University of St Andrews

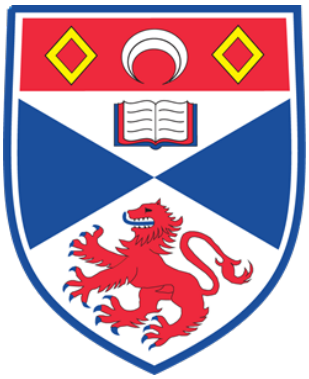
Machine Learning

CS5014

Classification

Author:
150008022

April 17, 2019



Goal

The goal of this practical is to analyse a dataset in order to produce a classification model that can make predictions based on a set of inputs.

Contents

1 Loading Data **1**

2 Cleaning Data **1**

3 Data Visualisation and Analysis **1**

4 Feature Selection **4**

5 Model Selection and Training **5**
- 5.1 Random Forest Classifier 9
- 5.2 Stochastic Gradient Descent Classifier 10

6 Evaluation and Comparison **11**
- 6.1 Binary 12
- 6.2 Multiclass 13

7 Discussion **13**

1 Loading Data

To load the data, the paths to the relevant files are supplied as arguments to the *__main__.py* script. The *pandas* module was used to load the file contents into *DataFrames*.

A test set was isolated from the original data using an 80%-20% split. Stratification was used to ensure that all classes were represented in the training data. Since the dataset was originally grouped by output class, the order of the samples were shuffled. This would avoid the later model being trained on several similiar instances in a row, which can have an affect on some algorithms performance.

2 Cleaning Data

When originally loading the CSV files the parameter to raise an exception on missing or extra columns was included, and so it could be assumed that all rows had the same number of columns. The dtype=**float** argument was also passed when loading the data to ensure that each column contained the expected numerical data. Any rows containing empty or NaN values were dropped from the dataset.

3 Data Visualisation and Analysis

The input CSV was understood to have the structure shown in figure 1. Each value is either the mean, minimum, or maximum reading from 100 radar pulses for a single component of a channel. Each channel is comprised of 256 components.

The mean, min, and max values were plotted for each channel for each sensor. The plots of the means of each channel for the book and plastic case objects are shown in figures 2 and 3 respectively. The difference between the resulting signals from the two objects are very clear.

In the binary dataset, the minimum and maximum components observed all followed a similiar shape as the average, but the book class did contain one severe outlier in two plots. The full plots are included in the submission under `plots/binarybook.png` and `plots/binaryplasticcase`, in which the plot of the minimum components in channel one and the maximum components

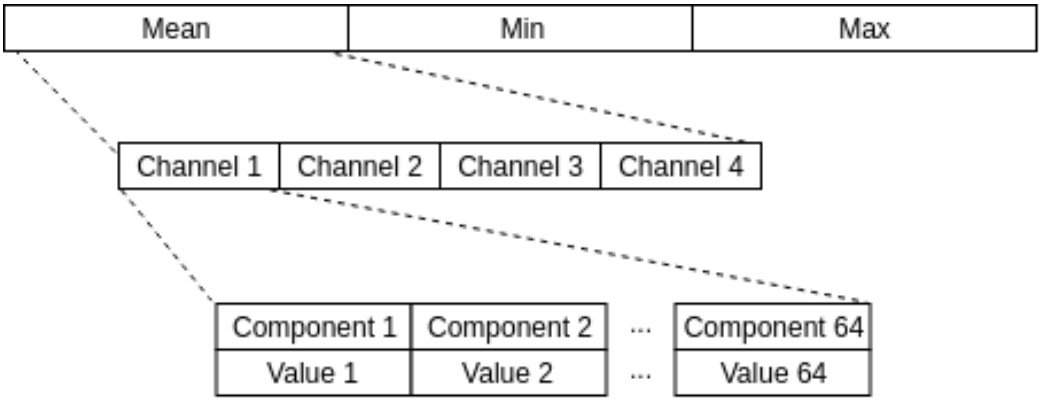


Figure 1: The structure of each row of the CSV file which is repeated for minimum and maximum values.

