Data Mining & Machine Learning F20DL
Coursework 2

Group 4
Lewis Wilson, Sam Fay-Hunt, Kamil Szymczak, Chun Man

December 7, 2020

Contents

1 Introduction

Based on our experience from coursework 1 we were aware that many machine learning models would perform poorly with a poorly balanced class distribution, we took measures to resolve this class distribution problem in coursework 1 and this resolved the overfitting issues.
We chose not to use these techniques so we could test the machine learning models with an inadequate dataset, to give us insight into how the different models perform in this situation.

We wrote all experiments in Python 3, using SKlearn because it was easy to implement the models under an interface and pass the discrete models as parameters to "run experiment" functions, we used Tensorflow+Keras for the CNN implementation for ease of use, and speed of training (because of easy integration with cuda).
To record and visualise experimental data we used Weights and Biases, this allowed us to perform experiments discretely and share all the results as they were generated.

The purpose of these experiments is to explore how variance in data can affect machine learning models with respect to overfitting.
To do this we ran 4 Experiments with each model, covering two machine learning paradigms, we could then compare the behaviour, and try to gain insight into if and why they perform so poorly with data using bad class distributions.
For all but the CNN we recorded the Accuracy, F1, Precision, and Recall metrics, additionally we plotted the confusion matrix, precision-recall curves, ROC curve, class proportions, calibration curve, and learning curve.

1.1 Methodology

We used the same seed when moving the data from the training set to the testing set, so all experiments for a given classifier used the same training and testing data.

2 Variation in performance with size of the training and testing sets

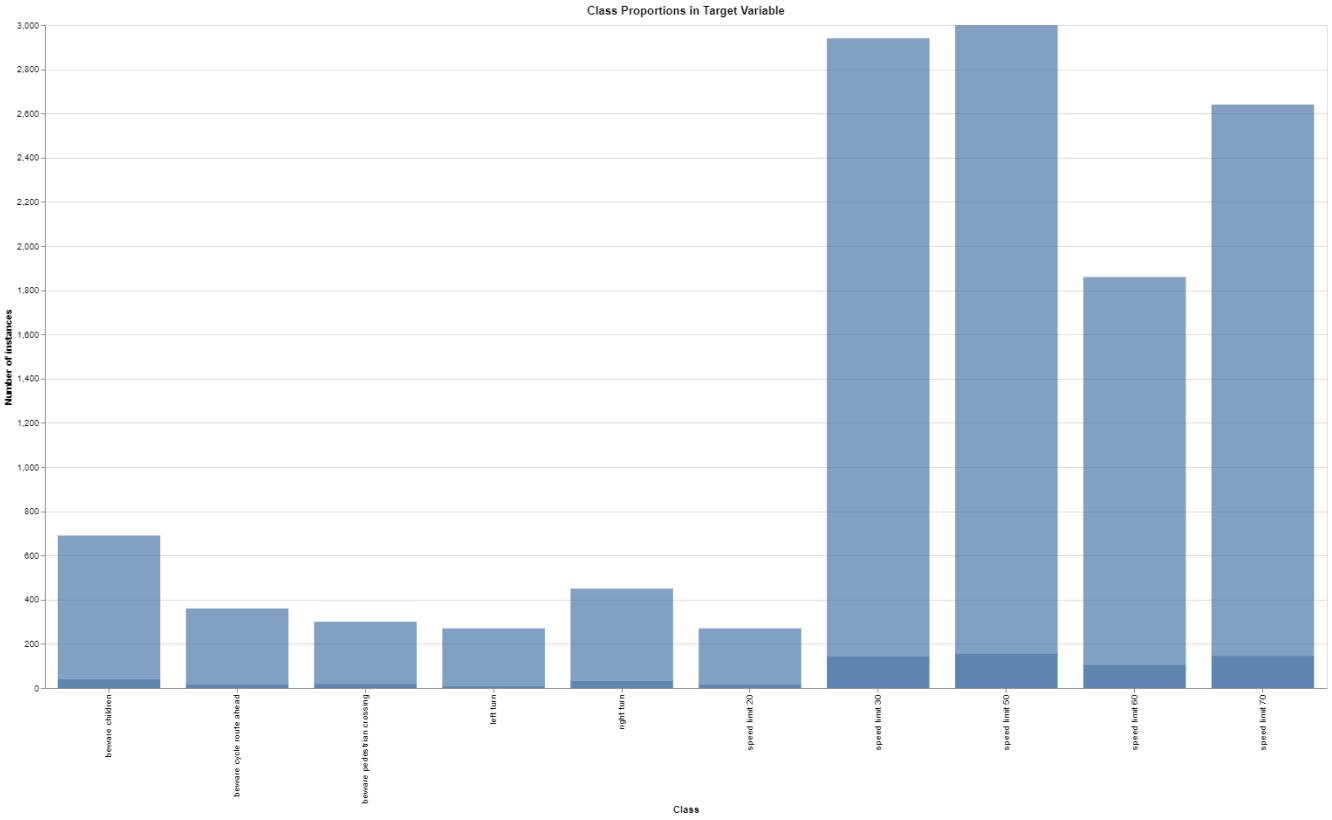
To investigate the variation in performance we selected a single class with a very poor class distribution; “Turn left”.

2.1 Inspecting Turn Left results

The turn left dataset has an extremely small number of instances in its training dataset for 9000 moved to testing, Thus we would expect the worst prediction accuracy, in this case, the situation was so bad that our accuracy scores became misleading. Because the representation of ‘turn left’ was so poor in our dataset most of the classifiers simply predicted that there were no instances of turn left in the dataset and recorded a deceiving 97% accuracy (because 97% of the dataset is not in fact turn left). It turned out this problem appeared for all binary classifications with the Random Forest classifier and Multilayer perceptron.

The only classifiers that did not exhibit this behaviour were the Logistic Regression (LR) and Convolutional Neural Network classifiers (CNN). CNN scored a staggering 99.9% accuracy with the same minuscule training data, while LR managed 98.3% with 200 false negatives, 5 false positives and 60 true positives.

Figure 1: Class proportions in target variable



An example distribution of labels between train (dark blue) and test (light blue), for the Beware cycle route ahead class.

