# 1. Decision Tree Learning

## 1.1. Introduction

In this section we will analyse a number of decision tree classifiers and determine how effective they are at categorising sale price bands. This will be done using a data set that represents Melbourne property prices and sales between 2016-2018. This data set is highly noisey and contains many samples with missing information. As such, we will also talk about how the data was prepared before being used.

Additionally, this section will compare and contrast all of the decision tree classifiers and recommend which model would be the best to use for this particular problem.

The decision tree classifiers will be evaluated by the following criterion:

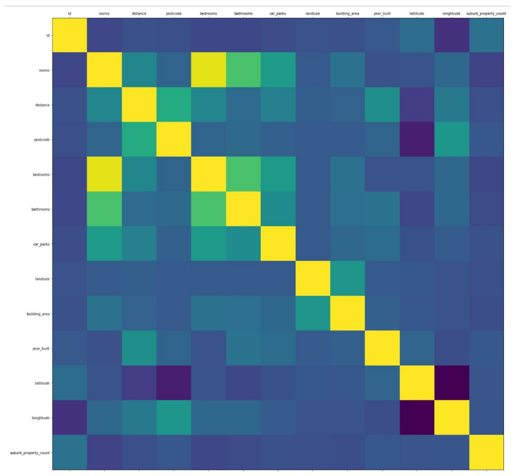
1. Precision: It is important to know for each price band, what percentage of the classifier's identifications were accurate. As if the precision is low, then the classifier is likely saying a large number of properties are in a price band which they are not (Lots of false positives).
2. Recall: It is important to know on average (across all sales pricing bands), what the percentage likelihood that the classifier will correctly identify a property in a given price band. If our model has a low recall then the classifier is failing to classify a significant amount of properties into the correct price band (Large amount of false negatives).
3. F1 Score: Due to this problem not having any reason to be weighted toward recall or precision, the F1 Score is the most important evaluation metric as it gives a harmonic mean of how the classifier performs across both metrics. It is worth noting that F1 is not always the most important metric for classification problems. For example, if this was a data set where we were testing to see if someone had cancer then the most important metric would be recall, as that classifier wants to minimize the amount of people it says don't have cancer when they actually do.
4. Fit to data set: We want the chosen classifier to be general enough to work effectively not only on the Melbourne data set, but also generalise well to other similar data sets (IE Sydney property prices). As such KFold Cross Validation was used to help contextualise all other scores within this metric. The KFold function was calibrated on 5 folds for all the models.

## 1.2. Data Preparation and Pre-Processing

The first action that was taken to prepare the data was to remove all of the independent variables from the dataset. Independent variables being variables that have no effect on the target field (in this case price_bands). To do this we looked at the correlation information for these fields and combined with expert knowledge removed the following fields:

1. Id, Address, Latitude and Longitude, were all removed as they are generally unique to the property being sold and as such will not help show any trend in the data. Please note, that the street from the property is on may correlate, however the data also contains the number on the street and as such is not as useful.

2. Date. Although it could potentially help find identify an abnormal price caused by a property boom, in testing this feature of the data set appeared to throw off the classifiers. Furthermore it did not appear to correlate in with the price_bands field based on the numbers.



Correlation Figure (Lighter colours means higher correlation)

The second action that was taken to prepare this data, was to find a way of handling all null values. For the null values an SKlearn Imputer was used to fill the null values with the relevant median value. This appeared to outperform the mean, when we used Label Encoders on the data.

Finally all properties that had 0 rooms, bedrooms or bathrooms were dropped from the data set as these data samples. This was done because a number of them were incorrectly labeled and as such negatively influenced the effectiveness of the classifiers

As mentioned earlier, after all the data had been prepared, it was encoded using the SKlearn preprocessing tool LabelEncoder. This was done as the SKlearn classifiers cannot handle string data and as such encoding like string with IDs enables the classifiers to effectively handle the data.

