Machine Learning Engineer Nanodegree

Capstone Project

Sri Athithya Kruth B - Feb 14th 2019

Definition

Project Overview

This project is for an open source prosthetic control system which would enable prosthetic devices to have multiple degrees of freedom. https://github.com/cyber-punk-me

The system is built of several components. It connects a muscle activity (EMG, Electromyography) sensor to a user Android/Android Things App. The app collects data, then a server builds a model specifically for this user. After that the model can be downloaded and executed on the device to control motors or other appendages.

This dataset can be used to map user residual muscle gestures to certain actions of a prosthetic such as open/close hand or rotate wrist.

These are some reference papers in this and related domains:

https://core.ac.uk/download/pdf/55626122.pdf

http://human.ait.kyushu-u.ac.jp/publications/mori-icpr2006.pdf

Problem Statement

The Problem statement here is to design a model that will help the user of the prosthetic device play the common rock-paper-scissors game. This will help the user in general model a variety of actions, as the closed fist(indicating rock) and the open hand (indicating papers) can be transferred to usage in other actions as well. (ex) A hi-five needs an open palm.

This is a multi-class Classification problem. It falls into the category of supervised learning.

The given problem can be clearly modelled as a Classification Problem in Supervised Learning. The data given can be cleaned and processed to convert it to a form suitable for processing by any standard Machine Learning Algorithm (CNNs, SVM, Random Forests etc.).

The data will have to split into the training, cross validation and test sets in such a way that there is a sufficient amount of each of the classes in all of the sets so that the algorithm can learn properly, and not be biased in one way or the other.

Metrics

We shall be using two main metrics here, namely accuracy and F1-score.

It is using true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) that we shall be computing these metrics.

True Positive (TP) indicates the numbers of events that were true, and which the classifier predicted to be true.

True Negative (TN) denotes the number of events which were false and which the classifier predicted to be false.

The False Positive (FP) is the number of incorrect predictions that an instance is true.

Finally, False Negative (FN) is the number of incorrect predictions that an instance is false.

	Predicted Yes	Predicted No
Yes	TP	FN
No	FP	TN

Accuracy is a common metric for classifiers; it takes into account both true positives and true negatives with equal weight.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

This metric was used when evaluating the classifier because false negatives and false positives would have an impact on the user experience when trying to use the prosthetic.

We can use accuracy to get a general indicator as to how well the model is performing, whether it is working for some parts of the dataset at least, and how well it is working can be reasonably estimated with accuracy.

Next, we shall be using the F1 score metric which is a harmonic mean of the recall and precision scores achieved by the model.

We can precision as the number of points predicted to be positive out of all the points that were actually positive. Precision is defined as the number of true positives (TP)over the number of true positives plus the number of false positives (FP).

$$\text{Precision} = \frac{tp}{tp + fp}$$

We can define Recall as the number of the correctly predicted positives versus the total number of predicted positives. Recall is defined as the number of true positives (TP) over the number of true positives plus the number of false negatives (FN).

$$\text{Recall} = \frac{tp}{tp + fn}$$

As mentioned earlier, the F1 metric is used. The F1 metric is defined as the Harmonic mean of Precision and Recall.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

There is no impetus to predict with precision or recall in this case, so we can pick the F1 score as a metric as it strikes an even balance between them. In this way, we get a more nuanced understanding of whether the model is capable of predicting the target variable well.

Analysis

Data Exploration

The dataset we are using contains thousands of entries for each of the classes we are predicting. This makes it a well-balanced dataset.

Four classes of motion were written from MYO armband with the help of our app https://github.com/cyber-punk-me/nukleos. The MYO armband has 8 sensors placed on skin surface, each measures electrical activity produced by muscles beneath.

Each dataset line has 8 consecutive readings of all 8 sensors. so 64 columns of EMG data. The last column is a resulting gesture that was made while recording the data (classes 0-3) So each line has the following structure:

[8sensors][8sensors][8sensors][8sensors][8sensors][8sensors][8sensors][8sensors][GESTURE_CLASS]

Data was recorded at 200 Hz, which means that each line is 40ms of record time.

A classifier given 64 numbers would predict a gesture class (0-3). Gesture classes were : rock - 0, scissors - 1, paper - 2, ok - 3.

