In the name of God

University of Tehran

Collage of electrical and computer engineering

Machine Learning Project

Soheil Salehi – Nika Emami – Navid Dehban – Romina Omidi

810198422 – 810198356 – 810198390 – 810198539

Spring 1401

Contents

Introduction

In this project, our goal is to implement supervised and unsupervised methods to classify or cluster our data. The dataset we are using in this project is a set of voices from different ages, genders, and emotions. The goal is to determine each voice record's gender, emotion, and also in some cases, the different sentences that are in each record.

To implement a model for such a dataset, the first and the most important step is to implement a noise reduction on our dataset. After noise removal, we have to do some preprocessing on our data to prepare data for classification. In preprocessing step, first, we would calculate the Fourier transformation to simplify our data and be able to extract features from the data. The next step is to normalize our data. This step is used for changing our features to the same scale so that all features would affect the result in the same weight.

After Normalization, we can now start the main process which is feature extraction and feature selection. After selecting the best features for training data, we can now split our data into train and test data and use supervised and unsupervised methods to classify or cluster our data.

At last, we examine the quality of each model and with the use of metrics, we can determine the accuracy or other metrics used for measuring our model such as ROC table or confusion metrics to verify our models. At last, we will compare all these methods and define which algorithm works better for this dataset.

Preprocessing

First of all, we want to talk about what is preprocessing and why is it important in machine learning and model training.

preprocessing is the steps taken to format data before they are used by model training and inference. This includes, but is not limited to, resizing, orienting, and color corrections.

Thus, a transformation that could be augmentation in some situations may best be a preprocessing step in others. Consider altering image contrast. A given dataset could contain images that are generally of low contrast. If the model will be used in production on only low contrast in all situations, requiring that every image undergo a constant amount of contrast adjustment may improve model performance.

This preprocessing step would be applied to images in training and testing. However, if the collected training data is not representative of the levels of contrast the model may see in production, there is less certainty that a constant contrast adjustment is appropriate. Instead, randomly altering image contrast during training may generalize better. This would be augmentation. Knowing the context for data collecting and model inference is required to make informed preprocessing and augmentation decisions.

Preprocessing is required to clean image data for model input. For example, fully connected layers in convolutional neural networks required that all images are the same sized arrays.

Image preprocessing may also decrease model training time and increase model inference speed. If input images are particularly large, reducing the size of these images will dramatically improve model training time without significantly reducing model performance

Image augmentation creates new training examples out of existing training data. It’s impossible to truly capture an image that accounts for every real-world scenario a model may encompass. Adjusting existing training data to generalize to other situations allows the model to learn from a wider array of situations.

This is particularly important when collected datasets may be small. A deep learning model will (over)fit the examples shown in training, so creating variation in the input images enables the generation of new, useful training examples.

Now in this project, we will implement some of these methods to prepare data for training our models.

Before we start describing the project steps, here are all the libraries used for this project:

```
import os
import pandas as pd
import librosa
import soundfile
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, f1_score
from google.colab import drive
!pip install pydub
from pydub import AudioSegment
from pydub.utils import mediainfo
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from scipy.signal import lfilter
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from scipy.stats import loguniform
from sklearn.model_selection import RandomizedSearchCV
```

The first step is to read data and sstoredata and all labels and information about them in an array for processing theer. Here is a cod that would do so:

```
def Extract_Data(data_path, voice_path):
    emotions = []
    features = []
    loaded_data = []
    Counter = 0
    datas = Read_CSV(data_path)
    datas['voice id'] = datas['voice id'].astype('str') + '.wav'
    os.chdir(voice_path)
    for file in os.listdir():
      #print(file)
      Counter += 1
      if Counter == 1000:
        break
      if file in datas['voice id'].values and file not in loaded_data:
          loaded_data.append(file)
```

As we can see in the function, after reading files from the path and adding them to a list, we have called a function for extracting data. Here is where the preprocessing startThe firstrst step for preprocessing is to denoise our Torder to extract better features, we need to reduce noise in the background for each data.

Noise reduction

Noise reduction is the process of removing noise from a signal. Noise reduction techniques exist for audio and images. Noise reduction algorithms tend to alter signals to a greater or lesser degree.

In order to extract better features and training our model based on the model and not the effect of, noise it's always better to remove or reduce the noise in voice data.

In order to do so, we have used a function in python called lfilter() which uses a low pass filter to reduce the noise. Here is the code for doing the noise reduction:

```
voice_reduced = lfilter(b,a,voice)
```

Normalization

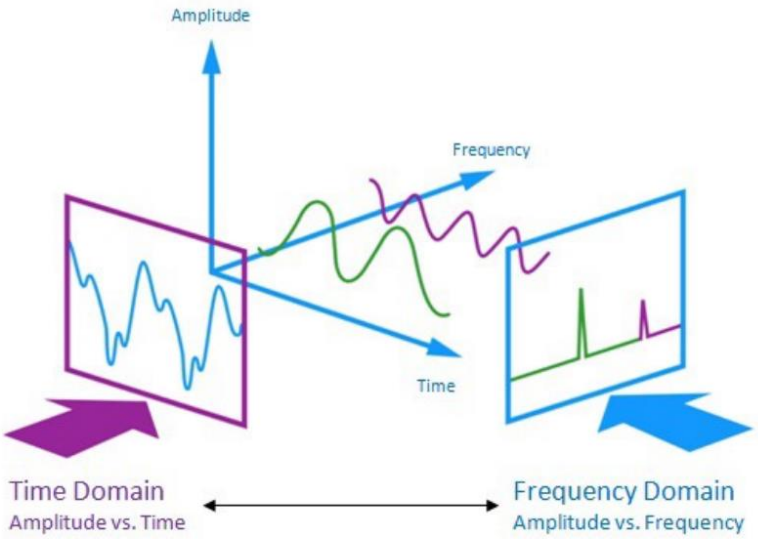
Data normalization is generally considered the development of clean data. Diving deeper, however, the meaning or goal of data normalization is twofold: Data normalization is the organization of data to appear similar across all records and fields.

In order to have a faster and more accurate model one of the most important steps is normalization. Normalization let all the data and their feature be all from the same scale so that all of the features would affect the result in the same way. Also, in some, an algorithm not implementing normalization may cause the convergence slow and sometimes the result would not converge to its optimum solution. So, we used the following code in order to have a higher accuracy:

```
scaler = preprocessing.MinMaxScaler()
normalize_voice = scaler.fit_transform(voice_reduced.reshape(-1, 1))
```

Feature extraction

The second step is feature extraction. To extract feature from a voice data, a very simple and efficient way is to convert the data using Fourier transform. Using Fourier transform let us minimize and compress all the usefull data used in a signal voice which was in a time domain into the frequency domain and this would simplify our data space:



For implementing this algorithm, we have used the code below:

```
mfcc = np.mean(librosa.feature.mfcc(y = normalize_voice.reshape(len(normalize_voice,)), sr = sample_rate, n_mfcc = 50).T, axis = 0)
```

As we can see there is a part called mfcc rate which would filter some frequencies or cluster the frequencies that are close to each other in a single part. We choose this rate to be 50 it means that it would classify the feature into 50 classes. And finally we would extract all data using all these algorithms and return them to be used in further examinations.

Feature reduction

Feature reduction, also known as dimensionality reduction, is the process of reducing the number of features in a resource heavy computation without losing important information. Reducing the number of features means the number of variables is reduced making the computer's work easier and faster.

So in order to see if using feature reduction algorithms make the accuracy better or not we are implementing our model both by using PCA (which is a dimension reduction algorithm) and without using PCA.

The number of input variables or features for a dataset is referred to as its dimensionality. Dimensionality reduction refers to techniques that reduce the number of input variables in a dataset. More input features often make a predictive modeling task more challenging to model, more generally referred to as the curse of dimensionality. High-dimensionality statistics and dimensionality reduction techniques are often used for data visualization.

In this project, we used the PCA method for implementing the dimension reduction. The most common approach to dimensionality reduction is called principal components analysis or PCA. PCA helps us to identify patterns in data based on the correlation between features. It aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions than the original one. The orthogonal axes (**principal components**) of the new subspace can be interpreted as the directions of maximum variance given the constraint that the new feature axes are orthogonal to each other.

After applying PCA on our data, we got 14 features as a result. (from both datasets, with 62 and 384 features)

By comparing the accuracy of our models both before and after applying PCA to our dataset, we saw that in some cases it helped improve the model, but in some cases the results became worse. The complete analysis and the effect of applying dimension reduction to our models, is later explained in this report.

The first set of features were extracted using MFCC and Chroma STFT and combining the results of the 2 methods, giving us a total of 62 features (50 from MFCC and 12 from Chroma STFT).

MFCC(Mel-Frequency Cepstral Coefficients) is one of the most important methods to extract a feature of an audio signal and is used majorly whenever working on audio signals. The mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope. Chroma_STFT performs short-time Fourier transform of an audio input and maps each STFT bin to chroma. In fact, it computes a chromagram from a waveform or power spectrogram. STFT represents information about the classification of pitch and signal structure. It depicts the spike with high values in low values.

The first feature extraction method code is as follow:

```
def Extract_First1_Feature(voice_name):
  voice, sample_rate = librosa.load(voice_name)
  n = 15
  b = [1.0 / n] * n
  a = 1
  voice_reduced = lfilter(b,a,voice)
  scaler = preprocessing.MinMaxScaler()
  normalize_voice = scaler.fit_transform(voice_reduced.reshape(-1, 1))
  cf = np.mean(librosa.feature.chroma_stft(y = normalize_voice.reshape(len(normalize_voice,)), sr = sample_rate).T, axis = 0)
  return cf
```

```
def Extract_Data_First_Feature(data_path, voice_path):
    emotions = []
    features = []
    loaded_data = []
    Counter = 0
    datas = Read_CSV(data_path)
    datas['voice id'] = datas['voice id'].astype('str') + '.wav'
    os.chdir(voice_path)
    for file in os.listdir():
      if file in datas['voice id'].values and file not in loaded_data:
          loaded_data.append(file)
          try:
            feature = Extract_Third_Feature(file)
            features.append(feature)
            index = datas[datas['voice id'] == os.path.basename(file)].index[0]
            emotion = Emotions[datas['emotionID'][index]]
            emotions.append(emotion)
          except Exception as e:
            print(e)
    return features, emotions
```

The second set of features were extracted using librosa.feature.tempogram. This feature is used when the music recording reveals significant tempo changes and the detection of locally periodic patterns becomes a challenging problem. In the concept of cyclic tempograms, the idea is to form tempo equivalence classes by identifying tempi that differ by a power of two. The resulting cyclic tempo features constitute a robust mid-level representation that reveals local tempo characteristics of music signals while being invariant to changes in the pulse level. Being

the tempo-based counterpart of the harmony-based chromagrams, cyclic tempograms are suitable for music analysis and retrieval tasks, where harmony-based criteria are not relevant. We extracted a total of 384 features in this case, and compared the results to the previous set before and after applying PCA as a method of dimension reduction on our data.

And the second feature extraction method code is as follow:

```
def Extract_Data_Second_Feature(data_path, voice_path):
    emotions = []
    features = []
    loaded_data = []
    Counter = 0
    datas = Read_CSV(data_path)
    datas['voice id'] = datas['voice id'].astype('str') + '.wav'
    os.chdir(voice_path)
    for file in os.listdir():
      if file in datas['voice id'].values and file not in loaded_data:
          loaded_data.append(file)
          try:
            feature = Extract_Third_Feature(file)
            features.append(feature)
            index = datas[datas['voice id'] == os.path.basename(file)].index[0]
            emotion = Emotions[datas['emotionID'][index]]
            emotions.append(emotion)
          except Exception as e:
            print(e)
    return features, emotions
```

Evaluation metrics of the supervised learning algorithms

At last we need some metrics in order to examine our model and test their accuracy so here are some methods to measure our models:

1. ROC curve:

   When we need to check or visualize the performance of the multi-class classification problem, we use the ROC (**Receiver Operating Characteristics**) curve. ROC is a probability curve and tells how much the model is capable of distinguishing between classes. The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis. A higher X-axis value indicates a higher number of False

positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance between False positives and False negatives.

2. Confusion matrix:

   It is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

   It is extremely useful for measuring Recall, Precision, Specificity, Accuracy, and most importantly AUC-ROC curves.

   True Positive: Interpretation: You predicted positive and it's true.

   True Negative: Interpretation: You predicted negative and it's true.

   False Positive: Interpretation: You predicted positive and it's false.

   False Negative: Interpretation: You predicted negative and it's false.

   In confusion matrices, we calculate some important values like accuracy, precision, and recall, which are explained below:

   Accuracy: From all the classes (positive and negative), how many of them we have predicted correctly. Accuracy should be high as possible.

Precision: From all the classes we have predicted as positive, how many are actually positive. Precision should be high as possible.

Recall: From all the positive classes, how many we predicted correctly. Recall should be high as possible.

3. Learning curve:
   A learning curve is a plot of model learning performance over experience or time. Learning curves are a widely used diagnostic tool in machine learning for algorithms that learn from a training dataset incrementally.

Reviewing learning curves of models during training can be used to diagnose problems with learning, such as an underfit or overfit model, as well as whether the training and validation datasets are suitably representative.

Train-Test Split:

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. It is a fast and easy procedure to perform, the results of which allows us to compare the performance of machine learning algorithms for the predictive modeling problem.

In this project, after extracting the features and labels, first we split the data into test and train sets, and then implement the supervised learning methods on the split data as mentioned later in the report.

Supervised Learning

In supervised machine learning algorithms, a labelled training dataset is used first to train the underlying algorithm. This trained algorithm is then fed on the unlabeled test dataset to categorize them into similar groups.

There are several supervised learning methods, in this project we implemented the models of SVM, Logistic Regression, KNN, MLP (Neural Networks).

Here is a brief explanation of each of the methods named above:

1. **Logistic Regression:**
   LR helps in finding the probability that a new instance belongs to a certain class. Since it is a probability, the outcome lies between 0 and 1. Therefore, to use the LR as a binary classifier, a threshold needs to be assigned to differentiate two classes.

2. **Support Vector Machine (SVM):**

   Support vector machine (SVM) algorithm can classify both linear and non-linear data. It first maps each data item into an n-dimensional feature space where *n* is the number of features. It then identifies the hyperplane that separates the data items into two classes while maximizing the marginal distance for both classes and minimizing the classification errors.

   To perform the classification, we then need to find the hyperplane that differentiates the two classes by the maximum margin.

3. **K-Nearest Neighbor:**

   The K-nearest neighbor (KNN) algorithm is one of the simplest and earliest classification algorithms. The KNN algorithm does not require to consider probability values. The '*K*' is the KNN algorithm is the number of nearest neighbors considered to take 'vote' from. The selection of different values for '*K*' can generate different classification results for the same sample object.

4. **MLP (Neural Networks):**

Neural Networks are a set of machine learning algorithms which are inspired by the functioning of the neural networks of human brain. Likewise, NN algorithms can be represented as an interconnected group of nodes. The output of one node goes as input to another node for subsequent processing according to the interconnection. Nodes are normally grouped into a matrix called layer depending on the transformation they perform. Nodes and edges have weights that enable to adjust signal strengths of communication which can be amplified or weakened through repeated training. Based on the training and subsequent adaption of the matrices, node and edge weights, NNs can make a prediction for the test data.

Implementation of different supervised learning methods

(using scikit library functions in Google Colab**):**

In the previous parts of this project, we extracted some features from our initial dataset (voices from different genders, emotions and text IDs). Now with having different features from about 17000 samples, and labels for emotions and genders for each sample, by using the scikit library we implement each of the mentioned models above and measure the accuracy of each.

Here are the methods for classifying the dataset based on gender:

1. Logistic Regression:
2. SVM:
3. KNN:
4. MLP:

Here are the methods for classifying the dataset based on emotion:

1. Logistic Regression:
2. SVM:
3. KNN:
4. MLP:

In the following pages we are going to show all the algorithms and methods that we used in each method(rather using PCA or not) and also 2 ways of feature extracting as we talked about fully in feature extraction chapter. In each method we are going to give a summary of how the algorithm works and show the confusion matrix and ROC table. And at the end we will compare all these methods together.

SVM

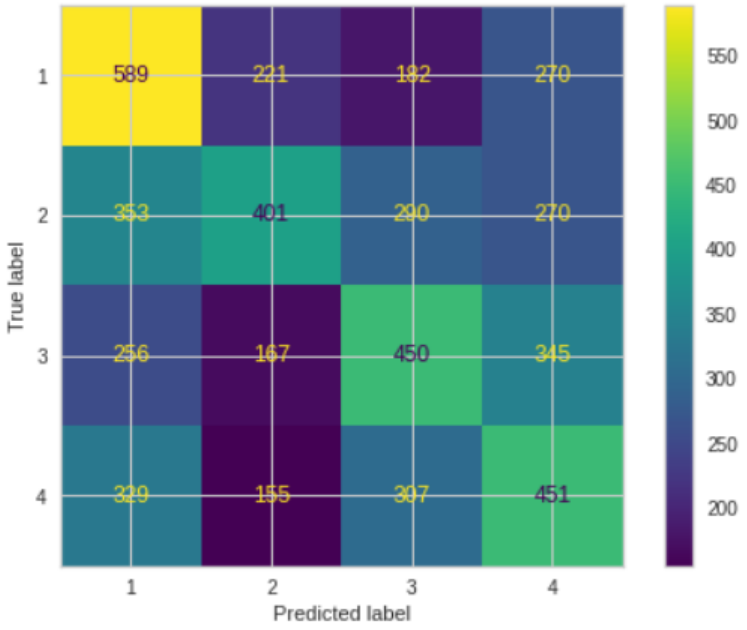
Support vector machine (SVM) algorithm can classify both linear and non-linear data. It first maps each data item into an n-dimensional feature space where *n* is the number of features. It then identifies the hyperplane that separates the data items into two classes while maximizing the marginal distance for both classes and minimizing the classification errors.
To perform the classification, we then need to find the hyperplane that differentiates the two classes by the maximum margin.

In this part we are doing a classification based on emotions, without using PCA. Also, we are using the first method of feature extraction(348)ZZ.
First, we do a grid-search on dataset to find the best hyper parameters:

```
from sklearn.svm import SVC
kernel = ['linear', 'rbf', 'sigmoid']
gamma = ['scale', 'auto']
for k in kernel:
  for g in gamma:
    clf = make_pipeline(StandardScaler(), SVC(kernel=k, gamma=g))
    print(clf.fit(x_train, y_train).score(x_test,y_test))
```

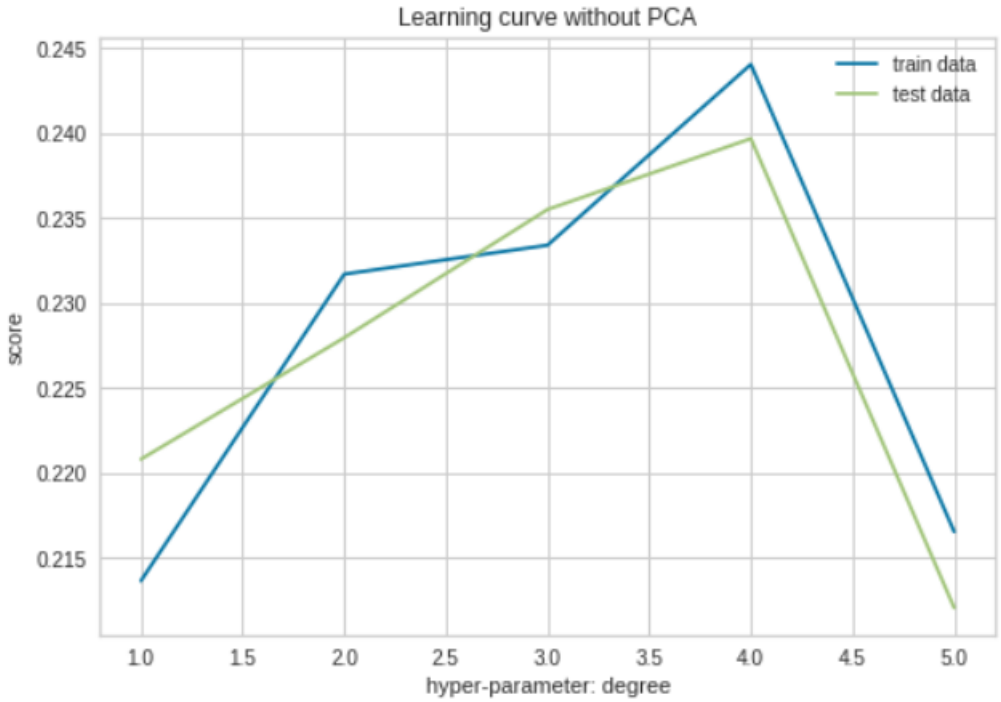
```
0.4019062748212867
0.4019062748212867
0.48451151707704526
0.48451151707704526
0.31731532962668785
0.31731532962668785
```

As we can see the best kernel is linear and also the best value for gamma parameter is scale. So, the accuracy for this method without using PCA is 40.19%. Then we plot the confusion for this algorithm:



As we can see algorithm works well in angry emotion. Also the number of data that their emotion has predicted correctly are high too but the algorithm predict a lots of data in a wrong classification. The number of wrong data in each part in confusion matrix might be less than correct ones but in total they make up the half of data set and so the accuracy is near 50%.

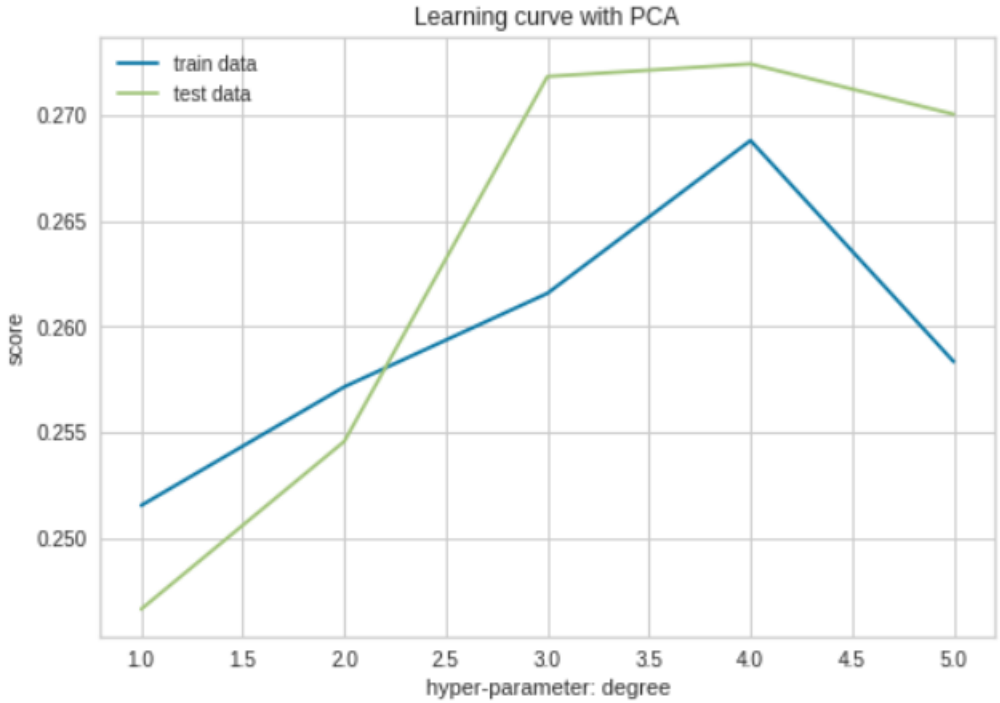
Learning curve for this model is as follow:



As we can see at first as the hyper parameters grow, the accuracy score rise but after a thresh hold, it will fall to zero.

Now we want to determine another model. This time, everything is the same but we are using PCA method in order to reduce dimension:

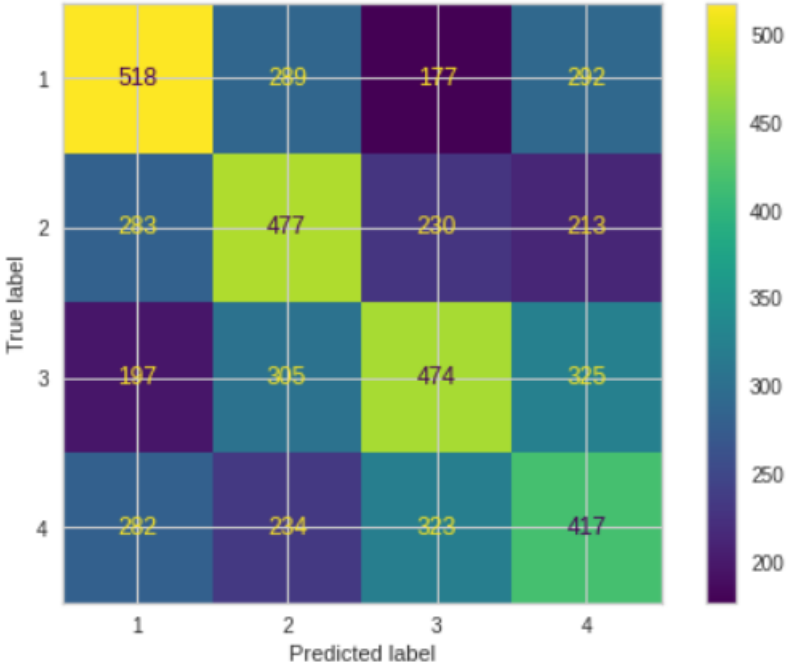
Here is the learning curve:



It's a little same to the previous curve but the different is that the test data is having a higher accuracy than train data which mean in previous algorithm,

model were being overfit a little bit and by reducing dimention we can have a better model.

Also here is the confusion matrix:



As we can see it's a little better than previous model and it will predict more correct data. But still the best emotion that the model can classify is angry.

KNN

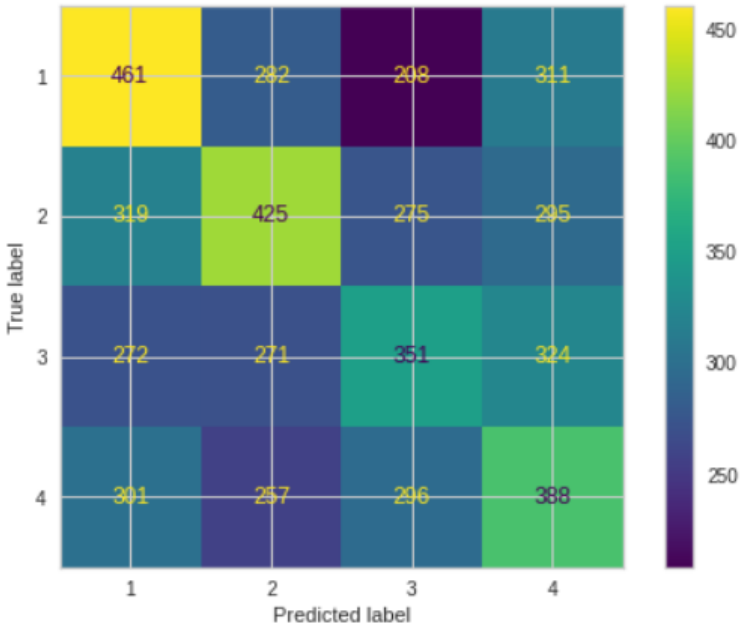
The K-nearest neighbor (KNN) algorithm is one of the simplest and earliest classification algorithms. The KNN algorithm does not require to consider probability values. The '*K*' is the KNN algorithm is the number of nearest neighbors considered to take 'vote' from. The selection of different values for '*K*' can generate different classification results for the same sample object.

