the list of features, the result shall not change much, if any at all. By similar arguments, one can exclude even more features from the training set. A second plot is obtained, this time including only the features: acousticness, energy, instrumentalness, key, speechiness. For the resulting plot, see figure 3.

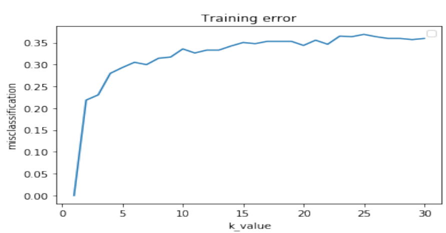

Figure 2: Missclassification error

Figure 3: Missclassification error: selected features

A slightly better result emerges, though not significantly better. The code for this plot and the first one is included in appendix. For both of these missclassification error plots, the missclassification is the error rate for 100 validation data points and the size of the training data set were 750-100 = 650 data points. Hence for the second plot, we can see that for k = 10 we have an error rate of below to 20%, meaning that more that 80% of the songs were classified correctly. The decision boundary is .5 for both cases. Varying this boundary between .4 to .6 could give different results, although not much different that could make an impact.
As a final word it's worth to mention that, although the K-NN method does indeed produce a good result in this case, we choose to use another method in production.

2.2 Discriminant analysis

Within the confines of this course two machine learning methods using discriminant analysis (DA) are discussed: Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis QDA). Although this project will focus on LDA, comparisons to QDA will be made. DA methods don't work by fitting training data to a predefined model, but instead look at each data point and calculate the probability of it falling in a certain class. The method assumes that the training data follows the Gaussian distribution and uses the mean and variance values of this data. Since there are multiple input factors, we're dealing with an average mean ($\hat{\mu}$) and a co-variance matrix ($\hat{\mathbf{\Sigma}}$). In LDA this mean is assumed to differ between classes, while the co-variance matrix is constant.

$$p(k|\mathbf{x}_*) = \frac{\hat{\pi}_k \mathcal{N}\left(\mathbf{x}_*|\hat{\mu}_k, \hat{\mathbf{\Sigma}}\right)}{\sum_{j=1}^{K} \hat{\pi}_j \mathcal{N}\left(\mathbf{x}_*|\hat{\mu}_j, \hat{\mathbf{\Sigma}}\right)} \tag{1}$$

Equation 1 calculates the probability of a new input point $\mathbf{x}_*$ belonging to class k. The numerator in this equation features the occurrence rate of class k in the data ($\hat{\pi}_k$) multiplied by the chance of this new input point existing in a Gaussian/normal distribution with previously mentioned average mean for class k and co-variance matrix. The denominator meanwhile sums the nominator calculations for all classes k. Even though we might not know the actual distribution of data y, we can still learn the parameters in the equation from the training data. We can make a prediction for a certain data point $\hat{y}_*$ by inserting the input values of this data point ($\mathbf{x}_*$) and choose class k which has the highest probability of occurring.

The results for LDA and QDA were created using the LinearDiscriminantAnalysis and QuadraticDiscriminantAnalysis functions from the sklearn.discriminant_analysis python package. These functions work by creating linear decision boundaries in the domain, which the boundary is located in the input space where the different class predictions for the data points meet.

Figure 5 shows the first attempt at classification and reveals that LDA appears to outperform QDA in this scenario. The average accuracy across all 10 folds in the K-fold validation is about 0.81 for the LDA method and 0.76 for QDA. We're comparing that with an 'always true' prediction, where the outcome of the model is positive for every data point, and which has an accuracy of about 0.6. One way to improve the performance of methods is to focus on a specific number of input factors, instead of including all of them. Figure 4 shows the results of attempting to predict the outcome (like or not like) using only one input factor. The strongest input factors appear to be *accousticness*, *energy*, *loudness* and *speechiness*. Including only those factors in the model slightly elevates the performance of the models (see figure 6 and table 1).

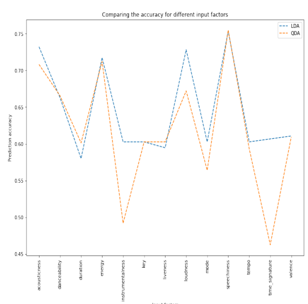

Figure 4: Classification accuracy for models with only one input factor at a time.

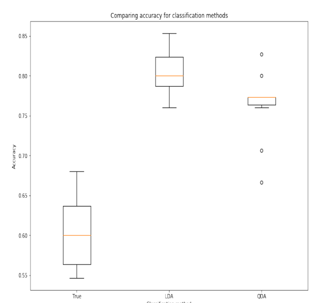

Figure 5: Classification accuracy for methods with all input factors included in model.