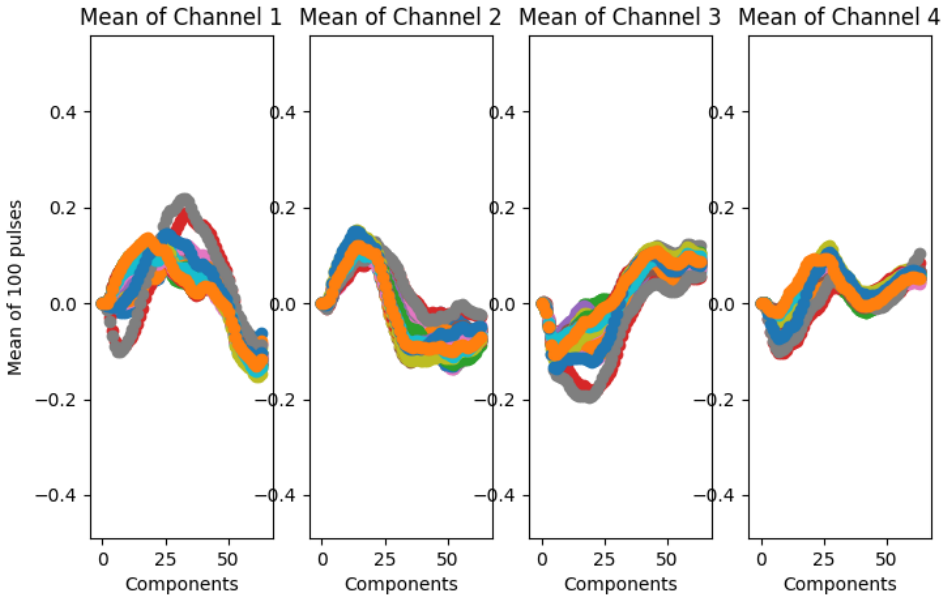


Figure 2: Mean of each channel measured for the book

in channel three both include one row of outliers. Since the average did not deviate from other components for that class, it seemed fair to say that these maximum and minimum readings were outliers. Instead of removing them and risking producing a biased model, the row was left in the data set. A

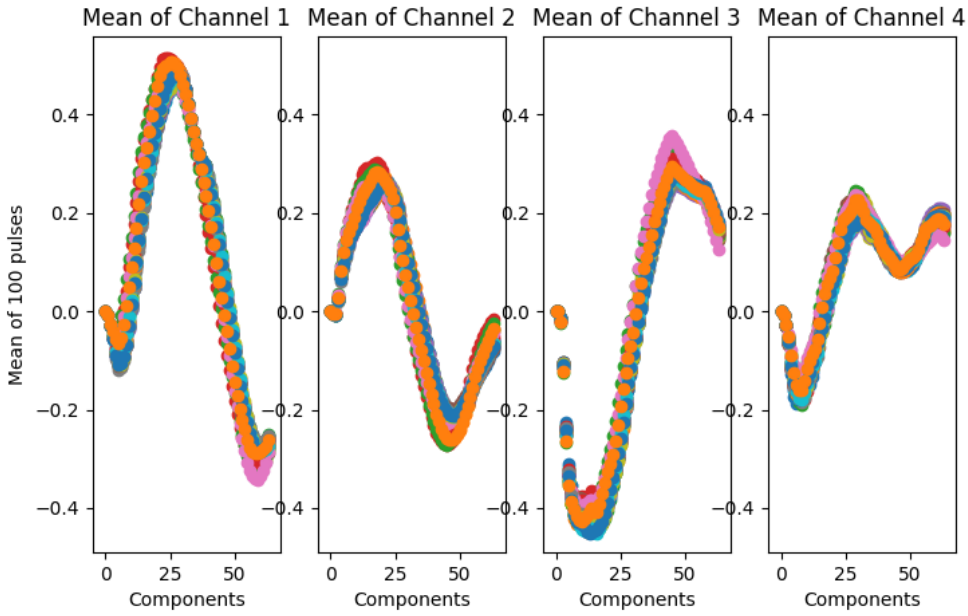


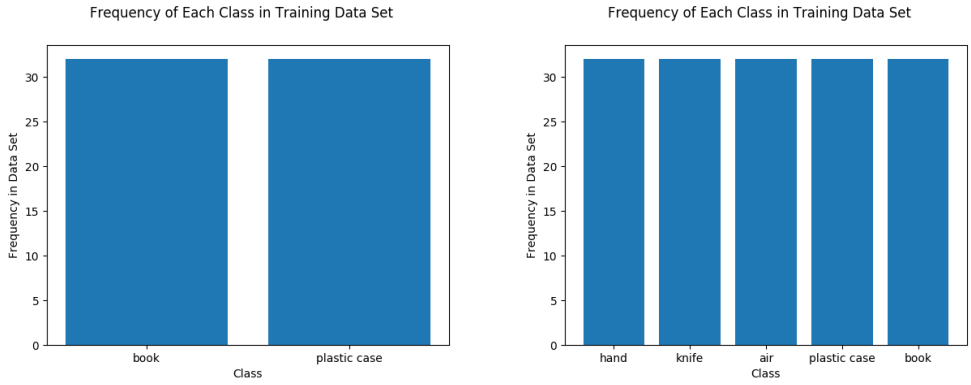
Figure 3: Mean of each channel measured for the plastic case

