Predicting prices and trends in the virtual economy of World of Warcraft

Thor V.A.N. Olesen - *tvao@itu.dk*,
Dennis Thinh Tan Nguyen - *dttn@itu.dk*,
Daniel Rosenberg Hansen - *daro@itu.dk*

Abstract—This project investigates the possibility of applying data mining algorithms to predict price movement and commodity trends in a virtual economy system. Notably, the video game World of Warcraft is used as the primary platform since the video game has a relatively large player base and a working economic system. Trading between players happens through an auction house system where in-game commodities can be listed. This project will try to apply various data mining algorithms to derive and predict interesting information and trends about the market. The methods used are a combination of Artificial Neural Networks and Decision Trees for price movement classification and Apriori for mining frequent patterns on sold commodities to derive trends.

Index Terms—Artifical Neural Network, Decision Trees, Random Forest, Apriori, Machine Learning, EconomicPrediction

1 Introduction

World of Warcraft (WoW) is a massively multiplayer online role-playing game (MMORPG) developed and released by Blizzard Entertainment. WoW remains one of the most subscribed MMORPG with more than five million subscribers worldwide[1]. The players interact in a dynamic fantasy environment where each can participates in a virtual economic system that centers around a global in-game auction house. The simplified virtual economics in WoW may share many attributes of a real-life economic system and may thus provide an interesting and simple platform for deriving knowledge that can help clarify how more non-virtual complex economics systems works.

For this purpose, data mining may be applied on the virtual economy in WoW to discover new knowledge that may be used to reveal trends, patterns, and help predict price movements in the market.

1.1 Economy of World of Warcraft

The majority of in-game transactions made by the players happens in the auction house of WoW. The Players can list auctions of items they have in possession, where one auction may last between eight, twelve and twenty-four hours. The currency in WoW consists of coins split into three subsets of Gold, Silver, and Cobber coins, where Gold has most value and cobber has the least value. That is, for one-hundred Cobber can be converted to one silver and for one-hundred Silver can be converted to one Gold.

The players may then set up an auction with the minimum bid price and a buyout price. A transaction is complete once the time runs out of an auction that has a bidding player or if a player instantly buys the item with the buyout price. Based on the author's personal experiences of the game, it seems that the tendency is players are mostly buying out items instead of bidding. Thus many transactions occur more frequently in a single day than if players were bidding. This is especially true for common trade goods which are items that players need to progress in the game.

Based on that observation, common trade goods may have the largest number of listed auctions amount as well as transactions. Thus it is interesting to investigate this part of the market with data mining algorithms.

1.2 Research Question

The following research question has been investigated throughout the project:

Research Question: How may one reveal farming trends and predict price movements in the virtual economy of World of Warcraft using frequent pattern mining and supervised classification algorithms?

Sub-questions to answer the main question:

- 1) How may one use the Apriori algorithm to reveal popular farming[1] areas players use to gather commodities in World of Warcraft?
- 2) How may one use an artificial neural network to predict price movements of common trade goods in World of War craft?
- 3) How does the neural network perform compared to decision trees and random forests for classification?

1. A repetitive task in a game where one collects a bunch of certain commodities in order to gain power or currency.

2 Technology Used

The project implementation has been developed using Scala, SBT (i.e. Scala Build Tool), and Apache Spark. Apache Spark is a great cluster-computing framework used for big data processing that enables fast in-memory data processing, distributed parallel computing, and lazy evaluation. Arguably, using Spark may be deemed overkill for the purpose of this data mining application that runs on a single test machine. Nonetheless, it has proven to help immensely reduce training times in our classification algorithms (e.g. neural network), make the data analysis less time consuming, and allows us to solve preprocessing and machine learning problems at much greater scale in the future.

By comparison, Java and Python were also considered as candidates for the purposes of the project. Namely, Java and Python are very easy to develop and read the code. Despite being familiar with Java, we chose not to use it due to it being very verbose, and having a long time to market. Namely, Scala and Python enables rapid prototyping by providing a REPL interpreter (i.e. Read-Evaluate-Print Loop) that makes it possible to take a rapid, iterative approach to analyzing and modeling large-scale datasets using Spark. Thus, the team deemed Scala to be the right tooling to boost our productivity and help us test out ideas together in a live coding context while having the REPL audit our code. Scala and Spark also teaches you useful abstractions in functional programming that may be used to leverage scalable distributed data analysis and increase parallelism.

Nonetheless, the group did experience a steeper learning curve in Scala. Specifically, if you want to get anything significant done in Scala you are forced to absorb a whole host of programming language concepts in order to become a comfortable professional. On the other hand, Python might have provided a a simpler faster coding process than using a compiled language. Also, Python has a powerful set of packages that may be used for a wide range of data science needs including *NumPy*, *Pandas*, *Scikit*, and *Tensorflow*.

However, the group had no prior experience with Python but was already familiar with Scala and Spark from a previous course and deemed Scala a better match for modern data processing, since it makes it very easy to leverage parallel processing while keeping the code succinct, expressive, and free of boilerplate. The benefits of functional programming (with focus on concurrency) include immutability, pure functions, refential transparency, lazy evaluation and composability. Most importantly, instead of using shared state that needs to be synchronized in a multithreaded environment, one may use immutable data that can be safely transferred between threads leading to less bugs.

Arguably, the choice between Scala and Python boils down to a trade off between complexity and efficiency. On one hand, Python seems to offer a simple programming model that is easy to learn and understand with great support for big data processing. On the other hand, Scala and Spark provides efficient distributed processing, rapid prototyping, and succinct code at the cost of complexity. In the end, the choice of technology is driven by what people are already comfortable with.

3 The Dataset

The auction house data set has been made available in a JSON format by Blizzard through their Game Data API that publishes data related to the game itself. The auction API provides rolling batches of data about current auctions. By implication, this means that Blizzard does not currently store historical auction house data but only the most current daily data. Thus, a PHP script has been used to gather the auction data twice daily on a server hosted by one of the group members. In total, 6 million auction house records have been aggregated during the project that span over 64 days with approximately 100.000 daily auctions. Arguably, one might assume this few daily samples lead to a prediction model that is more variant or overfit, which will be elaborated in the algorithm section.

As shown below, the auction house record contains 12 feature attributes by default providing information about the auction, item being sold, owner behind the auction, the realm on which the item auction is listed, the bid price, the buyout price, how many items are being sold along with some other data. Besides this, the auctions have been enriched by us with a timestamp in order to do analysis of auctions over time and distinguish between week days and weekends:

```json
{
 "date":"2018-04-09 06:24:15",
 "auc":1659658801,
 "item":2592,"owner":"Nagios",
 "ownerRealm":"Exodar",
 "bid":6327,"buyout":7109,"quantity":1,
 "timeLeft":"VERY_LONG","rand":0,
 "seed":0,"context":0
}
```

Listing 1: Auction JSON Schema

This World of Warcraft auction house data has been chosen, since the virtual economy in WoW is known to share many attributes with real economies of our world. Thus, an understanding of this simplified economy may advance our knowledge of more complicated economic prediction systems in real life. This may be perceived analogous to predicting stock markets motivated not so much by a desire for material gain, but from the challenge of doing a better job in predicting the daily variations of the market than professionals.

3.1 Preprocessing

Many of the data mining algorithms are prone to noise and outliers. As a result, the most time-consuming part in this project has been data preprocessing. Significant preprocessing has been applied to ensure the accuracy of the neural network, which included removal of outliers in the data and the hand selection of specific commodities to do price movement predictions on in our model. Tasks involved in preprocessing include data cleaning, data integration, data transformation, and data reduction amongst others.

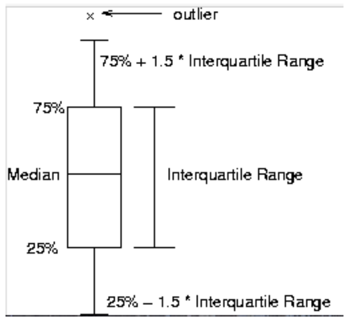
3.1.1 Data Reduction

Firstly, the auction data set has been reduced by removing any irrelevant information based on our research question. Namely, the dimensionality reduction has been applied in which the following attributes have been disregarded: *server name, server id, item upgrades, dungeon id, pet information, and seed used to calculate item upgrades* (see B.1).

3.1.2 Data Cleaning

In terms of data cleaning, the auction data is already provided in a consistent JSON format so cleaning the data did not involve filling in any missing values. However, removing outliers, resolving noisy data and inconsistencies was performed. Firstly, auctions without a buyout price have been removed. Secondly, the quantity of each auction has been standardized to be 1 by default (see B.2). Finally, outliers have been detected and removed to reduce any potential skewness in the predictive models (see B.3).

To do so, interquartiles are used to measure the central tendency and spread of auctions based on their mean buyout price and help identify outliers. Alternatively, one might have measured the median that is more robust towards outliers but this was very computationally expensive and did not seemingly change the outcome in our test. The Q1 lower quantile (i.e. cut point dividing distribution of data) is set to the 25% auctions with lowest buyout price and Q3 upper quantile is set to the 25% auctions with highest buyout price. In this regard, the interquartile range constitute the difference between upper and lower quartiles (IQR = Q3 - Q1) and any auction data points that are 1.5 below or above these quantiles are deemed outliers:


outlier
75% + 1.5 * Interquartile Range
75%
Median
Interquartile Range
25%
25% – 1.5 * Interquartile Range

Initially, we found the quantiles based on the minimum and maximum buyout price for each item but later realized that they were sometimes too distant from the central tendency so we changed it to the maximum and minimum mean buyout values. Arguably, this realization had been made possible at an earlier stage if we had used visualization in addition to our numerical measurements to identify the outliers in the data (see code in B.3).

Nonetheless, one should proceed with auction when removing data points, as this may also bias the distribution of data and make it appear normal. However, having used the auction house in WoW extensively, we have experienced that it is very normal that a certain percentage of items are sold at an extreme buyout price. In order to further validate this, we cross referenced some items in the game to see whether they were being sold at the extreme 5% tail buyout prices. In general, the with extreme buyout prices are deemed outliers, since they are rarely sold in practice. Thus, they do not seemingly contribute to the price movement model in this project and removing them may help improving the prediction accuracy. However, no data points have been found that constitute a measurement error or a value that has been recorded incorrectly, thanks to the standardized format provided by Blizzard.

