School of informatics

THE UNIVERSITY of EDINBURGH

Predicting Future Product Prices

Georgios Pligoropoulos

MSc Artificial Intelligence

School of Informatics

University of Edinburgh

2017

Abstract

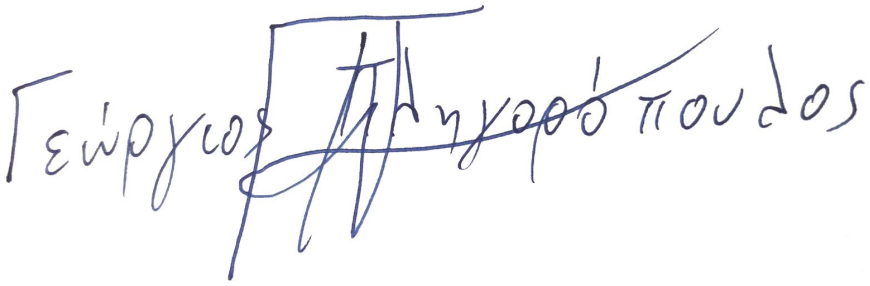
One of the most famous quotes of Larry Page, CEO of Google, is that "Lots of companies don't succeed over time. What do they fundamentally do wrong? They usually miss the future.". This dissertation addresses the question of whether the prices of retail products can be predicted for the near future. In particular, we are concerned with products that are subject to fierce competition, such as mobile electronic devices. In the era of online electronic competition, the prices of such products are subject to highly dynamic algorithmic changes on a daily basis. Traditionally the very interesting and very difficult problem of future price prediction has been dealt with autoregressive models or by building algorithmic models which take as inputs information such as supply and demand for each product. In this dissertation, we are exploring if state of the art machine learning techniques could create a predictor for future product prices based only on inputs which are generally publicly available, such as the static properties of a product and its price history. We are arguing that the past contains hidden information about the future in patterns undetectable to bare observation or to linear models and that more complex highly nonlinear models could take advantage of them. More particularly we are starting with simple recurrent neural networks architectures that have famously succeeded in capturing the correlations of sequences of data in domains like music and based on the results of our experiments we are evolving them to more advanced sequence to sequence architectures that combine the static features of a product and the dynamic information of its price history. We are training and optimizing advanced sequence to sequence neural network models, and we are showing that they are able to learn useful features in order to predict the future price history in low fidelity. We are expanding the notion of a typical neural network autoencoder to work for time series of variable length in order to cluster our dataset. In addition, we are preprocessing the price histories to become stationary and we are discovering that there are cross correlations between the price histories. We conclude with a discussion of how the low fidelity of the future price trend is more predictable than the sudden peaks that can be observed on the actual target sequences and that more exploration is needed before we make any conclusive statement of whether these are predictable or not.

Acknowledgements

I would like to thank my supervisor, Michael Herrmann, for the enthusiasm that he shared with me, his contribution to every step of my efforts and mentoring me throughout this thesis. His excellent cooperation played a key role in the successful completion of this thesis.
I would also like to thank my colleagues at the university as well as the global online community of data scientists and machine learning practitioners whose advice inspired me on my research.
Last but not least, I would like to thank Skroutz.gr company for giving me free access to their API and enabling the work of this thesis to be based on real-world data.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text and that this work has not been submitted for any other degree or professional qualification except as specified.



(Georgios Pligoropoulos)

Table of Contents

Chapter 1

Introduction

The retail industry has been disrupted the last few years by technology allowing retailers sell their products to a much larger audience by selling online. Competing on a national or even on a worldwide scale has brought opportunities as well as challenges. The consumers have the tools to compare value vs price in ways that they could never do in the past just by visiting a local store. Therefore in order to be competent, the retailers need to dynamically change their prices based on various factors which are both objective and subjective and vary per product or category of products.

This dynamic strategy on multiple products on a daily basis would be impossible to be adjusted manually from the typical size of the personnel of an e-shop. Therefore computer algorithms are being deployed to accommodate this need [1]. These algorithms and the business rules that they follow are usually proprietary and unknown to the world outside of the company of the electronic shop in hand. Given this dynamic environment where consumers, retailers and other stakeholders of the market operate, it would make sense to make more informed decisions if you could predict the trend of the price of a product in the upcoming future. This would be both beneficial for the consumers to know if and when to buy a product as well as for the retailers to make a better planning of their expected income following one strategy over the other.

On the other hand, models created with machine learning algorithms, either a single one of them or multiple models in a series of operations, have been able to capture patterns in multidimensional data of many instances that would be infeasible to us humans. The main goal of this thesis is using state of the art machine learning algorithms that will uncover hidden patterns in the the price history of a product along with any other information that we have available and research whether this history holds any information for the upcoming future that we aim at predicting, given that any such patterns exist.

Note that any model which is effectively predicting the future prices is destined to be useful as long as it is exploited by only a few agents. The case where the information of future prices are generally available to both the customers and the retail industry bring a new level of complexity. It is expected that all agents involved will try and exploit the power of knowing the future in their behalf which in turn will make them adjust their behavior in a more dynamic way than before. This suggests a very dynamic behavior of the system which would make the price prediction task even harder than it currently is, and it is not explored in this dissertation.

Chapter 2
Background

Our research did not reveal any literature to be directly related to the prediction of prices of any kind of retail products. However since our analysis is concerned with price histories, this is very closely related to the analysis of time series and the predictions of stock prices in the financial industry [19].

A more concrete example of stock price predictions can be seen here [26]. Similarly to our case this paper has access to information that is publicly available, and not with any extra information that would be available to the owners of the companies involved. The inputs being used are the opening price, the high price, the low price, the closing price and the volume of stocks. In this study, artificial neural networks have been able to predict the stock prices and the accuracy is analogous to the number of input data that are being used per instance. Based on these findings we can hypothesize that models based on artificial neural network architectures are going to produce similarly good results for our prediction task of the future price of a product.

Also, our smaller challenge for predicting the price of a product based on its attributes has been encountered multiple times as a research task [20]. Since the dynamic pricing of products is a relatively new functionality in the retail industry little attention is been given to recent research.

On the other hand, we found as a resource a company named Price Frontier which has implemented a proprietary algorithm which predicts the future of price moves of a product. Even if the Quantitative Price Prediction Model is proprietary and not freely available there is information about it in here [18].
Quote:
“Product prices move in a series of peaks and troughs. The direction of these peaks and troughs defines the trend of the price. Upward trends have rising peaks and troughs. Downward trends have falling peaks and troughs. A trend has three directions, upward, when demand outweighs supply, downward, when supply outweighs demand, or sideways when demand and supply are in equilibrium”
The system described requires more data resources than what is typically publicly available. For example, it is required to know the rate of purchase for a particular product category which indicates the demand, and what is the actual inventory of the electronic shop to indicate the supply. This information might not be available for the average e-shop. That is why the application of this system is limited to resourceful retailers such as Amazon.

Chapter 3
Data

3.1 Choosing Appropriate Data

Data Science and Machine Learning are still named as "black arts" because there are special cases and different treatments on every task and these might change based on the data we have at hand. The vision of Artificial Intelligence is to have a single model to rule them all but this has not been achieved yet, therefore choosing an appropriate dataset would make our data analysis more meaningful and appropriate for real-world deployment.