Figure 2: Class proportions in target variable

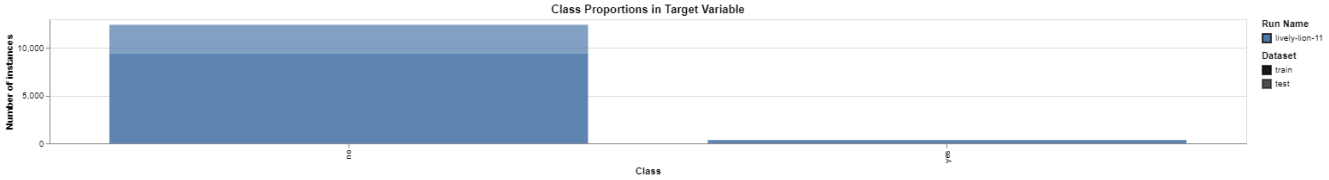


Figure 3: Class proportions in target variable

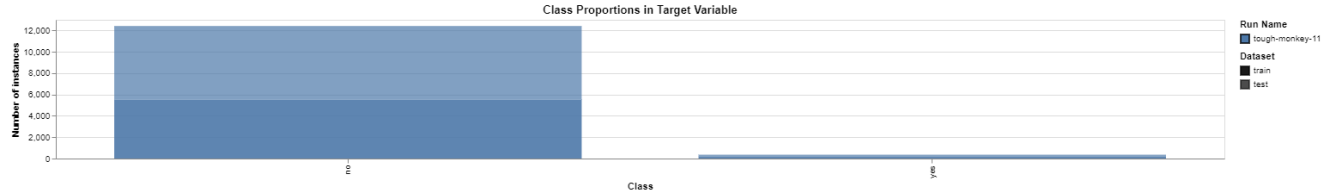
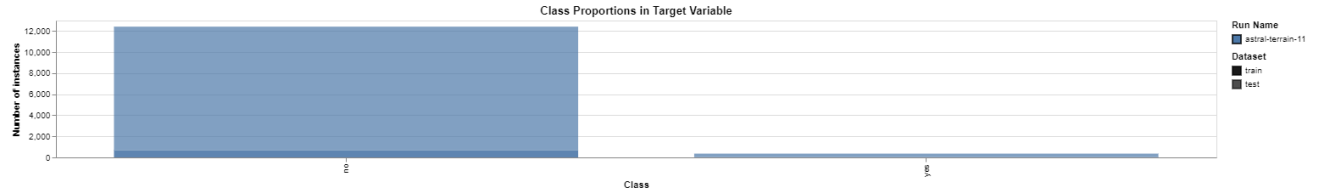
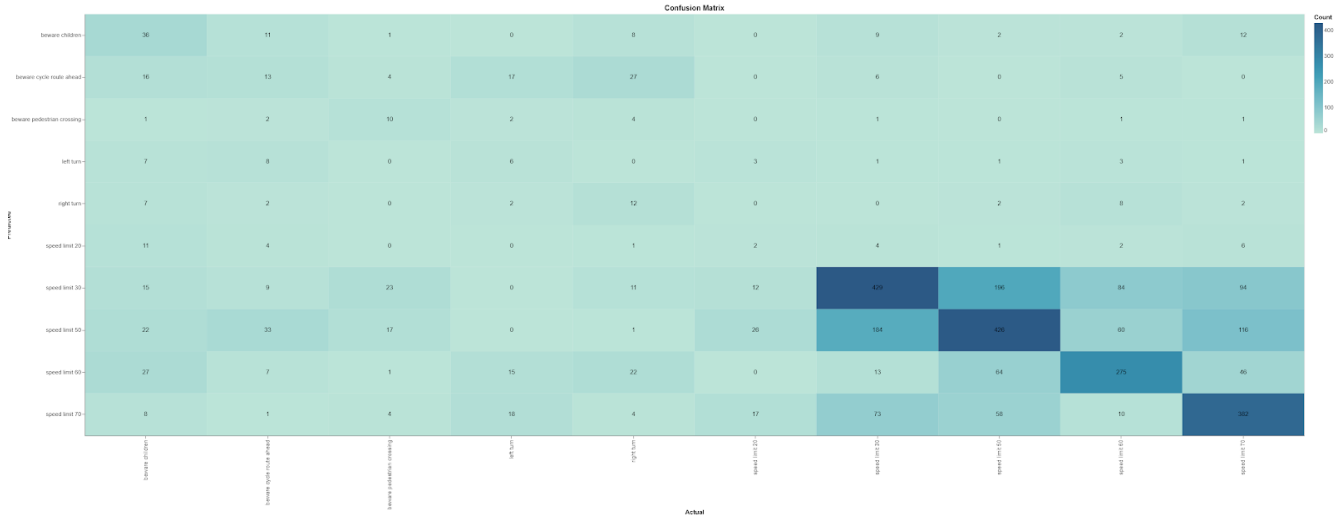


Figure 4: Class proportions in target variable



Starting out we observed the results of J48, the issues are immediately clear, even in the experiment with the full training set J48 is poor at attempting to predict on this kind of data, when working with all classes it is unable to reliably distinguish between the speed limit signs, and a non-speed limit sign, but beyond that it consistently quite evenly classified with any class label.