3.1.3 Data Transformation

In WoW, money is some amount of copper, silver, and gold pieces. Further, 100 copper pieces constitute 1 silver, 100 silver is the same as 1 gold, so 10.000 copper is equal to 1 gold. By default, the auction bid and buyout prices are given in copper, which may range from 0 to millions. Thus, it is important to normalize the data to fit between the same standardized range, which is 0 and 1 in this project. Initially, we observed that having a huge difference between the maximum and minimum copper numerical values resulted in unwanted biases and longer training times. Instead, we used minmax normalization to scaled features down between 0 and 1 using the formula: **x to y = (x-min) / (max-min)** . Also, we check that max - min is equal to zero in which we normalize with **1 / totalItems** (see B.4).

Finally, we transform the auctions by enriching it with summary information (e.g. mean). Again, we did not use the median due to it being too computationally expensive and disregarded the standard deviation. Specifically, we deemed it unnecessary to use the standard deviation, since outliers should have been removed by using the interquartile range before computing mean buyout prices. Otherwise, it is a good idea to use standard deviation, since outliers have a significant impact on the mean.

Thus, auctions grouped by item are averaged using mean, which results in a reduced representation of the data that extract and predict the same outcome. Further, we enrich the auctions with information related to whether the auction is being sold on a weekend. Finally, we label the auctions with a categorical price movement value that is either **Up, Down, or Stationary**. This is done based on a price movement threshold that is currently set to 2%, meaning auctions that have increased by more than 2% in buyout price in a day are classified as going Up and auctions that have decreased by more than 2% in buyout price in a day are classified as going Down. Ultimately, all of these transformations are performed to yield a better prediction accuracy in our models. For example, measuring the central tendency by using summary statistics (in this case the mean average) should provide a better estimation of the daily price movement of each auction item.

3.2 Data Visualization and Measurement

In order to perform any data preprocessing and analysis, one may first do some simple statistical measures and visualizations. For this purpose, we have tried to visualize the distribution in price and price movement labels both across specific commodities as well as all auctions based on averaged daily mean buyout and bid prices.

Initially, 7.030.017 total auctions are loaded in and collapsed into 839950 auctions grouped by item and averaged by day. By example, 18.511 auctions are registered with the popular 'linen' cloth commodity auction price averaged by day. The following table shows its distribution of price movements averaged by day with a total of 64 days:

Linen Cloth Daily Price Movement Distribution	
Price Movement	Count
Up	34
Down	31

As can be seen, the data is heavily skewed towards price movements being either Up or Down but never Stationary. Presumably, this might be due to outliers skewing the averaged auction prices to stray from the central tendency. In order to validate this hypothesis, one may look at the price distribution after preprocessing when the extreme minimum and maximum outliers have been removed.

Initially, we naively removed the 5% highest and 5% lowest buyout auctions for each item In this case, a total of 983.705 auctions were removed with a total of 6.046.312 auctions left being averaged by commodity type down to 767.908 auctions. In terms of linen cloth, this resulted in 16.572 auctions with the following buyout price distribution and daily price movement distribution:

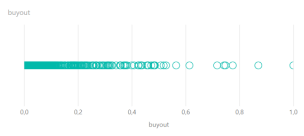


Figure 1: Linen Cloth Price Distribution Before

Linen Cloth Daily Price Movement Distribution	
Price Movement	Count
Up	32
Down	28
Stationary	5

Many outliers are still present in the maximum buyout range but the increased amount of stationary price movement labels suggest that outliers previously skewed the mean buyout price in the auction data. Arguably, before preprocessing no auctions move by more than 2% in price (i.e. stationary) due to high minimum or maximum extreme values skewing the average buyout price. In order to remove the remaining outliers, we used interquartile range instead. Ultimately, this resulted in having more outliers and less unintentional auctions being removed with 6.362.169 auctions left. As a result, the distribution of prices has improved dramatically when looking at the popular 'linen cloth' commodity:

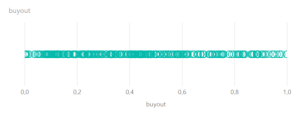


Figure 2: Linen Cloth Price Distribution W/O Outliers

Finally, one may use visualization to help validate outlier removal by easier interpretation or use clustering to identify groups of prices before removing potential outliers. Otherwise, one may end up confirming a known hypothesis by biasing the data unintentionally. In our case, we used a scatter plot to visualize the buyout and bid price distribution. Interestingly, the auctions are mainly scattered in the maximum buyout range shown in the lower-bottom right corner. This might be because players always buy the cheapest auctions but rarely buy expensive auctions.

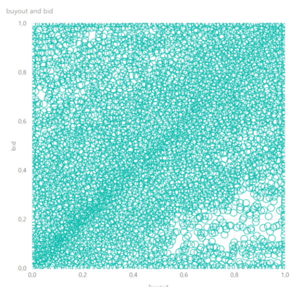


Figure 3: Raw Buyout (X) and Bid Price (Y) Distribution

The scatter plot allowed us to realize that the data was not uniformly distributed and identify outliers visually. As a result, we used interquartile range to make the data less skewed with a lower central tendency that aligns better with our data analysis based on the assumption that players always buy the cheapest commodities but rarely buys the over expensive commodities:

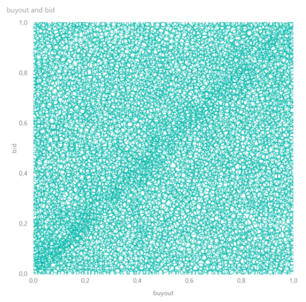


Figure 4: Price Distribution after outlier removal

As can be seen, the distribution of price now aligns better towards the central tendency in the auction data set, which should help improve the accuracy of our prediction models.

4 Applied Algorithms

In today's era, a proliferation of large data is present and today most companies have data coming in with high volume, velocity, and variety (i.e. 3 Vs of Big Data). With an enormous amount of data, it is increasingly important to develop powerful tools for data analysis and mining interesting knowledge from it. Data mining is the process of inferring knowledge from such huge data. In this regard, the following section describes the classification and frequency pattern mining algorithms used to discover properties in the auction house data.

4.1 Artificial Neural Network - Classification

An artificial neural network with a multi-layer feedforward topology has been used to predict the price movement of auctions in. This is done in Spark ML using its Multilayer perceptron classifier (MLPC) that is a classifier based on the feed forward neural network. MLPC consists of multiple layers of nodes where each layer is fully connected to the next layer in the network. The initial neural network was designed to have eight input nodes, five nodes in the hidden layers, and three distinct output nodes. The inputs were mean bid price, mean buyout price, quantity, min buyout price, min bid price, isMorning, isEvening, and isWeekend. The outputs were the expected buyout price movement labeled Up, Down or Stationary. as shown below:

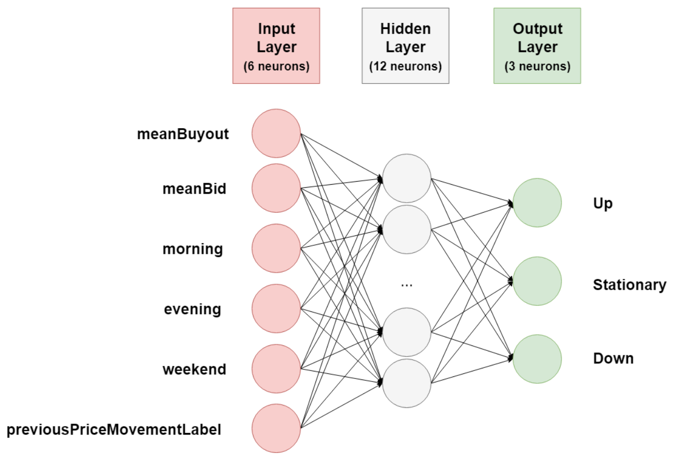


Figure 5: Artificial Neural Network Configuration

The maximum number of iterations is set to 100 and the step size (i.e. learning rate) and convergence tolerance is set to 1E-6 by default. Adjusting the tolerance to a lower value leads to a higher accuracy with the cost of more iterations. MLPC employs backpropagation for learning the model and the intermediate layers use a sigmoid (logistic) loss function. In terms of tuning, one may adjust parameters such as the learning rate, weight initialization, activation function in hidden layers, the amount of hidden layers, and number of neurons in the hidden layer. In our case, the final neural network is configured to have six input nodes, twelve hidden input nodes and three output nodes. As a result, the accuracy was improved (see B.6 for code. It was found that increasing the number of hidden nodes from five to twelve improved performance substantially, but increases beyond this were negligible.

4.2 Apriori - Frequent Pattern Mining

Frequency pattern mining to help reveal trends in what popular items individuals sell together in WoW. This may help discover what areas in the game are used for gathering of such items. In general, finding such patterns using frequent pattern mining is at the essence of data mining, since it tries to take a set of data and apply statistical methods to find interesting and previously unknown patterns within the auction data set. In this regard, we are not trying to classify auctions or perform clustering but simply want to learn patterns of item subsets which emerge frequently in across auction traders. This is analogous to market basket analysis that attempts to identify associations or patterns between various items that have been chosen or sold by a particular individual and placed in the market basket (i.e. auction house). In order to represent patterns, one may use association rules like the single example given below: $Starflower, Frostweed \ => \ Fireweed[support = 5\%, confidence = 96\%]$
In order to determine whether such rule is interesting or insightful, one may use the support and confidence metrics. Namely, support is a measure of frequency and confidence is a measure of correlative frequency. In the above example, the support of 5% indicates that the two items on the left side were purchaced in 5% of all auction house transactions. Further, the confidence of 96% indicates that this was the 96% of those who sold the two items also sold Fireweed. In our application, association rules are generated within a minimum threshold of 1% (i.e. 70.000 auctions) support and 60% confidence by default.

Apriori is used to enable frequent pattern mining on items sold by individuals in WoW. From an initial dataset of auctions, Apriori computes a list of candidate itemsets. Based on the candidate itemsets generated, an itemset is determined to be frequent if the number of transactions that it appears in is greater than the support value. Thus, Apriori computes all item rules that meet the minimum support and confidence values. Interestingly, the generation of new rules does not rely on previous rules so one may parallelize the computation.

In our Apriori implementation, we accumulate frequent itemsets recursively using an efficient accumulator pattern, meaning we never hold on to previous item sets. Hence, the implementation is tail recursively optimized, meaning it does not build up a large stack of calls to itself and waste memory by holding on to previous candidate sets. Due to this, we did not experience a huge memory consumption and were able to run Apriori on all 7.000.000 million auctions in less than a minute despite Apriori being known to have a slow candidate generation. Otherwise, one might have used the FP-growth algorithm that does not read the transactions for every iteration. However, one can easily parallelize Apriori as opposed to FP-growth where data is more inter dependent [5].

