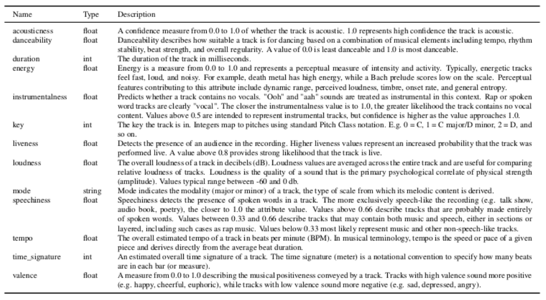
Statistical Machine Learning
Group 34 S413(3 Members) : Mini Project Report

1 Introduction

Companies often use recommendation engines to suggest other movies, songs or product to their customers. A crucial feature of such an engine is recommending a product which the user likes and not recommending a product that the user dislikes, without the user having explicitly said they like or dislike this particular product. To predict whether a user will like or dislike a particular song, we can use different machine learning methods to predict whether a specific user will like or dislike a particular product.

In this project, we will attempt to predict which songs Andreas Lindholm (Course Instructor) likes and dislikes songs in a dataset of 200 songs using a labelled dataset of 750 songs and different machine learning classification algorithms. The datasets consists not of the songs themselves, but of high-level features extracted using the web-API from Spotify. These high-level features describe characteristics such as the acousticness, danceability, energy, instrumentalness, valence and tempo of each song. Figure 1.1 shows all the features present in the dataset. The data set to classify is available as *songs_to_classify.csv*, and the training data is available as *training_data.csv* [1]..



Name	Type	Description
acousticness	float	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
danceability	float	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
duration	int	The duration of the track in milliseconds.
energy	float	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
instrumentalness	float	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
key	int	The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C major/D minor, 2 = D, and so on.
liveness	float	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	float	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
mode	string	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived.
speechiness	float	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
tempo	float	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
time_signature	int	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
valence	float	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

Figure 1.1: Details on the available features (from the Spotify API documentation) [2]

2 Data

2.1 Data Normalization

Normalization refers to the process of standardizing the values of independent features of a dataset. Since many of the machine learning techniques use distance to compute the difference between two or more distinct samples, a feature within these samples that has a broad range of values will dominate the process. In order to avoid this, the range of all features are normalized so that each feature contributes approximately proportionately to the computation. As discussed later in the report, methods such as 3.1, 3.2, 3.3 work really well if the data is normalized.

2.2 Dimensionality Reduction

Machine learning algorithms based on distance perform better if the number of features is low [7]. Hence we used PCA to reduce the number of features. A divide-and-conquer approach was used to determine the number of features to reduce to. We applied PCA with number of components as 13 so that we can check how many features were contributing in the variation of data with the particular label. From Figure 2.1 we can see that if we take 9 as $n\_components$ in PCA we are able to capture almost 97% of the variation in the data.

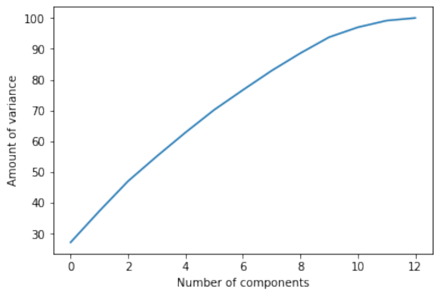


Figure 2.1: Variation in data captured with the different components in PCA

3 Methods

3.1 Logistic Regression

Regression analysis is a predictive modelling technique that describes the relationship between an input variable x(independent) and a quantitative output variable y(dependent). Logistic regression(LR) is a classification algorithm mostly used in predicting the probability of a categorical output variable. Usually the output variable is a binary outcome that can be either 1 (LIKE a song) or 0 (DISLIKE a song) making the model to predict P(y=1) as a function of x.

Linear regression is not used in binary classification as it needs us to decide a threshold value for which classification to be done and it is very sensitive to unequally sized class in the training data [4]. Unlike this, logistic predicts probabilities(values between (0,1) range) for the classification and these probabilities are estimated well enough by using sigmoid function when compared with the probabilities predicted by other methods such as Naive Bayes.