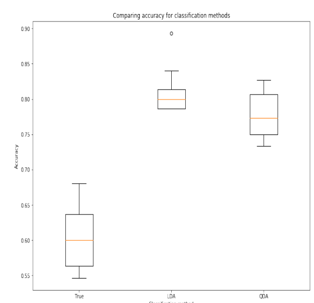

Figure 6: Classification accuracy with four selected input factors included in model.

Another way of optimizing the accuracy using the input factors, is by using an algorithm to compare all the different combinations of input factors and selecting the one with the highest accuracy. The code for this algorithm is featured in the appendices and was used on LDA by comparing all possible combinations of three input factors. Including any more factors took too much time for the algorithm. This resulted in *accousticness*, *loudness* and *speechiness* being sleected, with an accuracy of 0.82. This is the highest accuracy we found using any of the DA methods.

Table1: Classification accuracy of methods with either all input factors, or only *accousticness*, *energy*, *loudness* and *speechiness* included.

Method	Accuracy all inputs	Accuracy selected inputs
Always true	0.6027	0.6027
LDA	0.8053	0.8107
QDA	0.7627	0.7773

2.3 Logistic Regression

In the binary classification problem we use logistic regression. We want probability to determine belonging to each class i.e if probability p > 0.5 then 'like'. Logistic regression is a parametric model and we are trying to learn a model for probabilities $p(y=1|x)$ and $p(y=0|x)$ where:

$$p(y=1|x) = \frac{\mathrm{e}^{\beta^T x}}{\mathrm{e}^{\beta^T x}+1} \tag{2}$$

by maximum likelihood approach to determine β parameters from the training data.We do this by fitting a probability Sigmoid function (2) which has a maximum likelihood to fit the observed data points. Graph of this Sigmoid curve represent dependent variables on the horizontal axis and probability $\in [0,1]$ on the vertical axis. Likelihood of all data points is equal to the product of observed and unobserved data points probabilities. The algorithm chooses a model for the Sigmoid curve that maximizes the log likelihood of all observed data.

The training data was split 80% and 20% between training and test data i.e (600 and 150). 'Mode' co-variate is changed to categorical variable. Logistic Regression function was used to chose the appropriate Sigmoid curve together with the β parameters.Firstly, all the variables were used to describe the model. Resulting in accuracy ($\frac{TN+TP}{n}$) of 0.6467 and a Receiver Operating Characteristics (ROC) curve as shown in figure 7. We reduce the model to include the most correlated covariates:acousticness,danceability,energy,loudness,speechiness.Based on the confusion matrix, this reduction in model complexity results in accuracy=0.78 and the ROC curve as in figure 8. Furthermore dropping danceability results in model accuracy=0.8 and ROC as in figure 9, where greater area indicates greater probability of detection.

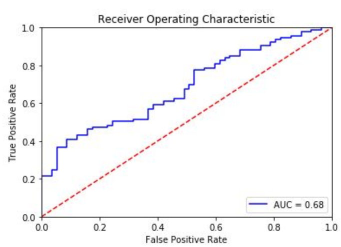

Figure 7: model 1

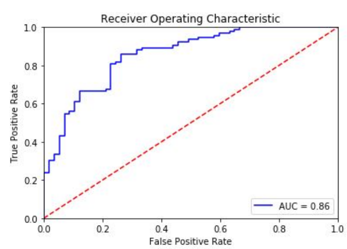

Figure 8: model 2

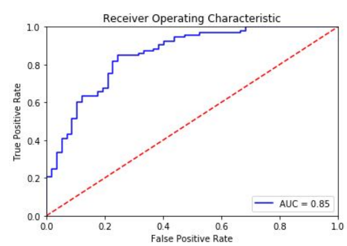

Figure 9: model 3

In order to tune our model we optimize the regularization hyper-parameter C, that is selecting the values of the model that maximize the accuracy, by means of grid search[2]. Where the GridSearchCV function uses C and penalty parameters combinations on a grid, cross-validates the performance of each model and suggests the one obtaining the best results. The combinations can be found in table 2.

Table 2: All combinations of model, penalty parameters and C

model	penalty	C
model 1	l1	2.7825594022071245
model 2	l2	59.94842503189409
model 3	l1	2.7825594022071245

The 10-fold cross validation on the tuned model was carried out. The method was carried out manually, where first we randomize our data points, divide our training data into 10 groups and treat 1 as test data and the remaining as a training data. The process is repeated until each fold has been used once as a test data. Test error is calculated per each fold and average over the 10 folds is obtained as a more accurate estimate of E_{new}. The resulting error rates are as follows: model 1= 0.2, model 2=0.16, model 3=0.16. To help us decide between two last models we use cross-validation score function which again uses 10-fold validation to estimate correct predictions. It allows us to select model2 with accuracy score of 0.8107 winning over model 1 with 0.8035 and model 3 with 0.8085.

