CLASSIFICATION AND CLUSTERING OF PHONES BY PHYSICAL SENSOR DATA

Machine Learning Engineer Nanodegree – Capstone Project Report

By Benoît Boucher, 2018-07-28

CONTENTS

1 DEFINITION

1.1 PROJECT OVERVIEW

This is my captsone project for Udacity's *Machine Learning Engineer Nanodegree*. It was made with the collaboration of CONTXTFUL, the company that supplied my data set.

In this project, the primary goal was to identify smartphones' make-model based solely on their physical sensor data. This was measured by a F1 score.

As a secondary target, we tried to cluster the phones to see if a natural separation of the samples occurs based on their make-models. This was measured by weighting the representation of each phone make-models into the clusters.

This work is part of the *Wearable Activity Recognition* field of study, which tries to interpret the context of human interactions based on the readings of physical sensor data.

All the details regarding this project are freely available on my Github repo.

1.2 DOMAIN BACKGROUND

CONTXTFUL is a web marketing company that specializes in using smartphone physical sensor data, such as the gyroscope and acceleration sensors, to detect the context of users. For instance, are they at rest or walking? Are they looking at their screen or not?

Other entities have tackled a similar problem. For instance, academics Ordóñez and Roggen used Deep Convolutional and Long Short-Term Memory (LTSM) Recurrent Neural Networks for Multimodal Wearable Activity Recognition. Among other things, their article shows how to solve two families of human activity recognition problems. Another good example comes from the company *Sentiance* who also used LSTM to analyze sensor data, which allows them to recognize behavioral patterns and interpret real-time context. *Sentiance* calls it The Internet of You. It is also worth citing David Smolders who wrote in his blog about Predicting physical activity based on smartphone sensor data using CNN + LSTM. In his study, he used the Human Activity Recognition Using Smartphones Data Set from UCI.

For CONTXTFUL, interpreting real-time human behavior helps them push context-aware ads to people when they are most receptive.

One of the problems that CONTXTFUL encountered is detecting the type of phone used to generate this data. This is relevant because every phone has different sensors which report their own version of sensor data that can be classified into their different categories (active, at rest, looking at screen, etc.). Knowing the make and model of the phone would allow CONTXTFUL to develop and use make-model specific Machine-Learning models, which would result in faster compute time and more accurate results.

Up until now, CONTXTFUL has been extracting the make-model information from a special collector agent, but this agent has problems of its own and it would be preferable to skip it entirely.

1.3 Problem Statement

In this project, the primary goal was to identify smartphones' make-model based solely on their physical sensor data. This was measured by a F1 score.

As a secondary target, we tried to cluster the phones to see if a natural separation of the samples occurs based on their make-models. This was measured by weighting the representation of each phone make-models into the clusters.

Both targets are actually a continuation of what CONTXTFUL had already done. They used LDA to classify the phones. Doing so, they also noticed that clusters were formed. Therefore, I tried to improve the classifier using a number of supervised learning models, and I verified the clustering using all unsupervised learning models at my disposal.

1.4 Evaluation Metrics

1.4.1 Classification: F1 score

I have used the F1 score instead of the simple accuracy score because my target data was not uniform. F1 score is the harmonic mean of ***precision*** and ***recall*** where ***precision*** describes how many selected items were relevant and ***recall*** scores how many relevant items were selected.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \quad \textbf{EQUATION 1}$$

where:

$$precision = \frac{true\_positives}{true\_positives + false\_positives} \quad \textbf{EQUATION 2}$$

$$recall = \frac{true\_positives}{true\_positives + false\_negatives} \quad \textbf{EQUATION 3}$$

1.4.2 Clustering Score: Homebrew

Because of the unique nature of the problem (specifically: verifying that labels are grouped into unique clusters), I came up with a homemade metric to measure the quality of my clustering. The basis for the calculation is a matrix where the rows are the true labels, the columns are the new clusters, and each ***item(i,j)*** in the matrix is the percentage of samples that has ***label(i)*** and was classified into ***cluster(j).*** Every entry in this matrix will be passed into the following special function:

$$f(x) = 2 \times |x - 0.5| \quad \textbf{EQUATION 4}$$

This function was calibrated to return a score close to 1 if the input ratio is either close to 0 or 1. A good score signifies that the labels were indeed clustered into clusters in an exclusive fashion. In other words, a specific label should be found inside a unique cluster and nowhere else. I then average the score of every item in the label-cluster matrix to give the final score.

There is a caveat to this metric: it is highly sensitive to outliers. Clusters can ultimately contain only one outlier, thus fooling the calculation and generating a very high score.

2 ANALYSIS

2.1 DATA EXPLORATION

The dataset was derived from a measure of a phone's physical sensors. What CONTXTFUL calls a *raw_data_vector* is generated every second and includes all data points from the various sensors (X-Y-Z gyroscope, X-Y-Z accelerometer, etc) sampled at the phone's sample rate. This rate varies by model, but is generally in the range of 16-32 Hz. Moreover, not all phones are equipped with the same sensors. For instance, some make-models do not have a gyroscope. Consequently, *raw_data_vectors* do not have a consistent length. This means that they cannot be used as input for most ML models.

Being proprietary, the *raw_data_vectors* are not available to the public in this project.

What was available for my ML models is what CONTXTFUL refers to as *full_stat_vector*, which is a features vector engineered from the *raw_data_vector*. The *full_stat_vector* is a vector of consistent length containing a slew of various statistical metrics. For instance: maximum_gyroscope_speed_Z, first_derivate_linear_acceleration_Y, mean_euclidean_speed, etc. The total number of features is 1652.

I provided an anonymized version of the *full_stat_vector*, in which specific information about the phones' make-model and the featured statistics has been removed. A simple `Pandas.read_pickle` command will uncompress the data set.

Physical sensor data is generated by phones while users browse websites on which the CONTXTFUL plugin is installed. The specific websites wishes to remain anonymous.

There were 12 110 samples. Since they were obtained from real-life browser visits, the distribution was far from being uniform, as per the bar chart below.

