Machine Learning Assignment 2 Report

Craig Barry (s3601725) & Neil D'Souza (s3600251)

Decision Tree Task Analysis

Preprocessing

On first analysis of the data there is obviously a lot of missing data. The first step taken was to remove any rows in which the price_band attribute was 'Unknown', this is due to the fact that if we don't know the target value it is not helpful for training or testing purposes.

The next step was to add in the missing attributes. This was done by splitting the data into frames where each frame contains the rows where the price_band value was the same. In each column of the new frames, we take an average of the existing data in the column and use it to fill in the rest of the missing values. 3 different kinds of averages were attempted to calculate the missing values including: median, most frequent & mean. However, all three types of averages only made a difference of 0.7% on the overall accuracy on a standard decision tree classifier, median was chosen but all 3 provided very similar results.

The final step in preprocessing was to encode some of the data-types, a classifier could not be fit to the set unless the non-numeric features were converted to numeric features. These columns that needed to be encoded included:

- Address
- Date
- Price_bands
- Council_area
- Region_name
- Suburb
- Realestate_agent
- Type
- Method

Feature Selection

Figure 1a

Feature selection was used to identify which features of the data set actually meaningfully contributed to the price band the property would belong to. Using a standard decision tree classifier the number of features was incremented until an optimal number of features were found. As shown in ***Firgure 1a*** more than 12 features provides no additional helpful information to help classify the data. The features selected by SelectKBest feature selector using the f_classif selector were:

- Rooms
- Type
- Method
- Realestate_agent
- Distance
- Postcode
- Bedrooms
- Bathrooms
- Car_parks
- Landsize
- Building_area

Interestingly almost identical results and the same optimal feature set was returned by using a SelectKBest feature selector with a chi2 selector and using a VarianceSelector.

Classifier Evaluation

Before we can start training decision trees we need to be able to evaluate their performance. This was done using a K-Fold cross validation to split the data into training & testing sets and get consistent evaluation. The actual evaluation was simply an accuracy function as predicting property price bands incorrectly would not have significant importance if the false prediction was a false-negative or false-positive, it only matters that we have the highest possible accuracy. Therefore accuracy was seen as a simple & effective evaluator for this problem.