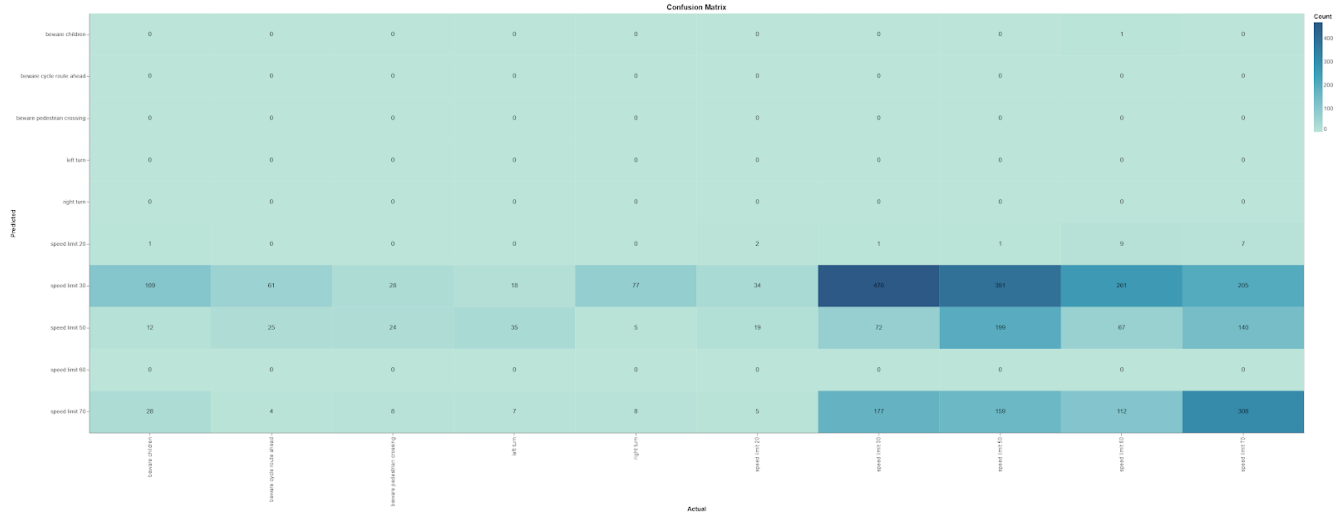
Figure 5: J48 Confusion Matrix



As the test set grows we can observe a similar story, the J48 classifier gets substantially worse at distinguishing between speed limit signs and other signs because the representation of those signs in the dataset gets very low.

With all the classifiers except Logistic Regression and the CNN we see the same story, Multi-layer perceptron and random forest particularly suffers from this poor class representation issue in the data. We can see for figure [x] that the Multilayer perceptron assigns most labels for 3 classes: Speed limit 30, 50, 70. It also assigns a small number of labels for speed limit 70.

Figure 6: Multilayer Perceptron Confusion Matrix



3 Variation in performance with the change in the learning paradigm (Decision Trees versus Neural Nets)

- Decision trees performed comparatively poorly, particularly when the training data was reduced.
- Multilayer Perceptron had overfitting issues, Logistic Regression was the best, aside from the CNN

We have found decision trees performed very poorly particularly when the training data was reduced and also so did the MultiLayer Perceptron which had overfitting issues. Both of these classifiers had 0 recall and 0 precision for predicting individual classes datasets. This means they have not predicted correctly a single sign and should not be used in prediction road signs from images. Our results perfectly support the weak points about decision trees such as that there are variable relationships that decision trees just can't learn and they are unstable as a small change in the data can lead to a large change in the structure of the optimal decision tree [7].

However Logistic Regression performed considerably better, it was actually the best performer from all classification techniques we have used and tested, aside from the CNN.

Logistic Regression gives the best results because the benefit of logistic regression is its interpretability as it understands which pixels are important to determine what class an image belongs to.

Convolutional Neural Networks worked phenomenally as expected, as they are widely used in image classification.

4 Variation in performance with varying learning parameters in Decision Trees

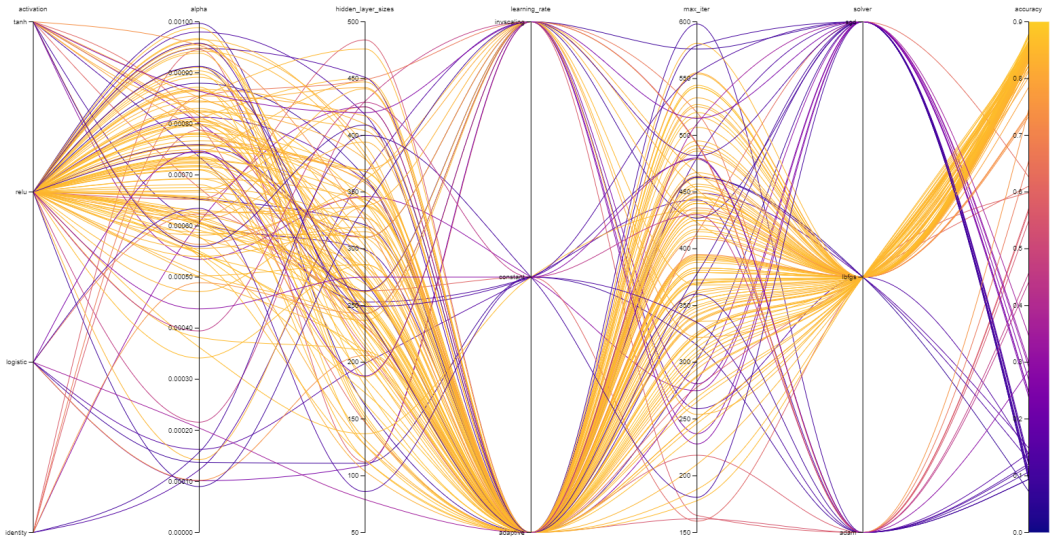
4.1 Optimising Hyperparameters

[6]

Our hyperparameter optimization used Bayesian Optimization [2]. This gaussian process models the function and then chooses parameters to optimize by determining the probability of improvement according to the accuracy metric.

We opted to use the 'all classes' dataset for these sweeps because in our preliminary experiments we noticed the worst prediction accuracy with 10 classes.

Figure below shows high level overview of the MLP sweep, dark blue lines indicate lower accuracy and yellow had the highest accuracy. (See Figure 5.2)



For the evaluation of our hyperparameters, we gathered data on the linear correlation between each hyperparameter and their accuracy. THe feature importance is calculated from training a random forest with hyperparameters and its metrics. [3]

The feature importance metric is useful to identify the features that have a larger impact on the output of the model, coupled with correlation we can find the direction of the relationship. It may be the case that a very important feature is extremely bad for the result, in this case we will see a strong negative correlation, this information helps us hone in on the best hyperparameters to use to maximise accuracy.

This however caused the models to instead maximise hyperparameters tendency to overfit this very imbalanced dataset.

4.2 J48

Discussed below is the hyperparameter [6] importance concerning the performance of J48 trees:

max_depth - The maximum depth of the tree.
min_impurity_decrease - Splits a node if this split induces a decrease of the impurity greater than or equal to this value.
min_samples_leaf - the minimum number of samples to be considered a leaf.
min_weight_fraction_leaf - The minimum number of samples required to be at a leaf node.
criterion_value -['Gini','Entropy']: Measures the quality of the split. **max_features -** ['auto','log2','sqrt'] : max features considers the number of features when looking for the best split. Auto just picks the best result so had the same result as log2.
Splitter - ['best', 'random']: The strategy used to choose the split at each node.