real world application of the sensor would likely involve noise, and so it made sense to train the model to be able to cope with anomalies. The existence of these outliers was however noted when choosing a cost function however in order to try and minimise their affect.

The same plots were made for the multiclass dataset, and from this it was clear that each material produced very different results, with varying levels of consistency. For example, the data aquired when the radar sensor was applied to a human hand varied wildly, whilst the readings for the plastic cover were a very clear sinusoidal shape. These plots can be found in `plots/multiclass*`.

Since the radar signature was determined by the reflection of the radar pulses on the surface and interior structure of each object, the resulting plots were understandable. For example, the plastic case shows a very consistent pattern likely due to the fact that it is composed of a single material in a uniform structure, whilst the human hand produces a very chaotic signature since it is composed of many different materials, especially fluids in motion.

The training data set was shown not be skewed by plotting the distribution of each class (figure4). The equal distribution of each class meant that



(a) Binary (b) Multiclass

Figure 4: Frequencies of each class in the binary and multiclass training data sets

cross validation of a classifier that always guesses the same class will have a ratio of correct predictions inversely proportional to number of classes in the training data set.

All data from the feature set had values between -1 and 1, and from the plots of all classes it was noticed that the different classes had different global minumum and maximums for each channel. For example, the components of samples for air did not surpass 0.1, 0.25 for books, 0.5 for plastic case, and 0.75 to 1 for hand and knife. Based on this and the fact that there were no outliers and the values were evenly distributed, normalisation was used over standardisation. Plotting the normalised data and comparing it to the plots of the original data reinforced this decision, as the shape and scale of the resulting plots had been maintained.

4 Feature Selection

The large number of features available in the dataset makes computation of any model more expensive. In order to have an effective classifier, there should be at least five examples of each combination of values from each feature in the training data [1], which our dataset is unable to provide. Therefore a reduction of the feature set was considered necessary.

The visualisations of the training data showed that each class had very

different levels of variation between each sample, and this variation could be used to identify a class. However our model would need to identify the class based on a single sample, and so this variation could not be relied on.

It was observed that the plots of each channel for the mean, minumum, and average were very similiar for each sample within a class. For this reason, all components were dropped except those based on the average of 100 pulses.

Principal component analysis (PCA) was then used to further reduce the number of features. PCA works by projecting the data onto the hyperplane that retains the most of the original variance in the dataset. By producing vectors that map and combine the original features to another set, PCA concentrates as much information into the first principle component, and then as much of whatever remains into the second component, and so on. Since the mapping combines features, the resulting components do not have any real meaning, though the original data can be recovered from the resulting principal components.

Initially, to try and understand PCA a plot of the first two principals components was produced (Figure 5. The first two principal components alone accounted for 82% of the original variance.

To show how variance changes as the number of principal components increases, the explained variance ratios were also plotted against the number of dimensions in figure 6. The return on increasing dimensions becomes miniscule after 20 dimensions. For the actual model, the PCA was configured to select the number of components that would retain 95% of the original variance. This reduced the feature set from 256 to 6.

The PCA used method requires loading the entire dataset into memory. Alternative methods exists such as incremental PCA and random PCA that can be used for online and batched learning.

5 Model Selection and Training

Multiclass classifcation can be implemented using binary classifiers through two methods, one-vs-one and one-vs-all. One-vs-all involves training an individual classifier per class, and each classifier is only able to state if the input represents its class or not. Since the training data used has an equal distribution of each class, each classifier would be trained on skewed data with far more negatives than positives as demonstrated in figure 8. This method would require as many classifiers as there are classes [2].