In this part we are doing a classification based on emotions, without using PCA. Also, we are using the first method of feature extraction(348).
First, we do a grid-search on dataset to find the best hyper parameters:

```
for i in range(1,15):
  clf = KNeighborsClassifier(weights='distance',metric='manhattan',algorithm='ball_tree',n_neighbors=i)
  print(clf.fit(x_train, y_train).score(x_test, y_test))
```

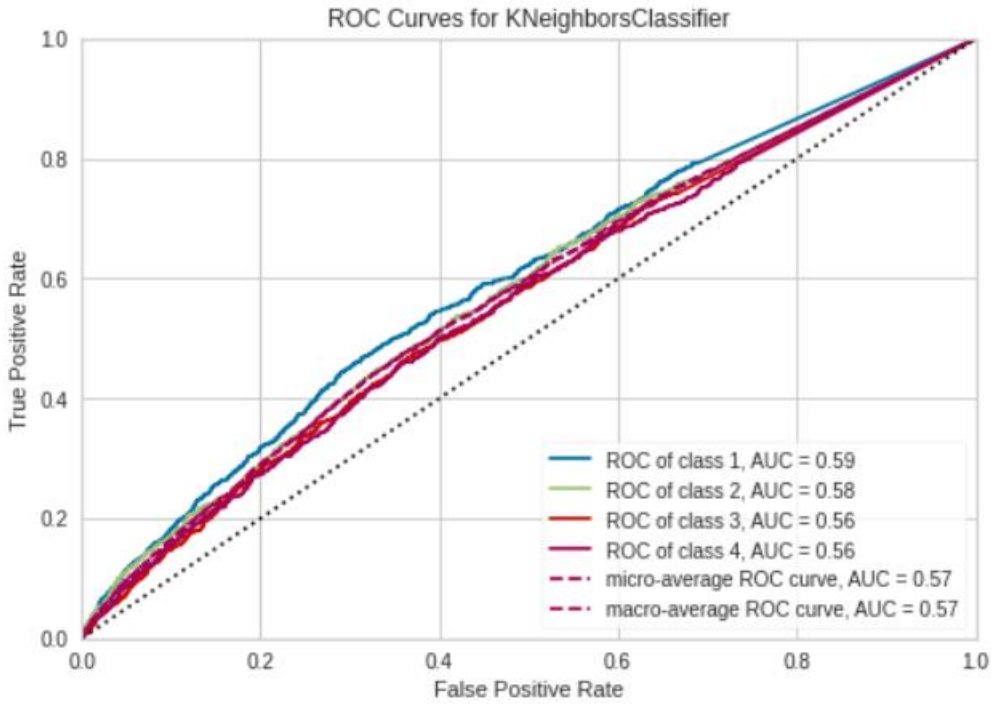
```
0.494440031771247
0.494241461477363
0.5077442414614773
0.5150913423351866
0.5160841938046068
0.5146942017474185
0.5127084988085783
0.5091342335186656
0.5041699761715648
0.5051628276409849
0.5045671167593329
0.5015885623510723
0.4954328832406672
0.4918586179507546
```

As we can see the best value for k which is the number of neighbors is 5. So, the accuracy for this method without using PCA is 51.60%. Then we plot the confusion for this algorithm:



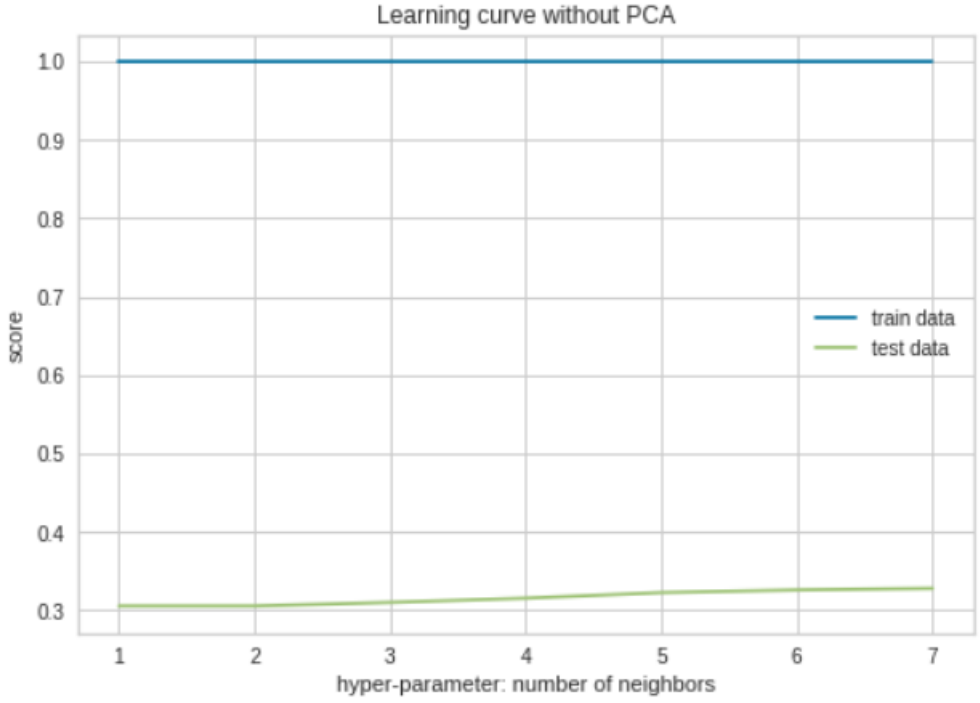
As we can see algorithm works well in angry emotion. Also the number of data that their emotion has predicted correctly are high too but the algorithm predict a lots of data in a wrong classification. The number of wrong data in each part in confusion matrix might be less than correct ones but in total they make up the half of data set and so the accuracy is near 50%.

Now we plot the ROC curve for KNN classifier:



As we can see this algorithm is not that accurate but still it is above the doted line which means it better than the random classifier.

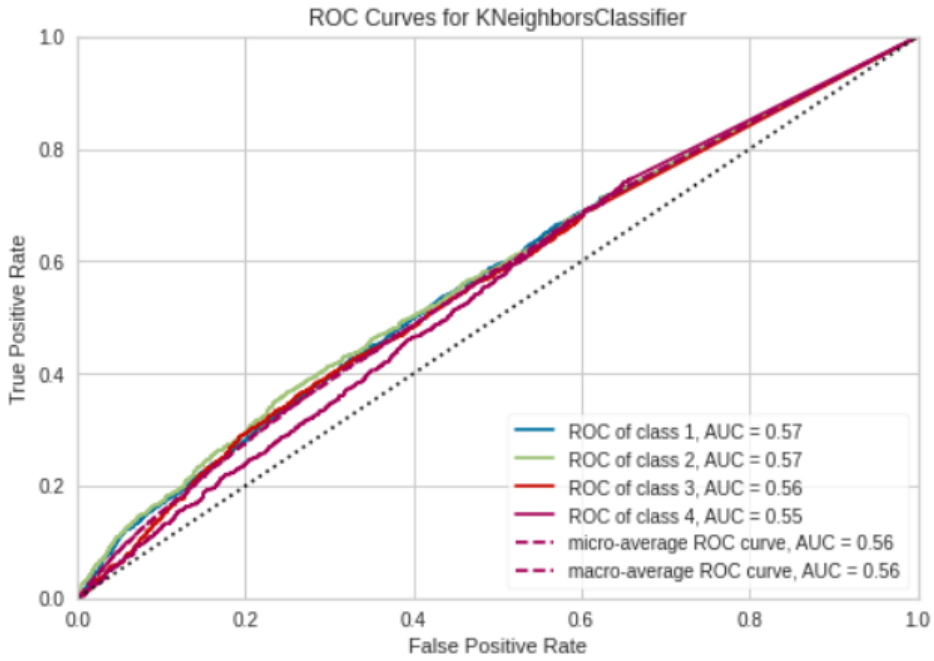
And then we plotted the learning rate curve:



As we can see KNN works with accuracy of 1 in train data but in test data the learning rate is very low and it is about 30%.

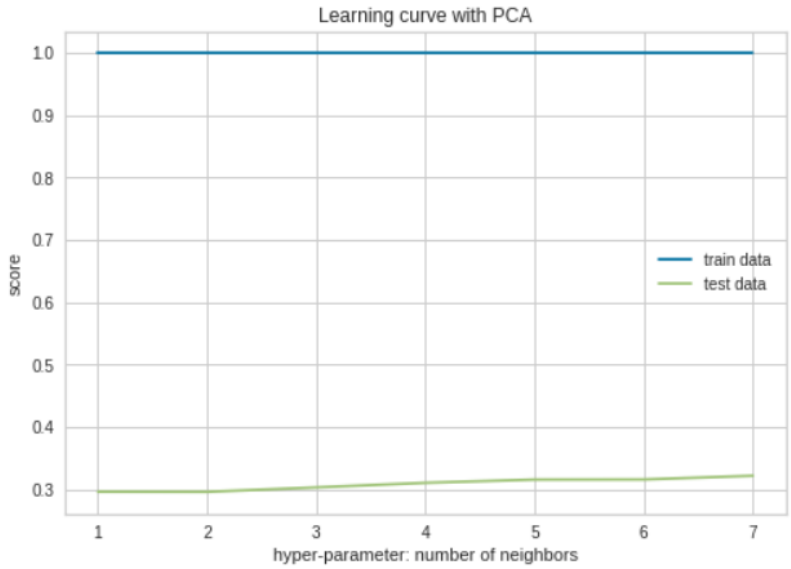
Now we want to determine another model. This time, everything is the same but we are using PCA method in order to reduce dimension:

The ROC curve for this model is as follow:



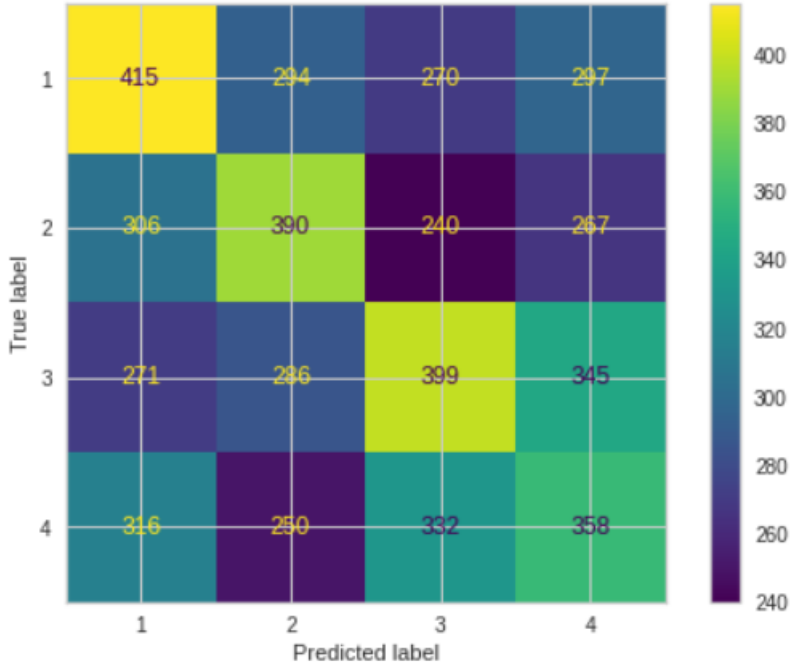
As we can see the accuracy has been become worse than the model without using PCA. Which means that sometimes, using PCA method would make the model worst and less accurate.

Learning rate for this method:



As it obvious in the model, using PCA has not affected the learning rate curve.

Also the confusion matrix is as follow:



as we can see, the confusion matrix colors is so close to the previous model in this different that it has less wrong predicted data and the blue squares has become lighter which means it has become better.

Logistic Regression

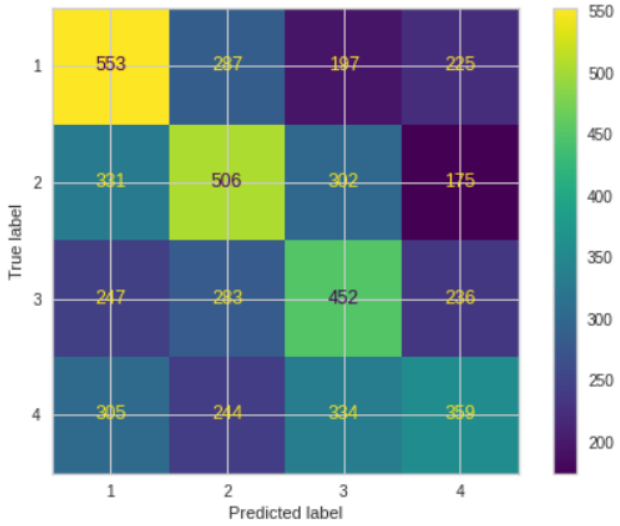
LR helps in finding the probability that a new instance belongs to a certain class. Since it is a probability, the outcome lies between 0 and 1. Therefore, to use the LR as a binary classifier, a threshold needs to be assigned to differentiate two classes.

In this part we are doing a classification based on emotions, without using PCA. Also, we are using the first method of feature extraction(348).

First, we do a grid-search on dataset to find the best hyper parameters:

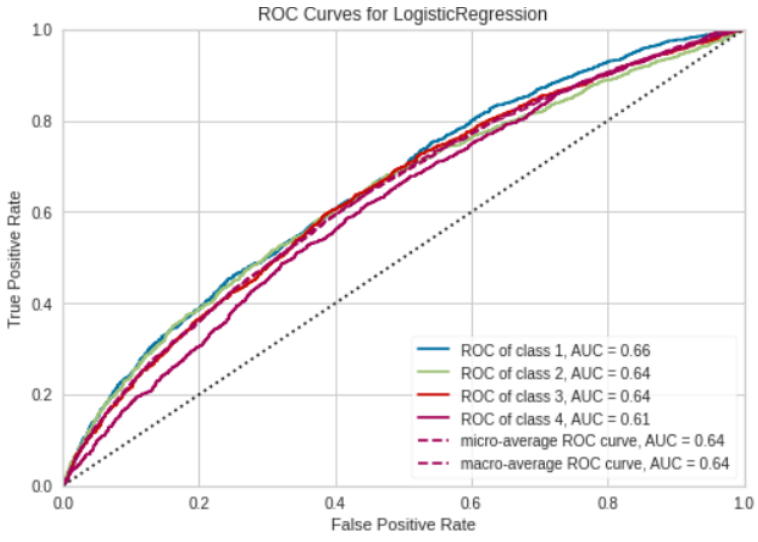
```
: solver = ['saga', 'sag', 'liblinear', 'lbfgs', 'newton-cg']
max_iter = [250, 500, 100]
penalty = ['l2']
for p in penalty:
  for s in solver:
    for i in max_iter:
      clf = LogisticRegression(penalty=p, solver=s, max_iter=i)
      print('penalty:', p, 'solver:', s, 'iteration:', i, 'accuracy:', clf.
fit(x_train, y_train).score(x_test, y_test))
```

Then we plot the confusion for this algorithm:



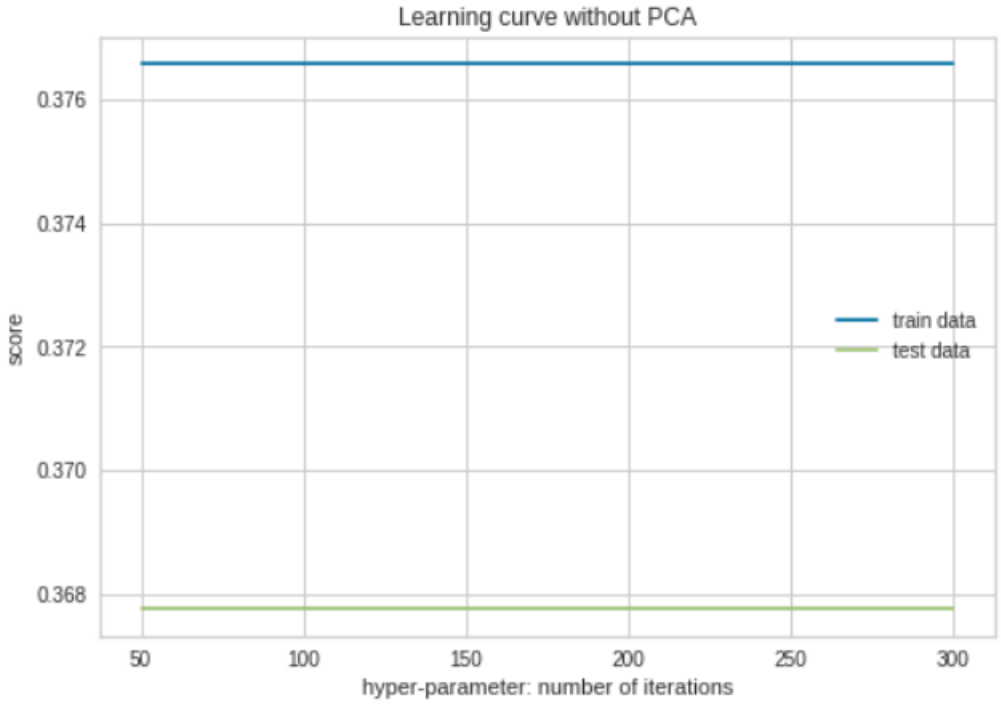
As we can see algorithm works well in angry emotion. Also the number of data that their emotion has predicted correctly are high too but the algorithm predict a lots of data in a wrong classification. The number of wrong data in each part in confusion matrix might be less than correct ones but in total they make up the half of data set and so the accuracy is near 50%.

And now we plot the ROC curve:



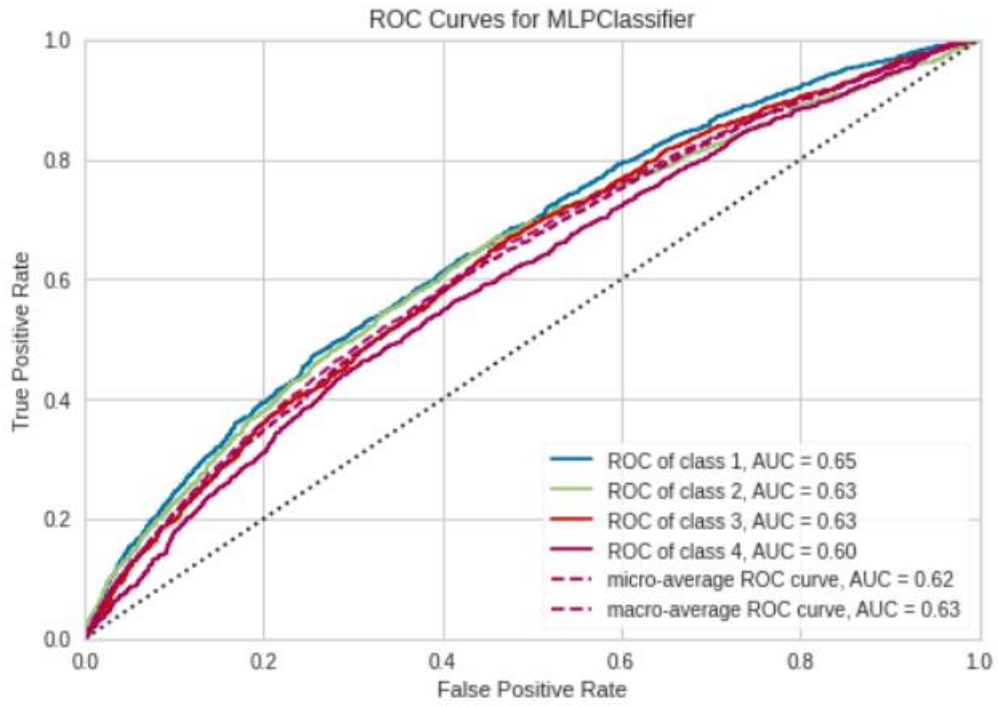
As we can see this algorithm is not that accurate but still it is above the doted line which means it better than the random classifier. Also, it seems that it is better than KNN. Which isn't surprising because the logistic regression is based on similarity and byes probability and so it should have the highest accuracy.

Learning rate curve for logistic regression:



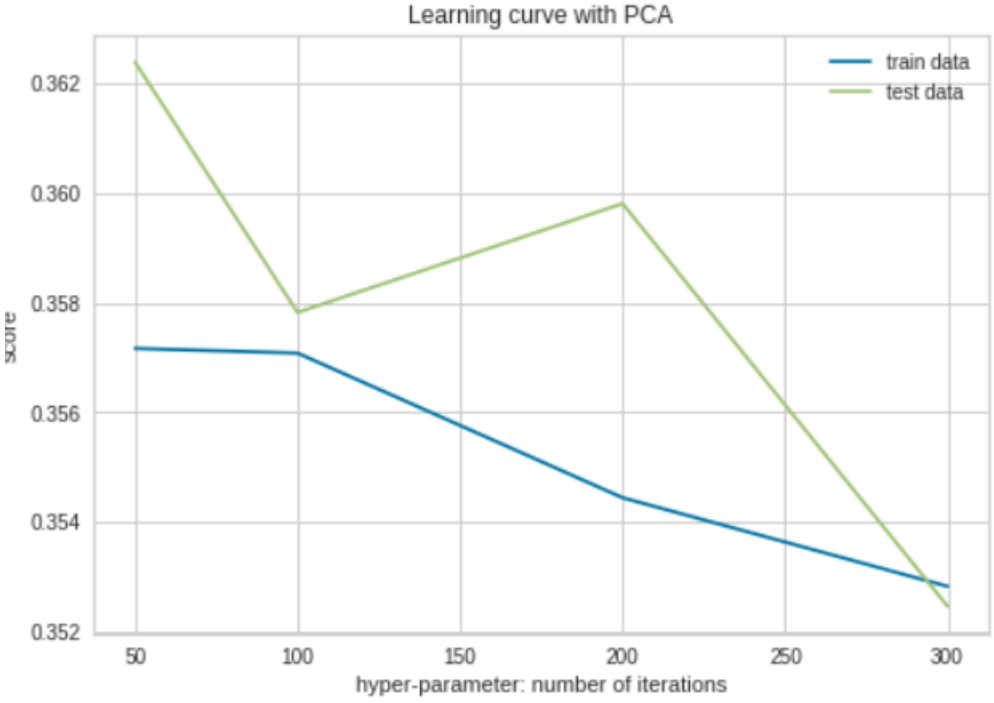
Based on the plot, both test and train data learning curve is stable and does not change. Also, they both have a lo learning rate but it does not change the reason for that is than in logistic regression, learning rate is one of the hyper parameters and it does not change during iteration.

Now we want to determine another model. This time, everything is the same but we are using PCA method in order to reduce dimension:



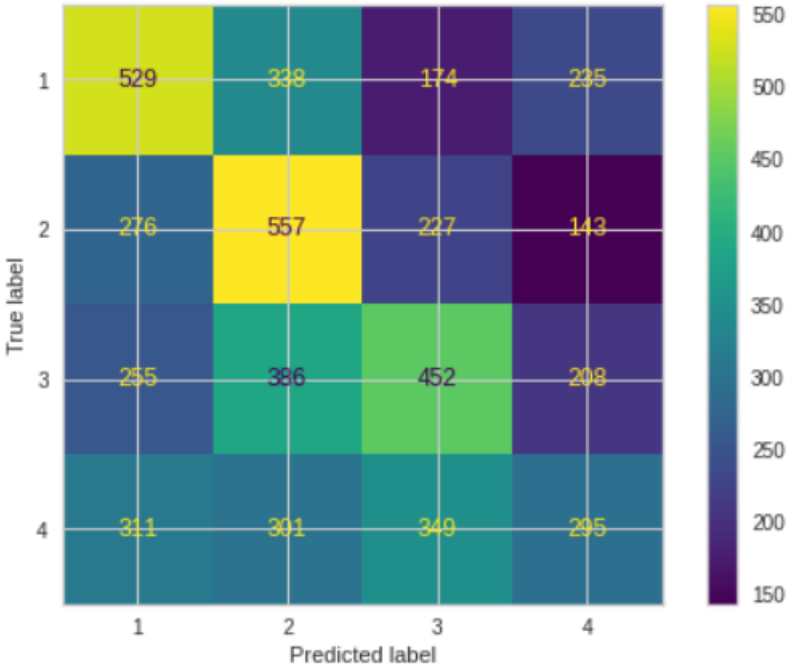
As we can understand, the ROC curve for this model has become better and get distinct from the middle line, which means the accuracy of the model has become better. As we can also figure from the curve, the average accuracy has reached 62%. So using PCA in MLP causes better accuracy.

Also here is the learning curve:



It can be see that as the hyper parameters get increases the accuracy would fall.

Also the confusion matrix is as follow:



The accuracy in angry emotions was higher in previous model but the accuracy of happy emotion is higher when using PCA.

MLP

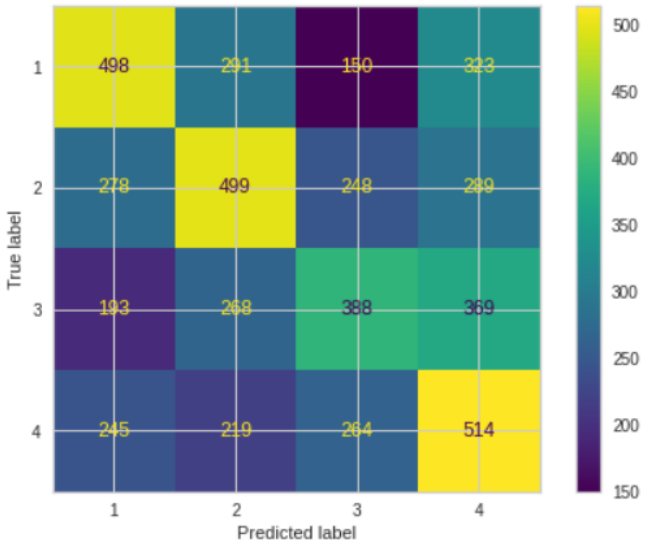
Neural Networks are a set of machine learning algorithms which are inspired by the functioning of the neural networks of human brain. Likewise, NN algorithms can be represented as an interconnected group of nodes. The output of one node goes as input to another node for subsequent processing according to the interconnection. Nodes are normally grouped into a matrix called layer depending on the transformation they perform.

In this part we are doing a classification based on emotions, without using PCA. Also, we are using the first method of feature extraction(348).

First, we do a grid-search on dataset to find the best hyper parameters:

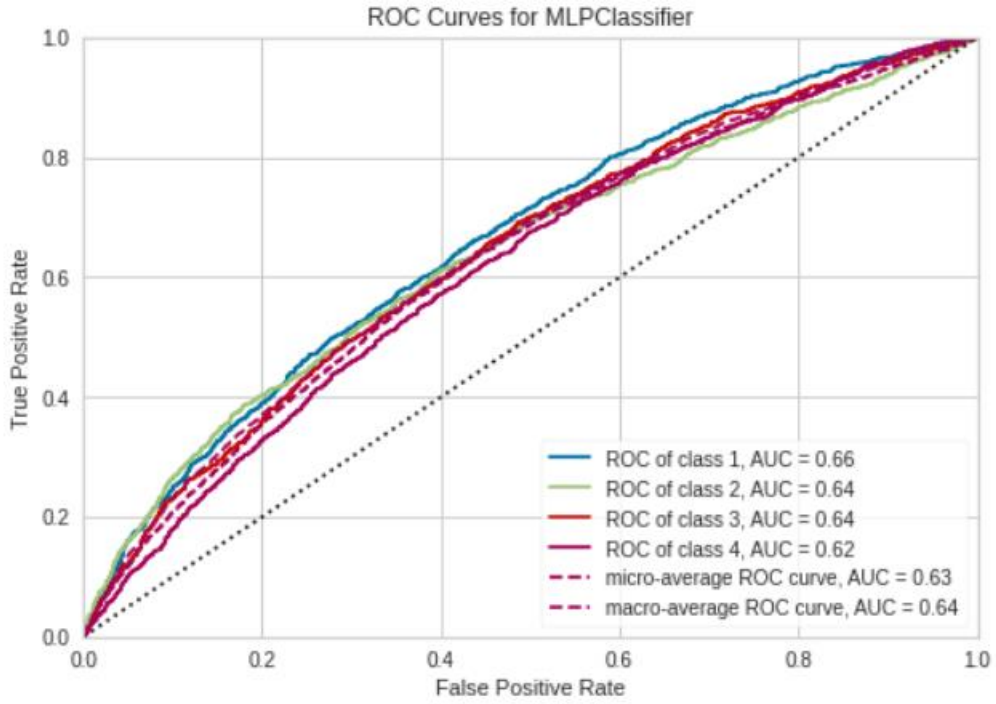
```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
hidden_layer_sizes = [100, 200, 300, 400, 500]
activation = ['relu', 'logistic', 'tanh']
solver = ['lbfgs', 'sgd', 'adam']
max_iter = [250, 500, 100]
for h in hidden_layer_sizes:
  for f in activation:
    for s in solver:
      for i in max_iter:
        clf = MLPClassifier(hidden_layer_sizes=(h,), activation=f, solver=
s, max_iter=i)
        print('hidden layer:', h, 'activation:', f, 'solver:', s, 'iteratio
n:', i, 'score:', clf.fit(x_train, y_train).score(x_test, y_test))
```

Then we plot the confusion for this algorithm:

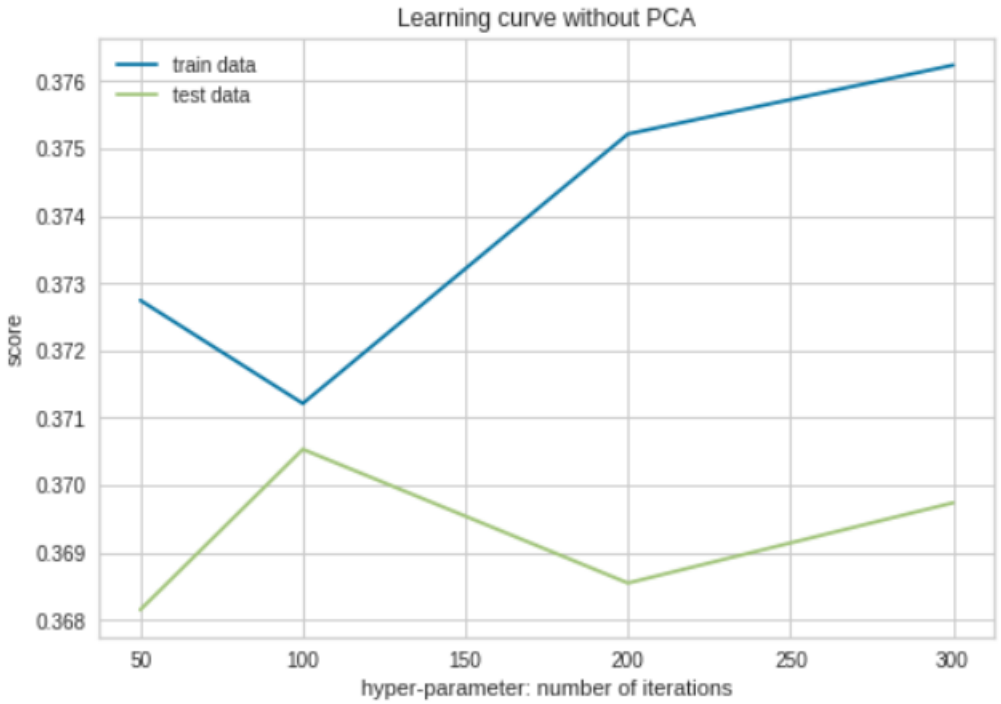


As we can see algorithm works well in neutral emotion(against other algorithms). Also, the number of data that their emotion has predicted correctly are high too but the algorithm predict a lots of data in a wrong classification. The number of wrong data in each part in confusion matrix might be less than correct ones but in total they make up the half of data set and so the accuracy is near 50%.