State of the art, machine learning algorithms are trained easier when we have a large enough amount of examples/instances that will make them learn and generalize better for unknown inputs. One solution is to generate product data and use them for our analysis and prediction task but this would mean that we would have knowledge of the distribution from which our price history data come from, but this is unknown to us. Therefore using generated data would be misleading in the sense that we would optimize our algorithms to work well on this particular rules that were used to generate these data. As a conclusion, we will avoid using any kind of generated data.

As our source, we are going to use the API provided by Skroutz.gr company because it gives us a collection of more than 5 million products which includes retail products of more than 2000 categories. An additional strong reason of choosing to have access to this particular data is because it refers to the vast majority of the electronic retail shops that exist in Greece and all of the e-shops, each one at its own category, are competing online as Skroutz.gr is itself a market place that refers to all of them. This makes the Skroutz API ideal for doing data analysis and price prediction for products that are competing with each other as this comes in consistency with the purpose of our thesis as explained in the introduction.

3.2 Choosing Product Category

For simplifying our modeling we could hypothesize that products are competing within the same product category and cases where the consumer compares products of different categories before making a purchase decision will be treated as a negligible minority and they will be neglected. This has an immediate effect that any kind of model should be trained and be evaluated separately for each product category.

Doing a qualitative comparison of price histories of various product categories we can see, in figure 1, that for many product categories the evolution of the prices of the products are mostly static and

thus predicting the trend of the future price can be easily modelled by fitting a very simple model where the predicted price is given by a rule such as using the same price as the one last seen without having to use any machine learning algorithms.
Another reason of avoiding to use static price histories is the fact that by being static they hold zero information of any potential trends that might be latent in the price history.

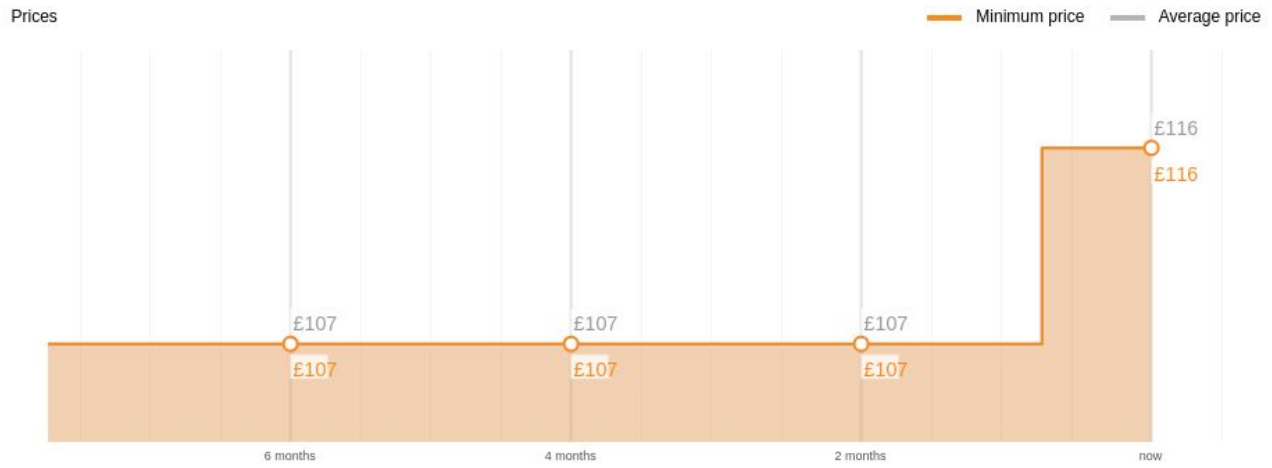


Figure1 Ray Ban Glasses Price History

While plotting, in figure 2, the price histories of most mobile phones we observe dynamic price changes on a daily basis.

Figure2 Sony Xperia XZ Price History

Let's plot the histogram of the variance for the price histories of all the mobile phone products, as seen in figure 3, to verify that there is a significant amount of products that have variance significantly larger than zero.

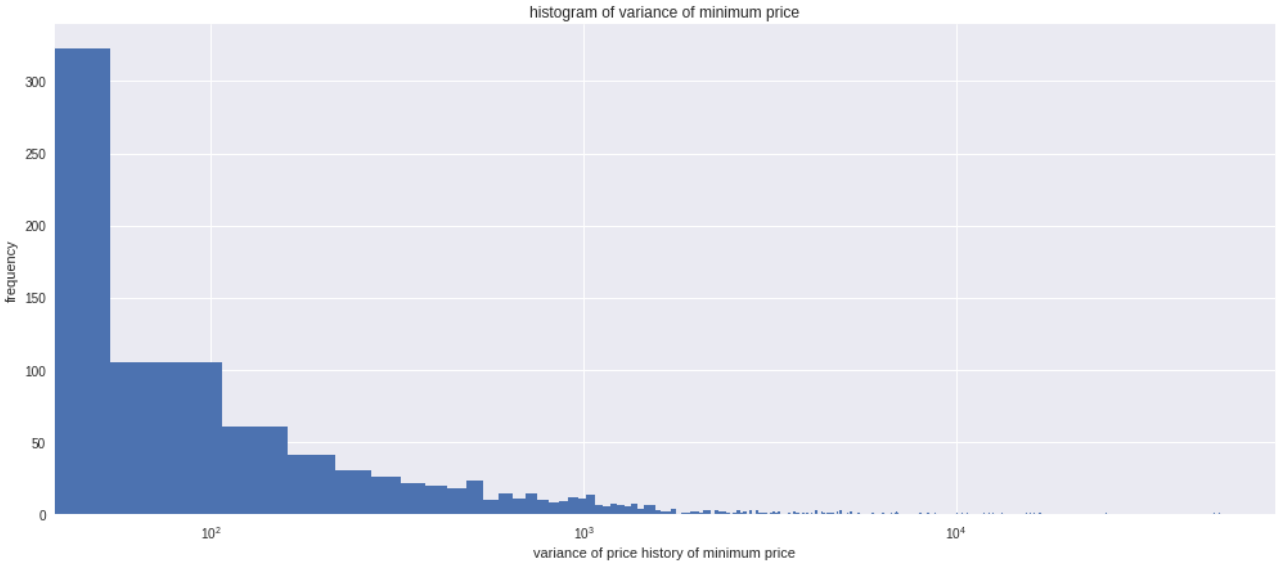


Figure3 Histogram of Variance of Price Histories of Mobile Phone Products with log-scaling of x-axis

We observe that there is a large number of mobile phone products with a small variance near zero, but there is also a heavy tail in this histogram showing the presence of many items with a large variance. Notice that there are a small number items with a very large variance that could be outliers.