2.4 Random Forest Classifier with ADA Boost

Random Forest Classifier comprises of various individual algorithms which combined is termed as an 'ensemble' algorithm. These individual algorithms are nothing but decision trees which in turn are probability spaces borne out of conditional probabilities of data labels given by repetitive partition of the given dataset. With every recursive training of the model the algorithm will calculate the optimum partition of the dataset that would lead to minimum generalization error. Individual trees are weak learners, but Random Forest Classification algorithm takes a majority vote out of many such trees to finally predict the class of the test set. Following is a Validation Score plot of Number of trees Vs Accuracy using Random Forest Classifier.

Boosting is a reinforcement method based on the idea of improving the accuracy by combining many relatively weak learners or classifiers. In the case of Adaptive Boosting (AdaBoost) weak models are attached sequentially and trained using the weighted training data. This goes on till the user-defined number of weak learners have been reached or no refinement on the accuracy can be further made. AdaBoost predicts by calculating the weighted average of all the said weak classifiers. Below is the validation curve plot of Number of estimators Vs Accuracy Score for Cross Validation for 5 splits.

In this section we choose to boost Random Forest Classifier because AdaBoosting works best with non linear models such as Decision Trees which are weak learners. Another reason to choose decision trees is because they are fast learners - the training process is reasonably fast. This is nice as we test in the range of 1 to 600 estimators to get the best possible parameters.

For parameter tuning with AdaBoosting we reduce the learning rate and increase the amount of estimators along with increasing the number of trees for the estimator - Random Forest Classifier.

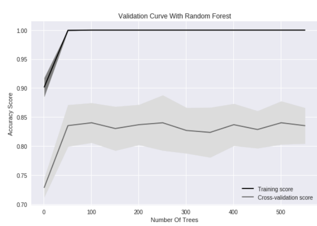

Figure 10: Random Forest Classifier Alone

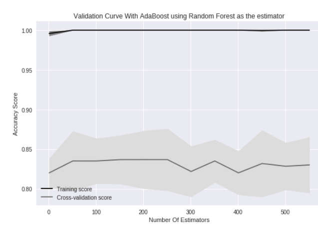

Figure 11: First execution of the model

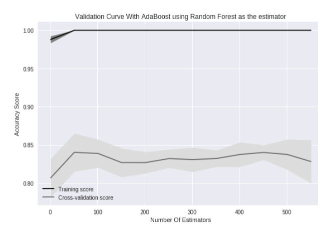

Figure 12: Plot after parameter tuning

As is visible from the accuracy plots, the error for the final tuned model during cross validation which came out to be 0.14 which is by far the best amongst all the models used so far. An accuracy of 81% was achieved in the leaderboard after implementing the solutions of this model.

3 Model Application/Conclusion

We based out optimal model choice on the comparison of 10-fold cross validation obtained from the cross-validation score function. Only the best tuned models from each family were compared and the scores are illustrated in table 3. The Random Forest Clasiffier with ADA Boosting was implemented on the available test data and achived a performence of 0.81% correct predictions on the leader board. Therefore we can conclude that Random Forest with ADA Boosting is the most effective way in predicting music preferences taking into account features extracted from songs.

Table 3: Accuracy results of the best tuned version of the models.

Model	Accuracy
K-Nearest Neighbours	0.608
Linear Discriminant Analysis	0.82
Logistic Regression	0.811
Random Forest Classifier with ADA Boost	0.860

4 Reflection

Every machine learning engineer is a consumer of some service and has therefore a dual outlook on ethics of data collection

We would like to discuss the scenario where the music preference data of the user has been used to suggest digital advertisements in the form of popups or online advertisements. More specifically for users with extreme cases of emotional imbalance such as depression who might listen to music of such liking or mood. Song streaming companies can sell such information to pharmaceutical companies who in turn can give out targeted online advertisements to this specific group of unaware users. Within the bracket of said users, the cause of this event can range from redundant expenditures to irrevocable addictions or worse, medication error fatalities. On the 'upside', if the music preference data of the aforementioned section of users is given to health authorities or public health organizations, there is a possibility wherein they contact the family/guardian of the said user in a professional manner and provide the necessary attention if required. Although, it is very likely the user will be uncomfortable knowing the data has been used for this particular reason, it is subjective to agree that it has been done so for the benefit of the user and that user alone.

On the other hand, there can be less severe yet still unwarranted outcomes of such data usage. Suggesting google ads for sports shoes/equipments for users who build playlists for working out or jogging for instance is not as harmful. In such scenarios between encouraging unwanted user expenditures or helpful recommendations in some cases and the 'morality' of the framework, one can safely agree to process the user data as not only has the privacy policy been agreed upon but quite a few consumers do end up finding the right goods/services.