4.3 Decision Tree - Classification

For this project, it is interesting to compare the performance of the artificial neural network with another supervised

classification algorithms such as the decision trees. Spark supports decision trees for multiclass classification using both continuous and categorical features.
The features used in the prediction model consist of five categorical features and two continuous features. That is, four categorical values denoting whether it is *morning, evening, weekend or if tomorrow is weekend* and one multi-categorical feature *previousPriceMovementLabel* that shows the previous price movement prediction. Further, the two continuous feature are *mean bid* and *mean buyout*. The categorical features denote the current state of time, which is interesting since the market activity is based on the current time of day but also whether it is weekend or not. For example, one may assume that player activity is higher during weekends than on weekdays, thus increasing market activity. The two continuous features, mean bid and mean buyout, denote the current state of a given item price.

Both features are essential since the decision tree may need to consider a split point based on e.g. whether a given item has reached a peak and thus predict whether the price will go down, up or remain stationary (see B.7 for model implementation). In order to come up with a split point, one may either use *Gini* or *entropy* as attribute selection measures used to find the highest information gain (i.e. the most homogeneous branches with most similar class labels). Both measures yielded the same results in our case so the *Gini* impurity measure was used.

The construction of the decision tree model stops if any of the following conditions are met.

- Node is equal to *Max Depth*
- No split candidate satisfy min default information gain

Based on these stopping rules, the max depth is customizable. Namely, a very deep tree tends to overfit so one may decrease the max depth to prevent overfitting (five as default). The depth is selected based on trial and error, which is further elaborated in section 5.2 (see B.8 for implementation).

4.4 Random Forest - Classification

As mentioned, decision trees are prone to overfitting, especially in deep trees due to the amount of specificity leading to smaller sample of events that meet previous assumptions. Instead, one may use a random forest that is more robust against errors and limit overfitting by using a collection of decision trees whose results are aggregated into one final result [4].
In this regard, Spark provides a random forest implementation with multiclass classification using continuous and categorical features. By comparison, the features are the same used in the decision tree. In general, we strived to use supervised classification methods that support the same input model to enable later comparison of their price movement prediction accuracy.
In practice, Spark creates an ensemble of decision trees and train each separately. During the training process, Spark introduces randomness by subsampling the original dataset for each tree and by considering different random subsets of features to split at each tree node[3]. The random forest implementation is nearly identical to the decision tree except for the number of trees that is parameterized in the random forest and set to 64, since any number above or below this did not provide again gain in our test results.

5 Validation and Results

The following section describes how well our mining models perform against test data using 10-fold cross validation and box plots to measure the variance in accuracy and quality of our price movement prediction models. Also, we have tried to establish a relationship between association rules found on auction items by the Apriori algorithm and the areas in which these items may be found.

5.1 Artificial Neural Network results

In general, a high accuracy was found on popular commodities that were cheap goods traded by many users in the game. On the other hand, commodities less traded did not perform quite as well in our price movement predictions. For example, the popular 'linen cloth' commodity yielded a 100% accuracy in our best prediction model and a generally high accuracy even on non-fit data. One the other hand, a rare item like the Pet cage fluctuates a lot more in accuracy, which stems because it is not a heavily traded commodity with perfect competition as. This is shown in the following boxplot that displays the variance in accuracy recorded in a 10-fold cross validation of the ANN prediction model (see actual results in D.1). In this case, ANN was used to predict the price movement of auctions selling three popular and highly volatile items:

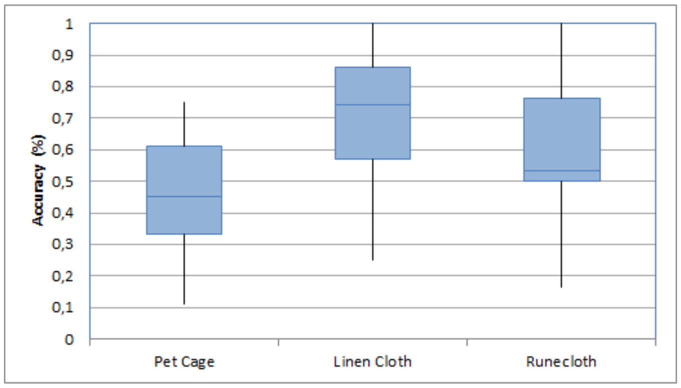


Figure 6: ANN Box Plot Accuracy Variance

Interestingly, the boxplot visualization of our cross validation result validates our hypothesis that it is easier to predict price movements on general commodities than rare items. Most importantly, the data set contains 7 million auctions but these are only distributed across 64 daily data points. Despite this lack of historical data points, our neural network prediction model manages to provide rather good results. For example, 'linen cloth' has a median accuracy of 75% on average, meaning the prediction model has a higher accuracy than this on half of the training examples. Nonetheless, this is biased by the fact that our test set is currently very small (e.g. 10% of 60 daily auction data points only constitute 6 test training examples to predict).

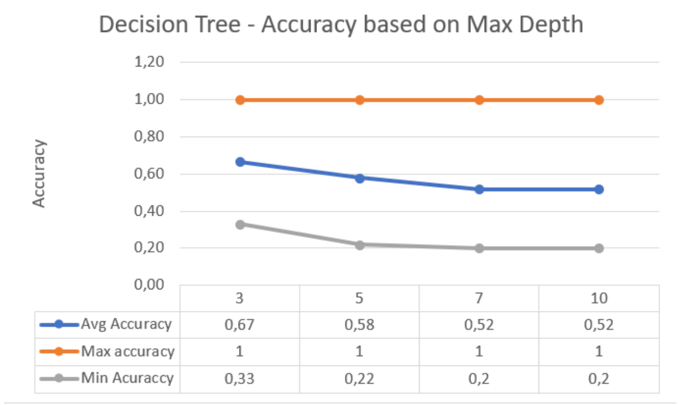
In general, using a 10-fold cross validation often resulted in extreme accuracies between 10% and 100%. In this regard, it was found that the prediction model provided some very deviant prediction results depending on the distribution of class labels in the training and test set. In order to improve this, we increased the convergence tolerance threshold of the neural network from 1E-10 up to 1E-6. Consequently, the neural network training time is reduced to avoid overfitting. In general, a smaller convergence tolerance of iterations will lead to a higher accuracy with cost of more iterations. However, our prediction model became almost equally accurate but less variant by increasing this parameter a little (see A.4, figure 19).

Finally, in order to mitigate the effects of a limited test sample, we tried using 5-fold cross validation in which a 20% random test sample is used. As a result, the variance in accuracy decreased dramatically at the cost of a lower general accuracy. Arguably, this might still be perceived a good result, since the prediction model is no longer affected by strong deviations in the distribution of samples in the test set training examples. Also, the distribution of accuracy converges towards the actual accuracy of the prediction model when comparing the 10-fold cross validation boxplot with the 5-fold version (see A.4, figure 20.

As a final note, we would have liked more time to investigate a recurrent neural network (i.e. RNN) using Tensorflow to predict auction house prices. In brief, an RNN is a neural network with self-loop in its hidden layer(s), which enables it to use the previous state of hidden neuron(s) to learn the current state given the new input. Thus, RNN is good at processing sequential time series data. In our case, one might have used a time series of N daily auctions to predict the N+1th time seried auctions. In general, using this would have enabled us to do forecasting on an auction time series model where past price patterns can be used to predict future price behavior.

5.2 Decision Tree results

The performance of the decision tree was tested based on the depth of the tree. In this regard, the following max depths used is *3, 5, 6 and 10*. 10-fold Cross-validation was used to validate the model and the commodity that was used for testing is *Linen Cloth*. The accuracy based on max depth size can be seen in figure 7 It can be seen that all test cases have a max accuracy of 100 percent, which is clearly because of overfitting for that given partition. Taking a look at the sample with a decision tree of max depth 5, the accuracy was 100 percent as seen at the sample at appendix D.2.
It is clear that resulting accuracy is due to lack of representative instance in the test data such that the other labels were tested as well. Based on the above sample, the random sub sampling of test data constitutes of only price-movement-labels "down". Both were predicted correctly and thus the accuracy of test was one 100 percent. However, this may not have been the case if other labels were included in the sample as seen on the sample at appendix D.3



	3	5	7	10
Avg Accuracy	0,67	0,58	0,52	0,52
Max accuracy	1	1	1	1
Min Acuraccy	0,33	0,22	0,2	0,2

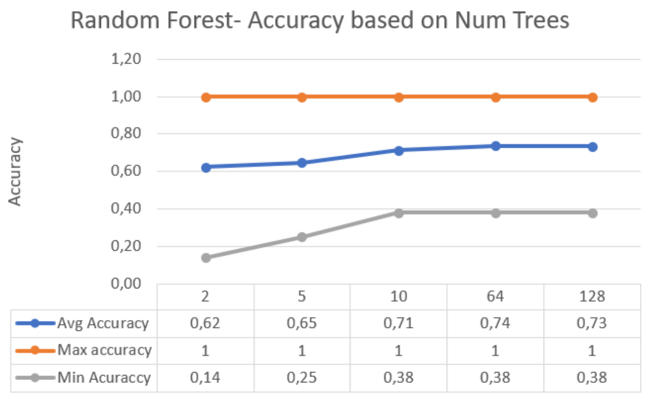
Figure 7: Decision Tree Accuracy vs Max-Depth

The resulting accuracy is 75 percent which is more realistic with a diverse test sample. Yet the model is fitted towards "up" and "down" since the distribution is skewed towards them. The label "Stationary" was underrepresented during training and thus when testing with a data point that was labeled stationary, the model was unable to correctly classify it. To avoid this issue one may have created test data and training data with a balanced distribution of labels.

Regardless, it can be observed in the graph that the accuracy decreases as one increases the max-depth. This is expected as a deeper tree tend to overfit the training data as a deeper tree is more complex and may contain branches that reflects anomalies in the training data[4]. Thus pre-prunning the tree with a low max depth results a shallower tree and may hence yield better results compared to a deep decision tree. Finally based on the test results the Decision Tree performed fairly well with an average accuracy of more than 50 percent on all depth.