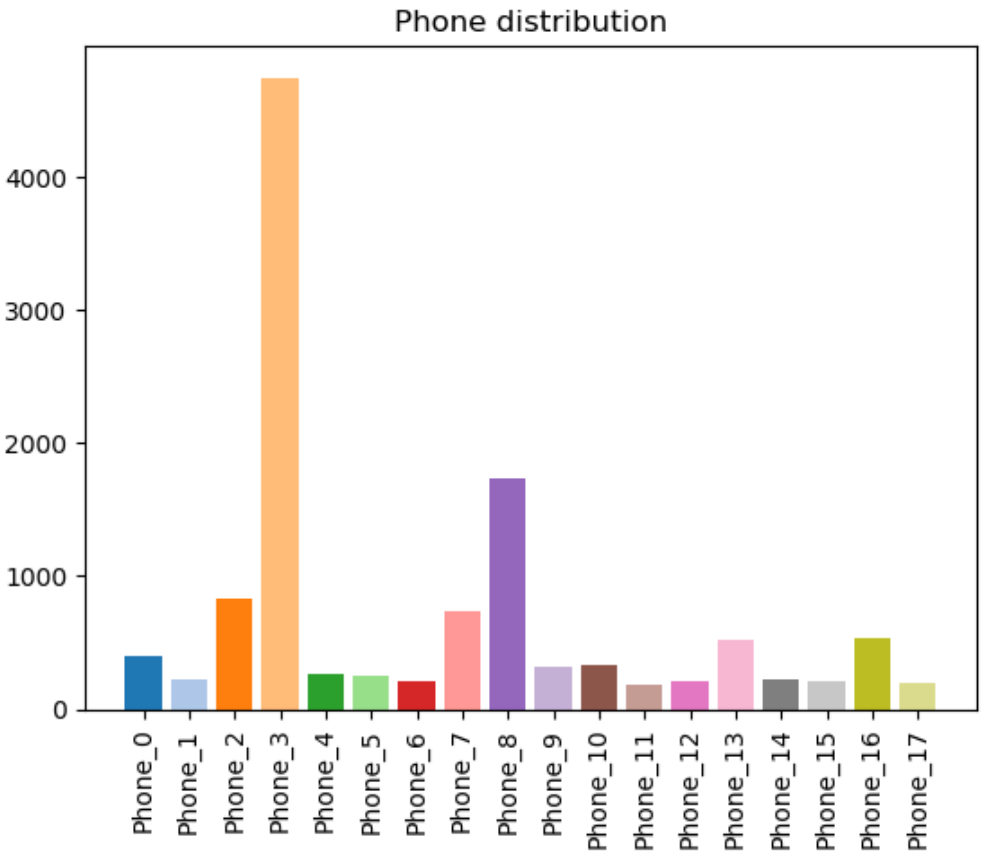

FIGURE 1: QUANTITIES OF EACH MAKE-MODEL.

2.2 Algorithms and Techniques

2.2.1 Classification problem

The solution to the primary goal was to apply a Supervised Learning model. The following algorithms were tried:

- Stochastic Gradient Descent (SGD) using each of the available loss functions
- Neural Network, using various node-layer-dropout combinations
- Random Forest

They were evaluated by their F1 score.

In order to optimize the SGD and Random Forest, we used Random Search Cross Validation.

Because the SGD works best when the input data has 0 mean and 1 standard deviation, we have normalized it.

Just to be sure, we also tried to use normalized data in the other algorithms.

We also looked for the optimal train-test split by plotting learning curves. We did it on all the SGD models, and assumed the results would hold true for the other classifiers.

We follow with more information on each model and why they were used in this setting.

2.2.1.1 SGD

Sickit-learn tells us that SGD is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. It has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. While my data was not sparse, the number of features was rather large (1652, which represents about 1/7 the number of samples). The advantages of Stochastic Gradient Descent are efficiency and ease of implementation (lots of opportunities for code tuning). However, SGD requires a number of hyperparameters to tune and is sensitive to feature scaling.

SGD works by optimizing a loss-function through the Gradient Descent algorithm. There are 9 loss-functions to choose from in scikit-learn and I have tried them all.

2.2.1.2 NEURAL NETWORK

As explained on Wikipedia, artificial Neural Networks (NNs) systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems *learn* to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as *cat* or *no cat* and using the results to identify cats in other images. They do this without any prior knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

A NN is based on a collection of connected units or nodes called artificial neurons which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it.

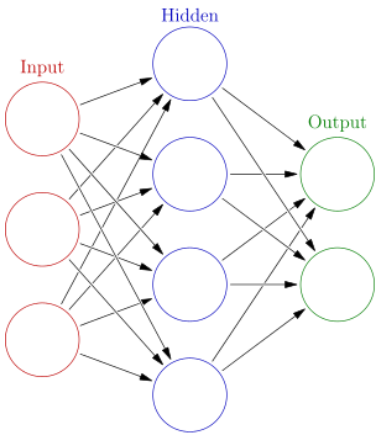

FIGURE 2: (FROM WIKIPEDIA) CONCEPT BEHIND THE NEURAL NETWORK MODEL – INTERNCONNECTED CLASSIFIERS (NODES)

The reason why I chose this model is because it can lead to surprising results which are not linear at all. Historically, they have been at the heart of all the best classifiers, as demonstrated in public contests such as Kaggle and other attention-grabbing media outlets. The downside is that NN are extremely hard to calibrate properly with their large number of hyperparameters and infinite combinations of nodes and layers.

2.2.1.3 RANDOM FOREST

A Random Forest is an *ensemble model*, which is a *wrapper model* that makes its predictions based on the votes of its sub-models. In the case of a Random Forest, the sub-models are Decision Trees.

In a nutshell, a Decision Tree will split the samples into 2 groups based on a criterion. For instance: all samples with temperature>20 go in one group, and all with temperature<=20 go in another. A series of such decisions is made until the samples are sufficiently split into the desired labels. To keep with the example, perhaps the last layer could be: presence of snow, yes or no? Once the training data is sufficiently split, the model would be able to make a prediction on new samples by having it go through the same decision process, answering all the questions in series, until it lands in a classification label; in this example, which season is it?

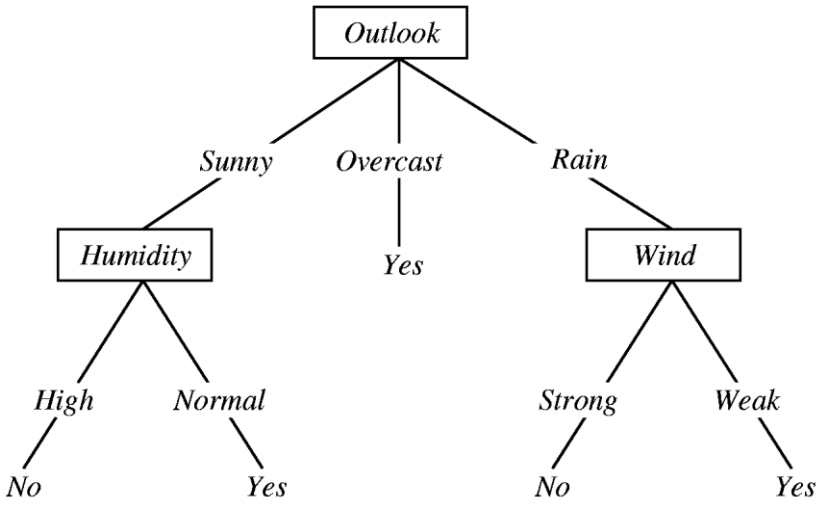

FIGURE 3: (FROM JOSÉ M. VIDAL) EXAMPLE OF A DECISION TREE

I picked this model because it can perform very well even with heterogeneous data. Also, I chose the forest as a way to compensate the tendency of single trees to overfit the data. Moreover, it is a conceptually different approach than the previous two models explored, so I was expecting results that would be very different as well, hopefully for the better.

2.2.2 CLUSTERING PROBLEM

As for the solution to the clustering goal, we tried the following algorithms:

- Mean Shift
- Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
- Agglomerative
- K Means

Mean Shift and DBSCAN have no hyperparameters to optimize, while we can only adjust the number of clusters in Agglomerative and K Means. Therefore we have tried manually various numbers of clusters until we found a satisfying result, which was determined by the homemade clustering metric.

The main reason I picked these algorithm is because they represent a wide array of unsupervised algorithms. Since I did not know what to expect from my data, and because I was looking for a specific behavior (labels clustered naturally in an exclusive fashion), I had to try many approaches that were significantly different from each other.

We follow with more information on each model and why they were used in this setting.

2.2.2.1 MEAN SHIFT

Mean shift clustering aims to discover *blobs* in a smooth density of samples. It is a centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids.

Since Mean Shift is good for many clusters of uneven size, I thought it would be appropriate given the uneven distribution of my target labels.

2.2.2.2 DBSCAN

The central component to the DBSCAN is the concept of *core samples*, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples).