Furthermore, as an ML engineer working for a company, one has limited options on the course of action to take once it is learnt that the data collection was non-consensual. Thus, to overlook self-integrity for purely economic incentives becomes more of a survival strategy in cases of non sensitive data. It stands to reason that only a freelance or research ML engineer who has a direct influence over a data collection would have a chance to 'care' for the consent.

In addition, the morality of non-consensual data collection may be explained by the colloquial 'aim justifying means'. According to the EEE Code of Ethics (IEEE 2018): the professional responsibility is to 'to hold paramount the safety, health, and welfare of the public'. As mentioned earlier, in certain situations, working for the common good might be in conflict with the responsibility to the individual. A perfect example for this is IBM's machine learning system that analysed patients' speech patterns to predict psychosis among patients, a serious mental disorder characterized by an impaired relationship with reality. This is again a scenario of personal choice to gauge the damage level for the individual against the benefits for the society. This raises the ideas of a decision boundary - how sensitive is the collected data? In practical everyday life this is not an easily measurable quantity, if at all.

ML engineers in addition respect the humanitarian right of the individual which is entitle to privacy under Article 12 of the 1948 Universal Declaration of Human Rights: 'No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honour and reputation.'

The ethical issues are therefore very limited to legal regulation, business practices and largely dependent on the personal sense of responsibility toward another human being.

5 Code

This section contains the code of all the four methods implemented in this report.

5.1 KNN

```python
np.random.seed(1)
X_train = trainingSet.drop(columns = ['label'])
Y_train = trainingSet['label']

misclassification = []
for k in range(30):
    model = skl_nb.KNeighborsClassifier(n_neighbors=k+1)
    beta = model.fit(X_train, Y_train)
    prediction = beta.predict(X_train)
    misclassification.append(np.mean(prediction != Y_train))

K = np.linspace(1,30, 30)
plt.plot(K, misclassification)
plt.title('Training error')
plt.xlabel('k_value')
plt.ylabel('misclassification')
plt.legend()
plt.show()
```

K-NN with selected features

```python
np.random.seed(1)
X_train = trainingSet[['acousticness', 'energy', 'instrumentalness', 'key',
    'speechiness']]
Y_train = trainingSet['label']

misclassification = []
for k in range(30):
    model = skl_nb.KNeighborsClassifier(n_neighbors=k+1)
    beta = model.fit(X_train, Y_train)
    prediction = beta.predict(X_train)
    misclassification.append(np.mean(prediction != Y_train))

K = np.linspace(1,30, 30)
plt.plot(K, misclassification)
plt.title('Training error')
plt.xlabel('k_value')
plt.ylabel('misclassification')
plt.legend()
plt.show()
```

5.2 LDA

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb

trainingMusic = pd.read_csv('C:/Users/Erik Jan/Dropbox/MSc Computational
    Science/UU-61812 Statistical Machine Learning/Project/Data/training_data.csv')
Nr_points = len(trainingMusic)
```

```
Nr_parts = 10
SetDivisor = Nr_points//Nr_parts #Will include all the values. 10 parts of 75 data
    points each

acc_LDA = Nr_parts*[0]
acc_QDA = Nr_parts*[0]
MeanAccLDA = 13*[0]
MeanAccQDA = 13*[0]

outputColumn = 'label'
inputColumns = ['acousticness', 'danceability', 'duration', 'energy'
    ,'instrumentalness' , \
'key', 'liveness', 'loudness' ,'mode', 'speechiness', 'tempo', 'time_signature',
    'valence']

for factor in range(0,13):
    inputFactor = [inputColumns[factor]]
    for i in range(0,Nr_parts):
        # Setting up data
        train = trainingMusic[ : SetDivisor*i]                #First part of
            training data
        train = train.append(trainingMusic[SetDivisor * (i+1) : ]) #Adding second
            part to the training data
        validation = trainingMusic[SetDivisor*i : SetDivisor * (i+1)]

        X_train = train[inputFactor]
        Y_train = train.loc[:, outputColumn]
        X_validation = validation.loc[:, inputFactor]
        Y_validation = validation.loc[:, outputColumn]

        # LDA
        modelLDA = skl_da.LinearDiscriminantAnalysis()
        modelLDA.fit(X_train, Y_train)
        prediction = modelLDA.predict(X_validation)
        acc = np.mean(prediction == Y_validation)
        acc_LDA[i] = acc

        # QDA
        modelQDA = skl_da.QuadraticDiscriminantAnalysis()
        modelQDA.fit(X_train, Y_train)
        prediction = modelQDA.predict(X_validation)
        acc = np.mean(prediction == Y_validation)
        acc_QDA[i] = acc

    MeanAccLDA[factor] = np.mean(acc_LDA)
    MeanAccQDA[factor] = np.mean(acc_QDA)