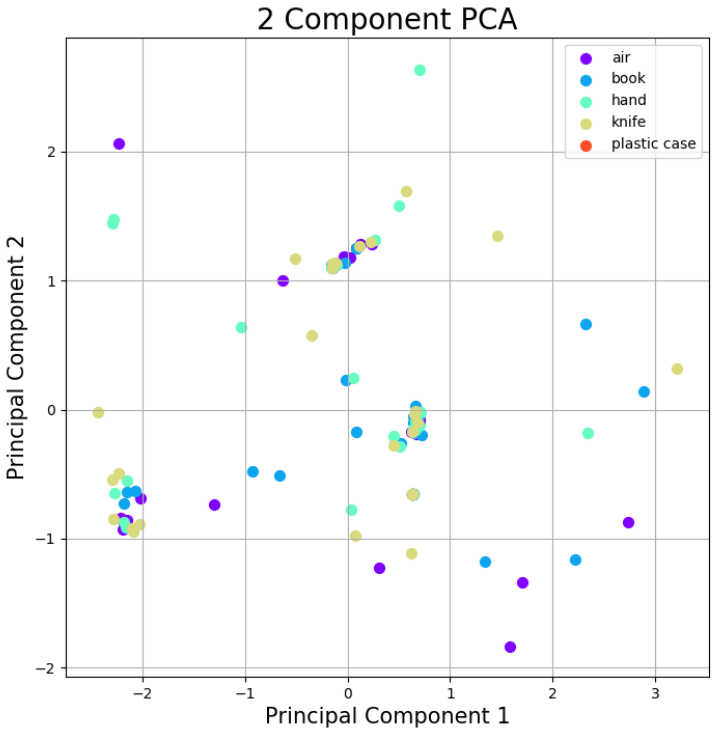


Figure 5: The first two principal components, coloured according to their class.

A one-vs-one classifier requires a classifier exists for each pair of classes. Each classifier will predict a value from the two classes it was trained to identify, and the resulting prediction will be the class that was chosen by the most classifiers. This method requires $n(n-1)^2$ classifiers, where n is the number of classes.

There also exist dedicated multiclass classifiers with different advantages and implementations.

Since the RadarCat [3] technology is intended to be able to allow users to identify and catalogue various every day objects, both the one-vs-one and one-vs-all approaches would involve a large number of classifiers being trained. Therefore an approach that required fewer classifiers would probably be preferred.

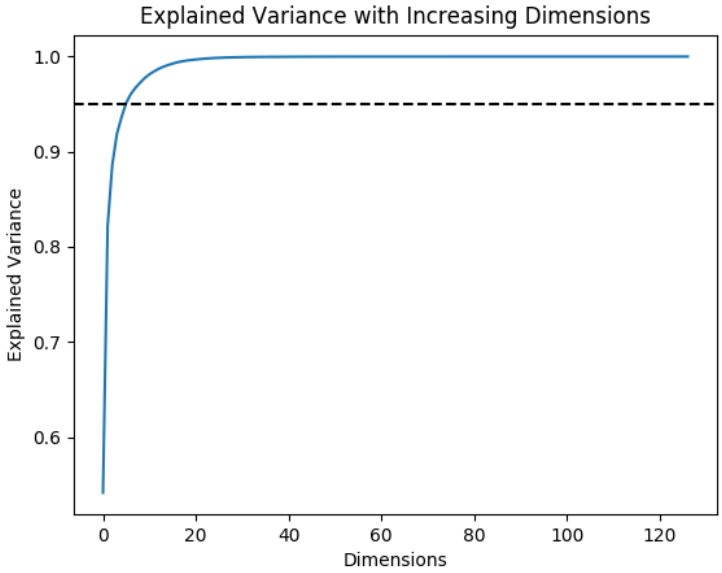


Figure 6: The elbow curve of explained variance that results from increasing the number of dimensions. The dashed line is drawn at our models chosen variance.

On the other hand, the Soli [4] technology used by RadarCat was designed with the intention of recognising hand gestures, and so being able to generalise well when trained on fewer classes would likely be favoured in that case (i.e. handling many variations of the same hand gesture).

Another possible solution that could reduce the number of classifiers needed as the number of classes grow is to create a hierarchy of classifiers. Each node has an "is-a" relationship with its parent in the tree, and prediction involves starting with a very generic classifier, and gradually getting more specific. [5] for example used such classifier to identify the musical genre of an audio clip using a heirachical structure. Though it performed similiarly to a flat classifier approach, it would be easier to include new classes. This would be useful for the RadarCat use case as the number of objects it is used to identify grows.

