1 - Deliverables achieved

- Within the provided python scripts there are:
  - The training and calibration of four classification models:
    - Logistic Regression;
    - K-Nearest Neighbours;
    - Single layer neural network.
    - Gradient Boosted Trees;
  - Aggregation of above models into an ensemble voting classifier that achieves 100% accuracy on unseen test data.
- Analysis contained herein that justifies model selection.
- Appendices with predicted classes for unlabelled binary and multi-class data.

  Reader note: There is no separate extension section. In my solution, I independently researched, explained and then applied to following topics not yet covered in CS5014: dimensionality reduction with PCA; gradient boosted trees; and ensemble voting classifiers. I hope these are considered enough of an extension compared to a basic solution.

To run, use command ‘python3 Analysis.py’ (program takes circa 20 minutes to completely run through).

- To print correlation heat maps, the Seaborn plotting library is required.
- The labs machines are running an old version of sklearn (19.3). In my program, sklearn version 20.2 is used. If the reader uses 19.3, then all else equal, grid search with the MultiLayerPerceptron module generates models with lower accuracy.
- So the py script still works for version 19.3, but just the output differs slightly (but still runs on lab machines). This is discussed further in Section 6 and and Appendix C.

2 – Initial Problem Discussion

<u>Objectives</u>
The objective of analysing the data is to be able to use various radar patterns to quickly and accurately identify which objects are placed in front of the radar device.

It is worth comparing this task to P1 (i.e. regression using residential energy data). My interpretation of P1’s objective was that we sought to better understand how a particular house design impacts heating and cooling loads, so it could ultimately help us make better, real world decisions for building designs. Understanding the drivers of the model was almost as important as the model’s accuracy (at least that was my opinion), and therefore I chose to design an easily interpretable linear model (i.e. not a black-box).

Here, the ultimate desired outcome is not a better understanding of radar’s reflection and refraction properties to further physics, but rather it is the fast, accurate classification of objects. As such, when selecting a model, I will not restrict myself to choosing interpretable models only, and will focus on accuracy and speed to classify.

Initial Thoughts

Before beginning any real analysis of the dataset, we can make the following observations just from the problem description:

- **We should expect a lot of overlapping information in the independent variables.** There are four channels each outputting 64 radar signals within a small wavelength band. Each signal has three features (mean, min, max) which will be somewhat related to each other. This suggests there could be a lot of correlation between variables.

- **There is a limited amount of data given the high-dimensionality of features.** With 80 observations for the binary set, and 200 for the 5-category multi-set set, this equates to only 40 observations per class for both sets. This reduces further after taking into account splits into training, validation and test sets. This is much smaller than the 768 features available, so we will need to take extra care to avoid over-fitting.

- **The binary classifier will likely be more accurate than the multi-class classifier.** Intuitively, for a fixed amount of data (40 observations per class), I expect it is harder for a model to separate five different classes than it is to separate two. For this reason, I expect the multi-class classifier to perform worse.

- **Ensemble learning could be helpful.** This is a classification task and an ensemble classifier, i.e. the aggregated estimate of a collection of models, could be a good approach because:
  - With such sparse data, it might be difficult to develop one single, fine-tuned, 'killer' model without over-fitting.
  - In contrast, if we have several 'acceptable' models that generalise well (say accuracy +80%), and have different strengths and weaknesses, then the combined output could be greater than the sum of their individual parts. This is achieved through diversification of errors.
  - It is also very difficult to ignore that ensemble approaches have very strong track records when it comes to classification problems and often win most competitions (such as those on Kaggle). This is discussed further in Section 5.

3. Loading the data

For the majority of my data handling, I chose the Pandas module (as opposed to NumPy arrays) because I prefer the in-built functions (such as *head(), info()*).

The data was loaded into a Dataframe object using *'pd.read_csv'*. The *.head()* and *.info* methods were used to check that the data:

1. Had been loaded correctly (i.e. all the variables captured with correct names, and correct number of observations).
2. Is entirely made up of integer or float values (for dependent and independent variables respectively).
3. Contained no missing values (signified by the 'non-null' count not being raised).

These are positive developments because it has confirmed we don't need to undertake any major data cleaning. Point 3 is particularly good news because we don't need to consider dropping rows or implementing imputation strategies for missing variables.

Now, before continuing further, we will randomly split the entire data set into three, non-overlapping partitions of training, validation and testing sets with a ratio of 80:10:10 respectively. Stratified sampling on the dependent variable category was used to ensure we have coverage across all categories (i.e. avoiding lop-sided datasets).
Going forward, each of these will be used in the following way:

- **Training:** This will be used to train the model parameters.
- **Validation:** This will be used as a proxy for the test set, and to validate if the overall model chosen is giving sensible results. It can also be use to fine-tune any hyper-parameters used.
- **Testing:** This will be used to test how well our model generalises. To avoid data-leakage, this will not be used until the end of the evaluation process.

With a finite dataset of 80 and 200 rows, we have competing needs of desiring more data in our training/validate process to generate the best parameter estimates, but we also want a test set large enough to be comfortable that the model generalises well. In my opinion, the ratio of 80:10:10 strikes a good balance between these requirements.

4. Visualising/analysing data, creating new input features from the given dataset

As noted in Section 2, we expected there to be a lot of overlapping information between the 768 features. Plotting the correlation heat-maps confirms this (Figures 1 and 2).

Binary Data Heat-map

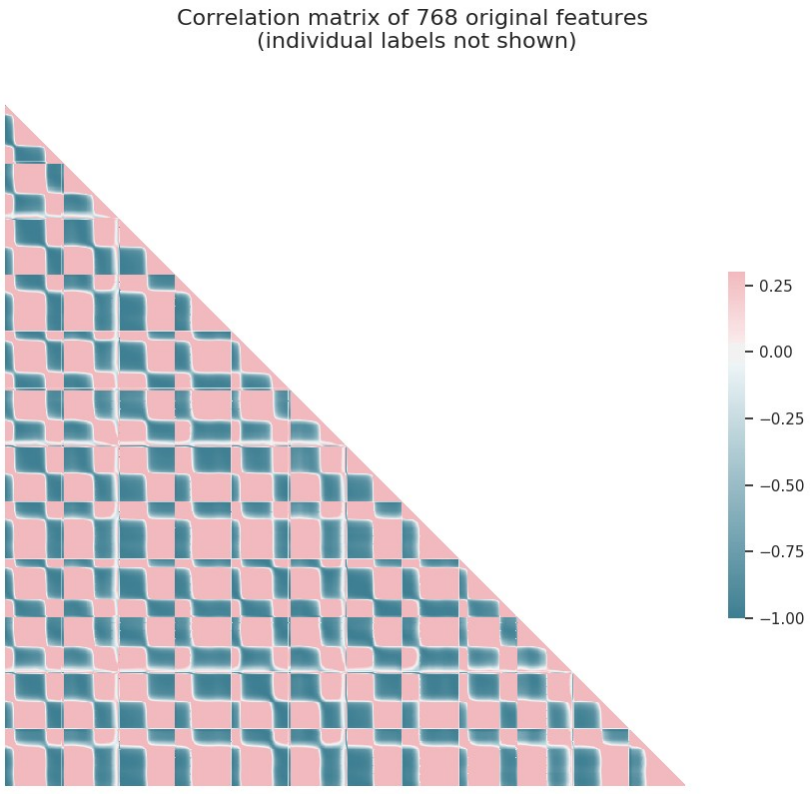

Figure 1: Correlation heat-map of original binary set input features. Labels not shown to improve readability.

