Discovering Patterns in the Russian Housing Market for Analysis and Prediction

3804ICT Assignment Part II | Final Project Report | Trimester 2, 2019

Joshua Russell s5057545
Joshua Mitchell s5055278
Hayden Flatley s5088623

{joshua.russell2, joshua.mitchell4, hayden.flatley}@griffithuni.edu.au

1. Introduction

Our project explored the discovery and utilisation of patterns in the Russian housing market for detailed analysis and prediction. We investigated different applications of data mining algorithms within this domain. Specifically, this entailed predicting the sale price of properties, determining the most likely Russian sub-areas given some property and neighbourhood characteristics, and finding interesting relationships, as well as associations, between property and neighbourhood features. We were motivated to experiment with algorithms within this property market domain because of the wide variety of possible applications for the technology. For example, an algorithm that is able to predict the sale price of properties, could be used to assist home sellers with finding out the expected sale price of their home. Additionally, it could be used to help home buyers with making strategic offers on properties, and it could be used by banks and real estate agencies to provide an "unbiased" opinion on the value of a property for valuations and home appraisals. The potential for data mining algorithms to provide significant utility within this domain was clear to us in theory. Hence, we decided to evaluate our intuition and test these algorithms in practice.

The investigation was conducted with a Russian housing market dataset provided by Sberbank [1]. Sberbank is one of the largest financial services and banking companies in Russia. They use the aforementioned dataset in their daily operations to help their customers, who may be renters, developers, or lenders, plan budgets for properties. The dataset consists of both categorical and numerical valued attributes, describing property features, such as the number of rooms and the total living area in square meters, and information about the surrounding neighbourhood, such as the sub-area's population and the distance to the nearest park. In total, the dataset contains 38,122 property entries (i.e. data samples) described by 292 attributes. The plethora of information contained within this dataset proved to be useful for the data-driven algorithms we used within our investigation. Although this project used a Russian housing market dataset, we believe that our findings and developed algorithms are relevant to housing markets around the world.

To test the application of data mining algorithms within the housing market domain, we decided to conduct five separate studies. The first two studies looked at predicting the sale price of properties, posed firstly as a regression problem and then as a forecasting problem. To solve the regression problem, we implemented a multilayer perceptron, and for the forecasting problem, we used exponential smoothing. The third study was based upon a classification problem, where we wanted to classify property and neighbourhood characteristics into Russian sub-areas. We were able to achieve this by applying the Naïve Bayes Classifier on the problem. In our fourth study, we were interested in discovering interesting frequent patterns within the housing market. Accordingly, we mined such frequent patterns with the Frequent Pattern-Growth algorithm. Finding insightful relationships between property entries and attributes within the Russian housing market dataset was the goal of our final study. Since clustering allows us to find such relationships, we posed the problem as a clustering problem and solved it with the k-means algorithm. The following investigation details the findings of these studies.

2. Solution

This section details the solutions to the five studies of our investigation. Jupyter Notebooks were developed for each of these studies. Therefore, we will not comment extensively on the studies within this report and instead refer the reader to the appropriate notebooks provided. In short, these notebooks introduce the aim of the study and the algorithm used to solve the proposed problem. They detail the required pre-processing steps that were applied to the data for compatibility and effectiveness with the algorithms used. They contain two library solutions and one manual implementation of the algorithm, and lastly, they present the obtained results and metrics of the study.

2.1. Multilayer Perceptron

The aim of this study was to develop a model for predicting Russian house prices based upon information about the property and the surrounding area. We used a feedforward artificial neural network known as the multilayer perceptron for this task of regression. Besides being implemented manually, the multilayer perceptron was also implemented using two Python libraries (Scikit-learn and Keras). For complete details, please refer to the Jupyter Notebook entitled *"Multilayer Perceptron"*. Sources utilised in the development of the solution are referenced within the notebook [2, 3, 4, 5].

2.2. Exponential Smoothing

The goal of this study was to develop a model for forecasting Russian house prices. We used a method known as Holt-Winters (i.e. triple exponential smoothing) for this task. Besides being implemented manually, the Holt-Winters exponential smoothing model was also implemented using a Python library (StatsModels) and an R library (HoltWinters). For complete details, please refer to the Jupyter Notebook entitled *"Exponential Smoothing"* (note that the R (HoltWinters) library implementation can be found within the Jupyter Notebook entitled *"Exponential Smoothing-R"*). Sources utilised in the development of the solution are referenced within the notebook [11, 12, 15].

2.3. Naïve Bayes Classifier

This study aimed to develop a classifier for predicting the sub-areas (districts) of properties in Russia based upon property and neighbourhood attributes. We used a probabilistic classifier based on Bayes' theorem, known as the Naïve Bayes Classifier, for this task of classification. Besides being implemented manually, the Naïve Bayes Classifier was also implemented using two Python libraries (Scikit-learn and the Natural Language Toolkit (NLTK)). For complete details, please refer to the Jupyter Notebook entitled *"Naive Bayes Classifier"*. Sources utilised in the development of the solution are referenced within the notebook [6, 7, 8].

2.4. Frequent Pattern-Growth

The objective of this study was to find interesting relationships among property and neighbourhood attribute values in the form of frequent patterns. We used the FP-Growth algorithm for this task of frequent pattern mining. Besides being implemented manually, the FP-Growth algorithm was also implemented using two Python libraries (Mlxtend (Machine Learning Extensions) and PyFPGrowth). For complete details, please refer to the Jupyter Notebook entitled *"FP-Growth"*. Sources utilised in the development of the solution are referenced within the notebook [9, 10].

2.5. K-Means

The goal of this study was to find interesting patterns and similarities between attributes of the Russian Housing Market. To discover such patterns and similarities, we posed the problem as a clustering problem and used the k-means clustering algorithm. Besides being implemented manually, the k-means algorithm was also implemented using two Python libraries (Scikit-Learn and Mlxtend (Machine Learning Extensions)). For complete details, please refer to the Jupyter Notebook entitled *"K-Means"*. Sources utilised in the development of the solution are referenced within the notebook [13, 14].

3. Results and Metrics

Within this section we discuss the key results and metrics of our five studies. The purpose of our investigation was to evaluate whether data mining algorithms were practical within the housing market domain. Consequently, we collected all results and metrics by applying the algorithms on the Sberbank Russian housing market dataset. This in turn allowed us to assess the effectiveness of the algorithms within the housing market domain for each of the proposed problems. In addition to this, we provided detailed comparisons between the two library implementations of the algorithms and our own. Lastly, we explored potential practical applications of the algorithms within the housing market domain.

As with *2. Solution*, we will not cover all the investigated results and metrics within this section. Furthermore, we will not go into details explaining the evaluation measures that were used to collect and compare these results. For such additional information, please refer to the *Results and Metrics* section within the Jupyter Notebook corresponding to the particular study of interest.

3.1. Multilayer Perceptron

To reiterate, within this study we used a multilayer perceptron to predict the sale price of Russian properties based upon information about the property and surrounding area. To evaluate the implementations of the algorithm and the suitability of the algorithm within the housing market domain, we performed a series of analytical studies. Firstly, we examined the general performance of the algorithm on the housing market dataset and compared the three different implementations. Secondly, we investigated the effects of different hyperparameters on the performance of the multilayer perceptron, and finally, we explored and experimented with a potential use case for the algorithm within the domain of the Russian housing market.

We note that for this study, we normalised the target value, property sale price, into the range $[0,1]$, and standardized the feature vector attributes to have zero mean and unit variance. The reasoning behind the normalisation and standardisation is discussed within our notebook.

Dataset Experimentation and Algorithm Implementation Comparison

Here we examined the effectiveness of the multilayer perceptron at predicting Russian house prices based upon information about the property and the surrounding area. Additionally, we conducted a comparative analysis between the two library implementations and our own.

For the results reported within *Figure 1*, *Figure 2*, and *Figure 3*, we used the same multilayer perceptron architecture, consisting of 286 neurons in the input layer, 1024 neurons in the single hidden layer, and 1 neuron in the output layer. Furthermore, we used the Glorot normal initialiser to initialise the weights of the multilayer perceptron, and we trained the network for 1000 epochs with a learning rate of 0.001, and a mini-batch size of 128 samples [5].

Firstly, as shown in *Figure 1*, we analysed the training performance of the implemented algorithms. We observed that the mean squared error (MSE) between the actual property prices of the training samples and the predicted property prices of the networks decreased over training epochs. This provided evidence that all implementations of the multilayer perceptron were able to accurately learn a function that maps a given feature vector of property and neighbourhood attributes to a corresponding property sale price. This result supported the notion that the algorithm would be practical for the proposed problem within the housing market domain. However, to ensure that the function learnt by the algorithm could generalise beyond the seen training examples, we still needed to test its performance on the test set.

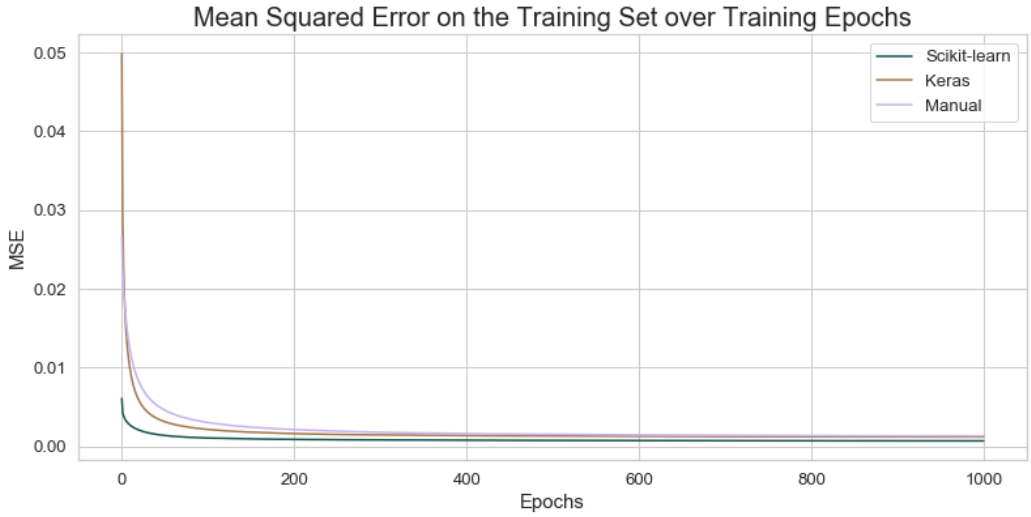

Figure 1. Training set mean squared error (MSE) over training epochs for each implementation.

Returning to *Figure 1*, we also observed slight differences between the implemented algorithms in their MSE over training epochs. The Scikit-learn implementation was first to converge at a local minimum in the error function after approximately 200 epochs of training. Whereas both the Keras and manual implementation converged approximately 100 epochs later. We assume that the cause of this difference was due to slight differences in the training protocol and implementation of the Scikit-learn multilayer perceptron. Moreover, we found that these slight differences also led to a more optimal local minimum being found by the Scikit-learn implementation, evident by the lower MSE at epoch 1000 in comparison to the other two implementations. As stated previously, this may have been due to the differences in the training protocol or in the implementation. However, it may also have been due to the weight initialisation of the Scikit-learn network being "closer" to a more optimal minimum in the error function.

	Scikit-learn	**Keras**	**Manual**
MSE	0.0019	**0.0018**	0.0021
MAE	**0.0232**	0.0255	0.0255

Figure 2. Test set mean squared error (MSE) and mean absolute error (MAE) for each implementation.

Following our analysis on the training performance of the algorithms, we investigated how well the learnt functions of the multilayer perceptrons could generalise beyond seen training examples. To evaluate the generalisability of the algorithms, we recorded the predictions of the networks on 1746 test set feature vectors. We then used two evaluation measures, mean squared error (MSE) and mean absolute error (MAE), to measure the difference between the predictions of the networks and the actual target values from the test set. The results are presented in *Figure 2*.

We found that the two library implementations and our own were able to quite accurately predict the sale price of properties given information about the property and the surrounding area. Since the MSE and MAE of the algorithms were all significantly small. Based upon these findings, we can conclude that the multilayer perceptron can be effectively applied to the problem of property sale price prediction. We built upon this conclusion later on in the study to explore how the algorithm could be used within a practical application.