In this data-set, the field for houses between 0k and 200k was dropped entirely. The reason for this decision was done due to the amount of incorrect data in the set, where the true sale price for those homes did correlate with the price bracket the dataset assigned it. Furthermore a number of the sales that were correct were sold under abnormal circumstances (Quite a few where owner occupy was not possible). Due to the dataset being very small, it was easy to look up official sales records for these properties and as such this would be an example of expert knowledge playing a part in how the data was prepared. Furthermore due to the lack of support in the dataset relative to the size of the data set, the classifiers were not going to be able to be effectively trained to classify this price bracket. (Realestate.com.au, 2018).

Additionally, the field for houses with an unknown price band was also dropped. This was done because the goal of the classifier is to determine what price band an unknown input set of house data would (likely) fall into and as such having sample data where there was no true determined answer for what the price band for the sale was (accurate or not), appeared counter productive.

After the Dataset was prepared it was important to identify which parts of the dataset were relevant to classifying data to the correct price band. This was accomplished by using one of SKlearn's feature selection algorithms called SelectKBest. SelectKBest is a forward feature selection algorithm, this was important as we wanted to find the ten most relevant (correlated) features of the dataset and use them, rather than just pruning the least relevant features until reaching 15, it is worth noting that based on our testing 15 was the ideal number of features.

## 1.3. Models

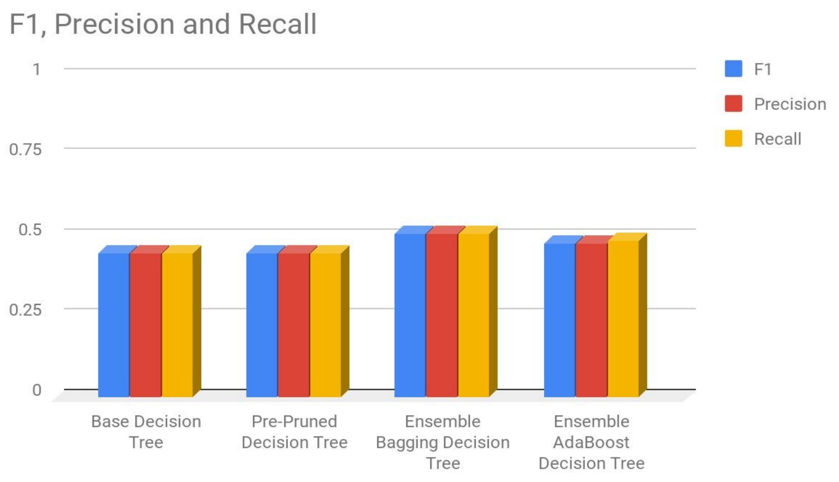
**Decision Tree Classifier:**

A Decision Tree Classifier repeatedly divides up the data-set into sub areas by identifying parts of the dataset (values of features) that can be sectioned and used to classify the datapoint into its relevant category. Decision Trees by their nature have a tendency to over fit a given data-set and as such doesn't generalise as well to noisier data-sets.

A number of decision tree classifiers were trained to and tested (using K-Fold Cross Validation), to effectively analyze how well the classifier performs on the data set. The following models were chosen:

1. A Baseline Decision Tree Classifier with no pre-pruning.
2. A Decision Tree Classifier that was limited to depth twenty (pre-pruning).
3. A Ensemble Bagging Classifier that used a pre-pruned Decision Tree as its base model with fifty estimators.
4. A Ensemble Boosting Classifier that used a pre-pruned Decision Tree as its base model with fifty estimators.