Multi-class Data Heat-map

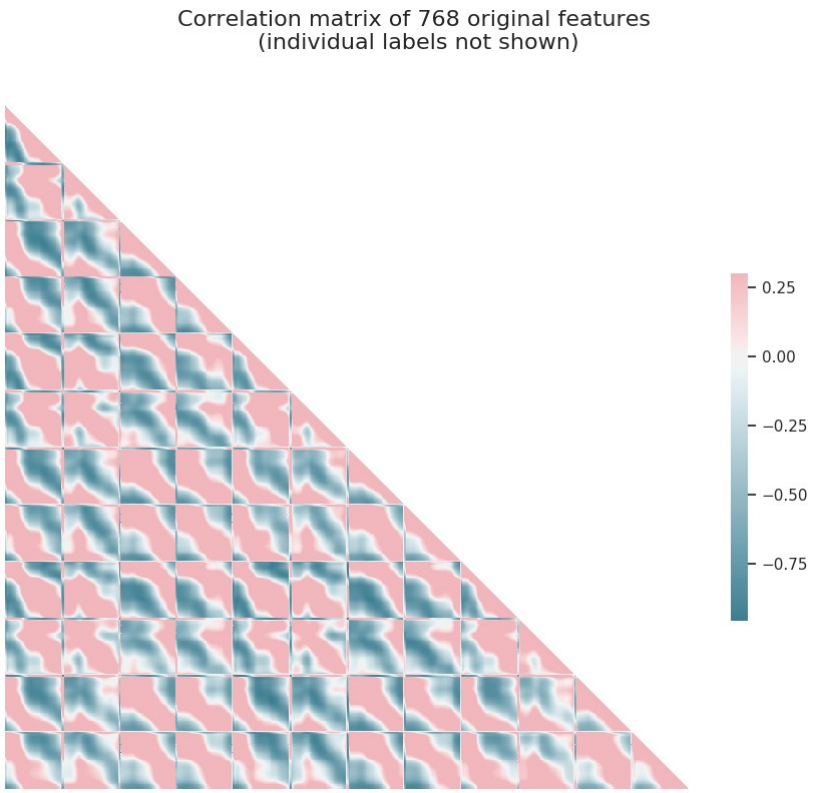

Figure 2: Correlation heat-map of original multi-class set input features. Labels not shown to improve readability.

One approach to reduce the high-dimensionality and remove the repetition of overlapping information in the data could be to manually remove and/or engineer features. For instance I could:

- Analyse (by plotting more charts) which channel explained the most variation in Y, and afterwards removed the less important channels; and/or
- Create new features such as: '*signal width i = (max signal i) - (min signal i)'* and then remove the two input variables (so that two existing variables are reduced to a single new variable).

Both of the above approaches might be successful in reducing dimensionality without losing too much of the original feature variation.

Given the large number of variables though, this could be a tedious approach. Instead, Principle Component Analysis ('PCA') was performed to project the original 768 variables into a much lower dimensional space. Using a threshold of 99.9% of the original X's variation, sklearn's PCA module extracted 32 principle components for the binary dataset and 53 for the multi-class set.

In other words, my new features represent almost all (99.9%) of the original X dataset with the benefit of being significantly smaller in dimension (95% smaller for binary and 93% smaller for multi-class). Further benefits are:

- Lower likelihood of over-fitting (by virtue of their being much fewer variables to add);
- No longer need to perform manually feature selection as I know my variables contain 99.9% of X's variation;
- Lower computational requirements during training (again because of lower dimensionality); and
- By design, the PCA features will not have correlation.

Plotting a correlation heat-map for the binary PCA components demonstrates the final bullet point above (i.e. no correlation). A similar result is seen for the multi-class dataset (not shown).

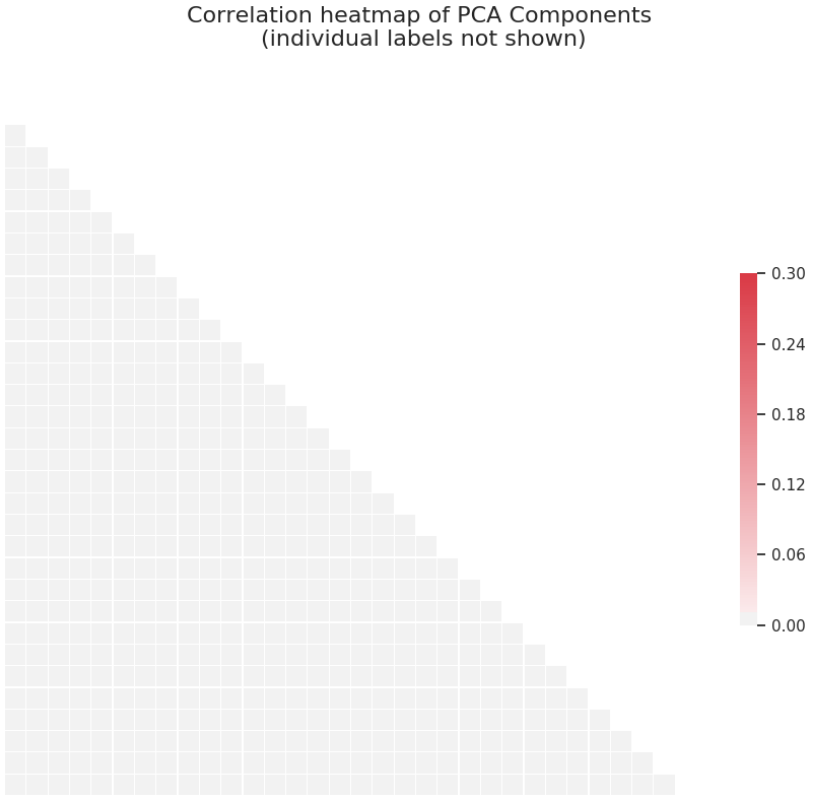

Figure 3: Correlation heatmap of binary set PCA features

Lastly it is worth noting:

- When compared to the original feature set, the PCA features are quite abstract. As per Section 2 though, my interpretation of the task does not place importance on explainable model features, so I am comfortable with this choice.
- Before performing PCA, the data was standardised to remove the effects of different scales.