In the given data set, we have ~2900 items for each of the 4 classes in the dataset. This is a well balances dataset

Exploratory Visualization

To better understand our data, we use the following plots and visualizations:

- Histogram

We plot histograms for all of the input features in our data to help us better visualize the distribution of the data, and to examine the scale of various features in the dataset.

The histograms plotted are given below:

From the histograms, we can see the some of the features are distributed normally – the majority of the data lie in the middle of these graphs.

For others, we see that the data is distributed in a skewed normal form, with most of the data falling before or after the middle point in the X-axis of the graphs.

While some others are distributed somewhat randomly, with a considerable amount of data being around the normal, though the amount of data before/after the normal being varying.

From the histograms, we can see that the different features vary in scale. Some of them and in the range [-50,50] and others in the range [-100,100].

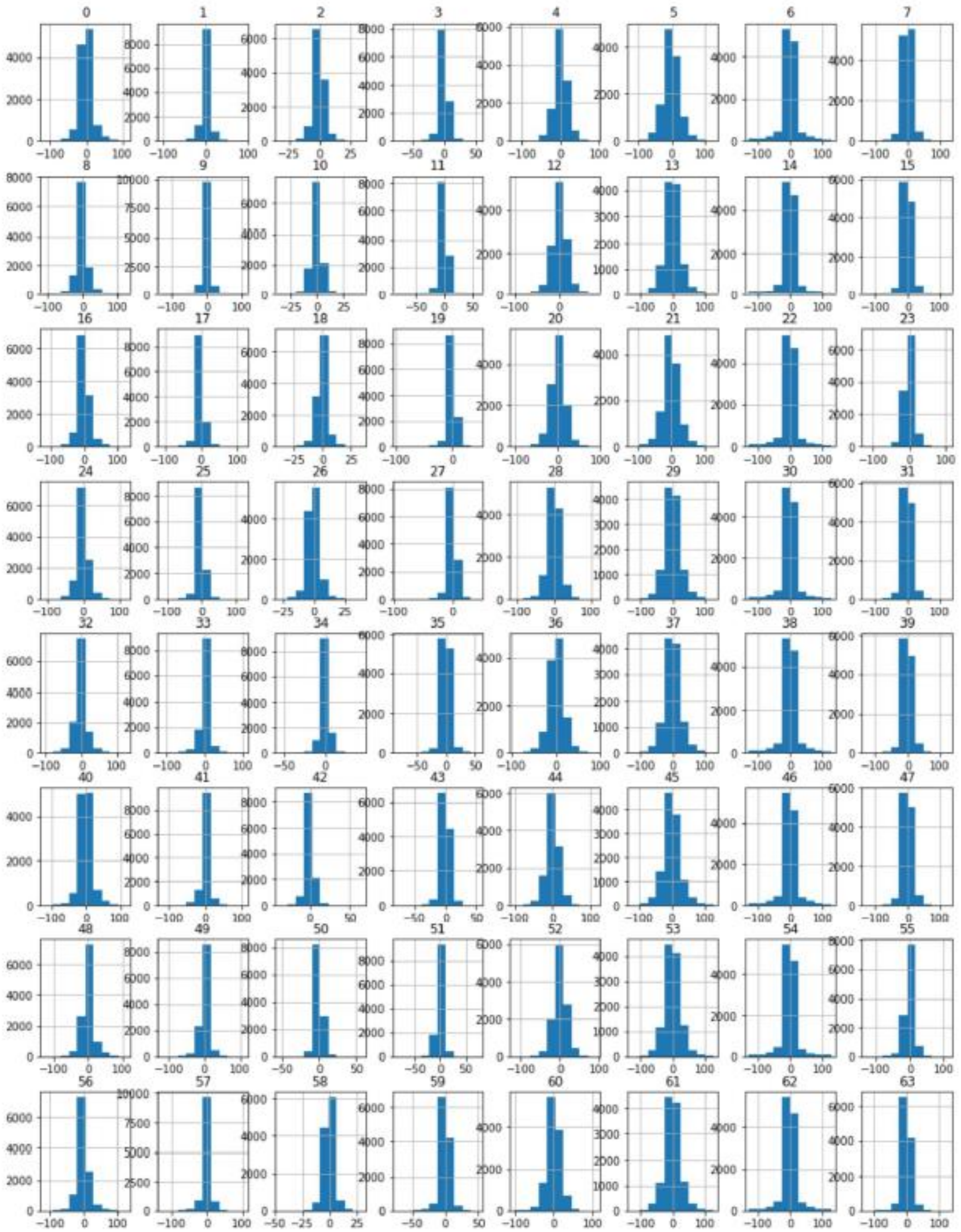


Fig 1. Histogram plots of input features

We can also see that the data is not distributed evenly over the features, or that most of the features do not seem to be in the form of a Gaussian distribution.