# Plotting outcomes
plt.plot(inputColumns , MeanAccLDA, linestyle = 'dashed', label = 'LDA')
plt.plot(inputColumns , MeanAccQDA, linestyle = 'dashed', label = 'QDA')
plt.rcParams["figure.figsize"] = (12,9)
plt.legend()
plt.title('Comparing the accuracy for different input factors')
plt.xlabel('Input factors')
plt.xticks(inputColumns, inputColumns, rotation='vertical')
plt.ylabel('Prediction accuracy')
plt.show()

#Comparing methods
acc_AlwaysTrue = Nr_parts*[0]
acc_LDA = Nr_parts*[0]
acc_QDA = Nr_parts*[0]

outputColumn = 'label'
```

```
# Here you can select for which combination of input factors you want to test.
inputColumns = ['acousticness', 'danceability', 'duration', 'energy' ,'instrumentalness' , \
'key', 'liveness', 'loudness' ,'mode', 'speechiness', 'tempo', 'time_signature', 'valence']
#inputColumns = ['acousticness', 'energy', 'loudness', 'speechiness']

for i in range(0,Nr_parts):
    # Setting up data
    train = trainingMusic[ : SetDivisor*i]                #First part of training data
    train = train.append(trainingMusic[SetDivisor * (i+1) : ]) #Adding second part to the training data
    validation = trainingMusic[SetDivisor*i : SetDivisor * (i+1)]

    X_train = train.loc[:, inputColumns]
    Y_train = train.loc[:, outputColumn]
    X_validation = validation.loc[:, inputColumns]
    Y_validation = validation.loc[:, outputColumn]

    # Always assume 'like'
    prediction = len(X_validation)*[1]
    acc = np.mean(prediction == Y_validation)
    acc_AlwaysTrue[i] = acc

    # LDA
    modelLDA = skl_da.LinearDiscriminantAnalysis()
    modelLDA.fit(X_train, Y_train)
    prediction = modelLDA.predict(X_validation)
    acc = np.mean(prediction == Y_validation)
    #print('Error rate for LDA: ' + str(err))
    acc_LDA[i] = acc

    # QDA
    modelQDA = skl_da.QuadraticDiscriminantAnalysis()
    modelQDA.fit(X_train, Y_train)
    prediction = modelQDA.predict(X_validation)
    acc = np.mean(prediction == Y_validation)
    #print('Error rate for QDA: ' + str(err))
    acc_QDA[i] = acc

print("The input columns are: ")
print(inputColumns)
print
# Printing outcomes
print("The average accuracy for AlwaysTrue is: " + str(np.mean(acc_AlwaysTrue)))
print("The average accuracy for LDA is: " + str(np.mean(acc_LDA)))
print("The average accuracy for QDA is: " + str(np.mean(acc_QDA)))

# Plotting outcomes
plt.boxplot([acc_AlwaysTrue, acc_LDA, acc_QDA])
plt.xticks([1, 2, 3], ['True', 'LDA', 'QDA'])
plt.title("Comparing accuracy for classification methods")
plt.xlabel("Classification method")
plt.ylabel("Accuracy")

### THREE INPUT FACTORS ###
# NOTE: Model takes at least 2 minutes to run. You can out-comment the print statements in the code below to keep track of progress.
# There are thirteen inout factors to choose from. Let's try to find a combination of three factors which form the best combination.

MaxAccuracy = 0
optimalFactorI = 0
optimalFactorJ = 0
```

```
optimalFactorK = 0

outputColumn = 'label'
inputColumns = ['acousticness', 'danceability', 'duration', 'energy'
    ,'instrumentalness' , \
'key', 'liveness', 'loudness' ,'mode', 'speechiness', 'tempo', 'time_signature',
    'valence']

modelLDA = skl_da.LinearDiscriminantAnalysis()

for i in range(0,12):
    for j in range(0,12):
        for k in range(0,12):
            if ((i !=j) & (i!=k) & (j!=k)):
                #print("The values of i, j and k are: " + str(i)+ ", " + str(j) + "
                    and " + str(k))
                TotalAccuracy = 0
                for time in range(0,Nr_parts):

                    # Setting up the data
                    train = trainingMusic[ : SetDivisor*time]
                    train = train.append(trainingMusic[SetDivisor * (time+1) : ])
                    validation = trainingMusic[SetDivisor*time : SetDivisor *
                        (time+1)]

                    X_train = train.loc[:, [inputColumns[i], inputColumns[j],
                        inputColumns[k]]]
                    Y_train = train.loc[:, outputColumn]
                    X_validation = validation.loc[:, [inputColumns[i],
                        inputColumns[j], inputColumns[k]]]
                    Y_validation = validation.loc[:, outputColumn]