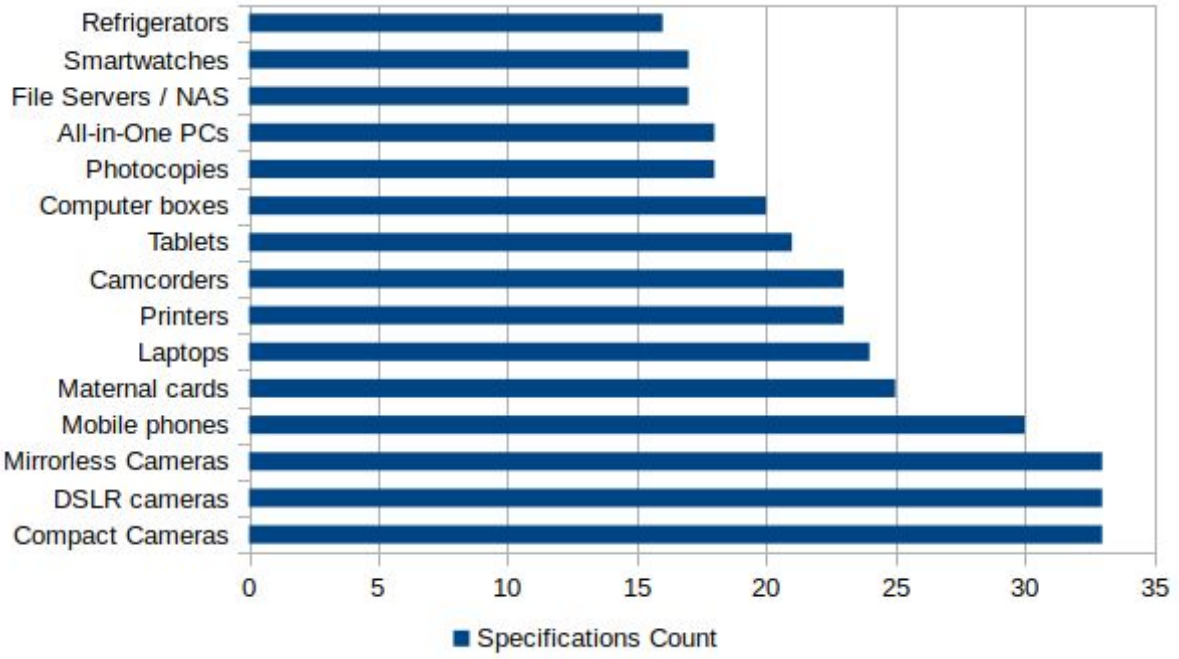


Figure4 Specifications Count for product categories with the largest specification count as provided by the Skroutz API

Therefore the first argument for choosing to work with the mobile phones category in this thesis is the dynamic price history which contains more rich information and makes the need for prediction stronger.

In addition not all product categories have the same number of attributes as we see in figure 4. Having many static attributes product will help our machine learning algorithms build custom features around them and training is expected to converge and also converge faster.

3.3 Importing Data

Skroutz API is of JSON format therefore we use the Unirest Python package [see Appendix 1] to fetch the latest data from the server using GET requests. Different request calls are being used to fetch the specifications and price history of each product. Fetched information is synchronized and saved in files in a numpy array format [see Appendix 1].

This results in a total of 798 instances of mobile phone products.

Note that each product has more than one price histories because its separate electronic shop that happens to sell a particular product has its own price which in general is different from the prices of other electronic shops.
However, we are keeping only the daily minimum price which corresponds to the most competitive price of the product on each particular day. The most competitive price is expected to attract the most customers.

The static attributes of each mobile phone are converted from JSON to tables in Pandas DataFrames [see Appendix 1] which in turn are saved in CSV file format.

3.4 Making Attributes Numerical

All of the attributes provided by the Skroutz API are available to the user of the platform. These filterable attributes help potential customers make more informed decisions for a purchase of a mobile phone product. Therefore we can hypothesize that these features play a smaller or larger role determining the value of the product thus making it more or less competitive when the perceived value is compared to its price. Taking this into consideration we are going to exploit as many of these attributes as possible and convert them to a numerical format that can be used as input to machine learning algorithms.

We are splitting the attributes into the following categories based on how we need to process them:

- We have Real-values or Integer attributes that they are already in numeric format without requiring extra conversion
- Binary attributes correspond to a 0 or an 1 indicating absence or presence in the mobile phone product correspondingly
- Categorical attributes assign the mobile phone product to a specific category and need to be preprocessed to one-hot encoding: Each category is assigned a vector of zeros which is equally long to the number of categories that exist and the only value that is not zero is the attribute that corresponds to the category for the current product. For example, an attribute with 4 categories gets transformed to a vector similar to this one: [0 0 1]. Note that the last

column is always dropped because it is colinear with the rest of the columns
- Special Attributes are attributes in string format which contain useful numerical information. Each one has to be preprocessed in a special way
- Minimum and maximum Price are target attributes that should be removed from training because they are directly related to what is the output of our prediction
- Certain attributes are dropped because they bring no useful information that could be related to the value of the product

For each one of the bullets above, there is a corresponding table in Appendix 3 which lists all of the attributes, their units, sample values, and reasons for why are including them or not in the final dataset. If an attribute is chosen to be included, then it is explained how our domain knowledge as customers suggests that this attribute relates to the value or price of a mobile phone product.

Chapter 4
Imputation

After the analysis we did in the previous chapter we have kept a range of attributes for each instance that we are going to use for the price prediction task. For a single instance, the range of attributes forms a vector in a space of dimensionality equal to the number of attributes.
The models that we are building in this dissertation will not work with input vectors that are of variable dimensionality or that they contain empty or invalid values. It is important to fill missing values of all our attributes before proceeding. An alternative is to drop attributes especially in the case where most of the instances are missing a particular attribute.

4.1 Ratio of Empty Instances per Attribute

In figure 5 we can examine the ratio of the instances which are empty per attribute.
Note that the attributes which are not mentioned in figure 5 have no missing values at all.
We notice that the majority of the instances, more than 80%, lack the “sar attribute. The result of this analysis is to drop the “sar” attribute completely from all instances before applying any imputation algorithm.
The rest of the attributes have at least 50% of the instances filled with valid values and we choose to keep all of them.

4.2 Imputation Algorithms

In order to fill the missing values we are going to use five different imputation algorithms:

- KNN: Nearest neighbor imputation which weighs samples using the mean squared difference on features for which two rows both have observed data
- Soft Impute: Uses the Nuclear Norm as a regularizer to provide simple and very efficient convex algorithm for minimizing the reconstruction error subject to a bound on the nuclear norm. Soft Impute Algorithm iteratively replaces missing elements with those obtained from a soft-thresholded SVD [2].
- Iterative SVD: Matrix completion by iterative low-rank SVD decomposition [3].
- MICE: Multivariate imputation by chained equations, sometimes also called “fully conditional specification” or “sequential regression multiple imputations” has emerged in the statistical literature as one principled method of addressing missing data. Creating multiple imputations, as opposed to single imputations, accounts for the statistical uncertainty in the imputations. In addition, the chained equations approach is very flexible and can handle variables of varying types (e.g., continuous or binary) as well as complexities such as bounds or survey skip patterns [4].