Figure 2 not only provided us with a proof of concept for applying the multilayer perceptron to the proposed problem, but it also validated the correctness of our implementation. A low MSE and MAE on the test set showed that our manually implemented network was able to find a local minimum in the error function and in turn learn a general function for the proposed problem. In addition to this, *Figure 2* also presented results for comparing the library implementations with our own. We observed that there was no implementation that outperformed the others in both evaluation measures. Although the Keras implementation had the smallest MSE, and the Scikit-learn implementation had the smallest MAE, the difference in evaluation measures among the three implementations was not significant. This point further validates our implementation, as the MSE and MAE of our implementation were competitive with those of the library implementations for the multilayer perceptron.

Figure 3. Total training time for each implementation in seconds.

Our final comparative analysis between the implementations was in regard to training time. More specifically, this investigation measured the total amount of time it took for the network to train over the specified 1000 epochs, as well as the relatively miniscule time of doing a single forward pass for making predictions on the test set. Therefore, by definition, the investigation measured the efficiency of the implementations, as efficiency is defined as the total time to construct and use a model. Nevertheless, we report the findings as training time. Unlike the previous comparisons, we found that there were distinctive differences in training time between the three implementations. The Keras implementation trained significantly faster than the others, taking approximately four minutes to complete. In comparison, the Scikit-learn multilayer perceptron took approximately four times longer to train, at 16 minutes, and the manual implementation took approximately six times longer, at 24 minutes. From this investigation, we were able to see the differences in code optimization between the two libraries and our own implementation, and how significant such optimizations can be when training a computationally expensive model such as the multilayer perceptron.

In summary, we found that the multilayer perceptron could be successfully applied to the task of property sale price prediction within the housing market domain. Moreover, through a comparative analysis, we validated the correctness of our manual implementation and found that it was competitive with the library solutions in terms of predictive accuracy (in regard to the evaluation measures MSE and MAE). Lastly, we observed that our implementation took significantly more time than the library implementations to train, and consequently recognised the importance of code optimization on models such as the multilayer perceptron.

Variation in Hidden Layer Size

Within the hyperparameter experimentation section of the study, we investigated the effects of different hyperparameters on the training efficiency and predictive performance of the multilayer perceptron. In our notebook, we report results for variations in the learning rate, mini-batch size and hidden layer size. However, for this report, we will only address the investigation on various hidden layer sizes. Consequently, we refer the reader to the notebook for the results of the other experiments.

This investigation looked at the effects of using various numbers of hidden layer neurons for the multilayer perceptron. We used the manual implementation of the algorithm for the experiment. Furthermore, we used a multilayer perceptron architecture of 286 neurons in the input layer, varying numbers of neurons in the single hidden layer, and 1 neuron in the output layer. The network was trained for 100 epochs with a learning rate of 0.001 and a mini-batch size of 128 samples. Once again, we used the Glorot normal initialiser to initialise the weights of the network [5].

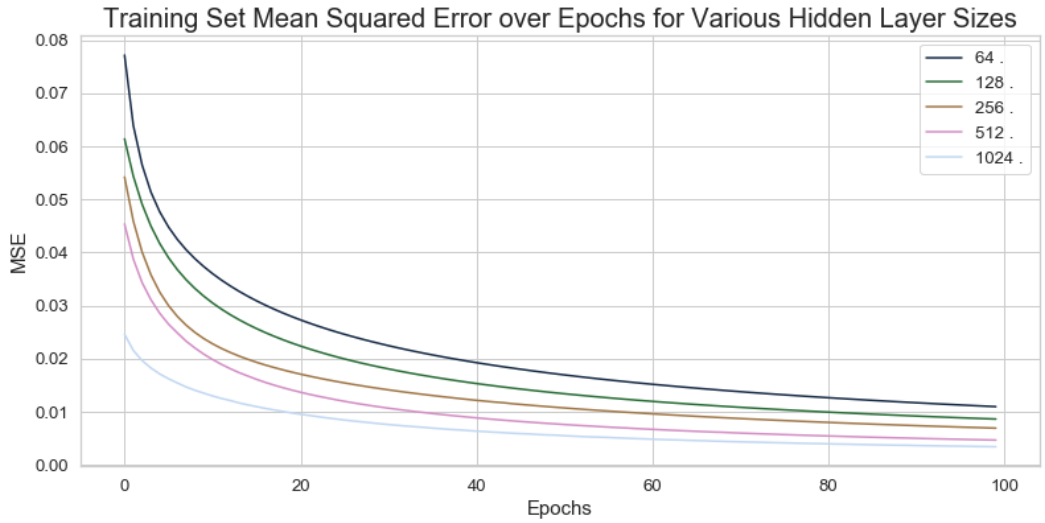

Figure 4. Training set mean squared error (MSE) over training epochs for multilayer perceptrons with varying numbers of hidden layer neurons.

To examine the effects of varying numbers of hidden layer neurons, we firstly explored training performance and efficiency. Our results are presented in *Figure 4*. We found that as the number of neurons in the hidden layer increases, the network converges more and more quickly to a local minimum in the error function. This is evident in the decreasing mean squared error (MSE) values over epochs in *Figure 4*. Not only does this allude to the fact that a large number of neurons in the hidden layer is able to learn the desired mathematical function more quickly, thus resulting in increased training efficiency, but it also suggests that the mathematical function we are trying to learn may require more parameters to accurately map given feature vectors to a continuous valued prediction of property sale price. Since the total number of mappings between 286 input variables to a single output value is incredibly large, we believe that this is indeed the case, as having more neurons allows the network to learn more complex mappings. Consequently, we hypothesise that a network with more hidden layer neurons, or perhaps a deeper architecture with multiple hidden layers, would be able to outperform a rather simple multilayer perceptron, with fewer hidden layer neurons or hidden layers, at this task of property sale price prediction.

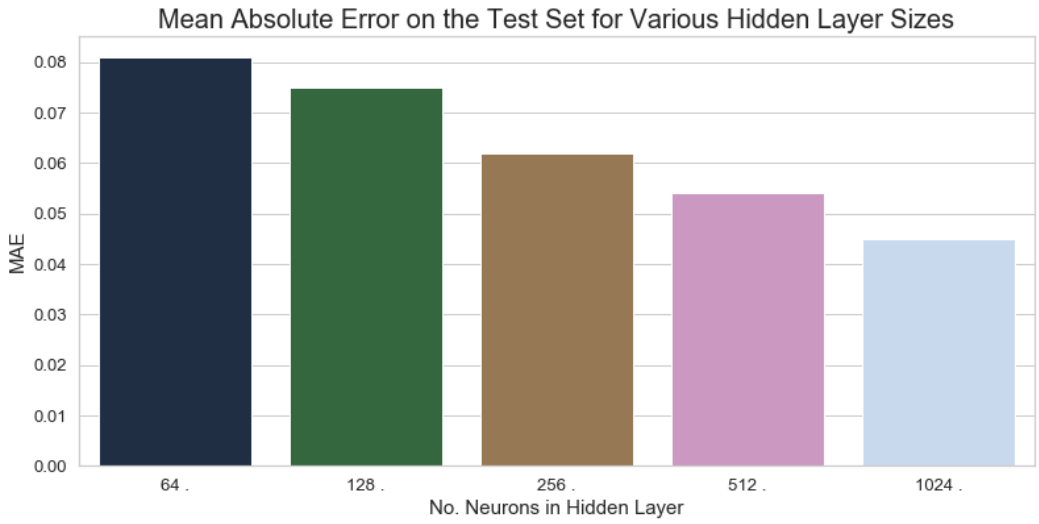

Figure 5. Test set mean absolute error (MAE) for multilayer perceptrons with varying numbers of hidden layer neurons.

The drawback of using too many parameters for learning the mathematical function is that the parameters can overfit the training data, as they effectively learn to "memorise" the input-output pairs that the network is shown during training. Within the context of property sale price prediction, overfitting would lead to a very low mean squared error on the training set, meaning that the network could accurately predict property sale prices for training set data samples. However, when tasked with generalising its predictive capabilities to unseen examples in the test set, its mean squared error or mean absolute error would be high, meaning that it cannot accurately predict the property sale prices of unseen data samples, since all it has done is memorise the mappings in the training set. With this in mind, we used the multilayer perceptrons with varying numbers of hidden layer neurons to predict the property sale price of test set feature vectors. We then calculated the mean absolute error (MAE) using the actual target values from the test set. These results are shown in *Figure 5*.

We observed in our results that as the number of hidden layer neurons increased the test set MAE decreased, meaning that the predictive accuracy on unseen data samples increased. These results were inline with our previously stated observation, that in order to accurately learn the desired mathematical function we may indeed need more parameters in the network. Furthermore, these results confirmed that the multilayer perceptron was not overfitting on the training data, but it was instead utilising the additional parameters to learn a more complex function for the task. We believe that to truly test our hypothesis that a network with more parameters would perform better at this task of property sale price prediction, we would need to train the current algorithms for longer and implement other networks, with more neurons and hidden layers, for a comparative analysis. However, this analysis is out of scope for our investigation.

Practical Application for the Multilayer Perceptron within the Housing Market Domain

Here we explored a potential practical application for the multilayer perceptron within the domain of the Russian housing market. Although this investigation was conducted using a Russian housing market dataset, we believe that the developed algorithm could be applied to any housing market domain around the world. Similar to the previous experiments, we looked at estimating the sale price of a property, as it has many useful benefits which are elaborated on within the notebook. However, for this practical application, we assumed that a user would not want to enter 286 values for the feature vector to receive

a prediction. Hence, we decided to implement an algorithm that predicts the sale price of properties given only some information about the property and surrounding area. To accomplish this task, we firstly trained the manually implemented multilayer perceptron on a training set for 1000 epochs with a learning rate of 0.001 and a mini-batch size of 128. The algorithm we implemented used this trained multilayer perceptron to accurately map feature vectors to property sale prices.

On a website, or within an application, users are not going to specify the values of attributes that are directly understandable by our network. For example, for the sub-area (district) attribute, people will enter "Ajeroport" or "Zjuzino", not 0 or 144. Consequently, the next step in our algorithm was to pre-process the attribute values provided by the user. We did this by remembering the encoding functions and standardization functions that we applied to each attribute in the training set, and then simply performed the relevant encoding and or standardization function on the provided attribute values. Following this step, we had a number of user-specified values ready to pass to the network. To fill in the remaining values of the input feature vector, we decided to take the training set mean of each of those remaining attributes. This approach essentially used the "average" characteristics of properties and neighbourhoods in Russia for the missing attributes. Thus, as users would specify their ideal property and surrounding area characteristics, the deviations from this "average" would either lead to an increase or decrease in predicted sale price. We hoped that the multilayer perceptron learnt meaningful characteristics about the attributes so that changes in this "average", such as a decrease in the number or rooms, would lead to a realistic change in predicted property sale price.

Once we had determined the feature vector based upon the user's preferences and the attribute means, we would then simply pass the feature vector to the multilayer perceptron and record its prediction. Since the network was trained to predict normalized property sale prices, we transformed the predicted price back to Russian rubles before returning it to the user.

After implementing the algorithm described above, we began to experiment with different possible use cases. As stated previously, for a practical application like this, we hoped that the multilayer perceptron was able to learn meaningful information about the different attributes. In the short amount of testing that we conducted, we found that the network did indeed learn the "general" meaning behind some attributes. We display this finding in *Figure 6*, where the algorithm predicted a less expensive property sale price for an older home.

```
# Example use case 2
property_price_predictor(predictor, {"build_year": 2017, "num_room": 5, "product_type": "OwnerOccupier"})
```

```
The predicted property price is 9051830 rubles (i.e. $208192 AUD)
```

```
# Example use case 3
property_price_predictor(predictor, {"build_year": 2000, "num_room": 5, "product_type": "OwnerOccupier"})
```

```
The predicted property price is 8837248 rubles (i.e. $203256 AUD)
```

Figure 6. Example use cases of the property price predictor algorithm showing that the multilayer perceptron learnt meaningful information regarding the build year of a property.

