SPOKEN LETTERS RECOGNITION USING MACHINE LEARNING

EXECUTIVE SUMMARY

In this work, the goal/ task was to classify the English letter spoken by the subject. A variety of speakers had to speak each alphabet twice and extracted features such as spectral coefficients, contour features, sonorant features, pre-sonorant features, and post-sonorant features were used in this task of classification. In this work, clustering and classification techniques were applied in classifying the English letter spoken. A multitude of feature selection/ reduction techniques were applied, and the machine learners were tuned for hyper-parameters using grid search techniques. Any evidence of overfitting was dealt with in this approach and since the dataset had 617 attributes, principal component analysis – a dimensionality reduction technique was also applied. As a result, a random forest classifier capable of classifying the spoken letters with 92.7% accuracy was built.

MAIN REPORT

In this work, the goal was to classify the English letter spoken by the speakers. The data used was collected from the UCL Machine Learning Repository. A total of 150 speakers were asked to speak the English alphabet twice. Hence each speaker recorded 52 times. The speakers were further categorized into 5 groups of 30 speakers each. The recordings from the first 4 groups were used as training data and the recordings from the last group were used as test data. Hence, 80% is used for training and 20% is used for testing.

The main goal of the project is to classify the spoke English alphabet with high accuracy. This task was a multi-class classification task. In this approach, I followed the knowledge discovery in databases process also called as the KDD process.

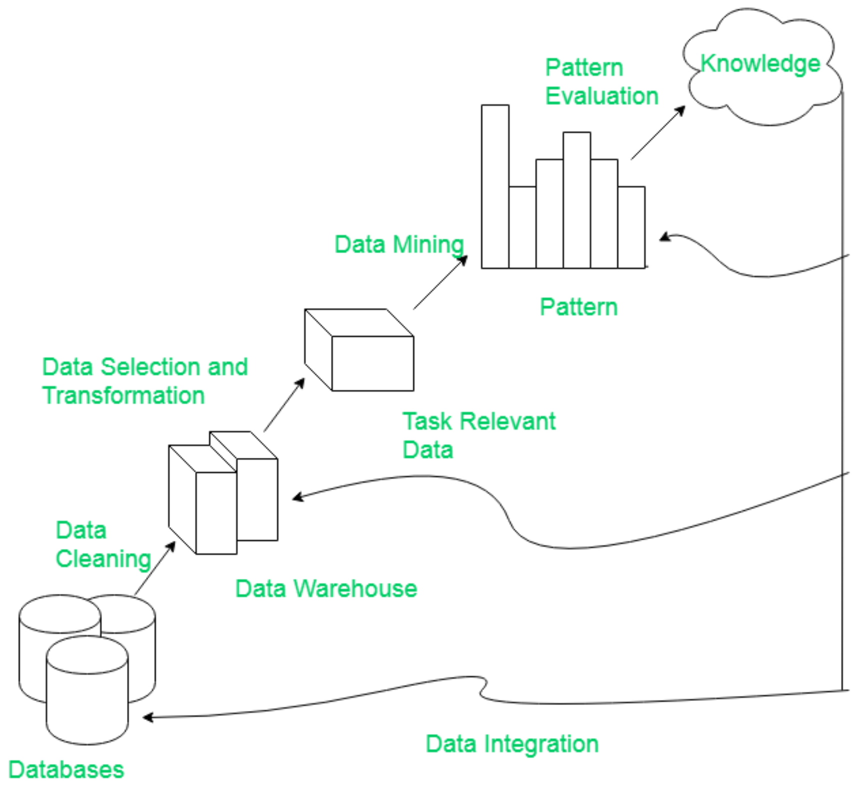


Figure 1 shows the steps in the knowledge discovery process

DATA CLEANING

In this work, the first step was to clean the data. For the same purpose, the data was checked for null values. It was found that the data set was already clean. The website had mentioned that three records were eliminated due to difficulties in recording. Two records from the train data were eliminated and one record from the test data was eliminated.