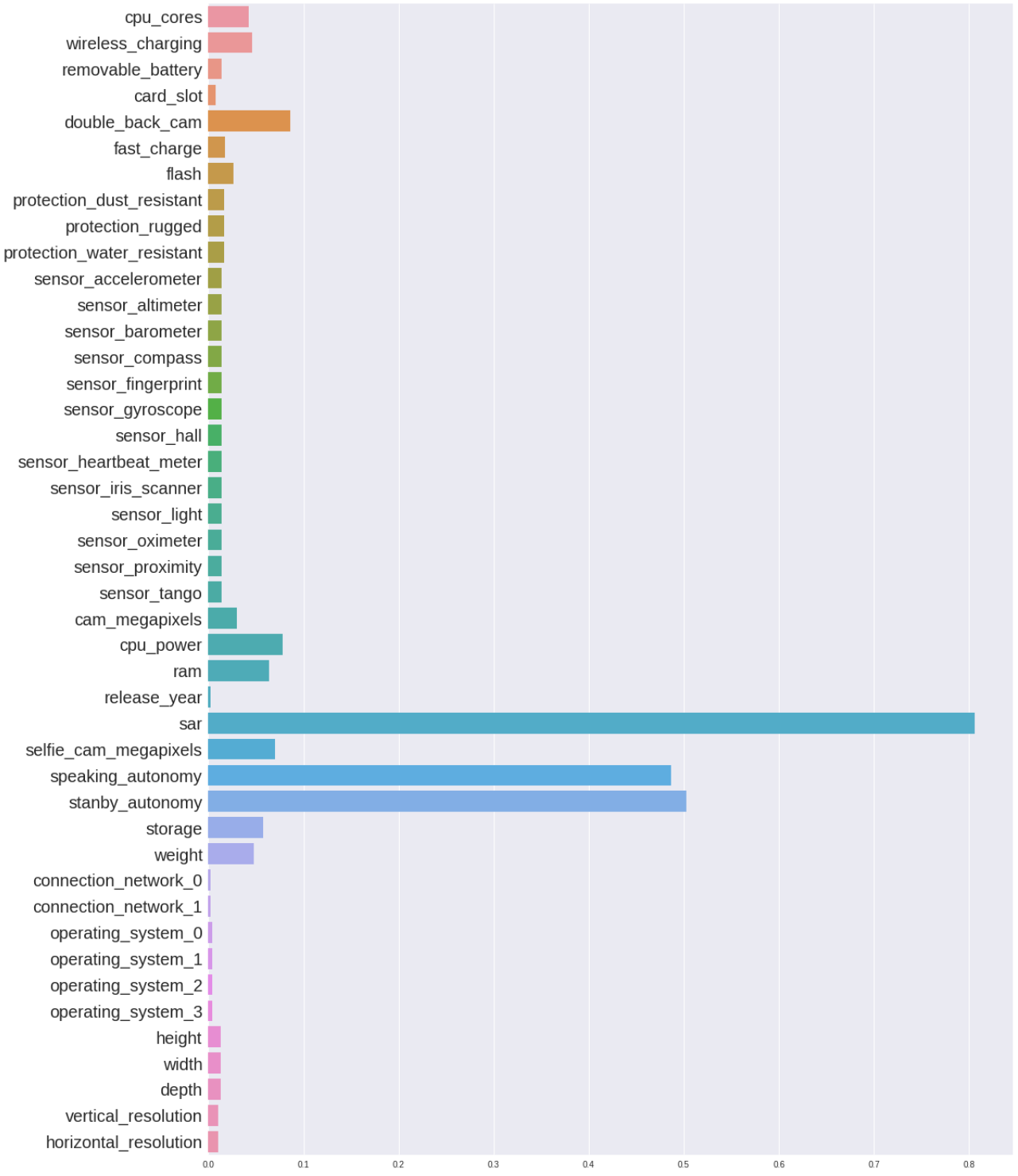


Figure5 Ratio of Empty Instances Per Attribute of Static Mobile Phone Product Features

- Nuclear Norm Minimization: One can perfectly recover most low-rank matrices from what appears to be an incomplete set of entries. The program described here [5] finds the matrix with the minimum nuclear norm that fits the data.

4.3 Filling Empty Values

For every one of the attributes with missing values, we are using a majority-vote criterion to choose the best algorithm for each attribute.
Our algorithm follows these steps for each attribute:

1) All the algorithms are executed and for a particular instance and we get distinct values corresponding to each one of our imputation algorithms
2) The algorithm that is closer to the average of the filled values is chosen as the best algorithm for the particular instance
3) We repeat steps 1 and 2 for all instances
4) From those instances, whose attributes were filled, we take a majority vote to pick the winner algorithm for the current attribute

The table in Appendix 4 shows the results of executing our majority vote algorithm for each attribute. Now we have which imputation algorithm works best for each attribute.
Finally we apply the corresponding imputation algorithm for each of the attributes with missing values and we get a complete matrix that can be used by all of our models.

Chapter 5
Static Attributes Preprocessing & Analysis

5.1 Outliers

Removing Outliers is a useful preprocessing of the dataset to remove instances that seem to originate from a different distribution from the rest of the majority of the instances because at least some of their attributes contain extreme values.

Here we have a limited dataset of 798 instances of mobile phones and therefore we need to be cautious of not removing too many instances. We are choosing to remove approximately only 1% of the instances that would be classified as outliers.

5.1.1 Removing Outliers with Quartiles

The first one and most commonly used outlier detection and removal algorithm is applied by checking if the value of an attribute is within the range:
Lower Bound: $Q1 - k * InterQuartile\ Range$
Upper Bound: $Q3 + k * InterQuartile\ Range$
This is also called Tukey's test [4].

Smaller k values correspond to more instances being classified as outliers while a larger k corresponds to more instances being classified as inliers.

Here we use a large k=6 which removes 9 instances of our current dataset which is, 9/798=0.0113, approximately 1% of the instances.

5.1.2 Removing Outliers with Isolation Forest

Sometimes there is the issue that outliers form groups which make them more difficult to detect and they are mistaken for normal. The Isolation Tree uses a tree structure where normal vs anomalies are separated by defining the internal and external nodes of the trees respectively.
There are various formulas to give the anomaly score. The algorithm is called i-trees and it is a recursive process which constructs isolation trees. There is the training stage where the anomaly score is asked to be optimized. There are multiple ways to evaluate the process such as the Orca k-nearest neighbor which provides linear complexity. One technique which helps is sub-sampling, by taking/sampling smaller amounts of data points. [7]

Here the contamination parameter of the Isolation Forest is set to be 1% and as a result, we get 8 instances classified as outliers.

To remove outliers we are considering both algorithms.
The union of the outlier sets is our final outlier set.
It is important to note that the union of the outliers is a total of 9+8=17 instances which means that the algorithms did not overlap not even in one instance. However 17/798 = 2.1% is an acceptable loss of our instances.
We are saving the remaining instances in a new DataFrame with a total of 781 instances.

5.2 Price Regression Potential Per Attribute

Before tackling the much more difficult problem of predicting the future price of a product we are going to apply simple regression algorithms which are trying to predict the single minimum (most competitive) price of a product based only on its static attributes.
This is useful to give us a sense of how well our static attributes are related to the price of the product.

In Appendix 5 we are plotting all real-valued features against the minimum most competitive price of each one of the mobile phone products of our dataset. After rendering the plots we cannot observe that a single attribute has the ability to predict the price with a significantly stronger correlation than others. The strongest relation we can observe between an attribute and the price is the CPU-Power attribute.

5.3 Standard Scaling

We are applying Standard Scaling to all attributes, except the binary ones, to bring mean to zero and standard deviation to one. We are not applying standard scaling to binary attributes or attributes which were transformed to binary via one-hot encoding because we cannot imply or assume a gaussian distribution for them.

5.4 Ridge Regression

We are using Ridge Regression, Linear Regression with L2 regularization, to see if a linear combination of all the attributes is able to perform well enough to our simplified regression task. Another important reason for applying Ridge regression, is to do an analysis of the weights of the model to check which attributes play the larger role in the regression task.

5.4.1 Train-Test Split

Before doing any kind of training we are separating 10% of our dataset to be used solely for testing purposes.