Lastly, we were interested in examining the predictive accuracy of the algorithm as the total number of actual attribute values provided increased. In order to observe this relationship, we firstly selected an arbitrary test sample from the test set. We then passed zero actual attribute values to the algorithm and used the trained multilayer perceptron to predict the property sale price of the feature vector filled with 286 attribute means from the training set. Next, we provided the algorithm with one of the actual attribute values from the selected test sample and predicted the property sale price again, this time with one actual attribute value and 285 training set attribute means. Following this, we used a feature vector

with two actual attribute values and 284 training set attribute means, so on and so forth, until we did the final property sale price prediction on the feature vector containing all actual 286 values from the selected test sample. We expected that as the number of actual values provided to the network increased, the predicted sale price would become closer and closer to the actual target value of the test sample. However, the most common pattern we observed was similar to the one displayed within *Figure 7*, where the predictions would oscillate around the actual target value and eventually converge near the actual price, or simply stay at a dissimilar price. These patterns showed that both individual attribute values, and collections of attribute values can significantly impact the multilayer perceptron's predictions.

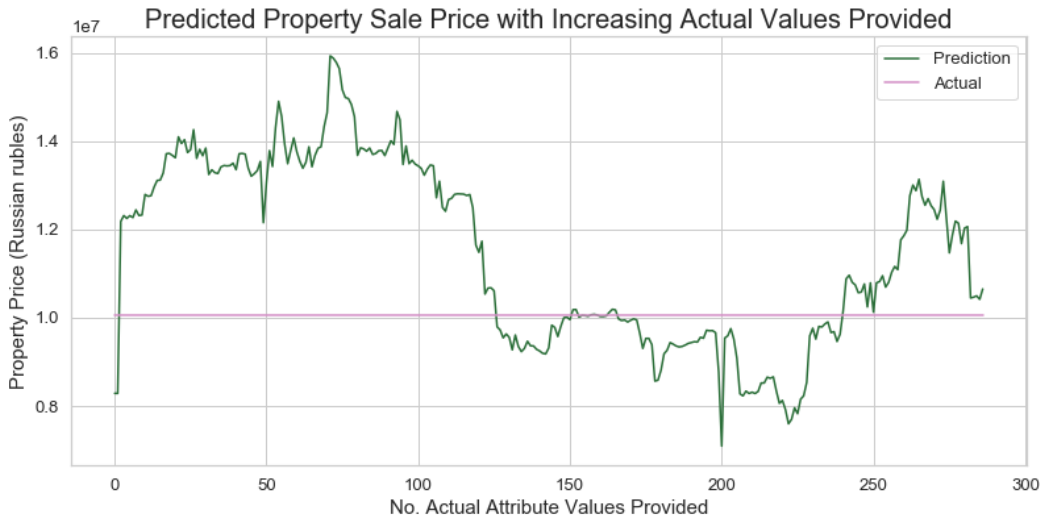

Figure 7. The property price predictor algorithm predicting the sale price of a test sample property as the number of actual attribute values provided from the test sample feature vector increases.

3.2. Exponential Smoothing

Now we provide results and compare the StatsModels and R Holt Winters library solutions with our manual implementation on the Russian Housing Market Dataset. To compare methods, we average predictions and then take the mean absolute error and mean square error.

Optimising Seasonal Period

Because neither of our library implementations optimise for the most appropriate seasonal period that best fits our data. We start by calculating the MSE of the whole time series using StatsModels for each seasonal period and simply find the one which produces the minimum error. This guides in selecting the seasonal period for all of our implementations for all further analysis. Each of these error values are also stored used within our results and metrics section.

Figure 8. The StatsModels library implementation of Holt-Winters triple exponential smoothing forecasting algorithm used for optimising the seasonal period.

Manual Implementation Component Decomposition

When computing our fitted values, we combine each of the exponential smoothing components in order to model increasing complex characteristics of our time series. In our manual implementation we returned both the fitted values and the individual components. Decomposing our model allows us to see how each one contributes to accuracy of our fitted model. We can see that the level component is equivalent to a weighted moving average, placing the model at roughly the same height as the observations. Here the trend only has a very small influence on our model compared to level and seasonality, slightly nudging it in a general direction. Seasonality on the other hand has a strong influence on our models' predictions. By looking our models' components, we can see what information lies within our data. A strong seasonal pattern indicates that Russian housing prices are therefore strongly influenced by the time of year that they're being sold in. Ranging from a minimum of -1, 000, 000 rubles near the start of the year to 500, 000 toward the end in fluctuations. Because our model has such a minuscule trend component we can see that the housing market is relatively stable over the period of our data. We notice that there are also oscillating cycles within the trend spanning periods of around 2 years.

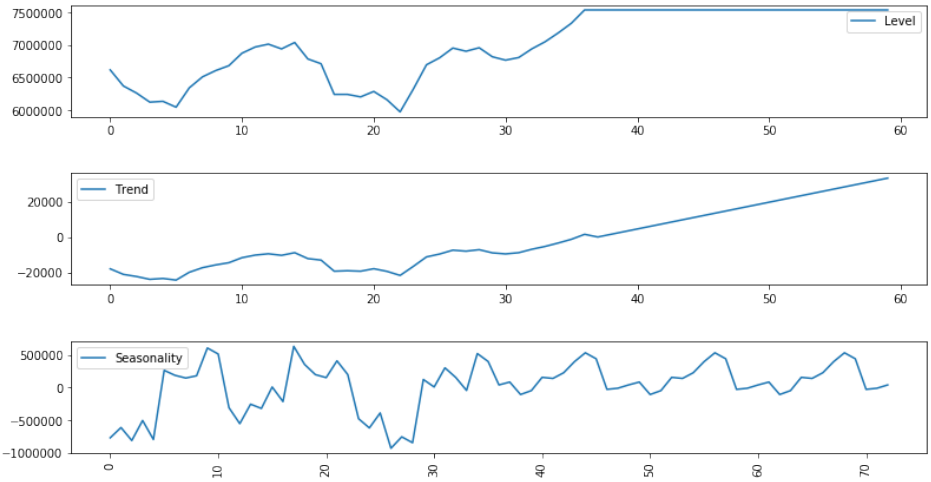

Figure 9. The manual implementation component decomposition figure.

Predictions and Forecasting of all Implementations

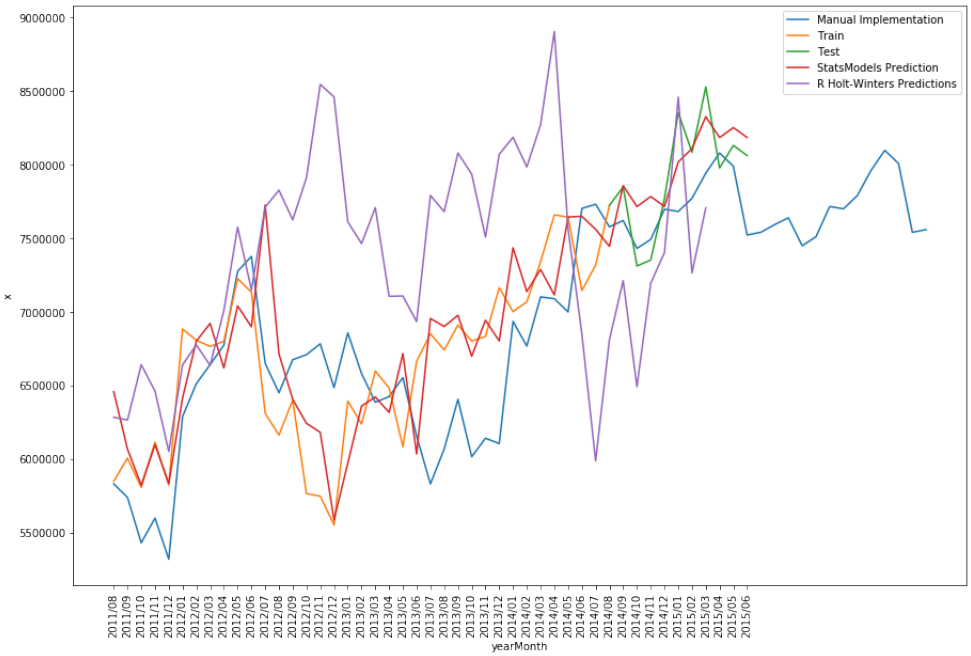

Figure 10. Predictions and forecasting of all implementations.

Now we examine the predictions and forecasts of each of our implementation alongside training and test sets. We're looking for the model that best fits the time series across both the training and test sets most closely. In our results we formalise the comparison between the implementations by using the MSE and MAE errors. For now, we make visual comparison from our own judgement. Firstly, the clearly worst performing implementation is R's Holt-Winters library. After it reaches the end of the first seasonal period it shows very inconsistent predictions which seem very sporadic and don't follow the sales price time series data. In contrast the StatsModels and manual implementation show far better predictive performance by closely fitting the sales price consistently. This provides proof that the Holt Winters exponential smoothing model is sufficiently sophisticated and can effectively be applied to the Russian housing market domain.

Error Comparison

Below we compare the implementations by calculating the MSE and MAE errors, again for seasonal periods from 2 to 17 months but with our manual implementation. Error for our R Holt-Winter library implementation are exported to a csv from our R Forecasting notebook and read into a data frame so that we plot all implementations together in a single graph.

Both StatsModels and our manual implementation produce reasonable results. Generally, our manual implementation consistently has a lower error across all seasonal periods while StatsModels produces the minimum error on all seasonal periods from 2 to 8 and 12 months. In all of our implementations where're using both and additive trend and seasonal components without dampening equations. Therefore, the difference between the implementations may come down how they calculate their initial component values and the slight variations in how each component is calculated. Our manual implementation uses the original holt-winters equations and thus the library solutions may use other variations that have come about since.

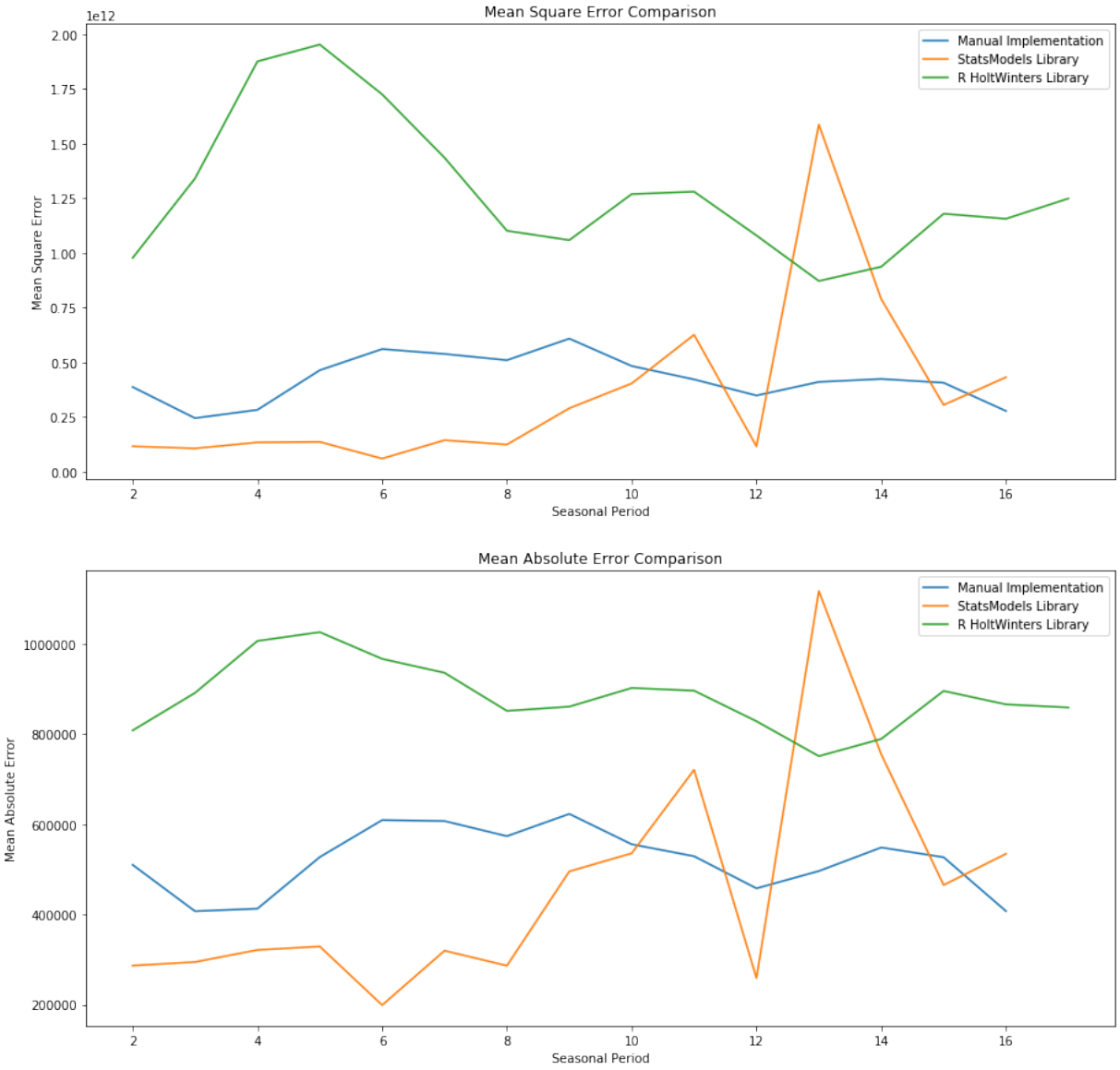

Figure 11. MSE and MAE error metric for comparing implementations across seasonal periods.