Then we plot the ROC curve:

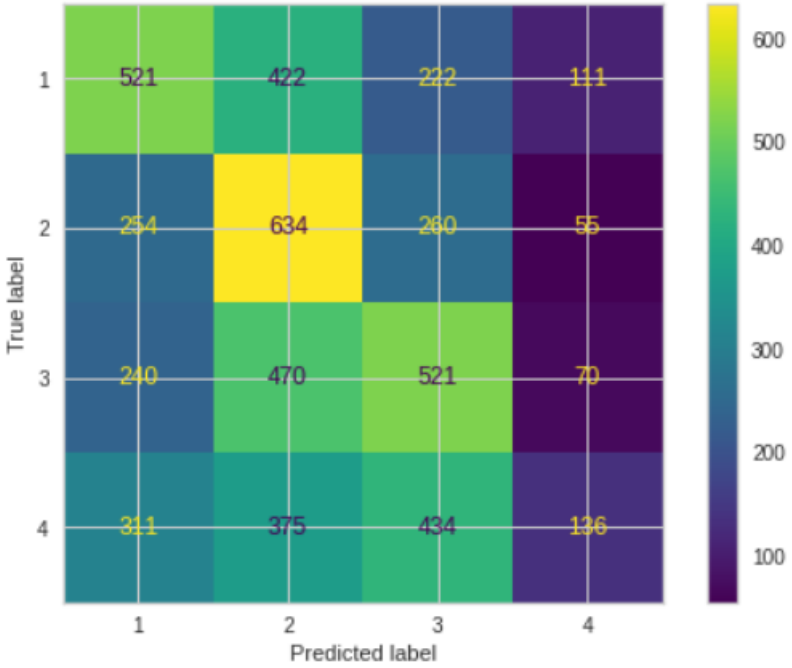


As we can see, class 1 has the best roc which means that angry data set has a higher accuracy. Also, the average ROC is 60% which is pretty good until now it means it can classify data in average with 60% confidence.
Learning curve :



As the hyper parameters get bugger the model would have a higher accuracy.

Also, here is the confusion matrix:



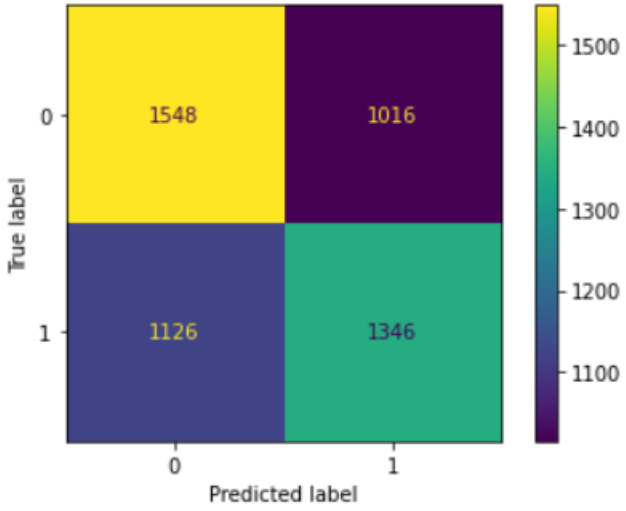
This model has become worst in all emotions, first of all the worst emotion prediction which is neutral, but as we can see it was better in previous version. Also, other emotions prediction has become worst to.

Gender classification

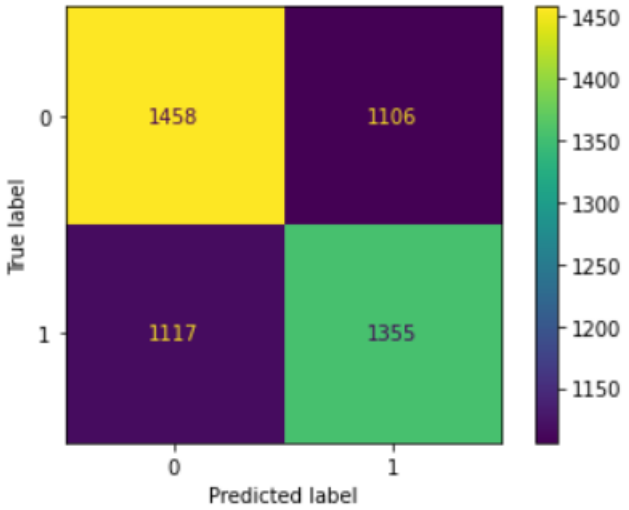
Without using PCA:

First we will plot all confusion matrix for each model and then compare all:

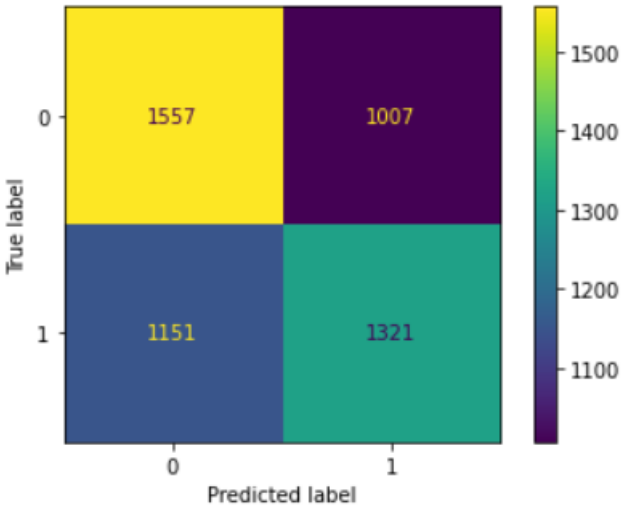
SVM:



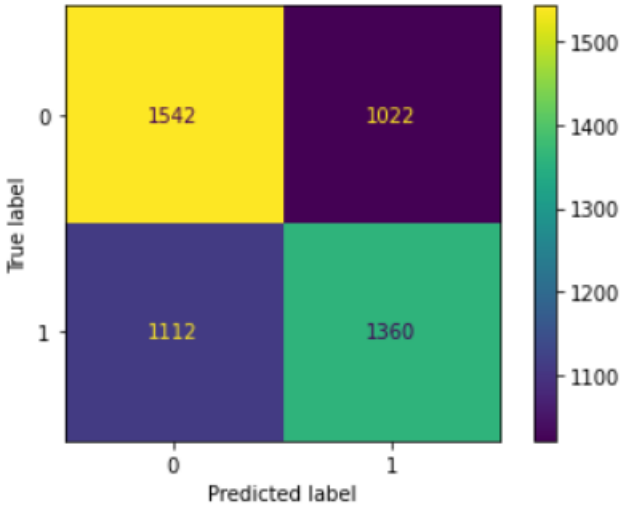
KNN:



Logistic regression:



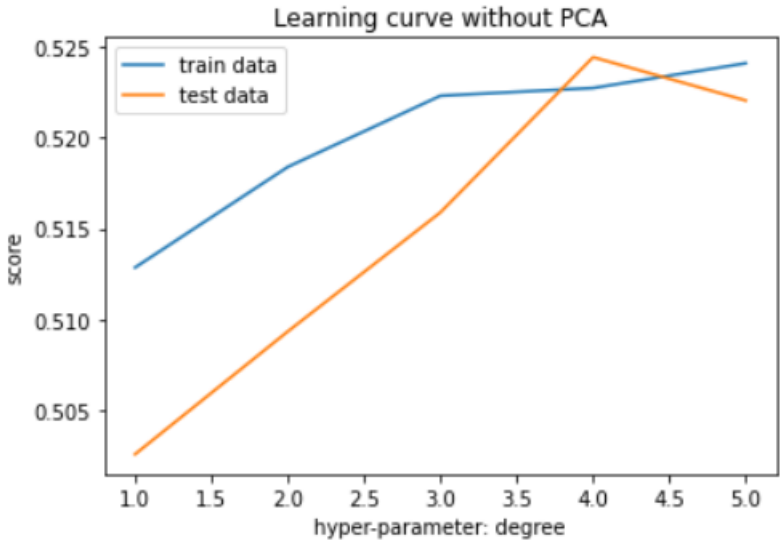
MLP:



as we can see the best model is KNN because it has predicted the most correct data and also lower amount of wrong predictions. After that SVM has the best confusion matrix and in the third place belongs to MLP. (based on the true positive-True negative- False positive and False negative)

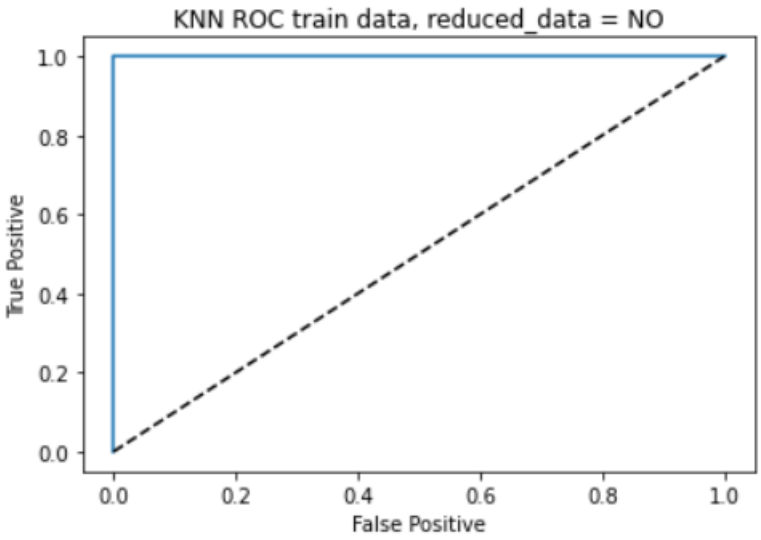
now we are going to compare their learning curve:

SVM:

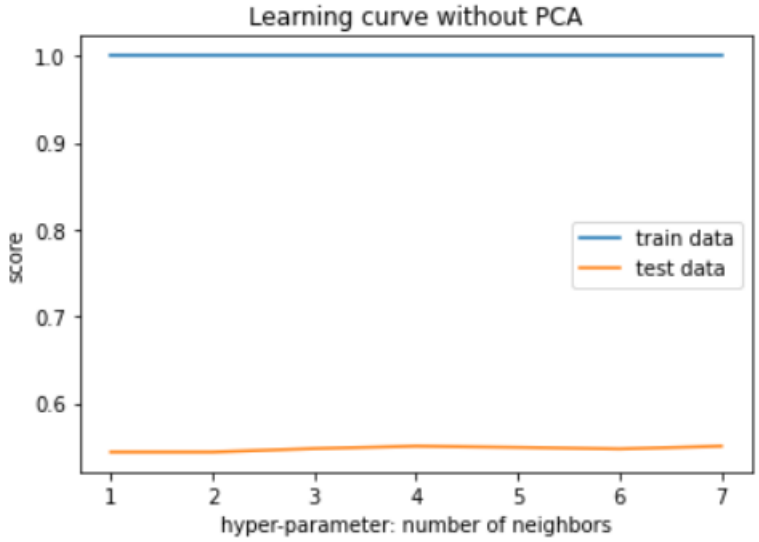


KNN:

ROC:



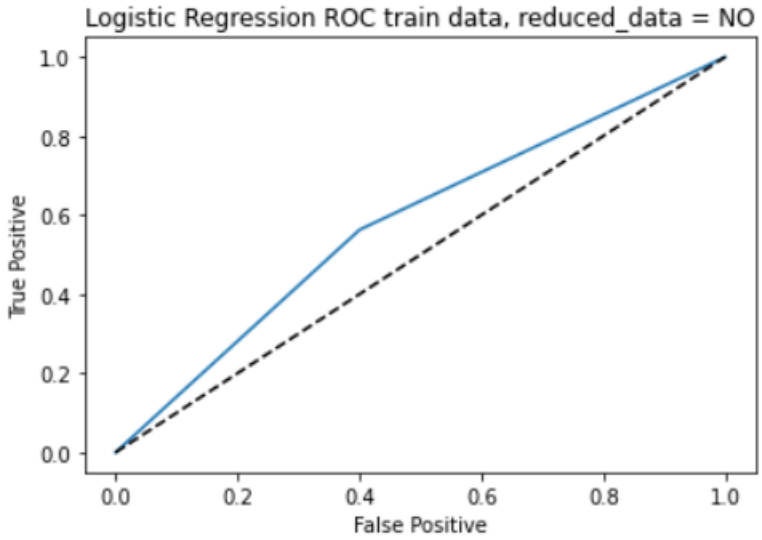
Learning rate:

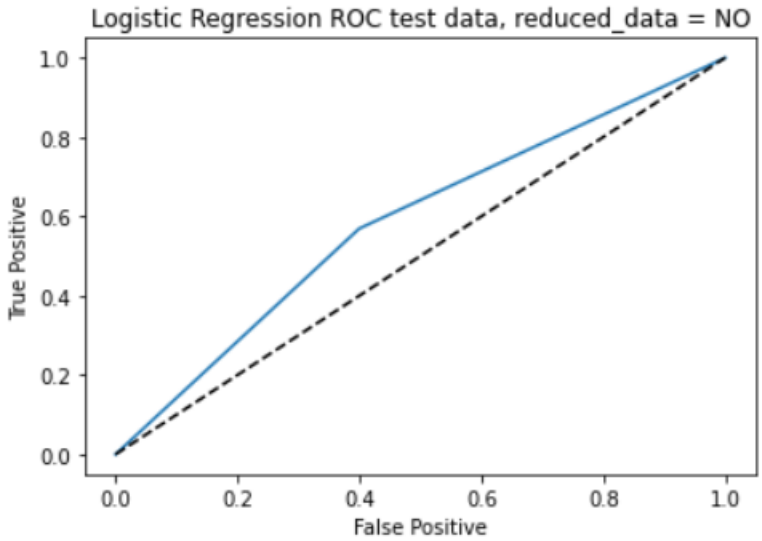


As it can be understood, the learning rate have a higher value in this method also the ROC on train data is not accurate enough.

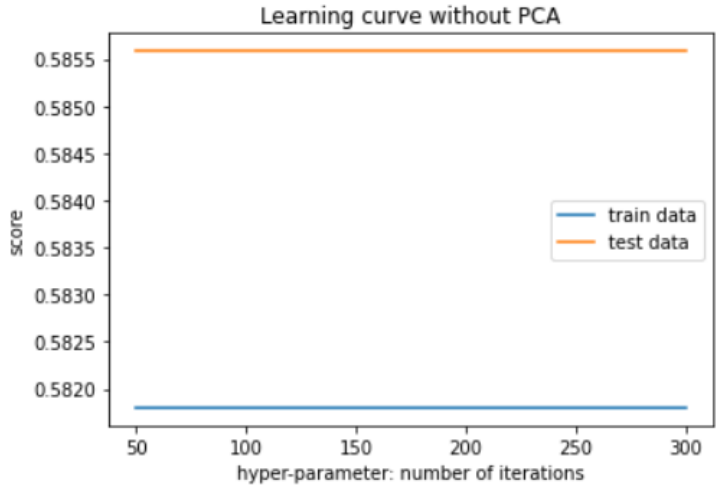
Logistic regression:

ROC:





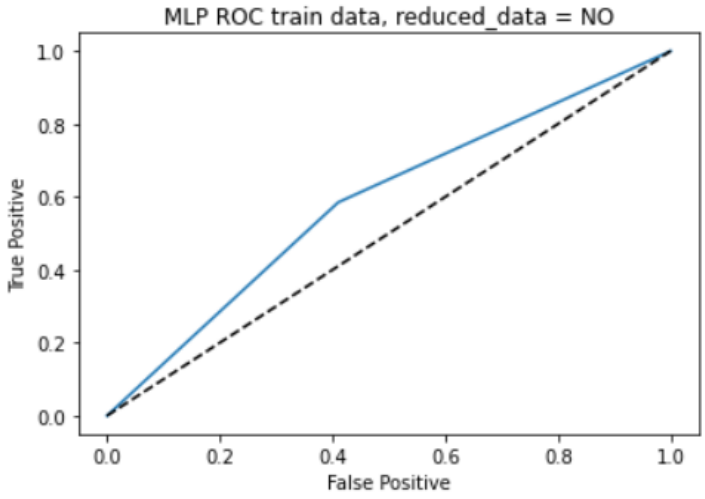
Learning curve:



As it can be understood, the learning rate have a higher value in this method also the ROC on train data is not accurate enough.

MLP:

ROC:

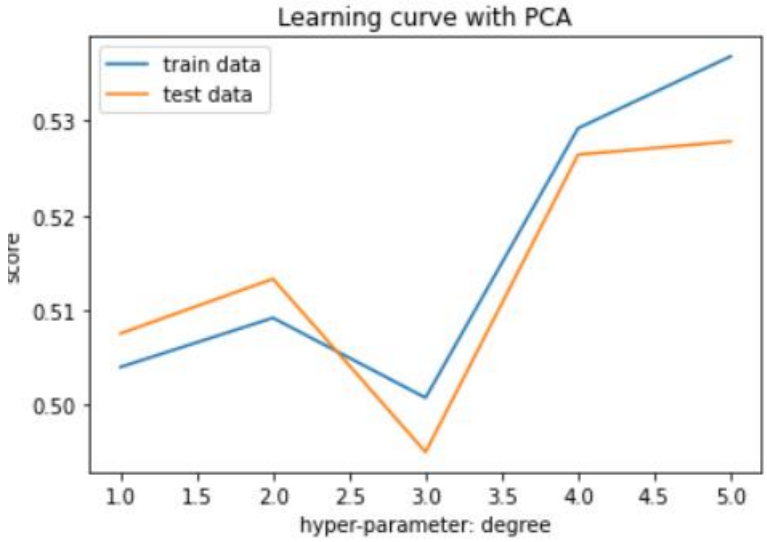


Learning curve:

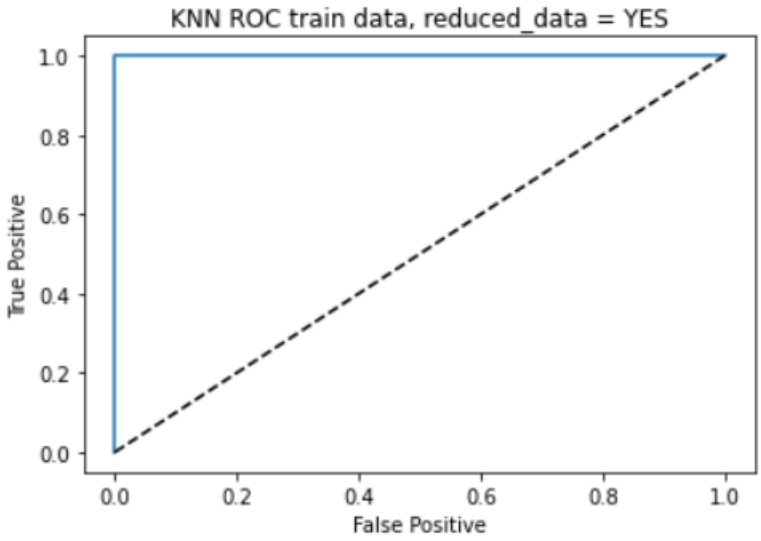
As it can be seen in the curves, the model has an average of 60% accuracy both in train and test data which is not a very high accuracy on this model and dataset. But the learning curve is increasing as the hyper parameters increases. Wich mean it might get better over a very high value for hyper parameters.

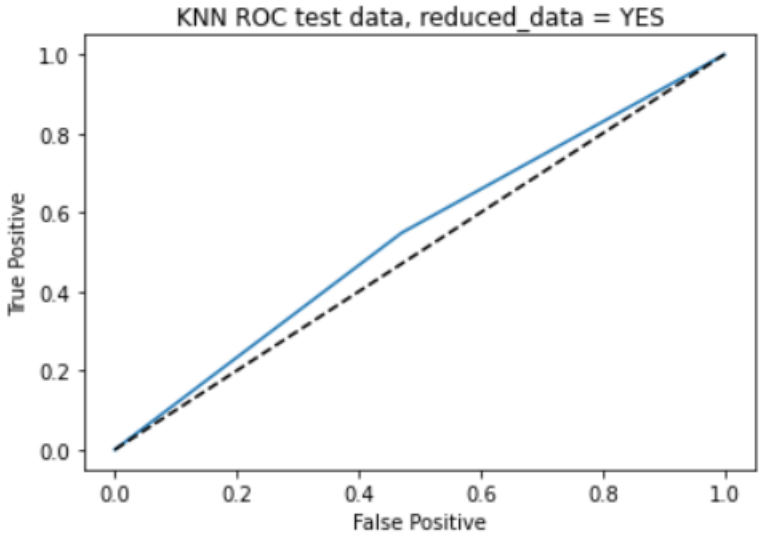
Now we want to check the models with the usage of PCA:

SVC:

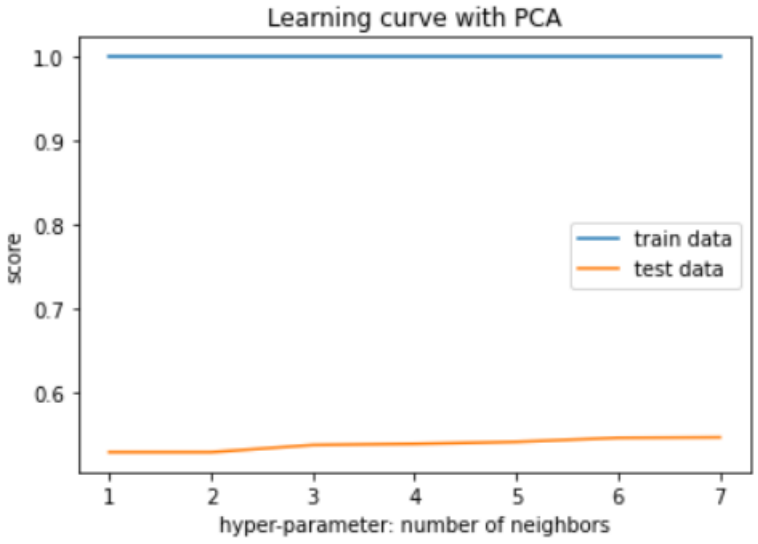


KNN:



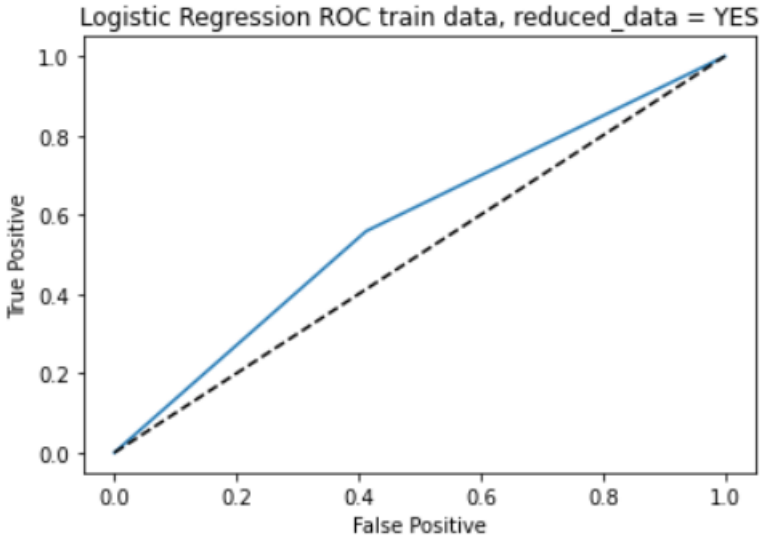


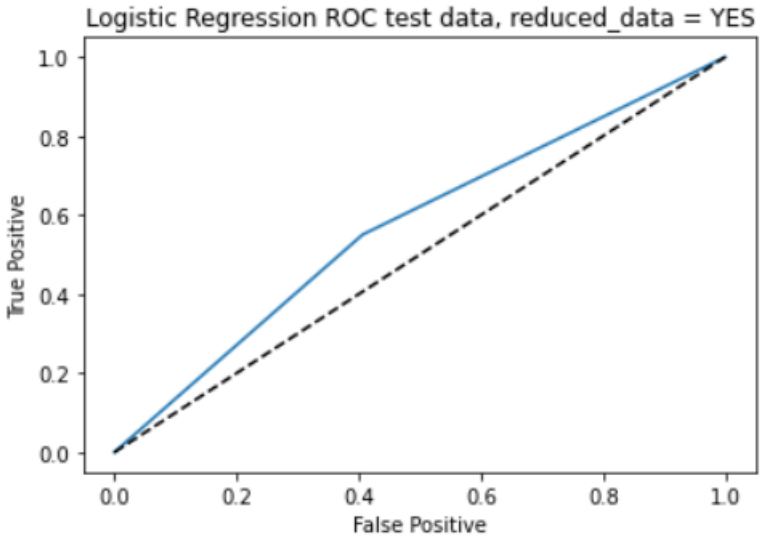
Learning curve:



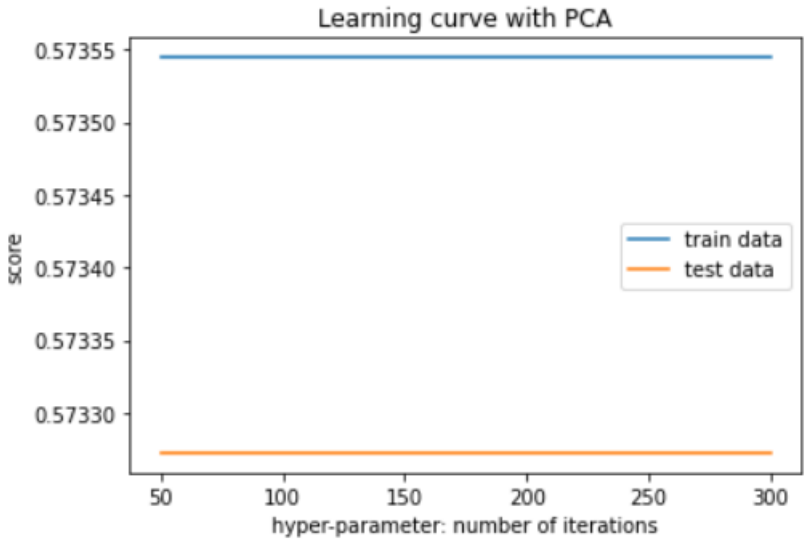
as we can see, in SVM, using PCA caused a better accuracy in higher values for hyper parameter than not using PCA. Also in the accuracy of the model has decreased when we used PCA and the learning rate is not getting better by increasing the hyper parameters wich mean using PCA for KNN caused a bad result.