The performance of these models across the metrics of precision, recall and F1 are modeled below:



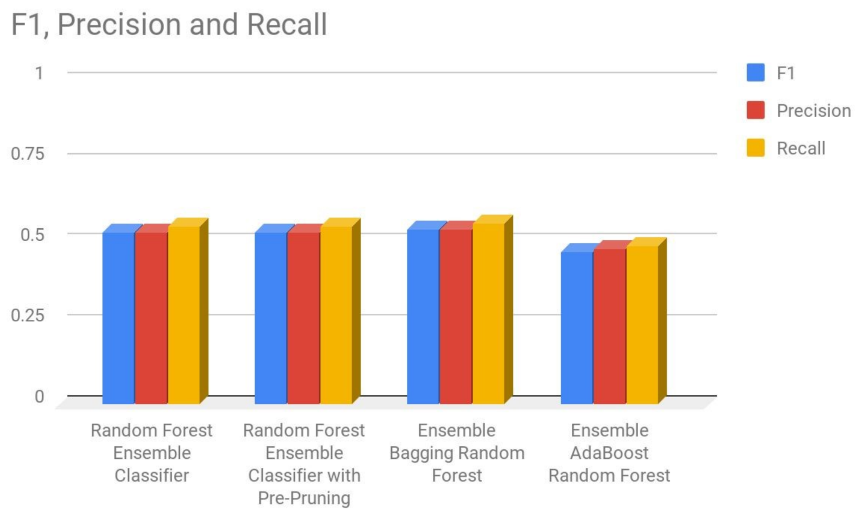
As can be seen in the figure above, the Ensemble Bagging Decision Tree performed best across all metrics.

**Random Forest Classifier:**
Random Forest Classifiers are ensemble classifiers that use multiple decision trees fitted to multiple sets of training data and then averages (Using the mode) predictions across trees using the most occurring classification for a given data point. Random Forests use of multiple decision trees which are fitted to different segments of the data helps to address the issue decision trees have with overfitting.

A number of Random Forest Ensemble Classifiers were trained to and tested (using K-Fold Cross Validation) with fifty estimators, to effectively analyze how well the classifier performs on the data set. The following models were chosen:

1. A Baseline Random Forest Classifier with no pre-pruning.
2. A Random Forest Classifier that was limited to depth twenty (pre-pruning).
3. A Ensemble Bagging Classifier that used a pre-pruned Random Forest as its base model with fifty estimators.
4. A Ensemble Boosting Classifier that used a pre-pruned Random Forest as its base model with fifty estimators.

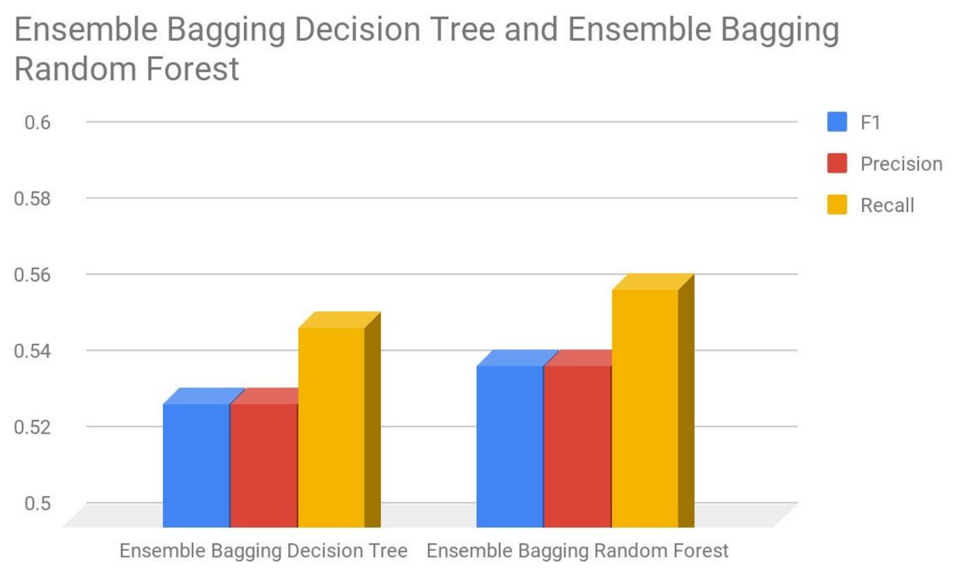
The performance of these model across the metrics of precision, recall and F1 are modeled below:



Though not super easy to determine from the figure above, the Ensemble Bagging Random Forest performed as good or better than every model across all metrics.