5.4.2 Bayesian Optimization

Bayesian Optimization offers a probabilistic model of the objective function. The advantage is that we do not need a closed form of the objective function.
Note that the objective function must obey some rules:
1) be smooth (similar inputs should give similar outputs)
2) be continuous
So we have this formula: $P(f \mid Data) \propto P(f)P(Data \mid f)$
P(f) is the Prior and the term P(Data | f) is the Acquisition Function. Acquisition function gives most probable point to observe next.

Here we are using Bayesian Optimization to optimize the lambda factor of the L2 regularization.
We are setting cross validation with a factor of k=10.
Each cross validation score is the average of the ten coefficient of determination scores.
The objective function returns the negative cross validation score. We want to maximize the coefficient of determination or in other words, we want to minimize the minus coefficient of determination.
The range of allowed values of lambda, L2 regularization factor, are from 1e-4 to 10 with a log-uniform distribution.
The optimization executes 5 times with a random initialization of the L2 regularization factor and continues for another 25 calls using GP Bayesian Optimization.

We see in figure 6 that we have found the best value within the first 13 calls.
The best L2 regularization factor is: 0.6295

5.4.3 Testing Best Model

Fitting the model to the entire training dataset with the best L2 and getting a score on unseen data, on the testing set we get the following coefficient of determination: 0.8792
The coefficient determination which is just squared correlation coefficient is a measurement that allows us to determine how certain one can be in making predictions from a model.
In our case, approximately 88% of the total variation can be explained by a linear relationship between the real target values and the predictions, which is a strong indicator of a good model.

In figure 8 we see that there is indeed a visible linear correlation between the targets and the predicted values.
This is an important outcome because it verifies our original assumption that attributes provided by the Skroutz API are features that the potential customers use to determine the value of a product.

This perceived value affects their final decision about a purchase which indirectly affects the price that the mobile phone product will have on the market.

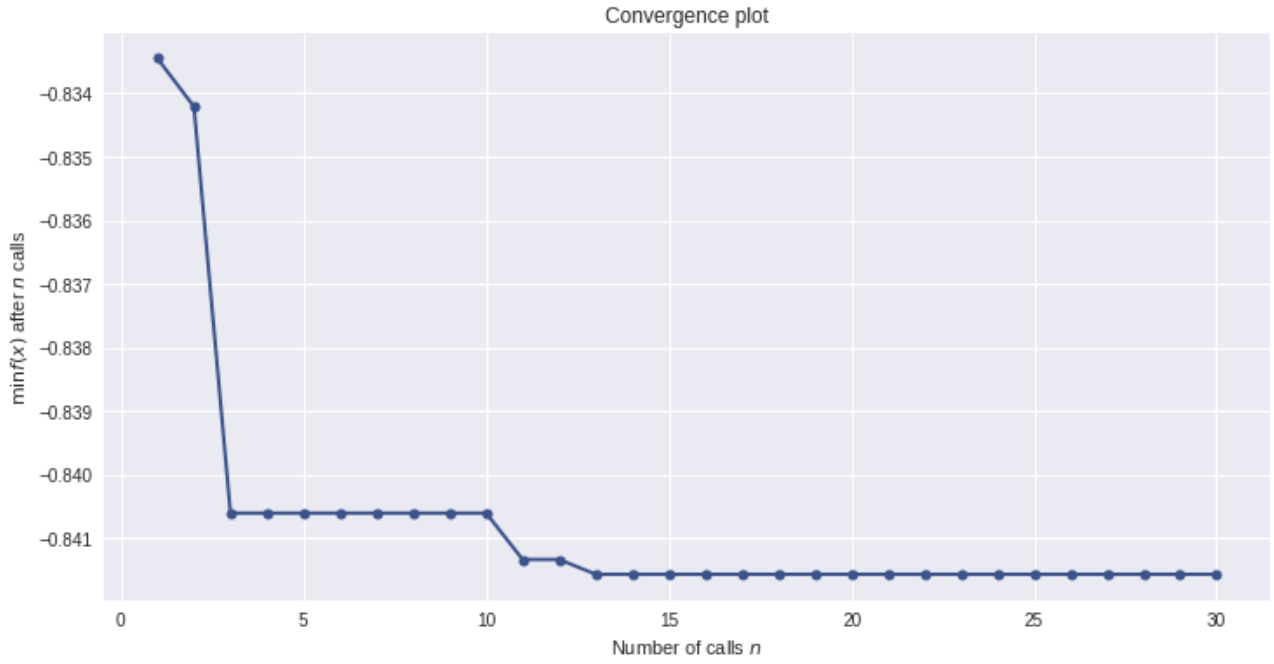


Figure6 Convergence Plot of Optimizing L2 Regularization Price Ridge Regression Factor

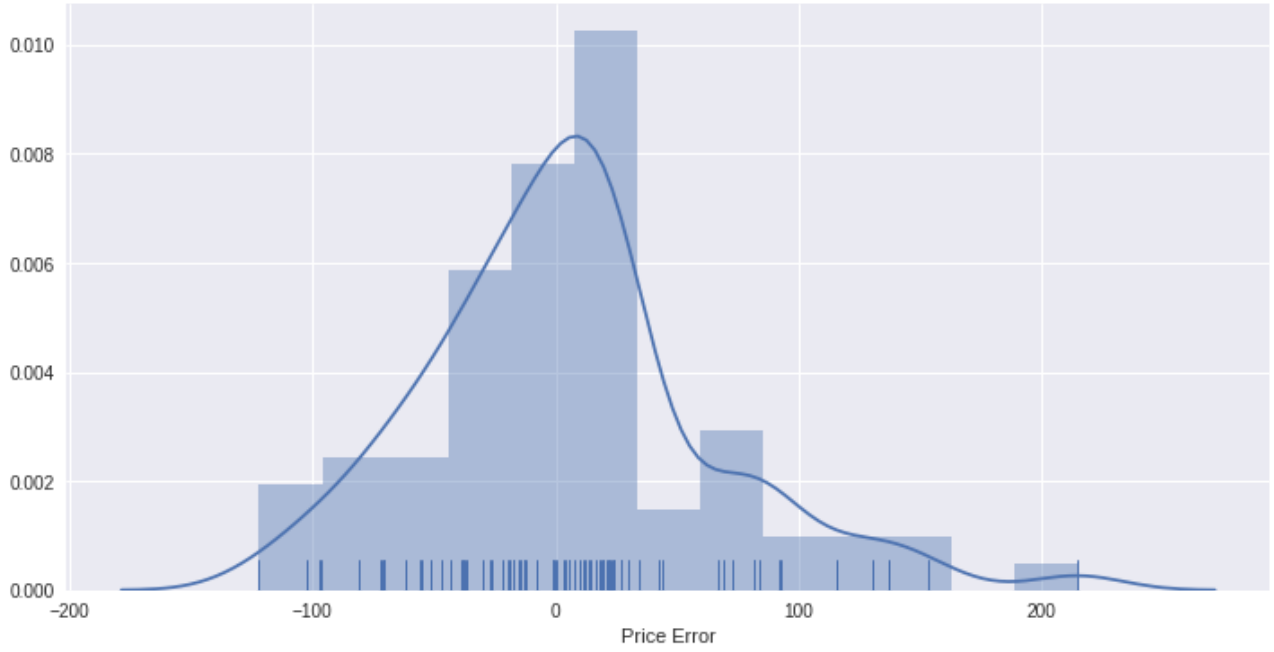


Figure7 Price Error Distribution

As we see in figure 7 the distribution of the price error is similar to a Gaussian but the samples are not that many to be very confident.