DATA SELECTION AND TRANSFORMATION

As a second step, the data were studied. All the 617 features in the data set were found to be numeric. Moreover, all the attributes lie between -1 and +1 values. Since these features were already scaled, no further transformation was required as a part of the analysis. Further, nowhere in the website or in the web were the description of the features were provided. We only know that the features in the data are spectral coefficients, contour features, sonorant features, pre-sonorant features, and post-sonorant features. We don't know in which order are these features present in the data set, which was a limitation of the dataset itself.

DATA MINING AND PATTERN RECOGNITION

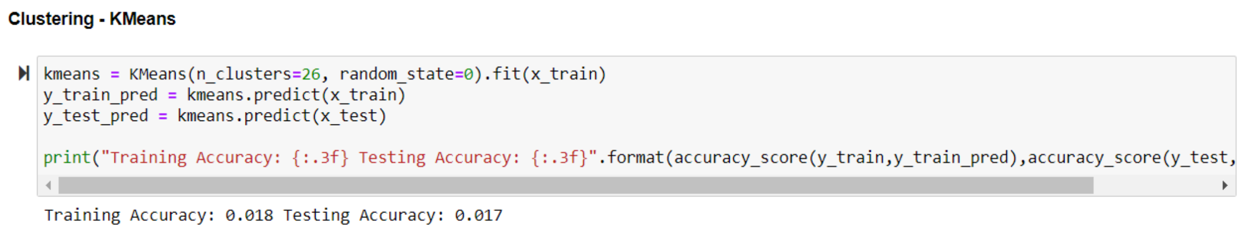
As a part of the data mining and pattern recognition step, we tried different techniques in understanding the patterns in the data and in modelling these patterns. The following techniques were implemented.

- Unsupervised technique – Clustering
  - K-Means Algorithm
- Supervised techniques – Classification
  - K-Nearest Neighbor Algorithm
  - Decision Tree Classifier
  - Ensemble Methods
    - Random Forest Classifier

UNSUPERVISED TECHNIQUE – CLUSTERING

K-MEANS ALGORITHM

Even though I had ground truth labels for the data, I tried clustering techniques learnt in class. K-Means clustering technique was used to cluster the different patterns in the data. We had 617 features and 26 different labels in the data. And hence, I tried to group the data into 26 different clusters. The grouped clusters were then compared to the ground truth labels. The results produced were less than 1% on both train and the test data. The reason behind this result is that the names of the clusters grouped by the k-means algorithm could be varying from the ground truth even though they were clustered together. And because of this reason, I had to give up on the clustering techniques and approach the supervised techniques for controlled results.


Clustering - KMeans

```
kmeans = KMeans(n_clusters=26, random_state=0).fit(x_train)
y_train_pred = kmeans.predict(x_train)
y_test_pred = kmeans.predict(x_test)

print("Training Accuracy: {:.3f} Testing Accuracy: {:.3f}".format(accuracy_score(y_train,y_train_pred),accuracy_score(y_test,
```

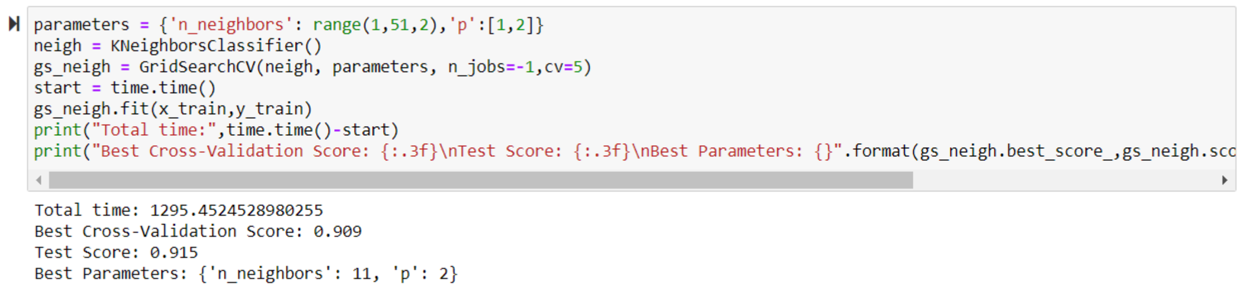
```
Training Accuracy: 0.018 Testing Accuracy: 0.017
```

SUPERVISED TECHNIQUE – CLASSIFICATION

K-NEAREST NEIGHBOR ALGORITH

The first classification technique to be implemented was the k-nearest neighbor algorithm. Often times in predictive modeling, simpler algorithms are powerful than complex models. This was very true in this work. It was found out that the KNN algorithm was able to classify 26 different classes with 91.5% accuracy on the test data. In KNN algorithm, the k-value and the distance measure to be used are two of the hyper-parameters that can vary and can impact the performance of the model.