                    # Using the model
                    modelLDA.fit(X_train, Y_train)
                    prediction = modelLDA.predict(X_validation)
                    acc = np.mean(prediction == Y_validation)
                    TotalAccuracy = TotalAccuracy + acc

                MeanAccuracy = TotalAccuracy/Nr_parts
                #print(MeanAccuracy)

                if (MeanAccuracy > MaxAccuracy):
                    MaxAccuracy = MeanAccuracy
                    optimalFactorI = i
                    optimalFactorJ = j
                    optimalFactorK = k

print
print("The maximum accuracy is: " + str(MaxAccuracy))
print("Optimal factor 1: " + str(inputColumns[optimalFactorI]))
print("Optimal factor 2: " + str(inputColumns[optimalFactorJ]))
print("Optimal factor 3: " + str(inputColumns[optimalFactorK]))
```

5.3 Logistic Regression

```
np.random.seed(1)
songs=pd.read_csv('training_data.csv')
songs.columns.values

songs["mode"] = songs["mode"].astype('category')
#model 1 start:
trainI=np.random.choice(songs.shape[0], size=600, replace=False)
trainIndex=songs.index.isin(trainI)
```

```
train=songs.iloc[trainIndex]
test=songs.iloc[~trainIndex]

model=skl_lm.LogisticRegression()

X_train=train[['acousticness', 'danceability', 'duration', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence']]
Y_train=train['label']

X_test=test[['acousticness', 'danceability', 'duration', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence']]
Y_test=test['label']

model.fit(X_train, Y_train)
print('Model summary:')
print(model)
predict_prob=model.predict_proba(X_test)
print('The class order in the model:')
print(model.classes_)
print('Examples of predicted probablities for the above classes:')
predict_prob[0:5]# inspect the first 5 predictions

prediction=np.empty(len(X_test), dtype=object)
prediction=np.where(predict_prob[:,0]>=0.5,'0','1')
prediction[0:5]
print(pd.crosstab(prediction, Y_test))

p = prediction.astype(int) #need to convert since error otherwise
y=Y_test.astype(int)
np.mean(p==y)

#Cross Validation

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = model, X = X_train, y = Y_train, cv = 10)
accuracies.mean()

# try the ROC curve:
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = model.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(Y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#model 2 start:
#Reduce model complexity by selecting only the most corrolated covarites: ie
    acousticness, energy, loundess, speechiness, dacebility
X_train2=train[['acousticness', 'danceability','energy','loudness','speechiness']]
Y_train2=train['label']

```

```
X_test2=test[[’acousticness’, ’danceability’,’energy’,’loudness’,’speechiness’]]
Y_test2=test[’label’]

model2=model.fit(X_train2, Y_train2)
print(’Model summary:’)
print(model2)

predict_prob2=model2.predict_proba(X_test2)
print(’Examples of predicted probablities for the above classes:’)
predict_prob2[0:5]# inspect the first 5 predictions

prediction2=np.empty(len(X_test2), dtype=object)
prediction2=np.where(predict_prob2[:,0]>=0.5,’0’,’1’)
prediction2[0:5]

print(pd.crosstab(prediction2, Y_test2))

p2 = prediction2.astype(int) #need to convert since error otherwise
y2=Y_test2.astype(int)
np.mean(p2==y2)

#Cross Validation

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = model2, X = X_train2, y = Y_train2, cv =
    10)
accuracies.mean()

import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = model2.predict_proba(X_test2)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(Y_test2, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title(’Receiver Operating Characteristic’)
plt.plot(fpr, tpr, ’b’, label = ’AUC = %0.2f’ % roc_auc)
plt.legend(loc = ’lower right’)
plt.plot([0, 1], [0, 1],’r--’)
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel(’True Positive Rate’)
plt.xlabel(’False Positive Rate’)
plt.show()

#model 3 start:
#reduce model again:
X_train3=train[[’acousticness’,’energy’,’loudness’,’speechiness’]]
Y_train3=train[’label’]

X_test3=test[[’acousticness’,’energy’,’loudness’,’speechiness’]]
Y_test3=test[’label’]

model3=model.fit(X_train3, Y_train3)
print(’Model summary:’)
print(model3)

predict_prob3=model3.predict_proba(X_test3)
print(’Examples of predicted probablities for the above classes:’)
predict_prob3[0:5]# inspect the first 5 predictions

prediction3=np.empty(len(X_test3), dtype=object)
prediction3=np.where(predict_prob3[:,0]>=0.5,’0’,’1’)
```

```python
prediction3[0:5]

print(pd.crosstab(prediction3, Y_test3))

p3 = prediction3.astype(int) #need to convert since error otherwise
y3=Y_test3.astype(int)
np.mean(p3==y3)