Parameter	**Type**	**Conclusion**
max_depth	int	Provided low importance value of 0.012 and provided some correlation 0.084.
min_impurity_decrease	float	Small importance value of 0.051 and a strong negative correlation of -0.241.
min_samples_leaf	int or float	Low importance of 0.014 and provided a tiny positive correlation of 0.007.
min_weight_fraction_leaf	float	Gives a strong negative correlation in terms of accuracy, meaning the higher the min_weight_fraction_leaf value, the lower the accuracy.
criterion	string	Both parameters provided a very low importance value of 0.003. Entropy had a tiny negative correlation of -0.022 and Gini a positive correlation of 0.022.
max_features	string	Log2 provided an importance value of 0.133 and a high negative correlation of -0.372. Log2 provided tiny importance of 0.003 but a relatively good correlation of 0.196.
splitter	string	'best' had an importance of 0.048 and random 0.044. 'best' has a positive correlation of 0.051 whereas random -0.051.

See Parameter Importance (Figure B.1)

4.3 Random Forest

[1]

min_samples_split - The minimum number of samples required to split an internal node
min_samples_leaf - The minimum number of samples required to be at a leaf node.
n_estimators - The number of trees in the forest.
min_weight_fraction_leaf - The minimum number of samples required to be at a leaf node.
max_features - The number of features to consider when looking for the best
criterion - The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

Parameter	**Type**	**Conclusion**
max_features	string	Contains the options "auto", "sqrt" and "log2". We discovered that "sqrt" has a higher accuracy overall, the accuracy of "log2" varies between the lower end and the median accuracy value.
min_samples_split	int or float	The minimum samples required to split a node has very little impact on accuracy.
criterion	string	Gives a perfect negative correlation with respect to accuracy. Correlation values being [Gini = -0.404], [Entropy = 0.404].
n_estimators	int	This is defined as the number of trees in the forest, it seems to have very little correlation but high importance.
min_samples_leaf	int or float	Gives a strong negative correlation in terms of accuracy, meaning the higher minimum samples at a leaf node, the lower the accuracy.
min_weight_fraction_leaf	float	Has a somewhat positive correlation to accuracy. e.g. total weight required at a leaf node varies between 76% and 89% accuracy

See Parameter Importance (Figure C.1)

5 Variation in performance with varying learning parameters in Neural Networks

5.1 Linear Classifier

[4]

For a linear classifier we have decided to use logistic regression. Shown below is the hyperparameter importance concerning the performance of Logistic Regression.

The accuracy fluctuates depending on which hyperparameters are used. Logistic Regression has a 'solver' hyperparameter in sklearn which is the algorithm to use in the optimization. We have tested the following: 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'. Sag and Saga give the best results with 89% accuracy. Liblinear isn't far behind with 88Other solvers have a negative impact on the performance, this can be seen in the Solver parameter importance table where their correlation values are negative.

Additionally 'sag' and 'saga' guarantee fast convergence on features with approximately the same scale [R41] which is the case with our datasets as pixel values have the same scale. This can be seen as the number of iterations between 200-800 don't change much in the accuracy and from the scatter chart of the accuracy of the Logistic Regression vs the number of sweeps created.

Parameter	**Type**	**Conclusion**
C	float	The smaller this value is the stronger the regularization. For this we settled on 0.08027. Although this parameter did not make a big d ifference and provided similar results for values between 0.8 and 1.2
fit_intercept	bool	From testing we have found out that for this dataset adding a bias works better than not having bias thus True
max_iter	int	We settled on 342 maximum number of iterations taken for the solver to converge. This parameter does not have a big importance but we found 342 gives accurate results.
solver	string	Contains the options 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga', default='lbfgs'. 'Saga' gives the best accuracy overall. Solver is the most influential hyperparameter as we can see from the importance and correlation table
tol	float	tolerance for stopping criteria which tells the algorithm to stop searching when some tolerance is achieved. This parameter did not make a difference, we settled on 0.0002277

See Parameter Importance (Figure D.1)

5.2 Multilayer Perceptron

[5]

Our sweep on MLP indicated that the relu activation function provided the highest accuracy metric for all classes, our search computed this as the most important metric. However using this we observed severe overfitting on the speed limit signs in all experiments.

Activation - Activation function for the hidden layer.
Alpha - L2 penalty (regularization term) parameter.
Hidden_layer_sizes - Number of neurons in each hidden layer.
Max_iter - Maximum number of iterations. The solver iterates until convergence
Solver - The solver for weight optimization.
Learning_rate - The learning rate controls how quickly the model is adapted to the problem.

Parameter	**Type**	**Conclusion**
alpha	float	When alpha increases the loss increases therefore we went with a relatively low number of 0.0005508.
solver	string	'Lbfgs' solver did provide the best accuracy consistently when compared to other solvers such as 'adam' and 'sgd' which supports the statement from [5] staying that 'Lbfgs' works best for small datasets.
max_iter	int	Maximum number of iterations. We used 520 as it gives good results but that is the max so it converges earlier, as long as value above 200 is used will give high accuracy.
activation	string	Contains the options: 'identity', 'logistic', 'tanh', 'relu'. We found relu worked best for our use case
learning_rate	string	'adaptive' achieves the highest accuracy while, 'constant' and 'invscaling' vary widely.
hidden_layer_sizes	tuple	337 layers worked well for us. But this parameter is not very important as long at its above a fairly high value such as 200.

See Parameter Importance (Figure E.1)

5.3 Convolutional Neural Networks

Layer type	Output Shape	Param #
conv2D	(None, 48, 48, 75)	750
BatchNormalization	(None, 48, 48, 75)	300
MaxPooling2D	(None, 24, 24, 75)	0
Conv2D	(None, 24, 24, 50)	33800
Dropout	(None, 24, 24, 50)	0
BatchNormalization	(None, 24, 24, 50)	200
MaxPooling2D	(None, 12, 12, 50)	0
Conv2D	(None, 12, 12, 25)	11275
BatchNormalization	(None, 12, 12, 25)	100
MaxPooling2D	(None, 6, 6, 25)	0
Flatten	(None, 900)	0
Dense	(None, 512)	461312
Dense	(None, 512)	262656
Dropout	(None, 512)	0
Dense	(None, 10)	5130

5.3.1 Critical limitation of CNN experimental methodology:

We were unable to disable memoization between experiments over each loop, so all the parameters are restored between runs within a given experiment, consequently we noted only significant changes to the loss when training on all classes, after which convergence was achieved in a single epoch. We attempted to clear the model parameters, clear the GPU memory, and model save the randomly initialised parameters and load them back in between experiments but could not prevent the CNN model using data encoded from previous experiments. This does invalidate our results, when compared to the discrete experiments in the previous sections, however this can be a demonstrable advantage of a CNN, in this situation it consistently correctly classifies the images despite changing the dimensions of the output layer and the classification targets (albeit on the same dataset).