## 1.4. Conclusion

As can be seen of the models analysed the two best models are the Ensemble Bagging Random Forest and the Ensemble Bagging Decision Tree. As can be seen in the figure below, the Random Forest outperforms the Decision Tree across all metrics. This leads us to the conclusion that the best model for this data set is the Ensemble Bagging Random Forest Classifier. It is not surprising that the best version of both models used the Bagging meta-algorithm. This is because Bagging is designed to improve the accuracy and stability of the base model. Furthermore it also reduces variance, which would likely help performance on a more generalised data set or a cross validated data set.



This result illustrates that the design of Random Forest Classifiers to avoid overfitting and generalise better to a similar data-set (Scikit-learn.org, 2018). It is also worth noting that these scores are impacted by the amount of noise in the data, as though the dataset was prepared and pre-processed to improve the quality of the training data, there is still noise in the data that will be affecting the results.

# 2. Multi-Layer Perceptron

## 2.1 Introduction

In this section, we created a multi layer perceptron network for the purpose of image classification. The model was trained on the MNIST fashion dataset, containing 70,000 28 x 28 greyscale images of various clothing items from 10 different categories. Our primary objective was to observe the changes in the classification accuracy when certain parameters were modified. Specifically, the number of nodes in the hidden layer of the model and to identify an optimum number of nodes in order to maximize classification accuracy.

## 2.2.1 Designing a MLP in Python and Keras

Building a multi layer perceptron using Python and Keras is generally similar to building any other type of machine learning model (Such as linear regressor). However some additional data pre processing is required before it can be used to train a given model.

In the case of the dataset we were working with, the dataset itself was quite large and separated into many files that cannot be imported into a dataframe in the same manner as a regular data file would be. Instead, the data needed to be iterated through and placed into arrays. This process is much slower and took approximately three minutes for the 70,000 files in the MNIST fashion dataset. Next, each image must be reshaped into a one dimensional array before it can be used as an input to the model. The pixel data, which currently consists of values from 0 to 255, also needed to be normalized into values ranging from 0 to 1. This process was not entirely necessary to train a model however, we found in our testing that overall it did help improve the speed.

The next step we took was to build and train the model to classify images from the previously imported dataset. This process was very similar to other machine learning models we have trained wherein the model contains several modifiable parameters such as:

1. The number of hidden nodes/layers.
2. The activation functions
3. The number of epochs for training.

The configuration we decided to use for this task was the Sequential model as it is the most basic model available from Keras. We then only added one hidden layer into the model as experimenting with multiple hidden layers was not within the scope of the project. However, an interesting point for further research would be seeing how multiple hidden layers would alter the performance of a network, in comparison to simply adding more nodes to the same layer. For consistency, all parameters remained unaltered during our experiments except for the number of hidden nodes in the model.

To analyse the performance of the model, we measured the accuracy and loss on the training and validation data. This was recorded at the end of each training epoch and graphed to show the improvement of accuracy over time. We chose to use accuracy because it evaluates how many images have been correctly classified and as such shows how effective the classifier is at classifying images in the dataset. Loss was chosen as another evaluation metric we wanted to use because it evaluates the price paid for inaccuracies in the models predictions, therefore the lower the loss, the better the model was performing (Rosasco et al., 2004).

Visualizing the created model can be done with additional plugins such as ann-visualizer. However due to the large size of our network, containing over 700 input nodes and hundreds of hidden nodes, the visualizer was unable to create a true visual representation of our model, instead creating a much smaller version with only 10 input nodes.