#Cross Validation

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = model3, X = X_train3, y = Y_train3, cv =
    10)
accuracies.mean()

# try the ROC curve:
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = model3.predict_proba(X_test3)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(Y_test3, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#tune the parameter - hyperparameter optimisation using grid search: for every
    available model; then ccompare again:
# try to find optimal hyper paramter:
# Load libraries
import numpy as np
from sklearn import linear_model, datasets
from sklearn.model_selection import GridSearchCV

# Create logistic regression
logistic = linear_model.LogisticRegression()

# Create regularization penalty space
penalty = ['l1', 'l2']

# Create regularization hyperparameter space
C = np.logspace(0, 4, 10)

# Create hyperparameter options
hyperparameters = dict(C=C, penalty=penalty)

# Create grid search using 5-fold cross validation
clf = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0)

# Fit grid search
best_model1 = clf.fit(X_test,Y_test)

# View best hyperparameters
print('Best Penalty:', best_model1.best_estimator_.get_params()['penalty'])
print('Best C:', best_model1.best_estimator_.get_params()['C'])

```

```
#improving model 1:
#apply this penalty:
model_best1=skl_lm.LogisticRegression(penalty = 'l1', C =
    2.7825594022071245,random_state = 0)
model_best1.fit(X_train, Y_train)
predict_probbest1=model_best1.predict_proba(X_test)


prediction1=np.empty(len(X_test), dtype=object)
prediction1=np.where(predict_probbest1[:,0]>=0.5,'0','1')
prediction1[0:5]

#Cross Validation
model_best1c=skl_lm.LogisticRegression(penalty = 'l1', C =
    2.7825594022071245,random_state = 0)

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator =model_best1c , X = X_train, y = Y_train, cv
    = 10)
accuracies.mean()

randomize_indices=np.random.choice(songs.shape[0], songs.shape[0], replace=False)
misclassification=np.zeros((10,200))

for i in range(10):
    n=np.ceil(songs.shape[0]/10)# number of samples in each fold
    validationIndex=np.arange(i*n,min(i*n+n, songs.shape[0]),1).astype('int')
    randomize_validationIndex=randomize_indices[validationIndex]
    train=songs.iloc[~songs.index.isin(randomize_validationIndex)]
    validation=songs.iloc[randomize_validationIndex]

    X_train=train[['acousticness', 'danceability', 'duration', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence']]
    Y_train=train['label']
    X_validation=validation[['acousticness', 'danceability', 'duration', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'mode',
       'speechiness', 'tempo', 'time_signature', 'valence']]
    Y_validation=validation['label']

model_f1=skl_lm.LogisticRegression(penalty = 'l1', C =
    2.7825594022071245,random_state = 0,solver='liblinear')
model_f1.fit(X_train, Y_train)
prediction=model_f1.predict(X_validation)
err=np.mean(prediction!=Y_validation)
print('Error rate for logistic regression:'+str(err))

# Fit grid search for second reduced model:
best_model2 = clf.fit(X_test2,Y_test2)

# View best hyperparameters
print('Best Penalty:', best_model2.best_estimator_.get_params()['penalty'])
print('Best C:', best_model2.best_estimator_.get_params()['C'])

#improving model 2:
#apply this penalty:
model_best2=skl_lm.LogisticRegression(penalty = 'l2', C =59.94842503189409
    ,random_state = 0)
model_best2.fit(X_train2, Y_train2)
predict_probbest2=model_best2.predict_proba(X_test2)


prediction2=np.empty(len(X_test2), dtype=object)
prediction2=np.where(predict_probbest2[:,0]>=0.5,'0','1')
prediction2[0:5]
```

```

#Cross Validation
model_best2c=skl_lm.LogisticRegression(penalty = '12', C =
    59.94842503189409,random_state = 0)

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator =model_best2c , X = X_train2, y = Y_train2,
    cv = 10)
accuracies.mean()

#for second penalized model:
randomize_indices2=np.random.choice(songs.shape[0], songs.shape[0], replace=False)
misclassification2=np.zeros((10,200))

for i in range(10):
    n=np.ceil(songs.shape[0]/10)# number of samples in each fold
    validationIndex2=np.arange(i*n,min(i*n+n, songs.shape[0]),1).astype('int')
    randomize_validationIndex2=randomize_indices2[validationIndex2]
    train2=songs.iloc[~songs.index.isin(randomize_validationIndex2)]
    validation2=songs.iloc[randomize_validationIndex2]

    X_train2=train2[['acousticness',
        'danceability','energy','loudness','speechiness']]
    Y_train2=train2['label']
    X_validation2=validation2[['acousticness',
        'danceability','energy','loudness','speechiness']]
    Y_validation2=validation2['label']

model_f2=skl_lm.LogisticRegression(penalty = '12', C =59.94842503189409
    ,random_state = 0,solver='liblinear')
model_f2.fit(X_train2, Y_train2)
prediction2=model_f2.predict(X_validation2)
err2=np.mean(prediction2!=Y_validation2)
print('Error rate for logistic regression:'+str(err2))