Furthermore, it also preserves the marginal probabilities of the training data. The model parameters of LR are found by maximum likelihood by maximizing the log-likelihood function. A consideration we need to take into account when using this methods is that input and output variables don't require linear relationship and all the input variables must be categorical. However, LR cannot handle large number of categorical variables and that's why need to perform data normalization on input variables. This method wouldn't be ideal if we needed to predict more than two class labels in the output.

3.2 Logistic Discriminant Analysis

Logistic regression is the combination of linear regression and applying sigmoid function into the equation of linear regression to model p(y|**x**) [3]. In Logistic Discriminant Analysis (LDA) we assume that p(**x**|y) is a gaussian distribution. This assumption give us a model classifier which is easy to learn and comes in handy when the gaussian assumption about p(**x**|y) isn't met.

The unique thing about LDA is that it models the distribution of predictors separately in each of the response classes, and then it uses Bayes' theorem to estimate the probability [6]. Bayes theorem describes a basic relationship between conditional and marginal probabilities i.e p(y = k) denotes the marginal (prior) probability of class k $\in \{1, .., K\}$ and $p(x|y = k)$ denotes the conditional probability density of x for an observation that comes from the kth class [5]. Bayes classifier is expressed using bayes theorem as

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{\sum_{k=1}^{k} p(\mathbf{x}|k)p(k))} \tag{3.1}$$

In LDA we focus on the right hand side of the equation 3.1, with the assumption that p(**x**|y) has a gaussian distribution. Parameters of the gaussian distribution i.e. the mean vector μ and covariance matrix Σ are learned by the lda classifier through training data. The mean vector μ is assumed to be different for each class, but the covariance matrix Σ is assumed to be the same for all classes.

3.3 K-Nearest Neighbour

The methods discussed in 3.1 and 3.2 have a fixed set of parameters and these parameters are learned from training data. By increasing the training data, parameters can be estimated more accurately but with no significant improvement in flexibility of the model. We focus on the methods where we do not rely on fixed structure and set of parameters i.e KNN 3.3 and Tree based methods 3.4.

Assume we define a set R_x = {i : x_i is one of the k training data points closest to x_* } then we can write the classifier as :

$$p(y = j|x_*) = \frac{\sum_{i \in \{R_x\}} \Pi\{y_i = j\}}{k} \tag{3.2}$$

A kNN classifier follows the principles of a bayes classifier; simply classifies samples by a majority vote of its neighbors, with a sample being assigned to the class with the majority vote among the `k` nearest neighbors [7]. The hyper-parameter `n_neighbors` determines the number of neighbors to use to choose the label of a sample data. The value of `k` is user-chosen integer greater than 1.

3.4 Tree based methods

Classification trees are a very different approach than the above mentioned methods in 3.3 and in 3.2. The basic idea of is to partition the data samples and in each region a constant value for the predicted class probability p(y | **x**) is assigned. Classification trees are a hierarchical way of partitioning the space. We start with the entire space and recursively divide it into smaller regions so that we can assign every region to a class label [8].

As we mentioned, the tree represents the recursive splitting of the data samples. Each region corresponds to one node of interest in the original data samples. If we put together two child nodes that occupy two different regions then we get the same region as that of the parent node. In the end, each and every leaf node is assigned with a class and test sample point get assigned with the class of the leaf node it falls into.

The advantages of using these methods are that they are easy to understand, interpret and visualize, and implicitly handle variable screening or feature selection. They can handle numerical and categorical data with relatively little efforts from users for data preparation or processing. Sometimes these trees can make a over-complex model that might overfit the data, we need to be careful when it comes to tuning these methods. They handle variance in the data poorly, which can be compensated by other methods such as bagging or boosting.

Deep tree models generally have low bias but high variance, it might overfit the data. In bagging we create many deep tree models and then take the average of them to compensate, which results in low bias and reduced variance due to the division with number of deep tree models.

Random forest (RF) is an ensemble learning method for classification that works by creating multiple decision trees during training and predicting the class that is the mode of the classes of the individual decision trees [7]. Random forest is almost equal to bagged trees but variation is further reduced by decorrelating the B ensemble tree members.