5. Preparing the inputs and choosing a suitable subset of features

Preparation of independent variables

As per Section 4, we have already scaled the data and then distilled the features into a smaller set of principle components. No further data preparation has been performed.

If we find the performance of our initial models does not hold promise, then some later steps we could take are:

- Consider basis expansions (such as polynomial) on the PCA components; and/or
- Increasing the explained variable ratio in our PCA step to increase the number of model features.
- Scrap the PCA transformation and consider some of the manual feature engineering approaches discussed in Section 4.

For now however, the actions taken thus far seem sufficient to get a basic logistic regression model running to check if we are on the right path.

Preparation of dependent variables

No processing was performed on any of the categorical dependent variables. This is because:

- The binary dataset is already in one-hot-encoding form (so no further processing was required).
- While the multi-class dataset was encoded in categorical integers (i.e. from 0 to 4, each representing a class), the sklearn documentation says that classifiers "*do multiclass classification out-of-the-box*", and so nothing further is required.
- If we were not using sklearn, then we would need to perform one-hot-encoding to ensure that categories are treated as equidistant apart when seeking the best model parameters.

6. Selecting and training a model

General approach for both binary and multi-class datasets

As per Section 2, I am interested in ensemble classifiers given:

- With the small amount of data per category suggests it could be difficult to train a single accurate model. The aggregated classifications of several different models with 'OK' accuracy might allow us overcome this by diversification of errors;
- Evidence in both academia (Kittler, Hatef, Duin, & Matas, 1998) and actual data-science competitions (Géron, 2017; Murray, 2018) suggest that ensemble methods can work well for classification problems;
- For my own personal development, they allow me to experiment with several new models I haven't used before.

Therefore my approach is as follows:

- **Step 1:** Train a regularised logistic regression model. Measure performance on training and validation sets to obtain an initial accuracy benchmark.
    a) Adjust regularisation parameter (L2) as appropriate if under or over-fitting occurs.

- **Step 2:** Train the following models we get satisfiable accuracy. Use grid-search with cross-validation to tune hyper-parameters.
    a) A gradient boosted tree ('GBT') classifier;
    b) K-Nearest Neighbour ('KNN') classifier (note: KNN doesn't require training, only hyper-parameter tuning);
    c) A feed-forward neural network classifier;

- **Step 3:** Combine all four models into a voting ensemble classifier and test performance on training and validation sets.

- **Step 4:** Combine training and validation data and retrain model parameters and predict against test set. This step will provide guidance if our ensemble approach is appropriate.

- **Step 5:** Use all data points (training, validation and test), retrain the model parameters and then use this to test the unlabelled 'XToClassifyData'. This last step is to ensure we make our final predictions with the most amount of data as possible.

Justification for model selection

Per guidance in Kittler et al, combinations of classifiers are *"particularly useful if they are different"* (Kittler et al., 1998). As such, each of the proposed models was chosen because it offers different qualities (strengths/weaknesses) to the ensemble voter:

Logistic regression:

- Provides linear decision boundaries and so for linearly separable data, this approach should work well.
- Takes the entire dataset into consideration when creating decision boundaries.
- Because *"logistic regression wants to create a large margin between boundary and (all of) the data"*, it can result in decision boundaries tilting in odd angles to accommodate outliers near the decision boundary (Minka, 2003) .

Gradient Boosted Trees:

- The GBT model is itself an ensemble learner that gradually adds and trains multiple decision-tree models where each incremental tree is trained on the residuals of the prior tree. It is somewhat similar to forward stage-wise liner regression (Hastie et al, Chapter 4) in the sense that the models next tree is determined by the residuals of the model's prior iteration.
- Tree methods such as GBT are robust to outliers (Hastie, Tibshirani, & Friedman, 2008).
- The combination of these trees will produce linear decision boundaries that are orthogonal to the PCA axes. This will be different to the decision boundaries of the other models.
- Lastly, it is difficult to ignore that GBT approaches tend to feature in many winning data-science entries such as Kaggle and the Netflix Prize (Koren, 2009).

K-Nearest Neighbour:

- KNN is a probabilistic classifier that formulates classifications based on the 'K' closest data points and nothing more (in this sense, the 'K' majority voting system shares similarities to an ensemble classifier).
- Per Hastie et al, KNN's produces piece-wise linear decision boundaries which can be used to represent "irregular" and non-linear class boundaries (Hastie et al., 2008).
- Hastie et al also details how KNN are successful in many real world applications including electrocardiography signals. From my naive perspective, pictures of electrocardiography waves don't look dissimilar to the waveforms in the RadarCat article. This leads me to believe KNN could be worth attempting.
- Russell and Norvig also note that *"in low-dimensional spaces with plenty of data, nearest neighbours works very well"*(Russell & Norvig, 2016). Due to the curse of dimensionality, KNN might have been infeasible in the original, larger feature set, but with the smaller PCA features, the data-to-feature ratio is much higher, and so there is a possibility it will perform well.
- Unfortunately, as the size of the data set gets very large, KNN will take longer and longer to classify (as it calculates the distance from every data point). This is discussed further in the model evaluation (Section 8).

Neural network:

- NN's can represent complex non-linear decision boundaries between classes and work well in instances where there is high-signal-to-noise ratios (Hastie et al., 2008).
- NN's are well suited to classification tasks, particularly when *"modelling real world complex relationships"* (G. P. Zhang, 2000) (which this problem is considered as).

Step 1: Initial Logistic Regression (ridge-regularisation)

Binary Dataset
The initial logistic regression model was trained on the default sklearn parameters (i.e. log-loss function with a ridge-regularisation parameter of 1.0). The default solver used was LIBLINEAR which sklearn recommends for small datasets.

Somewhat surprisingly, the trained model returned 100% accuracy on the training set. While I initially had concerns about over-fitting, these were alleviated when the validation set also produced the very strong results below.

Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

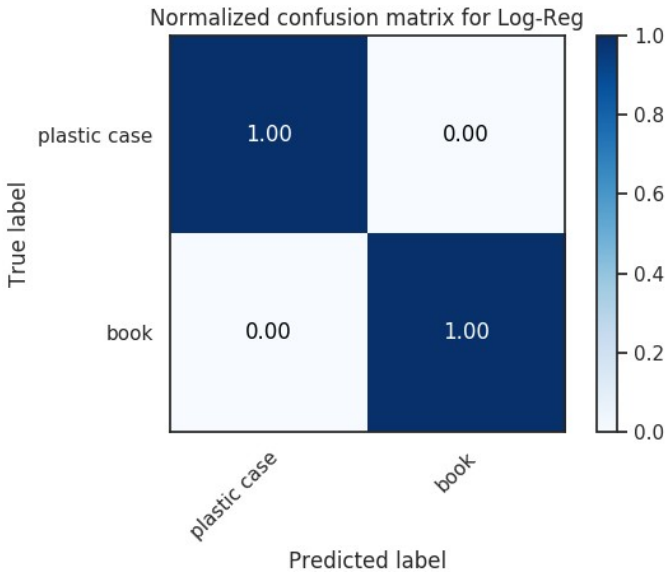

Figure 4: Confusion matrix for logistic regression on binary validation set

The results above demonstrate :

a) 100% of classifications are successful (i.e. accuracy);
b) 100% of the ‘plastic case’ classifications (i.e. binary value 1) are actually plastic cases (i.e. precision);
c) 100% of the ‘plastic case’ observations have been correctly classified as plastic cases (i.e. recall);
d) The balance between recall and precision is the best it can be (i.e. perfect F1 score).

Given that we had 100% accuracy, points (b)-(d) follow quite naturally (in our other models though, these figures will vary). The ‘macro’ versions of these scores were used because they compute metrics on a category-by-category average basis. This is considered appropriate because we desire a model that performs well for all categories (not just for the categories with the most observations). For binary classification, the choice of ‘macro’ is less relevant but it will become important when benchmarking our multi-class model.

Multi-class Dataset
When the logistic regression model training was completed for the multi-class set, very strong results were seen once again. For the training set, it obtained 100% accuracy. The validation set results are below:

Model: Log-Reg
Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

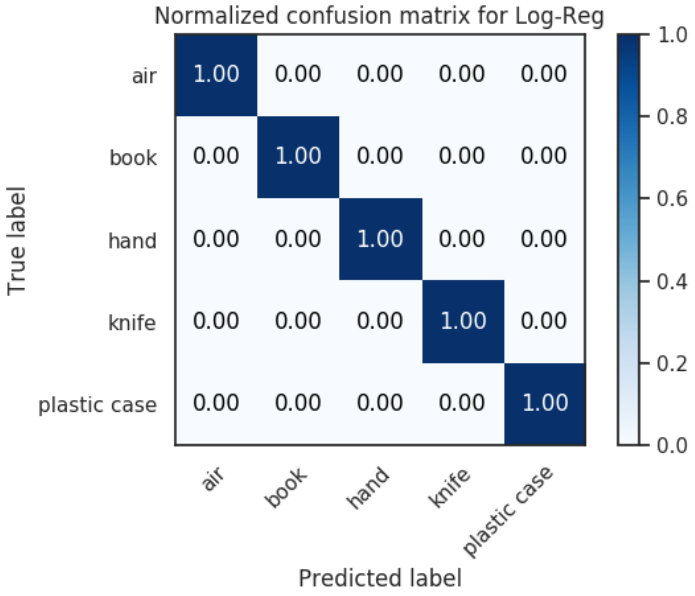

Figure 5: Confusion matrix for logistic regression on multi-class validation set

Conclusion: Logistic regression produced strong results, and so will be included in our ensemble voting classifier. This also suggest the current data set is most likely linearly separable.

Reader note: Going forward, for the other models with 100% accuracy, I will not provide confusion matrix diagrams given they offer little insight.

Step 2 (A): Gradient Boosted Trees

Binary Dataset
As previously discussed, GBT's work by incrementally building a collection of decision trees where each tree is trained on the residual of the prior model. Per Geron, GBT's have hyper-parameters to control the decision trees and also the ensemble of tree themselves (Géron, 2017). As a general comment, the more 'detailed' and 'intricate' the collection of trees are (e.g. more depth, more features, more estimators), the more at risk we are to over-fitting.

Sklearn's *'GradientBoostingClassifier'* module was used in combination with *'GridSearchCV'* to tune the following hyper-parameters:

- Max features – The maximum number of features used for deciding a split within an individual decision tree (if lower, less likely to over-fit).
- Max depth – The maximum depth of an individual decision-tree (if lower, less likely to over-fit).
- Learning rate - The contribution of each tree to the model (if lower, less likely to over-fit)
- Number of trees/estimators – The number of trees used in the final model (if too high, could potentially over-fit).

An initial grid search yielded another model with 100% training accuracy with the following hyper-parameters:

Best hyper-parameters: {'learning_rate': 0.1, 'max_depth': 2, 'max_features': 6, 'n_estimators': 10}

When this model was applied on the separate validation set, it also exhibited the perfect results below:

Model: GrBoost
Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

This suggests it will be useful in our ensemble classifier.

Multi-class Dataset
When GBT was applied to more categories, it was worse at distinguishing classes. It performed well on the training set 98.1% accuracy with the following parameters:

Best hyper-parameters: {'learning_rate': 0.1, 'max_depth': 2, 'max_features': 6, 'n_estimators': 40}

But when this same model was applied to the validation set, the accuracy was much lower (i.e. it was over-fitting).

Model: GrBoost
Accuracy: 0.778
Precision (macro): 0.900
Recall (macro): 0.810
F1 score (macro): 0.816

In the confusion matrix below (figure 6), we can see this low accuracy in effect. Particularly, we see the 'hand' classification is being applied over-zealously by our GBT model with classes affected more than others. For example, 50% of plastic case observations were mistaken as hands.

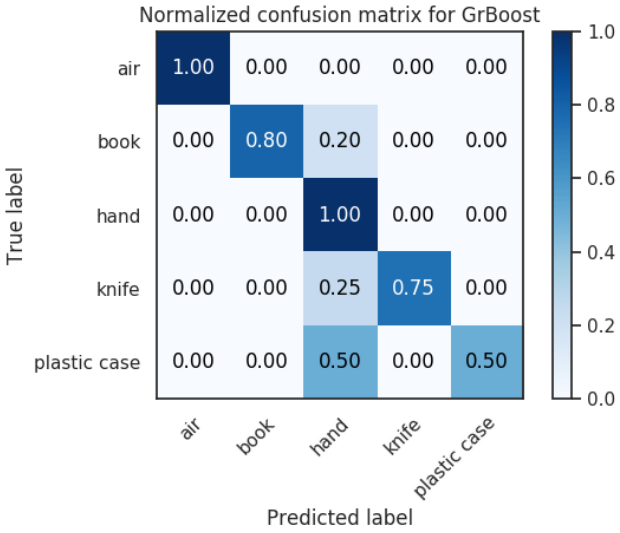

Figure 6: Confusion matrix for GBT on multi-class validation set.