During all experiments with training and testing data, at worst our CNN scored an accuracy of 97% on the test dataset when classifying all classes, the CNN was capable of correct classification even when 9000 instances were moved from the training set to the testing set.

The CNN was unsurprisingly the best performing classifier, In many of the kfold experiments the CNN managed an accuracy of 100%, but this could be due in part to the memoization effect between experiments, this is particularly noticeable on the kfold experiments.

6 Variation in performance according to different metrics (TP Rate, FP Rate, Precision, Recall, F Measure, ROC Area)

6.1 J48 Experiments

J48 with default test set cycle route ahead binary classification gave a very high precision (0.95) and a really low recall (0.2) Precision shows the ratio of true positives to all the positives, therefore in our use case we have a ratio of signs that the classifier classified correctly to the total the classifier thought are positive.

Using cycle route ahead dataset as an example, for all signs that are actually cycle route ahead signs the recall tells us how many the classifier classified as cycle route ahead signs. Therefore J48 is very picky and doesn't think many signs are cycle route ahead signs. Pretty much all signs it thinks are cycle route ahead signs are indeed cycle route ahead signs. However it also misses a lot of cycle route ahead signs, because it is very picky.

With the default test set, J48 provides a precision, recall and f1 scores of 0 for the right turn, left turn, beware pedestrian crossing and speed limit 20 classes the reason is due to 0 true positive classifications.When we investigate why this is for the beware pedestrian crossing class, we noticed the data has a heavy skew of class proportions towards 'No' with very few instances of 'Yes' found. In fact, this is the case for all of the class labels with scores of 0 for precision, recall and f1.

6.2 Random Forest

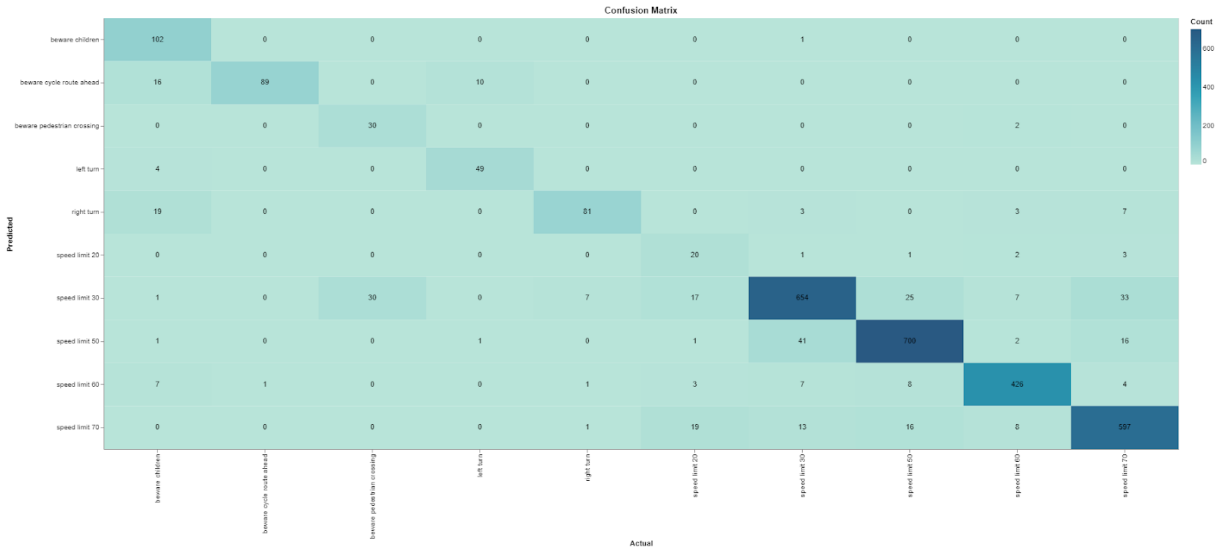
We found random forest to be one of the worst performers overall with the training and test sets. With binary class labels it scored 0 for precision, recall, and F1 in all experiments. We hypothesize this is caused by the same reasons described in section [1]. Likewise all classes perform poorly by classifying all classes as exclusively either speed limit 30, 50, and 70 in the training and testing data sets, k folds simply classifies all classes as speed limit 50; the most well represented classes in our dataset.

6.3 Multi-Layer Perceptron

NEEDS COMPLETED

6.4 Logistic Regresssion

Logistic Regression is excellent at predicting classes for the dataset, which contains all signs, which is not easy to do. Logistic Regression has relatively good precision and recall when compared to other classifiers with 89 for both. This means it is very good at predicting all different signs.


Confusion Matrix
Count
600
400
200
0
Predicted
beware children
beware cycle route ahead
beware pedestrian crossing
left turn
right turn
speed limit 20
speed limit 30
speed limit 50
speed limit 60
speed limit 70
102 0 0 0 0 0 1 0 0 0
16 89 0 10 0 0 0 0 0 0
0 0 30 0 0 0 0 0 2 0
4 0 0 49 0 0 0 0 0 0
19 0 0 0 81 0 3 0 3 7
0 0 0 0 0 20 1 1 2 3
1 0 30 0 7 17 654 25 7 33
1 0 0 1 0 1 41 709 2 16
7 1 0 0 1 3 7 8 426 4
0 0 0 0 1 19 13 16 8 597
Actual

Appendices

A Appendix A

A.1 Workload split

Team member	Involvement
Lewis Wilson	text here
Chun Man	Contributed to the experimenting and analysis of Random Forest, Report writing.
Sam Fay-Hunt	text here
Kamil Szymczak	text here

As a team we are happy with everyone's contributions to the project. All team members were punctual and showed up to all scheduled meetings. Sam took the lead as project manager throughout the project delegating the workload and providing support to others.