This algorithm fits my data because it scales well with large numbers of samples and produces uneven cluster sizes, which is appropriate given the uneven distribution of target labels.

2.2.2.3 AGGLOMERATIVE

The Agglomerative Clustering object performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together.

This model might lead to good results because it uses non-Euclidean distances, which is well suited for a collection of statistics like my *full_stat_vectors.*

2.2.2.4 K MEANS

The K Means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the *inertia* or *within-cluster sum-of-squares*. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

I tried it because it is general purpose and scales well with sample size and feature size.

2.3 BENCHMARK

For the classification problem, CONTXTFUL has obtained good results using Linear Discriminant Analysis (LDA).

2.3.1 CLASSIFICATION BENCHMARK

I applied my chosen metric to obtain the **Classification Benchmark: F1 score of 78.33%.**

Because LDA is a feature-reduction technique, CONTXTFUL had the idea to plot the labels against the two axes that explained the most variance. Specifically, they explained 80% and 5% of the variance.

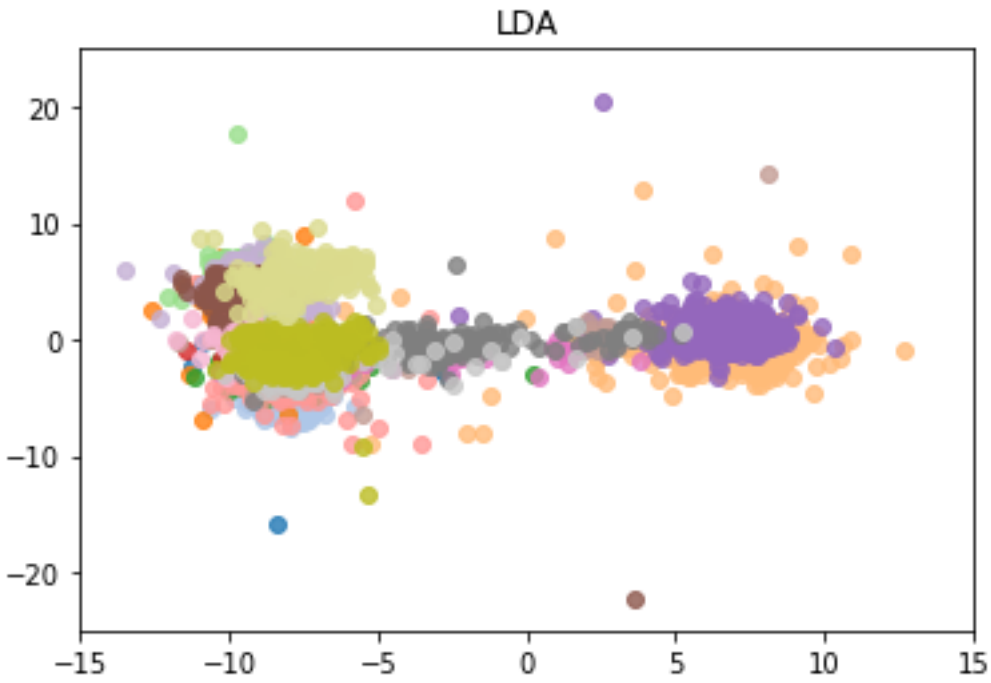

FIGURE 4: PLOTTING OF PHONE LABELS ON THE FIRST 2 TRANSFORMED AXIS OF THE LDA.

Note that we had to zoom on the zone of highest concentration because there are outliers.

In this graph, each color represents a specific make-model. As we can see, clusters have formed. However, those were *supervised* clusters. The next logical step was to try *unsupervised* clusters and see if the phones would still group up into clusters in an exclusive fashion.

Unfortunately, the results were not as conclusive as they had hoped. Their current best model at the time of writing was a Gaussian Mixture, which was optimized using the Bayesian Information Criteria (BIC) score.

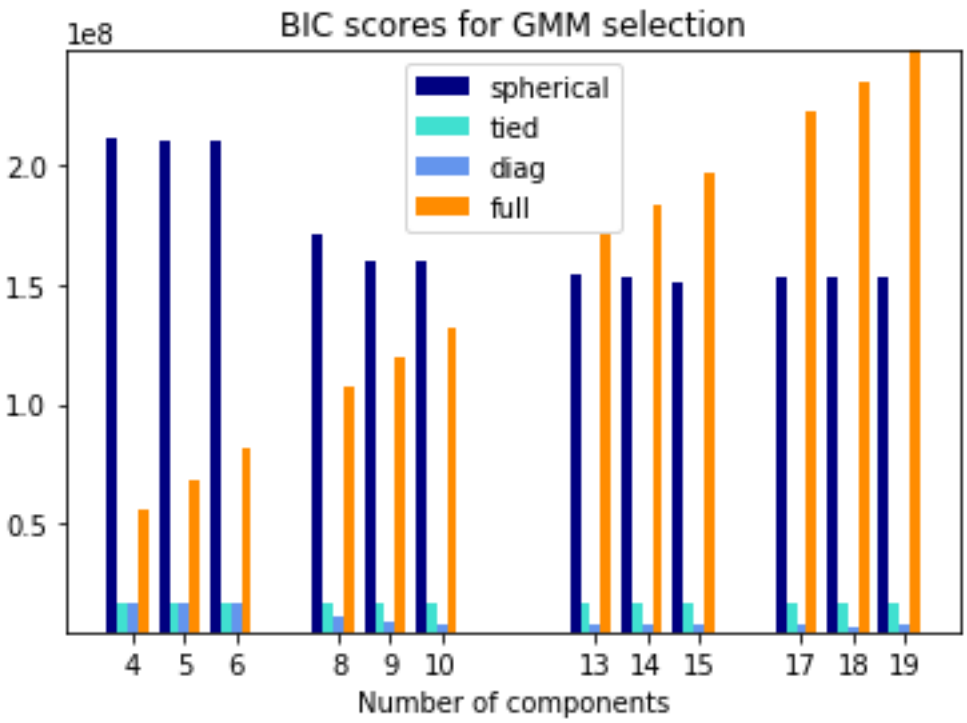

FIGURE 5: BIC SCORES USED TO DETERMINE THE BEST GMM HYPERPARAMETERS. LOWER IS BETTER.

Since a lower BIC is best, the "diag" cv_type is the best for calculating the variance. As for the number of components, 18 seemed to be the best, but others were also considered since they had similar scores. We started with 18 because it also corresponds to the number of labels. The following graph shows the results of the GMM clustering with diagonal variance and 18 clusters. Each subplot is a cluster, and each bar represents the ratio of a specific phone in that cluster. To make interpreting easier, note that we made the colors consistent across all subplots (and across all graphs found in this report). Thus the dark red bar will always represent Phone_6.