Evaluation on Rolling Forecasting Origin

In order to determine with confidence how well our models perform on novel data we need to evaluate it across more training and test sets. In order to accomplish this without extending our dataset we can use a time series cross validation technique known as a rolling forecast origin. We start with a small training set and a fixed forecast step size. We roll the forecast origin through time, increasing the size of the training set. Once we've reached the end of the dataset we average the mean absolute error from all of the test sets. One thing to note is that our training set must have a minimum size threshold since our forecasts are based on prior observations and accurate models cannot be produced from small training sets.

Here we iterate through the dataset, increasing the forecast origin by one step and calculate the mean absolute error between our forecasted results in the data's test set. The test set size is fixed so that we're always analysing how the model performs at a set forecasting horizon. This is because changing the horizon will lead to inconclusive results because make forecasts further into the future leads to

increasingly unreliable results. The idea of having a fixed test set size can be thought of as a sliding window that we move across our time series. At each step of moving the forecasting origin forward we also increase the size of the training dataset. By providing more data to the algorithm they should be able to better optimise their components to fit the underlying time series model and utilise more information contained with historical data to make better predictions of the future. Once we've cross validated and calculated errors across a larger portion of the dataset we're given a better idea of the algorithms ability to model our time series. After which we take the average of the mean absolute errors so that we may compare implementations with a single value.

Finally, we're expecting the error to reduce as the forecasting average increases along with the average cross validation error. Our findings show that indeed both the StatsModels library and manual implementation have a decreasing error. Unexpectedly, the R Holt Winters implementation has the opposite trend. This may be the result of an error in our implementation or simply a difference in implementation of the library itself. The diagram below shows the training set in blue and how far to forecast ahead in red with the test set being the remaining points.

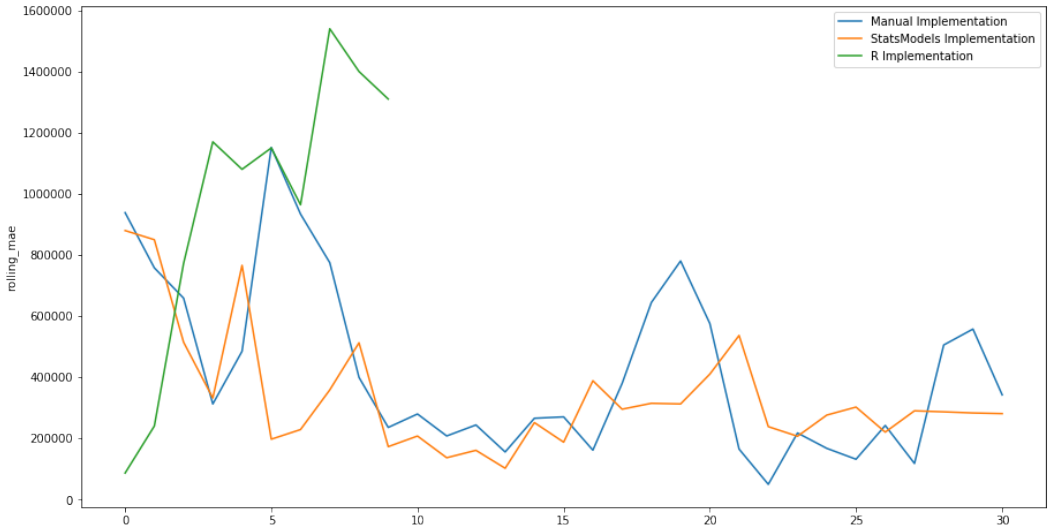

Figure 12. Rolling forecast origin.

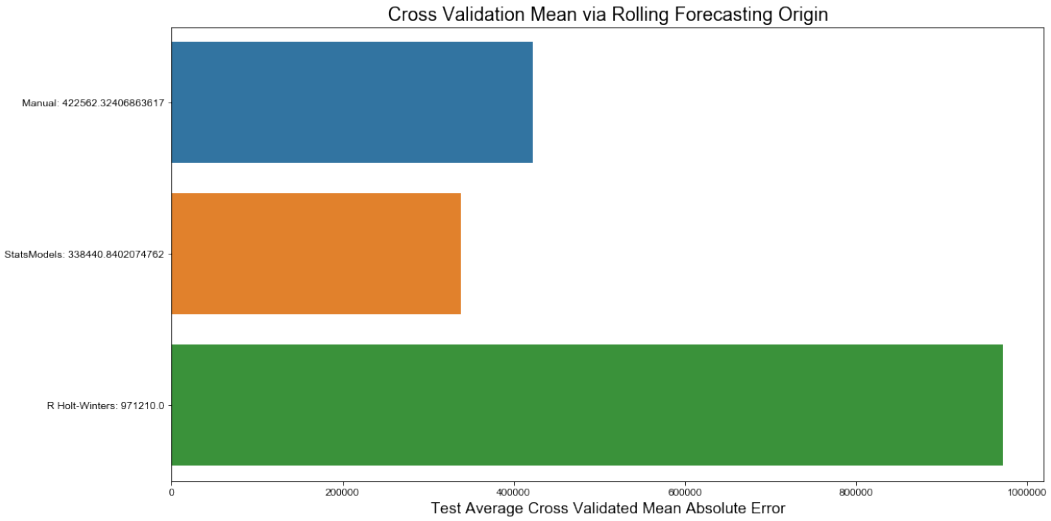

Figure 13. Mean of cross validation.

Runtime Comparison

An important metric for evaluating an implementation of an algorithm is the time it takes to execute. This is because algorithms may be time sensitive, run at a certain frequency or in the case of data mining be scalable as data sets become increasingly large. For time series forecasting an example maybe making accurate stock market predictions in a timely manner so that you outcompete other traders. Our library solutions performed similarly as R HoltWinters executed within 0.01 second ahead of StatsModels at 0.01326 seconds. To our surprise the manual implementation by far performed the best at 0.00579 seconds coupled with close results to StatsModels accuracy. This makes it the most reliable if it were to be applied in a time critical application.

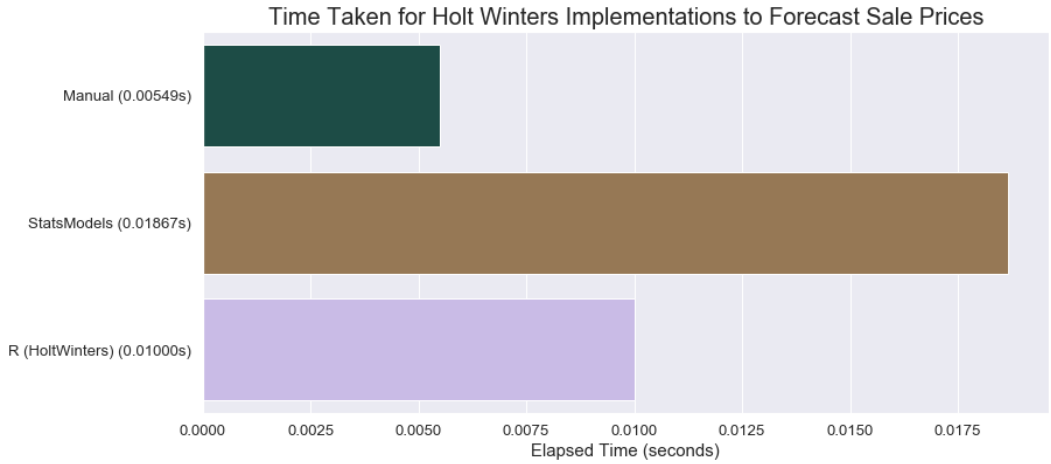

Figure 14. Runtime comparison of various implementations.

Analysing Residuals

Our fitted values are those which use observations from the time series to make predictions, combining our different characteristic components. Only values which don't involve observations are forecasts. After removing our fitted model by taking the difference with our observations we're left with what's called the residuals. This should be the remaining noise within the data that isn't a true reflection of the data generating mechanism. Attempting to model it will only lead to less generalised performance. For this to be the case, our residuals must be uncorrelated otherwise there's remaining information that should be used in our model and they must have a mean of zero otherwise our forecast is biased. Solving the bias problem is as simple adding the mean of the residuals to the forecasts.

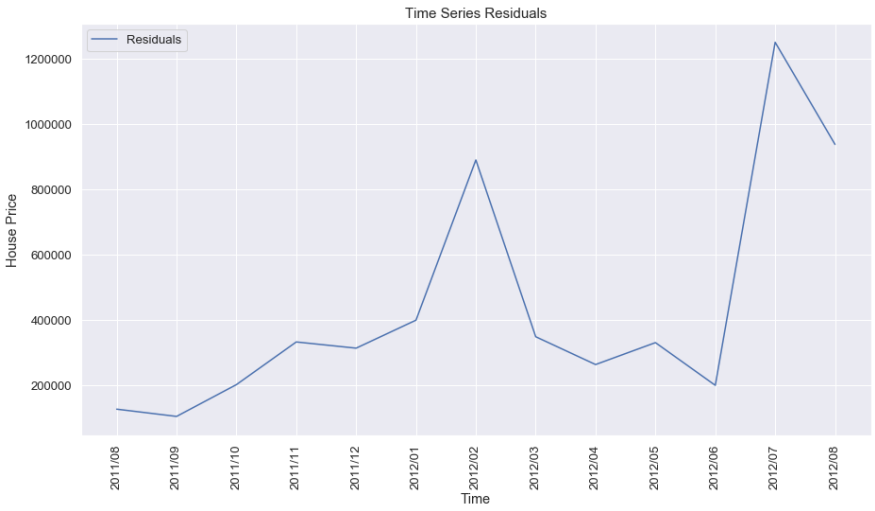

Figure 15. Residuals of model predictions and observations.

3.3. Naïve Bayes Classifier

In this study we used multiple variants of the Naïve Bayes Classifier to predict the sub-areas (districts) of properties in Russia based upon property and neighbourhood attributes. We firstly conducted experiments to assess the algorithm's effectiveness within the housing market domain. Secondly, we studied and compared the performance of the different variants of the classifier, as well as the different implementations (those being the two library implementations using Scikit-learn and NLTK, and our own). Lastly, we implemented a district predictor algorithm to evaluate the potential practical applications of this algorithm within the Russian housing market domain.

We used a subset of 18 attributes to represent the feature vector for each data sample within this study. This was due to the fact that the time taken to predict data samples with many attributes took rather long for the manual implementations of the algorithm. Consequently, we explored the effects of using fewer attributes for the task of classification. We also note that the manual implementations were validated for correctness by testing the classifiers on known solutions from the 3804ICT lecture slides. We refer the reader to the notebook for more information in regard to these validation tests.

Within this study we implemented different variants of the Naïve Bayes Classifier. Thus, we briefly introduce them here. The Gaussian Naïve Bayes Classifier assumes that the attribute's values are sampled from a Gaussian distribution, the Categorical/Multinomial Naïve Bayes Classifier assumes the attributes are categorical/nominal, and the Hybrid Naïve Bayes Classifier calculates the likelihood based upon the Gaussian Naïve Bayes Classifier for numerical/continuous valued attributes, and for categorical/nominal attributes it calculates the likelihood based upon the Categorical/Multinomial Naïve Bayes Classifier. We refer the reader to the notebook for a more detailed explanation of these variants.