Logistic regression:



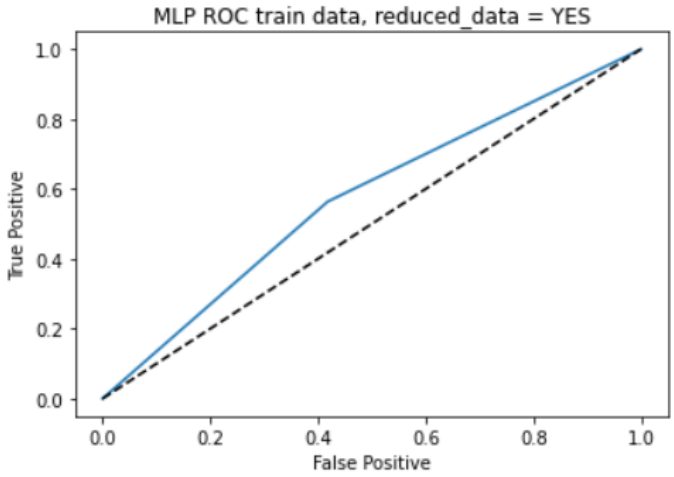


Learning curve:


Learning curve with PCA
0.57355
0.57350
0.57345
0.57340
0.57335
0.57330
score
train data
test data
50
100
150
200
250
300
hyper-parameter: number of iterations

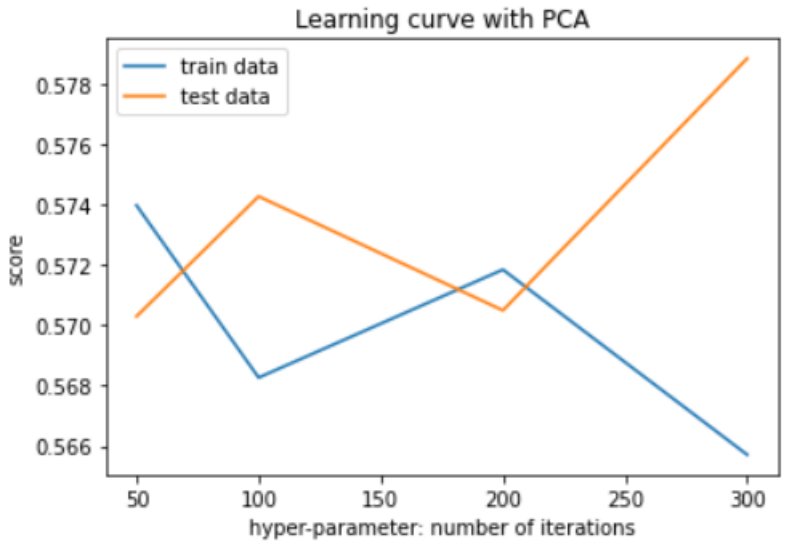
MLP:

Roc:


MLP ROC train data, reduced_data = YES
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0


MLP ROC test data, reduced_data = YES
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

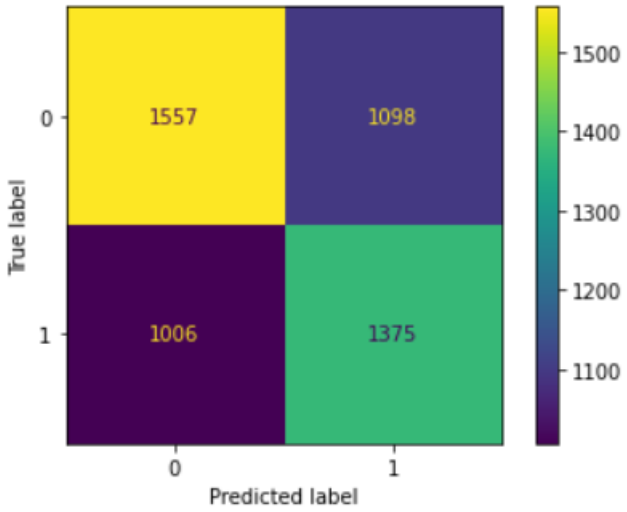
Learning rate:



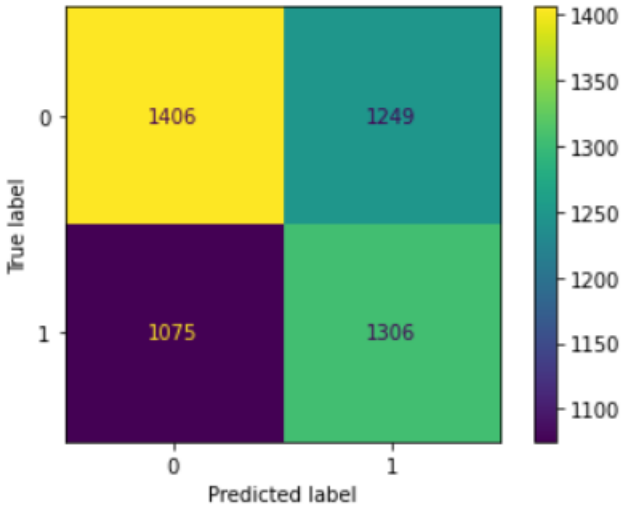
As we can see, the learning rate have gone higher for test data but it fall sfor the train data. Which means the model is being underfit.

Confusion matrixes:

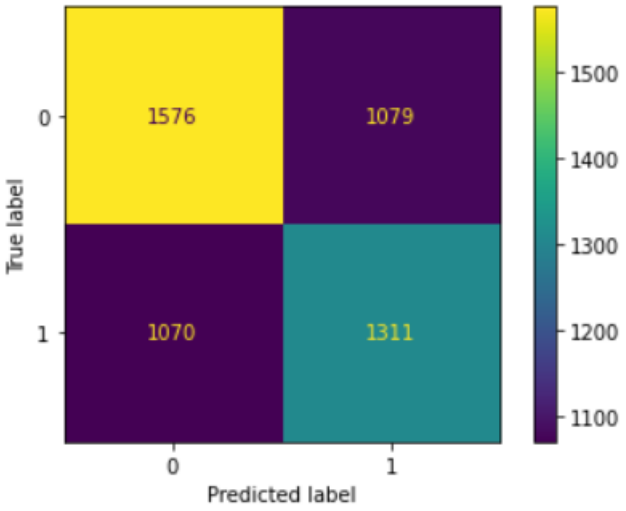
SVM:



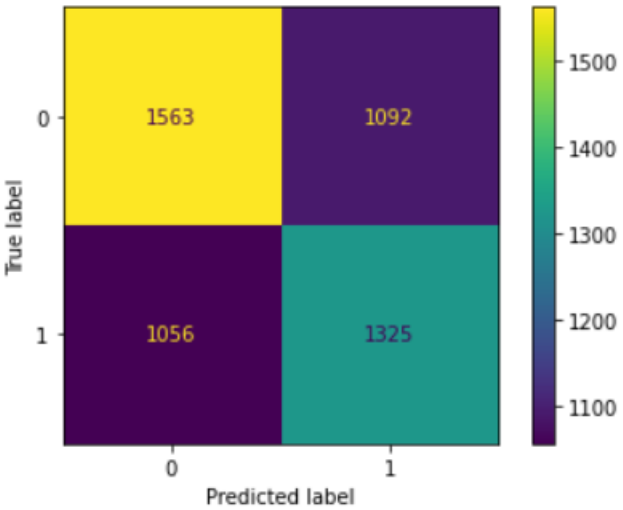
KNN:



Logistic Regression:



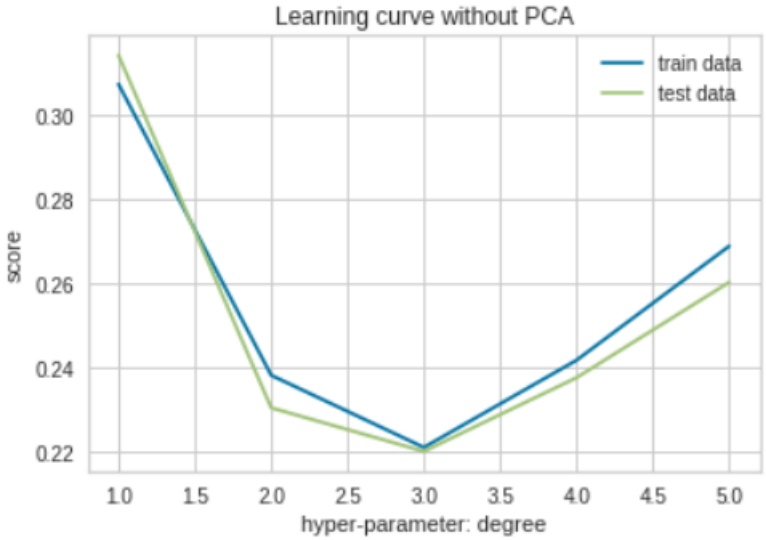
MLP:



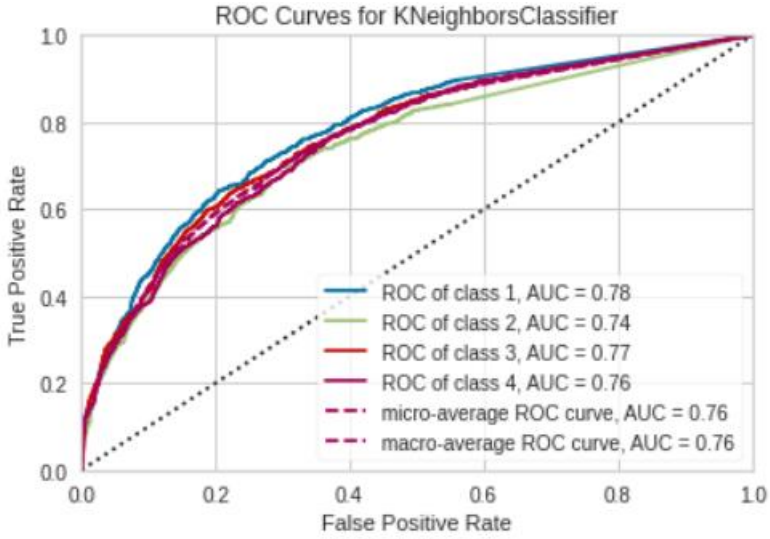
If we want to compare all these models when using PCA and by using the first method of feature extraction, the logistic regression has the highest accuracy, then the MLP and after that SVM and KNN. This is because in binary classification, LR has the highest accuracy and other methods would not work as well as this model. But as we saw in emotion classification, LG and MLP wouldn't work well as other models.

Now it's time for changing the feature extraction method:

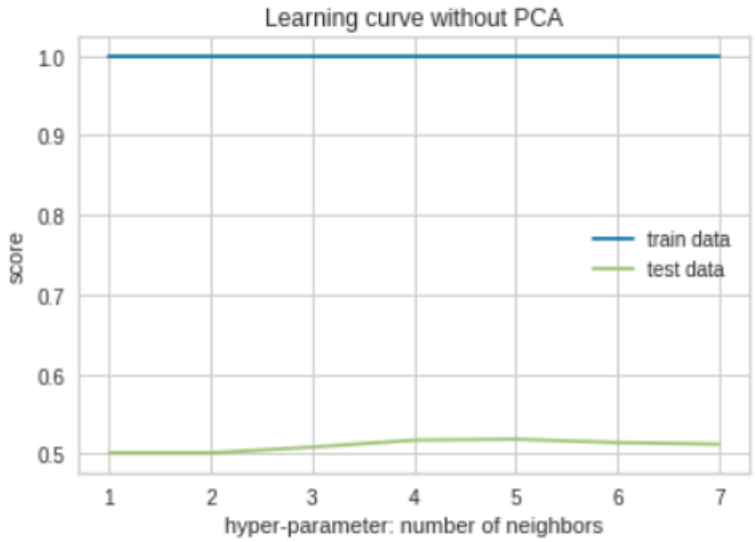
SVM:



KNN:

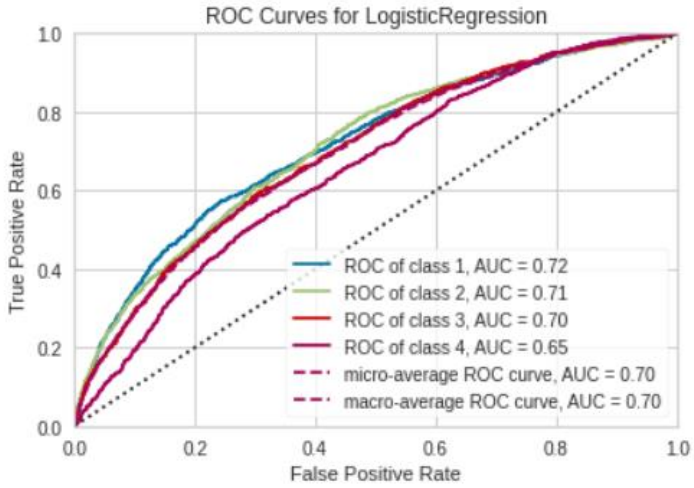


Learning curve:

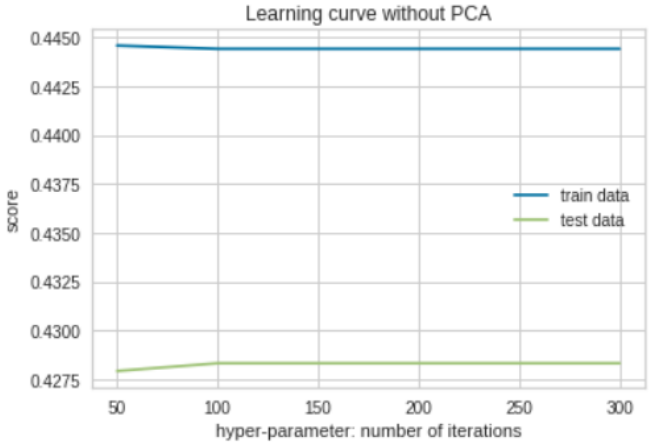


As we can see the accuracy and the ROC plot has improved so much. Using larger number of features caused a very high accuracy in our model.

Logistic regression:

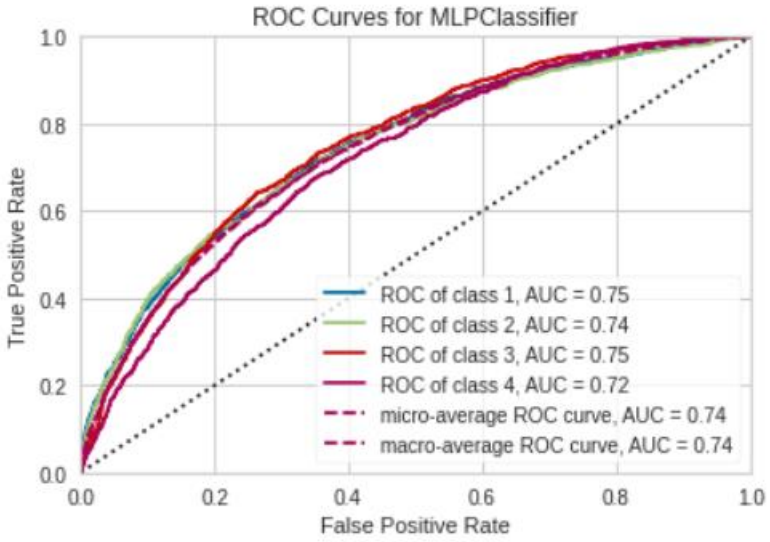


Learning curve:

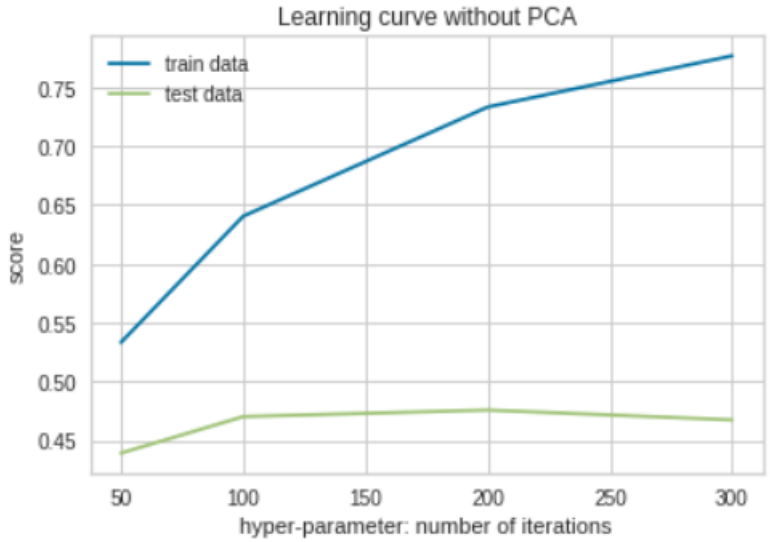


Also in logistic regression we can see that the second method has caused a very high accuracy.

MLP:



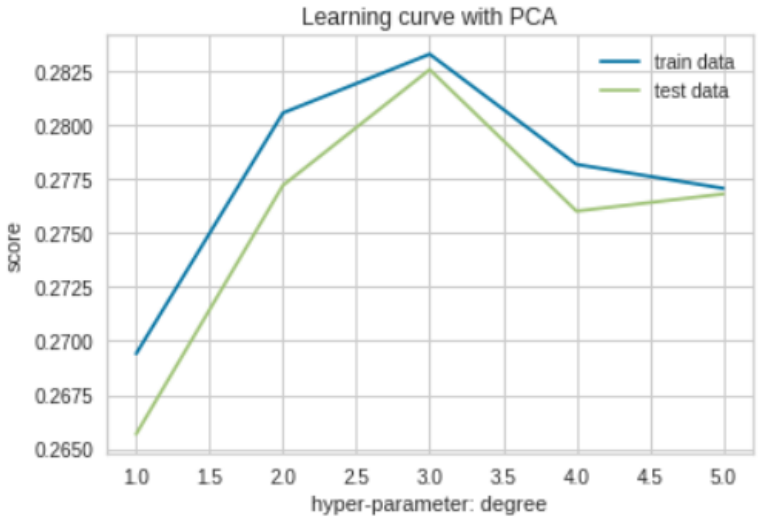
Learning curve:



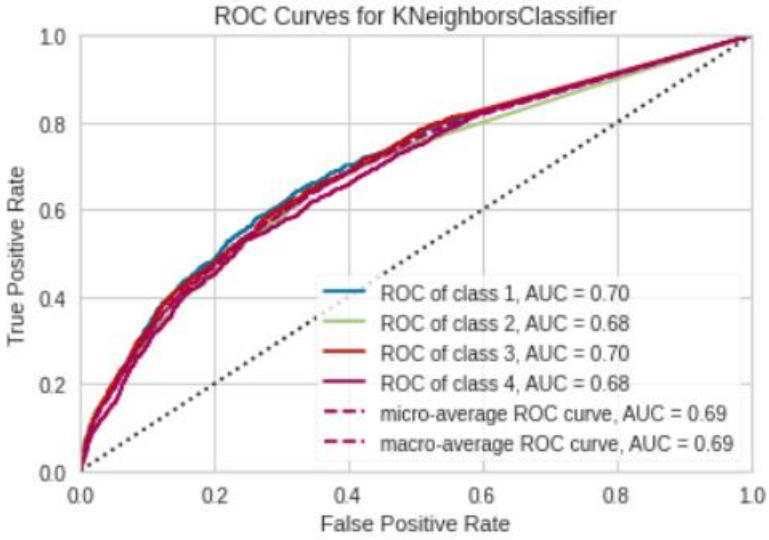
As we can wee in mlp the accuracy has also increased and also the learning curve has become better.

Now we will use the PCA method:

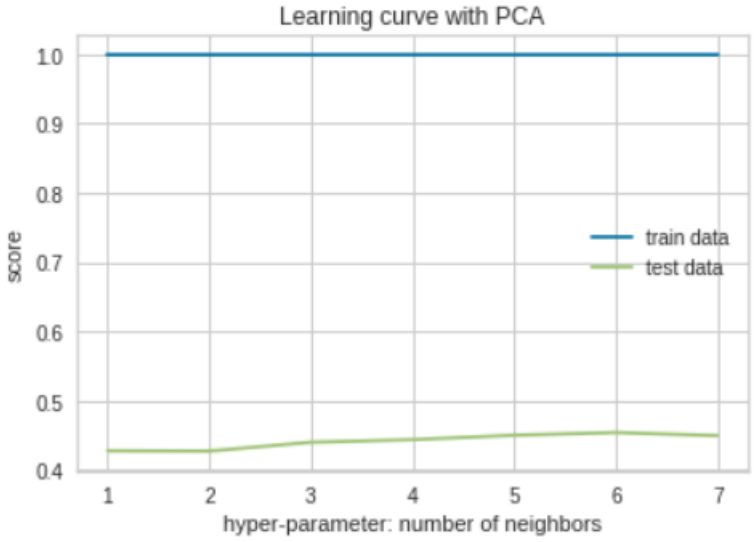
SVM:



KNN:

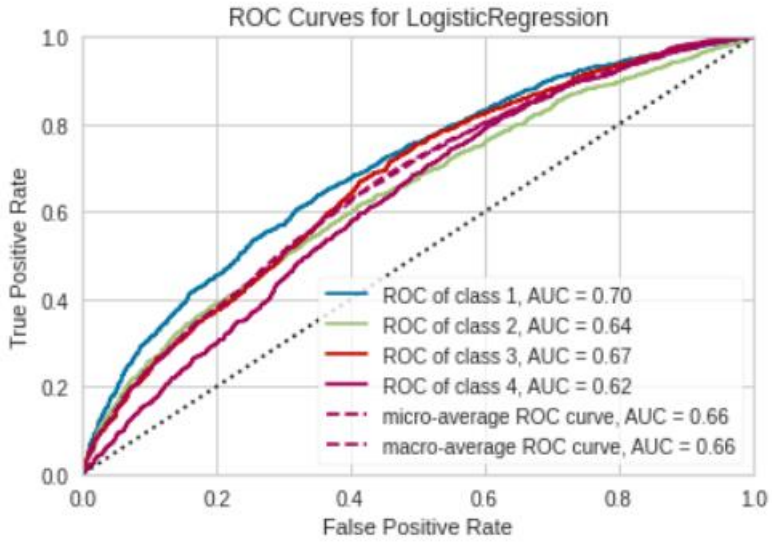


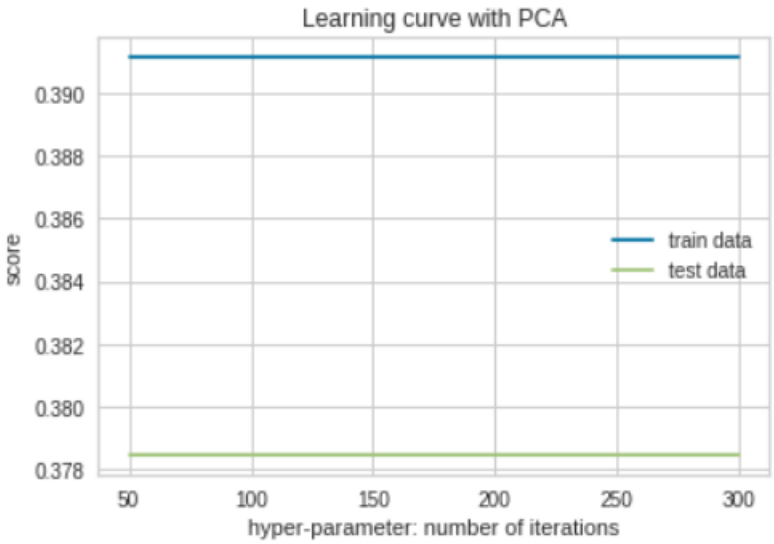
Learning curve:



As we can see by using the PCA method the accuracy for KNN decreases. Although it had better True Positive rate, in general the accuracy of model has decreases. In SVM the accuracy of model is convex and it has a global optimum and its limited.

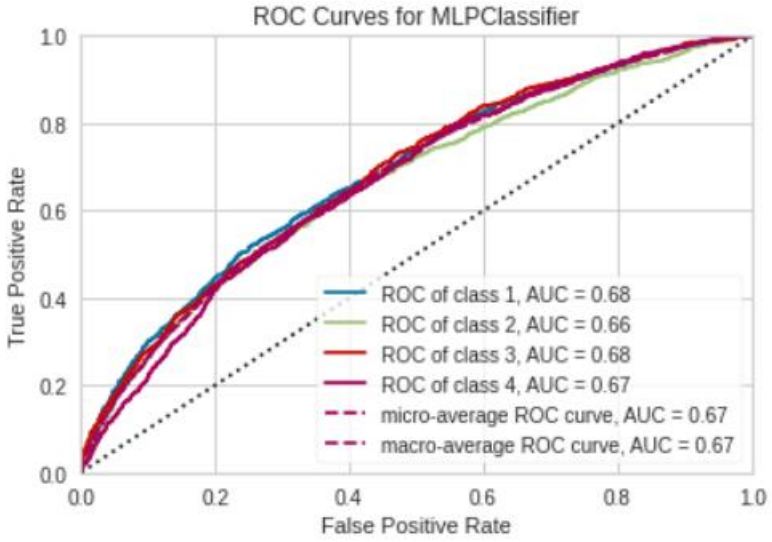
Logistic regression:

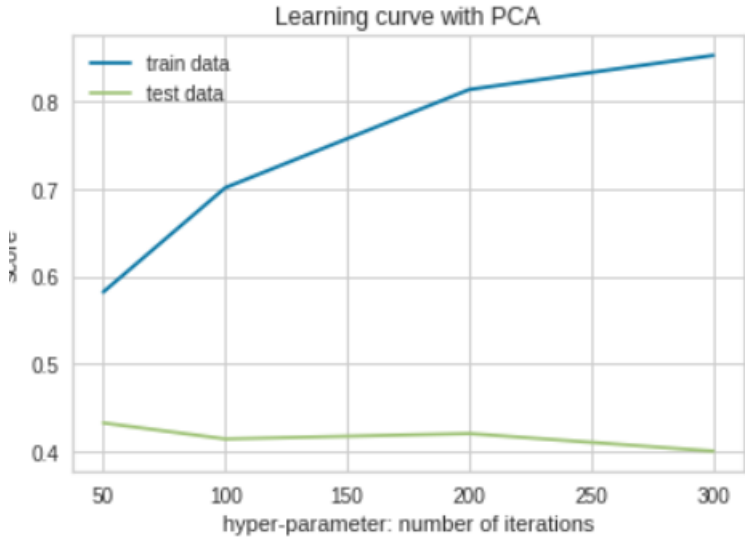




Using PCA on LR also cause a decrease in the accuracy of the moded and still the learning rate is constant and it does not change.

MLP:

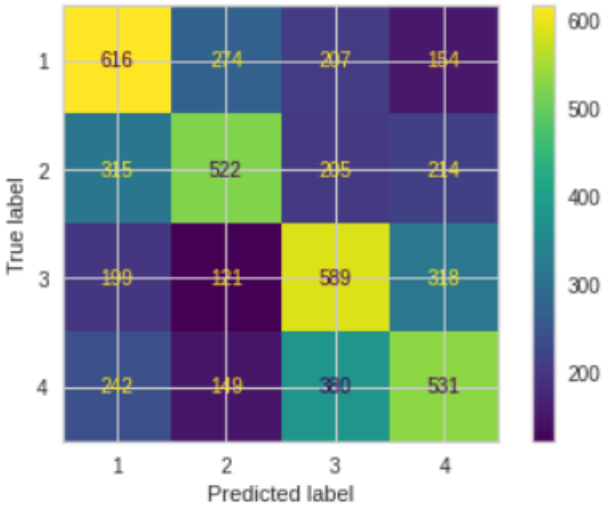




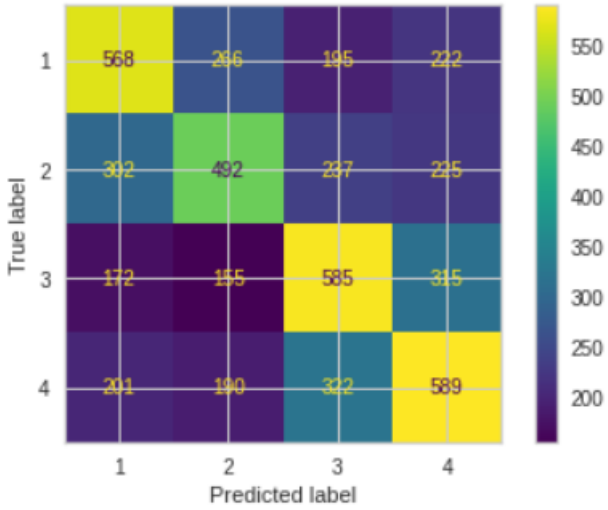
As we can see the accuracy on test data in MLP by using PCA is very low and it will decrease as the hyper parameters grow.