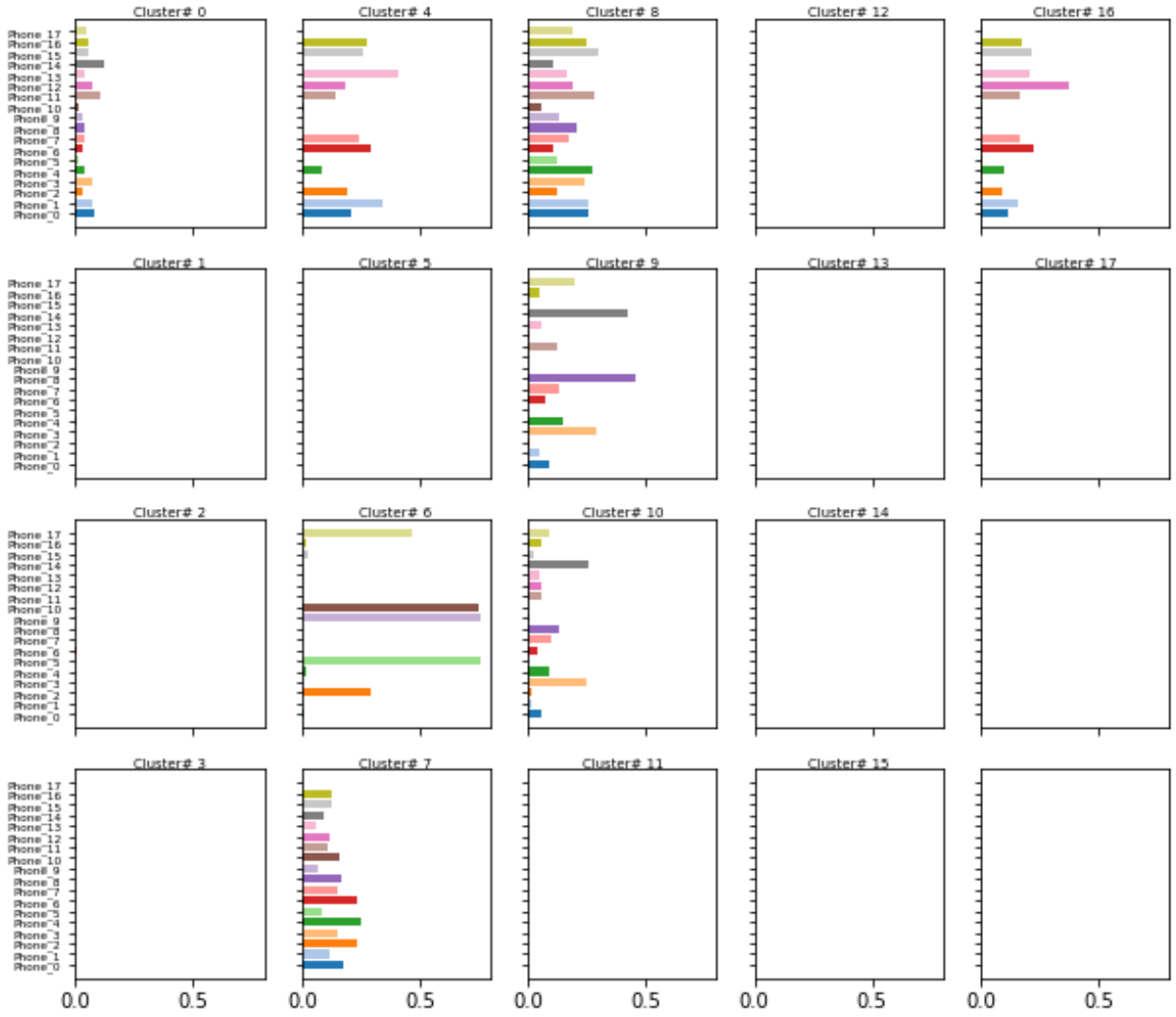

FIGURE 6: RESULTS OF THE GMM CLUSTERING USING THE BEST HYPERPARAMETERS (DIAG COVARIANCE AND 18 CLUSTERS).

The first striking observation is that half of the clusters are "empty" (specifically, they contain exactly 1 sample each). The second observation is that some clusters are sparse, like #4 and #10, which is what we were looking for, but some others contain a uniform quantity of all phones, like #7 and #8, which is the opposite of what we wanted.

CONTXTFUL then tried other cluster sizes. The problem of having half of the clusters empty persisted. In the end, they decided that the best results were obtained with 8 clusters, as shown next.

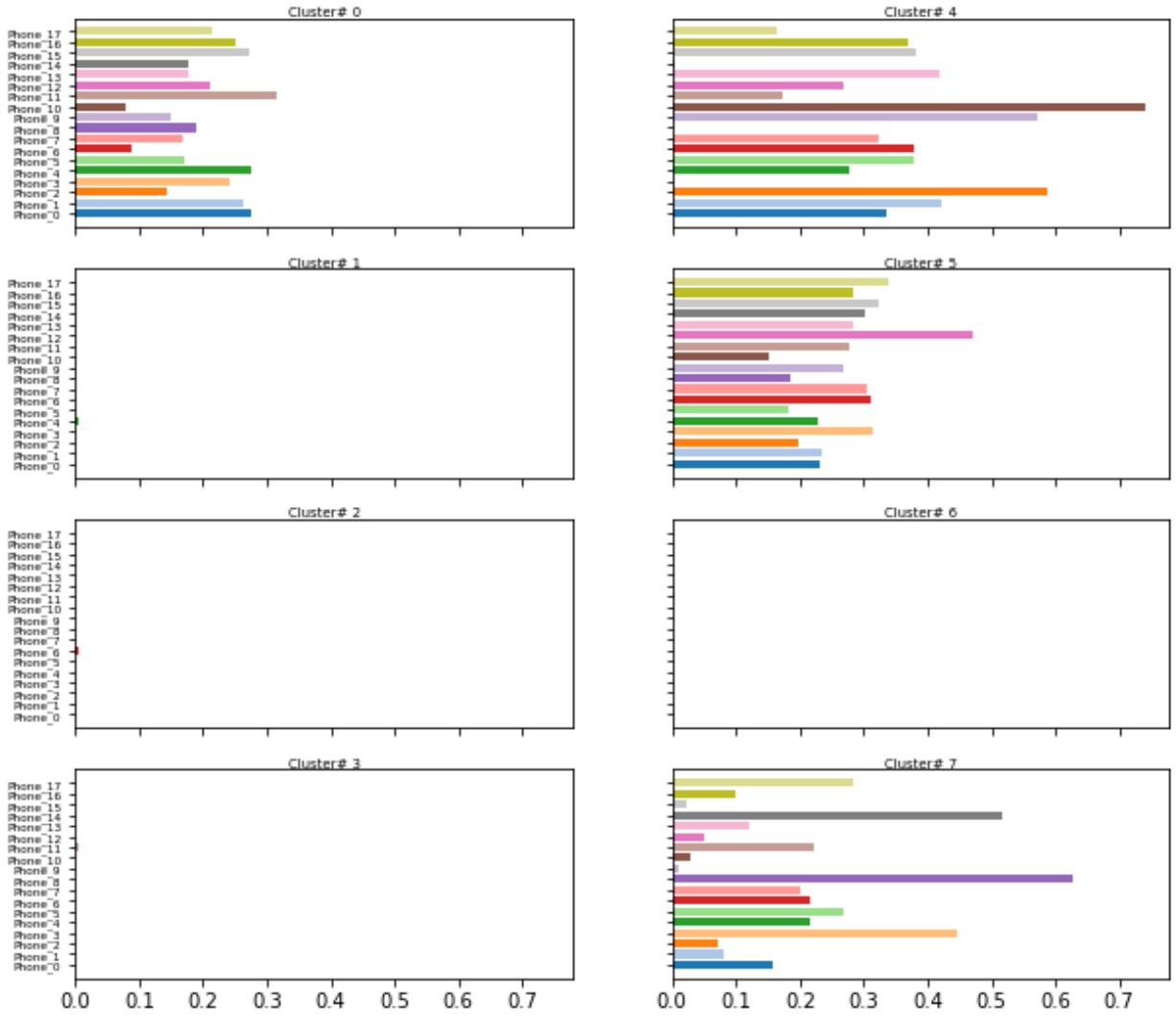

FIGURE 7: CLUSTERING BENCHMARK - GMM WITH 8 CLUSTERS.