Classification

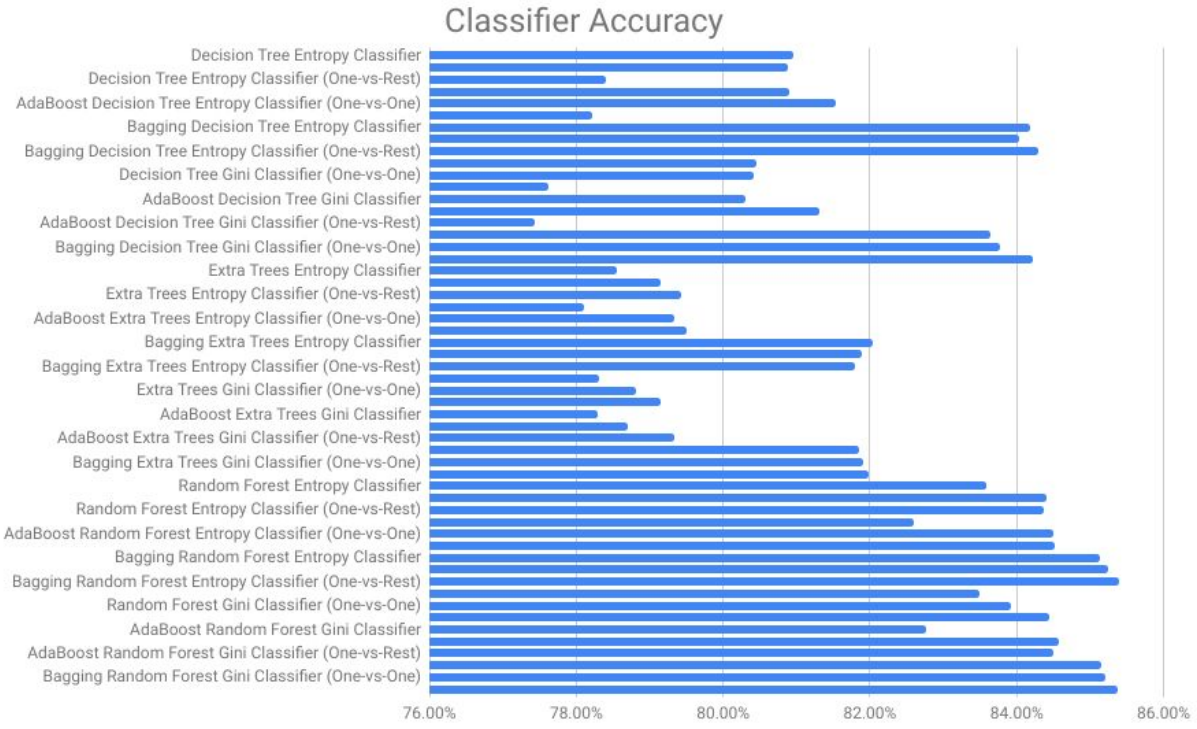

Figure 1b

Classification involved training different kinds of decision tree classifiers and then applying various ensemble learners & multiclass optimizers to try and improve their performance. Three main classifiers were used to trained to attempt to predict values from the property prices dataset:

1. Decision tree classifier
2. Random forest classifier
3. Extra trees classifier

These standard classifiers were combined with combinations of:

1. AdaBoost
2. Bagging
3. Training a classifier for each output class (One-vs-Rest classification)
4. Training a classifier for each pair of classes (One-vs-One classification)

Each of the classifiers were also trained using both measures of entropy & gini. Although the measure used made very little difference in the overall accuracy.

The accuracy of each classifier using the evaluation method discussed in the **Classifier Evaluation** section can be seen in ***Figure 1b***. Overall the random forest classifiers with bagging provided the best overall performance, as shown by the top 10 classifiers list in ***Figure 1c***.

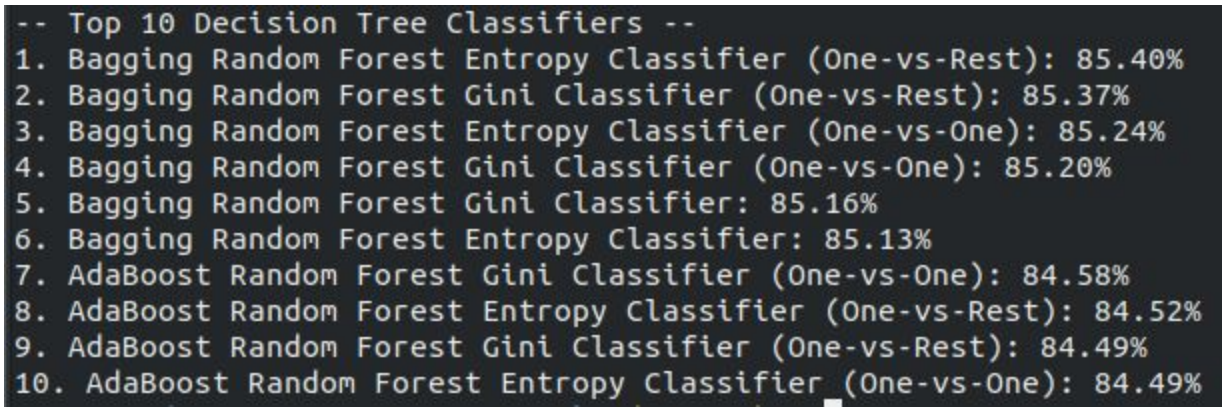

Figure 1c

Conclusion

Overall we were able to obtain approximately ~86% accuracy when predicting the price bands of the dataset with decision tree classifiers. However, using decision trees to classify the dataset was both simple and provided decent performance, even if a decision tree was not able to produce the best performance model it would provide good insight into what attributes were useful and what roles they played in the final classification, making it at the very least an excellent starting point to training an accurate model. Ultimately I would use the random forest classifier with bagging to classify the price bands as the random forest had much stronger more consistent performance and bagging provided better performance than any other of the decision tree optimizations on each of the classifiers.

Finally I don't believe price bands are a good indicator of property prices and in fact may produce very misleading information, for example 2 houses in the 2M-20M price band could have a price variation of up to 1000%, this is a huge generalization to make about the price of a house, such generalizations can really only produce very general information.

Perceptron Tree Task Analysis

Final results

Run no.	Epochs	No. Hidden Nodes	Training Accuracy %	Testing Accuracy %
1	10	74	0.8564%	0.8664%
2	**10**	**200**	**0.8713%**	**0.8810%**
3	10	250	0.8614%	0.8737%
4	10	431	0.8627%	0.8471%
5	10	748	0.8546%	0.8391%

Figure 2a

The method of choosing the correct amount of Hidden nodes was to take the halfway mark between the two recommended endpoints (74 and 748) and to pick somewhere roughly halfway. The process was repeated that process for the remaining two parameters, then run in ascending order.

The process of choosing the correct amount of epochs was not explored in this project but there was some evidence that more epochs does not always correspond with better results.