3.5 Boosting

Boosting is also a type of tree based methods which sequentially uses an ensemble of weak models, each of them describing some relationship between data and combine all these weak models to form

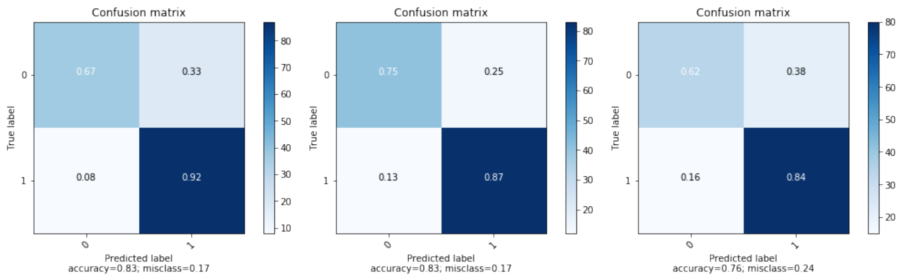
		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

(a) Confusion Matrix (b) Recall (c) Precision

Figure 4.1: Confusion matrices for methods run with PCA dimensionality reduction.



(a) LDA (b) LR (c) KNN

Figure 4.2: Confusion matrices for methods run with PCA dimensionality reduction.

one strong model. Building models sequentially is really important so that it can correct the mistakes made by the previous models.

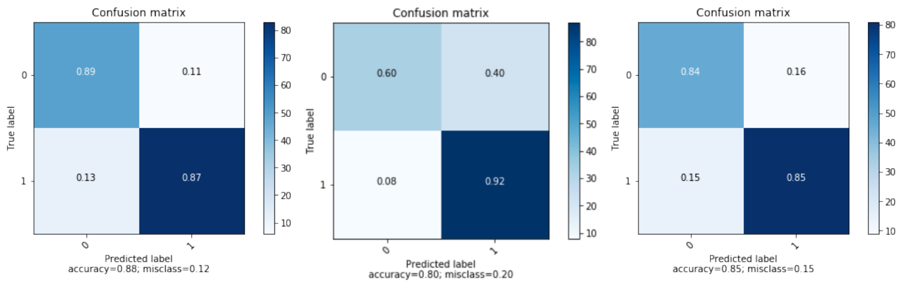
There are various boosting algorithms, namely - AdaBoost, Gradient boosting and XGBoost algorithms. All these algorithms are based on tree based classifiers with XGBoost being the most popular among the three. We use the gradient boosting classifier for our experiments and get pretty good results. This algorithm is similar to Adaptive Boosting(AdaBoost) but differs from it on certain aspects. In this method we try to visualize the boosting problem as an optimization problem, i.e we take up a loss function and try to optimize it.

4 Result

We evaluate the performance of a model not only on accuracy but we looked at recall and precision as well. A model with higher recall and accuracy value will be able to find the songs liked by Andreas more efficiently.

4.1 Logistic Regression

Logistic regression performs fairly well on the given dataset, resulting in overall 83% accuracy and 87% recall. Figure 4.2b shows the confusion matrix of Logistic regression.



(a) Boosting (b) Decision tree (c) Random Forest

Figure 4.3: Confusion matrices for methods run without PCA dimensionality reduction.

4.2 Logistic Discriminant Analysis

We used LDA with both svd *solver* and *eigen* solvers. Based on our experiments, we observed the svd solver performed better. We get an accuracy of 83% and 92% recall value for *svd* while *eigen* gives 78% accuracy and 80% recall. Figure 4.2a is the confusion matrix with solver as *svd*

4.3 K-Nearest Neighbour

For K-Nearest Neighbour, we have one hyper parameter- *n_neighbours*. We use cross_val_score for hyper parameter tuning and observe that the model performs best for *k*=3. We get an accuracy of 76% and 84% recall value when the model is trained k=3. Figure 4.2c shows the confusion matrix for the same parameters.

4.4 Tree Based Methods

Figure 4.3b is the confusion matrix we got after tuning the hyper parameters of decision tree. We focused more on tuning hyper parameters for random forest as it's an ensemble of decision tree. We selected $\{\textit{max-depth}, \textit{max-features}, \textit{min-samples-leaf}, \textit{min-samples-split}, \textit{n-estimators}\}$ as the hyper parameters which to tune. Then we ran the RF algorithm for each of parameters, adjusting only that parameter, and plotting the accuracy under curve measure of accuracy for the training and the validation data.