Dataset Experimentation and Algorithm Implementation Comparison

For our initial experiments within this study we examined the accuracy, unweighted average precision and unweighted average recall of each of the classifiers on the test set. We found that there was no classifier nor classifier variant that outperformed the rest on all evaluation measures. However, the manual implementation of the Gaussian Naïve Bayes Classifier appeared to perform the best based upon the metrics used, ranking second best in accuracy and first best in average precision and average recall. We highlight the results of this classifier in *Figure 16*.

	Accuracy	**Avg. Precision**	**Avg. Recall**
Scikit-learn – Gaussian	0.380	0.255	0.233
Scikit-learn – Multinomial	0.240	0.144	0.122
NLTK – Multinomial	**0.480**	0.299	0.286
Manual – Gaussian	0.460	**0.339**	**0.341**
Manual – Multinomial	0.420	0.241	0.280
Manual – Hybrid	0.340	0.197	0.220

Figure 16. Accuracy, unweighted average precision and unweighted average recall for each classifier on the test set.

The overall performance of the various classifiers reported in *Figure 16* highlights that the Naïve Bayes Classifiers did not perform very well at the task of predicting Russian districts based upon information about the property and the surrounding area. We hypothesised that this was due to the small number of attributes being used to represent data samples within the training and test sets. Hence, we explored this further within the subsequent investigation.

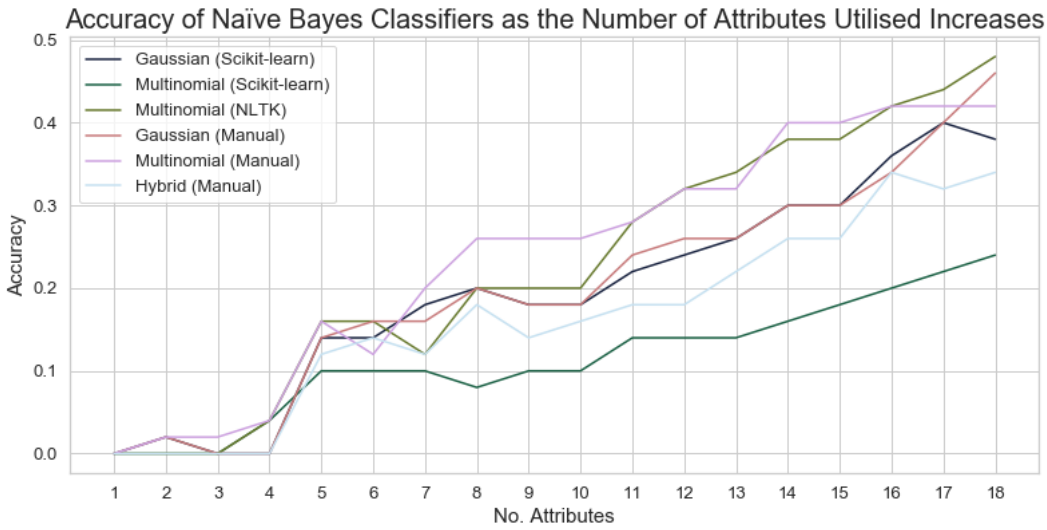

Figure 17. Accuracy of classifiers as the number of attributes used in the training and test set increases.

The experiment presented in *Figure 17* was conducted to determine whether the previously stated hypothesis, regarding whether the poor performance of the classifiers was due to a small number of attributes being used to represent data samples, was correct. We found that as the number of attributes decreased, the accuracy of the classifiers on the test set also decreased, and in turn, as the number of attributes increased, the accuracy of the classifiers on the test set also increased. The general trend of this relationship appeared to be linear (i.e. there was a linear relationship in the results showing that as the number of attributes increases, the accuracy of the classifier also increases). These results supported our hypothesis. Based upon this analysis, we believe that utilising the complete 286 attributes used within the multilayer perceptron study would allow these Naïve Bayes Classifiers to achieve a significant increase in classification accuracy, average precision and average recall. However, as stated previously, due to testing times on the manual implementations taking a long amount of time, we will not be able to explore the performance of these classifiers with larger numbers of attributes in the data samples. Nevertheless, the findings reported above were insightful and the investigation provided an explanation for the performance of the classifiers.

Our subsequent investigation examined the efficiency of the various Naïve Bayes Classifiers. Here we defined efficiency as the total time to construct and use the classifiers. Thus, the longer the time taken the worse the efficiency. The results are shown in *Figure 18*. As evident in the graph, the library solutions, especially the Scikit-learn implementations, were significantly more efficient than our own implementations. Although our manually implemented Gaussian Naïve Bayes Classifier appeared to perform the best from the evaluation measures shown within *Figure 16*, we can appreciate that the efficiency of the algorithm makes it almost incomparable to the highly optimised library implementations. Because of the testing restrictions brought upon by this poor efficiency, we were unable to experiment with larger test sets nor with more attributes within the data sample feature vectors. Since the library solutions would be efficient enough to handle the 286 attributes used within the multilayer perceptron study, we assume that they would be able to outperform our Gaussian implementation in the previously discussed evaluation measures by simply utilising more attributes. This analysis outlined the importance of efficiency and code optimization in data mining algorithms.

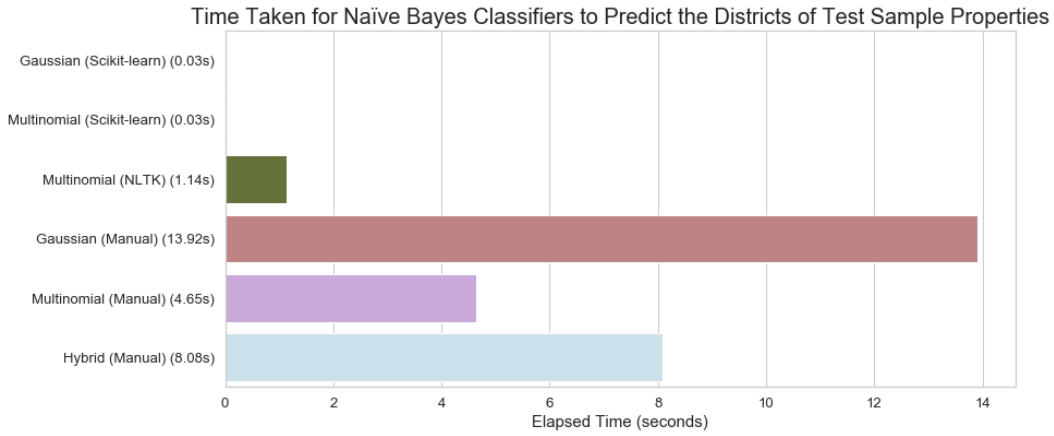

Figure 18. Efficiency of classifiers on the complete test set (i.e. all 18 selected attributes).

We decided to dive deeper into investigating the efficiency of the implementations by assessing the scalability of the classifiers. For this study, we defined scalability as the efficiency of a model as data size increases. We refer the reader to the notebook for a more in-depth explanation of this metric. We firstly examined scalability as the training set size increased. *Figure 19* presents the results of this investigation. We found that the slope in efficiency for all the implementations was rather small (i.e. the efficiency of the classifiers only slightly worsened as the number of training samples increased). This suggested that the implementations were quite scalable in regard to increases in the training set size.

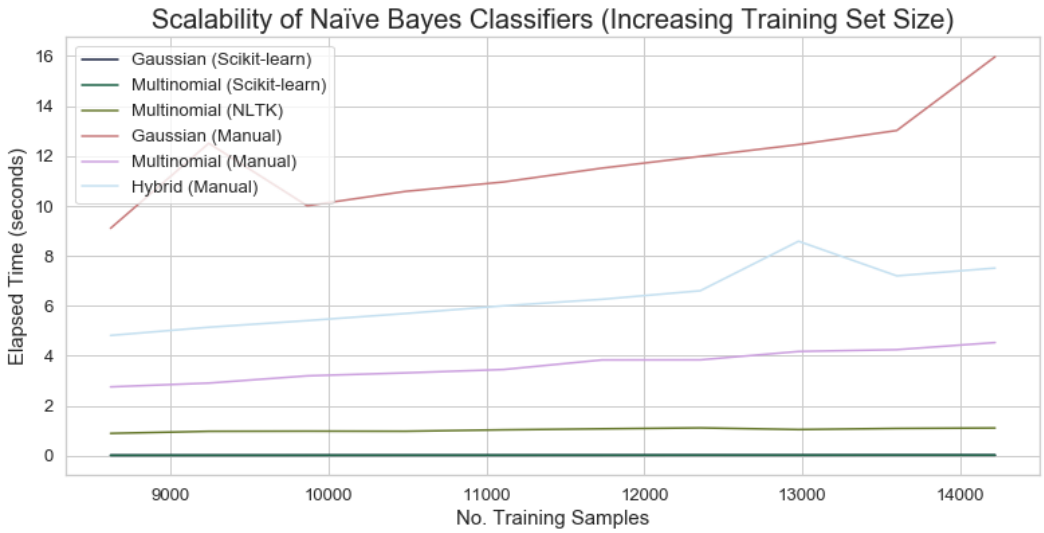

Figure 19. Investigating the scalability of classifiers by measuring efficiency (i.e. time taken to construct and use the model) as the training dataset increased in size.

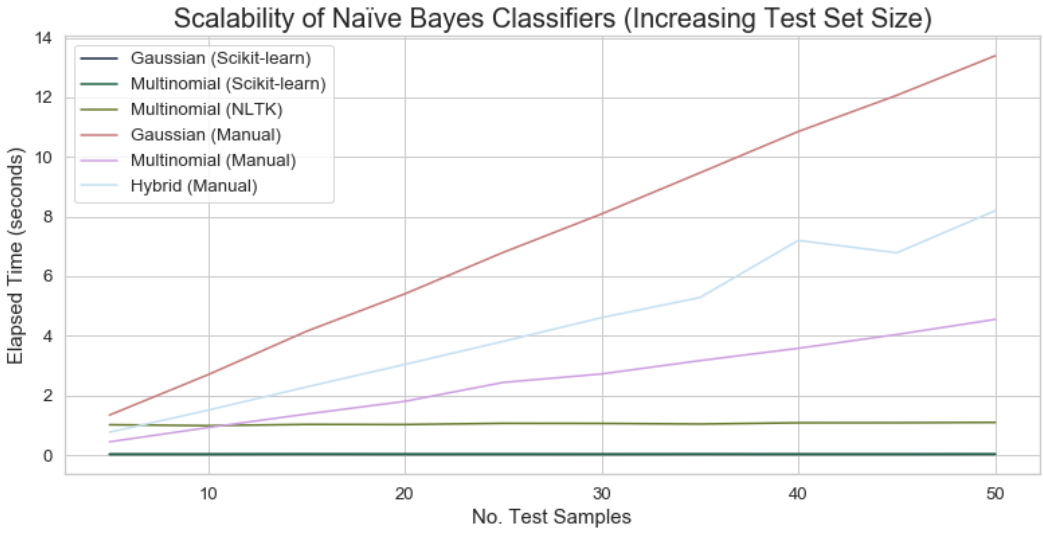

Figure 20. Investigating the scalability of classifiers by measuring efficiency as the test dataset increased in size.

To further investigate the scalability of the classifiers, we decided to assess scalability as test set size increases. The findings of this analysis are shown in *Figure 20*. We found that the library solutions had an efficiency slope of approximately zero, whereas our implementations appeared to follow a linear time complexity with respect to the number of test samples used, having increasingly larger efficiency slopes for the multinomial, hybrid and Gaussian variants. Consequently, we concluded that our implementations were not very scalable in regard to test set size increases. This explained why testing our implementations on large test sets, or on data samples with many attributes, would take a significant amount of time. Furthermore, this analysis also provided us with interesting insights into where we could focus on improving our implementations of the Naïve Bayes Classifiers.