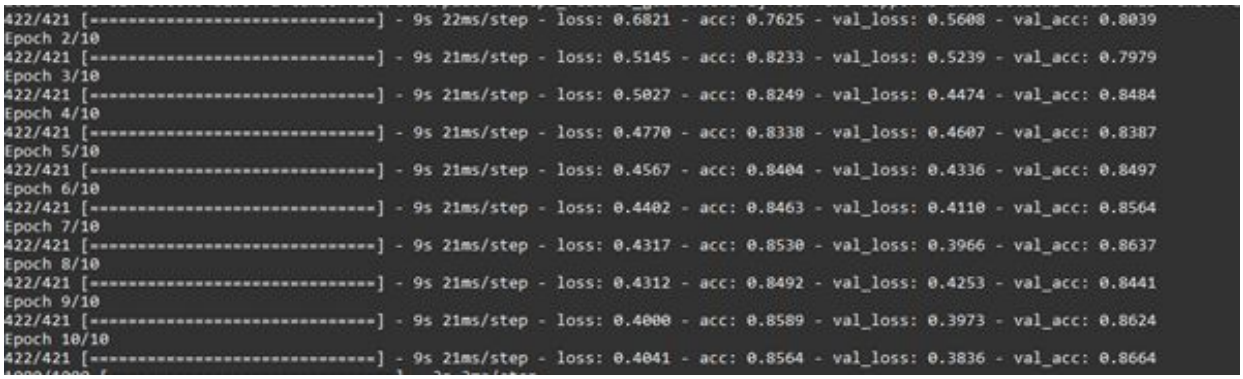

Figure 2b

Above are the verbose logs for the 74 hidden node model. Despite becoming incrementally better over most epochs, the loss function stopped decreasing on the final epoch. While 10 epochs is not enough trials to determine if this was just a slump or a trend, it is quite obvious that there will be a point where having more epochs will no longer improve the models

performance. A method of determining the correct number of epochs may be to choose a large number of epochs and ‘early-stop’ the process when the loss function stops decreasing.

Training

The training process is carried out in the MLP (Multi-layered perceptron) from the Keras sequential API. Keras models are trained using Numpy arrays as input data and labels for categorisation. After pre-processing, loading and configuring data, the perceptron can begin training for a number of epochs meaning iteratively over the dataset. Typically the training outputs will get more accurate over time as there will be more inputs which have been analysed creating a robust set of viewed samples. The output is determined by a probabilistic distribution known as the softmax. Softmax is well suited to categorical classification due to being able to determine the probability of many classes and not just one or the other. By the time an output is given the multi-layered network can compare that output to the correct classification and calculate the error with a predefined loss function (in this case ‘categorical-classentropy’). This error is important to calculate so it can be sent back in the network to adjust the weights of which neurons are fired. The optimiser used to adjust the weights in this perceptron is adam, an alternative to scholastic gradient descent.

Underfitting and Overfitting

Underfitting refers to the case of when the model has not generalised well to the training data and will also not be a good generaliser of new data. If the MLP is under fitting it may for example be failing to discern any meaningful differences between any clothing items or any similarities between items which belong in the same class. This can be detected quite easily with the use of metrics. The accuracy metric was used in this MLP, it functions by outputting a numerical value similar to how the error is calculated, however while the error was mentioned in training because it is used to adjust weights, metrics are not sent back in the network and instead provide a good case of underfitting with a low score.