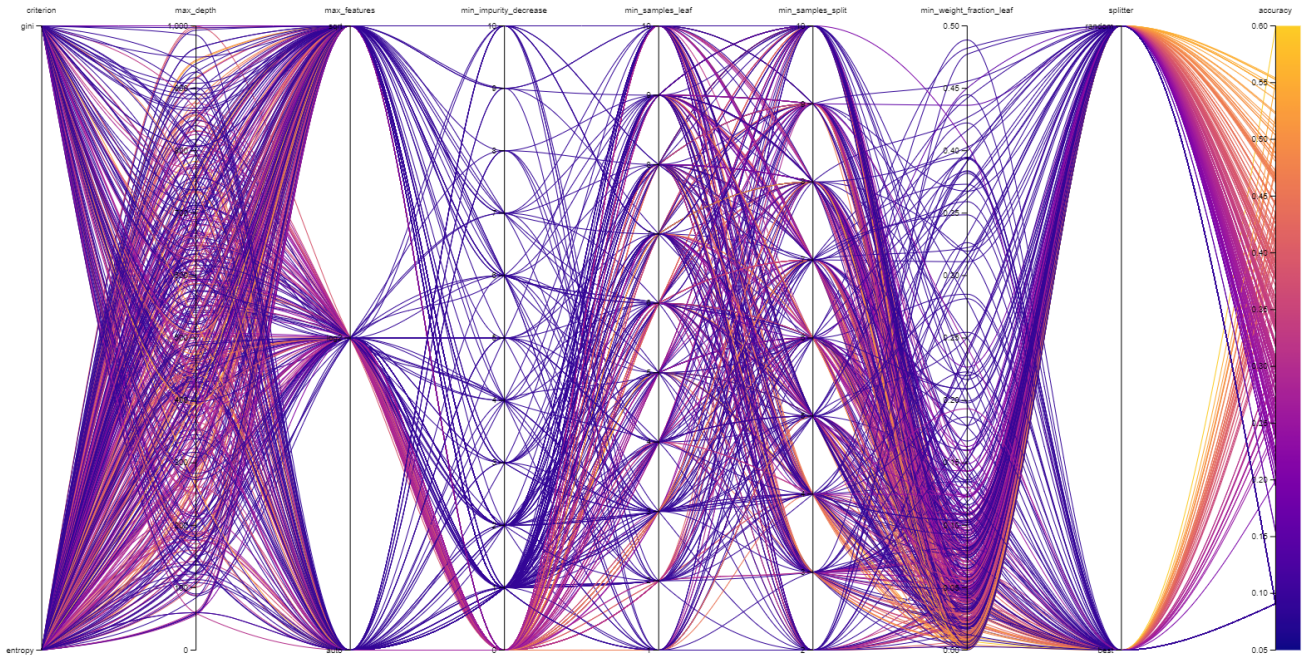
B J48

B.1 J48 Parameter Importance

Parameter Config	Importance	Correlation
min_weight_fraction_leaf	0.668	-0.666
min_impurity_decrease	0.052	-0.241
min_samples_leaf	0.016	0.007
max_depth	0.012	0.084
min_samples_split	0.009	-0.108

Parameter Config	Importance	Correlation
max_features.value_log2	0.133	-0.372
max_features.value_sqrt	0.003	0.196
splitter.value_best	0.049	0.051
splitter.value_random	0.043	-0.051
criterion.value_entropy	0.002	-0.022
criterion.value_gini	0.002	0.022

Figure 7: J48 Parameters



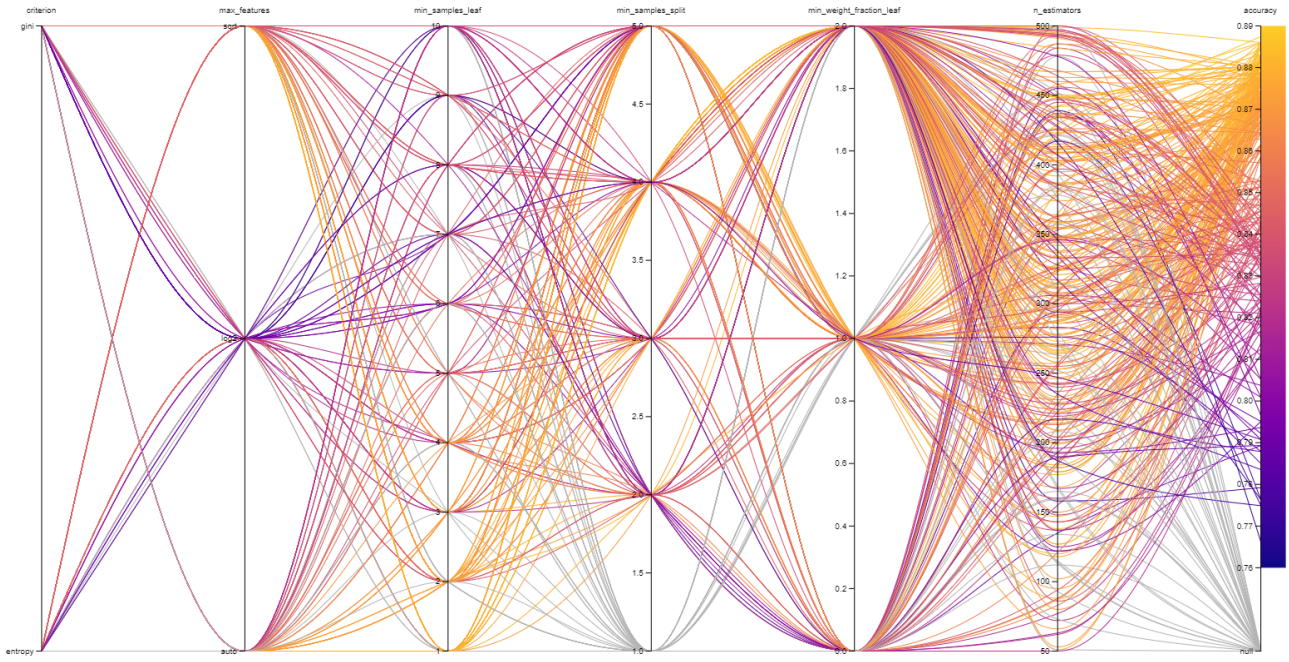
C Random Forest

C.1 Random Forest Parameter Importance

Parameter Config	Importance	Correlation
min_samples_split	0.005	0.306
min_samples_leaf	0.375	-0.725
n_estimators	0.016	0.092
min_weight_fraction_leaf	0.013	0.123

Parameter Config	Importance	Correlation
max_features.value_sqrt	0.001	0.563
max_features.value_log2	0.565	-0.752
criterion.value_entropy	0.012	0.404
criterion.value_gini	0.012	-0.404