- Heatmap

A heatmap helps us indicate the amount of correlation between the features in the data. This is especially important to us, as we plan to perform Principal Component Analysis (PCA).

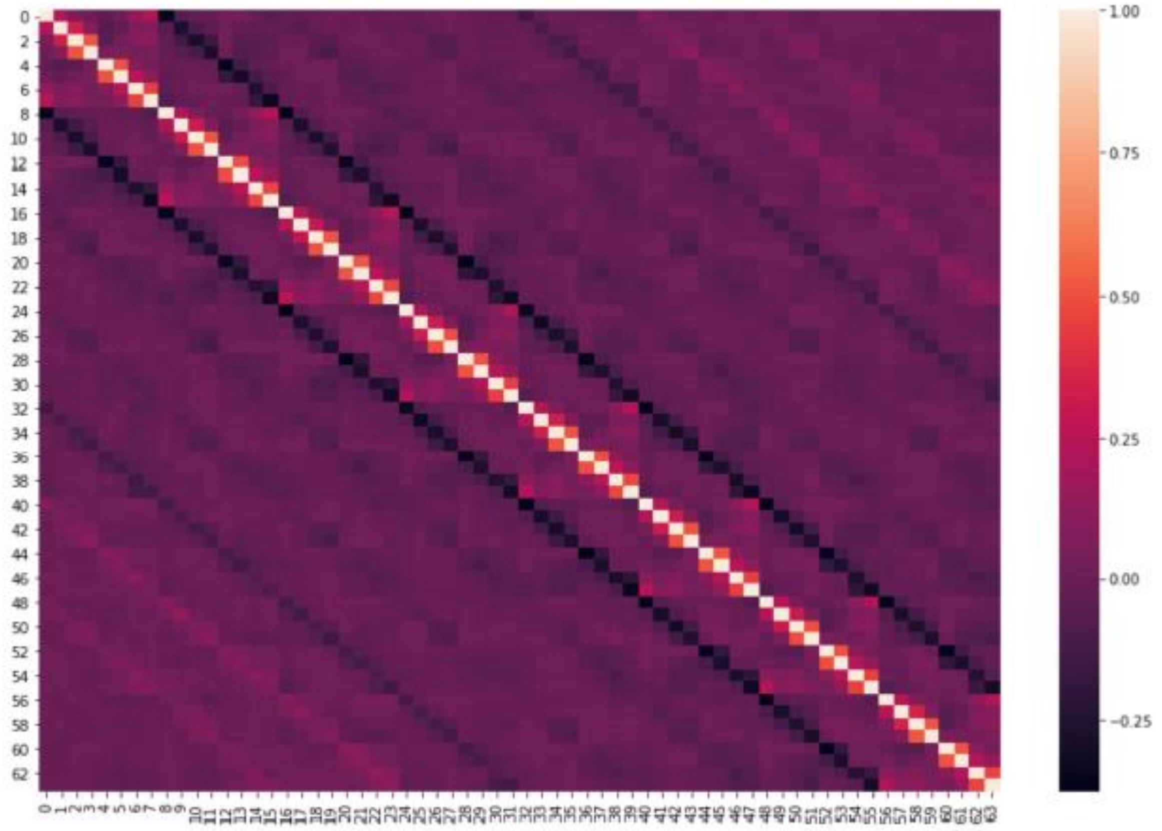


Fig 2. Heat map

The heatmap indicates very little correlation between our features. Considering these features are sensor data from 8 sensors, this could make sense. Also, for most of the features in the dataset we can see the colour generally indicates [0-25 %] correlation in the positive or negative direction.