Confusion matrixes:

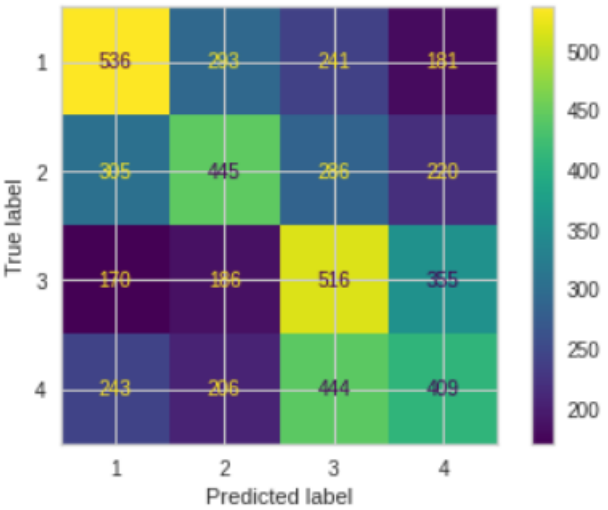
SVM:

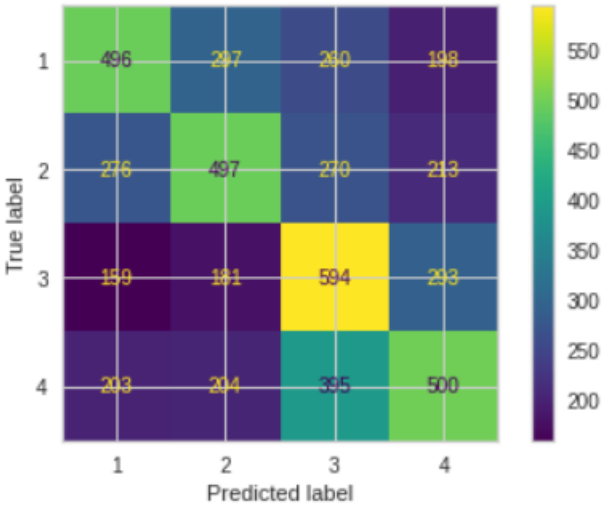


KNN:



Logistic regression:



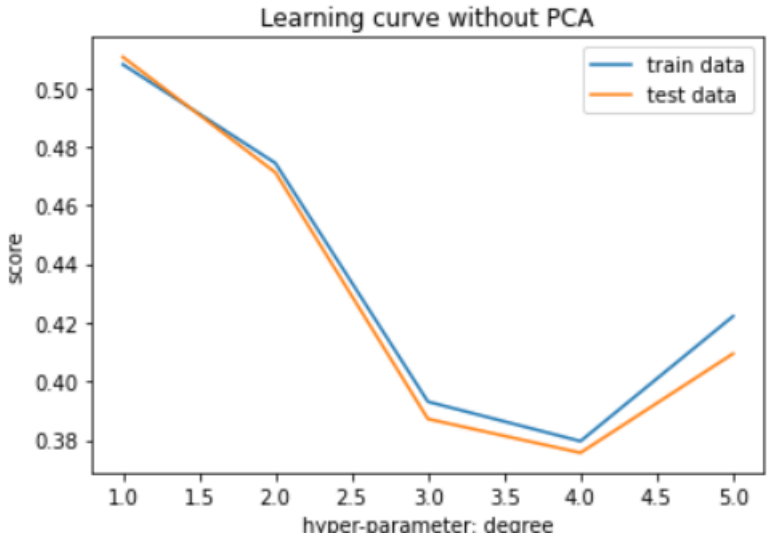


If we compare the two set of confusion matrix in using PCA and without using PCA we would see that the wrong answers has increased and the true data predictions has become less. As we expected, dimension reduction would reduce the accuracy in most cases.

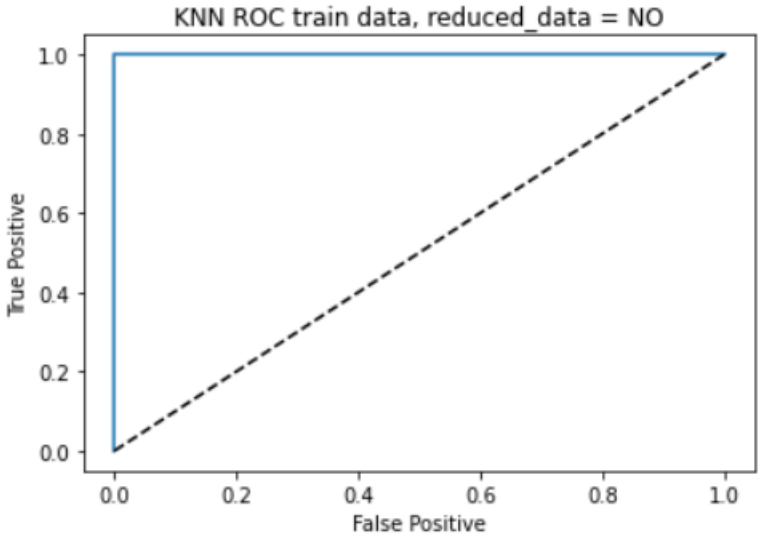
Now we will check the second feature extraction method for gender prediction.

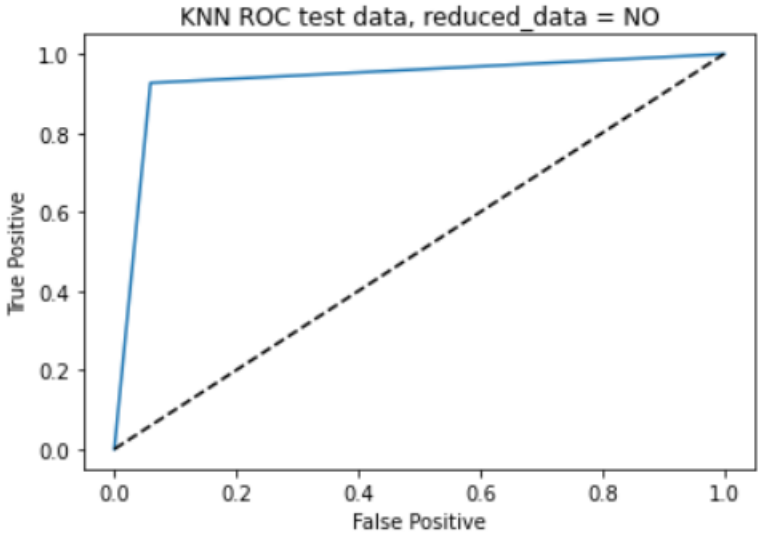
First we implement the models without using PCA:

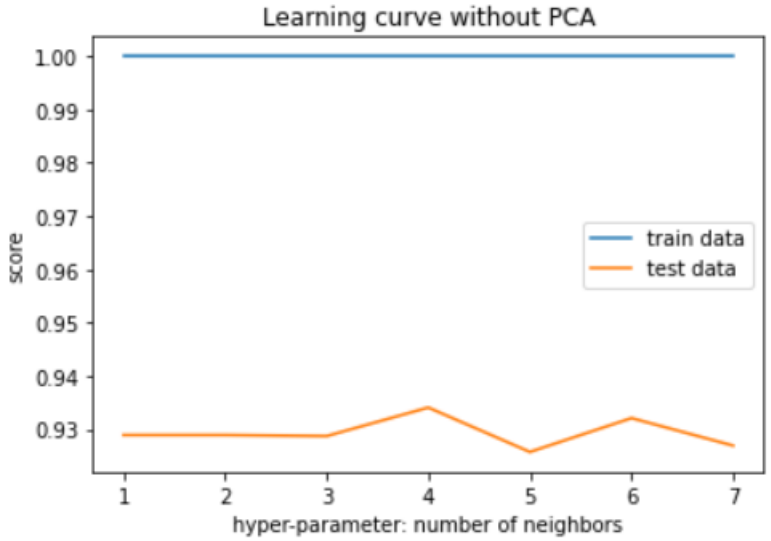
SVM:



The ROC curve:

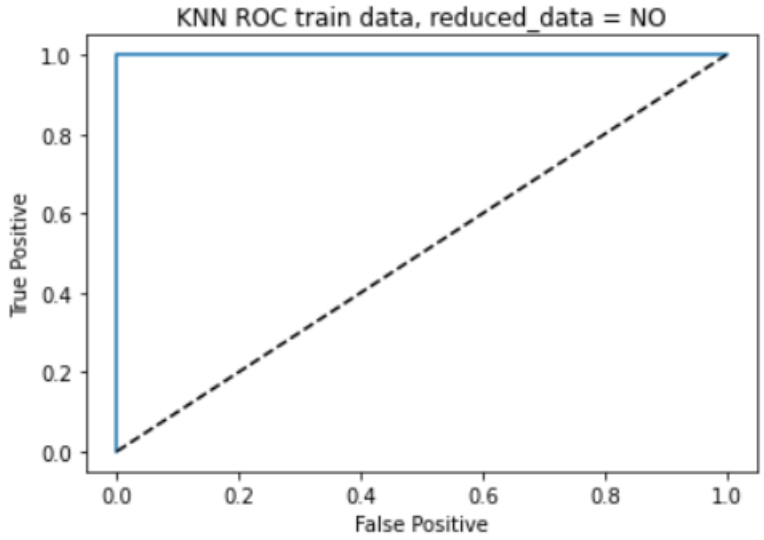


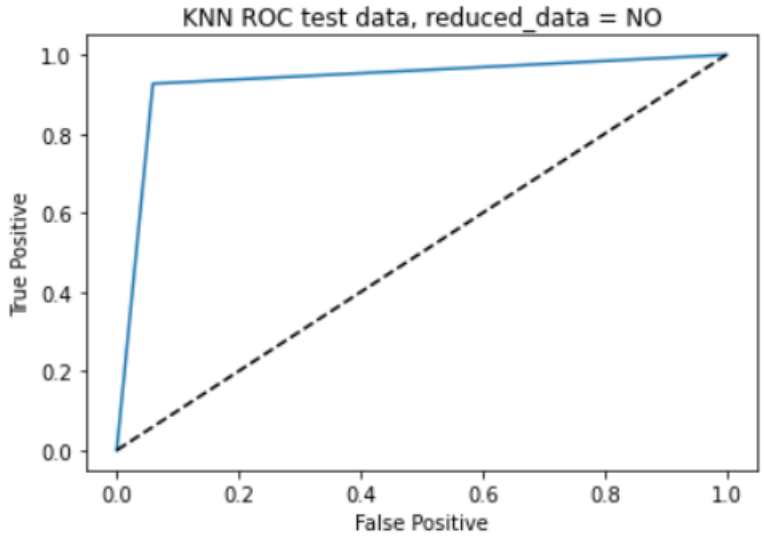


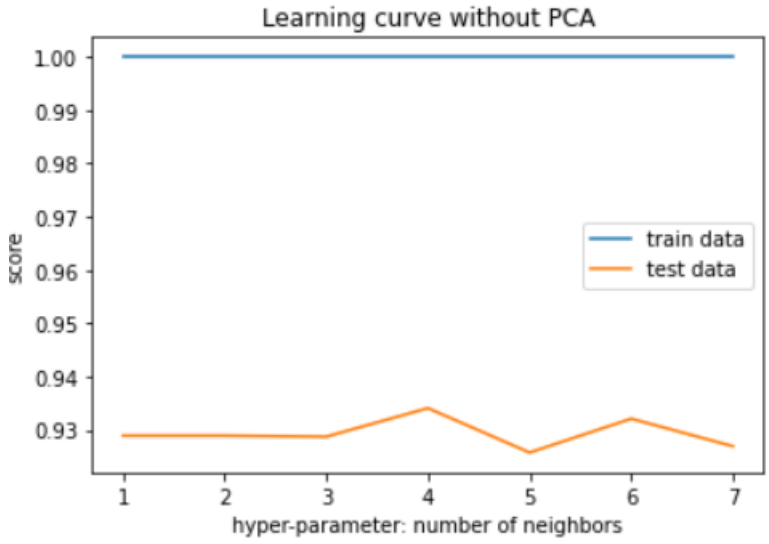


Using this method on gender prediction caused a nearly optimum classifier.

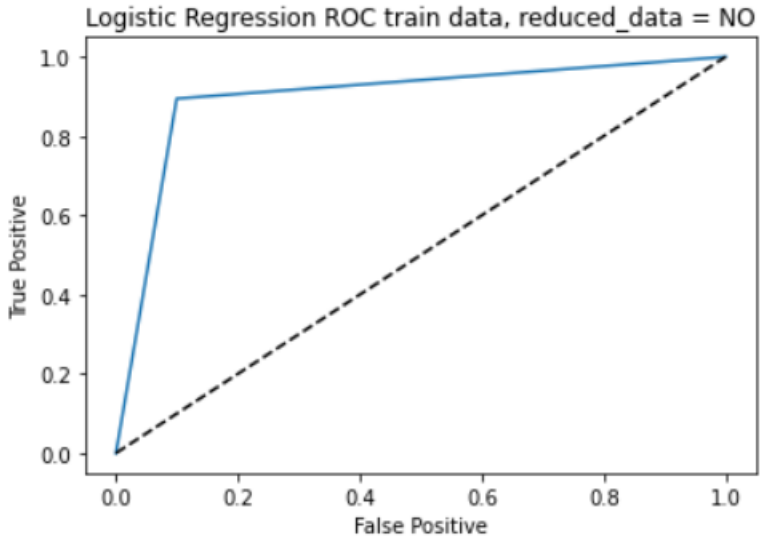
KNN:



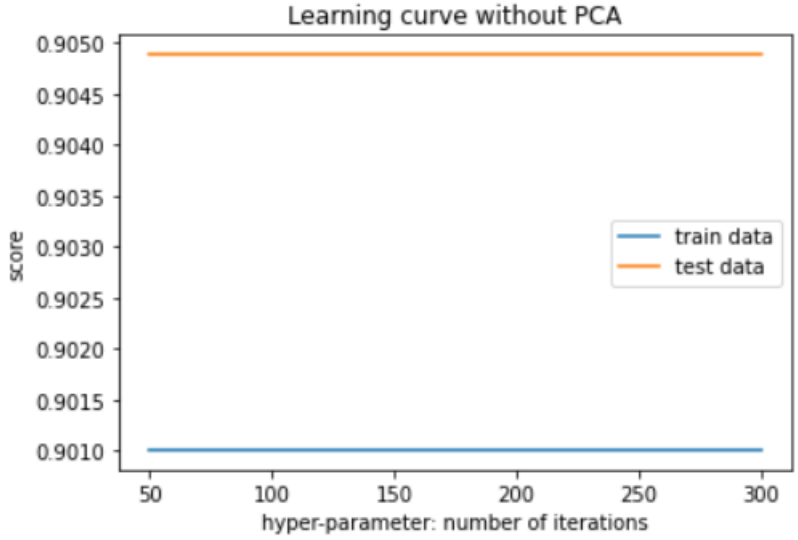




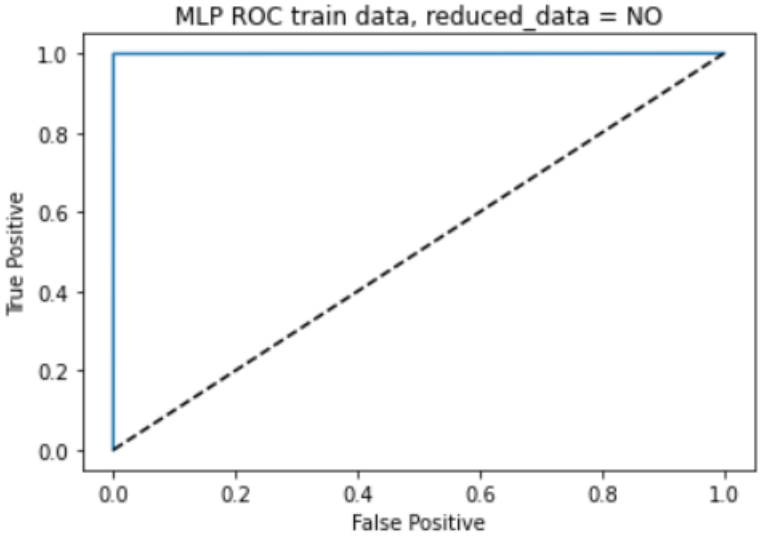
Logistic regression:

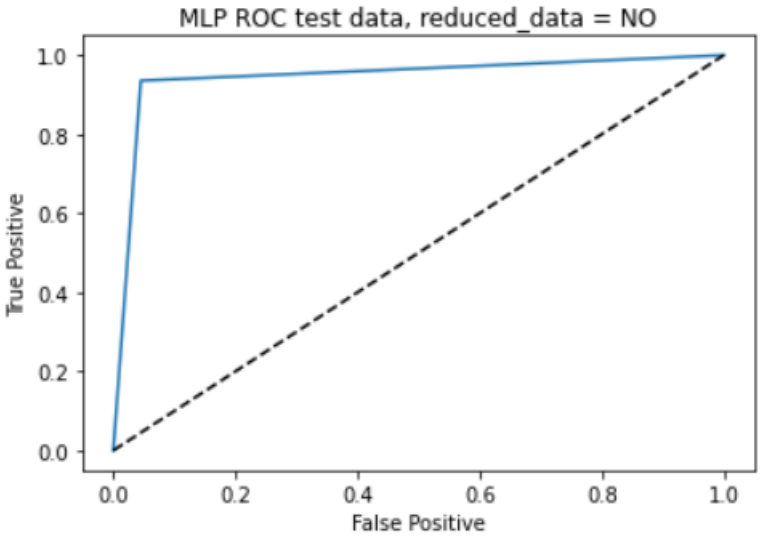

Logistic Regression ROC train data, reduced_data = NO
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

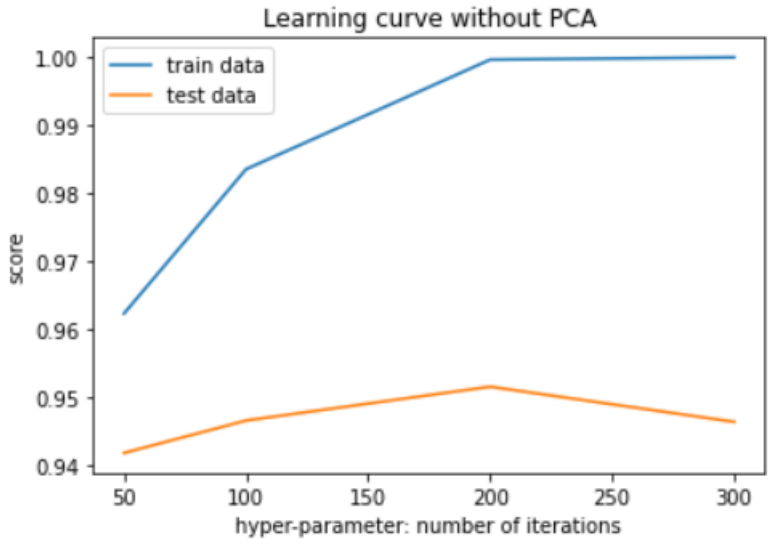

Logistic Regression ROC test data, reduced_data = NO
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0


Learning curve without PCA
score
hyper-parameter: number of iterations
train data
test data
0.9050
0.9045
0.9040
0.9035
0.9030
0.9025
0.9020
0.9015
0.9010
50
100
150
200
250
300

MLP:

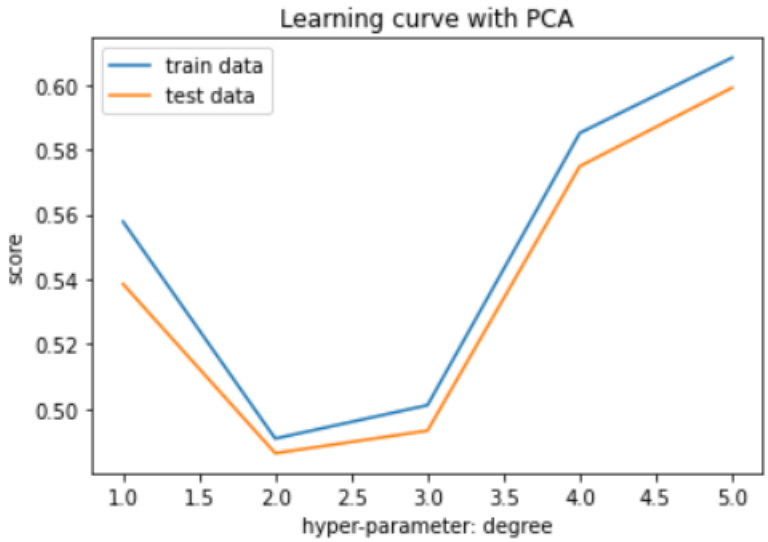

MLP ROC train data, reduced_data = NO
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0


MLP ROC test data, reduced_data = NO
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

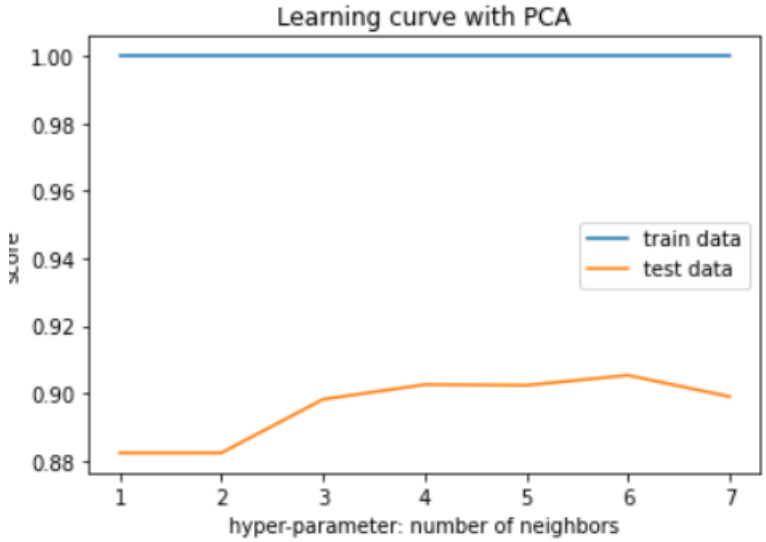

Learning curve without PCA
train data
test data
score
hyper-parameter: number of iterations
1.00
0.99
0.98
0.97
0.96
0.95
0.94
50
100
150
200
250
300

If we compare the two set of confusion matrix in using PCA and without using PCA we would see that the wrong answers has increased and the true data predictions has become less. As we expected, dimension reduction would reduce the accuracy in most cases.

Now we would use the PCA:

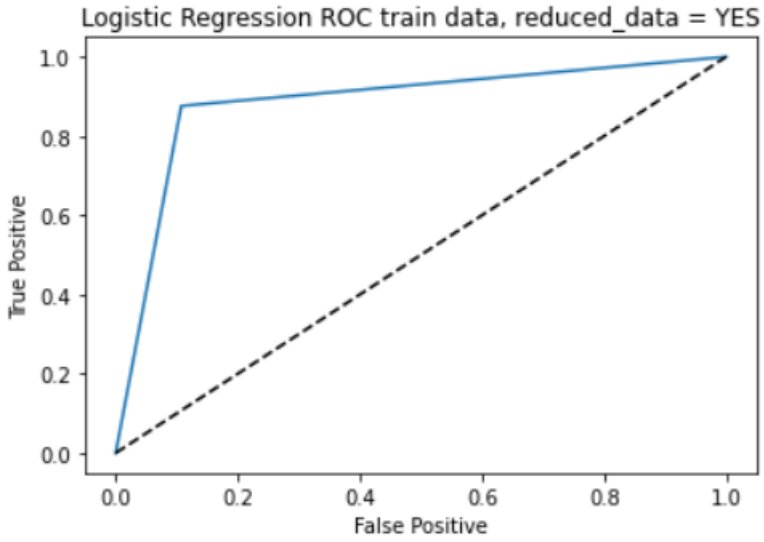


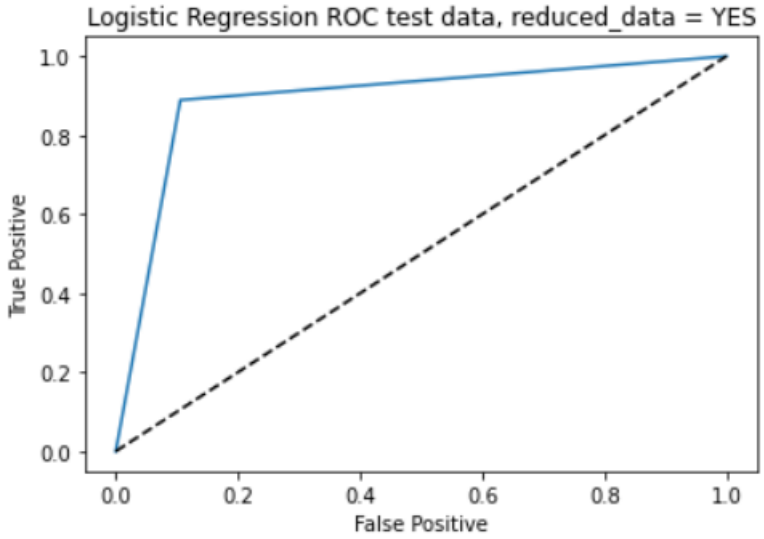
KNN:

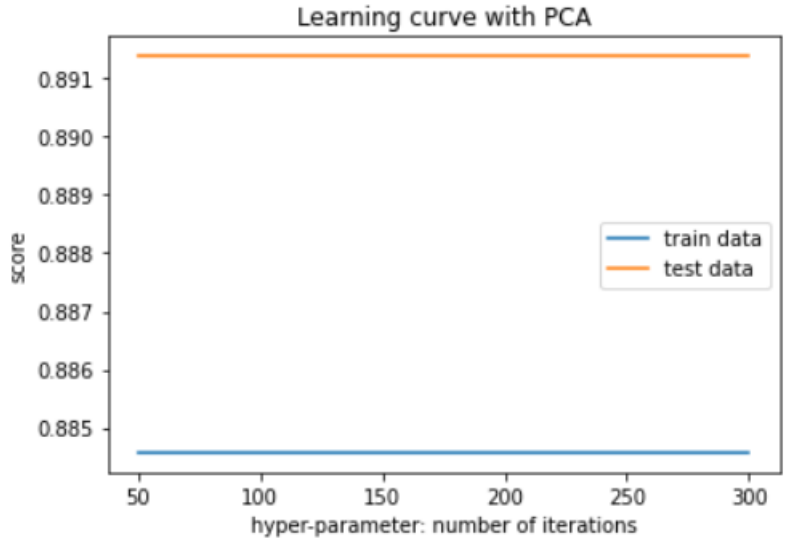


As we can see all the accuracies has been reduced after implementing the dimension reduction(PCA) on our feature space.

Logistic regression:

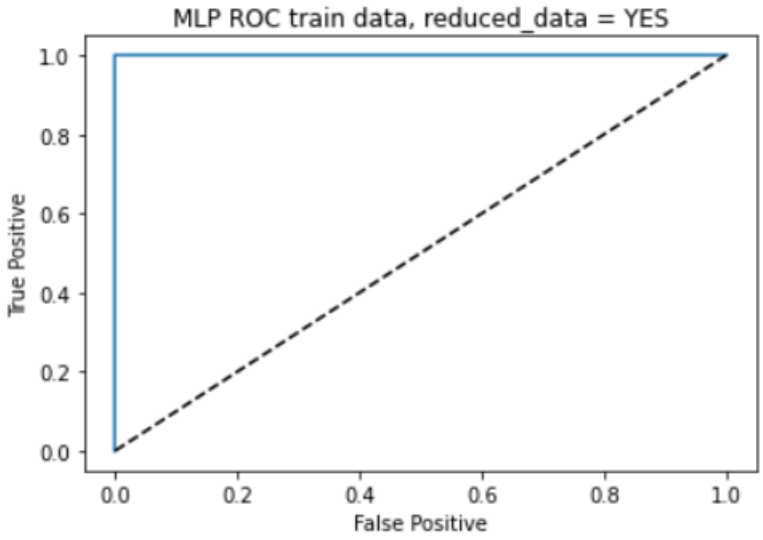

Logistic Regression ROC train data, reduced_data = YES
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

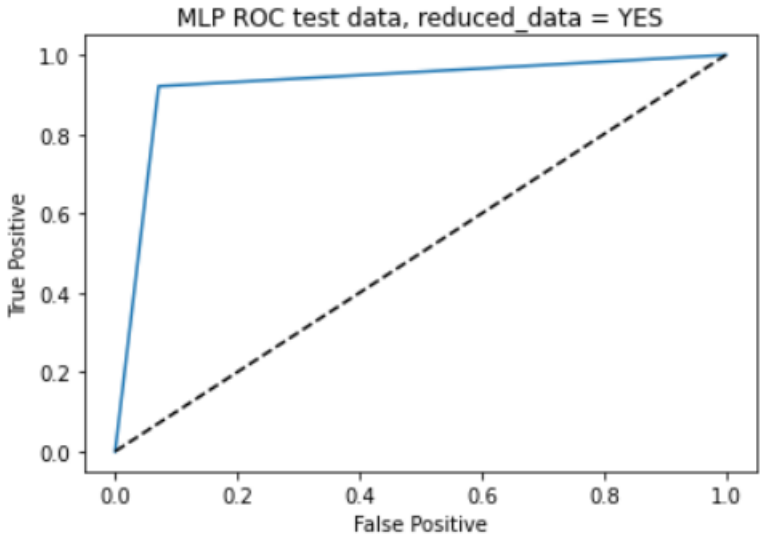

Logistic Regression ROC test data, reduced_data = YES
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

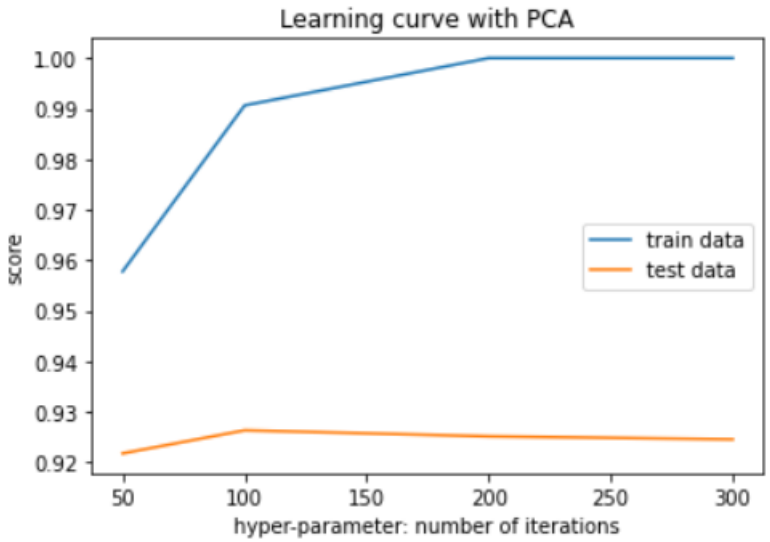

Learning curve with PCA
score
hyper-parameter: number of iterations
train data
test data
0.885
0.886
0.887
0.888
0.889
0.890
0.891
50
100
150
200
250
300

As we can see the accuracy is and also the learning rate is high both for train and test data, also the ROC is close to optimum classifier which is no false positive and all true positive.