Practical Application for the Naïve Bayes Classifier within the Housing Market Domain

The final investigation of this study looked at how the Naïve Bayes Classifier could be applied to a practical application within the housing market domain. Since the classifier is able to take property and neighbourhood characteristics as input and output the most probable districts based upon those characteristics, we thought that a potential application for the algorithm could be recommending districts to home buyers or businesses looking for an ideal sub-area in Russia. With such an algorithm, home buyers could find districts with certain property and neighbourhood characteristics that are ideal for themselves and/or their families, while businesses could use the algorithm to determine where to build their next office space, or where to start their next advertisement campaign. The users of a website or application that uses this kind of algorithm would not want to specify values for all the 18 (or however many) attributes required as input into the classifier. Consequently, we decided to implement an algorithm for predicting districts based upon only some information about the properties and the surrounding area.

For the implementation of this algorithm, we used the manual implementation of the Gaussian Naïve Bayes Classifier. In order to provide district predictions, we would collect and pre-process the user inputs for particular attributes and add them to a feature vector. For the remaining attributes that were not specified, we would determine the training set mode of the attribute and use that value in the feature vector. With a complete feature vector, we then used the training set and the Gaussian Naïve Bayes Classifier to predict the top n most probable districts given the provided information. We would then

return the results to the user after transforming the predicted districts from encoded labels back into strings representing the district names. We refer the reader to the notebook for more information regarding the implementation of the district predictor algorithm.

```
# Example use case 2
district_predictor(predictor, {"life_sq": 50, "build_year": 2000, "railroad_terminal_raion": "yes"})
```

```
Districts with property and neighbourhood characteristics most similar to those you specified:
  1. Mar'ina Roshha
  2. Butyrskoe
  3. Krasnosel'skoe
```

```
# Example use case 3
district_predictor(predictor, {"life_sq": 100, "build_year": 2000, "railroad_terminal_raion": "yes"})
```

```
Districts with property and neighbourhood characteristics most similar to those you specified:
  1. Krasnosel'skoe
  2. Mar'ina Roshha
  3. Butyrskoe
```

Figure 21. Example use cases of the district predictor algorithm showing that the Naïve Bayes Classifier gave interesting district predictions based upon the specified user input.

Figure 21 provides some example use cases of the district predictor algorithm. We observed that the Gaussian Naïve Bayes Classifier was able to make some interesting predictions based upon the provided characteristics. For example, in *Figure 21*, the most likely district for homes that have a total living area of 50 square meters, that have been built in the year 2000, and that are within a district that has a railroad terminal, is "Mar'ina Roshha". However, when we change the total living area to 100 square meters, and keep the other characteristic the same, we see that the most likely district is instead "Krasnosel'skoe", and "Mar'ina Roshha" becomes the second most probable district. From this observation, we can assume that there are most probably larger homes in Krasnosel'skoe in comparison to Mar'ina Roshha, and besides this difference they are relatively similar districts. For a home buyer, they could use this information to decide where they should buy a home, perhaps in Krasnosel'skoe if they were going to live with somebody else, or in Mar'ina Roshha if they were going to be living on their own.

We conducted one last experiment to examine how accurate the Gaussian Naïve Bayes Classifier's district predictions were as the number of actual values provided in the input feature vector increased. This was a way to measure how accurate the predictions of the classifier would be when fewer attributes were specified by the user (and thus when more attribute modes were used within the feature vector). Since this classifier is a statistical classifier, we assumed that as the number of actual attribute values provided increased, the most probable district would converge towards the actual target district. *Figure 22* displays these results for a particular test sample. We observed that the classifier converged on the actual district after it was provided with a total of 14 actual attribute values. The classifier then continued to predict the actual target district as it was provided with additional actual attribute values from the test sample feature vector. We observed that the top three district predictions of the classifier appeared to oscillate around the actual target district for this test sample. However, since the district attribute is nominal, this oscillating behaviour, and moreover the "closeness" to the actual target district, does not hold any significance. We hypothesis that if we were able to determine the similarity between each of the districts through a method such as clustering, we could then use this similarity metric to order the districts such that the oscillations and "closeness" in the predictions within *Figure 22* become meaningful. However, this was outside the scope of the investigation so it was not investigated. Nevertheless, we observed that in most test samples the district that was most probable given the provided property and neighbourhood characteristics ended up converging on the actual target district of the test sample. We assume that the second and third most probable districts were similar to the first.

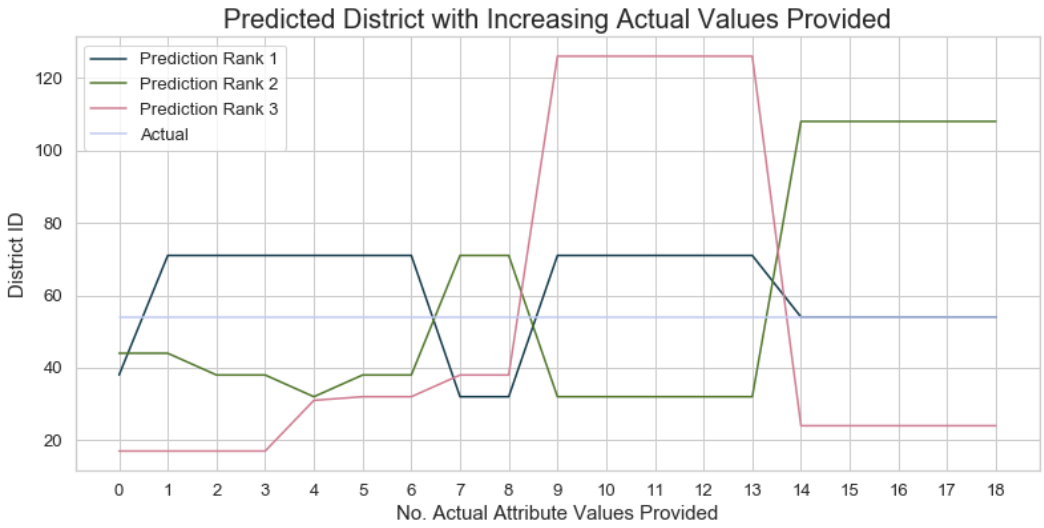

Figure 22. The district predictor algorithm predicting the district of a test sample property as the number of actual attribute values provided from the test sample feature vector increases.

3.4. FP-Growth

Here we conducted a study on mining frequent patterns within the domain of the Russian housing market. In order to mine such frequent patterns and find interesting relationships among properties and neighbourhoods in Russia, we used the FP-Growth algorithm. Our first investigation for the study involved finding frequent patterns within the Sberbank Russian housing market dataset, and comparing the various implementations of the algorithm (i.e. two library implementations and our own). Following this, we performed a qualitative analysis on the found frequent patterns to judge their interestingness.

We note that for *Figure 23* and *Figure 24*, we used 5000 property entries from the Russian housing market dataset to mine frequent patterns. Moreover, we used a minimum support count threshold of 4000 (i.e. a minimum support threshold of 0.8) to determine whether or not a pattern was frequent.

Dataset Experimentation and Algorithm Implementation Comparison

Within this section of the study, we investigated the effectiveness of applying the FP-Growth algorithm to the housing market domain in order to discover useful patterns. Furthermore, we compared the two library implementations of the FP-Growth algorithm as well as our own.

Our first analysis examined the total number of frequent patterns each implementation of the algorithm found. The results of this analysis are presented in *Figure 23*. We found that the two library implementations of the FP-Growth algorithm found the same total number of frequent patterns, whereas our manual implementation of the algorithm found approximately half the amount of the library implementations. Since the total number of patterns found by the library implementations were so similar, we assume that our manual implementation of the FP-Growth algorithm had some faults. Nevertheless, we found that it was still able to find a relatively large number of frequent patterns within the provided property entries. We further tested the manual implementation on a dataset from the 3804ICT Lecture 5 lecture slides to determine where it was missing frequent patterns. We found that the main FP-tree constructed by the manual implementation was correct with respect to the provided solution. However, there were some errors in the conditional FP-trees that were constructed for frequent 1-characteristic-sets. As a result of our project encompassing the implementation of five algorithms, we

did not have sufficient time to fix the manual implementation of the FP-Growth algorithm. Consequently, for this study we left the implementation as it was and left fixing and perfecting the algorithm for future work.

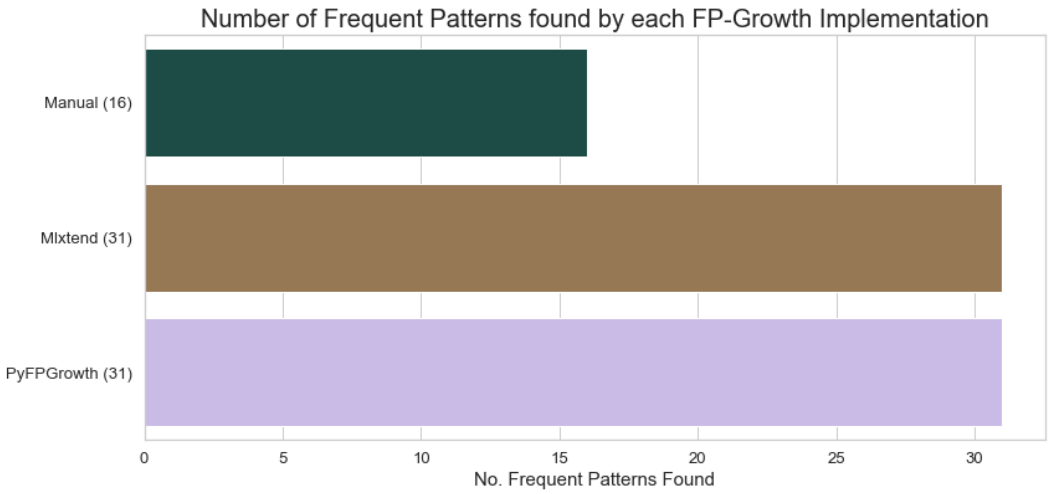

Figure 23. The number of frequent patterns found by each of the FP-Growth implementations.

The results presented in *Figure 23* also show that there are useful patterns present within the Russian housing market dataset which can be mined. Based upon these findings, we decided to explore the interestingness and usefulness of the found patterns further in an additional investigation. This additional investigation is discussed below within *Interestingness of Found Frequent Patterns*.

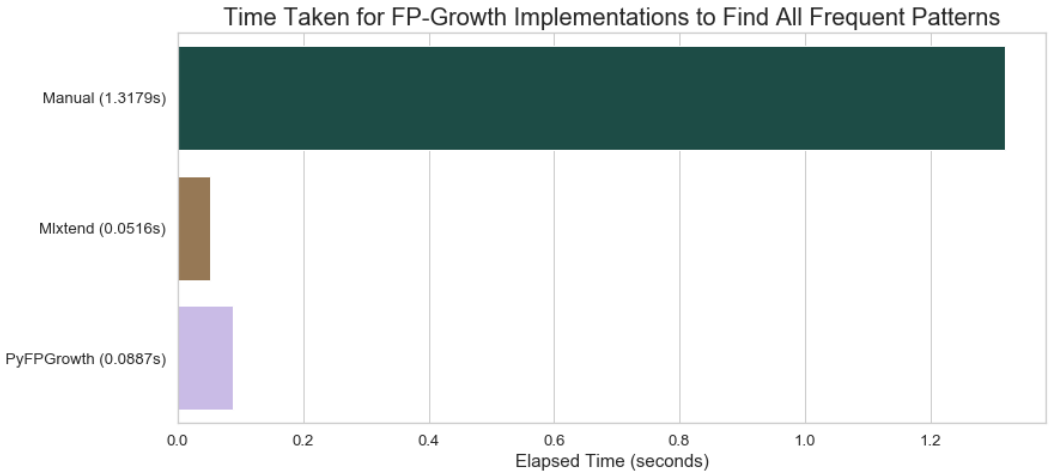

Figure 24. Total time taken in seconds for each implementation to find all frequent patterns.

Following this investigation on the number of frequent patterns found by each of the implementations, we explored the time efficiency of the implementations of the FP-Growth algorithm. The results of this analysis are shown in *Figure 24* We found that the manual implementation of the algorithm took significantly more time to find frequent patterns in comparison to the library implementations. Thus, the manual implementation was found to be less time efficient than the library implementations. There

appeared to be no significant difference in time taken to find frequent patterns between the two library implementations Mlxtend and PyFPGrowth. In frequent pattern mining, we are generally concerned with finding patterns within datasets that contain significantly large numbers of data samples. Consequently, the time efficiency of an algorithm is critical when judging how practical it is for a given task. We can therefore conclude that our implementation of the algorithm would need to be optimized for improvements in regard to correctness and time efficiency.