Hence, in this work, I used grid search technique to tune my model for hyper-parameters. K-values of 1 through 49 and distance measures such as the Manhattan distance and the Euclidean distance were tuned. The optimal k value was 11 and distance measure was Euclidean distance. I performed a 5-fold cross validation to find the optimal parameters and the optimal model classified the letters spoken with 91.5% accuracy in the test data. This was promising and I wanted to try more advanced methods such as the decision tree classifier.

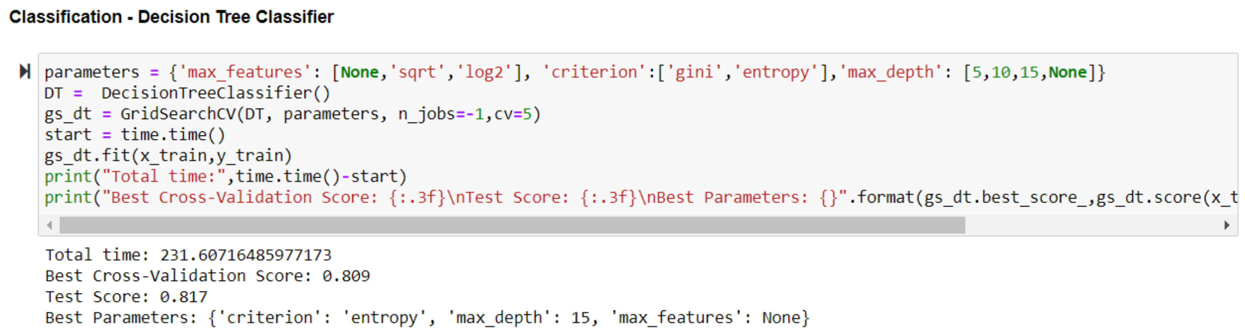


```
parameters = {'n_neighbors': range(1,51,2),'p':[1,2]}
neigh = KNeighborsClassifier()
gs_neigh = GridSearchCV(neigh, parameters, n_jobs=-1,cv=5)
start = time.time()
gs_neigh.fit(x_train,y_train)
print("Total time:",time.time()-start)
print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters: {}".format(gs_neigh.best_score_,gs_neigh.sco
```

```
Total time: 1295.4524528980255
Best Cross-Validation Score: 0.909
Test Score: 0.915
Best Parameters: {'n_neighbors': 11, 'p': 2}
```

DECISION TREE CLASSIFIER

Secondly, the decision tree classifier was used to classify spoken English letters. In this approach, again I searched for optimal parameters using the grid search technique. I performed 5-fold cross validation to find the optimal parameters. The parameters that were tuned were maximum features to use for the classifier, the impurity measure to split the values at each node, and the maximum depth that the tree can grow. The optimal model built was able to classify with an accuracy of 81.7%. This performance was significantly lower than that of the KNN algorithm. This called for more advanced decision tree classifier. And hence, I tried an ensemble of decision trees. I used the random forest classifier to classify the English letters.