2.3.2 Clustering Benchmark

Applying the homebrew clustering metric, we obtained a **Benchmark Cluster Score of 76.51%.**

3 Methodology

3.1 Data Preprocessing

As mentioned in the ***Data Exploration*** section, the dataset was derived from a measure of a phone's physical sensors. We took *raw_data_vectors* which contained raw sensor inputs and calculated 1652 different statistics on it to obtain the *full_stat_vectors.* Examples of specific features include second_derivate_angular_speed_X and min_linear_acceleration_Y.

Being proprietary, both the *raw_data_vectors* and the details of how they were transformed into *full_stat_vectors* are not available to the public in this project.

I provided an anonymized version of the *full_stat_vector,* in which specific information about the phones' make-model and the featured statistics has been removed. A simple `Pandas.read_pickle` command will uncompress the data set, which contains 12 110 samples.

After that, standard pre-processing was undertaken. Classification tasks require the data to be split into a testing set and a training set.

Also, some algorithms respond better when data is normalized.

No other special steps needed to be taken. For one, the missing values have been taken care before obtaining the *full_stat_vectors*. Second, by their nature, they are only comprised of floating points, so neither string transformation nor one-hot-encoding were necessary.

3.2 Implementation

3.2.1 Libraries

Data processing as well as all the clustering and classification algorithms was done using the scikit-learn library in Python3; all except the neural network, which was implemented using Keras.

While normalization and Principal Component Analysis (PCA) reduction were tried on all models, PCA never helped and normalization was sometimes helpful in clusters and completely required in SGD.

The supporting functions found in Utilities.py can be classified in two groups: one is about plotting graphics, and the other is about calculating metrics. The metrics were then either reported directly, as was the case for the homebrew Cluster Score, or used to feed the graphic methods, just like the Label Cluster Matrix. In Utilities.py, some functions were inspired by work found on the web. Credit is duly given in the appropriate functions' doc-string.

3.2.2 Selecting algorithms

The process of picking an algorithm was more random than anything. I tried every clustering technique available and manually optimized the hyperparameters (number of clusters) by hand. My initial goal was not to try them all, but none of them provided the results I was looking for.

When doing classification, I picked my initial models on the fact that they were from different families. Specifically, SGD was a linear model, Neural Network was a neural model, and Random Forest was both an ensemble and a tree method. None of my models overlapped each other in terms of techniques.

3.2.3 Cluster Score weakness

Going through this process, I realized that my homebrew metric had a weakness: "empty" clusters greatly inflate it. However, empty clusters occur when the actual clustering fails. Thus, my worst models ended with the best Clustering Score. I solved this problem by ignoring any score obtained from a non-cluster. However, it would have been preferable to address it in a more robust way, one that preferably does not need human analysis.

Below is a good example of what I call "non-clusters". It is the result of the DBSCAN technique. The phones were almost all grouped into the first cluster, then there are 30 clusters that contain a single sample each, and finally,

there is the Cluster #-1, which contains the items that were "rejected" by the scan. While qualitatively, this is an extremely poor result, the quantitative metric was a remarkable 96.4%!

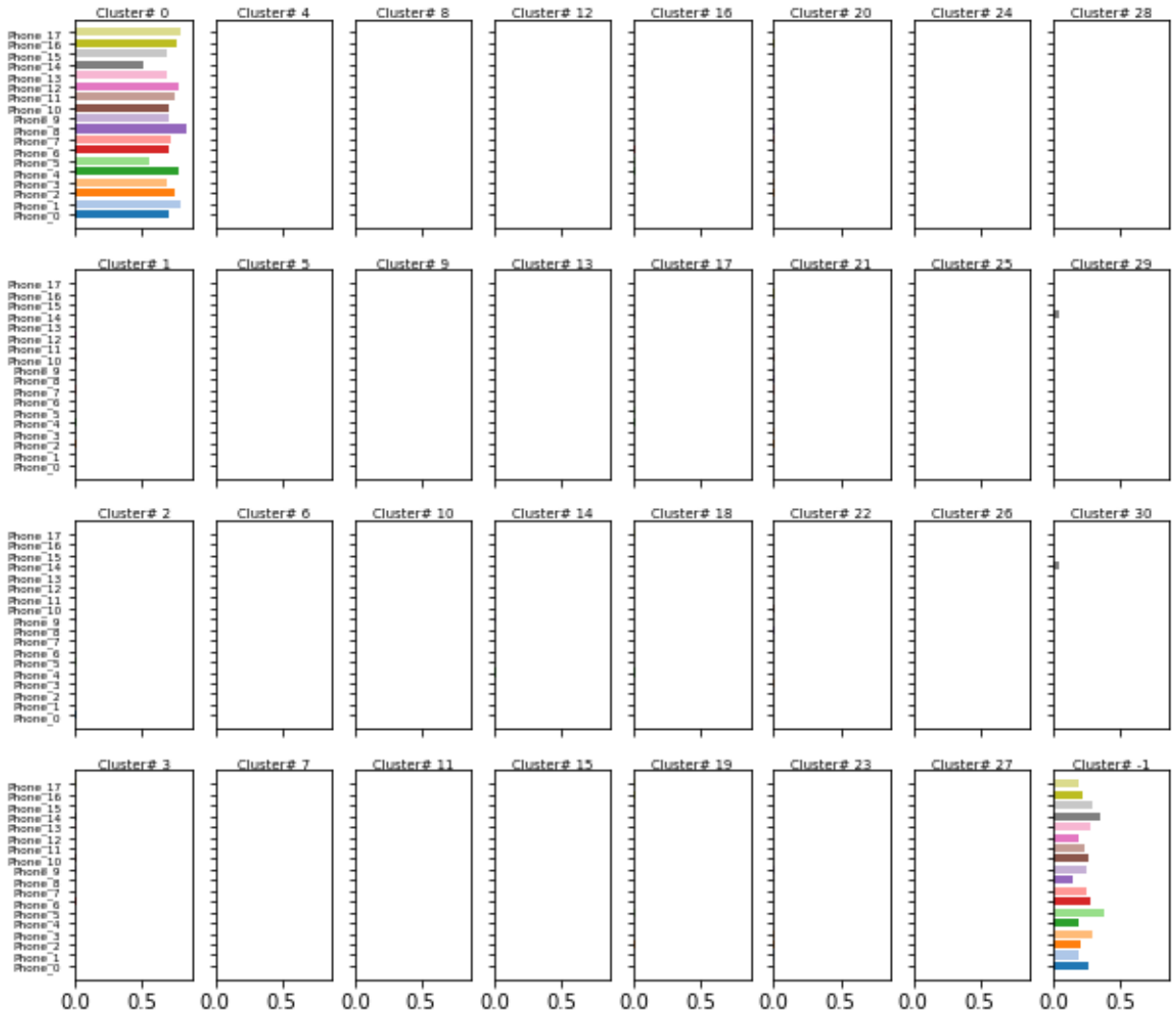

FIGURE 8: ILLUSTRATION OF THE HOMEBREW CLUSTERING METRIC'S WEAKNESS USING THE RESULTS OF DBSCAN. THE SCORE WAS 96.4% EVEN THOUGH NO TRUE CLUSTERING TOOK PLACE.

3.2.4 Hyperparameter Optimization

I optimized the SGD and the Random Forest with RandomSearchCV. This was the best technique known to me in order to efficiently find the best hyperparameters.