Reader note: the 'macro' recall/precision/F1 scores won't perfectly align to the CM figures above because they are calculated differently.

Given the initial grid-search had quite coarse hyper-parameter bandings, I performed a second 'fine-tuning' grid search with the training set. This process sought to more subtly adjust the 'best' hyper-parameters that the initial grid search found in order to improve accuracy (revised model below).

Best hyper-parameters: {'learning_rate': 0.05, 'max_depth': 3, 'max_features': 5, 'n_estimators':38}

While this fine-tuning resulted in a similar training set accuracy (98.8%), the validation set demonstrated significantly improved performance:

Model: GrBoost - finetuned
Accuracy: 0.889
Precision (macro): 0.933
Recall (macro): 0.910
F1 score (macro): 0.909

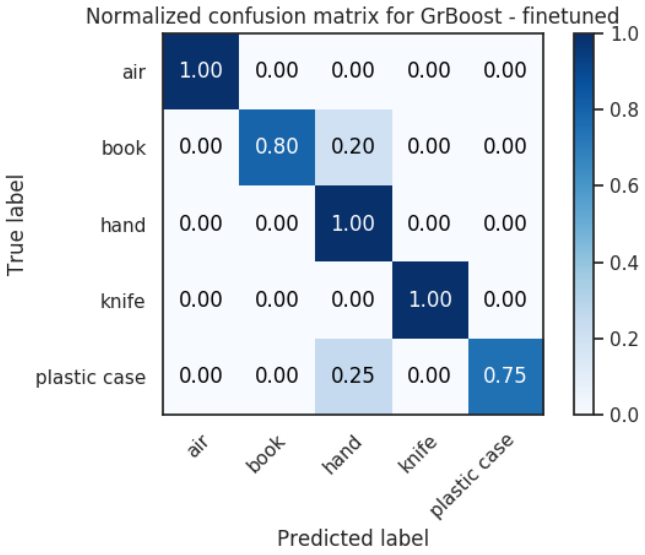

Figure 7: Confusion matrix for GBT on multi-class validation set (after fine-tuning)

Performance is still not as strong as our binary GBT, but 89% accuracy is still sufficient enough to be useful in our ensemble classifier. Compared to Figure 6, Figure 7 demonstrates a significant improvement in ‘hand’ classifications.

If I was to hypothesise why it performed worse than the binary model, it would be because ensemble methods with many estimators (such as GBT) rely on the law of large numbers to make accurate predictions on average. As such, I suspect there needs to be many more observations required for each class to help ‘average out’ the parameter values and obtain a generalisable model. When we apply a random process like grid search to another random process like GBT’s, we are splitting an already small data set into smaller pieces across 5 classes, and so perhaps this is not enough data for the impact of the law of large numbers to be useful.

Lastly, it would be reasonable to question if the 98% training accuracy versus 89% validation accuracy is indicative of over-fitting. This is a valid question and also something I considered. While 10% is a notable gap, the validation data set is only 18 values, so 89% accuracy corresponds to only two misclassifications – which is not considered material. If the validation set was much larger, and we continued to get much lower accuracy (relative to training), then it might be cause for concern.

Step 2 (B): K Nearest Neighbours

Binary Dataset
In order to determine the appropriate number of nearest neighbour distances to consider for classification 'votes' (i.e. the 'K' parameter), GridSearchCV was applied to KNN for values in between 1 and 10. The best K value was K=1 which yielded 100% accuracy on the training set. When this was applied to the validation, again we achieved the perfect performance below:

Model: K-Nearest
Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

This indicates it would be a suitable candidate for our ensemble classifier.

Multi-class Dataset
The same GridSearchCV approach above was also applied, and the best result again suggested a K=1 value. The training set accuracy was 93.4% and with the validation results below:

Model: K-Nearest
Accuracy: 0.944
Precision (macro): 0.900
Recall (macro): 0.960
F1 score (macro): 0.911

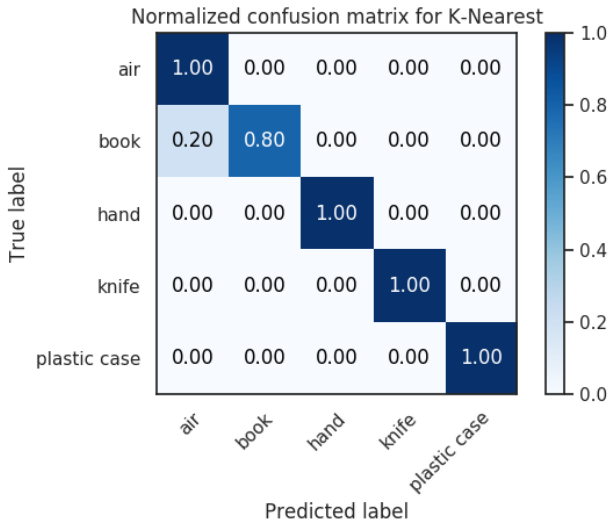

Figure 8: Confusion matrix for KNN on multi-class validation set

It is worth asking if we could fine-tune the K value further to improve upon this. Unfortunately, the cross-validation results are suggesting lower K values provide the best accuracy, and we cannot choose a K value lower than 1. Still, 94.4% is a good result, and so this model will be suitable in our ensemble classifier.

Lastly, please note the *diversity* in errors when comparing the Figure 8 to Figure 7 (GBT fine-tuned). KNN sometimes classifies 'books' as 'air', whereas GBT mistakenly classifies 'books' as 'hands'. These uncorrelated errors is what we desire when building an ensemble voter.

Step 2 (C): Feed-Forward Neural Network

Reader note: Per Section 1, different versions of sklearn output different final NN models (despite no changes with input parameters nor random seeds). The analysis from here on shows the model results for version 20.2 (see Appendix C for more information).

Binary Dataset
The key aspects that influence a neural network is described below:

- Network topology: the number of layers and the number of hidden units. Several different topologies were tested with grid search, including single layers with hidden units from 10 to 110 (increments of 20), and two-layers with each layer having hidden units from 10 to 110 (increments of 20). These ranges were chosen because they are in-between the number PCA features.

- Initial learning rate: this determines how much the gradients are adjusted after each iteration of gradient descent. While it is not strictly a hyper-parameter, I wanted to try several learning rates (0.01 to 1) so that I would increase my chances of find the global minimum. For instance, larger rates may be more likely to 'jump over' *local* minima, whereas the smaller rates are less likely to overshoot any *global* minima. By combining these, I hoped to find the best set of network weights.