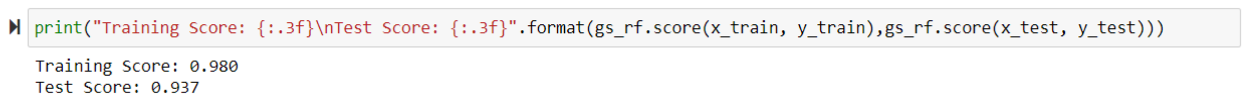
Classification - Decision Tree Classifier

```
parameters = {'max_features': [None,'sqrt','log2'], 'criterion':['gini','entropy'],'max_depth': [5,10,15,None]}
DT =  DecisionTreeClassifier()
gs_dt = GridSearchCV(DT, parameters, n_jobs=-1,cv=5)
start = time.time()
gs_dt.fit(x_train,y_train)
print("Total time:",time.time()-start)
print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters: {}".format(gs_dt.best_score_,gs_dt.score(x_t
```

```
Total time: 231.60716485977173
Best Cross-Validation Score: 0.809
Test Score: 0.817
Best Parameters: {'criterion': 'entropy', 'max_depth': 15, 'max_features': None}
```

ENSEMBLE METHOD – RANDOM FOREST CLASSIFIER

In this approach, we had to again deal with various number of hyper-parameters. As a first step, I built a random forest classifier with default values from the sklearn package. The number of estimators were set to 100. The classifier built produced 100% accuracy on the training data and 94.5% on the test data. This was clearly overfitting. I dealt with overfitting through early pruning techniques. Once again, I used the grid search technique to find the optimal values for early pruning such as maximum depth each tree can grow, minimum number of samples for split at the node levels and the leaf levels. The number of features to be used for each tree was also another hyper-parameter tuned. The optimal model was able to produce 98% accuracy on the training data and 93.7% on the testing data. This was the most optimal performance in the work.

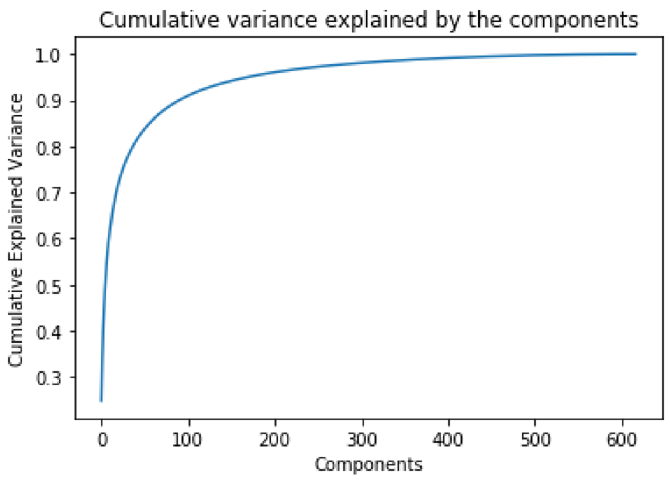


```
print("Training Score: {:.3f}\nTest Score: {:.3f}".format(gs_rf.score(x_train, y_train),gs_rf.score(x_test, y_test)))
```

```
Training Score: 0.980
Test Score: 0.937
```