Interestingness of Found Frequent Patterns

In this investigation we qualitatively examined the interestingness and usefulness of frequent patterns found within the Russian housing market dataset. We note that for this investigation we used 5000 property entries from the Russian housing market dataset to mine frequent patterns. Furthermore, we used a minimum support count threshold of 3000 (i.e. a minimum support threshold of 0.6) initially to determine whether or not a pattern was frequent, and then we used a minimum support count threshold of 2000 (i.e. a minimum support threshold of 0.4) to explore what additional frequent patterns would be found. We present some of the interesting frequent patterns that were found in *Figure 25*.

Minimum Support Threshold	Interesting Frequent Patterns
0.6	['full_sq_Low', 'life_sq_Low', 'floor_Low', 'max_floor_Low', **'product_type_Investment'**] ['full_sq_Low', 'life_sq_Low', 'floor_Low', 'max_floor_Low', **'material_1.0'**]
0.4	[**'radiation_raion'**, 'product_type_Investment'] ['full_sq_Low', 'life_sq_Low', 'floor_Low', 'max_floor_Low', **'radiation_raion'**, 'product_type_Investment']

Figure 25. Interesting frequent patterns found by mining the Russian housing market dataset with minimum support thresholds of 0.6 and 0.4.

We found that the attribute class values “full_sq_Low”, “life_sq_Low”, “floor_Low” and “max_floor_Low” appeared frequently within the dataset. We expected the occurrence relationship between “full_sq_Low” (total area in square meters) and “life_sq_Low” (living area in square meters) to be quite strong as these attributes are generally similar for a property. Moreover, we also expected that “floor_Low” (floor of the building) and “max_floor_Low” (number of floors in the building) would appear frequently together as apartment buildings with low maximum floors would in turn have more apartments on lower floors. However, the fact that “full_sq_Low”, “life_sq_Low”, “floor_Low” and “max_floor_Low” all appeared frequently together within the dataset implies that the majority of apartment buildings within Russia (based upon the context of this dataset) do not have many floors and are relatively small in size. Moreover, we found that with a minimum support threshold of 0.6, these four attribute class values also occurred frequently with the attribute class values “product_type_Investment” and “material_1.0”. These frequent patterns imply that the majority of apartments within Russia are investment properties. Moreover, these frequent patterns imply that the most common material used to build such apartments is panel (i.e. material 1.0 within the dataset). We

decided to relax the minimum support threshold to 0.4 in order explore additional frequent patterns which were still frequent within the dataset, but not as frequent as those found at minimum support thresholds of 0.6 and above. The most interesting frequent patterns we observed are presented in *Figure 25*. We found that the attribute class value "radiation_raion" and "product_type_Investment" occurred frequently together. The attribute class value "radiation_raion" means that the property is within a district where radioactive waste disposal is present. This finding is both insightful and interesting, as it shows us that a significant number of investment properties in Russia are within districts with radioactive waste disposal. The second interesting pattern reported for a minimum support threshold of 0.4 tells us that these investment properties within districts with radioactive waste disposal are again most probably apartments. Based upon these interesting frequent patterns that we reported, our conclusion for this analysis is that frequent patterns can provide us with insightful and useful information within the housing market domain.

3.5. K-Means

Within this study we use the clustering algorithm K-Means Clustering to discover interesting patterns and relationships between two or more attributes of properties in the Russian housing market. Our goal was to discover the not so obvious connections between attributes which could be used to classify the motive behind the purchase of the property, such as rental, short term flipping, family home etc. Our methodology to construct interesting clusters from the dataset was to focus firstly on the clustering tendency, then the optimal number of clusters k, followed by finally the quality of the clusters themselves.

To pick the attributes which we would explore with our implementations, we used the clustering tendency of the attribute. The most common way to calculate this is the Hopkin's statistic. This quantity describes how close a dataset of any dimension follows a uniformly random distribution. Further explanation of how the statistic is calculated can be found in the notebook. We consider around the top 50 attributes based off this metric and take the highest, removing any redundancy such as considering both cafe_count_5000 and cafe_count_3000. The function was only performed on numerical attributes.

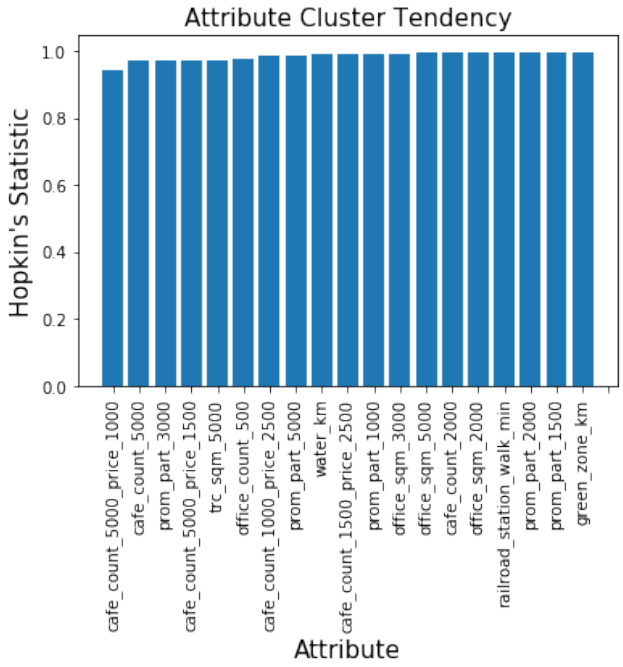

Figure 26. Hopkin's Statistic for Database Attributes

As shown in Figure 26, most statistics returned were very close to 1. This is an indication that there will be few significant clusters in the data, but further metrics will be explored to attempt to discover any patterns.

To commence the comparative analysis and validate our implementation of the k-means algorithm we generated some fake data with 3 dimensions, containing 3 clusters which are visually distinguishable and obvious when graphed. We performed all 3 implementations of the algorithm on this dummy set to validate the output of each, and tune the parameters of the libraries. Figure 27 demonstrates that all implementations were successful in clustering this basic dataset.

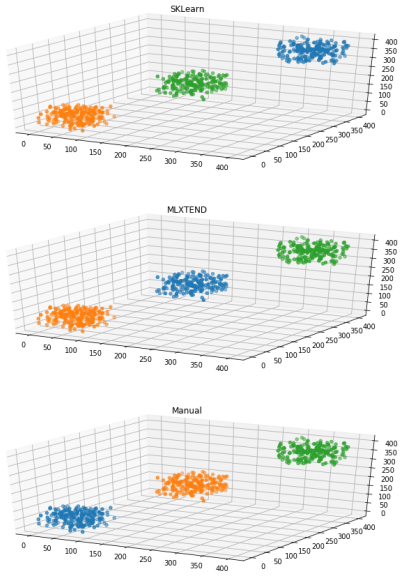

Figure 27. 3D Validation of Clustering

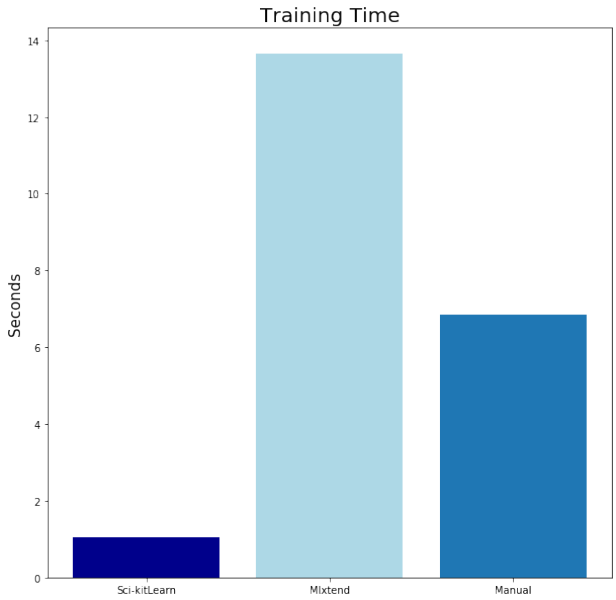

Figure 28. Comparative Analysis of Clustering Time

Continuing with comparative analysis we tested the training time required for each implementation to assign all points in a dataset to their clusters. The results of this analysis are presented in *Figure 28*. For this test each implementation was given the same two attributes from the normalised dataset, tasked to cluster for k = 3. As shown below there was a significant difference in performance from each. Scikit-Learn was the most efficient by far, finishing around 6 seconds faster than manual, and around 12 seconds faster than Mlxtend. A possible reason Mlxtend performed so slowly is that it accepts a parameter max_iter, which prevents further attempts to converge on a solution once that number of iterations has occurred. Having this value at the default of 10 or 100 result in terrible clustering, with it regularly failing the validation test (Figure 27). Setting this to be 1000 provides reliable results, but may cause the algorithm to be performing additional iterations. Whether or not the extra search time produces higher quality clusters will be explored in cluster quality exploration.

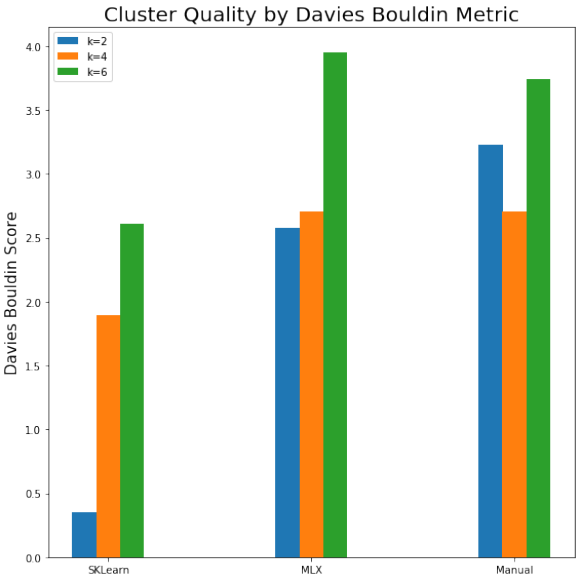

Figure 29. Davies Bouldin Score

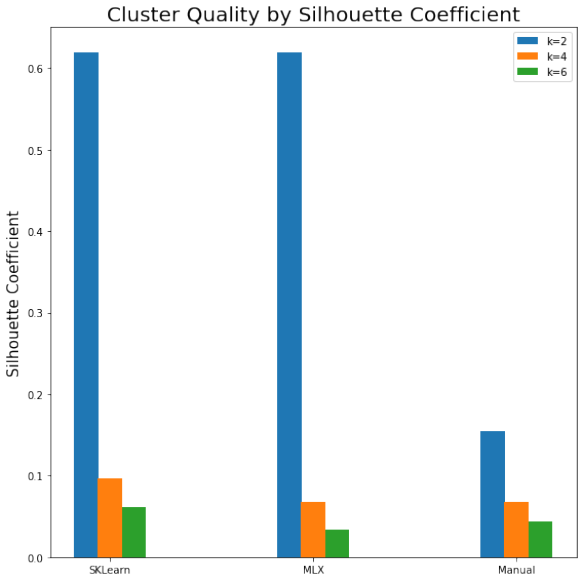

Figure 30. Silhouette Coefficient

To quantify the quality of clusters without any ground truth's the cluster model must be used to assess itself. The key metrics which describe good clusters are that the points within a cluster are dense, and very similar to each other, while also being strongly dissimilar to points within other clusters. The Davies Bouldin score quantifies this by considering the ratio between the average distance between points in 2 clusters, and the distance between cluster centres of the clusters. Seeing as we want a small average distance between points in a cluster, and we want large distances between different cluster centres, a low Davies Bouldin score represents good clustering.

The silhouette coefficient works similarly. It quantifies these metrics by considering the ratio of points being dissimilar to the other points in its cluster, and the points being similar to the points in the second-best fitting cluster for that point. Therefore, silhouette coefficients close to 1 represent a well clustered point, coefficients close to 0 represent points which could be moved to another cluster with little effect, and coefficients close to -1 represent points which should be moved to their neighbour cluster. Further explanations for both scoring functions can be found within the notebook.

Figure 29 demonstrates that Scikit-Learn again performs significantly better for every k value over Mlxtend and our manual implementation. Interestingly Mlxtend performs slightly better or equal to the Manual implementation for lower k values, but as k increases, our manual algorithm creates equal results at k = 4, and then superior results at k = 6.