- Regularisation parameter: to avoid over-fitting, L2 regularisation was applied for various values of alpha between 0.001 and 1.

- Early stopping: Similar to the above, sklearn's in-built validation-set/early-stopping features were also used to avoid over-fitting (I am particular concerned about over-fitting for NN's because there are many parameters/network weights and few observations).

The above values were tested within GridSearchCV. Other aspects worth noting are:

- Stochastic gradient descent ('SGD'), which trains using one observation at a time, was used (reason being I am less familiar with the other sklearn methods). Because SGD is used, the learning chosen will gradually reduce over time to ensure convergence.

- Sigmoid activation functions are used because I am more familiar with them.

After grid search, the following hyper-parameters produced training accuracy of 100%. (The below represents the best network with a single hidden layer of 10 units).

Best hyper-parameters: {'alpha': 0.001, 'hidden_layer_sizes': 10, 'learning_rate_init': 1}

When this model was applied to the validation set, it pleasingly produced the strong set of results below. This indicates there is no over-fitting and it would be a good candidate for our ensemble model

Model: NN
Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

Multi-class Dataset
The same approached described in the above binary section was applied to the multi-class data set. It generated reasonable accuracy of 93.2% with the following model hyper-parameters:

Best hyper-parameters: {'alpha': 0.001, 'hidden_layer_sizes': 90, 'learning_rate_init': 1}

When applied to the validation set, we saw the similar results below:

Model: NN
Accuracy: 0.944
Precision (macro): 0.960
Recall (macro): 0.960
F1 score (macro): 0.956

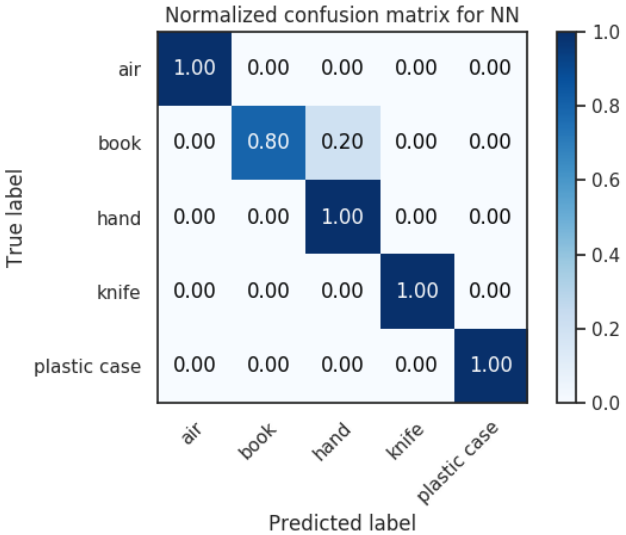

Figure 9: Confusion matrix for NN on multi-class validation set

To check if we can further improve the model, further hyper-parameter fine-tuning was applied on the training set. This improved training accuracy to 95.1% with the following hyper-parameters:

Best hyper-parameters: {'alpha': 0.0007, 'hidden_layer_sizes': 85, 'learning_rate_init': 0.9}

The fine-tuning also demonstrated improved results on the validation set (also, note there is no over fitting).

Model: NN - finetuned
Accuracy: 0.944
Precision (macro): 0.960
Recall (macro): 0.960
F1 score (macro): 0.956

(no change to above Confusion matrix)

Overall, these results are considered good enough to include in our ensemble classifier. Once again, when compared to our other confusion matrices, Figure 9 demonstrates the error diversity multiple models provides.

Step 3: Combine classifiers into voting ensemble classifier:

Using sklearn's *'VotingClassifier'* module, the above four models were combined using a majority or 'hard' voting system. This essentially gives each classifier its own vote in deciding the category.

Another alternative, called soft-voting, aggregates each classifiers estimated probabilities, and chooses the most likely class. This was not chosen because the KNN model with K=1 does not provide sensible probabilities (each KNN predicted class has an unrealistic 100% probability). This would have resulted in KNN's probabilities having overconfident, undue influence in the soft voting process.

Binary Dataset
When the voting classifier was applied to original training set, it unsurprisingly generated a training accuracy of 100%. Similarly for the validation set, we obtained the results below:

Model: Ensemble - Hard Voting
Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

These strong results are what we expected, because we know that each of the classifiers individually had 100% validation accuracy, so it makes sense that their combined effect would have 100% accuracy.

Multi-class Dataset
When we apply the voting classifier to the multi-class training data, we get training accuracy of 100%.

This begins to demonstrate the advantage of voting ensembles. Recall, only one of our classifiers had training accuracy of 100%.

Logistic regression:	*100%*
Gradient Boosted trees (fine-tuned):	*98.8%*
KNN:	*93.8%*
Neural network (fine-tuned):	*95.1%*

The aggregation of the above gives something greater than the sum of their parts. This is because:

1. Each classifier is more likely to be right than wrong (i.e. each has +50% chance of being correct);
2. The classifiers errors are not completely correlated (i.e. they are less likely to make the same mistakes, at the same time). The variation in the confusion matrices across each model is evidence of this.

When classifiers are wrong (i.e. bullet point 2), their incorrect 'votes' are unlikely to be concentrated together. This allows the correct classifiers to 'outvote' the incorrect classifiers more often than not.

When applying this model to the validation set, we get a strong, but imperfect result.

Model: Ensemble - Hard Voting
Accuracy: 0.944
Precision (macro): 0.960
Recall (macro): 0.960
F1 score (macro): 0.956

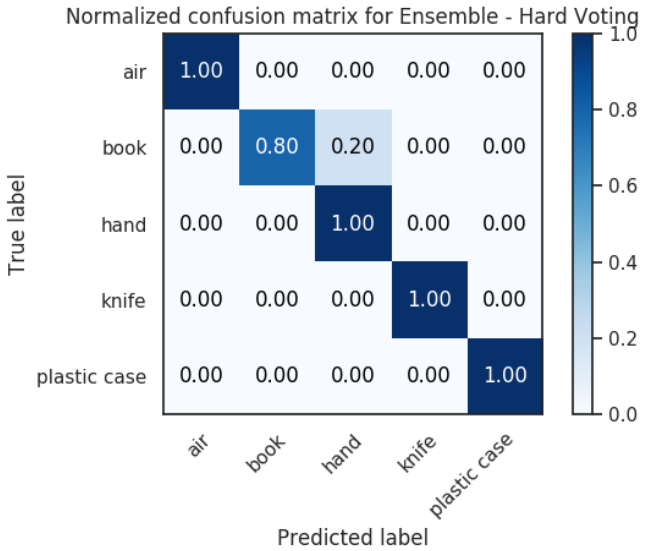

Figure 10: Confusion matrix for Ensemble on multi-class validation set

To better understand this, the voting results are shown below (incorrect classification highlighted red).