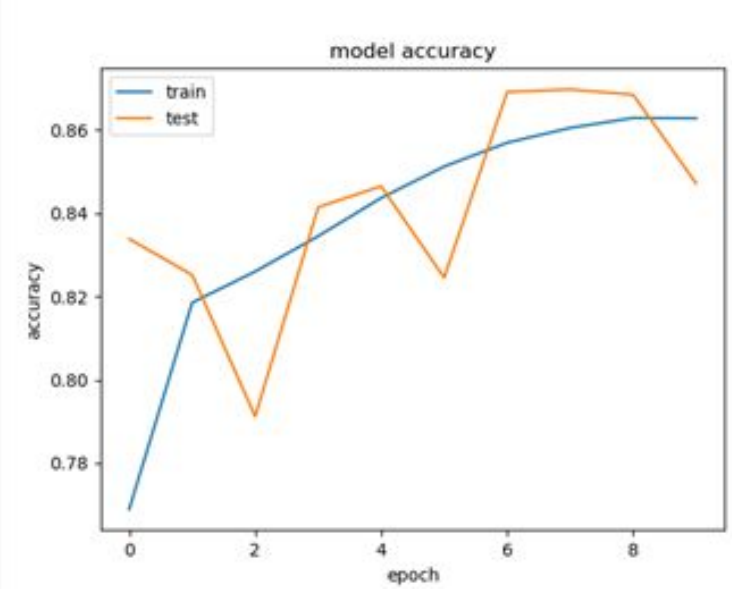

Figure 2c

The above graph indicates the accuracy over each epoch where a model looks to have under fitted to some degree, as the training line appears more linear than the actual test line, despite the overall trend being the same. Overfitting refers to the case when the model has specialised on the training data including things which should be disregarded such as irrelevant details and random noise. A model which overfits will yield a good accuracy score on the training data but not such a good score when it comes to classifying new data because it cannot generalise. A crude example of this might be the model deciding that something is only a T-shirt if it has a particular pattern on the front, being unable to generalise that T-shirts do not require a pattern. This can be detected by splitting the training data into training sets and testing sets, where the testing sets are not seen until the end of training, providing a much more accurate error score. K-folds cross validation is another technique which is used to create mini-test sets across the training data, allowing all of the training data to be used for testing and leaving the true test set 100% undiscovered until the final model is selected.

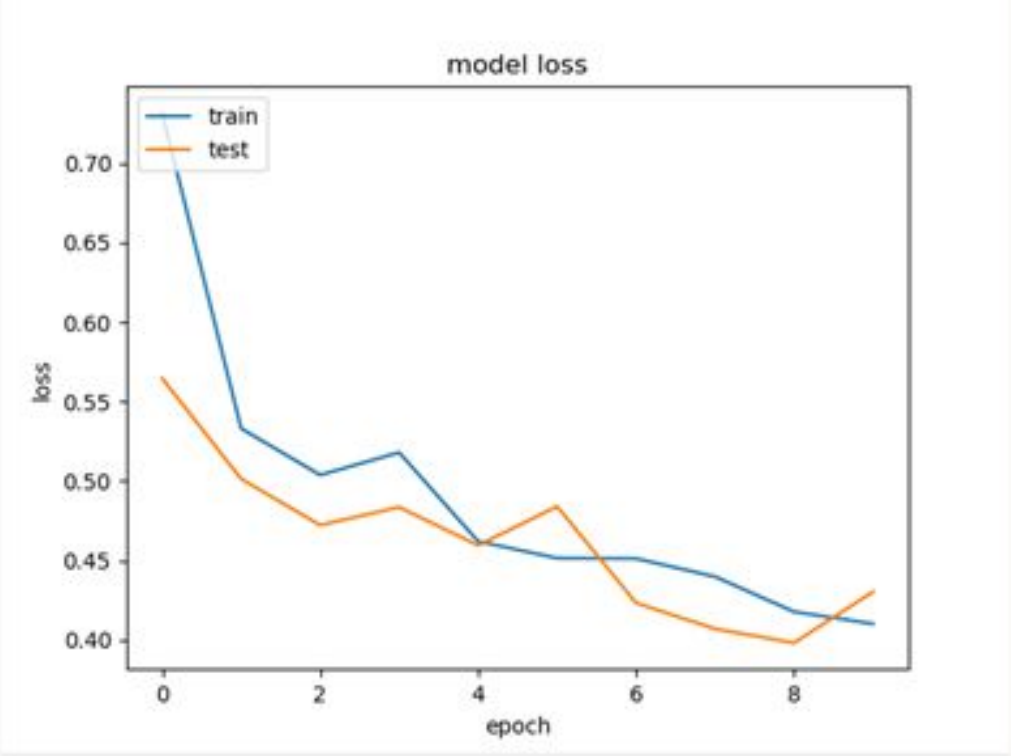

Figure 2d

The loss model with for 431 hidden nodes (indicated above) appears to be quite accurate on the surface. Looking at the table in the beginning of this section, this model scored the second highest training accuracy. However when looking at the validation accuracy which the model hadn't seen yet, the score was the lowest of all five. This would indicate that this model leans the closest to overfitting compared to all of the 5 models. Thus from the table, the optimally chosen parameter is 200 hidden nodes as it had the highest training accuracy without sacrificing validation accuracy by overfitting.