5.3 Random forest Network results

The performance of the Random Forest was tested based on the number of Trees in the forest. In this regard, the following number of trees used is *2, 5, 10, 64, 128* and all max depth were set to 5. 10-fold Cross-validation was used to validate the model, and the commodity that was used for testing is *Linen Cloth*. The accuracy based on the number of trees can be seen in figure 8 Similar to the results in the Decision Tree all test cases yielded with max accuracy of 100 percent. This is due to the aforementioned data sampling issue with 10-k folding and label distribution. However what is interesting here is the fact that when one increases the number of trees in the forest, the accuracy increases as well. Yet the accuracy difference starts to flatten out between 10, 64 and 128 trees compared to 2 and 10 trees where the accuracy increase was almost 10 percent. In general, it may thus be argued that an increased number of trees may result in better accuracy due to the final classification yields from an aggregation of all the trees results. Hence many trees may thus reinforce the confidence of a given classification compared to few trees. Finally, compared to the decision tree of depth 5, it is clear that the average accuracy is greater. For example, the accuracy is 70% in a random forest with 10 trees, whereas the accuracy of the decision tree is only 58%.



	2	5	10	64	128
Avg Accuracy	0,62	0,65	0,71	0,74	0,73
Max accuracy	1	1	1	1	1
Min Acuraccy	0,14	0,25	0,38	0,38	0,38

Figure 8: Random Forest accuracy vs number of trees

5.3.1 *Comparison: ANN vs Decision Trees*

The following graph shows the variance in accuracy when performing a k-fold cross validation from 1 to 10 on each classifier used during the project:

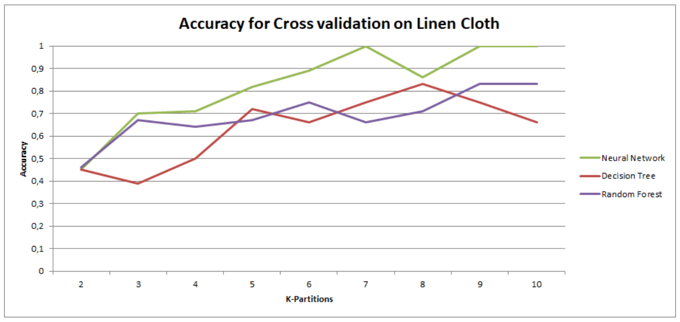


Figure 9: Classifier Prediction Accuracy Comparison

In general, the classifiers are close to each other in terms of accuracy but the neural network comes out with the highest overall accuracy. Not surprisingly, the random forest has a higher accuracy than the decision tree, which is most likely because it uses an ensemble of decision trees to come with the best prediction based on a majority voted label. Besides this, we experienced that decision trees are easier to interpret and faster to train, whereas the neural network yielded a higher accuracy at the cost of its increased training time.

5.4 Apriori results

The Apriori algorithm was used to try and reveal popular farming areas players use to gather commodities. One hypothesis was that players would primarily farm ingredients used in in-game professions.

Using a support of 0.5% and a minimum confidence of 80%, it was found that the hypothesis was proven to be a general tendency. In a test including all the gathered data, it was found that 30 out of 32 rules found were rules about flowers, which are used by multiple professions in WoW. The last two rules described cloth and gems, which are profession materials as well (see appendix C).

After looking at rules on the whole dataset, an investigation of whether commodities trend differently in different time spans was performed. As a result, it was found that the amount of rules drop significantly during weekends. Namely, only three rules were found with a minimum confidence of 80% all about flowers. This could be due to individuals playing more together with friends on weekends compared to weekdays. Arguably, weekends leave less time to individual gathering and more time completing dungeons together as a team. Therefore, the commodities that are gathered on weekends may be different from than on weekdays.

In comparison to weekends, 21 rules were found on weekdays. Many of the rules were about flowers but surprising association rules including gems and enchantments were also present during weekdays. Also, many of the associations included the same commodities. This information was used to reconstruct a potential farming route that one might take to collect commodities frequently sold together. This route shows a clear relationship between areas used to gather specific combinations of commodities and the item association rules found by the Apriori algorithm (see route in appendix A.6).

Monthly tendencies were also investigated but nothing of particular interest was found in this investigation. The months that have been gathered data in are not that interesting. If they had included any data from in-game events it would have been interesting to investigate if these would introduce new associations. This could be an interesting subject for further investigation. Finally, to validate the association rules we performed an integration test and tested whether the same results were found by the official FP-growth found in spark.

6 Conclusion

In conclusion, supervised classification and frequent pattern mining algorithms have been used to discover new knowledge about the virtual economy in World of Warcraft. Firstly, the Apriori algorithm helped find association rules that reveal popular items. Interestingly, these showed that items are often sold together based on profession or proximity and were used to help find popular farming areas. Further, a neural network and decision trees implemented in Spark and Scala were used to classify price movements. Most data mining algorithms are sensitive to noise so the auction data was normalized and outliers removed using the interquartile range. Further, cross validation was used to measure the variance of accuracy across the classifiers. The neural network yielded the highest accuracy but had the longest training time and its results were hard to understand. On the other hand, the decision tree classifiers were fast and easy to interpret with the random forest yielding a better accuracy than the decision tree. Finally, the lack of historical data proved to be a huge challenge experienced during the project, and having aggregated more historical data ultimately should have stirred a change of direction in the project and helped improve the final results of our predictions dramatically.

Acknowledgments

The authors would like to thank Blizzard Entertainment for providing an open API for fetching Data from the video game World of Warcraft. Also the authors would like thank Paolo Burelli, Sebastian Risi and all the TA's for teaching this course.

References

[1] Number of world of warcraft subscribers by quarter, Statista, 10 May 2018, https://www.statista.com/statistics/276601/number-of-world-of-warcraft-subscribers-by-quarter/
[2] Decision Trees - RDD-based API, Spark, 10 May 2018, https://spark.apache.org/docs/latest/mllib-decision-tree.html
[3] Random Forests, Spark, 10 May 2018, https://spark.apache.org/docs/2.2.0/mllib-ensembles.html
[4] Data Mining Concetps and Techniques - Third Edition , Jiawei Han, Micheline Kamber, Jian Pei - MK Morgan Kaufmann - ISBN 978-0-12-381479-1
[5] Apriori vs FP-Growth for Frequent Item Set Mining, 13 May 2018, https://www.singularities.com/blog/2015/08/apriori-vs-fpgrowth-for-frequent-item-set-mining

APPENDIX A
VISUALIZATIONS

A.1 Removal of extreme buyout values and the effects on the overall price distribution