A special mention should be made to the log_uniform function. It was created to support the RandomSearchCV method, because the only useful functions available out-of-the-box were the plain uniform distribution functions, whether applied to integers or floats. Because the variation of the number I wanted to sample was of 2 orders of magnitude, I implemented a log-uniform probability distribution function which followed the model of statistical functions in the scipy library. That distribution function was much more appropriate than its regular counterpart in this case because the regular function would have, statistically speaking, favored picking numbers in the larger order of magnitude.

3.3 REFINEMENT

As was mentioned in the ***Implementation*** section, I used 4 techniques to refine my results:

1. Manual hyperparameter selection. This was applied to the number of clusters in clustering technique and to figure out the best Neural Network configuration.
2. BIC score. This was applied to GMM.
3. Learning Curves. This was done to identify the best loss-function in SGD and the best train-test split.
4. RandomSearchCV. This was done for SGD and Random Forest classification.

3.3.1 TRIAL AND ERROR APPLIED TO CLUSTERING

Manual trial and error proved very efficient in finding out if a clustering algorithm was successful or not. To illustrate:

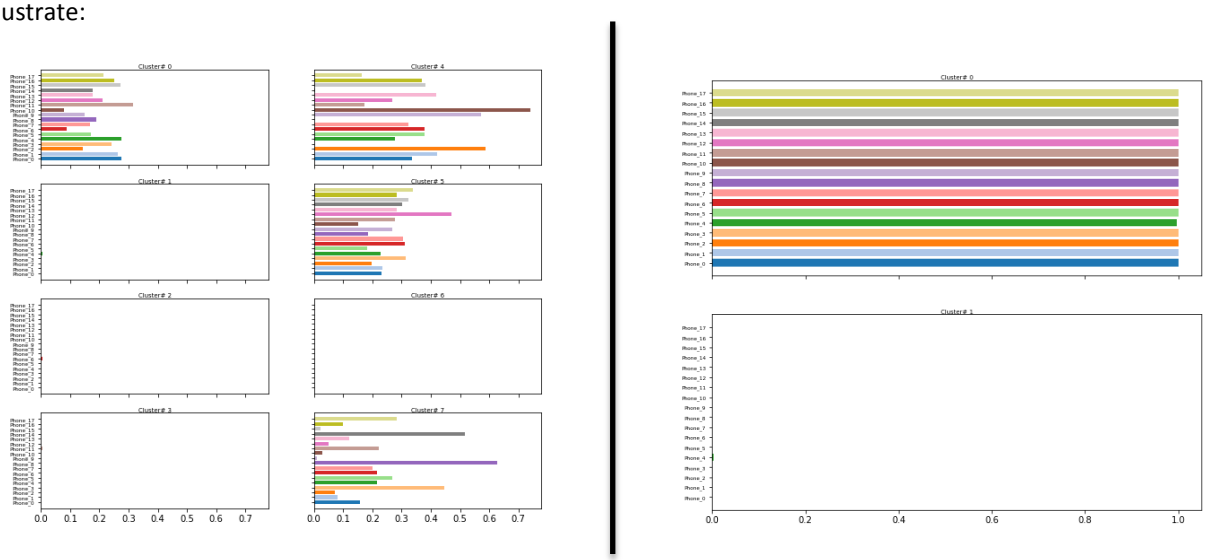

FIGURE 9: COMPARISON BETWEEN GMM WITH 8 CLUSTERS (LEFT) AND GMM WITH 2 CLUSTERS (RIGHT).

The figure of the left is GMM with 8 clusters. It clearly separated the phones better than on the right, where GMM was set to 2 clusters.

3.3.2 TRIAL AND ERROR APPLIED TO CLASSIFICATION

I created a neural network (NN) and fed it PCA transformed data. The accuracy got stuck to 39% throughout all epochs.

Then I fed it untransformed data, and the accuracy increased throughout training up to 70%.

Then I played with the layer configuration and eventually, my best score was 76.13%.

3.3.3 BIC

As explained in the ***Benchmark*** section, BIC helped me determine that the "diagonal" type covariance was the best. It also suggested what the best number of clusters would be, even though I ended up trying a couple of different ones.

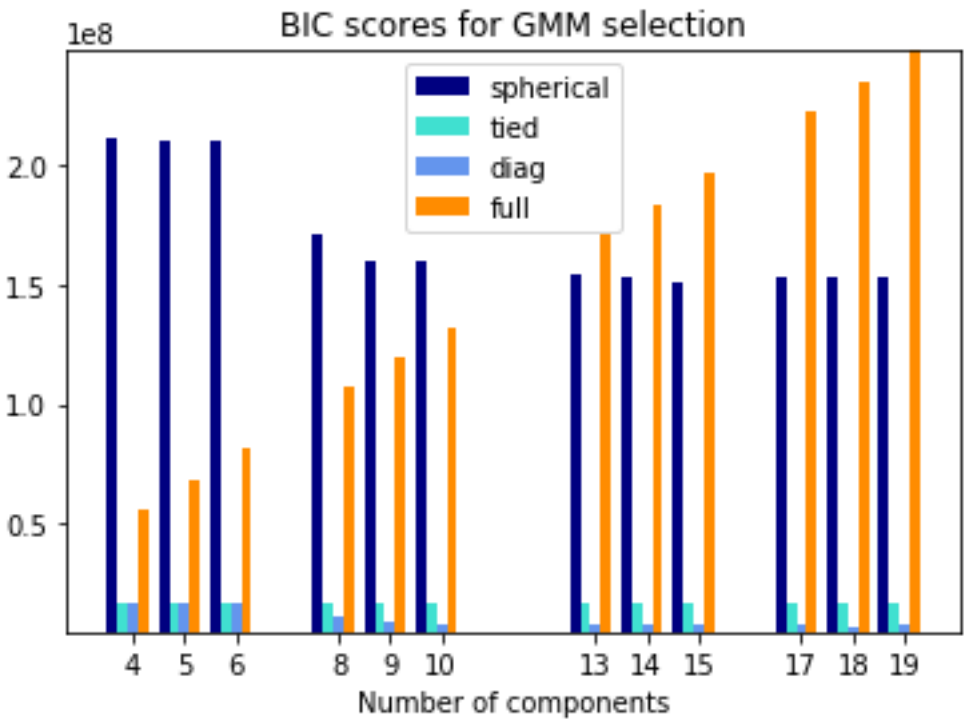

FIGURE 10: (RECAP OF FIGURE 3) BIC SCORES USED TO DETERMINE THE BEST GMM HYPERPARAMETERS. LOWER IS BETTER.