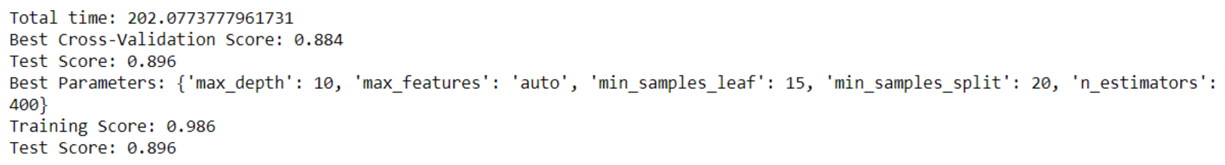
DATA SELECTION AND TRANSFORMATION

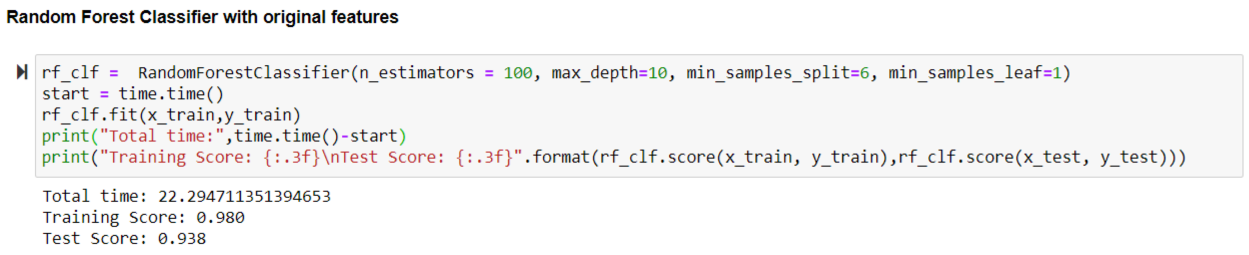
I once again went back to the previous step of the KDD cycle. Since I had 617 features in the data, I wanted to reduce the dimensions in the data. I applied principal component analysis (PCA) - a powerful feature reduction or dimensionality reduction technique. After applying PCA, it was found that 91 principal components were able to capture 90% of the variance in the data. And hence, I transformed the data from 617 features to 91 principal components.



MODELLING

Since the dimensions were reduced, I had to repeat the predictive modelling techniques again. The random forest classifier showed most promise from our previous analysis. And hence, I used the random forest classifier. The transformed data were used to train the random forest classifier and once again the hyper-parameters were tuned. The optimal model produced 98.6% on the training data and 89.6% on the testing data. The model trained using transformed data produced lesser performance than the model trained on untransformed original data. And hence, I use the performance of the model trained on untransformed data as my final results.


```
Total time: 202.0773777961731
Best Cross-Validation Score: 0.884
Test Score: 0.896
Best Parameters: {'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 15, 'min_samples_split': 20, 'n_estimators':
400}
Training Score: 0.986
Test Score: 0.896
```


Random Forest Classifier with original features

```
rf_clf =  RandomForestClassifier(n_estimators = 100, max_depth=10, min_samples_split=6, min_samples_leaf=1)
start = time.time()
rf_clf.fit(x_train,y_train)
print("Total time:",time.time()-start)
print("Training Score: {:.3f}\nTest Score: {:.3f}".format(rf_clf.score(x_train, y_train),rf_clf.score(x_test, y_test)))
```

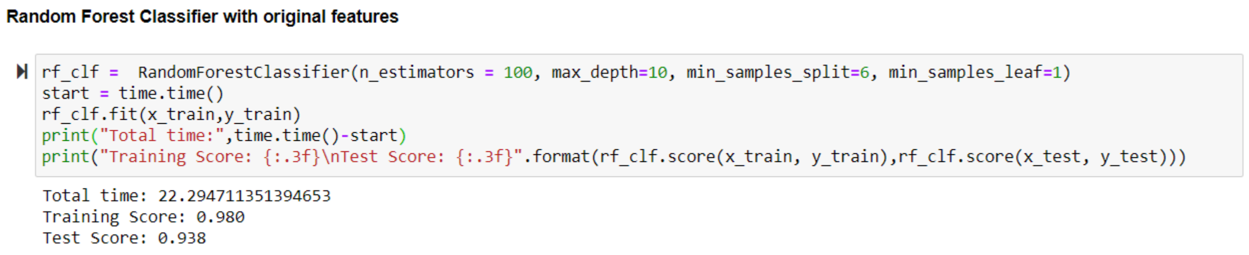
```
Total time: 22.294711351394653
Training Score: 0.980
Test Score: 0.938
```

KNOWLEDGE DISCOVERY

From the random forest model, I was able to extract the importance of each feature to the model. This is showed using the bar graph produced in the jupyter notebook html file. Since the feature description were not provided anywhere, I cannot show which feature in terms of the coefficients were able to impact the performance of the model.

RESULTS

In this work, I trained various classifiers in classifying spoken English alphabet using different audio features. In working towards the goal, I dealt with overfitting, extensive search of hyper-parameters, and high dimensionality. As discussed in the report, various tools and techniques were implemented to address these modelling issues. As a result, I built a random forest classifier capable of classifying spoke English alphabets with 93.8% accuracy.