The ability to provide online learning (or at least batch learning) would also be useful, as user feedback could provide a method for crowdsourcing samples for further supervised learning. Crowdsourced data collection for

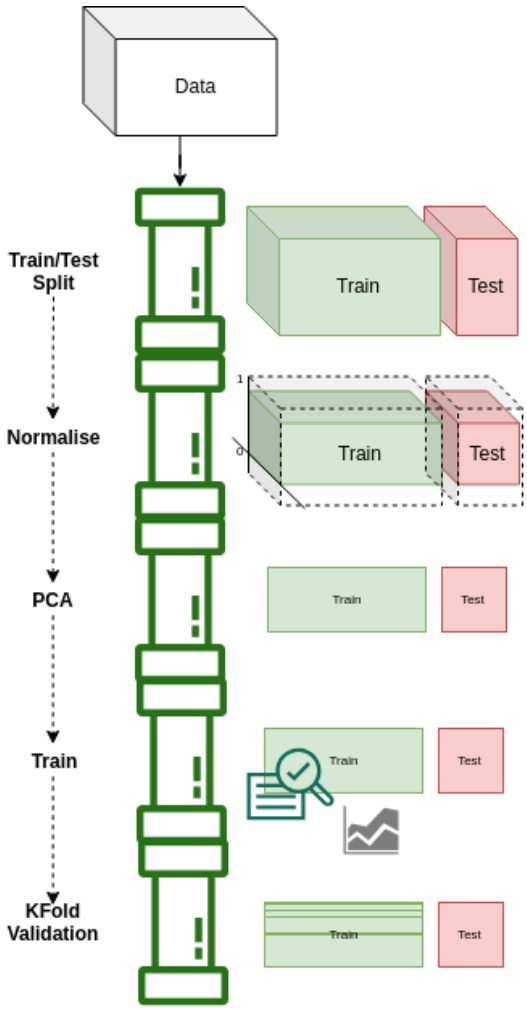


Figure 7: The pipeline used by the submission.

producing data for supervised learning has famously been applied by projects such as reCAPTCHA [6], and a similiar method could be used by RadarCat to improve its error rate.

With the previous information in mind (and for the sake of evaluation and comparison), an inherently multiclass method (Random Forest) and a one-vs-one method (Stochastic Gradient Descent) were chosen for the first and second models respectively.

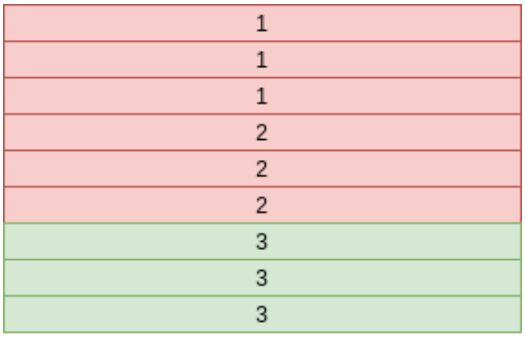


Figure 8: Distribution of negative and positive samples for a classifier trained to identify either class '3' or 'not 3' when each class is equally represented

5.1 Random Forest Classifier

The random forest classifier was chosen due to its robustness when dealing with noisy data, which would definitely be present in real world applications [7]. The number of classifiers would not grow with the number of classes, though training would be slower. Especially so in this case due to the high number of dimensions involved.

Random forests work by building a collection of decision trees which each have their own set of rules to narrow down the class and make a prediction. Instead of exclusively using a single decision tree for classifying, the random forest will take an average of all its decision trees to make its decision. Increasing the number of trees used can make predictions more stable, but will require more memory and computation.

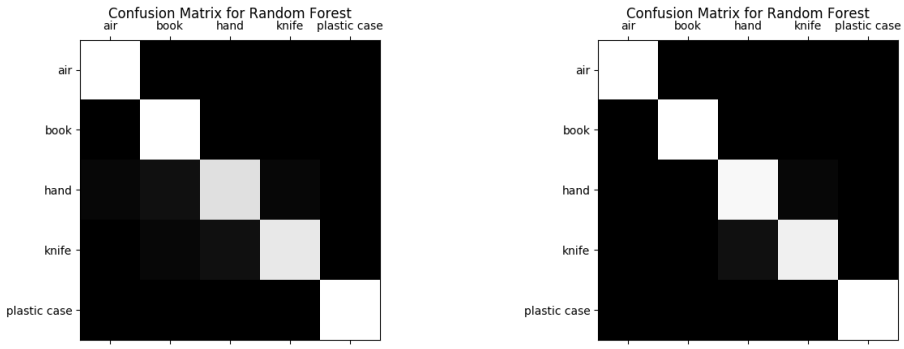
An rough starting point for finding the best hyperparameters for the random forest classifier was discovered by using a randomised search following the process described by [8]. Then the best hyperparameters were found by using a grid search. If grid search suggested that the best parameter was at the limit of a given range (for example, best max depth was ten, and the lowest depth in the search was ten) , the search was repeated with the range of values lowered/increased in case the best value was actually less than what the search yielded. This process was carried out for both the multiclass and binary datasets, as it was suspected that the multiclass random forest classifier would require a larger number of estimators (the number of trees). This turned out to be true, and the resulting hyperparameters that were different between binary and multiclass searches are shown in table 1. The