index	Y (correct values)	Log-Reg	K-Nearest	GrBoost - finetuned	NN - finetuned	Ensemble – Hard Voting
105	2	2	2	2	2	2
182	4	4	4	4	4	4
189	4	4	4	4	4	4
121	3	3	3	3	3	3
76	1	1	1	1	1	1
179	4	4	4	4	4	4
48	**1**	**1**	**0**	**2**	**2**	**2**
96	2	2	2	2	2	2
164	4	4	4	2	4	4
139	3	3	3	3	3	3
75	1	1	1	1	1	1
130	3	3	3	3	3	3
111	2	2	2	2	2	2
69	1	1	1	1	1	1
104	2	2	2	2	2	2
65	1	1	1	1	1	1
5	0	0	0	0	0	0
148	3	3	3	3	3	3

Figure 11: Validation set voting results for multi-class

With the exception of logistic regression, all of the classifiers found it difficult to correctly classify observation 48 despite their errors being uncorrelated (KNN produces a different error to GBT and NN).

Despite this, it could be that 48 is just a 'difficult' observation (perhaps very close to the decision boundaries), and when the models are retrained on more data (after combining the training and validation sets), the results may improve on the final test set.

The table below summarises the entire training process for each model.

Model	**Log-Reg**	**K-Nearest**	**GrBoost - finetuned**	**NN - finetuned**	**Ensemble – Hard Voting**
Binary training accuracy	1.00	1.00	1.00	1.00	1
Binary validation accuracy	1.00	1.00	1.00	1.00	1
Multi-class training accuracy	1.00	0.93	0.99	0.95	1
Multi-class validation accuracy	1.00	0.94	0.89	0.94	0.94

Reader note: The ensemble model exceeds or matches performance of all the individual models except for the multi-class logistic regression. This naturally raises the question, why bother with an ensemble of models if it doesn't demonstrate the best accuracy. This is discussed in Section 8.

7. Evaluating the performance of the model

Step 4: Combine training and validation data and retrain model parameters with chosen hyper-parameters.

In order to ensure we use every piece of data available to test our model, we combine the training and validation sets, and re-train our model parameters with the hyper-parameters identified in Step 2.

Binary Dataset
After retraining, 100% accuracy is achieve on the training+validation set (this is not surprising given the strong results in Step 3). When applied to the *unseen* test data, the following results are achieved.

Model: Ensemble - Hard Voting
Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

This is a very good result, and it is now reasonable to think that if tested against the unlabelled data in 'XtoClassify', that it would perform well.

Multi-class Dataset
Similarly, for the multi-class problem, after being retraining on the training+validation set, 100% accuracy is achieved. When this model is applied to the unseen test set, we obtain:

Model: Ensemble - Hard Voting
Accuracy: 1.000
Precision (macro): 1.000
Recall (macro): 1.000
F1 score (macro): 1.000

This is another strong performance, and it gives confidence that our model can be applied to the unlabelled data.

Step 5: Recombine training, validation and test sets and retrain model parameters with chosen hyper-parameters.

While the strong performance of the model in Step 4 could suggest that we are ready apply it to the unlabelled set, to be absolutely certain we will use the best model for our unseen, unlabelled data, we recombine all of the original labelled data, and retrain.

The predicted class results from the resultant model are contained in the Appendix A.

8. Critical discussion of the results and your approach

Result Evaluation

In Section 2, we outlined the desired criteria for our model. After training, a successful model must:

1. Classify items accurately; and
2. Classify items quickly.

For the first criteria, we have demonstrated in Section 7 that both models classifies data very accurately. In fact, neither models made any mistakes when classifying the unseen test set. On this criteria, the model looks to be an unqualified success (although I don't yet know the outcome of the unlabelled data).

For the second criteria, it is very important that we can generate predictions quickly, otherwise RadarCat users will grow frustrated of waiting for classifications, and potentially will stop using the device. Because we have implemented a combined four models instead of a single model, there is also a risk classification could take much longer.

Thankfully, this is not the case. When measuring how long it takes to classify an observation using the voting ensemble, the result below is obtained:

Average time taken for single observation (secs) 0.03544306755065918

Less than half a second is considered a very good result. For a point of comparison, it takes about 5-10 seconds to identify music on my phone with Google Assistant or Shazam (I appreciate that these are online services which need to contact servers etc, but I think the comparison holds merit). Therefore 0.35 seconds is acceptable.

A point worth making is that the KNN model will take longer and longer as the data set grows because KNN calculates the distance from every observation on record. If our data set was to grow into the millions of observations, then we would probably want to replace KNN with an model that scales better (e.g. a support vector machine). For the foreseeable future however, its performance is very good.

Both binary and multi-class models take circa 20 minutes to completely train which is mostly due to the GBT models taking a long time. To improve this, I tried using sklearn's 'n_jobs = -1' to apply parallelization in GBT calculations. While this would reduce training time to 10 minutes, it also would *sometimes* crash python on the lab machines ('thread syncing issue error'). Here, my script wasn't at fault but rather 'Python' on the Fedora OS would break (I experienced no issues at all on my personal laptop). Because it would fail about 20% of the time, I decided to remove GBT parallelization for the final submission. Despite this, 20 minutes to train a model is still considered acceptable given how well the model requirements above are met.

Overall Conclusion: Based on all of the above, it is my opinion the model performs very well.

Approach Evaluation

In my opinion, the approach to modelling was appropriate. This is demonstrated by:

- The choice of using an ensemble method is supported by both academic evidence (see references) and real-world examples (i.e. Netflix Prize, Kaggle competitions).

- The individual models within the ensemble classifier were specifically chosen because of their unique qualities which would diversify/reduce overall ensemble classification errors. As previously noted, this adheres to the guidance in academic literature.

- When training the models, design decisions were made to pre-emptively avoid over-fitting (e.g. both regularisation and cross-validation methods used).

- Data leakage was limited by not touching the test set until the very end of analysis.

- The original objectives were kept in mind at key decision points in the model such as using PCA to improve computational speed; and sacrificing model interpretability to improve accuracy.

- Fixed random seeds were used to ensure reproducibility of results.

Despite the above, it could be noted that my ensemble approach is over-engineering a problem that can be solved accurately using just one model – especially given each models' very strong individual performance.