For each parameter, we noted where the model started to overfit and used this as a guideline for the upper bound of the parameters in our *GridSearchCV* exhaustive search of parameters in order to decrease the computational complexity of the task. Figure 4.3c shows the confusion matrix we were able to achieve 85% with the particular hyper parameters values - $\{\textit{max-depth} : 4, \textit{max-features} : 5, \textit{min-samples-leaf} : 0.00867, \textit{min-samples-split} : 0.16594, \textit{n-estimators} : 32\}$

4.5 Boosting

For boosting algorithm, we used gradient boosting method, which has *learning_rate*, *max_depth*, *max_features*, *n_estimators* as hyper parameters that are particularly important. We tune these parameters using GridSearchCV method, and observe that the model performs best for $\{\textit{learning\_rate} : 0.1, \textit{max\_depth} : 5, \textit{max\_features} : 3, \textit{n\_estimators} : 80\}$. We train the model using these parameters and got the accuracy of 88% and 87% recall value as you can see in Figure 4.3a which is the main reason we submitted the prediction made by boosting as our final result.

5 Conclusion

We submitted the prediction result from boosting in the production as you can check in the above figure 4.3a, it not only has the highest accuracy but highest recall as well. We took recall into account as it tells the fraction of right result that have been made over the total amount of predictions. We ran into the the problem of overfitting with random forest because when we got 85% accuracy for random forest as mentioned in 4.4 but got 71% when submitted in production. We think this is because *GridSearchCV* overfitted the model to the training and validation data. Limiting the parameters input into *GridSearchCV* would be an intersting next step to investigate if this would decrease overfitting.

6 Reflection

We came up with two uses that would likely cause Andreas to be concerned and something he would likely not give his consent to.

- The music service is connected to an online gambling service and the collected data is used to select hyped up music that increases Andreas' energy and optimism whenever he is losing money on blackjack.
- The music service is connected to an automotive company and the collected data is used to play calm and soothing music while Andreas is driving to calm him down, as he is usually a reckless driver, to decrease the risk of a traffic accident.

It's our opinion that the circumstances are of foremost importance when evaluating whether these hypothetical scenarios could be considered ethical or not. Whether encouraging gambling or influencing the political opinion of someone can be considered ethical is certainly interesting to discuss, however, we focused on the correlation and collection of data in our discussion.

First off, the main ethical problem in these examples is that Andreas has given permission for these services to use his data, however, he is unaware the specific implementation and the assumption is that he would not have given consent had he known the full extent of the use and implementation of his data. Our understanding is that the services have not explicitly lied to Andreas, but rather obfuscated the extent to which his data will be used and to what purpose. If this obfuscation is intended to withhold information from the user which is crucial for their decision whether or not to accept this treatment of data, one can consider this to be deception.

Reasoning about this using Kant's Categorical Imperative becomes straightforward. An act which relies upon deception cannot be made into a categorical imperative and thus not upheld to universal law. Therefore it cannot be considered ethical, no matter the intent of the companies in question.

This, however, mostly pertains to the question of whether the manner of getting user consent can be considered ethical. In order for it to be up to discussion whether these practises are ethical, the consent of the user first has to be explicitly given with a reasonably clear understanding about the extent and purpose of the data scientific activities that will be undertaken.

In both of these hypothetical examples, the end of use of the machine learning algorithms is to change the emotional state of the user in order to influence their behavior. Our feeling is that the act of influencing behavior in and of itself is not necessarily problematic, unless the influence is too great, and essentially becomes manipulation. In this sense, it depends on how good the machine learning algorithms are at classifying behavior and to which extent this knowledge can be used to manipulate users.

In our discussion we assume that as machine learning engineers we are most likely not directly involved or responsible for the manner in which user consent is being granted. Either because we work on datasets which others have collected or because we work in a company which has a legal branch which deals with the question directly. Therefore someone else is legally and ethically responsible for the fact that the user data is given with the appropriate consent.

Let us entertain the fact that we are presented with a problem with which the data we have has not been collected with a full understanding by the user regarding the purpose the data will be used for. Maybe the data was collected many years ago or maybe, like in our examples above, the end of use of the data was not explicitly presented to the user. In this case we have an ethical dilemma.