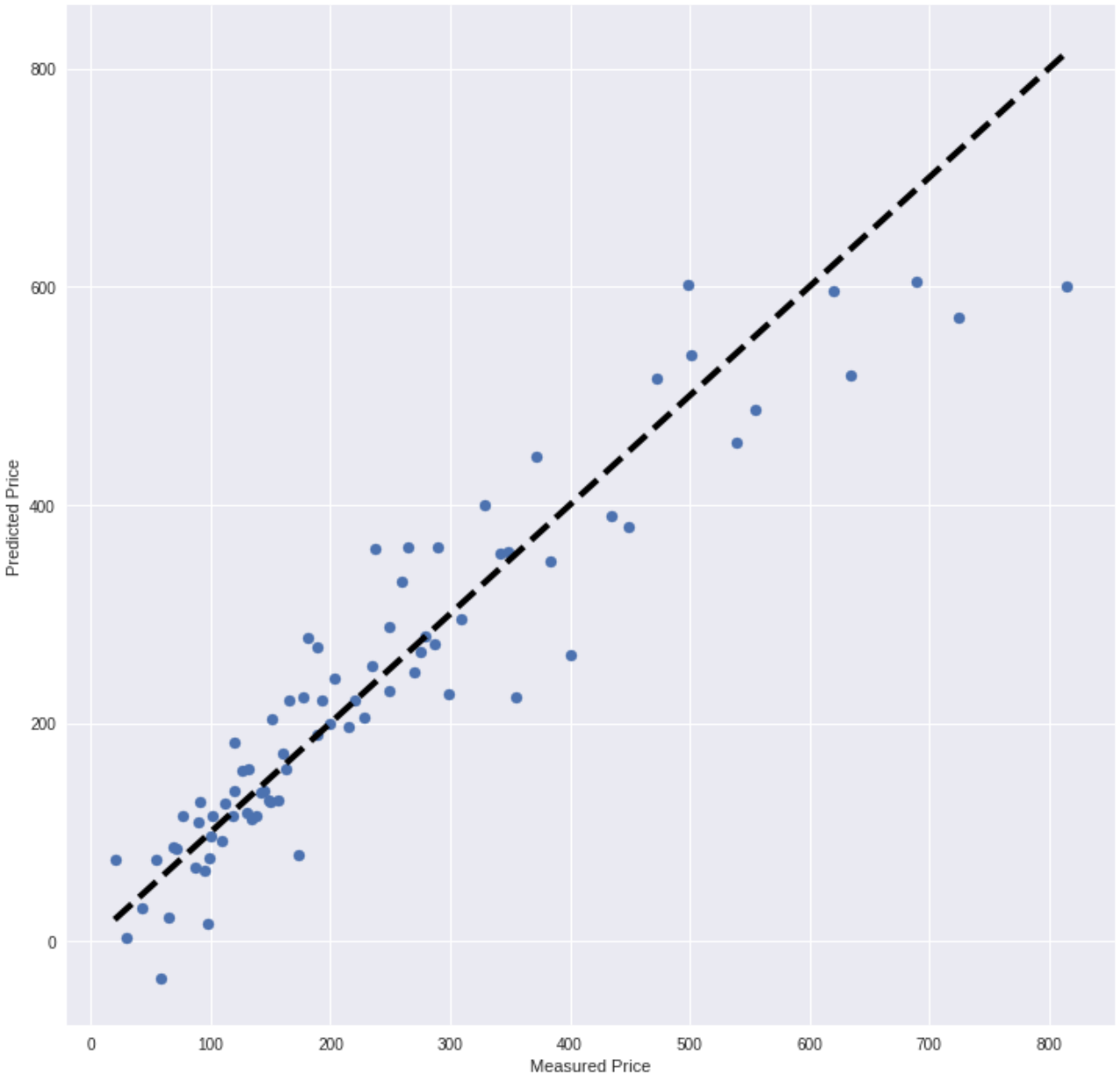


Figure8 Predicted Price vs Measured Price of Ridge Regression of Minimum Price

5.4.4 Which coefficient is stronger?

In figure 9 we see the absolute values of the coefficients which are stronger in our trained Ridge Regression model.

We notice that the manufacturer attribute plays a vital role in determining the price.

Also specialized sensors like the Tango sensor and specialized gadgets like the Iris Scanner security feature play a large role on determining the price of a mobile phone product.

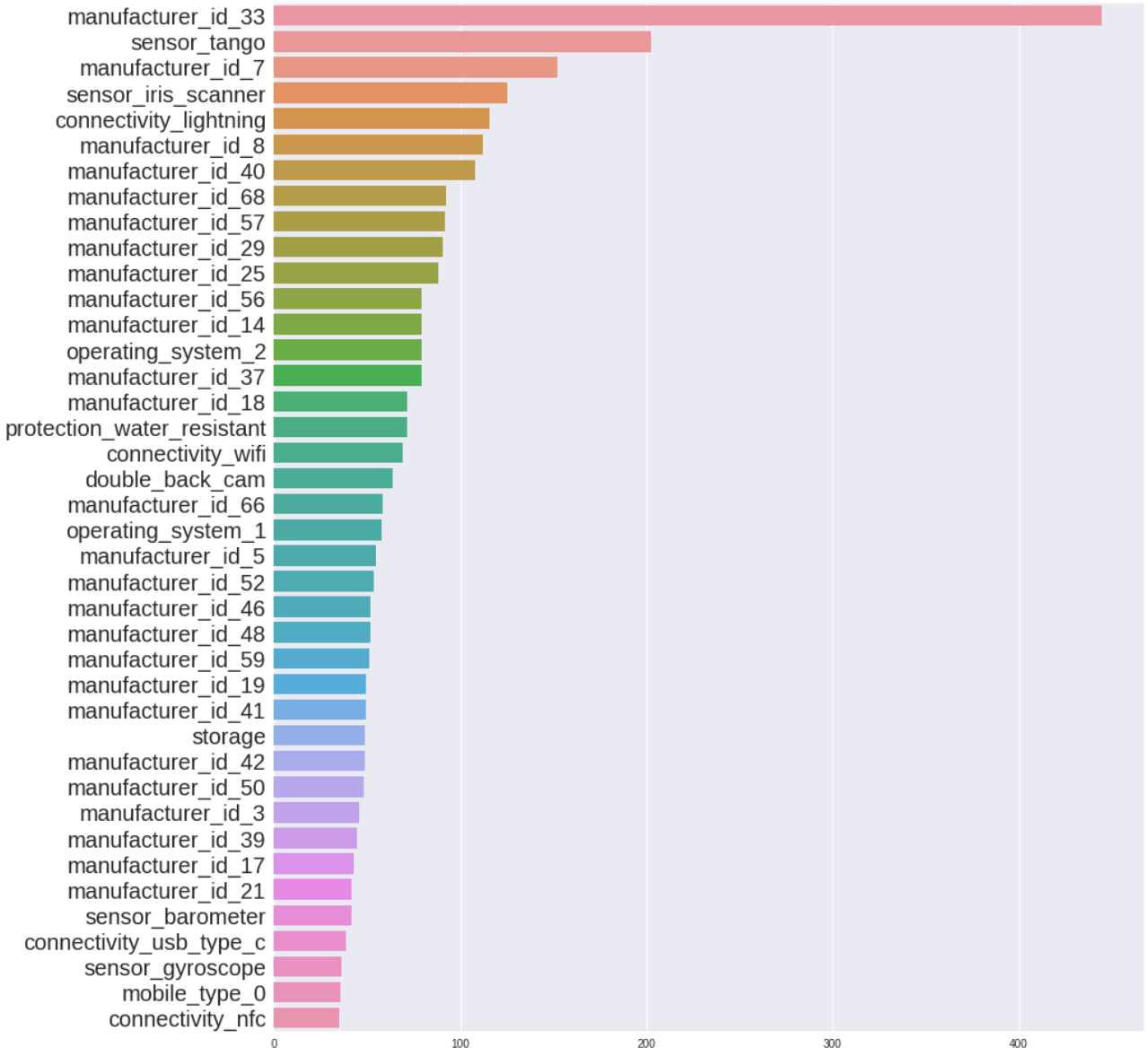


Figure9 40 strongest coefficients after training Ridge Regression model (absolute values)