Random Forest Classifier with original features

```
rf_clf =  RandomForestClassifier(n_estimators = 100, max_depth=10, min_samples_split=6, min_samples_leaf=1)
start = time.time()
rf_clf.fit(x_train,y_train)
print("Total time:",time.time()-start)
print("Training Score: {:.3f}\nTest Score: {:.3f}".format(rf_clf.score(x_train, y_train),rf_clf.score(x_test, y_test)))
```

```
Total time: 22.294711351394653
Training Score: 0.980
Test Score: 0.938
```

TOOLS USED

1. SKLEARN Package
2. Numpy
3. Pandas
4. Matplotlib
5. Graphviz

CODE

```
#!/usr/bin/env python

# coding: utf-8


# ### DSC 478 - Final Project

# ### Arun Gopal Govindaswamy

#


# In[35]:



import pandas as pd

import numpy as np

import time


from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy_score

from sklearn.metrics.cluster import homogeneity_score

from sklearn.metrics.cluster import completeness_score

from sklearn.decomposition import PCA


import matplotlib.pyplot as plt
import graphviz

from sklearn.tree import export_graphviz



# In[2]:



x_train = pd.read_csv("isolet1+2+3+4.data", header = None)

y_train = x_train.pop(617)

x_test = pd.read_csv("isolet5.data", header = None)

y_test = x_test.pop(617)

print("Train data shape: {}\nTrain labels shape: {}\nTest data shape: {}\nTest labels shape:
{}".format(x_train.shape,y_train.shape,x_test.shape,y_test.shape))



# ### KDD - Data Cleaning Step


# **Checking for null values**


# In[4]:



print("Number of null entries in the training data set:",x_train.isnull().sum().sum())

print("Number of null entries in the test data set:",x_test.isnull().sum().sum())
```

```
# There are no null values in both train and test data.


# ### KDD - Data Selection and Transformation Step


# **Data Description**


# In[5]:



print(x_train.describe())



# It is seen from the top and bottom 10 columns that the data is already normalized and the values lie between -1
and 1. Let us confirm this by printing minimum and maximum of each column in the data set.


# In[6]:



for index in x_train.columns:
    print("Column Index: {} \t Minimum: {:.2f} \t Maximum:
{}".format(index,x_train[index].min(),x_train[index].max()))



# All the feature values lie between -1 and +1. And, hence we do not need any more normalization.


# ### KDD - Data Mining and Pattern Recogntion step
# **Clustering - KMeans**


# In[29]:



kmeans = KMeans(n_clusters=26, random_state=0).fit(x_train)

y_train_pred = kmeans.predict(x_train)

y_test_pred = kmeans.predict(x_test)


print("Training Accuracy: {:.3f} Testing Accuracy:
{:.3f}".format(accuracy_score(y_train,y_train_pred),accuracy_score(y_test,y_test_pred)))



# KMeans clustering failed terribly on our dataset. Let's try doing classification.


# **Classification - KNearest Neighbor Algorithm**


# In[7]:



neigh = KNeighborsClassifier()

neigh.fit(x_train, y_train)

print("Training Accuracy: {:.3f} Testing Accuracy:
{:.3f}".format(neigh.score(x_train,y_train),neigh.score(x_test,y_test)))



# That was a very good start to classify. We got 93.3% on train and 91.3% on test data which is great considering
this is a multi class classification problem. We also do not see evidence of overfitting.
```

```
#

# We achieved this high accuracy with a default parameters. We will try tuning the model with hyper-parameters.


# In[18]:




parameters = {'n_neighbors': range(1,51,2),'p':[1,2]}

neigh = KNeighborsClassifier()

gs_neigh = GridSearchCV(neigh, parameters, n_jobs=-1,cv=5)

start = time.time()

gs_neigh.fit(x_train,y_train)

print("Total time:",time.time()-start)

print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters: 
{}".format(gs_neigh.best_score_,gs_neigh.score(x_test, y_test) ,gs_neigh.best_params_))



# We increased the performance on the test data marginally. Let's try a different classifier to see if it helps.


# **Classification - Decision Tree Classifier**


# In[22]:




parameters = {'max_features': [None,'sqrt','log2'], 'criterion':['gini','entropy'],'max_depth': [5,10,15,None]}

DT =  DecisionTreeClassifier()

gs_dt = GridSearchCV(DT, parameters, n_jobs=-1,cv=5)

start = time.time()
```

```
gs_dt.fit(x_train,y_train)

print("Total time:",time.time()-start)

print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters: {}".format(gs_dt.best_score_,gs_dt.score(x_test, y_test) ,gs_dt.best_params_))
```

# A singel decision tree did not work in our favour. The performance was lower than that of a KNN algorithm. Hence, experimenting if an ensemble of trees such as the random forest classifier produce better performance.