# Fit grid search for third reduced model:
best_model3 = clf.fit(X_test3,Y_test3)

# View best hyperparameters
print('Best Penalty:', best_model3.best_estimator_.get_params()['penalty'])
print('Best C:', best_model3.best_estimator_.get_params()['C'])

#improving model 3:
#apply this penalty:
model_best3=skl_lm.LogisticRegression(penalty = 'l1', C
    =2.7825594022071245,random_state = 0)
model_best3.fit(X_train3, Y_train3)
predict_probbest3=model_best3.predict_proba(X_test3)

prediction3=np.empty(len(X_test3), dtype=object)
prediction3=np.where(predict_probbest3[:,0]>=0.5,'0','1')

#for second penalized model:
randomize_indices3=np.random.choice(songs.shape[0], songs.shape[0], replace=False)
misclassification3=np.zeros((10,200))

for i in range(10):
    n=np.ceil(songs.shape[0]/10)# number of samples in each fold
    validationIndex3=np.arange(i*n,min(i*n+n, songs.shape[0]),1).astype('int')
    randomize_validationIndex3=randomize_indices3[validationIndex3]
    train3=songs.iloc[~songs.index.isin(randomize_validationIndex3)]
    validation3=songs.iloc[randomize_validationIndex3]

    X_train3=train3[['acousticness','energy','loudness','speechiness']]
    Y_train3=train3['label']
```

```
    X_validation3=validation3[['acousticness','energy','loudness','speechiness']]
    Y_validation3=validation3['label']

model_f3=skl_lm.LogisticRegression(penalty = 'l1', C = 2.7825594022071245
     ,random_state = 0,solver='liblinear')
model_f3.fit(X_train3, Y_train3)
prediction3=model_f3.predict(X_validation3)
err3=np.mean(prediction3!=Y_validation3)
print('Error rate for logistic regression:'+str(err3))
prediction3[0:5]

#Cross Validation
model_best3c=skl_lm.LogisticRegression(penalty = 'l1', C =
     2.7825594022071245,random_state = 0)

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator =model_best3c , X = X_train3, y = Y_train3,
     cv = 10)
accuracies.mean()
```

5.4 Random Forest Classifier with AdaBoost

```
#Read Data
np.random.seed(1)
data = pd.read_csv('Data/training_data.csv');
print(data.head())

#MainData
X = data.drop(columns=['label'])
y = data['label']

#Split Using Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
     test_size=0.2,random_state=0)

#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)

#Fitting the model
rfc = RandomForestClassifier()
ada = AdaBoostClassifier(n_estimators=100, base_estimator = rfc,learning_rate=0.001)
ada.fit(X_train,y_train)

#Cross Validation and AccuracyScore
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = ada, X = X_train, y = y_train, cv = 10)
accuracies.mean()

#Predictions and string output for inputting in the leaderboard
ada_predict = ada.predict(X_test)
string = ''
for i in ada_predict:
    string += str(i)
print(string)

#Confusion Matrix and Classification Report
print("=== Confusion Matrix ===")
print(confusion_matrix(y_test, ada_predict))
print('\n')
print("=== Classification Report ===")
```

```python
print(classification_report(y_test, ada_predict))
print('\n')

#Plotting Validation Curve
param_range = np.arange(1,600,50)
train_scores, test_scores = validation_curve(rfc,
                                            X_train,
                                            y_train,
                                            param_name="n_estimators",
                                            param_range=param_range,
                                            cv=5,
                                            scoring="accuracy",
                                            n_jobs=-1)
# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot mean accuracy scores for training and test sets
plt.plot(param_range, train_mean, label="Training score", color="black")
plt.plot(param_range, test_mean, label="Cross-validation score", color="dimgrey")

# Plot accurancy bands for training and test sets
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std,
    color="gray")
plt.fill_between(param_range, test_mean - test_std, test_mean + test_std,
    color="gainsboro")

# Create plot
plt.title("Validation Curve With Random Forest")
plt.xlabel("Number Of Trees")
plt.ylabel("Accuracy Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

References

1 [1] Article title: Machine Learning Basics with the K-Nearest Neighbors Algorithm Website title: Towards Data Science URL: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[2]Article title: Hyperparameter Optimization in Machine Learning Website title: DataCamp Community URL: https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models

[3]Article title: 3.5. Validation curves: plotting scores to evaluate models — scikit-learn 0.20.2 documentation Website title: Scikit-learn.org
URL: https://scikit-learn.org/stable/modules/learning$_{c}urve.html$