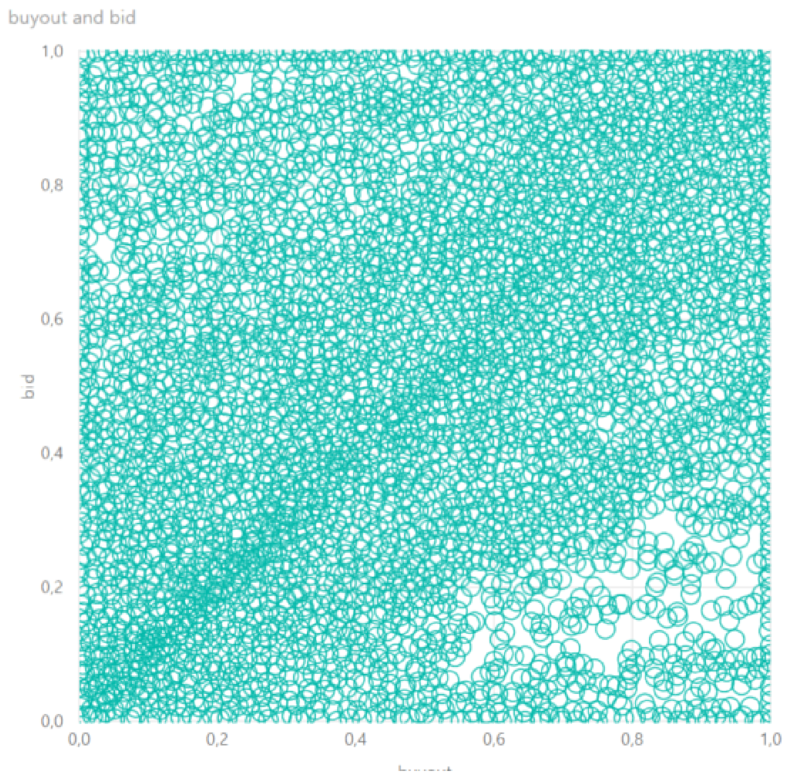


Figure 10: All data before outliers are removed

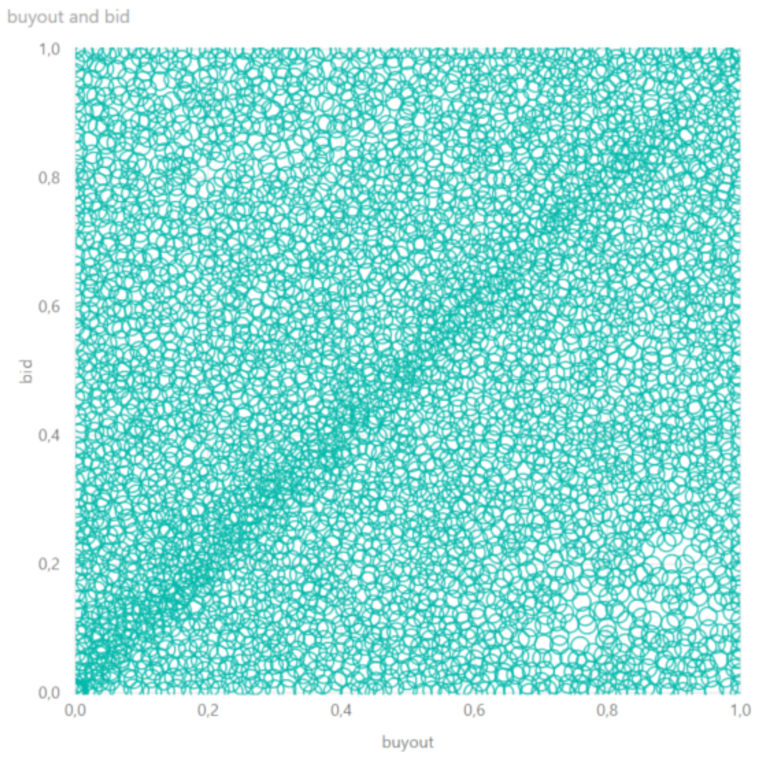


Figure 11: All data with 10 percent outliers removed

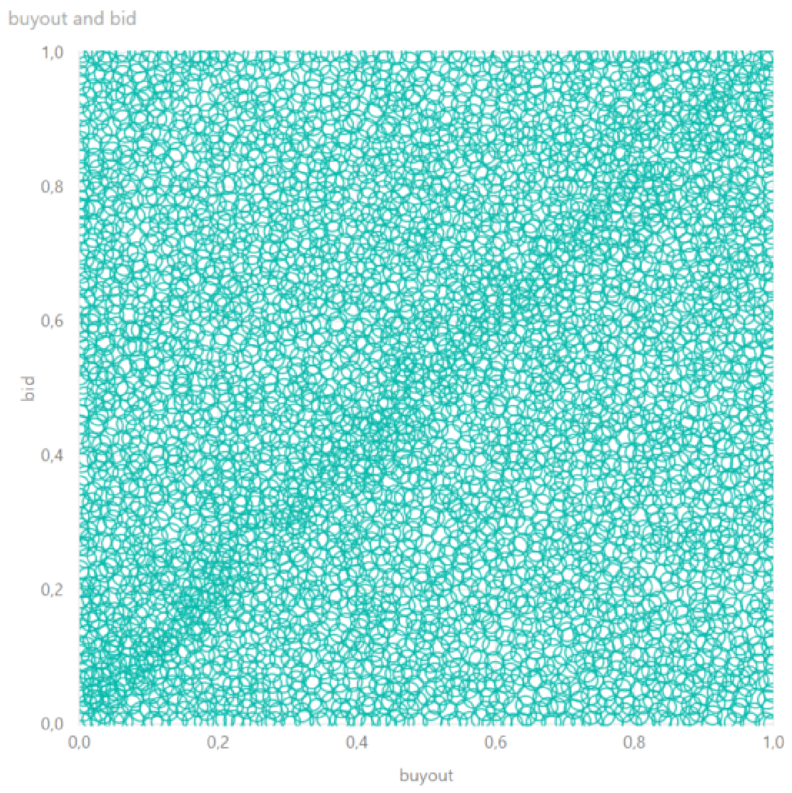


Figure 12: All data with 25 percent outliers removed

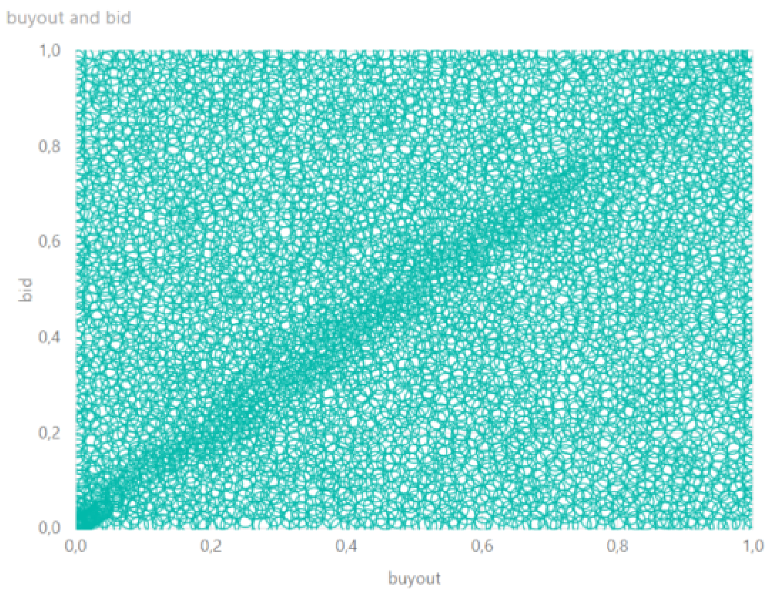


Figure 13: All data outliers removed with IQT

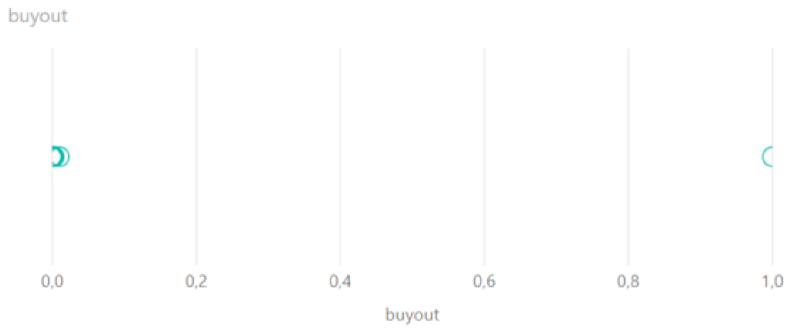


Figure 14: Linen Cloth before outliers are removed

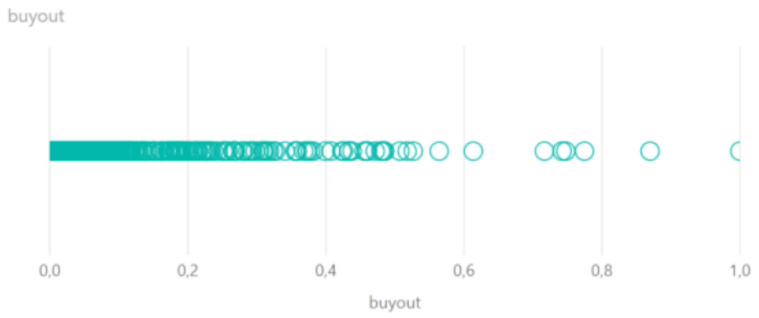


Figure 15: Linen Cloth with 10 percent outliers removed

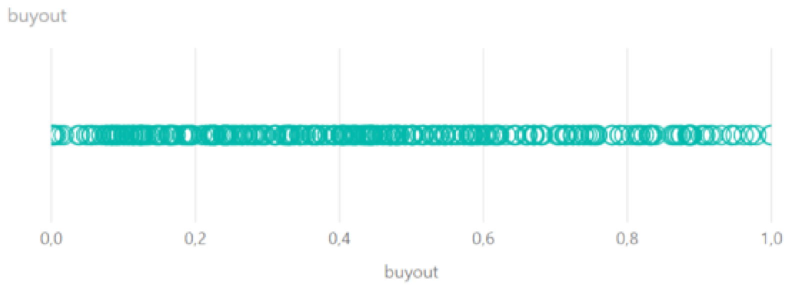


Figure 16: Linen Cloth with 25 percent outliers removed

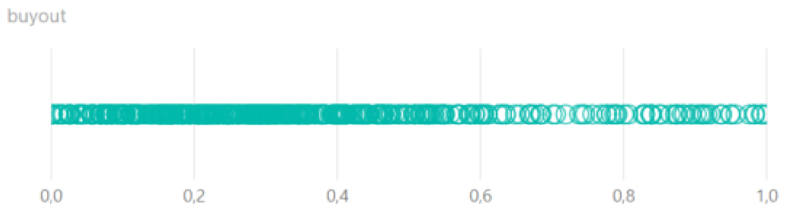
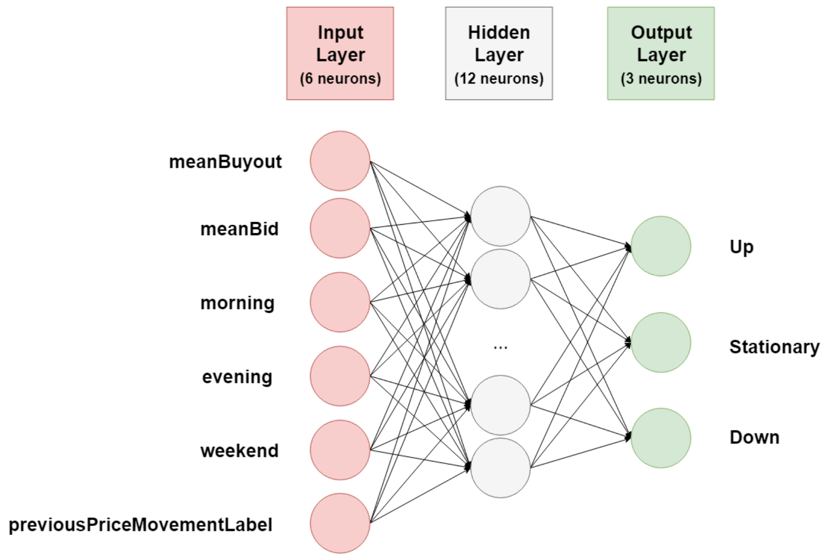


Figure 17: Linen Cloth with outliers removed with IQT

A.2 Removal of extreme buyout values and the effects on Linen Cloth's price distribution

A.3 ANN structure diagram



A.4 Cross validation accuracy difference for Neural network

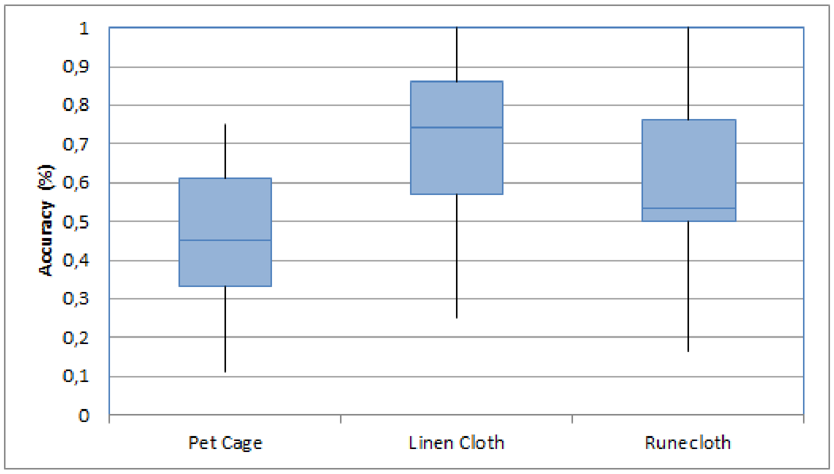


Figure 18: Low tolerance and 10-fold.