# **Classification - Random Forest Classifier**

# In[48]:

```
rf_clf =  RandomForestClassifier(n_estimators = 100)

start = time.time()

rf_clf.fit(x_train,y_train)

print("Total time:",time.time()-start)

print("Training Score: {:.3f}\nTest Score: {:.3f}".format(rf_clf.score(x_train, y_train),rf_clf.score(x_test, y_test)))
```

# We could see some clear evidence of overfitting here. Let us deal with overfitting by early pruning. Playing with the maximum depth of the tree, and number of samples for leaf and node split can reduce overfitting

# In[49]:

```
parameters = {'n_estimators':[100],'max_depth': [5,6,7,8,9,10], 'min_samples_split': range(1,10),
'min_samples_leaf': range(1,10)}

rf =  RandomForestClassifier()

gs_rf = GridSearchCV(rf, parameters, n_jobs=-1,cv=2, verbose = 8)

start = time.time()

gs_rf.fit(x_train,y_train)

print("Total time:",time.time()-start)

print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters:
{}".format(gs_rf.best_score_,gs_rf.score(x_test, y_test) ,gs_rf.best_params_))



# In[50]:



print("Training Score: {:.3f}\nTest Score: {:.3f}".format(gs_rf.score(x_train, y_train),gs_rf.score(x_test, y_test)))



# We were able to produce a random forest classifier capabale of predicting the spoken letter with 93.7% accuracy.

#

# Let's also try having different number of features for the decision tree



# In[3]:



parameters = {'n_estimators':[100],'max_depth': [10], 'min_samples_leaf': [6],
'max_features':['sqrt',None,'log2',100,200,300,400]}

rf =  RandomForestClassifier()

gs_rf = GridSearchCV(rf, parameters, n_jobs=-1,cv=2, verbose = 8)
```

```
start = time.time()

gs_rf.fit(x_train,y_train)

print("Total time:",time.time()-start)

print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters: 
{}".format(gs_rf.best_score_,gs_rf.score(x_test, y_test) ,gs_rf.best_params_))

print("Training Score: {:.3f}\nTest Score: {:.3f}".format(gs_rf.score(x_train, y_train),gs_rf.score(x_test, y_test)))



# ### KDD - Data Transformation Step


# **Principal Component Analysis**


# In[56]:



pca = PCA().fit(x_train)

plt.plot(pca.explained_variance_ratio_.cumsum())

plt.xlabel("Components")

plt.ylabel("Cumulative Explained Variance")

plt.title("Cumulative variance explained by the components")



# In[22]:



pca.explained_variance_ratio_.cumsum()[:91]
```

```
# With 91 components, we are able to capture 90% variance in the data. Hence, we are now going to train a
random forest classifier using these 91 components of data.


# In[24]:



x_train_transform = pca.transform(x_train)

x_test_transform = pca.transform(x_test)



# In[25]:



rf_clf_2 =  RandomForestClassifier(n_estimators = 100)

start = time.time()

rf_clf_2.fit(x_train_transform,y_train)

print("Total time:",time.time()-start)

print("Training Score: {:.3f}\nTest Score: {:.3f}".format(rf_clf_2.score(x_train_transform,
y_train),rf_clf_2.score(x_test_transform, y_test)))



# Once again we see evidence of overfitting, hence we try and reduce overfitting by early pruning


# In[26]:
```

```
parameters = {'n_estimators':[100],'max_depth': [5,6,7,8,9,10], 'min_samples_split': range(1,10), 'min_samples_leaf': range(1,10)}

rf_2 =  RandomForestClassifier()

gs_rf_2 = GridSearchCV(rf_2, parameters, n_jobs=-1,cv=2, verbose = 8)

start = time.time()

gs_rf_2.fit(x_train_transform,y_train)

print("Total time:",time.time()-start)

print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters: {}".format(gs_rf_2.best_score_,gs_rf_2.score(x_test_transform, y_test) ,gs_rf_2.best_params_))

print("Training Score: {:.3f}\nTest Score: {:.3f}".format(gs_rf_2.score(x_train_transform, y_train),gs_rf_2.score(x_test_transform, y_test)))
```