Depending on the ethical viewpoint, the end goal of us may or may not be an important consideration. Let's say that we are using unethically collected data in order to develop a machine learning algorithm for our second example. From a Kantian viewpoint, we are relying upon deception and thus our actions cannot be classified as an categorical imperative and the action as a whole cannot be upheld as a universal law and must be considered unethical. However, a Utilitarian viewpoint would consider our actions to be ethical as by saving lives we are achieving the greatest common good for the greatest amount of people. Which viewpoint to adopt needs careful consideration that can vary on a case by case basis.

It is our opinion that although we are not necessarily ultimately responsible for the ethical and legal aspects of getting user consent, we still need to reflect upon the issue and make our own judgement. Simply passing the buck is not sufficient, instead we need to consider the circumstances ourselves and make an informed decision.

References

[1] David Widmann Andreas Lindholm. Statistical machine learning - mini project. `http://www.it.uu.se/edu/course/homepage/sml/project/`, 2019.

[2] David Widmann Andreas Lindholm. Music classification mini project: Instructions. `http://www.it.uu.se/edu/course/homepage/sml/project/instructions.pdf`, 2019.

[3] Fredrik Lindsten Thomas B. Schön Andreas Lindholm, Niklas Wahlström. Supervised machine learning. `http://www.it.uu.se/edu/course/homepage/sml/literature/lecture_`

notes.pdf, 2019.

[4] Rajesh S. brid. Regression analysis. https://medium.com/greyatom/logistic-regression-89e496433063, 2019.

[5] Fredrik Lindsten. Lecture 4 – discriminant analysis, k-nearest neighbors. http://www.it.uu.se/edu/course/homepage/sml/lectures/Lecture4_handout.pdf, 2019.

[6] Marco Peixeiro. Classification—linear discriminant analysis. https://towardsdatascience.com/classification-part-2-linear-discriminant-analysis-ea60c45b9ee5, 2019.

[7] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321321367.

[8] The Pennsylvania State University. Tree based method. https://newonlinecourses.science.psu.edu/stat508/lesson/11, 2019.

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

train_path = 'data/training_data.csv'
test_path = 'data/songs_to_classify.csv'

df_train = pd.read_csv(train_path)
df_test = pd.read_csv(test_path)

print('Total␣number␣of␣samples␣in␣training␣dataset:␣\t%s' % df_train.shape[0])
print('Total␣number␣of␣features:␣\t%s' % len(df_train.columns.values))
print('Total␣number␣of␣samples␣in␣test␣dataset:␣\t%s' % df_test.shape[0])

def null_column(df):
    return df.isnull().any()

null_column(df_train)

from sklearn.preprocessing import normalize
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
# Separate features from their labels
features = [x != 'label' for x in df_train.columns.values]
train_values = df_train.loc[:, features].values
train_labels = df_train.loc[:,['label']].values
test_values = df_test.values

# Scale values between 0 and 1
#train_values = normalize(train_values, axis=0,norm='max')
scaler = StandardScaler()
scaler.fit(train_values)
train_values_n =scaler.transform(train_values)
test_values_n =scaler.transform(test_values)

pd.DataFrame(train_values_n).describe()
pd.DataFrame(test_values_n).describe()

corr=pd.DataFrame(train_values_n).corr()
print([{x:y} for x in range(0,13) for y in range(0,13) if corr.iloc[x,y]>0.65 and x!=y])
print([{x:y} for x in range(0,13) for y in range(0,13) if corr.iloc[x,y]<-0.65 and x!=y])
print("Hence,␣this␣columns␣are␣correlated")
corr.style.background_gradient()

from sklearn.metrics import confusion_matrix
import itertools
```

```
def plot_confusion_matrix(cm, labels):
    '''
    Plot confusion matrix of the specified accuracies and labels
    '''
    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 − accuracy

    cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(6, 4))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title('Confusion␣matrix')
    plt.colorbar()

    # Draw ticks
    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=45)
    plt.yticks(tick_marks, labels)

    # Normalize
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.2f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True␣label')
    plt.xlabel('Predicted␣label\naccuracy={:0.2f};␣misclass={:0.2f}'.format(accuracy, misclass))
    plt.show()




labels_list = np.unique(df_train.loc[:,['label']].values)
from sklearn.decomposition import PCA

pcaComps=PCA(n_components=13)
pcaComps.fit(train_values_n)
#The amount of variance that each PC explains
var= pcaComps.explained_variance_ratio_
#Cumulative Variance explains
var1=np.cumsum(np.round(pcaComps.explained_variance_ratio_, decimals=4)*100)

plt.plot(var1)
plt.xlabel('Number␣of␣components')
plt.ylabel('Amount␣of␣variance')
plt.show()