MLP:



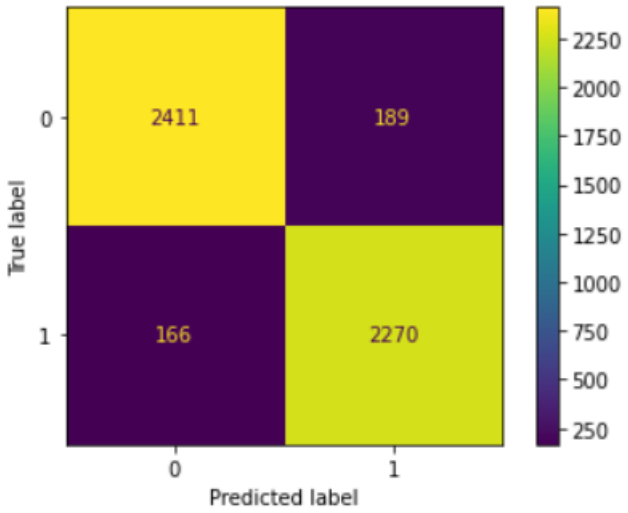




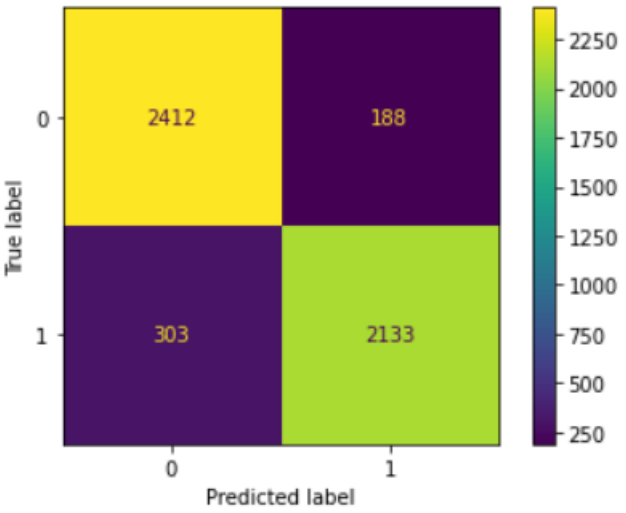
As we can see in MLP algorithm if we increase the hyper parameters, the learning rate would converge to 1. Also as we can see in the ROC, we can see that the accuracy n train data is 100% and on test data it is close to that.

The confusion matrixes:

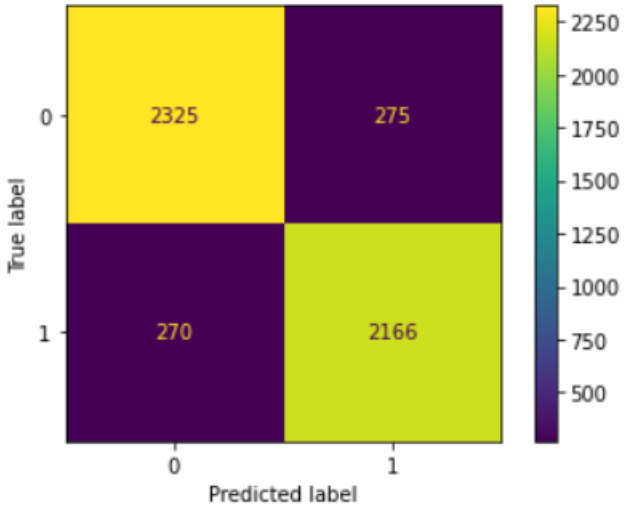
SVM:



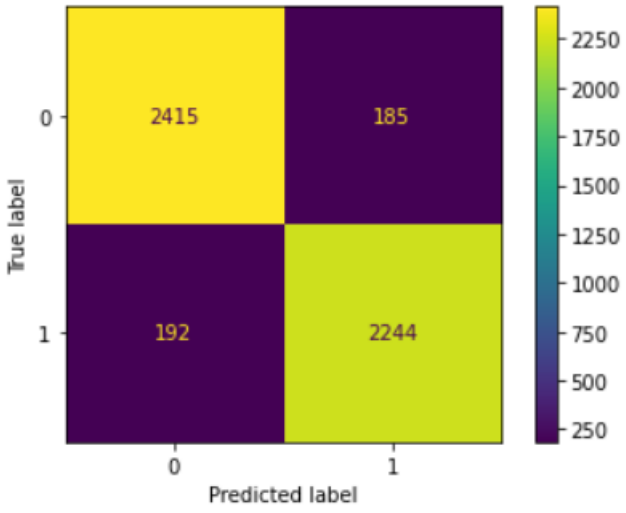
KNN:



Logistic regression:



MLP:



As you can see all the confusion matrixes are some how the same. The reason is that all these models would work well on the model and all of them have a very high accuracy.

Conclusion

In supervised learning we used 4 models, with 2 different method of feature extraction, on 2 kinds of prediction which were gender prediction and emotion prediction, and we used dimension reduction methods and without using them. Comparing all results, including : Accuracies, confusion matrixes and ROC plots, we find out that gender predictions had a higher accuracy (near to 90%) than emotion prediction and the reason is that the correlation between the data based on their emotions were to high that the models were not be able to classify them. About using PCA method in most of the cases it caused a lower accuracy which was as it was expected to be. Because using PCA algorithm which is dimension reduction method would cause some information in features to be removed from feature space and the models had to work and be trained with Lower amount of information. In feature extraction method, the second method which was a tempo-based algorithm had more accurate result. Because this methos is based on tempo recognition which became so useful for this project because the emotions are some how related by their tempo and less in frequency. So as we see we have achived an accuracy up to 70% in average for each class. At last to choose the best model, we can not choose one model as the best model because based on hyper parameters, usage of PCA, and feature extraction methods is some cases The SVM model had the best accuracy, in some points KNN had the highest accuracy and in some models, MLP or Logistic regression had the highest accuracy. But if we want to choose the best model between all these models that we have trained, the KNN algorithm without using PCA and with using second feature extraction method has the highest accuracy both in gender recognition and emotion recognition.

Unsupervised Learning

unsupervised learning is a machine learning technique in which models are not supervised using a training dataset. Instead, model themselves find the hidden patterns and insights from the given fact, **Unsupervised machine learning** is the process of inferring underlying hidden patterns from historical data. Within such an approach, a machine learning model tries to find any similarities, differences, patterns, and structure in data by itself.

In this project, we used different clustering methods. Clustering analysis is an unsupervised learning method that separates the data points into several specific bunches or groups, such that the data points in the same groups have similar properties and data points in different groups have different properties in some sense. There are several supervised learning methods, in this project,t we implemented the models of DBSCAN and HDBSCAN, Kmeans, and GMM.

Here is a brief explanation of each of the methods named above:

1. **DBSCAN and HDBSCAN:**
   Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.
   The DBSCAN algorithm uses two parameters:
   minutes: The minimum number of points (a threshold) clustered together for a region to be considered dense.
   eps (ε): A distance measure that will be used to locate the points in the neighborhood of any point.

2. **Kmeans:**
   K-means is an algorithm for exclusive clustering, also known as partitioning or segmentation. It puts the data points into the predefined number of clusters known as *K*. Each data item then gets assigned to the nearest cluster center, called *centroids.*

3. ***GMM:***
   Gaussian Mixture Models (GMMs) is an algorithm used in probabilistic clustering. Since the mean or variance is unknown, the models assume that there is a certain number of Gaussian distributions, each representing a separate cluster. The algorithm is utilized to decide which cluster a particular data point belongs to.

Implementation of different unsupervised learning methods

(using scikit library functions in Google Colab)

By using the scikit library we implement each of the mentioned models above and measure the accuracy of each.

First, we import the needed libraries as below:

```
from google.colab import drive
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn import metrics
from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import PCA
```

Here are the methods for classifying the dataset based on gender:

1. **DBSCAN:**
   First, we find the best epsilon to put in the DBSCAN algorithm:

```
[ ]  #finding best epsilon using knn algorithm

     nearest_neighbors = NearestNeighbors(n_neighbors=11)
     neighbors = nearest_neighbors.fit(features)

     distances, indices = neighbors.kneighbors(features)
     distances = np.sort(distances[:,10], axis=0)

     fig = plt.figure(figsize=(5, 5))
     plt.plot(distances)
     plt.xlabel("Points")
     plt.ylabel("Distance")
```

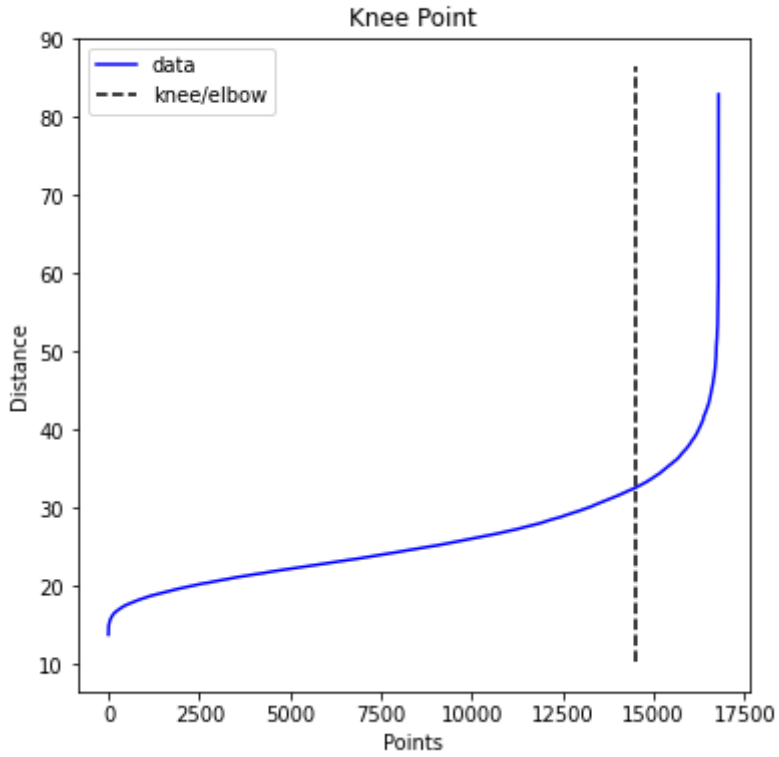
```
[ ]  !pip install kneed
     from kneed import KneeLocator
     i = np.arange(len(distances))
     knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp_method='polynomial')

     fig = plt.figure(figsize=(5, 5))
     knee.plot_knee()
     plt.xlabel("Points")
     plt.ylabel("Distance")

     print("best epsilon =",distances[knee.knee])
```

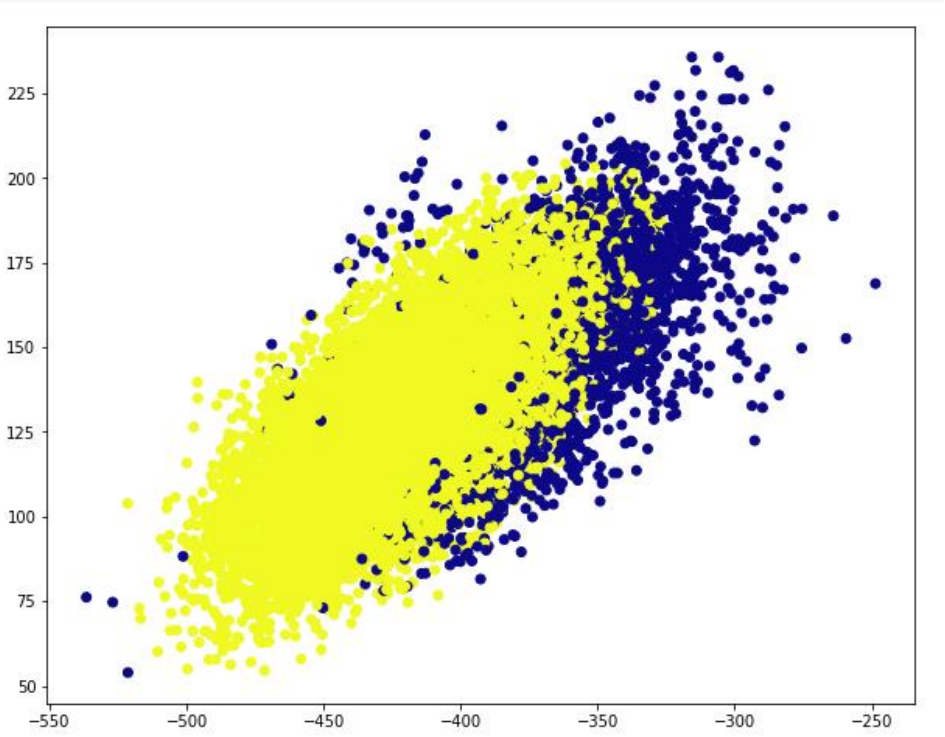
We see the results below, so we put epsilon equal to 33 in the following code.

```
best epsilon = 32.54772605754776
<Figure size 360x360 with 0 Axes>
```




```
[ ]  dbscan = DBSCAN(eps = 33, min_samples = 124).fit(features)
     dbscan_labels = dbscan.labels_
     plt.subplots(figsize=(10,8))
     plt.scatter(features[:, 0], features[:,1], c = dbscan_labels, cmap= "plasma")
     plt.show()
```

We see the results below:



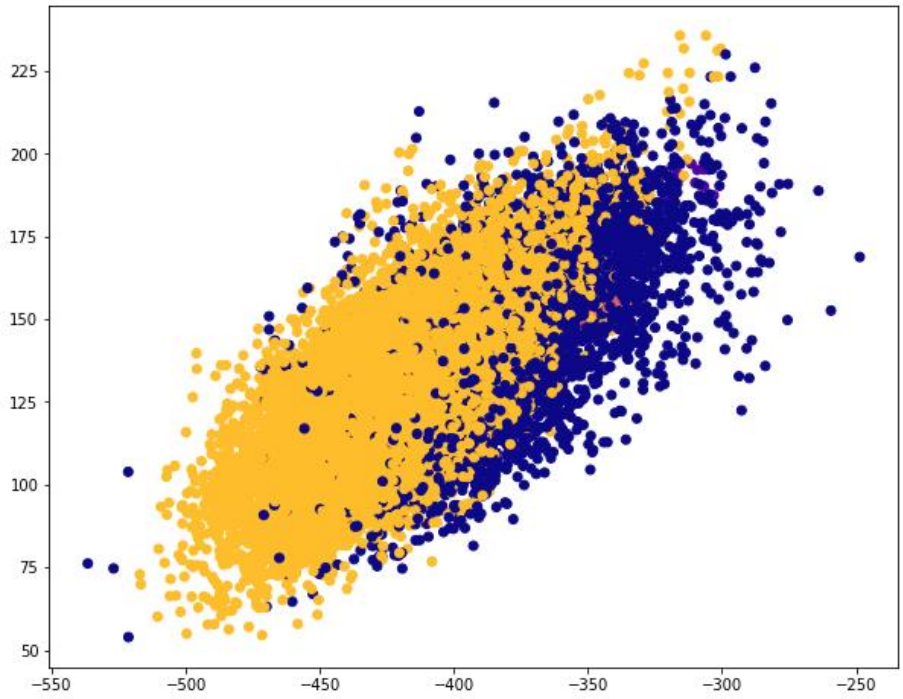
So, the DBSCAN algorithm puts the data into 2 clusters.

We then implement a model with HDBSCAN algorithm. The difference between DBSCAN and HDBSCAN algorithms is that HDBSCAN is basically a DBSCAN implementation for varying epsilon values and therefore only needs the minimum cluster size as single input parameter. Unlike DBSCAN, this allows to it find clusters of variable densities without having to choose a suitable distance threshold first. However, there are cases where we could still benefit from the use of an epsilon threshold.

2. **HDBSCAN:**
   Unlike DBSCAN, we can set the cluster size to the values we want.

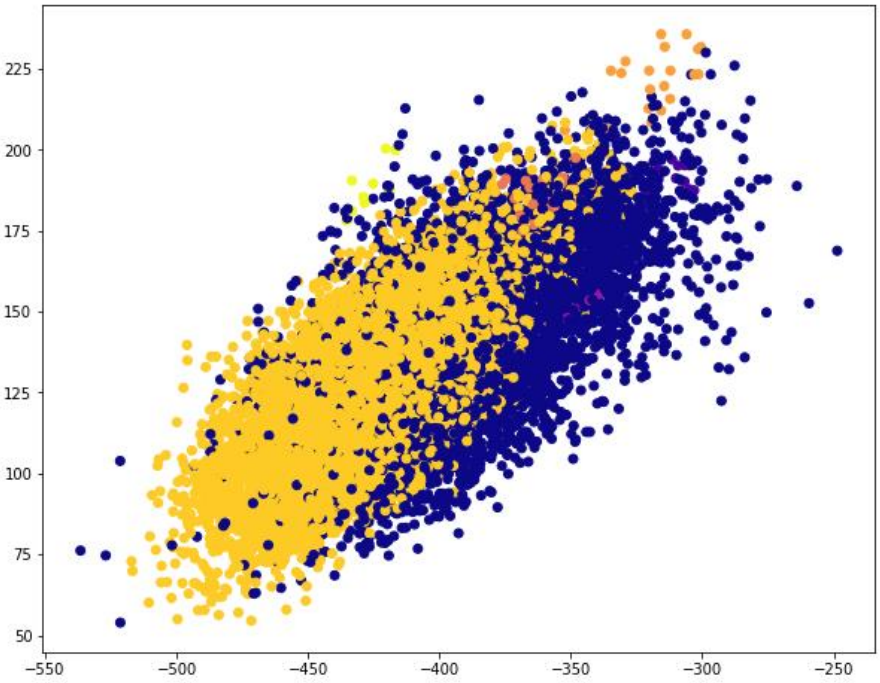
Here is the code for cluster size equal to 2:

```
!pip install hdbscan
import hdbscan
Hdbscan2 = hdbscan.HDBSCAN(min_cluster_size = 2, min_samples = 10).fit(features)
hdbscan2_labels = Hdbscan2.labels_
plt.subplots(figsize=(10,8))
plt.scatter(features[:, 0], features[:,1], c = hdbscan2_labels, cmap= "plasma")
plt.show()
```



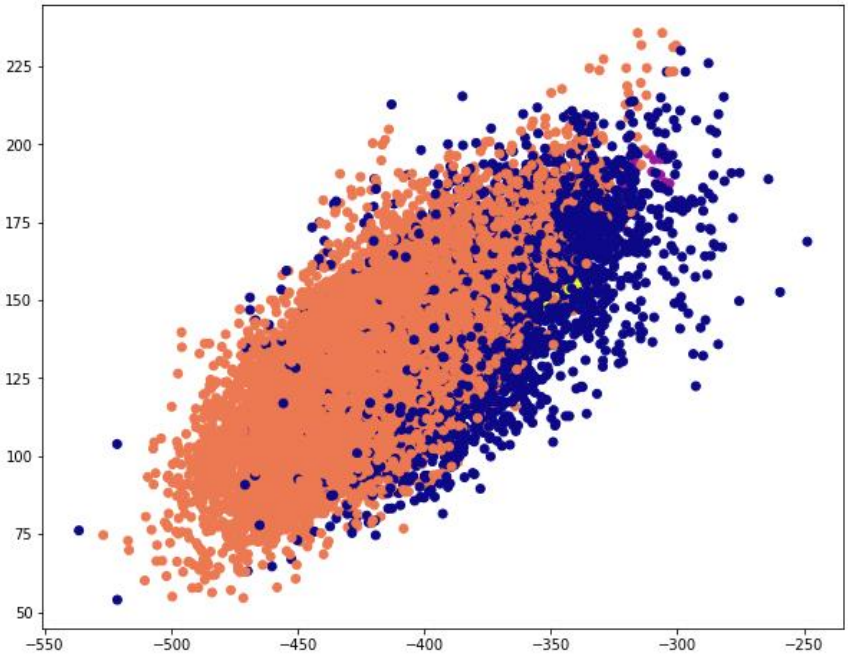
The code for cluster size equal to 4:

```
Hdbscan4 = hdbscan.HDBSCAN(min_cluster_size = 4, min_samples = 8).fit(features)
hdbscan4_labels = Hdbscan4.labels_
plt.subplots(figsize=(10,8))
plt.scatter(features[:, 0], features[:,1], c = hdbscan4_labels, cmap= "plasma")
plt.show()
```



The code for cluster size equal to 10:

```
[ ] Hdbscan10 = hdbscan.HDBSCAN(min_cluster_size = 10, min_samples = 8).fit(features)
    hdbscan10_labels = Hdbscan10.labels_
    plt.subplots(figsize=(10,8))
    plt.scatter(features[:, 0], features[:,1], c = hdbscan10_labels, cmap= "plasma")
    plt.show()
```

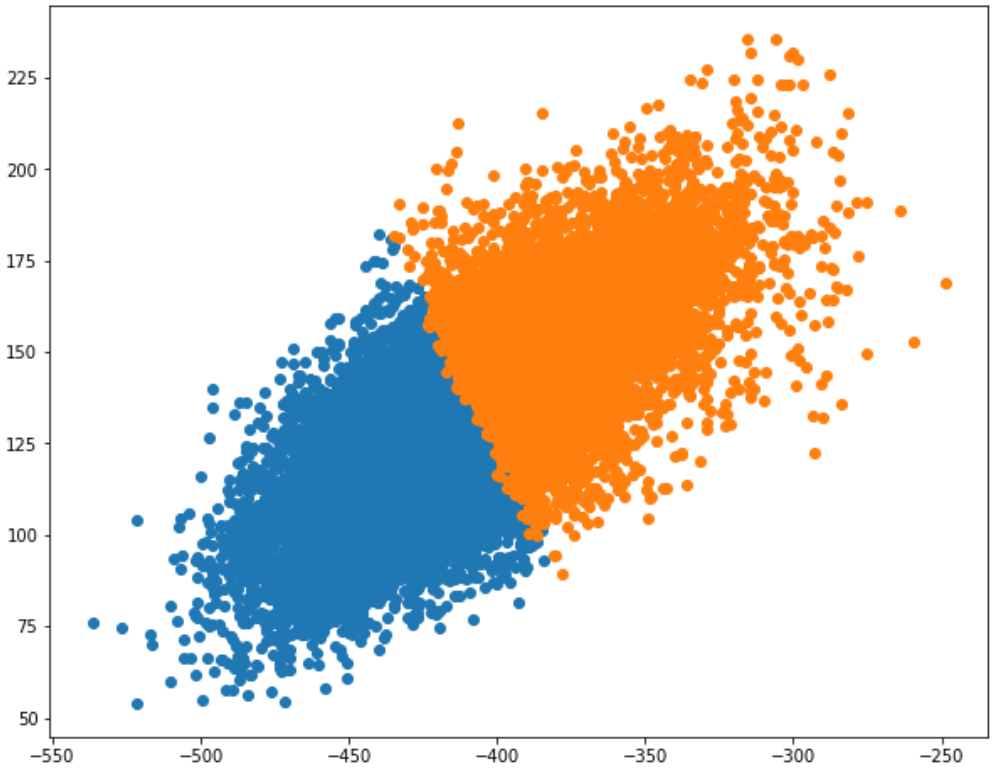


As we see from the above plots, the clustering for 2 clusters has the best results. We will evaluate these models with different evaluation metrics later on this report.