3.3.4 LEARNING CURVES

In total, I plotted 9 learning curves; one for each loss function readily available in scikit-learn's SGD classifier. The highest one told me which loss-function to use. The ensemble let me know that 75-25 train-test split was ideal in order to get a good accuracy without overfitting. Below are two examples:

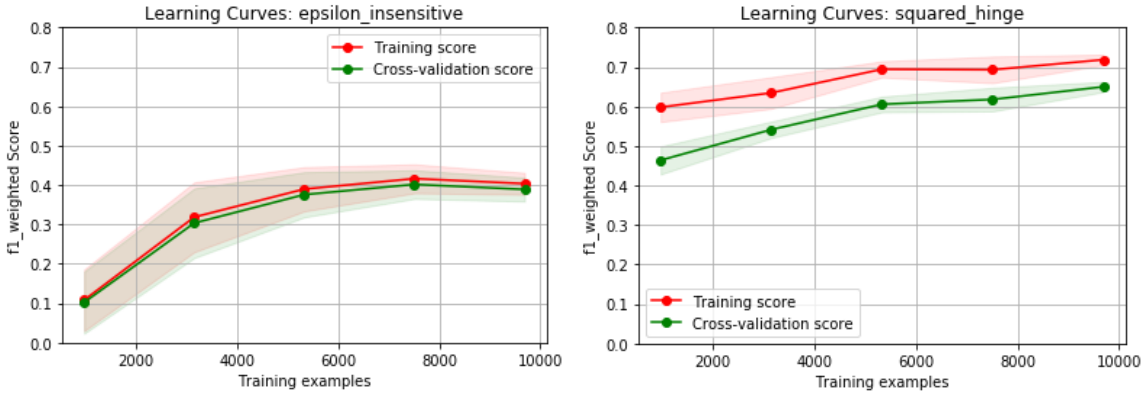

FIGURE 11: TWO OF THE 9 LEARNING CURVES CALCULATED, EACH USING A DIFFERENT LOSS-FUNCTION.

3.3.5 RANDOMSEARCHCV

The unoptimized SGD_squared_hinge F1 score was 63.58%. After going through random search, the score improved to 69.95%. While not jaw-dropping, the different was still significant.

I did not see the point in comparing before and after scores for the RandomForest. I just did random search.

4 RESULTS

4.1 MODEL EVALUATION AND VALIDATION

4.1.1 CLUSTERING

My best clustering model was the K Means. It is the best purely because it is the only one that was able to create clusters with a significant number of samples in every cluster.

Normalization of the input data is critical for it to converge.

The only hyperparameter of this model is the number of clusters, for which it was obviously sensitive to.

4.1.2 CLASSIFICATION

My best classification model was the Random Forest. Like most trees, it was prone to overfitting, so I used a cross-validation split of 66%-33% when optimizing the hyperparameters. Also, this model has many hyperparameters, so it was somewhat tricky to get to work perfectly.

In the end, the result of my random-search cross-validation gave the following recommendation for best hyperparameters:

- criterion: entropy
- max_features: 286
- min_samples_split: 7
- n_estimators: 45

While the default *criterion* is the Gini impurity, it appeared that entropy (information gain) was better with my data.

Empirical good default value *for max_features* is the square root of the number of features. In my case, that would be 41, which is seven times smaller than what was obtained through cross validation.

The default value for *min_samples* is 2, which therefore develops the tree fully. However, the random search determined that 7 would be better. This likely reduced overfitting significantly.

Finally, a high number of estimators will generally lead to a better model, but there is an asymptotic limit. There is also a cost in terms of computing power associated with each additional estimator. Thus, a compromise needs to be stricken. Of course, since I used Random Search, the compromise is heavily skewed towards a higher accuracy, so I ended up with 45 as the best value.

As for dimensionality reduction, my model was not sensitive to data normalization, but using PCA would break it.

4.2 JUSTIFICATION

4.2.1 CLUSTERING

The benchmark (GMM 8 clusters) is on the left, and my best model (KMeans 8 clusters) is on the right.

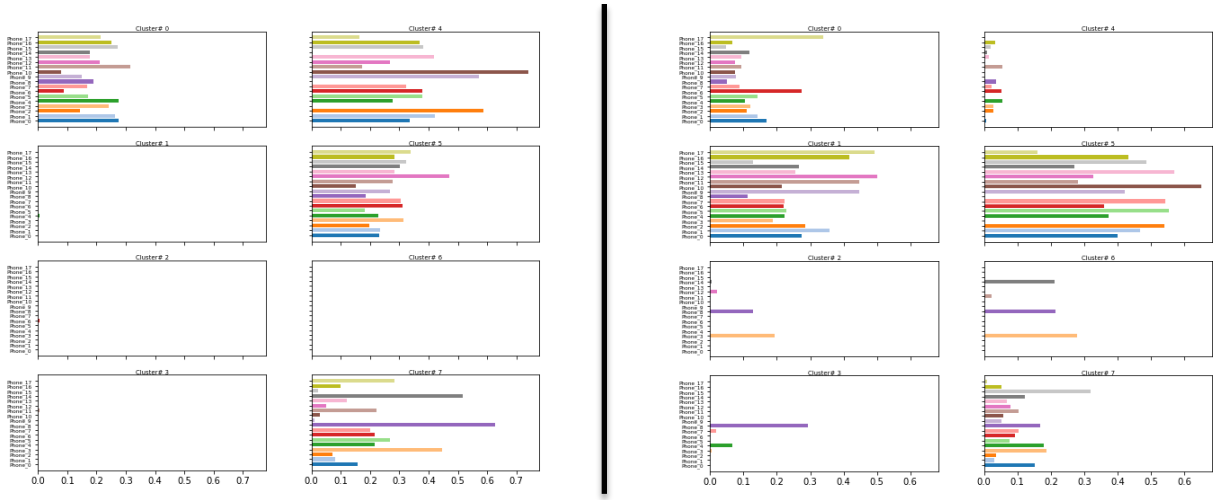

FIGURE 12: COMPARISON BETWEEN THE BENCHMARK (LEFT – GMM 8 CLUSTERS) AND MY BEST MODEL (RIGHT – KMEANS 8 CLUSTERS)

GMM Cluster Score: 76.51% **KMeans Cluster Score: 76.00%**

Visually, we can see that the benchmark contains empty clusters. Because of that, KMeans is clearly superior. Also because of that, the Cluster Score of KMeans should be considered higher. As explained in the ***Methodology/Cluster Score Weakness*** section, empty clusters inflate the score unduly. Therefore, an adjusted 76.51% for the GMM would most likely be much lower than the non-adjustment-required 76.00% of the KMeans.

So, while I consider my model to have beaten the benchmark, I would not go as far as saying that it solved the problem of clustering labels in an exclusive fashion. Clearly there are homogenous clusters, and the sparse clusters contain labels that are seen in other clusters.

4.2.2 CLASSIFICATION

- LDA F1 score: 78.33%
- Best RandomForest F1 score: 90.22%

The improvement of more than 10% in absolute terms is very significant. Also, any classifier above 90% should be considered very good.

Still, we know that the data was not uniformly distributed, so is this score so high only because of the accuracy of Phone_3 is better? This is the very reason why we used the F1 score, but let's look at the accuracy score per class to be sure.