# Hence, we will use 9 PCA components.




pca=PCA(n_components=9);
```

```
train_values_pca=pca.fit_transform(train_values_n);
test_values_pca=pca.transform(test_values_n);

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(train_values_pca, train_labels, test_size = 0.2, random_state = 0)

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

lda = LinearDiscriminantAnalysis()
model_lda_svd=lda.fit(X_train, Y_train.flatten())
predicted_labels = model_lda_svd.predict(X_val)
model_lda_svd_acc = accuracy_score(Y_val, predicted_labels)
model_lda_svd_acc_cm = confusion_matrix(Y_val, predicted_labels)
plot_confusion_matrix(model_lda_svd_acc_cm, labels_list)


lda = LinearDiscriminantAnalysis(solver='eigen',shrinkage='auto')
model_lda_eigen=lda.fit(X_train, Y_train.flatten())
predicted_labels = model_lda_eigen.predict(X_val)
model_lda_eigen_acc = accuracy_score(Y_val, predicted_labels)
model_lda_eigen_acc_cm = confusion_matrix(Y_val, predicted_labels)
plot_confusion_matrix(model_lda_eigen_acc_cm, labels_list)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
model_lr=classifier.fit(X_train,Y_train.flatten());
predicted_labels = model_lr.predict(X_val)
model_lr_acc = accuracy_score(Y_val, predicted_labels)
model_lr_acc_cm = confusion_matrix(Y_val, predicted_labels)
plot_confusion_matrix(model_lr_acc_cm, labels_list)

from sklearn.model_selection import train_test_split
from sklearn import neighbors
from sklearn.model_selection import cross_val_score

k_values = [2,3,4,5,6,7,8]

val_accuracy = []
for k in k_values:
    kNNClassifier = neighbors.KNeighborsClassifier(n_neighbors = k)
    accuracy = np.mean(cross_val_score(kNNClassifier,
                                       train_values_pca,
                                       y = train_labels.ravel(),
                                       cv = 5,
                                       n_jobs = -1))
    val_accuracy.append(accuracy)

print('Best␣K␣value:␣{}'.format(k_values[np.argmax(val_accuracy)]))

kNNClassifier = neighbors.KNeighborsClassifier(n_neighbors=3).fit(X_train, Y_train.flatten())
predicted_labels = kNNClassifier.predict(X_val)
knn_pca_acc = accuracy_score(Y_val, predicted_labels)
knn_pca_cm = confusion_matrix(Y_val, predicted_labels)
plot_confusion_matrix(knn_pca_cm, labels_list)

from sklearn import tree

dtClassifier = tree.DecisionTreeClassifier(max_depth=2)
dtClassifier.fit(X_train, Y_train)

import os
import graphviz
# os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'

dot_data = tree.export_graphviz(dtClassifier, out_file=None,
# feature_names = df_train.copy().drop(columns=["label"]).columns,
# class_names = model.classes_,
                                filled=True, rounded=True,
                                leaves_parallel=True, proportion=True)
graph = graphviz.Source(dot_data)
graph
#validation_labels = val_label.flatten()
predicted_labels = dtClassifier.predict(X_val)
print('Accuracy rate is %.2f' % np.mean(predicted_labels == Y_val))
pd.crosstab(predicted_labels, Y_val.flatten())

dt_pca_acc = accuracy_score(Y_val, predicted_labels)
dt_pca_cm = confusion_matrix(Y_val, predicted_labels)
plot_confusion_matrix(dt_pca_cm, labels_list)
## Random Forest
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
model.fit(X=X_train, y=Y_train.flatten())

predicted_labels = model.predict(X_val)
rf_pca_acc = accuracy_score(Y_val, predicted_labels)
rf_pca_cm = confusion_matrix(Y_val, predicted_labels)
plot_confusion_matrix(rf_pca_cm, labels_list)

from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(Y_val, predicted_labels)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc