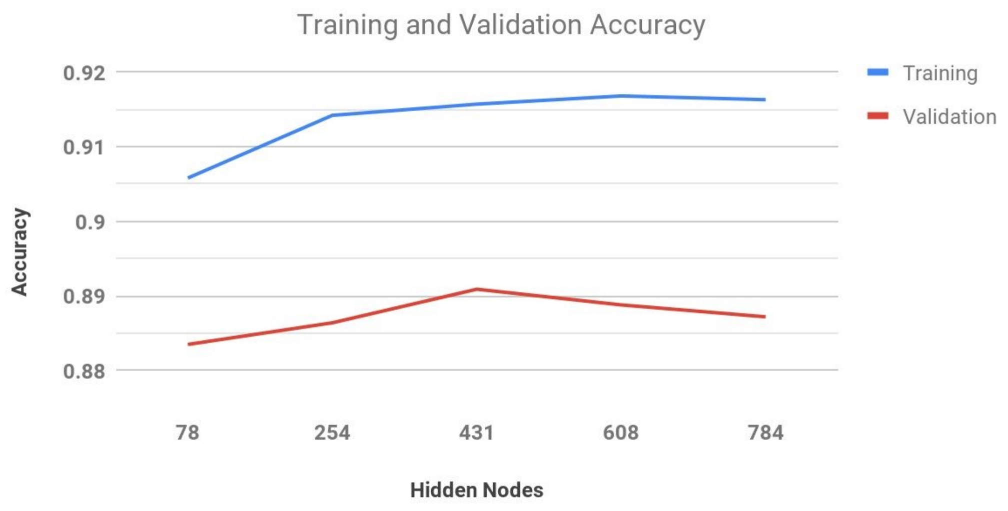
### 2.2.2 Experimenting with the Number of Hidden Nodes

For this task, we were required to experiment with changing the amount of hidden nodes in the model and to measure how changing this variable can affect performance. The project specification required that we test 5 different values ranging between 78 to 784. For our experiments, tested models with 78, 254, 431, 608 and 784 hidden nodes as they provided an even coverage of the desired range of values. It should be known that all hidden nodes were contained within a single layer, and the effects of distributing the nodes across several hidden layers was not tested.

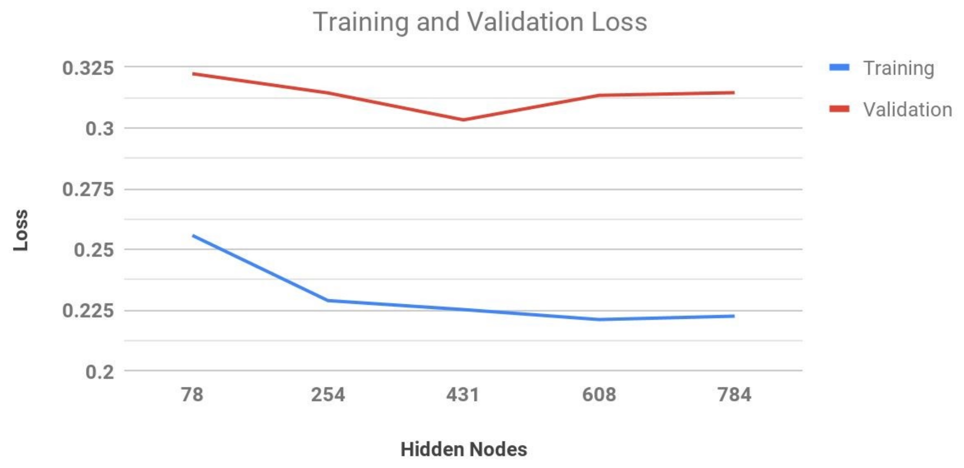
We chose to use the metrics accuracy and loss as our performance measurements across both the training data and the validation data. The time taken to train each model was also recorded. In order to ensure that the gathered measurements were accurate, every test was performed 5 times and the results were averaged. The results are presented in both tabular and graphical form below:

| Run No. | Epochs | No. of Hidden Nodes | Training Accuracy (%) | Validation Accuracy (%) |
|---|---|---|---|---|
| 1 | 10 | 78 | 90.58 | 88.35 |
| 2 | 10 | 254 | 91.42 | 88.64 |
| 3 | 10 | 431 | 91.57 | 89.09 |
| 4 | 10 | 608 | 91.68 | 88.88 |
| 5 | 10 | 784 | 91.63 | 88.72 |