The above heat map shows that there is little to no correlation between all of the features in our dataset. This could mean PCA would not be a great fit for our dataset.

Algorithms and Techniques

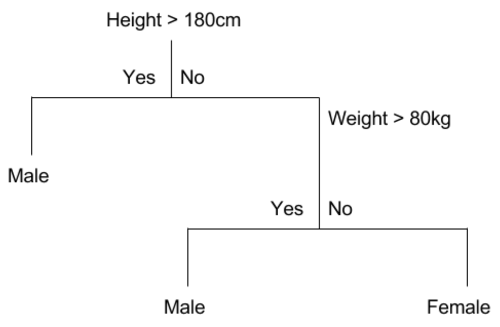
We shall be using a number of classifiers to learn from our data as we are not sure which models would be a good fit. The models we take into consideration are:

- Logistic Regression

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

- Decision Tree

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.



- Random Forests

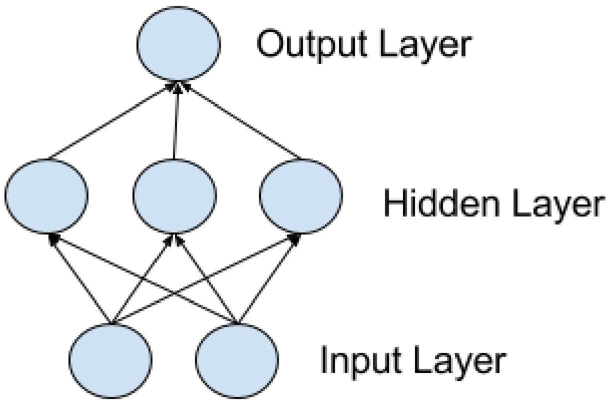
Random Forest is a supervised learning algorithm. Like you can already see from it’s name, it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

- AdaBoost

Ada-boost classifier combines weak classifier algorithm to form strong classifier. A single algorithm may classify the objects poorly. But if we combine multiple classifiers with selection of training set at every iteration and assigning right amount of weight in final voting, we can have good accuracy score for overall classifier.

- Multi Layer Perceptron (MLP)

This is a type of Neural Network – it is comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer.



Benchmark

We can attempt an out-of-the-box Decision Tree on our dataset and use the same for our benchmark model.

The results obtained are tabulated below.

Metric	Score
Accuracy	79.1%
F1-Score	79.1%

Methodology

Data Preprocessing

The steps involved in the pre-processing performed on the data is given below:

- Loading the data

In our dataset, we have 4 separate files where the data for each of 4 classes is stored. We load the data from each file, and then combine it into a single dataset. Following this, we also randomize it so the ordering does not cause any bias in the algorithms we run.

- Checking for null values

We check for any null or empty values in the dataset and we can replace these with a statistic that is representative of the datset (ex) mean, median etc.

- Train-Test split

The data is split into the training and testing sets.

- Feature Scaling

As we were able to see from the above visualizations, the different features in the data are in different scales. So, we shall perform feature scaling on these features so all our data's features are on the same scale. We shall be doing Min-Max scaling for our case.

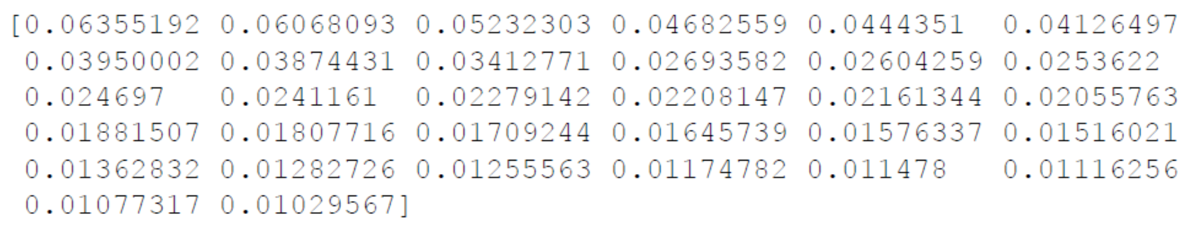
- Principal Component Analysis (PCA)

From our previous visualizations, we are uncertain of whether PCA will be a good pre-processing step to perform on our data. This had posed a challenge that could not be overcome without proper examination.