Y_train = Y_train.flatten()
x_train = X_train
x_test = X_val

def parameter_tuning_rf(parameter_name, parameter_value, x_train, x_valid, y_train, y_valid):
    training_results = []
    validation_results = []

    for i in parameter_value:
        parameters = {
            parameter_name : i
        }
        parameters["n_estimators"] = i if parameter_name == "n_estimators" else 32
        rf = RandomForestClassifier(**parameters, n_jobs=-1)
        rf.fit(x_train, y_train)
        training_prediction = rf.predict(x_train)
        false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, training_prediction)
        training_results.append(auc(false_positive_rate, true_positive_rate))
        y_pred = rf.predict(x_valid)
        false_positive_rate, true_positive_rate, thresholds = roc_curve(y_valid, y_pred)
        validation_results.append(auc(false_positive_rate, true_positive_rate))
    from matplotlib.legend_handler import HandlerLine2D
    line1, = plt.plot(parameter_value, training_results, "b", label="Train AUC")
    line2, = plt.plot(parameter_value, validation_results, "r", label="Test AUC")
    plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
    plt.ylabel("AUC score")
    plt.xlabel(parameter_name)
    plt.show()

parameter_tuning_rf("n_estimators", [2**i for i in range(9)], x_train, x_test, Y_train.ravel(), Y_val)
parameter_tuning_rf("max_depth", list(range(1,17)), x_train, x_test, Y_train.ravel(), Y_val)
parameter_tuning_rf("min_samples_split", np.linspace(0.01, 1.0, 10, endpoint=True), x_train, x_test, Y_train.ravel(), Y_val)
parameter_tuning_rf("min_samples_leaf", np.linspace(0.01, 0.5, 5, endpoint=True), x_train, x_test, Y_train.ravel(), Y_val)
parameter_tuning_rf("max_features", list(range(1,X_train.shape[1])), x_train, x_test, Y_train.ravel(), Y_val)
from sklearn.model_selection import GridSearchCV

X_train_tree, X_val_tree, Y_train_tree, Y_val_tree = train_test_split(train_values, train_labels, test_size = 0.2, random_state = 0)

parameters = {
    "n_estimators" : [2**i for i in range(3,6)]
    ,"max_depth" : list(range(2,5))
    ,"min_samples_split" : np.linspace(0.2, 0.6, 30, endpoint=True)
    ,"min_samples_leaf" : np.linspace(0.01, 0.3, 30, endpoint=True)
    ,"max_features" : list(range(2,5))
}

clf = GridSearchCV(RandomForestClassifier(), parameters, cv=10, n_jobs=-1, iid=False)

clf.fit(X_train_tree, Y_train_tree.ravel())
print(clf.score(X_train_tree, Y_train_tree))
print(clf.best_params_)

from sklearn import ensemble

X_train_tree, X_val_tree, Y_train_tree, Y_val_tree = train_test_split(train_values, train_labels, test_size = 0.2, random_state = 0)


n_estimators = 80;

gb = ensemble.GradientBoostingClassifier(n_estimators=n_estimators,
                                         random_state=0, learning_rate=0.1, max_depth=4,max_features=3)
gb.fit(X_train_tree,Y_train_tree.ravel())

predicted_labels = gb.predict(X_val_tree)

gb_acc = accuracy_score(Y_val_tree, predicted_labels)
gb_cm = confusion_matrix(Y_val_tree, predicted_labels)
plot_confusion_matrix(gb_cm, labels_list)

from sklearn.model_selection import GridSearchCV

parameters = {
    "learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    "max_depth":[3,4,5,6],
    "max_features":["log2","sqrt"],
    "n_estimators":[50,80,100,150,120,200]
    }

clf = GridSearchCV(ensemble.GradientBoostingClassifier(), parameters, cv=10, n_jobs=-1)

clf.fit(X_train_tree, Y_train_tree)
print(clf.score(X_train_tree, Y_train_tree))
print(clf.best_params_)

def post_process_prediction(prediction):
    return ''.join(str(int) for int in prediction)

rf = RandomForestClassifier(n_estimators=32,
                            max_depth=4,
                            max_features=5,
                            min_samples_leaf=0.0086,
                            min_samples_split=0.1659)
rf.fit(train_values_pca, train_labels.ravel())
prediction = rf.predict(test_values_pca)
post_process_prediction(prediction)
```