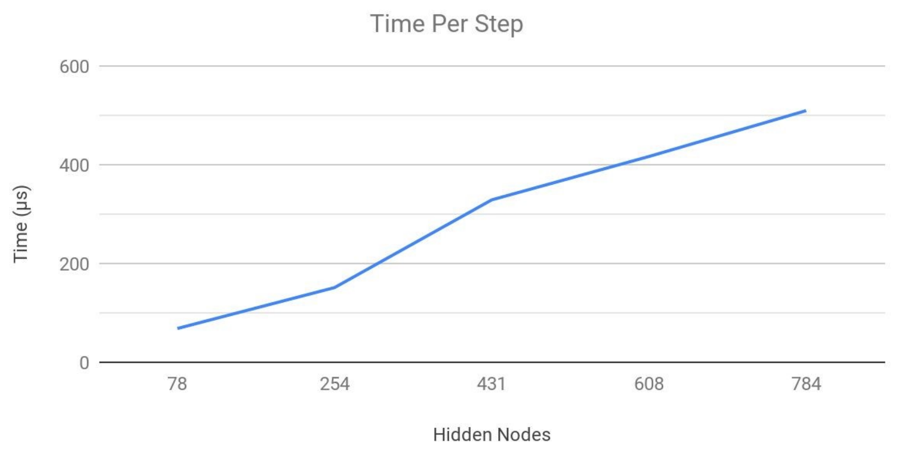
From the results gathered it can be seen that training accuracy generally tends to increase as the number of nodes grows. The rate of growth for the accuracy increased quickly at when there were less hidden nodes however as the amount of hidden nodes started to increase, the rate of growth in accuracy plateaued, with only a 0.21% increase in training accuracy between 254 and 784 nodes despite the size of the hidden layer increasing by three fold. From this we would hypothesise that as the number of hidden nodes approaches infinity, the rate of change for the accuracy in the training data set would approach zero.



The accuracy on the validation data however shows a different trend. The accuracy peaks at approximately 431 hidden nodes, and begins to slowly decrease after this point. This differs to the training dataset which continues to improve. This is likely because the model is overfitting on the training data and hence is less capable of correctly classifying the unseen validation data.



The graphical representation of the loss over the training and validation data shows very similar results to the accuracy. On the training set, loss shows the general trend of decreasing as the number of hidden nodes increases. However, on the validation set the loss reaches a minimum at approximately 431 nodes, and begins to increase after this point. This further supports our hypothesis that 431 is close to the optimum number of hidden nodes for this problem.



It is worth noting that the time taken to train the model increased linearly with the amount of nodes in the hidden layer. This gives further reason for programmers to be cautious when determining an appropriate size for a model, as a little bit of extra classification accuracy can come at the cost of a far longer training time. For example, our experiments showed that increasing the number of hidden nodes from 78 to 784 only improved validation accuracy by approximately 0.4%, at the cost of taking over five times longer to train. This only amounted to a few minutes for our model with 60,000 input images, but for large networks with millions of inputs and nodes the time cost could be much more substantial. As such, the size of the model should be carefully determined as the performance benefit may not be worth the added time cost.

## 2.2.3 Experimenting with K-Fold Cross Validation

In this task, we used K-Fold Cross Validation to further analyse the performance of our model. K-Fold Cross Validation works by splitting the training data up into k parts (folds), one fold is then designated as the validation set while the rest are used as training data for the model. This process is repeated k times until every fold has been used for validation. This technique is useful as it demonstrates the general performance of the model across different training data, allowing for easy identification of common issues such as overfitting. This can be invaluable when attempting to find the ideal parameters for building a model.