best parameters for the multiclass model were chosen for the end model, since only two models were to be submitted and using two sets of hyperparameters would effectively produce two different models.

Parameter	Binary	Multiclass
Max Depth	2	4
Number of Estimators	50	700

Table 1: Hyperparameters for random forest that were different when fitted on binary and multiclass data sets.

The benefits of the hyper parameter tuning can be seen in figure 9. Whilst the model still has some trouble telling the difference between the hand and knife signals, it no longer incorrectly identifies the hand signal as a book or as air.



(a) Before

(b) After

Figure 9: Confusion matrices produced by 10-fold cross validation before and after hyperparameter tuning.

5.2 Stochastic Gradient Descent Classifier

One-vs-one was implemented using the Stochastic Gradient Descent Classifier (SGDClassifier). It supports online and batch learning and it considers one point at a time rather than the entire dataset when learning. For the binary

dataset, one-vs-one would simply reduce to a single classifier, and for the multiclass model it would require a classifier for each pair of classes.

SGD generally works by selecting a random sample at each step, and computing the gradient using that sample in order to minimize a cost function. Due to this randomness, SGD does not suffer from the local minimum problem to the same degree as batch gradient descent. The learning rate determines how quickly the step size decreases, and can determine how close to the minimum the algorithm can get. [9] describe SGD as having two phases: a drift phase and a diffusion phase. The former is where the algorithm explores a large portion of the solution space, then as it converges upon a minimum it enters the far more chaotic diffusion phase.

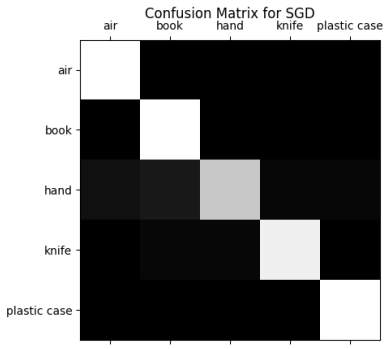
The SGDs learning rate is one of the available parameters that can determine its performance. The best value was determined using the same random search then grid search that was used for the previous model. The hinge loss function was chosen over logistic loss due to being less sensitive to outliers. The SGDClassifier also includes a penalty parameter which penalises higher degree polynomials which generally are a sign of overfitting in the model. This way simpler models are favoured over complex models, and complexity is only introduced if it reduces the error sufficiently. This process is called regularisation.

The two regularisers considered were ridge (L2) and lasso (L1) regression. L1 can reduce the contribution of less important features to zero in the model, which can produce sparse models, whilst L2 reduces all coefficients by the same factor which will not eliminate any features. L2 was chosen by comparing the performance of the model using both.

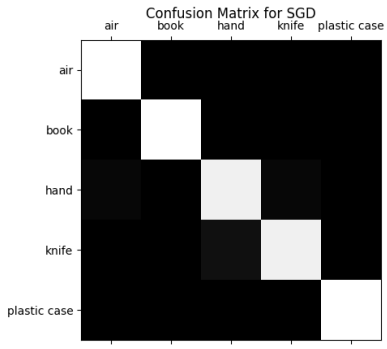
The model suffered from the same confusion between hand and knife samples, but would also missclassify hand as air unlike the random forest classifier.

6 Evaluation and Comparison

Finally, with both models trained, tuned, and evaluated, they were then applied to the test data set.