5.4.5 Good-Deal Metric

Having a good model to predict the price we can compare the predicted price with the actual price on the market to determine if it is a good deal according to its attributes or not.
In other words, if the predicted price is much lower than the actual price then this means that the mobile phone product is being sold for a larger price than what is predicted from its attributes, given the current model. Similarly if the actual price of the product is lower than the predicted price then this means that the mobile phone product has superior characteristics in comparison to its price and it is a good deal.
In figure 10 we observe the ten best deals and ten worst deals and how large is the difference between the predicted price and the actual price. The best deal which here is LeEco Le Max 2 (128GB) should have been sold for 200 more with the rich set of its characteristics. While on the other hand LG V10 (32GB) is sold for 300 more than it should.

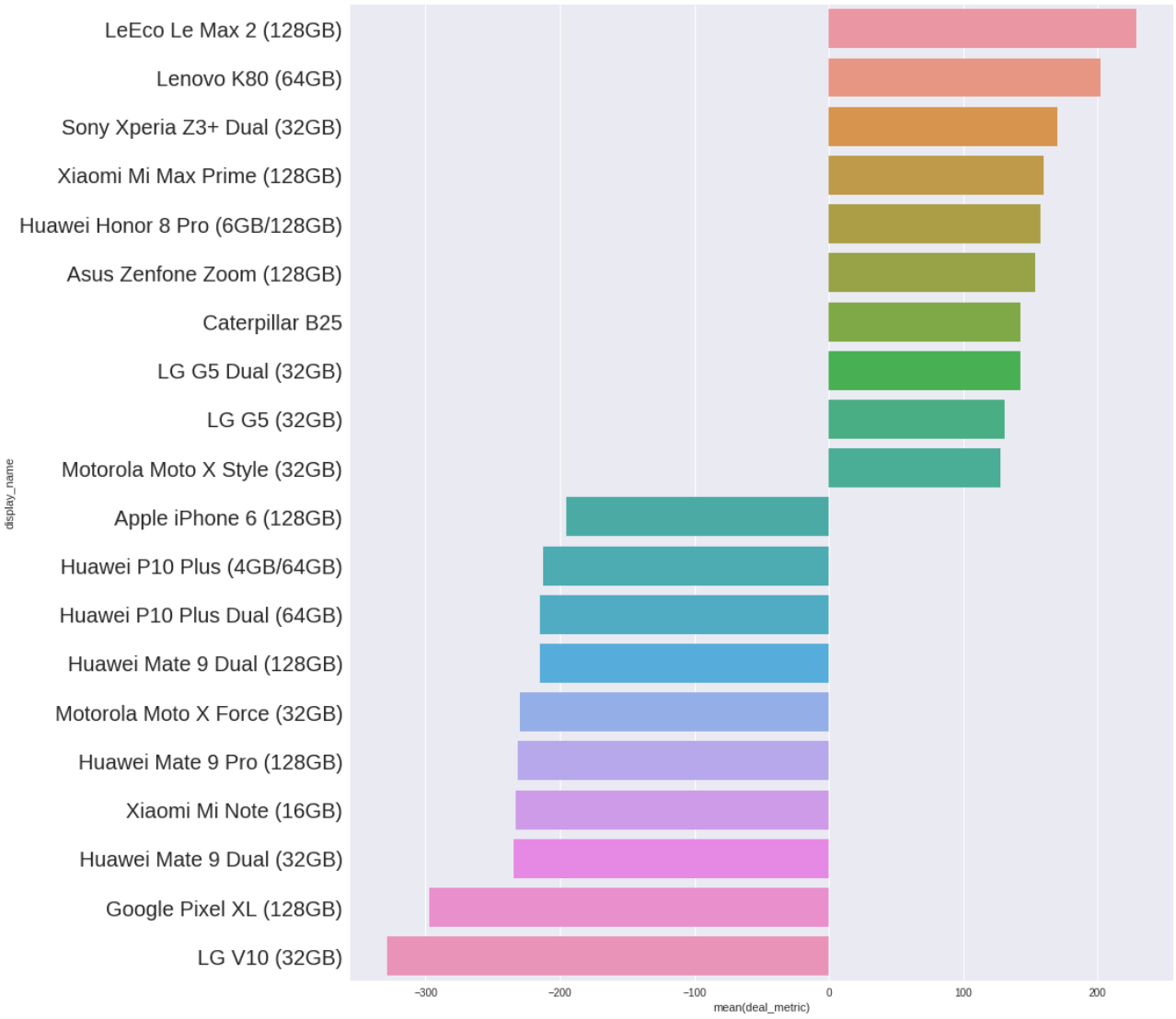


Figure10 Ten Best Deals and Ten Worst Deals according to Ridge Regression Model when comparing true price with predicted price

Chapter 6
Defining Evaluation

We will limit or future price prediction task to the first 30 days in the future.
We are using two metrics to evaluate if our predictors generate similar sequences as our targets.

6.1 Huber Loss

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Huber Loss is a more robust measurement of the residuals without being prone to outliers to either small or large values.
Huber Loss is used to comparing each individual amplitude of the 30 values f(x) of a predictor f, with the 30 values y, of the corresponding targets.
Huber Loss is a useful metric because it is differentiable.
The disadvantage of Huber Loss is that it does not take into account the sequential nature of the output vectors and only cares for the individual difference of each time step.
Another disadvantage of Huber Loss metric is that it cannot be used to compare instances of variable length.

6.2 Dynamic Time Warping (DTW)

The dynamic time warping algorithm is able to find the optimal alignment between two time-series. It is often used to determine time series similarity, for classification, and to find corresponding regions between two time-series. More particularly Dynamic Time Warping stretches two vectors, x and y, onto a common set of instants such that dist, the sum of the Euclidean distances between corresponding points, is smallest. To stretch the inputs, DTW repeats each element of the two vectors as many times as necessary [10].
For efficiency, it is necessary to use an approximation of the Dynamic Time Warping algorithms which is called FastDTW and is implemented in the *fastdtw* Python package.
The advantage of DTW is that it can compare sequences of variable length given that one sequence is similar to the other if it is stretched or compressed.
The disadvantage of DTW is that it is an algorithm and not a differentiable function that can be used by an optimizer.

Chapter 7
Generating Price History Dataset

Price Histories of our mobile phone products are of variable length for a few reasons. First of all they enter the market at a specific date and they might have left the market by the day we fetch the data. In addition depending on which date they entered the market, they might have a length which varies from one day to a few months.
The following figure 11 presents the histogram of the length of the price histories of our dataset.