The nature of the data seems to be that the points do not have much correlation. This would mean that initial attempts at training models on the data transformed with PCA were not fruitful. Proper examination and testing had to be done to get a deeper understanding of correlation (heatmap), and the whether PCA actually managed to capture the variance in a few features (Fig 3).

The basic idea of PCA is to reduce the dimensionality in the dataset whilst capturing the variance in the data as much as possible.

So, when we examine the features obtained as a result of PCA on our data, we see that the variance is distributed in the following manner:



```
[0.06355192 0.06068093 0.05232303 0.04682559 0.0444351  0.04126497
 0.03950002 0.03874431 0.03412771 0.02693582 0.02604259 0.0253622
 0.024697   0.0241161  0.02279142 0.02208147 0.02161344 0.02055763
 0.01881507 0.01807716 0.01709244 0.01645739 0.01576337 0.01516021
 0.01362832 0.01282726 0.01255563 0.01174782 0.011478   0.01116256
 0.01077317 0.01029567]
```

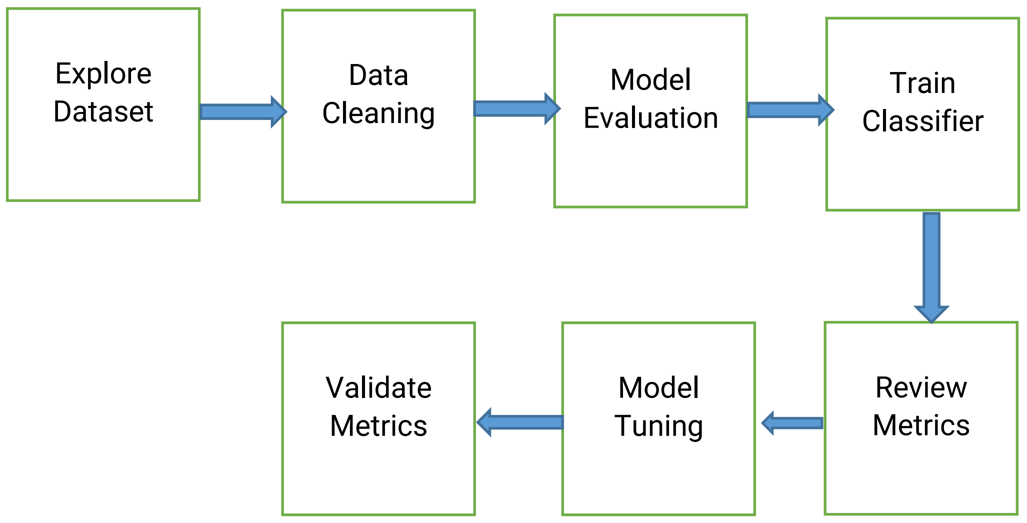
Fig 3. Variance distribution after PCA

As can be seen above, the variance distribution is not localized to only a few features. This means that the variance could not be captured in only a few features. The highest achieved was ~6%.

Also, when we tested the same classifier (Random Forest) before and after PCA, we saw a marked decrease in our metric score after PCA. This lead to the decision to not use PCA after all as a pre-processing step.

Implementation

We streamlined the flow of work in our project according to the below diagram:



According to the above diagram, we explored our data with visualizations, cleaned our data of null values (if any) and perform Min-Max scaling to bring our data to the same scale.

The models we had mentioned were:

- Logistic Regression
- Decision Tree
- Random Forest
- Adaboost

These models were trained on the pre-processed data.
The train-test split used on the dataset was 80-20.
The models, once trained were tested for Accuracy and F1-Score on the testing data.

However, for the Multi-Layer Perceptron, our method of implementation will be slightly different. We one-hot encoded the output variable for this model .We split the data twice to obtain training, cross-validation and testing sets, in the ratio 60-20-20.

The models, once trained were tested for Accuracy and F1-Score on the testing data.

Then we compared the metric scores we had achieved. We found that Adaboost performed the best. After this, we performed model tuning on the Adaboost with Gridsearch on various hyperparameters.

Refinement

In our project, we use K-fold cross validation with K=4 for training all of our models aside from the MLP. This means that the training set would be split into 4, with the model learning from any 3 and validation occurring on the remaining data.