K-folds Cross Validation

Fold	Epochs	No. Hidden Nodes	CV Accuracy %
1	10	200	88.28%
2	10	200	88.49%
3	10	200	88.75%
4	10	200	88.63%
5	10	200	88.88%
Mean	88.61%		
Standard Deviation	0.25%		

Figure 2e

An accuracy score on prediction correctness measures the models skill at classification over a certain data set. Cross validation is meant to measure that models skill at generalisation. The main difference is the evaluation on the validation set will be on *unseen* data. This would mean a relatively high score of 88.61% validation accuracy has a significantly lower chance of overfitting compared to the plain error score. The reason for using the k partitions instead of 3 (train, test, validate) is because in the prior table only 10% of the data was used as unseen data to validate the model. In k-folds cross validation 100% of the data was used for validation after k iterations. As we can see the overall accuracy is slightly better than before but within the margin of error. Another interesting statistic is the standard deviation. A low standard deviation would indicate that any choice of training data yields similar results, it can be thought of as a measure of overall noise in the training data. After finally deciding on our model in step one and using cross validation to confirm it can generalise properly, we can finally evaluate the model on the 10,000 test images.

```
10000/10000 [==============================] - 0s 17us/step
MLP Error on test set: 12.23%
MLP Accuracy on test set: 87.77%
```

Figure 2d

Unsupervised Learning Contrast

While the overall process and end product of a decision tree and multi layer perceptron differs quite widely, they still can be applied to general data analysis problems, this means we can compare and contrast their applicability to a problem primarily on efficiency accuracy and readability. A decision tree can be considered to be generally quicker by virtue of the fact that the algorithm will choose which features it would like select and toss away features deemed useless, hence decreasing the amount of computation required. A MLP will typically consider all of the features as inputs unless specified otherwise in the pre-processing phase. While it is certainly not true for every case it can be considered a general rule of thumb that the decision tree is more appropriate for problems in which timed performance of the model is important. However the MLP appears to have a much more robust algorithm behind it, capable of modelling non linear functions with arbitrary patterns much better than a decision tree would. The overall accuracy of MLP is ultimately dependant on the size and relevance of the training data. Assuming ample training data, the MLP will likely be greater than or equal to the decision tree on accuracy, this is likely due to the networks ability to autonomously and incrementally improve itself over iterations. The decision tree is easier to read and interpret compared to the MLP. A MLP neural network becomes extremely complicated to understand as the number of inputs increases. Beyond this the hidden layer nodes are essentially a black box, making the model only useful to a computer who can utilise the given weights. The tree's decision making process is much more easy to reason with as the visualisation can be followed similar to a flowchart. So for cases where interpreting and/or applying what the model is doing over certain problem domains, the decision tree is much more human interpretable. It should also be noted that both algorithms seemed to be prone to overfitting if no measures were taken to detect or even acknowledge the possibility. Beyond this any other comparisons between the algorithms may require further empirical analysis over the specific problem.

A generalisation over what supervised learning sets out to achieve would be to consider input variables X and output variables Y and find appropriate mappings between the two. Two tasks where this can be applied are regression and classification problems. This can be used to perform mundane repetitive tasks thousands of times in order to try to find relations between two different trends, having to classify large amounts of input data which is unrealistic to do manually, to even general fields of artificial intelligence such as recognising images and text or mapping sensory input to an appropriate action.

The decision tree classifier can be used to classify the MINST fashion set and the multi layer perceptron can be used to predict the sale price band of future property sales. This is because they are both machine learning algorithms which have the geral workflow:

1. Pre-process the data set
2. Train a classifier
3. Evaluate the classifier

Without extending the classifier with other algorithms the decision tree may perform worse on the fashion data, but this will likely be dependant on the features chosen. Appropriate choice of the features set as a preprocessing set and the addition of some specific features unique to certain clothing items could lend to a higher accuracy with a better speed as well. Such an approach would risk overfitting however, and the tree would likely have to be pruned in order to avoid losing its generalised accuracy. The multi layer perceptron would have a high chance of overfitting the data after a certain amount of epochs if used on the property prices data set. This is because it's default behaviour would be to consider all of the features. K-folds cross validation or any validation could be used to mitigate this and after that the MPI has a good chance of becoming more accurate than the decision tree. This is because the neural network is more capable of modelling arbitrary non linear relations. Considering the size of these datasets another algorithm with potential use would be K nearest neighbours algorithm. This algorithm would require minimal feature parameter tuning on the melbourne house prices prediction compared to the prior two. It may also have a better chance of classifying the fashion data set seeing as the nearest neighbours may have a better ability to find similarities. It should also be noted that extensions to the decision tree include random forest and adaptive boosting meta algorithms. The random forest in particular should generally perform more accurately at the cost of training speed and interpretability. Boosting involves training additional classifiers to makeup for where the main classifier lacks, however this also increased the chance of overfitting. Overall the decision tree appears to have been a happy medium in terms of training speed, interpretability and accuracy. (Likely to be faster than the MLP and more accurate than Linear Discriminant Analysis). The MLP should be more accurate for the general case.