Scikit-Learn also performed the best observing the Silhouette Coefficient (Figure 30). In contrast to the DB results, Mlxtend performed equally to Scikit-Learn for k = 2. A reason for this could be that for the first instance Mlxtend got caught in a local minimum with lower quality clusters. For k = 4 and k = 6 all implementations performed comparably bad, which is likely a reflection of the dataset. Due to the speed and cluster quality advantages that Scikit-Learn has demonstrated we decided to use it for exploring the dataset.

Now we have metrics for attributes with high cluster tendency and cluster quality. To find interesting clusters we first set k to 2, and find the clusters for all permutations of 2 attributes. We then calculate the Davies Bouldin score for each set of clusters, and return the 10 clusters with the highest quality metrics. We repeat this process for values of k up to 8. The table below (Figure 31) describes the output of this function for k = 2.

Attribute 1	Attribute 2	Davies-Bouldin Score
1	13	1.4615
1	3	1.6408
2	11	1.8048
1	2	1.8505
2	13	1.8958
2	3	1.9631
3	13	1.9771
1	11	1.9816
2	4	1.9905
1	4	2.0135

Figure 31. Table of top 10 Davies-Bouldin Score for all 2D Clusters

To display these 2D clusters we graph the clusters which have the highest DB Score for each k value. These clusters for k = 2 to k = 4 are shown below in Figure 32. These results are successful in terms of the clustering algorithm making as much separation as possible. Where they lack is in the clustering tendency of the data as a whole. The shape of the data seems quite continuous, with the boundary lines seeming to arbitrarily cut the data vertically into k slices, so it is strange that the DB score has classified these as the highest quality clusters.

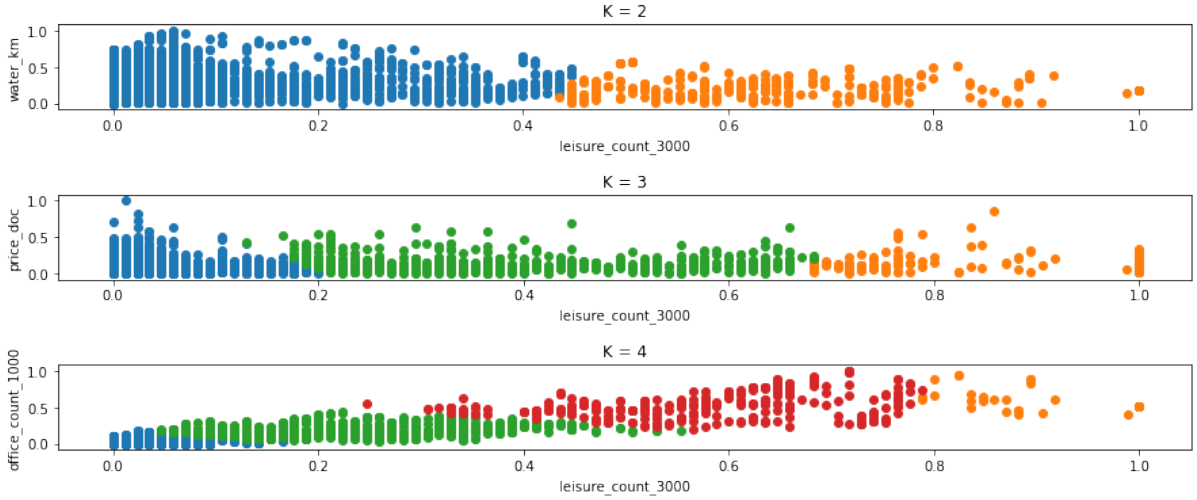

Figure 32. Plot of Highest Quality Clusters for Each K

To remedy this, we investigate whether a random restart will improve the discovery of more interesting clusters. To do so we performed the Scikit-Learn implementation on the same attributes with the random seed being changed at the start of each execution, to assign different starting cluster centres. Upon viewing the outputs and observing the cluster quality for these, we discovered that no changes to the output were achieved by including random restarts.

4. Summary

This section provides a summary of our project on discovering and utilising patterns in the Russian housing market for detailed analysis and prediction. Over five studies we investigated different applications of data mining algorithms within the Russian housing market domain. Collectively, we were able to show that data mining algorithms work within this domain and offer significant utility in a wide variety of applications.

In our first study, we used a multilayer perceptron to predict Russian house prices based upon information about the property and the surrounding area. The multilayer perceptron was able to learn how to accurately predict property prices. Our manual implementation of the algorithm achieved competitive predictive accuracy results with the two other library implementations (based upon the evaluation measures of mean squared error and mean absolute error), although the time taken to train the manual implementation was longer than that of the library implementations. In a hyperparameter investigation, we found that the larger the hidden layer size, the faster the network was able to converge to a local minimum in the error function, and the better the final predictive accuracy of the network. This suggested that the network may perform even better on the task with a larger amount of hidden layer neurons, or with an architecture consisting of multiple hidden layers. Regarding implementation, we were fascinated about the importance of weight initialization when training a network using the sigmoid activation function (we discussed this within the notebook). Moreover, we were amazed to find that the multilayer perceptron was able to learn the "general" meaning behind some of the property and neighbourhood attributes. In conclusion, this study showed that the algorithm has potential practical use within the housing market domain.

In our second study, we used the Holt-Winters exponential smoothing model to forecast Russian house prices. We thoroughly explored the Russian housing markets sale price time series which spanned from August 2011 to June 2016. To begin with we found the optimal seasonal period so that it could be used with all of our implementations. Then we began our manual implementation finding comparable results to the StatsModels library solution. In a separate notebook we explored R's implementation of triple exponential smoothing with the Holt-Winter library. Each implementations' performance was then formally compared by calculating the mean square and mean absolute errors revealing that R's Holt Winter's is far inferior. We also decomposed the manual implementations fitted values into their individual components which allowed us to uncover the housing markets price underlying characteristics. To more accurately determine the performance of our implementations we implemented cross validation via a rolling forecasting origin algorithm. This saw contradiction between R's holt-winters and the other implementations where the error increased as the training set become larger. Next, we wanted to determine the efficiency of each of the implementations whereby we calculated the runtime of each implementation. Impressively, our manual implementation saw a drastically lower runtime. Finally, we computed the residuals of our manual implementation finding that it had a comparably high positive bias which was solvable by taking the difference of the residual mean.

For our third study, we used multiple variants of the Naïve Bayes Classifier to predict sub-areas (districts) of properties in Russia based upon property and neighbourhood attributes. Our manual implementations of the classifiers were validated for correctness on known solutions. In the first investigation of the study, we observed that our manual implementation of the Gaussian Naïve Bayes Classifier performed the best overall at classifying test set feature vectors into Russian districts (based upon the evaluation measures of accuracy, unweighted average precision, and unweighted average recall). The performance of all the implemented classifiers, both library and manual, was relatively poor in regard to test set classification results, with the highest accuracy being only 0.48. Through an additional investigation, we found that this was due to there being a small number of attributes being used to represent the data samples (since we used a subset of 18 attributes for this study), and by increasing the number of attributes used, the accuracy of the classifiers would increase approximately

linearly. When examining the efficiency of the classifiers, we found that our implementations of the variants of the Naïve Bayes Classifier were significantly less efficient than the library implementations. This was in turn what limited our investigation to a small number of attributes being used in the data sample feature vectors. We further investigated efficiency by looking at scalability and discovered that all implementations were quite scalable with respect to increases in training set size. However, when examining scalability with respect to increases in test set size, we found that the library implementations were very scalable, whereas our manual implementations were not. Lastly, while exploring a potential practical application for the Naïve Bayes Classifier within the domain of the Russian housing market, we observed that the Gaussian Naïve Bayes Classifier was able to make interesting predictions that would be useful for an end user, such as a home buyer trying to determine which district to buy a home in. In summary, this study highlighted the importance of efficiency and scalability in data mining algorithms and demonstrated that the Naïve Bayes Classifier could indeed be used for a practical application in the housing market domain.

Our fourth study used the FP-Growth algorithm to mine frequent patterns in order to discover interesting relationships among property and neighbourhood attributes in the Russian housing market dataset. Within the first investigation of the study, we found that the manual implementation the FP-Growth algorithm was not able to find all the frequent patterns in the dataset based upon a given minimum support count threshold. In contrast, the library implementations were able to find all such frequent patterns. We then examined the time efficiency of the implementations by determining the total time taken for the implementations to find all frequent patterns. Our results showed that the manual implementation was significantly less time efficient than the library implementations. Consequently, we concluded that the manual implementation would need to be improved in both the areas of correctness and time efficiency. Lastly, we explored the interestingness and usefulness of the found frequent patterns within the Russian housing market dataset. We reported some of the interesting patterns that were found, and stated some of the implications behind such frequent patterns regarding the Russian housing market. In summary, this study highlighted that the manual implementation of the FP-Growth algorithm required additional tweaking to improve correctness and time efficiency, and that interesting and insightful information could be extracted from frequent patterns. Thus, proving the potential practical utility of the algorithm within the Russian housing market domain.

In our fifth study we used the K-Means Clustering algorithm to search for interesting patterns and relationships between attributes in the Russian Housing Market dataset. We had the aim of capturing interesting patterns which could possibly be used to classify new properties to a certain facet of real estate purchasing. Under comparison with some popular libraries our manual implementation performed quite competitively under areas such as cluster quality, and speed with reliability. It was capable of capturing clusters in multiple dimensions. We suspect that the issues with our model capturing interesting clusters are within the clustering tendency calculation. External resources generally show a much greater range of Hopkin's statistic values, with values generally ranging from 0 to 1. The implementation we used only provided values in a very small range, with around 70% of the values being 1. All values less than 1 were generally values greater than 0.9. As the only implementations we could find were from users on forums, we suspect that the code we used was faulty, at least when performing on one dimensional data. The first future improvement we would make would be to source new implementations of Hopkin's statistic, or program our own. The lack in speed of our implementation in comparison to Scikit-Learn reminded us of the importance of optimisation in data mining algorithms. Upon improving the cluster tendency attribute selection, we believe that the discovery of interesting patterns with clusters would be very applicable to housing markets.

5. References

[1] Sberbank. (2017, April). Sberbank Russian Housing Market. Retrieved August 20, 2019 from https://www.kaggle.com/c/sberbank-russian-housing-market

[2] Nielsen, M. (2015). *Neural Networks and Deep Learning*. Determination Press.

[3] DeFilippi, R. (2018). Standardize or Normalize? — Examples in Python. Retrieved 24 September 2019, from https://medium.com/@rrfd/standardize-or-normalize-examples-in-python-e3f174b65dfc

[4] Ronaghan, S. (2018). Deep Learning: Which Loss and Activation Functions should I use?. Retrieved 25 September 2019, from https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

[5] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal Of Machine Learning Research*, 9, 249-256.

[6] Bayes' Theorem and Conditional Probability. Retrieved 7 September 2019, from https://brilliant.org/wiki/bayes-theorem/

[7] CrashCourse. (2018). *You know I'm all about that Bayes: Crash Course Statistics #24* [Video]. Retrieved from https://www.youtube.com/watch?v=9TDjifpGj-k&t=205s

[8] Gandhi, R. (2018). Naive Bayes Classifier. Retrieved 11 September 2019, from https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

[9] Wang, C. (2019). Mining Frequent Patterns I. Lecture, Griffith University, Gold Coast.

[10] Wang, C. (2019). Mining Frequent Patterns II. Lecture, Griffith University, Gold Coast.

[11] Hyndman, R., Koehler, A., Ord, K. and Snyder, R. (2008). Forecasting with Exponential Smoothing. 1st ed. Springer.

[12] Hyndman, R., & Athanasopoulos, G. (2019). Forecasting: Principles and Practice. Retrieved 6 October 2019, from https://otexts.com/fpp2/

[13] Matevzkunaver. (2017). Hopkins test for cluster tendency. Retrieved 4 October 2019, from https://matevzkunaver.wordpress.com/2017/06/20/hopkins-test-for-cluster-tendency/

[14] Bishop, C. (2006). Pattern Recognition and Machine Learning (1st ed.). New York, NY: Springer-Verlag.

[15] Rolling Forecast Origin Diagram. (2019). [Image]. Retrieved from https://otexts.com/fpp2/accuracy.html