By doing so, we achieved the following results on training:

Model	Accuracy	F1-Score
Logistic Regression	~34%	~34%
Decision Tree	~77%	~77%
Random Forest	91.46%	91.44%
Adaboost	91.56%	91.54%

The neural network used had 3 hidden layers that all used the ReLu activation function, and used dropout layers with an increasing order of dropout value from 0.1. The final output layer comprised 4 nodes with the softmax function.

The results achieved are given below:

```
Accuracy

0.8831335616438356

F1 Score

0.8932669408962979
```

Results

Model Evaluation and Validation

Out of all the models we used, Adaboost was the one we chose to proceed further. To optimize the same, we used a GridSearch with different hyperparameters to tune it.

The different hyperparameters and the values used are given in the table below:

Parameter	Description	Values	Best Value
N_estimators	No of estimators used by the boosting algorithm	[50, 100]	100
Learning Rate	The learning rate controls how much we are adjusting the weights of our network with respect the loss gradient.	[0.01,0.05,0.1,0.3,1, 3, 5, 10]	1

Upon testing the trained model with the testing set, we were able to see the following metrics:

```
Accuracy

0.9225171232876712

F1 Score

0.9225171232876712


Classification Report:
                precision    recall  f1-score   support

           0       0.93      0.98      0.95       584
           1       0.96      0.90      0.93       586
           2       0.91      0.94      0.93       608
           3       0.89      0.87      0.88       558

   micro avg       0.92      0.92      0.92      2336
   macro avg       0.92      0.92      0.92      2336
weighted avg       0.92      0.92      0.92      2336
```

Justification

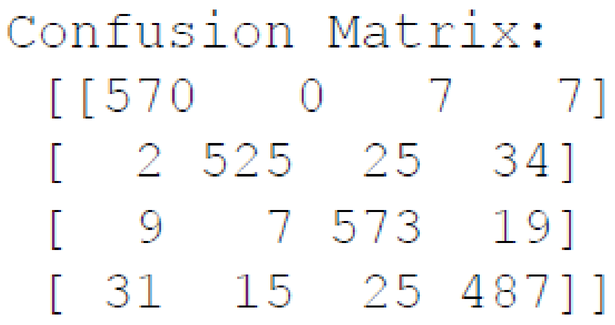
The results of parameter tuning have allowed for a small increase in the scoring metrics we are using. The model in the end, classifies with over 90% accuracy which is a significant improvement over the initial naïve model we used as the benchmark.