(a) Before

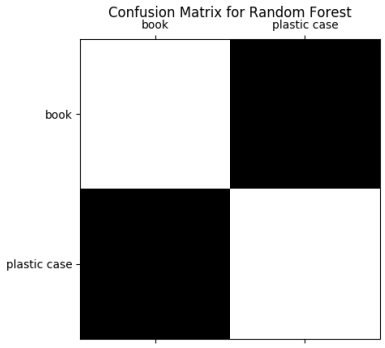


(b) After

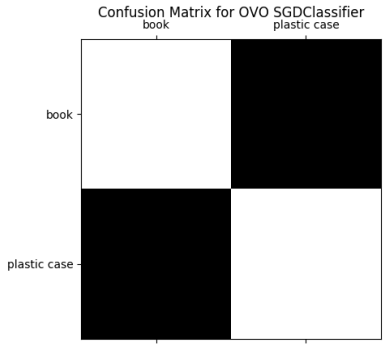
Figure 10: Confusion matrices produced by 10-fold cross validation before and after hyperparameter tuning.

6.1 Binary

Figure 11 shows the resulting confusion matrix produced by the model when applied to the binary test data. It appears to be very effective in predicting the difference between book and plastic case.



(a) Random Forest

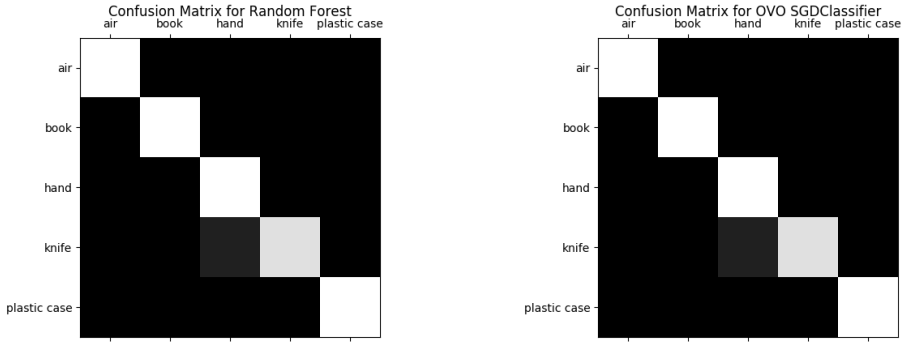


(b) SGD

Figure 11: Confusion matrices produced by the models applied to the binary test data.

6.2 Multiclass

Figure 12 shows the resulting confusion matrix produced by the model when applied to the multiclass test data. The test data shows better performance for both models when compared to the earlier k-fold cross validation evalations on the training data. This suggests that the models have not been overfitted.



(a) Random Forest

(b) OvO SGD

Figure 12: Confusion matrices produced by the models applied to the binary test data.

7 Discussion

Both seem to perform equally well, and both seem to missclassify knife as hand for some inputs. No synthetic features were produced during feature extraction mostly due to a lack of knowledge of exactly how the data was acquired and what it represented, but it is likely that finding other features that characterise the signals in the samples could reduce this missclassification.

Due to the similiar performance of the two models, it was difficult to choose one to classify the unlabeled test data given. However, the SGD classifier performed less well during k-fold validation and so the random forest classifier was chosen.

A lot of the time spent on this submission was focused on understanding PCA and the data that was being worked with. Given more time, I would

have like to produce more synthetic features. The signals were visually different as shown in the visualisation section, and so it is likely that there is a way of mathemtically defining this difference in such a way that can be used as a feature. PCA was used in the hopes that it would be able to identify which components were the most useful for identifying each signal, but due to the abstract nature of the features it produces it is difficult to understand if this was achieved.

References

[1] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, Inc., Orlando, FL, USA, 4th edition, 2008.

[2] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow*. O'Reilly Media, 2018.

[3] Hui-Shyong Yeo, Gergely Flamich, Patrick Schrempf, David Harris-Birtill, and Aaron Quigley. Radarcat: Radar categorization for input & interaction. pages 833–841, 10 2016.

[4] Jaime Lien, Nicholas Gillian, M Emre Karagozler, Patrick Amihood, Carsten Schwesig, Erik Olson, Hakim Raja, and Ivan Poupyrev. Soli: Ubiquitous gesture sensing with millimeter wave radar. *ACM Transactions on Graphics*, 35:1–19, 07 2016.

[5] Juan José Burred. A hierarchical music genre classifier based on user-defined taxonomies. 01 2005.

[6] Google. Introducing recaptcha v3: the new way to stop bots. 10 2018.

[7] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001.

[8] Will Koehrsen. Hyperparameter tuning the random forest in python. 01 2018.

[9] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. 03 2017.