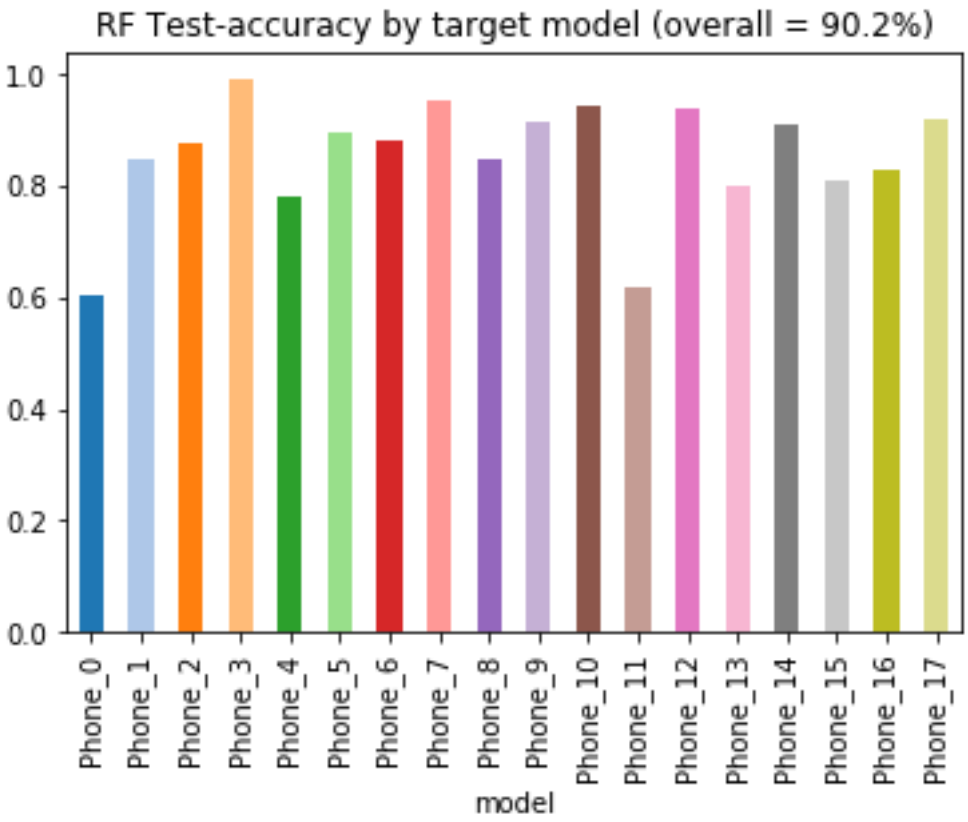

FIGURE 13: RESULTS OF MY BEST MODEL, THE RANDOM FOREST, SUBDIVIDED INTO EACH TARGET LABEL.

We can see that yes, Phone_3 has the highest accuracy, but most of the labels are around 90% too. Only Phone_0 and Phone_11 have objectively low scores in the 60%. After these two, the lowest goes to Phone_4, which beats the benchmark overall. Those are the reasons why I believe that this model solved the problem convincingly.

5 Conclusion

5.1 Free-Form Visualization

I would like to show the relative importance of the features used by the Random Forest.

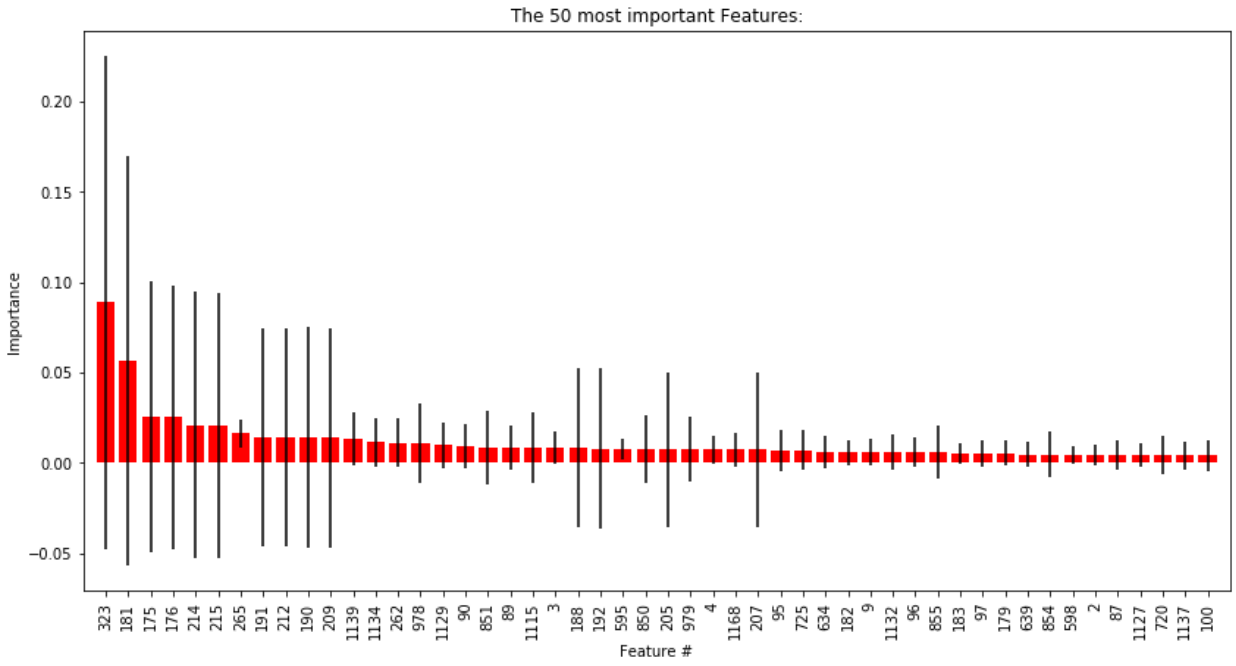

FIGURE 14: A LOOK UNDER THE HOOD OF THE BEST MODEL (RANDOM FOREST) – THE 50 MOST IMPORTANT FEATURES FOR THE CLASSIFIER.

In the plot above, the red bars are the feature importance of the forest, along with their inter-trees variability in black.

It is interesting to notice how there are really 2 main features despite having 1652 to choose from. Granted, their relative importance varies wildly amongst the trees. But once you get to 40+, they are all rather insignificant. On a funny side note, it would appear that all the trees agree that feature #265 is always the 7th most important.

5.2 Reflection

5.2.1 Summary

My Udacity class in Machine Learning Engineer required a Capstone project in order to award me my Nanodegree. I chose to work with CONTXTFUL, a web marketing company, to obtain my dataset. The data is comprised statistics about data obtained from the phones' physical sensors. The goal was two-fold: first, be able to classify the phones' make-model and second, see if an unsupervised clustering algorithm would naturally separate the make-model. In order to solve the classification problem, I tried various classifiers. I optimized them before comparing their F1 score to the benchmark. For clusters, I tried all the unsupervised learning algorithms that I know and compared them using a homebrew metric. In the end, I beat the classification benchmark significantly, but the clustering was not successful.

5.2.2 Interesting aspects

It was very interesting to use the same set of data for a different purpose than what it was intended for in the first place, and still obtain results. Specifically, the sensor data was collected to determine the context of a user (sitting, walking, looking at phone, etc.) but instead I managed to identify the phones' make-model with it.

It was also quite exciting to devise a homebrew clustering score because the situation demanded something that has not been done (or documented and easily found) before. And even though the chase for a perfect clusterer proved unfruitful, the search for the unknown was fun.

5.2.3 Difficulties

I did not foresee that so many clusterers would fail to simply separate the data at all. This lead to a poor metric that gave me the worst score to what I considered my best results, and vice-versa.

Some of the compute times were also very long, in the order of 24 hours, so it was hard to keep developing models without access to a computer or at the very least, the results of the previous model.

5.2.4 Expectations

There are no doubts in my mind that the Random Forest is a great model to use for classifying the phones with this type of data. On the other hand, I was expecting the Neural Network to perform better than it did. Then again, a NN can be tricky to configure and the possibilities are endless, so it's possible that I failed to find the true optimal configuration for it.

As for the clustering, I am disappointed that it did not provide the results we were looking for (to have the phones grouped in clusters in an exclusive fashion). At least the question has been answered: it's not possible to do it in an unsupervised way.

5.3 Improvements

The first improvement that could be made is obviously to the homebrew metric. A way to penalize empty clusters must be devised.

If we had access to the raw vectors, we might be able to add new features which could prove useful.

Finally, the Random Forest classifier has a "weight" parameter. I did not change it during the project, but it might be important considering that my phone distribution is far from being uniform.