# The gap between training and test score were reduced. Whereas, the performance is still low compared to earlier implementation without pca. So let's tune for number of estimators and number of features to be used.

# In[27]:

```
parameters = {'n_estimators':[100,200,300,400],'max_depth': [10,15,None], 'min_samples_split': [4], 'min_samples_leaf': [7], 'max_features':['auto',None,'sqrt',50]}

rf_2 =  RandomForestClassifier()

gs_rf_2 = GridSearchCV(rf_2, parameters, n_jobs=-1,cv=2, verbose = 8)

start = time.time()

gs_rf_2.fit(x_train_transform,y_train)

print("Total time:",time.time()-start)

print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters: {}".format(gs_rf_2.best_score_,gs_rf_2.score(x_test_transform, y_test) ,gs_rf_2.best_params_))
print("Training Score: {:.3f}\nTest Score: {:.3f}".format(gs_rf_2.score(x_train_transform,
y_train),gs_rf_2.score(x_test_transform, y_test)))
```

# With 400 trees, we were able to produce 91.7% accuracy on the test data. There is some overfitting here. Let's tune for early pruning here.

# In[33]:

```
parameters = {'n_estimators':[400],'max_depth': [5,10], 'min_samples_split': [15,20], 'min_samples_leaf': [20,15],
'max_features':['auto']}

rf_2 =  RandomForestClassifier()

gs_rf_2 = GridSearchCV(rf_2, parameters, n_jobs=-1,cv=2, verbose = 8)

start = time.time()

gs_rf_2.fit(x_train_transform,y_train)

print("Total time:",time.time()-start)

print("Best Cross-Validation Score: {:.3f}\nTest Score: {:.3f}\nBest Parameters:
{}".format(gs_rf_2.best_score_,gs_rf_2.score(x_test_transform, y_test) ,gs_rf_2.best_params_))

print("Training Score: {:.3f}\nTest Score: {:.3f}".format(gs_rf_2.score(x_train_transform,
y_train),gs_rf_2.score(x_test_transform, y_test)))
```

# Even after extensive search for hyper-parameters, our model was not able to produce better results than the random forest classifier trained on original untransformed data.

# And hence, we will be using only the original dataset from here on.

KDD - Predictive modelling step

```
# **Random Forest Classifier with original features**


# In[57]:


rf_clf =  RandomForestClassifier(n_estimators = 100, max_depth=10, min_samples_split=6, min_samples_leaf=1)

start = time.time()

rf_clf.fit(x_train,y_train)

print("Total time:",time.time()-start)

print("Training Score: {:.3f}\nTest Score: {:.3f}".format(rf_clf.score(x_train, y_train),rf_clf.score(x_test, y_test)))


# The highest performance on test data wsa 93.8% using the random forest classifier and original data without any transformation.


# ### KDD - Knowledge Discovery Step


# In[52]:


rf_clf.feature_importances_


# In[53]:


def plot_feature_importances(model, n_features, feature_names):
    plt.figure(figsize = (10,150))

    plt.barh(range(n_features), model.feature_importances_, align='center')

    plt.yticks(np.arange(n_features), feature_names)

    plt.xlabel("Feature importance")

    plt.ylabel("Feature")

    plt.ylim(-1, n_features)


features = x_train.columns

plot_feature_importances(rf_clf, len(features), features)




# We see from the bar graph that most of the features contribute to the model.


# The features used to classify human spoken letters were spectral coefficients; contour features, sonorant features, pre-sonorant features, and post-sonorant features. However the exact order of these features are unknown and not enclosed anywhere on the web.


# In[ ]:
```