3. Kmeans Algorithm:

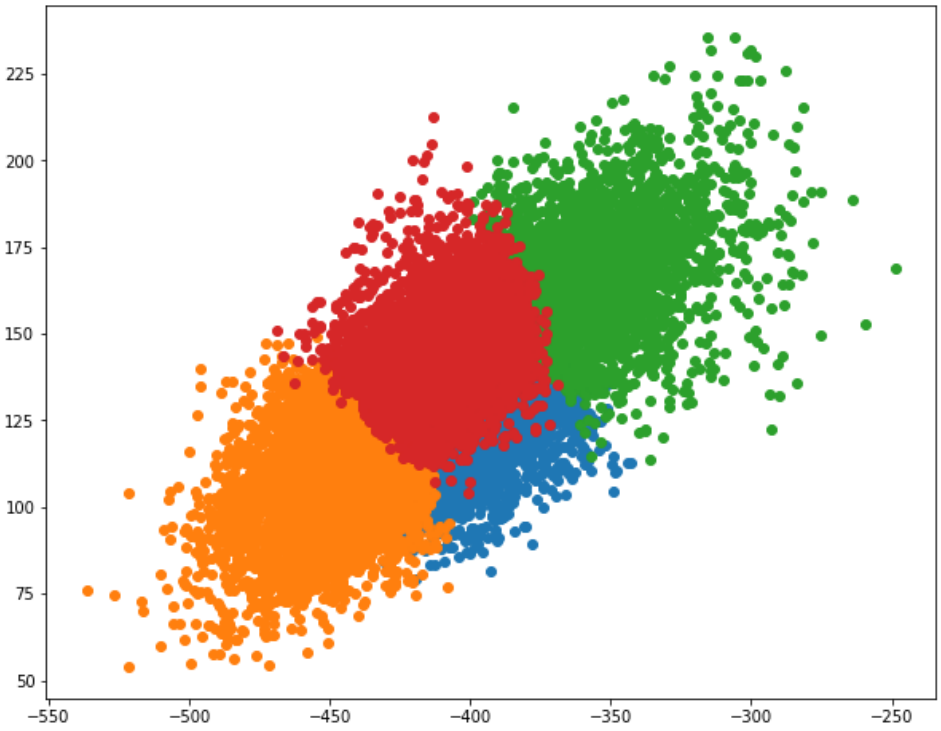
The code for cluster size equal to 2:

```
[ ]  #for num of clusters = 2
     Kmeans2_model = KMeans(n_clusters= 2, random_state= 42)
     Kmeans2_pred = Kmeans2_model.fit_predict(features)
     clusters_kmeans2 = unique(Kmeans2_pred)
     plt.subplots(figsize=(10,8))
     for cluster in clusters_kmeans2:
         row_ix = where(Kmeans2_pred == cluster)
         plt.scatter(features[row_ix, 0], features[row_ix, 1])
     plt.show()
```



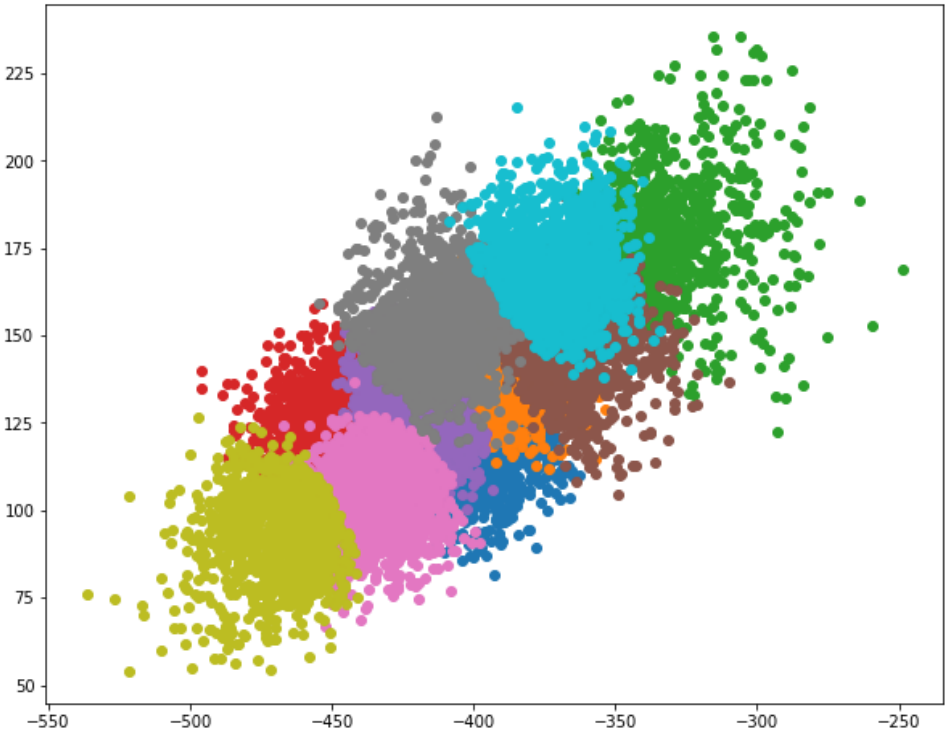
The code for cluster size equal to 4:

```
[ ]  #for num of clusters = 4
     Kmeans4_model = KMeans(n_clusters= 4, random_state= 42)
     Kmeans4_pred = Kmeans4_model.fit_predict(features)
     clusters_kmeans4 = unique(Kmeans4_pred)
     plt.subplots(figsize=(10,8))
     for cluster in clusters_kmeans4:
         row_ix = where(Kmeans4_pred == cluster)
         plt.scatter(features[row_ix, 0], features[row_ix, 1])
     plt.show()
```



The code for cluster size equal to 10:

```
[ ]  #for num of clusters = 10
     Kmeans10_model = KMeans(n_clusters= 10, random_state= 42)
     Kmeans10_pred = Kmeans10_model.fit_predict(features)
     clusters_kmeans10 = unique(Kmeans10_pred)
     plt.subplots(figsize=(10,8))
     for cluster in clusters_kmeans10:
         row_ix = where(Kmeans10_pred == cluster)
         plt.scatter(features[row_ix, 0], features[row_ix, 1])
     plt.show()
```

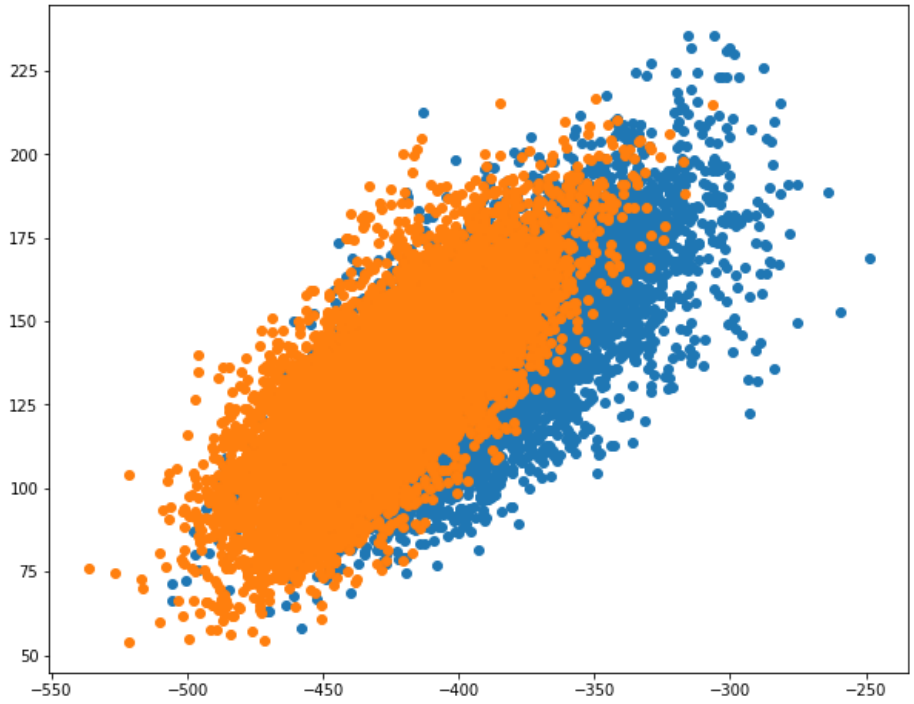


As we see from the above plots, the clustering for 2 and 4 clusters have better results than the 10 clusters. We will evaluate these models with different evaluation metrics later on this report.

4. GMM algorithm:

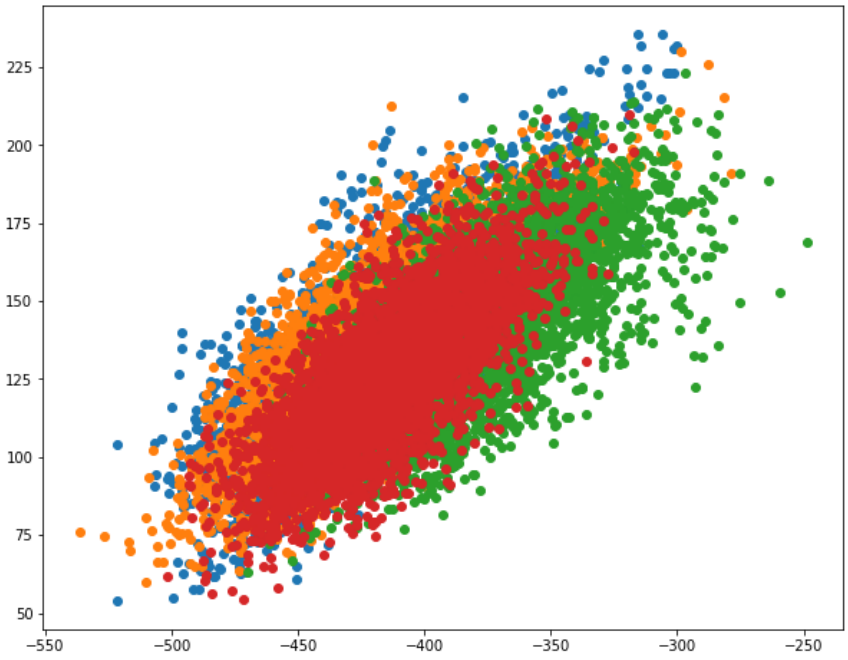
The code for cluster size equal to 2:

```
[ ]  #for num of clusters = 2
     GMM2_model = GaussianMixture(n_components=2)
     GMM2_pred = GMM2_model.fit_predict(features)
     clusters_GMM2 = unique(GMM2_pred)
     plt.subplots(figsize=(10,8))
     for cluster in clusters_GMM2:
         row_ix = where(GMM2_pred == cluster)
         plt.scatter(features[row_ix, 0], features[row_ix, 1])
     plt.show()
```



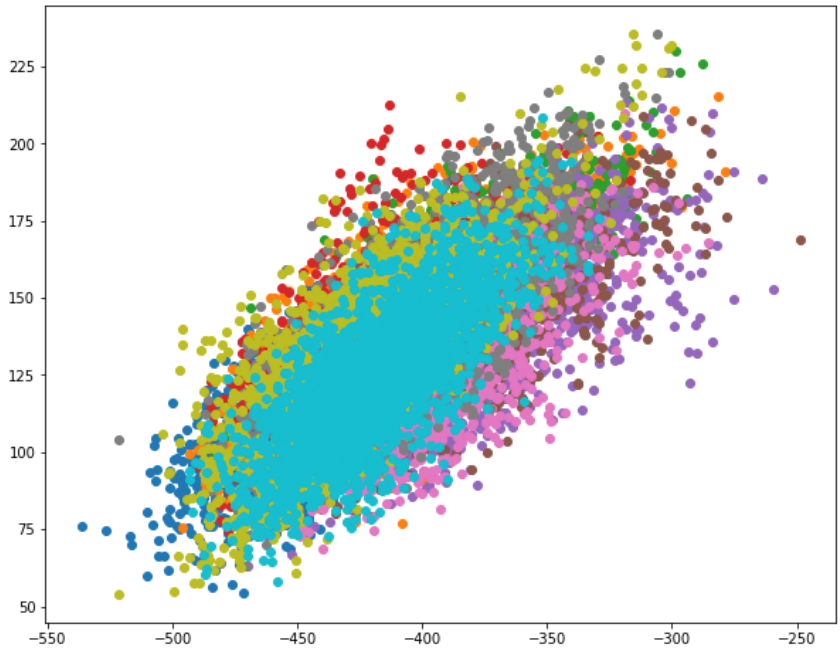
The code for cluster size equal to 4:

```
[ ]  #for num of clusters = 4
     GMM4_model = GaussianMixture(n_components=4)
     GMM4_pred = GMM4_model.fit_predict(features)
     clusters_GMM4 = unique(GMM4_pred)
     plt.subplots(figsize=(10,8))
     for cluster in clusters_GMM4:
         row_ix = where(GMM4_pred == cluster)
         plt.scatter(features[row_ix, 0], features[row_ix, 1])
     plt.show()
```



The code for cluster size equal to 10:

```
[ ]  #for num of clusters = 10
     GMM10_model = GaussianMixture(n_components=10)
     GMM10_pred = GMM10_model.fit_predict(features)
     clusters_GMM10 = unique(GMM10_pred)
     plt.subplots(figsize=(10,8))
     for cluster in clusters_GMM10:
         row_ix = where(GMM10_pred == cluster)
         plt.scatter(features[row_ix, 0], features[row_ix, 1])
     plt.show()
```



Now we need to evaluate all the models above.

When implementing a clustering algorithm for a dataset with no such target to aim for, an accuracy score is not possible. We therefore need to look for other types of measurement that give us an indication of performance. The most common is the distinctness of the clusters created.

Three important factors by which clustering can be evaluated are: Clustering tendency, optimal number of clusters, clustering quality. Clustering quality has both extrinsic and intrinsic measures.

Extrinsic measures are the measures which require ground truth labels. Examples are Adjusted Rand index, Mutual-information based scores, Homogeneity, Completeness and V-measure.

Intrinsic measures are the measures that don't require ground truth labels. Some of these measures are Silhouette Coefficient, Calinski-Harabasz index, and Davies-Bouldin index.

We start from calculating the intrinsic measures.

Calculating the **Silhouette Coefficient** for all the models above:

This score is between -1 and 1, where the higher the score, the more well defined and distinct the clusters are.

```
#silhouette score of the models:
print("silhouette score of dbscan:", metrics.silhouette_score(features,dbscan_labels))
print("silhouette score of hdbscan 2:", metrics.silhouette_score(features,hdbscan2_labels))
print("silhouette score of hdbscan 4:", metrics.silhouette_score(features,hdbscan4_labels))
print("silhouette score of hdbscan 10:", metrics.silhouette_score(features,hdbscan10_labels))
print("silhouette score of kmeans 2:", metrics.silhouette_score(features,Kmeans2_pred))
print("silhouette score of kmeans 4:", metrics.silhouette_score(features,Kmeans4_pred))
print("silhouette score of kmeans 10:", metrics.silhouette_score(features,Kmeans10_pred))
print("silhouette score of GMM 2:", metrics.silhouette_score(features,GMM2_pred))
print("silhouette score of GMM 4:", metrics.silhouette_score(features,GMM4_pred))
print("silhouette score of GMM 10:", metrics.silhouette_score(features,GMM10_pred))
```

```
silhouette score of dbscan: 0.28480473473081136
silhouette score of hdbscan 2: -0.036434168163789166
silhouette score of hdbscan 4: -0.034726678505026
silhouette score of hdbscan 10: 0.16306882711458648
silhouette score of kmeans 2: 0.2847176600287628
silhouette score of kmeans 4: 0.1896206009691554
silhouette score of kmeans 10: 0.10948630543611307
silhouette score of GMM 2: 0.09910263590870982
silhouette score of GMM 4: -0.006219543998652198
silhouette score of GMM 10: -0.07276504025055822
```

As we see, the highest Silhouette scores are for DBSCAN, Kmeans of 2 clusters and Kmeans of 4 clusters.

Calculating the **Calisnki-Harabasz** score for all the models above:

Like the Silhouette Coefficient, the higher the score, the more well-defined the clusters are. This score has no bound, meaning that there is no acceptable or good value, and must be tracked throughout the development of the model.

```
# calinski harabasz score of the models:
print("calinski harabasz score of dbscan:", metrics.calinski_harabasz_score(features,dbscan_labels))
print("calinski harabasz score of hdbscan 2:", metrics.calinski_harabasz_score(features,hdbscan2_labels))
print("calinski harabasz score of hdbscan 4:", metrics.calinski_harabasz_score(features,hdbscan4_labels))
print("calinski harabasz score of hdbscan 10:", metrics.calinski_harabasz_score(features,hdbscan10_labels))
print("calinski harabasz score of kmeans 2:", metrics.calinski_harabasz_score(features,Kmeans2_pred))
print("calinski harabasz score of kmeans 4:", metrics.calinski_harabasz_score(features,Kmeans4_pred))
print("calinski harabasz score of kmeans 10:", metrics.calinski_harabasz_score(features,Kmeans10_pred))
print("calinski harabasz score of GMM 2:", metrics.calinski_harabasz_score(features,GMM2_pred))
print("calinski harabasz score of GMM 4:", metrics.calinski_harabasz_score(features,GMM4_pred))
print("calinski harabasz score of GMM 10:", metrics.calinski_harabasz_score(features,GMM10_pred))
```

```
calinski harabasz score of dbscan: 2998.0390602501043
calinski harabasz score of hdbscan 2: 424.94116870181324
calinski harabasz score of hdbscan 4: 388.4098079172661
calinski harabasz score of hdbscan 10: 961.0191902267044
calinski harabasz score of kmeans 2: 9201.838922850844
calinski harabasz score of kmeans 4: 5886.817015205034
calinski harabasz score of kmeans 10: 3157.621357627711
calinski harabasz score of GMM 2: 2356.642133715847
calinski harabasz score of GMM 4: 899.3276383209715
calinski harabasz score of GMM 10: 477.028641364578
```

As we see, the highest Calisnki-Harabasz scores are for Kmeans of 2 clusters, Kmeans of 4 clusters, and Kmeans of 10 clusters.

Calculating the **Davies-Bouldin** score for all the models above:

This score measures the similarity of the clusters, meaning that the lower the score the better the separation there is between the clusters.

```
[ ]  # davies bouldin score of the models:
     print("davies bouldin score of dbscan:", metrics.davies_bouldin_score(features,dbscan_labels))
     print("davies bouldin score of hdbscan 2:", metrics.davies_bouldin_score(features,hdbscan2_labels))
     print("davies bouldin score of hdbscan 4:", metrics.davies_bouldin_score(features,hdbscan4_labels))
     print("davies bouldin score of hdbscan 10:", metrics.davies_bouldin_score(features,hdbscan10_labels))
     print("davies bouldin score of kmeans 2:", metrics.davies_bouldin_score(features,Kmeans2_pred))
     print("davies bouldin score of kmeans 4:", metrics.davies_bouldin_score(features,Kmeans4_pred))
     print("davies bouldin score of kmeans 10:", metrics.davies_bouldin_score(features,Kmeans10_pred))
     print("davies bouldin score of GMM 2:", metrics.davies_bouldin_score(features,GMM2_pred))
     print("davies bouldin score of GMM 4:", metrics.davies_bouldin_score(features,GMM4_pred))
     print("davies bouldin score of GMM 10:", metrics.davies_bouldin_score(features,GMM10_pred))
```

```
davies bouldin score of dbscan: 1.6336997476543926
davies bouldin score of hdbscan 2: 1.5231242358494455
davies bouldin score of hdbscan 4: 1.3769497878386594
davies bouldin score of hdbscan 10: 1.6806300722700183
davies bouldin score of kmeans 2: 1.2852571156938792
davies bouldin score of kmeans 4: 1.5740225167219135
davies bouldin score of kmeans 10: 1.8348467424439607
davies bouldin score of GMM 2: 2.4990362917528923
davies bouldin score of GMM 4: 6.659902754116131
davies bouldin score of GMM 10: 5.650554898248412
```

As we see the lowest Davies-Bouldin scores are for Kmeans of 2 clusters, HDBSCAN, and Kmeans of 4 clusters.

Now we continue by calculating the extrinsic measures.

Calculating the **adjusted rand index** for all the models above:

```
[ ]  #rand index for models:
     print("rand index of dbscan:", metrics.adjusted_rand_score(genders,dbscan_labels))
     print("rand index of hdbscan 2:", metrics.adjusted_rand_score(genders,hdbscan2_labels))
     print("rand index of hdbscan 4:", metrics.adjusted_rand_score(genders,hdbscan4_labels))
     print("rand index of hdbscan 10:", metrics.adjusted_rand_score(genders,hdbscan10_labels))
     print("rand index of kmeans 2:", metrics.adjusted_rand_score(genders,Kmeans2_pred))
     print("rand index of kmeans 4:", metrics.adjusted_rand_score(emotions,Kmeans4_pred))
     print("rand index of kmeans 10:", metrics.adjusted_rand_score(text_ID,Kmeans10_pred))
     print("rand index of GMM 2:", metrics.adjusted_rand_score(genders,GMM2_pred))
     print("rand index of GMM 4:", metrics.adjusted_rand_score(emotions,GMM4_pred))
     print("rand index of GMM 10:", metrics.adjusted_rand_score(text_ID,GMM10_pred))
```

```
rand index of dbscan: 0.01460115950154463
rand index of hdbscan 2: 0.02951996431899755
rand index of hdbscan 2: 0.050799873452305896
rand index of hdbscan 2: 0.02541355941770397
rand index of kmeans 2: 0.009089499243896296
rand index of kmeans 4: 0.02129346124346133
rand index of kmeans 10: 0.005656728145904376
rand index of GMM 2: 0.056964875164928964
rand index of GMM 4: 0.0072834213263010044
rand index of GMM 10: 0.0036994556474662524
```

as we see the highest rand indices are for GMM of 2 clusters, HDBSCAN, and Kmeans of 4 clusters.

Calculating the **Mutual Information** for all the models above:

```
[ ]  #Mutual Information score for models:
     print("mutual information score of dbscan:", metrics.mutual_info_score(genders,dbscan_labels))
     print("mutual information score of hdbscan 2:", metrics.mutual_info_score(genders,hdbscan2_labels))
     print("mutual information score of hdbscan 4:", metrics.adjusted_rand_score(genders,hdbscan4_labels))
     print("mutual information score of hdbscan 10:", metrics.adjusted_rand_score(genders,hdbscan10_labels))
     print("mutual information score of kmeans 2:", metrics.mutual_info_score(genders,Kmeans2_pred))
     print("mutual information score of kmeans 4:", metrics.mutual_info_score(emotions,Kmeans4_pred))
     print("mutual information score of kmeans 10:", metrics.mutual_info_score(text_ID,Kmeans10_pred))
     print("mutual information score of GMM 2:", metrics.mutual_info_score(genders,GMM2_pred))
     print("mutual information score of GMM 4:", metrics.mutual_info_score(emotions,GMM4_pred))
     print("mutual information score of GMM 10:", metrics.mutual_info_score(text_ID,GMM10_pred))
```

```
mutual information score of dbscan: 0.010980997390407782
mutual information score of hdbscan 2: 0.018745841763565255
mutual information score of hdbscan 4: 0.050799873452305896
mutual information score of hdbscan 10: 0.02541355941770397
mutual information score of kmeans 2: 0.00435183992582544
mutual information score of kmeans 4: 0.03346738199051055
mutual information score of kmeans 10: 0.03403679708649348
mutual information score of GMM 2: 0.028407201359370582
mutual information score of GMM 4: 0.011490972970172472
mutual information score of GMM 10: 0.02008988025118535
```

As we see, the highest mutual information scores are for Kmeans of 4 clusters, GMM of 2 clusters, and HDBSCAN.

Calculating the **Homogenity Score** for all the models above:

```
[ ]  #Homogenity score for models:
     print("homogenity score of dbscan:", metrics.homogeneity_score(genders,dbscan_labels))
     print("homogenity score of hdbscan 2:", metrics.homogeneity_score(genders,hdbscan2_labels))
     print("homogenity score of hdbscan 4:", metrics.homogeneity_score(genders,hdbscan4_labels))
     print("homogenity score of hdbscan 10:", metrics.homogeneity_score(genders,hdbscan10_labels))
     print("homogenity score of kmeans 2:", metrics.homogeneity_score(genders,Kmeans2_pred))
     print("homogenity score of kmeans 4:", metrics.homogeneity_score(emotions,Kmeans4_pred))
     print("homogenity score of kmeans 10:", metrics.homogeneity_score(text_ID,Kmeans10_pred))
     print("homogenity score of GMM 2:", metrics.homogeneity_score(genders,GMM2_pred))
     print("homogenity score of GMM 4:", metrics.homogeneity_score(emotions,GMM4_pred))
     print("homogenity score of GMM 10:", metrics.homogeneity_score(text_ID,GMM10_pred))
```

```
homogenity score of dbscan: 0.0158561760263728
homogenity score of hdbscan 2: 0.027068339623253786
homogenity score of hdbscan 4: 0.04226229446864987
homogenity score of hdbscan 10: 0.023976978661123985
homogenity score of kmeans 2: 0.006283904589829145
homogenity score of kmeans 4: 0.02414170285445999
homogenity score of kmeans 10: 0.014748817101755225
homogenity score of GMM 2: 0.041019004845976904
homogenity score of GMM 4: 0.008289015705895178
homogenity score of GMM 10: 0.008705342299627715
```

as we see, the highest homogenity scores are for GMM of 2 clusters, hdbscan, and kmeans of 2 clusters.

As we see, the highest homogeneity scores are for GMM of 2 clusters, HDBSCAN and Kmeans of 2 clusters.

Calculating the **V-measure Score** for all the models above:

```
[ ] #v-measure score for models:
    print("v-measure score of dbscan:", metrics.homogeneity_score(genders,dbscan_labels))
    print("v-measure score of hdbscan 2:", metrics.homogeneity_score(genders,hdbscan2_labels))
    print("v-measure score of hdbscan 4:", metrics.homogeneity_score(genders,hdbscan4_labels))
    print("v-measure score of hdbscan 10:", metrics.homogeneity_score(genders,hdbscan10_labels))
    print("v-measure score of kmeans 2:", metrics.homogeneity_score(genders,Kmeans2_pred))
    print("v-measure score of kmeans 4:", metrics.homogeneity_score(emotions,Kmeans4_pred))
    print("v-measure score of kmeans 10:", metrics.homogeneity_score(text_ID,Kmeans10_pred))
    print("v-measure score of GMM 2:", metrics.homogeneity_score(genders,GMM2_pred))
    print("v-measure score of GMM 4:", metrics.homogeneity_score(emotions,GMM4_pred))
    print("v-measure score of GMM 10:", metrics.homogeneity_score(text_ID,GMM10_pred))
```

```
v-measure score of dbscan: 0.0158561760263728
v-measure score of hdbscan 2: 0.027068339623253786
v-measure score of hdbscan 4: 0.04226229446864987
v-measure score of hdbscan 10: 0.023976978661123985
v-measure score of kmeans 2: 0.006283904589829145
v-measure score of kmeans 4: 0.02414170285445999
v-measure score of kmeans 10: 0.014748817101755225
v-measure score of GMM 2: 0.041019004845976904
v-measure score of GMM 4: 0.008289015705895178
v-measure score of GMM 10: 0.008705342299627715
```

as we see, the highest v measure scores are for GMM of 2 clusters, hdbscan of 2 clusters, and kmeans of 4 clusters.

As we see, the highest v-measure scores are for GMM of 2 clusters, HDBSCAN of 2 clusters, and Kmeans of 4 clusters.

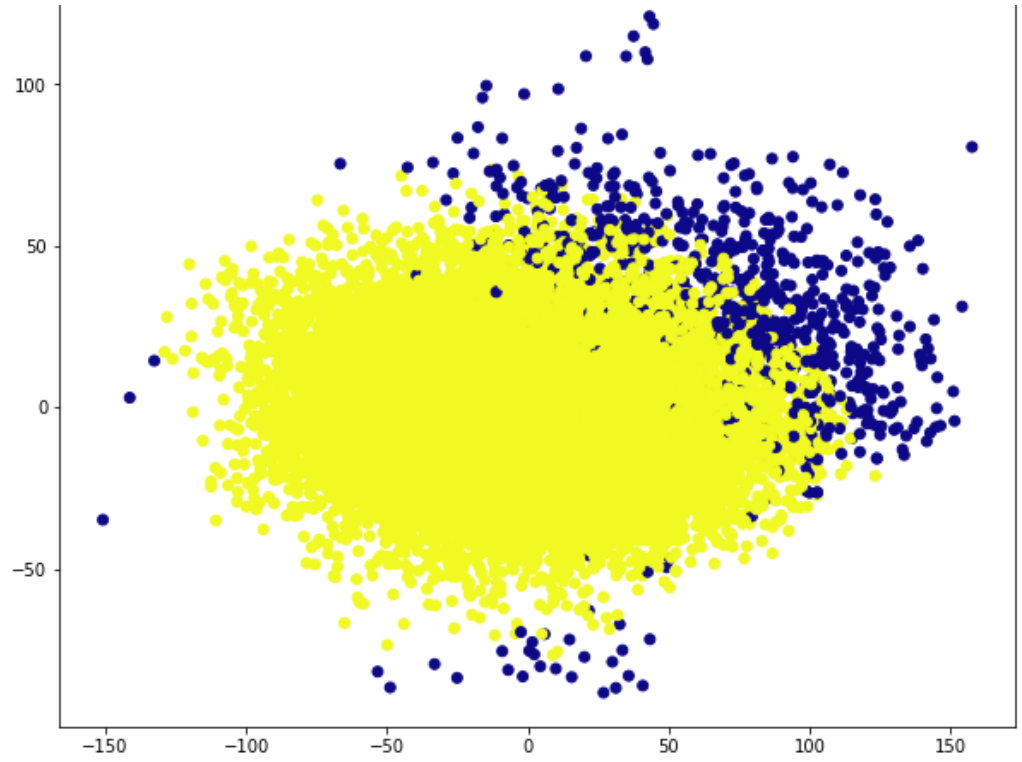
From what we see from the scatter plots of the different models and different types of scores that we evaluated, the best results are for Kmeans algorithms in general, specifically Kmeans of number of clusters equal to 2 and 4. other different clustering methods that put num of clusters equal to 2 have also good results, including DBSCAN and HDBSCAN, and GMM of 2 clusters.