Figure 8: Random Forest Parameters



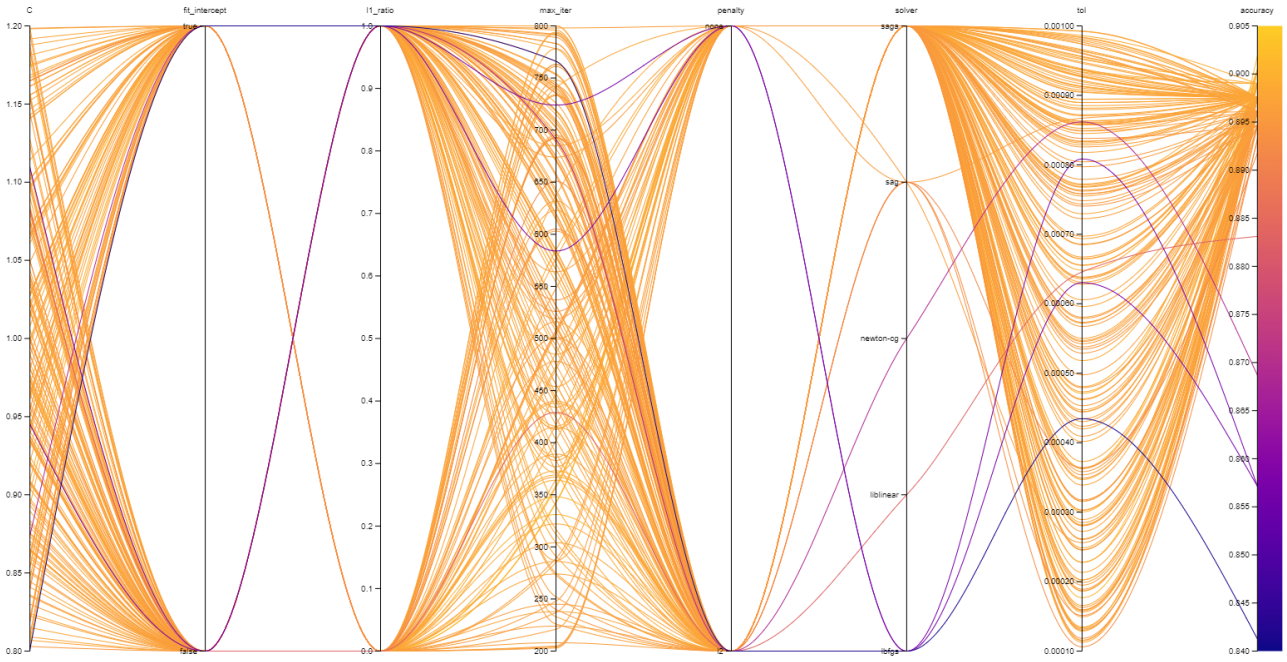
D Linear Classifier

D.1 Linear Classifier Parameter Importance

Solver Parameter Config	Importance	Correlation
lbfgs	0.786	-0.869
newton-cg	0.159	-0.271
liblinear	0.042	-0.037
saga	0.009	0.691
sag	0.003	0.170

Config Parameter Config	Importance	Correlation
max-iter	0.194	-0.634
l1-ratio	0.124	-0.510
fit-intercept	0.058	0.367
tol	0.046	-0.174
C	0.031	0.260

Figure 9: Random Forest Parameters



E Multilayer Perceptron

E.1 Multilayer Perceptron Parameter Importance

Parameter Config	Importance	Correlation
hidden_layer_sizes	0.095	-0.101
max_iter	0.091	0.072
alpha	0.061	0.171

Parameter Config	Importance	Correlation
solver.value_lbfgs	0.530	0.728
solver.value_adam	0.027	-0.245
solver.value_sgd	0.024	-0.640
activation.value_identity	0.098	-0.169
activation.value_relu	0.033	0.301
activation.value_tanh	0.018	-0.035
activation.value_logistic	0.006	-0.270
learning_rate.value_adaptive	0.012	0.507
learning_rate.value_constant	0.004	-0.442
learning_rate.value_invscaling	0.001	-0.224