Model	Accuracy	F1-Score
Tuned Model	~92%	~92%
Naïve Benchmark Model	79%	79%

Conclusion

Free-Form Visualization

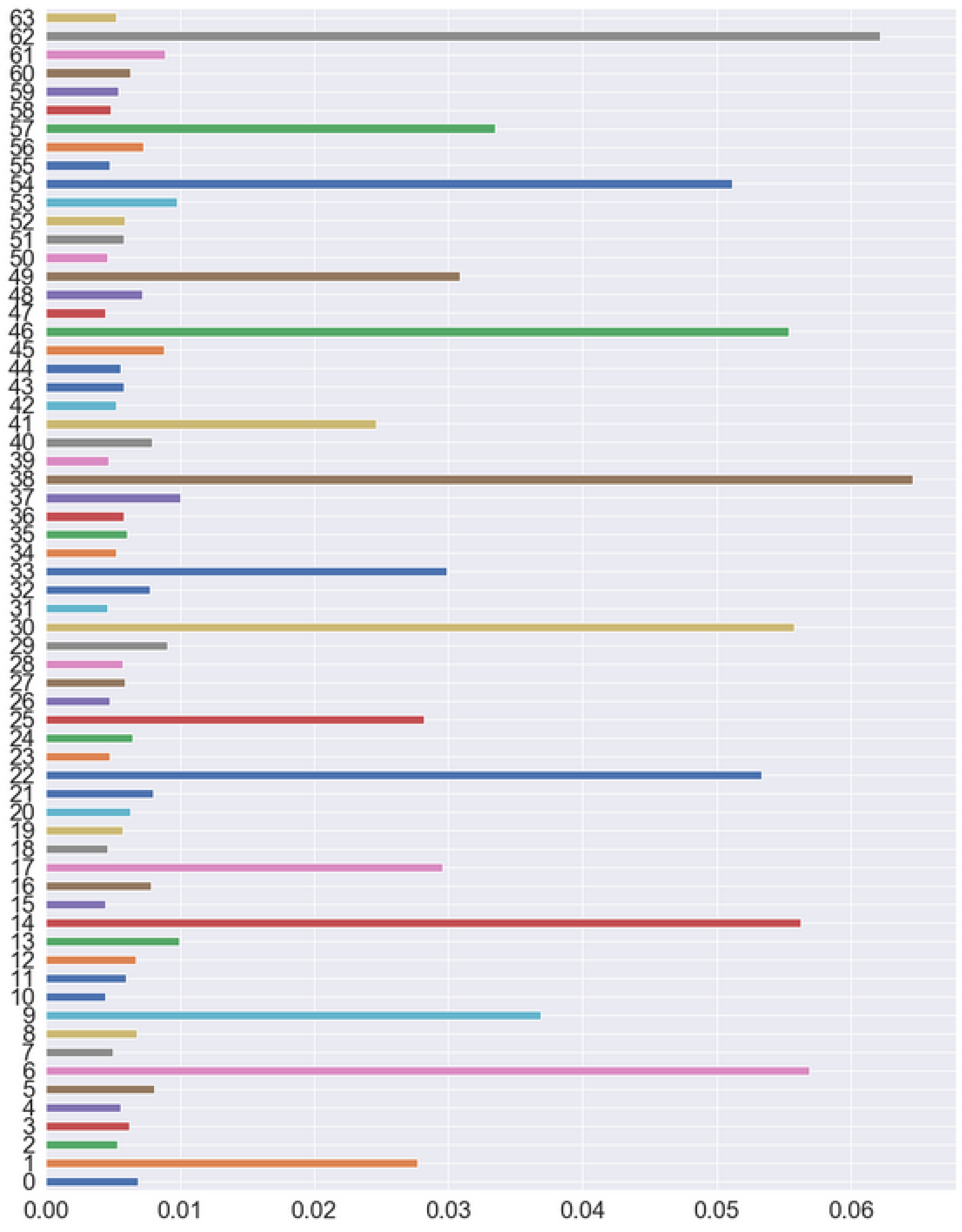
The below Confusion Matrix give us an idea of how the tuned classifier performs on various categories present in the data:



```
Confusion Matrix:
 [[570   0   7   7]
 [  2 525  25  34]
 [  9   7 573  19]
 [ 31  15  25 487]]
```

In addition, this is a plot of the feature importance of the various features used for classification:

(Next Page)



63
62
61
60
59
58
57
56
55
54
53
52
51
50
49
48
47
46
45
44
43
42
41
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0.00
0.01
0.02
0.03
0.04
0.05
0.06

Reflection

The process used for this project can be summarized in the following steps:

- The problem was identified.
- The relevant dataset was downloaded.
- The dataset was preprocessed and cleaned.

This part was slightly difficult, as PCA proved to be an unexpected complication as it did not perform as expected.

Initially I tested several models with PCA, but the results proved to be worse when compared to those obtained without performing PCA at all.

I was confused at this point, as I did not expect this to happen. Going through some of the course material and doing some checking, I remembered to plot a heat map which explained a lot.

This was an interesting challenge as it taught me the importance of visualization and how sometimes, an idea may seem good on paper but when in the actual implementation, proper care has to be taken to examine and evaluate evidence first before proceeding in the same direction.

- A benchmark was created using an out-of-the-box classifier.
- Several possible identifiers were identified, and trained on the data.
- Out of these, Adaboost was chosen as it was the best performing model.
- The model chosen was optimized with Gridsearch.

Doing this project helped me very much in consolidating several concepts I have learnt over the duration on this Nanodegree and applying these in a coherent manner to achieve tangible results.

Improvement

We can perhaps improve our model by training it on more data, if we had the opportunity to acquire more.

From the confusion matrix obtained as a result of testing the tuned classifier, we can see that the model performs well on almost categories except category 3. The model gets a score of >90% on the first 3 categories but the scores for category 3 are slightly lesser in comparison. This is an area where I believe having further data would help improve performance.