Cross Validation with five folds was used to evaluate the model we have identified to be optimal in the previous task. As identified in section 2.2, the model we will be using contains 431 hidden nodes. This model was chosen as it exhibited the highest accuracy and the least loss on the validation set.

| Fold | Epochs | No. Of Hidden Nodes | Cross-Validation Accuracy (%) |
|---|---|---|---|
| 1 | 10 | 431 | 88.90 |
| 2 | 10 | 431 | 89.33 |
| 3 | 10 | 431 | 88.78 |

| | | | |
|---|---|---|---|
| 4 | 10 | 431 | 89.08 |
| 5 | 10 | 431 | 89.11 |
| | | Mean | 89.04 |
| | | Standard Deviation | 0.19 |

The results show that the model performs quite consistently across the different data splits, having a standard deviation of only 0.19%. This suggests that our model has achieved a good balance of being complex enough to make accurate classifications whilst not overfitting to the training data.

Now that we determined the optimum number of hidden nodes for our network, we evaluated the model on the unseen test data set. This was done to analyse how well it would generalise as an image classifier on unseen data. Our model scored an average of 88.44% accuracy on the test data, which is consistent with our results from the cross validation process. However, when we ran our other models on the test data set, the performance actually improved. This shows that the test data set and the training data set must have been relatively similar as based on all our previous results, the model that was most likely to generalise best to an unseen data was the model with 431 hidden nodes.

This is an interesting result as it illustrates the value of having knowledge in the domain you want your model to be predicting. As if your training data set is going to be almost exactly the same as any data the model is asked to predict, then overfitting to the training data is not a huge problem. Whereas if there is more variance and noise in the dataset you are trying to predict overfitting will hurt the accuracy of the models predictions.

Ultimately, the power of image classification by a Multi-Layer Perceptron is demonstrated quite from this experiment. This is illustrated by the fact that with a relatively simple model, with very little setup and tuning, we were able to predict the what an image was with almost 90% accuracy.

# 3. Comparison

## 3.1. Use cases for Decision Trees and Multi Layer Perceptrons:

Both Decision Trees and Multi Layer Perceptrons can be effective tools for the classification of data. However, the models are definitely suited towards different types of data and therefore must be considered on a case to case basis depending on the data set that will be used.

Decision Trees are better for categorical data and do not tend to overfit the data set to the same degree as a Multi-Layer Perceptron. This means that for any problem with a noisy dataset which cannot be cleaned up a Decision Tree is likely the better choice. Furthermore, if you need to be able to explain how your model reached its conclusions than a Decision Tree is also the better choice.

A Multi-Layer Perceptron can model much more complex manners of categorising data. This means that it would likely outperform a decision tree on a data set that is a good representation of the data it will be predicting. Furthermore, it handles range data better than a decision tree as it can categorise data based on more functions than axis-parallel splits in values.

## 3.2. What are the takeaways from the two tasks that can be applied to future learning problems.

From these two tasks a number of trends became very clear that we believe apply to a large range of future machine learning tasks. The first one is that the quality of your input data really matters, even a slight improvement when cleaning up your input data can make a massive difference in the quality of your predictions.

The second trend that became clear was the value of domain knowledge. We found the more we understood both the data we were training our models on and the data we were trying to predict, the better our models became and the more we understood how to improve them further.

Finally the importance of choosing the correct model for a given problem is shown, as different models clearly have different strengths and weaknesses and therefore knowing which one to pick for a given problem is a huge advantage.

## 3.3 Comparison of Strengths and Limitations:

Decision Trees are faster once trained as they throw away features they don’t find to be useful in training, whereas Multi-Layer Perceptrons are slower and will use all provided features. In the case of our learning problems this difference in performance was noticeable when attempting to collect data from either model, for both tuning and evaluation purposes.