Figure 10: Multilayer Perceptron Parameters

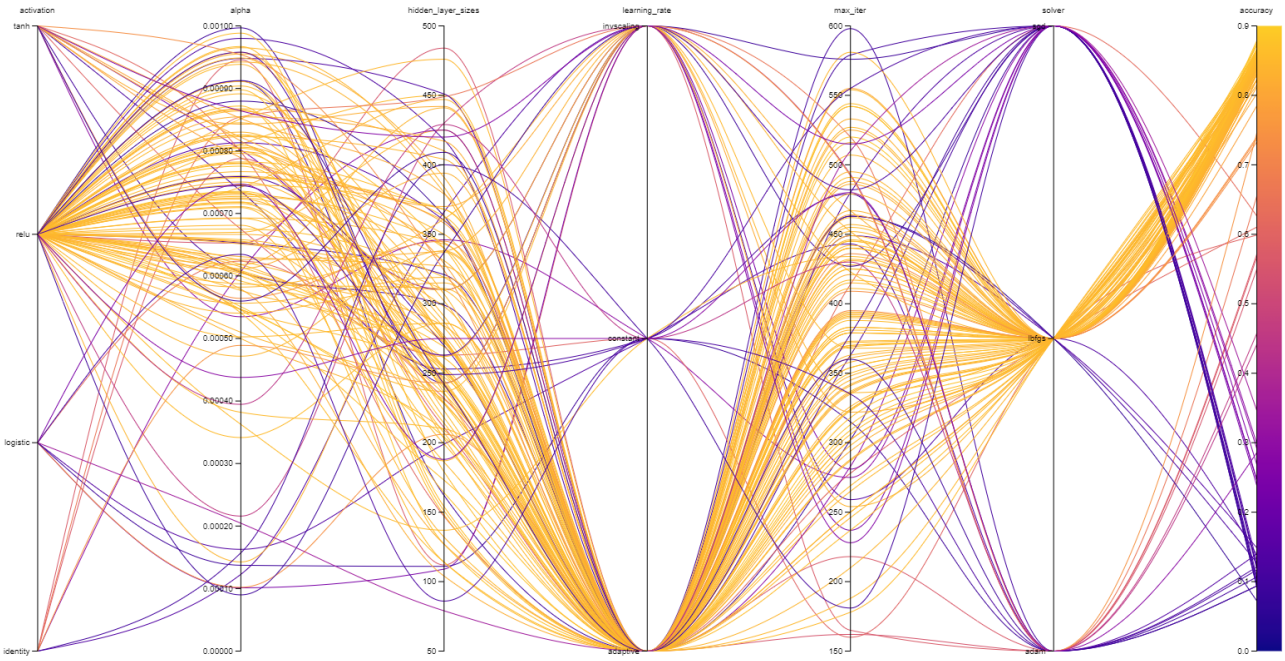


Figure 11: Data with evenly split groups

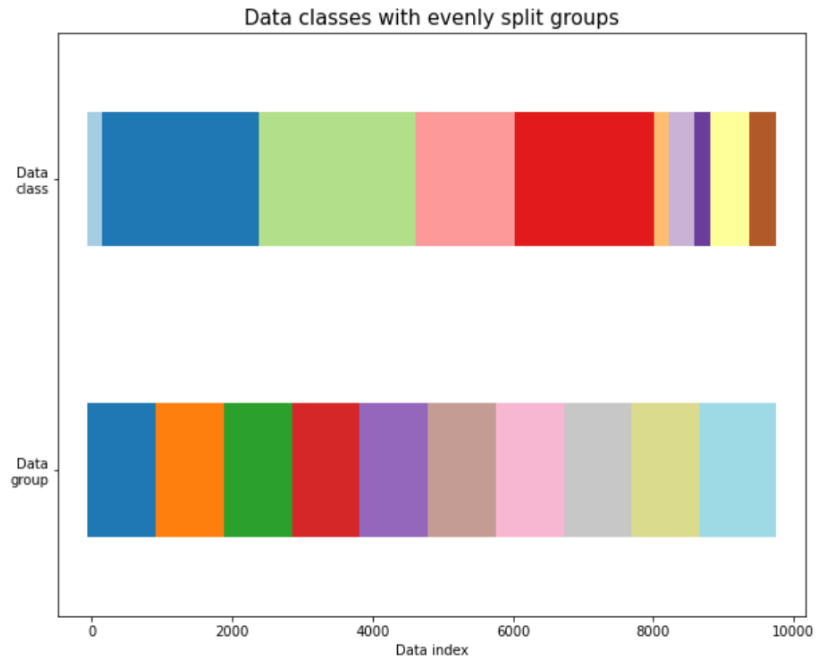


Figure 12: KFold

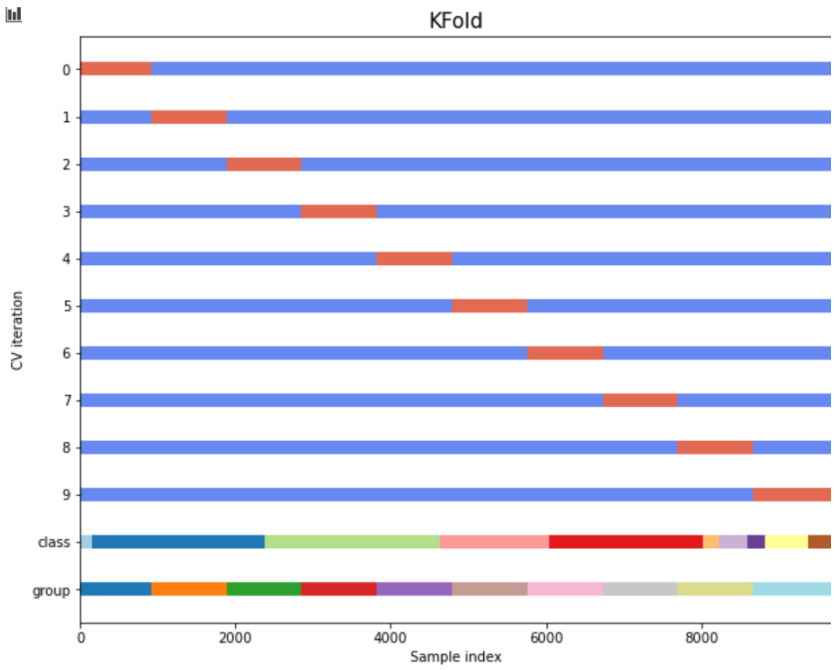
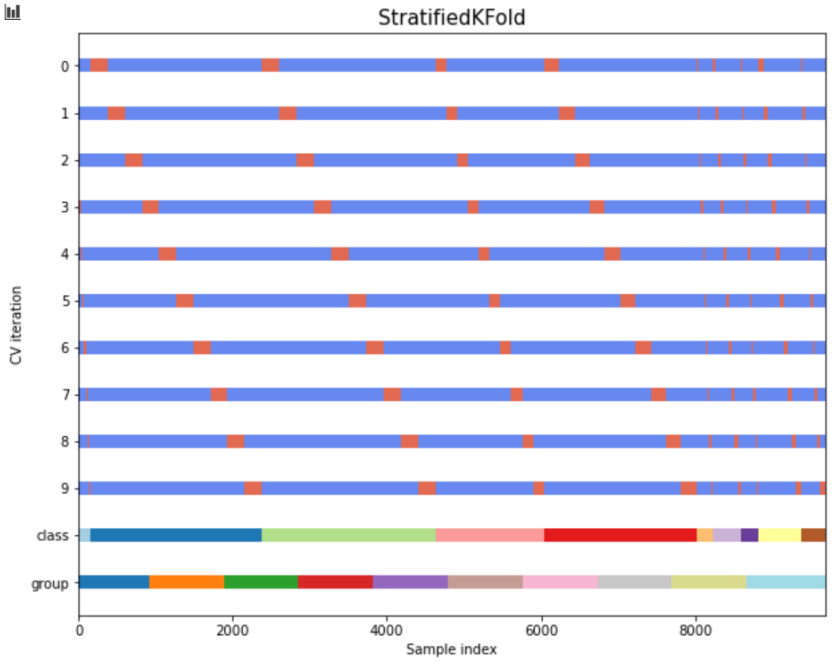


Figure 13: Stratified KFolds



References

[1] 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.23.2 documentation.

[2] Configuration.

[3] Parameter Importance.

[4] Sklearn.linear_model.LogisticRegression — scikit-learn 0.23.2 documentation.

[5] Sklearn.neural_network.MLPClassifier — scikit-learn 0.23.2 documentation.

[6] Sklearn.tree.DecisionTreeClassifier — scikit-learn 0.23.2 documentation.

[7] Y. Zhao and Y. Zhang. Comparison of decision tree methods for finding active objects. 41(12):1955–1959.