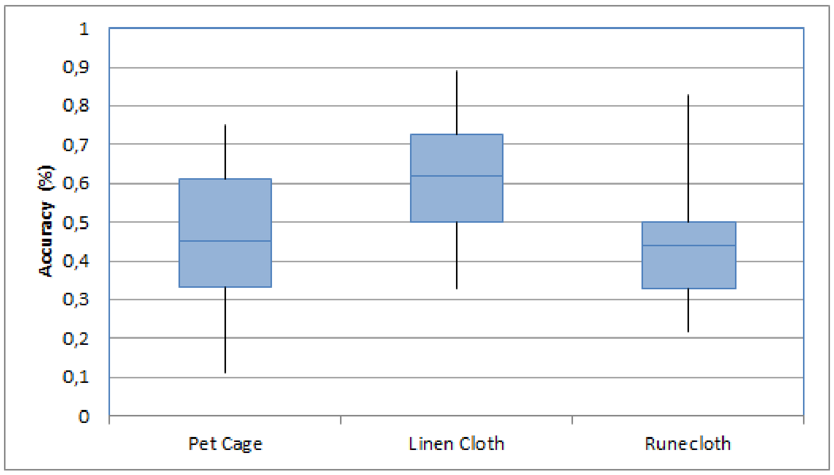


Figure 19: Higher tolerance and 10-fold.

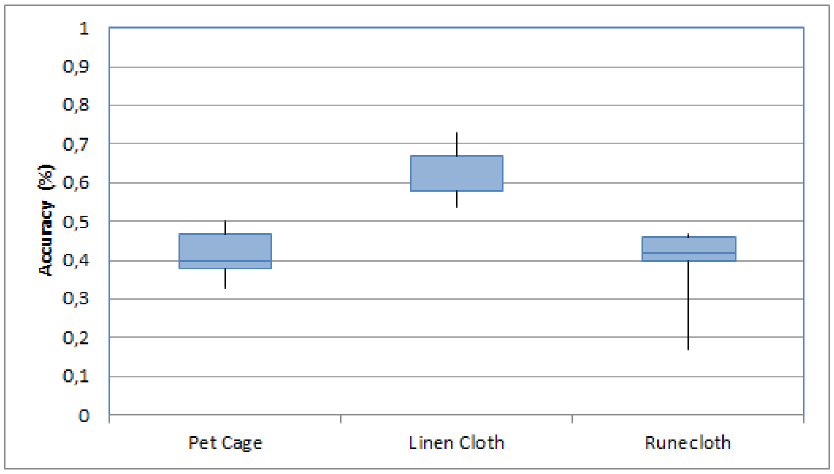


Figure 20: Higher tolerance and 5-fold.

A.5 Comparison of Classification algorithms

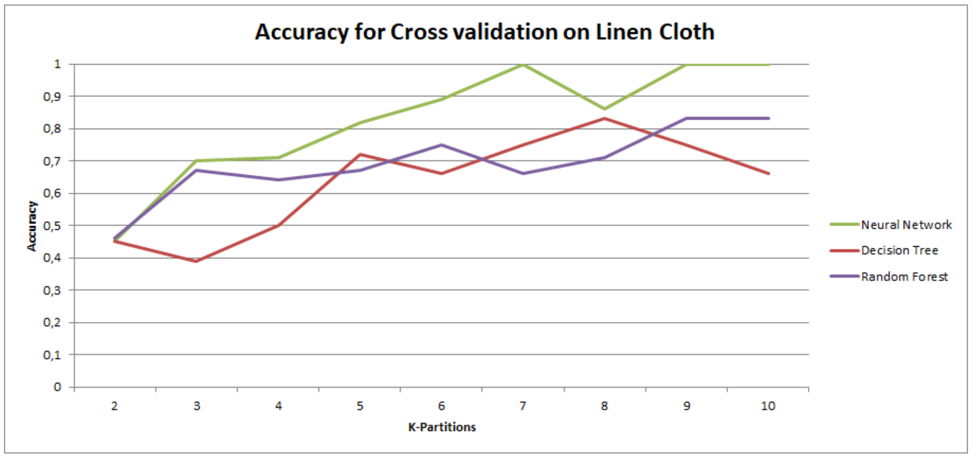


Figure 21: The set of data is split into k partitions and trained on k-1 partitions and tested on the last partition.

A.6 Reconstructed potential farming route from Apriori

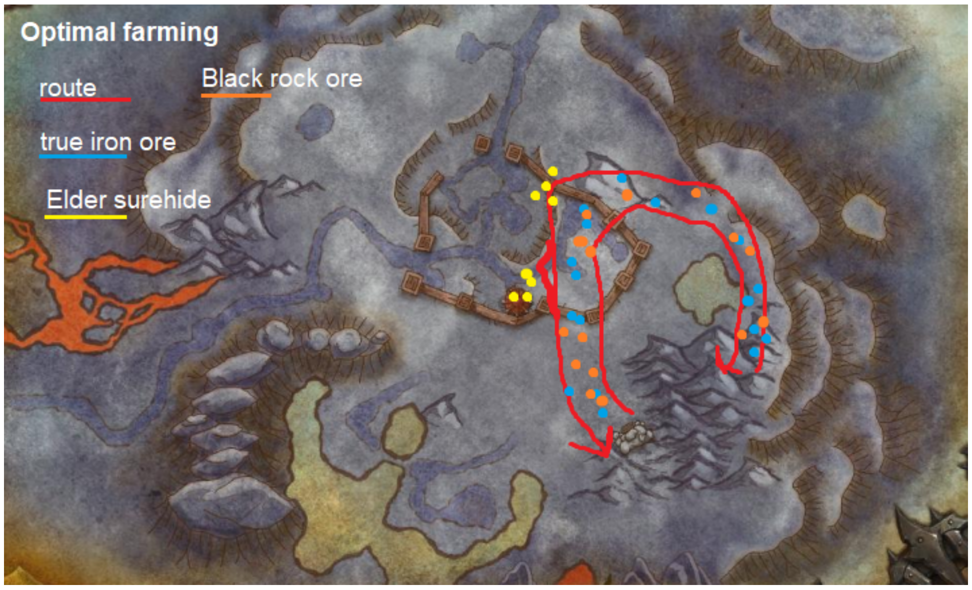
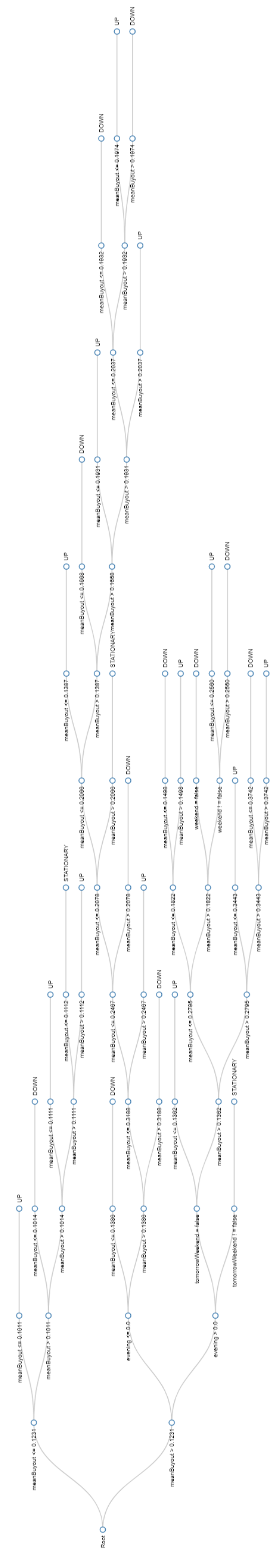


Figure 22: Gathering route

A.7 Illustration of Decision Tree trained with Linen Cloth



APPENDIX B
SCALA CODE SNIPPETS

B.1 Data Reduction

```
/**
  * Reduce the volume of data by removing columns that are not used for further analysis
  *
  * @param dataSet Dirty [[Auction]] data set to reduce.
  * @return Reduced [[Auction]] data set.
  */
override def reduce(dataSet: Dataset[Auction]): Dataset[Auction] = {
  val reducedAuctions = dataSet
    .drop("name") // Server name
    .drop("slug") // Server ID
    .drop("bonusLists") // Item upgrades
    .drop("context") // Dungeon id
    .drop("modifiers") // Modifiers decide final item lvl
    .drop("petBreedId") // Ignore pet info
    .drop("petLevel")
    .drop("petQualityId")
    .drop("petSpeciesId")
    .drop("rand") // Item stat boost
    .drop("seed") // Seed used to calculate stat boost of item
  reducedAuctions.as[Auction]
}
```

B.2 Data Cleaning

```
/**
  * Clean [[Auction]] data by filling in missing values, smoothing noisy data, removing
↪ outliers, and resolving inconsistencies.
  *
  * @param dataSet Dirty [[Auction]] data set to clean.
  * @return Cleaned [[Auction]] data set.
  */
  override def clean(dataSet: Dataset[Auction]): Dataset[Auction] = {
    //remove auctions with only bid
    val onlyBuyoutAuctions = nonEmptyAuctions.filter(_.buyout > 0)

    //standardize auctions so that we can always expect quantity to be 1
    val singleQuantityAuctions = onlyBuyoutAuctions.map(auc => auc.copy(bid =
     ↪ auc.bid/auc.quantity,buyout = auc.buyout/auc.quantity,quantity = 1))

    // Identify and remove outliers
    val filteredBuyOutliers = quantileOutlierDetection(singleQuantityAuctions,
     ↪ "meanBuyout")

    filteredBuyOutliers
}
```

B.3 Interquartile Outlier Detection

```
private def quartileOutlierDetection(dataSet: Dataset[Auction], column: String):
 ↪ Dataset[Auction] = {
  dataSet.createOrReplaceTempView("df") //create sql table

  //run a query finding the quantiles for each item
  val withQuartiles = sparkSession.sql(s"select item, percentile_approx($column,0.75) as
   ↪ Q1," +
                s"percentile_approx($column,0.25) as Q3 from df group by item")
  val withIQR = withQuartiles.withColumn("IQR",expr("Q3-Q1"))
  //join with the dataSet
```

```
    val combined = dataSet.join(withIQR, "item")
      .withColumn("lowerBound",expr("Q1 - 1.5 * IQR"))
      .withColumn("upperBound",expr("Q3 + 1.5 * IQR"))

    //filter the auctions not within the range.
    combined.filter(s"$column > lowerBound and $column < upperBound").as[Auction]
  }
```