For the given observations, and with a maximum of 5 categories this is reasonable criticism. The following points are worth noting however:

- If we chose logistic regression, our most accurate individual model, to be our sole classifier then it would perform very well for the 200 observations provided. If however we continued to use it and we observed new *outliers* that were located near the decision boundary, then for the reasons discussed in Section 6, logistic regression would not perform as well. It is in these cases where the votes from our outlier-resilient ensemble classifiers (GBT, KNN) help moderate this effect;

- Furthermore, if the task's scope was to be expanded to thousands of categories, then the decision boundaries required could become much more complex, and therefore difficult to represent in a single model.

Overall, while it did take much longer to learn about ensemble classifiers and implement four models instead of one, I am very satisfied my models performance in terms of accuracy and speed to classify.

Bibliography

G. P. Zhang. (2000). Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *30*(4), 451–462.

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. *Hands-on Machine Learning with Scikit-Learn and TensorFlow*. https://doi.org/10.3389/fninf.2014.00014

Hastie, T., Tibshirani, R., & Friedman, J. (2008). *The Elements of Statistical Learning The Elements of Statistical Learning*. https://doi.org/10.1198/jasa.2004.s339

Kittler, J., Hatef, M., Duin, R. P. W., & Matas, J. (1998). *On Combining Classifiers*. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* (Vol. 20). Retrieved from https://dspace.cvut.cz/bitstream/handle/10467/9443/1998-On-combining-classifiers.pdf?sequence=1

Koren, Y. (2009). *The BellKor Solution to the Netflix Grand Prize*. Retrieved from www.netflixprize.com/leaderboard

Minka, T. (2003). *MIT: Data Mining Course Notes*. Retrieved from http://alumni.media.mit.edu/~tpminka/courses/36-350/handout/handout24.pdf

Murray, I. (2018). Machine Learning and Pattern Recognition. Retrieved April 7, 2019, from http://www.inf.ed.ac.uk/teaching/courses/mlpr/2018/notes/w6a_netflix_prize.html

Russell, S., & Norvig, P. (2016). *Artificial Intelligence A Modern Approach*. *Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki* (Global Edi). https://doi.org/10.1017/S0269888900007724

Appendix A - Predictions for unlabelled binary data

Row,Prediction,Category
0 , 1.0 , plastic case
1 , 1.0 , plastic case
2 , 1.0 , plastic case
3 , 1.0 , plastic case
4 , 1.0 , plastic case
5 , 1.0 , plastic case
6 , 1.0 , plastic case
7 , 1.0 , plastic case
8 , 1.0 , plastic case
9 , 1.0 , plastic case
10 , 0.0 , book
11 , 0.0 , book
12 , 0.0 , book
13 , 0.0 , book
14 , 0.0 , book
15 , 0.0 , book
16 , 0.0 , book
17 , 0.0 , book
18 , 0.0 , book
19 , 0.0 , book

Appendix B - Predictions for unlabelled multi-class data

Row,Prediction,Category
0 , 2.0 , hand
1 , 2.0 , hand
2 , 2.0 , hand
3 , 2.0 , hand
4 , 2.0 , hand
5 , 2.0 , hand
6 , 2.0 , hand
7 , 2.0 , hand
8 , 2.0 , hand
9 , 2.0 , hand
10 , 0.0 , air
11 , 0.0 , air
12 , 0.0 , air
13 , 0.0 , air
14 , 0.0 , air
15 , 0.0 , air
16 , 0.0 , air
17 , 0.0 , air
18 , 0.0 , air
19 , 0.0 , air
20 , 3.0 , knife
21 , 3.0 , knife
22 , 3.0 , knife
23 , 3.0 , knife
24 , 3.0 , knife
25 , 3.0 , knife
26 , 3.0 , knife
27 , 3.0 , knife
28 , 3.0 , knife
29 , 3.0 , knife
30 , 1.0 , book
31 , 1.0 , book
32 , 1.0 , book
33 , 1.0 , book
34 , 1.0 , book
35 , 1.0 , book
36 , 1.0 , book
37 , 1.0 , book
38 , 1.0 , book
39 , 1.0 , book
40 , 4.0 , plastic case
41 , 4.0 , plastic case
42 , 4.0 , plastic case
43 , 4.0 , plastic case
44 , 4.0 , plastic case
45 , 4.0 , plastic case
46 , 4.0 , plastic case
47 , 4.0 , plastic case
48 , 4.0 , plastic case
49 , 4.0 , plastic case

Appendix C – Differences in sklearn versions 19.3 and 20.2 for MultiLayerPerceptron (NN) module

As noted in Section 6, different versions of sklearn provide different output for the same NN model. I cannot explain these differences, but thought I should highlight them in case the reader notices inconsistencies in the NN figures when comparing output to this paper. This is best illustrated by the below.

Sklearn 19.3 (lab machines)

Input for multi-class NN model

```
neural_network = nn(activation='logistic', random_state=40, max_iter=1500, momentum=0,
solver='sgd',early_stopping=True,  validation_fraction=0.2)
```

Output for model before fine tuning.

***** Grid search outcomes for neural network
Training score: 0.8209876543209876
Best hyper-parameters: {'alpha': 0.001, 'hidden_layer_sizes': 10, 'learning_rate_init': 1}
Model: NN
Accuracy: 0.889
Precision (macro): 0.867
Recall (macro): 0.910
F1 score (macro): 0.849

Output for model after fine tuning.

***** Grid search outcomes for neural network
Training score: 0.9012345679012346
Best hyper-parameters: {'alpha': 0.0007, 'hidden_layer_sizes': 15, 'learning_rate_init': 1.1}
Model: NN - finetuned
Accuracy: 0.944
Precision (macro): 0.960
Recall (macro): 0.960
F1 score (macro): 0.956

Now compare this to version 20.2

Sklearn 20.2 (current version)

Input for multi-class NN model (i.e. NO CHANGES FROM ABOVE, and same random seeds in rest of program)

```
neural_network = nn(activation='logistic', random_state=40, max_iter=1500, momentum=0,
solver='sgd',early_stopping=True,  validation_fraction=0.2)
```

Output for model before fine tuning.

***** Grid search outcomes for neural network
Training score: 0.9320987654320988
Best hyper-parameters: {'alpha': 0.001, 'hidden_layer_sizes': 90, 'learning_rate_init': 1}
Model: NN
Accuracy: 0.944
Precision (macro): 0.960
Recall (macro): 0.960
F1 score (macro): 0.956

Output for model after fine tuning.

***** Grid search outcomes for neural network
Training score: 0.9506172839506173
Best hyper-parameters: {'alpha': 0.0007, 'hidden_layer_sizes': 85, 'learning_rate_init': 0.9}
Model: NN - finetuned
Accuracy: 0.944
Precision (macro): 0.960
Recall (macro): 0.960
F1 score (macro): 0.956