Decision Trees are also very easy to interpret, this makes it easy for a human to understand how the decision tree came to its conclusion. This is very different to Multi-Layer

Perceptrons, which have a much more complex structure and as such are much hard to interpret. This is one of the main practical issues with Multi-Layer Perceptrons as when the make a decision, it is very difficult to determine how they came to that conclusion. This was not a massive issue as it related to achieving our goals for the project, however we did find it much easier to understand what the Decision Tree was doing (and as such improve the model), whereas with the Multi-Layer Perceptron, it was very difficult to understand how it had reached its conclusions.

A Decision Tree can only model axis-parallel splits in the data to classify a data point, whereas a Multi-Layer Perceptron can model a much wider array of functions to classify a data point. This is both an advantage and a disadvantage for a Multi-Layer Perceptron, as though it can find more potential ways to split data, it can also cause the model to overfit the training data set.

## 3.4. Using a Decision Tree for image classification:

Whilst a Decision Tree would be able to fitted to the MNIST fashion data set, we would not expect it to perform as well. This is because there are a number of features in the dataset that the Multi-Layer Perceptron was able to model to use to separate images that were not modeled by axis-parallel splits and as such a Decision Tree would not be able to identify them. Furthermore there was a lot of distinct numerical data that the Decision Tree would not have found useful and as such new features would have needed to be constructed to represent data ranges for some of these field.

## 3.5. Using a Multi-Layer Perceptron for predicting property prices:

A Multi-Layer Perceptron could certainly be trained on this dataset. However, the property price data set had a lot of noise and misclassified samples and as such the the Multi-Layer perceptron would learn a number of features that do not generalise well to an unseen data set. As the data is very noisy, it is unlikely the Multi-Layer Perceptron would find the data in its initial (or even the pre-processed format used) to be very useful. A feature that may be interesting to construct is an estimate of the size of the property based on number of rooms, suburb and type of property, as the average size information is available online and may help the perceptron in determining the value of a property based on an estimation of its size.

## 3.6. Other algorithms to consider for these problems:

For the image classification problem a Deep Neural Network may be a better option than just a conventional Multi-Layer Perceptron as the ability to identify important features through convolution and pooling would certainly help in this instance.

For the Melbourne Property Prices dataset a Rule Learner could be used to try and predict the output of the data, however we don’t believe this would perform any better than the decision tree as the main issues with this task was the amount of noise in the dataset.

## Bibliography

School, A., College, S. and College, S. (2018). *Property Report for 5/25 Ridley Street, Albion VIC 3020*. [online] Domain.com.au. Available at: https://www.domain.com.au/property-profile/5-25-ridley-street-albion-vic-3020 [Accessed 28 Sep. 2018].

School, B., School, S. and School, C. (2018). *Property Report for 171 Moreland Road, Coburg VIC 3058*. [online] Domain.com.au. Available at: https://www.domain.com.au/property-profile/171-moreland-road-coburg-vic-3058 [Accessed 28 Sep. 2018].

Realestate.com.au. (2018). *8/132 Rupert Street, West Footscray, Vic 3012 - Property Details*. [online] Available at: https://www.realestate.com.au/sold/property-apartment-vic-west+footscray-125083430 [Accessed 28 Sep. 2018].

Realestate.com.au. (2018). *6/13-15 Pyne Street, Caulfield, Vic 3162 - Property Details*. [online] Available at: https://www.realestate.com.au/sold/property-villa-vic-caulfield-122892906 [Accessed 28 Sep. 2018].

Realestate.com.au. (2018). *202/51 Gordon Street, Footscray, Vic 3011 - Property Details*. [online] Available at: https://www.realestate.com.au/sold/property-apartment-vic-footscray-123273454 [Accessed 28 Sep. 2018].

Scikit-learn.org. (2018). *3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.0 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [Accessed 28 Sep. 2018].

GitHub. (2017). MNIST Fashion Dataset [Online]
Available at: https://github.com/zalandoresearch/fashion-mnist [Accessed 30 Sep 2018]

Rosasco, L., Vito, E., Caponnetto, A., Piana, M. and Verri, A. (2004). Are Loss Functions All the Same?. *Neural Computation*, 16(5), pp.1063-1076.