B.4 Normalization

```
  /**
    * Normalize [[Auction]] data by adjusting the data to be consistent.
    *
    * @param dataSet Dirty [[Auction]] data set.
    * @return Normalized [[Auction]] data set.
    */
  override def normalize(dataSet: Dataset[Auction]): Dataset[PreprocessedAuction] = {
    val summary = dataSet
      .groupBy("item")
      .agg(
        min("Bid").name("minBid"),
        min("Buyout").name("minBuyout"),
        max("Bid").name("maxBid"),
        max("Buyout").name("maxBuyout"),
        count("item").as("totalItemAuctions")
      )

    val aggregated = dataSet.join(summary, "item").as[SummaryAuction]

    val normalized = aggregated.map(auc => auc.copy(
      bid = normalize(auc.bid, auc.minBid, auc.maxBid, auc.totalItemAuctions),
      buyout = normalize(auc.buyout, auc.minBuyout, auc.maxBuyout, auc.totalItemAuctions)
    ))

  normalized.as[PreprocessedAuction]
  }
```

B.5 Transformation

```
  /**
    * Fit [[Auction]] data to [[ArtificialNeuralNetwork]] prediction model.
    * This is done by adding averaged numeric values (bid, buyout, quantity) and day of
↪   auctions.
    *
    * @param dataset Data set of auctions.
    * @return [[LabeledAuction]] model used for prediction in [[ArtificialNeuralNetwork]].
    */
  def getAveragedAuctionsByDay(dataset: Dataset[PreprocessedAuction], itemID: Int):
   ↪  (Dataset[LabeledAuction],LabeledAuction) = {

    println(s"Filtering for: ${DataFetcher.getItemName(itemID)}")
    val auctionsById = dataset.filter(auc => auc.item == itemID)

    val averagedAuctions = auctionsById
      .groupBy("date", "item", "evening", "morning", "dayOfWeek")
      .agg(
        mean("buyout").name("meanBuyout"),
        mean("bid").name("meanBid"),
        min("buyout").name("minBuyout"),
        min("bid").name("minBid"),
        sum("quantity").name("totalQuantity"), //sum instead of mean, mean("quantity")
         ↪  does not provide us any relevant information
        stddev_pop("buyout").name("stdBuyout")
      )
    println(s"MOST VOLATILE ITEM WITH ID $itemID: ${DataFetcher.getItemName(itemID)} and
     ↪  ${auctionsById.count} AUCTIONS...")

    val auctionsByDay = averagedAuctions
      .withColumn("weekend", expr("dayOfWeek > " + DayOfWeek.FRIDAY.getValue))
      .withColumn("tomorrowWeekend", expr("dayOfWeek == " + DayOfWeek.FRIDAY.getValue))
      .withColumn("priceMovementLabel",lit(
          PriceMovementLabel.toString(PriceMovementLabel.Stationary)))
      .withColumn("previousPriceMovementLabel",lit(
      PriceMovementLabel.getNumericValue(PriceMovementLabel.Stationary)))

    val predictionModel = auctionsByDay.as[LabeledAuction].sort("item", "date","morning")

    // Label auctions by their price movement
    val labelFoldResult = {
      predictionModel
        .collect() // (list, currentAuction, previousAuction)
        .foldLeft ((List[LabeledAuction](), predictionModel.first(),
         ↪  predictionModel.first()))
        ((auctionsCurrentPairAcc, nextAuction) => {
          val (auctionsAcc, currentAuction, previousAuction) = auctionsCurrentPairAcc
          val priceMovement = PriceMovementLabel.getMovement(currentAuction.meanBuyout,
           ↪  nextAuction.meanBuyout)
          val labelledAuction = currentAuction.copy(
            previousPriceMovementLabel =
             ↪  PriceMovementLabel.getNumericValueFromString(previousAuction.priceMovementLabel),
            priceMovementLabel = PriceMovementLabel.toString(priceMovement)
          )
          (labelledAuction :: auctionsAcc, nextAuction, labelledAuction)
      })
    }

    val newestAuction = labelFoldResult._2 //the auctions for today
    val labelledPredictions = labelFoldResult._1.reverse.drop(1).toDS() // Reverse result
     ↪  and drop first element

    println(s"${labelledPredictions.count} AUCTIONS LABELLED FOR PREDICTION...")

    (labelledPredictions,newestAuction)
  }
```

B.6 Artificial Neural Network

```
def train(train: Dataset[LabeledAuction],
  test: Dataset[LabeledAuction]): PredictionModel[PipelineModel] = {

  val layers = Array(6, 12, 3)

  val labelIndexer = new
   ↪  StringIndexer().setInputCol("priceMovementLabel").setOutputCol("label")

  val featuresArr = Array("meanBuyout", "meanBid", "morning", "evening", "weekend",
  "previousPriceMovementLabel")

  val va = new VectorAssembler().setInputCols(featuresArr).setOutputCol("features")

  val trainer = new MultilayerPerceptronClassifier()
  .setLayers(layers)
  .setBlockSize(128) // Block size for putting input data in matrices for faster
   ↪  computation.
  .setSeed(1234L) // Seed for weight initialization if weights are not set.
  .setMaxIter(200) // Maximum number of iterations.
  .setStepSize(1E-6) // Set quantity by which weights are modified (learning rate).
  .setTol(1E-10) // Set convergence tolerance.

  val pipeline = new Pipeline().setStages(Array(labelIndexer, va, trainer))
  val model = pipeline.fit(train)
  val result = model.transform(test)

  val predictionAndLabels = result.select("prediction", "label")
  val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")
  val accuracy = evaluator.evaluate(predictionAndLabels)

  println(s"Test set accuracy = $accuracy")
}
```

B.7 Decision Tree and Random Forest Model

```
  //Help function to create vector for decision tree
  def getVector(data: LabeledAuction): Vector ={
    Vectors.dense(Array(
      data.meanBid,
      data.meanBuyout,
      toBinary(data.tomorrowWeekend),
      toBinary(data.weekend),
      toBinary(data.morning),
      toBinary(data.evening)
    ))
  }
```

B.8 Decision Tree Implementation

```
  override def train(...) = {
    //Some code has been omitted for clarity
    val itemName = DataFetcher.getItemName(train.first().item)
    val trainData = getModel(train)
    val testData = getModel(test)

    // Setup DecisionTree variables
    //Number of classification classes
    val numClasses = 3
    val maxDepth = 5

    //Number of bins used when discretizing continuous features
    val maxBins = 64

    //Setup spark decision tree and train it
    val model = DecisionTree.trainClassifier(
      trainData.labeledData,
      numClasses, impurity, maxDepth, maxBins)

    //test model with test data
    val testResult = testData.labeledData.map { point =>
      val prediction = model.predict(point.features)
      (point.label, prediction)
    }
    val labelAndPrediction = testResult.map { point =>
      NumericLabelPrediction(point._1, point._2)
    }
    // Return built tree
    PredictionModel(labelAndPrediction.toDF(), model)
  }
```

APPENDIX C
APRIORI ASSOCIATION RULES

```
Weekends (0.8, 0.005)
LHS: Set(Nagrand Arrowbloom, Starflower) => RHS: Set(Talador Orchid) | confidence: 0.82
LHS: Set(Fireweed, Nagrand Arrowbloom) => RHS: Set(Talador Orchid) | confidence: 0.89
LHS: Set(Starflower, Fireweed) => RHS: Set(Talador Orchid) | confidence: 0.84

Weekdays (0.8, 0.005)
LHS: Set(Starflower, Gorgrond Flytrap) => RHS: Set(Talador Orchid) | confidence: 0.82
LHS: Set(Quick Lightsphene, Deadly Deep Chemirine) => RHS: Set(Versatile Labradorite) | confidence:
LHS: Set(Fireweed, Gorgrond Flytrap) => RHS: Set(Starflower) | confidence: 0.82
LHS: Set(Enchant Cloak - Binding of Intellect, Enchant Cloak - Binding of Strength) => RHS: Set(Enc
LHS: Set(Starflower, Fireweed, Frostweed) => RHS: Set(Talador Orchid) | confidence: 0.86
LHS: Set(Talador Orchid, Nagrand Arrowbloom) => RHS: Set(Starflower) | confidence: 0.82
LHS: Set(Gorgrond Flytrap, Talador Orchid) => RHS: Set(Starflower) | confidence: 0.84
LHS: Set(Masterful Argulite, Deadly Deep Chemirine) => RHS: Set(Quick Lightsphene) | confidence: 0.
LHS: Set(Starflower, Talador Orchid, Nagrand Arrowbloom) => RHS: Set(Fireweed) | confidence: 0.83
LHS: Set(Starflower, Frostweed, Fireweed) => RHS: Set(Nagrand Arrowbloom) | confidence: 0.83
LHS: Set(Enchant Cloak - Binding of Strength, Enchant Cloak - Binding of Agility) => RHS: Set(Encha
LHS: Set(Versatile Labradorite, Deadly Deep Chemirine) => RHS: Set(Quick Lightsphene) | confidence:
LHS: Set(Talador Orchid, Frostweed) => RHS: Set(Starflower) | confidence: 0.83
LHS: Set(Fireweed, Talador Orchid, Frostweed) => RHS: Set(Starflower) | confidence: 0.89
LHS: Set(Versatile Labradorite, Masterful Argulite) => RHS: Set(Quick Lightsphene) | confidence: 0.
LHS: Set(Frostweed, Nagrand Arrowbloom) => RHS: Set(Starflower) | confidence: 0.8
LHS: Set(Versatile Labradorite, Quick Lightsphene) => RHS: Set(Deadly Deep Chemirine) | confidence:
LHS: Set(Talador Orchid, Nagrand Arrowbloom, Fireweed) => RHS: Set(Starflower) | confidence: 0.86

All (0.8, 0.005)
LHS: Set(Talador Orchid, Gorgrond Flytrap) => RHS: Set(Fireweed) | confidence: 0.82
LHS: Set(Fireweed, Starflower, Frostweed) => RHS: Set(Nagrand Arrowbloom) | confidence: 0.87
LHS: Set(Talador Orchid, Frostweed) => RHS: Set(Starflower) | confidence: 0.82
LHS: Set(Nagrand Arrowbloom, Frostweed) => RHS: Set(Fireweed) | confidence: 0.91
LHS: Set(Nagrand Arrowbloom, Starflower, Talador Orchid) => RHS: Set(Fireweed) | confidence: 0.87
LHS: Set(Starflower, Talador Orchid, Frostweed) => RHS: Set(Fireweed) | confidence: 0.87
LHS: Set(Fireweed, Frostweed) => RHS: Set(Starflower) | confidence: 0.83
LHS: Set(Fireweed, Nagrand Arrowbloom, Starflower) => RHS: Set(Talador Orchid) | confidence: 0.8
LHS: Set(Frostweed, Nagrand Arrowbloom) => RHS: Set(Fireweed, Talador Orchid) | confidence: 0.81
LHS: Set(Frostweed, Nagrand Arrowbloom) => RHS: Set(Starflower) | confidence: 0.84
LHS: Set(Fireweed, Nagrand Arrowbloom, Frostweed) => RHS: Set(Starflower) | confidence: 0.9
LHS: Set(Frostweed, Talador Orchid, Nagrand Arrowbloom) => RHS: Set(Fireweed) | confidence: 1.0
LHS: Set(Masterful Argulite, Deadly Deep Chemirine) => RHS: Set(Quick Lightsphene) | confidence: 0.
LHS: Set(Fireweed, Nagrand Arrowbloom, Talador Orchid) => RHS: Set(Starflower) | confidence: 0.83
LHS: Set(Nagrand Arrowbloom, Frostweed) => RHS: Set(Fireweed, Starflower) | confidence: 0.81
LHS: Set(Starflower, Fireweed, Frostweed) => RHS: Set(Talador Orchid) | confidence: 0.9
LHS: Set(Fireweed, Frostweed) => RHS: Set(Talador Orchid) | confidence: 0.83
LHS: Set(Talador Orchid, Gorgrond Flytrap) => RHS: Set(Starflower) | confidence: 0.82
LHS: Set(Fireweed, Frostweed) => RHS: Set(Nagrand Arrowbloom) | confidence: 0.81
LHS: Set(Fireweed, Nagrand Arrowbloom) => RHS: Set(Starflower) | confidence: 0.8
LHS: Set(Fireweed, Frostweed, Talador Orchid) => RHS: Set(Nagrand Arrowbloom) | confidence: 0.87
LHS: Set(Talador Orchid, Nagrand Arrowbloom) => RHS: Set(Starflower) | confidence: 0.81
LHS: Set(Nagrand Arrowbloom, Starflower, Frostweed) => RHS: Set(Fireweed) | confidence: 0.96
LHS: Set(Nagrand Arrowbloom, Talador Orchid) => RHS: Set(Fireweed) | confidence: 0.85
LHS: Set(Fireweed, Gorgrond Flytrap) => RHS: Set(Starflower) | confidence: 0.85
LHS: Set(Linen Cloth, Silk Cloth) => RHS: Set(Wool Cloth) | confidence: 0.89
LHS: Set(Gorgrond Flytrap, Fireweed) => RHS: Set(Talador Orchid) | confidence: 0.82
LHS: Set(Fireweed, Talador Orchid, Frostweed) => RHS: Set(Starflower) | confidence: 0.9
LHS: Set(Fireweed, Frostweed, Nagrand Arrowbloom) => RHS: Set(Talador Orchid) | confidence: 0.9
LHS: Set(Gorgrond Flytrap, Nagrand Arrowbloom) => RHS: Set(Starflower) | confidence: 0.81
LHS: Set(Frostweed, Nagrand Arrowbloom) => RHS: Set(Talador Orchid) | confidence: 0.81
```