We also take the 3 algorithms for number of clusters equal to 2, and perform dimension reduction on the data with PCA algorithm, and compare with the previous results:

```
[ ]  pca = PCA(n_components = 0.95)
     pca.fit(features)
     reduced_features = pca.transform(features)
```

DBSCAN

```
[ ]  dbscan_reduced = DBSCAN(eps = 33, min_samples = 124).fit(reduced_features)
     dbscan_labels_reduced = dbscan_reduced.labels_
     plt.subplots(figsize=(10,8))
     plt.scatter(reduced_features[:, 0], reduced_features[:,1], c = dbscan_labels_reduced, cmap= "plasma")
     plt.show()
     print("silhouette score of dbscan:", metrics.silhouette_score(reduced_features,dbscan_labels_reduced))
     print("calinski harabasz score of dbscan:", metrics.calinski_harabasz_score(reduced_features,dbscan_labels_reduced))
     print("davies bouldin score of dbscan:", metrics.davies_bouldin_score(reduced_features,dbscan_labels_reduced))
```



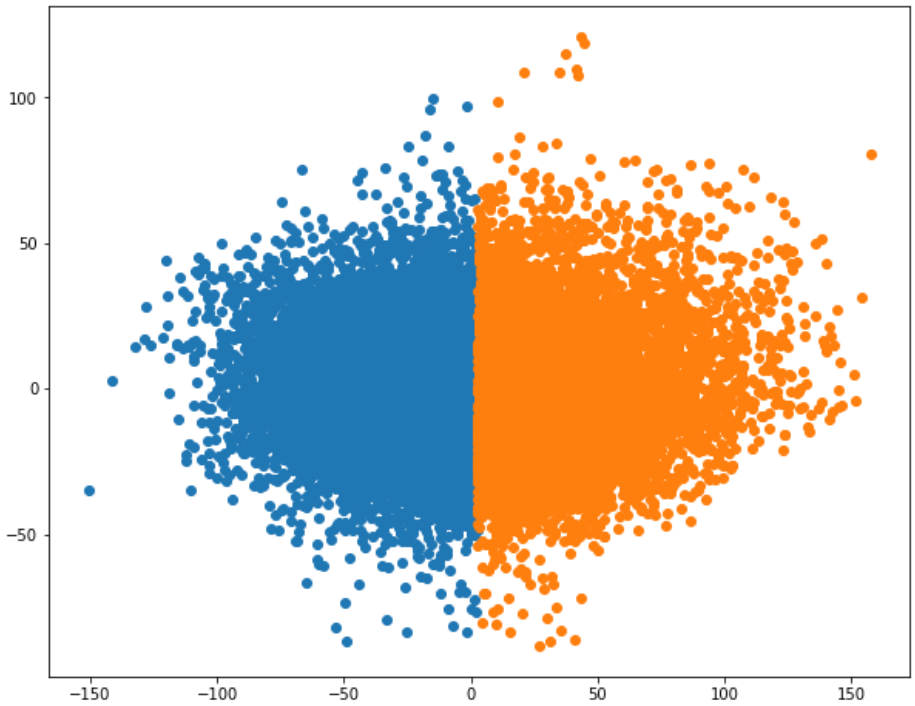

```
silhouette score of dbscan: 0.34714669659475655
calinski harabasz score of dbscan: 1511.5645001753876
davies bouldin score of dbscan: 1.4823861630682738
```

As we see, the Silhouette score is higher than before, but Calinski-Harabasz score and Davies-Bouldin scores are lower than before. Overall conclusion from the plot itself, is that without clustering the separation of classes was better.

Kmeans

2 cluster:

```
Kmeans2_model_reduced = KMeans(n_clusters= 2, random_state= 42)
Kmeans2_pred_reduced = Kmeans2_model_reduced.fit_predict(reduced_features)
clusters_kmeans2_reduced = unique(Kmeans2_pred_reduced)
plt.subplots(figsize=(10,8))
for cluster in clusters_kmeans2_reduced:
    row_ix = where(Kmeans2_pred_reduced == cluster)
    plt.scatter(reduced_features[row_ix, 0], reduced_features[row_ix, 1])
plt.show()
print("silhouette score of kmeans 2:", metrics.silhouette_score(reduced_features,Kmeans2_pred_reduced))
print("calinski harabasz score of kmeans 2:", metrics.calinski_harabasz_score(reduced_features,Kmeans2_pred_reduced))
print("davies bouldin score of kmeans 2:", metrics.davies_bouldin_score(reduced_features,Kmeans2_pred_reduced))
```




```
silhouette score of kmeans 2: 0.29936761998398076
calinski harabasz score of kmeans 2: 9972.451517431065
davies bouldin score of kmeans 2: 1.227230226316242
```

As we see, the Silhouette score and Calinski-Harabasz score are higher than before, and Davies Bouldin score is lower than before. Looking at the plots, both models with and without PCA are good clustering models, but the second one is a little better.

GMM

2 clusters:

```
GMM2_model_reduced = GaussianMixture(n_components=2)
GMM2_pred_reduced = GMM2_model.fit_predict(reduced_features)
clusters_GMM2_reduced = unique(GMM2_pred_reduced)
plt.subplots(figsize=(10,8))
for cluster in clusters_GMM2_reduced:
    row_ix = where(GMM2_pred_reduced == cluster)
    plt.scatter(reduced_features[row_ix, 0], reduced_features[row_ix, 1])
plt.show()
print("silhouette score of GMM 2:", metrics.silhouette_score(reduced_features,GMM2_pred_reduced))
print("calinski harabasz score of GMM 2:", metrics.calinski_harabasz_score(reduced_features,GMM2_pred_reduced))
print("davies bouldin score of GMM 2:", metrics.davies_bouldin_score(reduced_features,GMM2_pred_reduced))
```

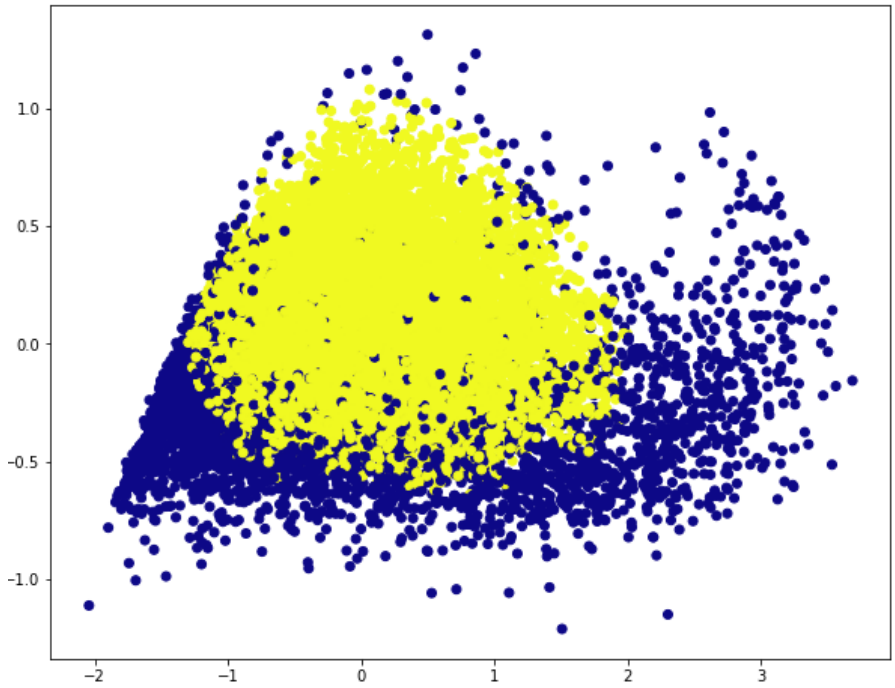

```
silhouette score of GMM 2: 0.1388377597378705
calinski harabasz score of GMM 2: 3020.6025068001786
davies bouldin score of GMM 2: 2.1777109026744514
```

as we see, the Silhouette score and Calinski Harabasz score are higher than before, and Davies Bouldin score is lower than before. Looking at the plots, both models with and without PCA are good clustering models but the second one is a little better.

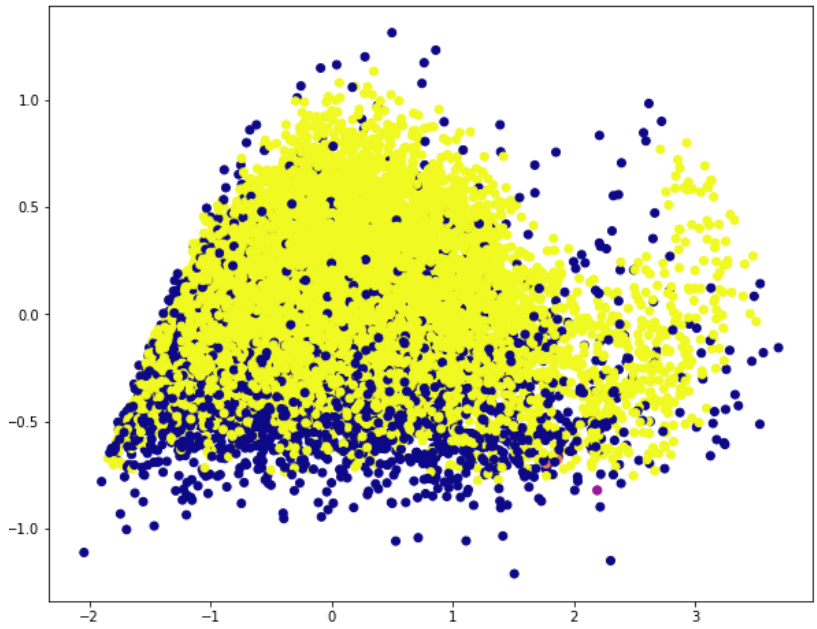
Now we repeat the steps above for the second set of features, which were developed as explained above. We have 384 features, by applying PCA we reduce the dimensions to 14.

The results are as below:

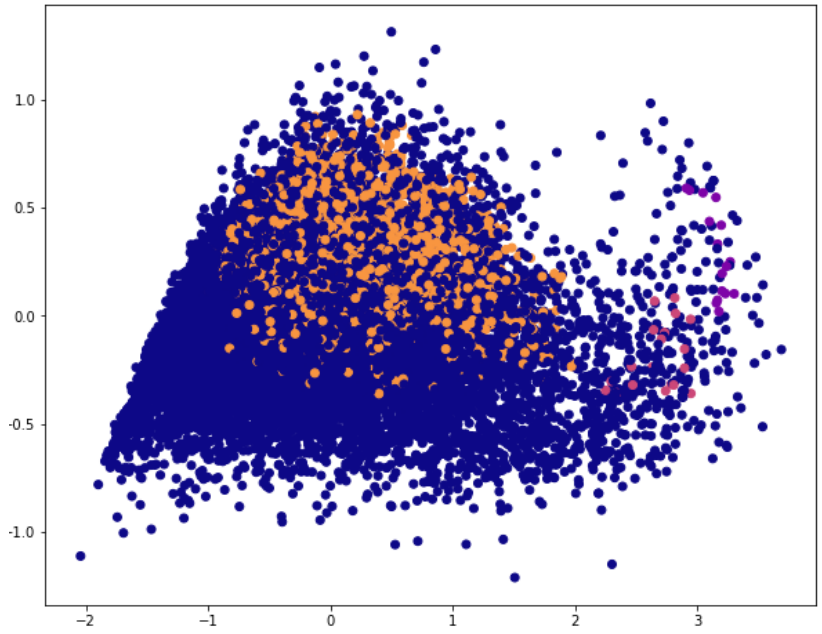
DBSCAN:



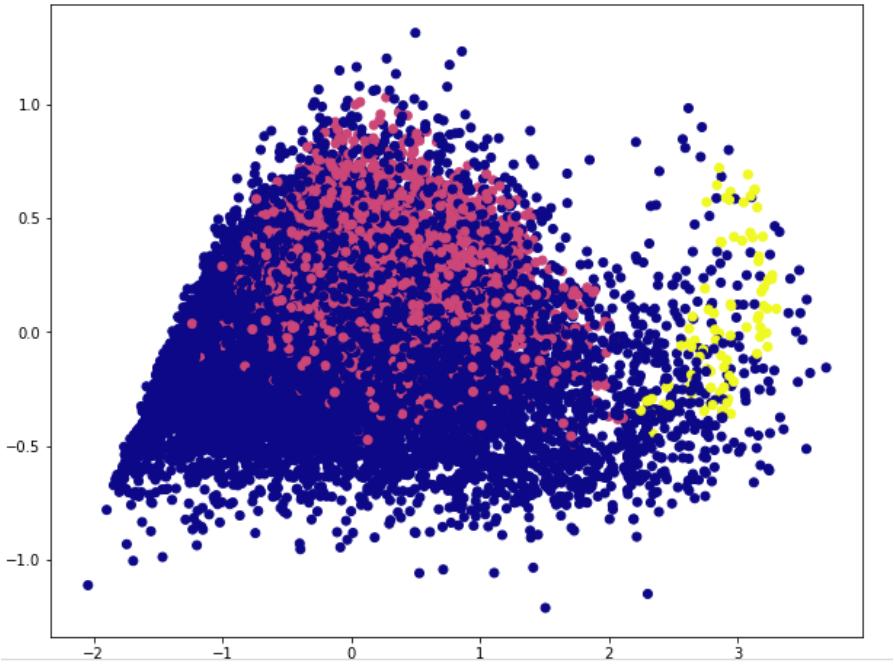
HDBSCAN FOR n=2:



HDBSCAN FOR n=4:

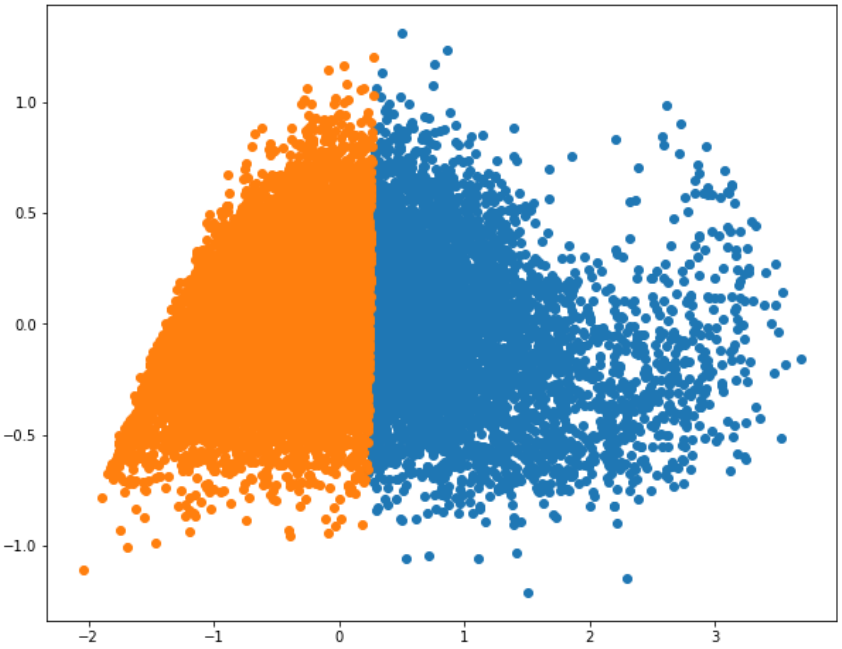


HDBSCAN for n=10:

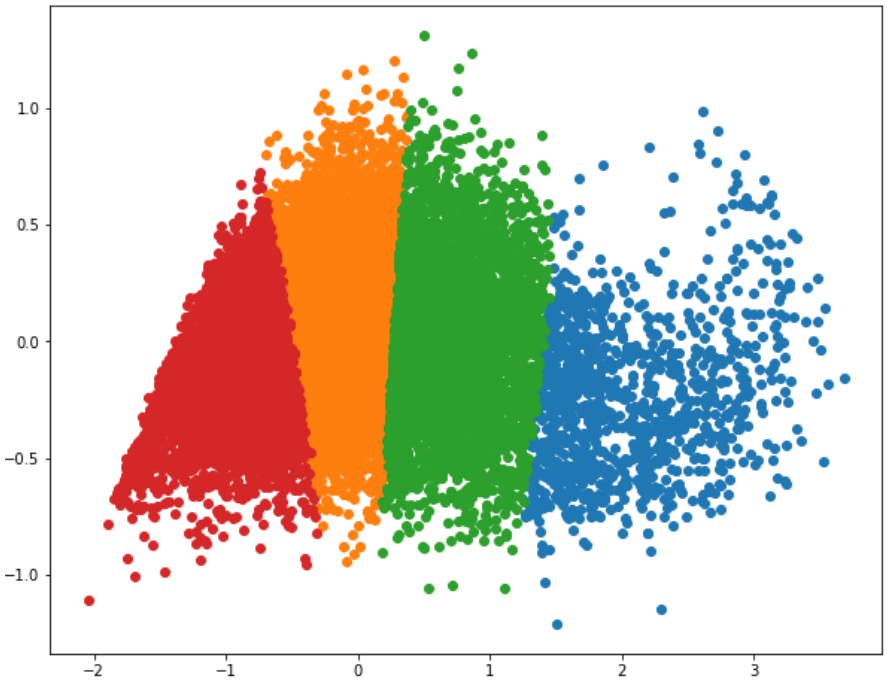


As we see from the plots, the HDBSCAN has the best results for 2 clusters, and by increasing the number of clusters to 4 and 10, the model can not classify the data well, and only produces about 3 main clusters as seen in the plots.

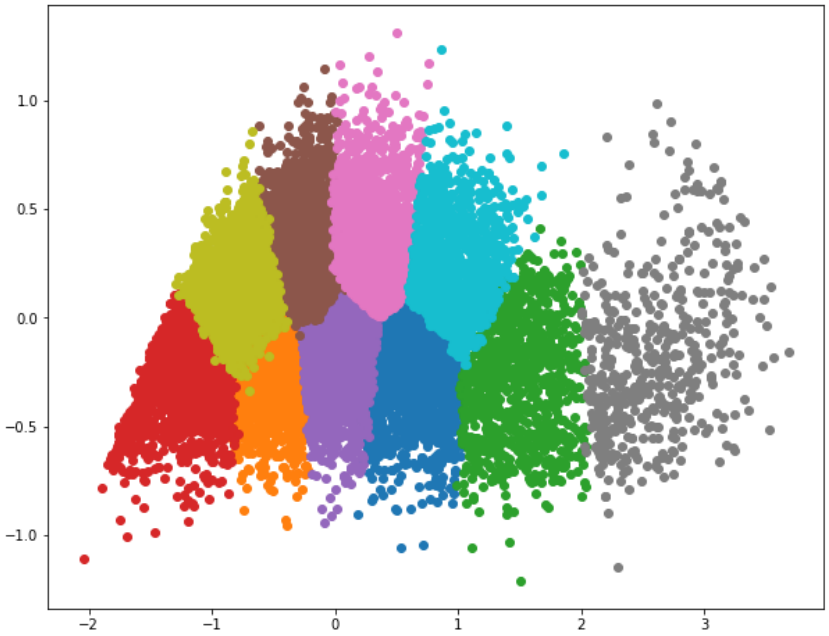
Kmeans for n=2:



Kmeans for n=4:

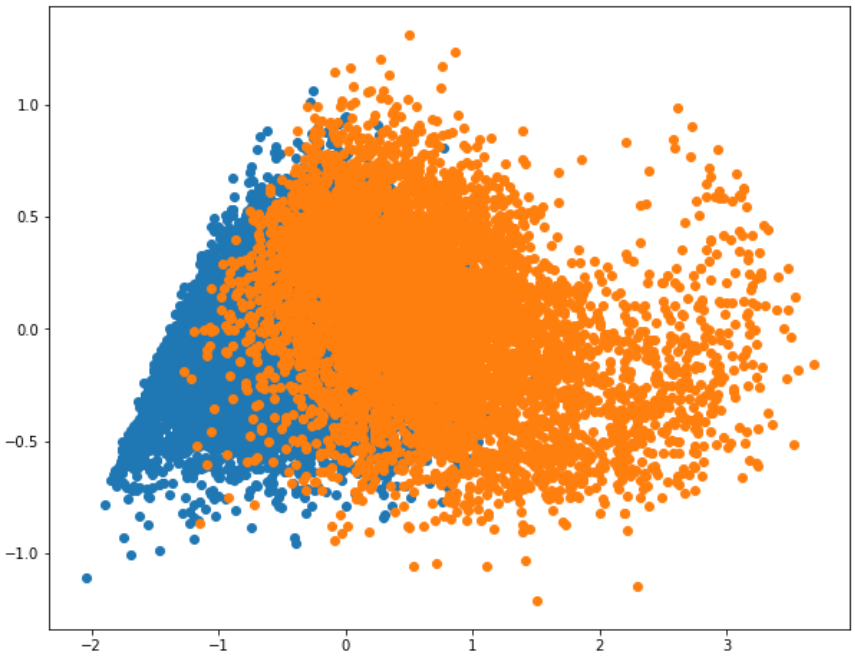


Kmeans for n=10:

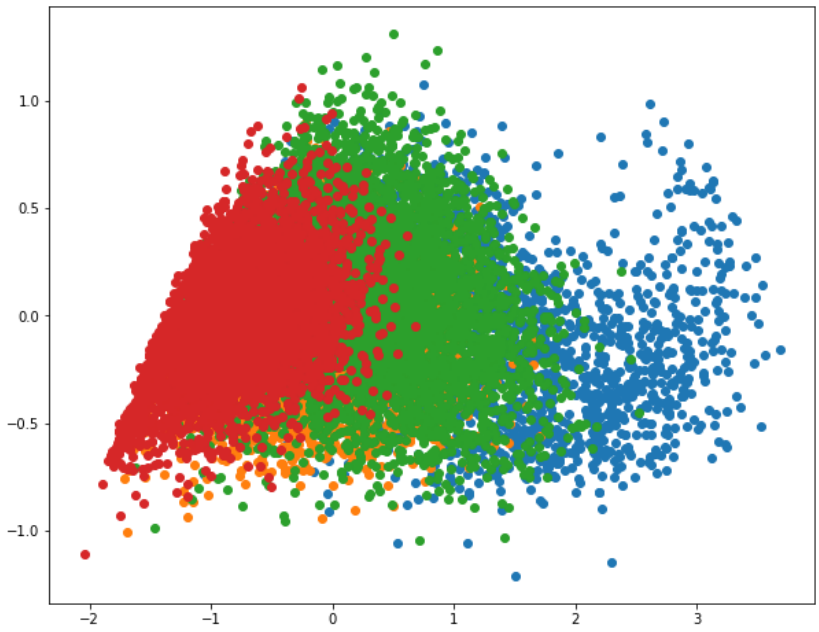


As we see from the plots the classifier works better than the one with previous features in Kmeans models. In all 3 models, the classifier has classified the data to the desired number of cluster well.

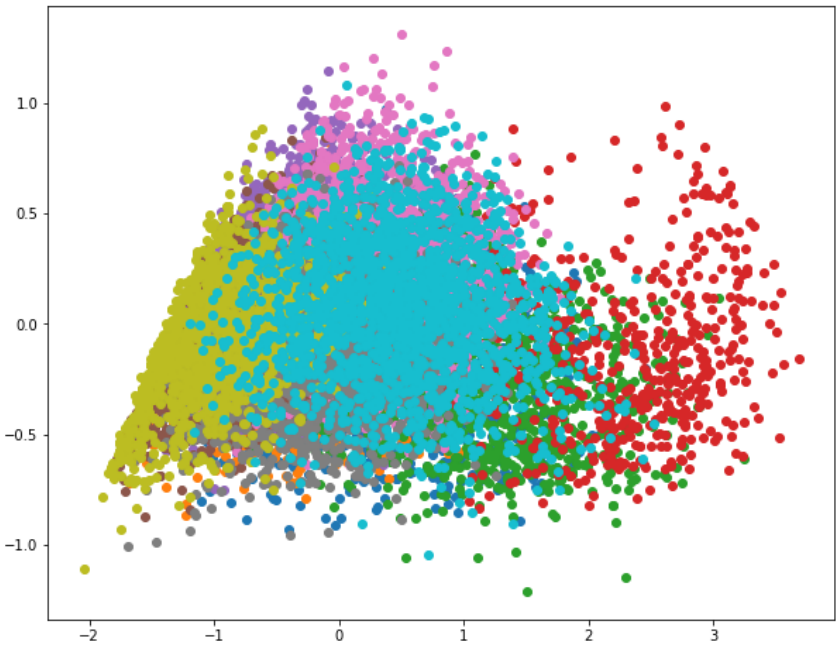
GMM for n=2:



GMM for n=4:



GMM for n=10:



As we see from the plots, the model acts almost good for n=2,4 but doesn't produce the desired number of clusters when n=10.

Now we use the same evaluation metrics explained above the compare the results of our different models:

The Intrinsic Measures:

Silouhette Scores:

```
silhouette score of dbscan: 0.24809462888333075
silhouette score of hdbscan 2: 0.06221021575625356
silhouette score of hdbscan 4: -0.24070582274955551
silhouette score of hdbscan 10: -0.041688940069919606
silhouette score of kmeans 2: 0.40125791069604405
silhouette score of kmeans 4: 0.2546726609159781
silhouette score of kmeans 10: 0.1666509234059923
silhouette score of GMM 2: 0.2530010439150373
silhouette score of GMM 4: 0.038913229377365355
silhouette score of GMM 10: -0.0584968786912401
```

The highest Silouhette Scores are for kmeans of 2, kmeans of 4, and GMM of 2, with highest score of 0.4 which is higher than the results from previous features.

Calinski-Harabasz Scores

```
calinski harabasz score of dbscan: 506.22646770134423
calinski harabasz score of hdbscan 2: 95.50745729940762
calinski harabasz score of hdbscan 4: 169.6274564202286
calinski harabasz score of hdbscan 10: 619.1605227317172
calinski harabasz score of kmeans 2: 14935.36689669014
calinski harabasz score of kmeans 4: 12039.959023890058
calinski harabasz score of kmeans 10: 7249.290878138621
calinski harabasz score of GMM 2: 7936.10808511462
calinski harabasz score of GMM 4: 3885.4652519578317
calinski harabasz score of GMM 10: 1852.0968794135556
```

The highest Calinski-Harabasz Scores are for kmeans of 2, kmeans of 4, and GMM of 2, with highest score of 14935 which is higher than the results from previous features.

David-Bouldin Scores:

```
davies bouldin score of dbscan: 5.026752775634968
davies bouldin score of hdbscan 2: 3.1198778623799113
davies bouldin score of hdbscan 4: 2.163204942329579
davies bouldin score of hdbscan 10: 2.930793266609374
davies bouldin score of kmeans 2: 0.9410759092395956
davies bouldin score of kmeans 4: 1.1683782190311085
davies bouldin score of kmeans 10: 1.4289886345000964
davies bouldin score of GMM 2: 1.3092177215329284
davies bouldin score of GMM 4: 2.117127532543442
davies bouldin score of GMM 10: 3.313212799235283
```

The lowest David-Bouldin Scores are for dbscan, kmeans of 2, and kmeans of 4, with lowest score of 0.026 which is lower than the results from previous features.

The Extrinsic Measures:

Adjusted Rand Indices:

```
rand index of dbscan: 4.265433675148219e-05
rand index of hdbscan 2: -0.0001646094791631494
rand index of hdbscan 4: 0.0045720711853102555
rand index of hdbscan 10: 0.005766064787364491
rand index of kmeans 2: 0.0025009058334804395
rand index of kmeans 4: 0.009991732055262341
rand index of kmeans 10: 0.02544958849468978
rand index of GMM 2: 0.004395630051311511
rand index of GMM 4: 0.009013527155037896
rand index of GMM 10: 0.06036364054380597
```

The highest Adjusted Rand Scores are for dbscan, Kmeans of 10, and Kmeans of 4, with highest score of 4.26 which is higher than the results from previous features.

Mutual Information Scores:

```
mutual information score of dbscan: 0.0012584796445544166
mutual information score of hdbscan 2: 0.001742693938171119
mutual information score of hdbscan 4: 0.0045720711853102555
mutual information score of hdbscan 10: 0.005766064787364491
mutual information score of kmeans 2: 0.0009033827559828955
mutual information score of kmeans 4: 0.017297041990934903
mutual information score of kmeans 10: 0.11151861717221759
mutual information score of GMM 2: 0.0018491823109499372
mutual information score of GMM 4: 0.01389310789852527
mutual information score of GMM 10: 0.23223063359894802
```

The highest Mutual Information Scores are for GMM of 10, kmeans of 10, and kmeans of 4, with highest score of 0.23 which is higher than the results from previous features.

Homogenity Scores:

```
homogenity score of dbscan: 0.0018172005747941334
homogenity score of hdbscan 2: 0.0025163890729882014
homogenity score of hdbscan 4: 0.003107895377083114
homogenity score of hdbscan 10: 0.0037946179327212792
homogenity score of kmeans 2: 0.0013044530918992089
homogenity score of kmeans 4: 0.012477224783362781
homogenity score of kmeans 10: 0.048323221598496856
homogenity score of GMM 2: 0.002670154557443912
homogenity score of GMM 4: 0.010021796228526314
homogenity score of GMM 10: 0.10063012485198787
```

The highest Homogenity Scores are for GMM of 10, Kmeans of 10, and Kmeans of 4, with highest score of 0.1 which is higher than the results from previous features.

V-measure Scores:

```
v-measure score of dbscan: 0.0018172005747941334
v-measure score of hdbscan 2: 0.0025163890729882014
v-measure score of hdbscan 4: 0.003107895377083114
v-measure score of hdbscan 10: 0.0037946179327212792
v-measure score of kmeans 2: 0.0013044530918992089
v-measure score of kmeans 4: 0.012477224783362781
v-measure score of kmeans 10: 0.048323221598496856
v-measure score of GMM 2: 0.002670154557443912
v-measure score of GMM 4: 0.010021796228526314
v-measure score of GMM 10: 0.10063012485198787
```

The highest V-measure Scores are for GMM of 10, Kmeans of 10, and Kmeans of 4, with highest score of 0.1 which is higher than the results from previous features.

As we can see, when we use 2 clusters, the models would cluster the dataset based on their gender. When we use 4 clusters it would cluster data based on emotions and at last, when we train the models into 10 clusters, models cluster data based on text-id.

So overall, we can conclude that the results are better with the second set of features in all the implemented models of different clustering methods.