APPENDIX D
CROSS VALIDATION

D.1 ANN 10-fold Cross Validation on Linen Cloth

```
CROSS VALIDATION PERCENTAGE: 10
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   27|
|              Down|   25|
|        Stationary|    4|
+------------------+-----+

TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    4|
|              Down|    6|
|        Stationary|    1|
+------------------+-----+

+----------+----------+
|     label|prediction|
+----------+----------+
|      Down|        Up|
|      Down|      Down|
|      Down|      Down|
|        Up|        Up|
|Stationary|        Up|
|      Down|      Down|
|        Up|        Up|
|        Up|        Up|
|      Down|      Down|
|      Down|      Down|
|        Up|      Down|
+----------+----------+
ACCURACY: 0.7272727272727273

CROSS VALIDATION PERCENTAGE: 20
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   28|
|              Down|   30|
|        Stationary|    5|
+------------------+-----+

TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    3|
|              Down|    1|
+------------------+-----+
+-----+----------+
|label|prediction|
+-----+----------+
|   Up|        Up|
| Down|        Up|
| Down|      Down|
| Down|      Down|
+-----+----------+
ACCURACY: 0.75

CROSS VALIDATION PERCENTAGE: 30
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   28|
|              Down|   28|
|        Stationary|    4|
+------------------+-----+

TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    3|
|        Stationary|    1|
|              Down|    3|
+------------------+-----+

+----------+----------+
|     label|prediction|
+----------+----------+
|Stationary|        Up|
|        Up|        Up|
|      Down|      Down|
|      Down|Stationary|
|      Down|      Down|
|        Up|        Up|
|        Up|Stationary|
+----------+----------+
ACCURACY: 0.5714285714285714

CROSS VALIDATION PERCENTAGE: 40
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   25|
|              Down|   29|
|        Stationary|    5|
+------------------+-----+

TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    6|
|              Down|    2|
+------------------+-----+
+-----+----------+
|label|prediction|
+-----+----------+
| Down|        Up|
| Down|        Up|
| Down|        Up|
| Down|        Up|
| Down|      Down|
|   Up|      Down|
|   Up|        Up|
| Down|      Down|
+-----+----------+
ACCURACY: 0.375

CROSS VALIDATION PERCENTAGE: 50
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   27|
|              Down|   27|
|        Stationary|    4|
+------------------+-----+

TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    4|
|              Down|    4|
|        Stationary|    1|
+------------------+-----+

+----------+----------+
|     label|prediction|
+----------+----------+
|        Up|        Up|
|        Up|        Up|
|Stationary|        Up|
|        Up|        Up|
|      Down|      Down|
|      Down|      Down|
|      Down|      Down|
|        Up|        Up|
|      Down|      Down|
+----------+----------+
ACCURACY: 0.8888888888888888

CROSS VALIDATION PERCENTAGE: 60
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   30|
|              Down|   30|
|        Stationary|    4|
+------------------+-----+
TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    1|
|              Down|    1|
|        Stationary|    1|
+------------------+-----+

+----------+----------+
|     label|prediction|
+----------+----------+
|        Up|        Up|
|Stationary|        Up|
|      Down|        Up|
+----------+----------+
ACCURACY: 0.3333333333333333

CROSS VALIDATION PERCENTAGE: 70
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   29|
|              Down|   26|
|        Stationary|    5|
+------------------+-----+

TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    2|
|              Down|    5|
+------------------+-----+

+-----+----------+
|label|prediction|
+-----+----------+
| Down|      Down|
| Down|      Down|
| Down|      Down|
| Down|      Down|
|   Up|      Down|
|   Up|      Down|
| Down|      Down|
+-----+----------+
ACCURACY: 0.7142857142857143

CROSS VALIDATION PERCENTAGE: 80
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   29|
|              Down|   28|
|        Stationary|    4|
+------------------+-----+
TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    2|
|              Down|    3|
|        Stationary|    1|
+------------------+-----+

+----------+----------+
|     label|prediction|
+----------+----------+
|      Down|        Up|
|Stationary|        Up|
|      Down|      Down|
|        Up|        Up|
|      Down|      Down|
|        Up|      Down|
+----------+----------+
ACCURACY: 0.5

CROSS VALIDATION PERCENTAGE: 90
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   28|
|              Down|   28|
|        Stationary|    5|
+------------------+-----+

TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    3|
|              Down|    3|
+------------------+-----+

+-----+----------+
|label|prediction|
+-----+----------+
|   Up|        Up|
| Down|      Down|
| Down|      Down|
|   Up|        Up|
|   Up|Stationary|
| Down|        Up|
+-----+----------+
ACCURACY: 0.6666666666666666

CROSS VALIDATION PERCENTAGE: 100
TRAINING SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|   28|
|              Down|   28|
|        Stationary|    5|
+------------------+-----+
TEST SET LABEL DISTRIBUTION
+------------------+-----+
|priceMovementLabel|count|
+------------------+-----+
|                Up|    3|
|              Down|    3|
+------------------+-----+

+-----+----------+
|label|prediction|
+-----+----------+
|   Up|        Up|
| Down|        Up|
|   Up|      Down|
|   Up|      Down|
| Down|      Down|
| Down|      Down|
+-----+----------+
ACCURACY: 0.5

Variance: 0.5555555555555556, minAccuracy: 0.3333333333333333, maxAccuracy: 0.8888888888888888
```

D.2 Decision Tree - Results depth 5 - Under represented test data

```
TRAINING SET LABEL DISTRIBUTION
|   Up | 26 |
|   Down |  26 |
|   Stationary| 5 |
+------------------+-----+
TEST SET LABEL DISTRIBUTION
|   Down | 2 |
+------------------+-----+
|RESULTS
|label|prediction|
+-----+----------+
| Down | Down |
| Down | Down |
+-----+----------+
ACCURACY: 1.0
```

D.3 Decision Tree - depth 5 - Okay results with a bit fitted towards ”up” and ”down”

```
TRAINING SET LABEL DISTRIBUTION
|   Up | 25 |
|   Down | 26 |
|   Stationary | 4 |
+------------------+-----+
TEST SET LABEL DISTRIBUTION
|   Up | 1 |
|   Stationary | 1 |
|   Down | 2 |
+----------+----------+
| RESULTS:
| label|prediction|
+----------+----------+
|   Up |  Up|
|   Stationary | Up |
|   Down | Down |
|   Down | Down |
+----------+----------+
ACCURACY: 0.75
```

D.4 Example of future predictions with Decision Tree top 10 items with the best tesult from 10-fold Cross Validation

```
+------------------------------+---------+--------+----------+
|item                          |yesterday|tomorrow|suggestion|
+------------------------------+---------+--------+----------+
|Netherweave Cloth             |Down     |Up      |BUY       |
|Runecloth                     |Down     |Up      |BUY       |
|Hexweave Bag                  |Up       |Down    |SELL      |
|Pet Cage                      |Up       |Down    |SELL      |
|Flask of the Countless Armies|Down     |Up      |BUY       |
|Flask of the Whispered Pact   |Up       |Down    |SELL      |
|The Hungry Magister           |Down     |Up      |BUY       |
|Potion of Prolonged Power     |Down     |Up      |BUY       |
|Defiled Augment Rune          |Up       |Down    |SELL      |
|Flask of the Seventh Demon    |Up       |Down    |SELL      |
+------------------------------+---------+--------+----------+
[success] Total time: 634 s
```