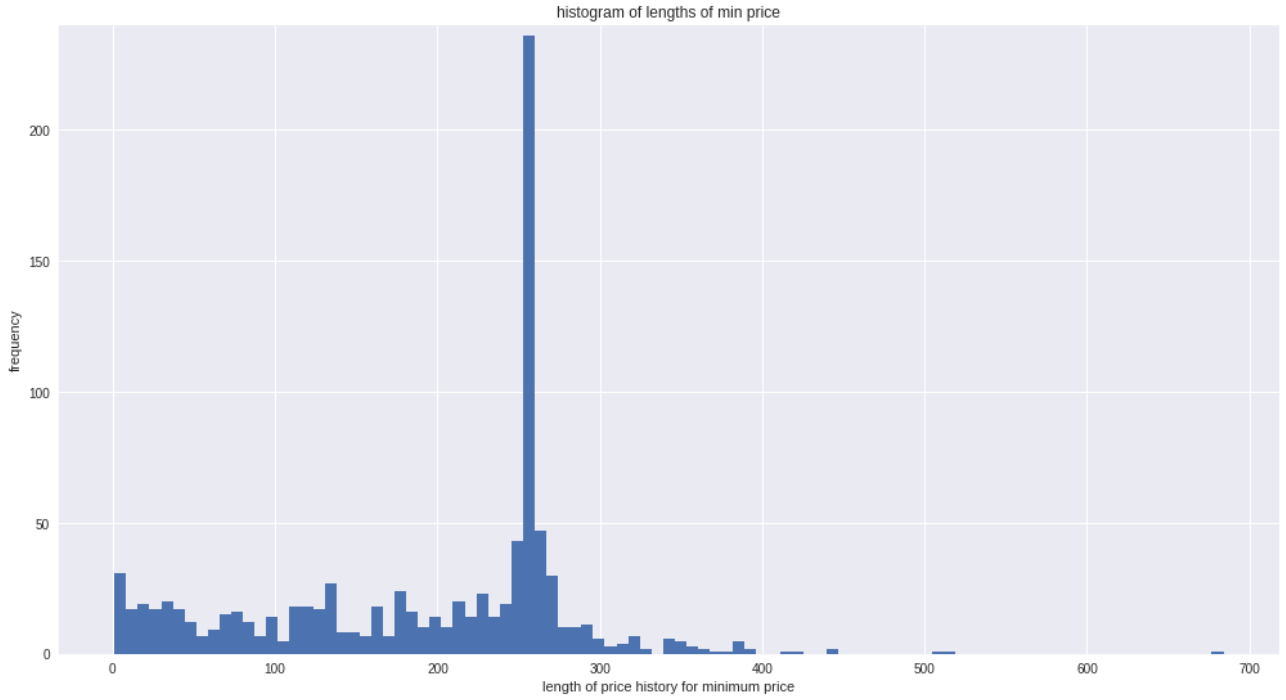


Figure11 Histogram of Length of Price Histories of the mobile phone products in our dataset

We notice in figure 11 that we have a wide range of lengths of price histories that range from the shortest possible price history of only one day to 682 days long.

The latest date in our price history dataset, which can be considered the current date ("now"), is 14th of June of 2017 and the longest price history, with 682 days length, goes all the way back to 3rd of August of 2015.

7.1 Outlier Detection in Price Histories

We are using Tukey's test [6] to check for outliers in every price history separately.
Our expectation is that the price of a product should be continuous meaning that the daily increase or decrease of a price should be small enough in comparison to the previous day in order to get

smooth curve at the end. We are not expecting the market to operate in a very dynamic way where the prices could have extreme fluctuations.

So we are checking for each price history separately if the prices are within the bounds:
Lower Bound: $Q1 - k * InterQuartile\ Range$
Upper Bound: $Q3 + k * InterQuartile\ Range$

We are expecting some spikes in the price histories due to the fact that electronic shops apply short term offers on their products which reduce the price of a mobile phone product for one or a few days and then the price returns to its latest, before the offer, value.

Taking the above into consideration we are setting a very large k=15 to treat as outliers only the extreme cases. The result is that 17 price histories are treated as outliers, which are plotted in figure 12.

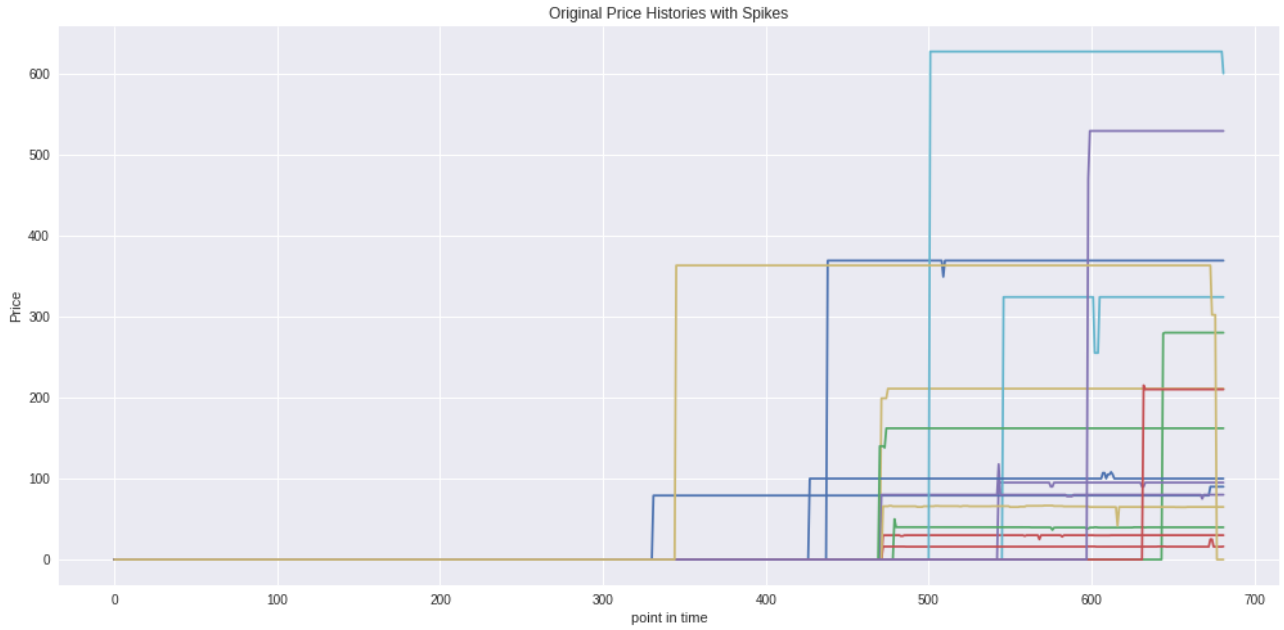


Figure12 Price Histories with extreme values are detected as outliers

We observe spikes for mobile phone products with the following two characteristics:

- The price history shows little variance and a constant trend
- The outliers in the price history occur only for one day causing the spikes

Our efforts to build a prediction system that can predict the price trend in the future is not expected to predict these sudden spikes as this would be a different problem that should be handled by a specialized model.
We notice that by taking into account only the price history of a single mobile phone product, there is no information to indicate that a sudden change in price is going to take place. This is because these sudden changes either have an individual underlying cause or they are depended only on the

date that a particular electronic shop created a special offer for the mobile phone product whose price history is rendered in figure 12.

To simplify our modeling and reduce noise while training our machine learning algorithms, we choose to programmatically remove these spikes.

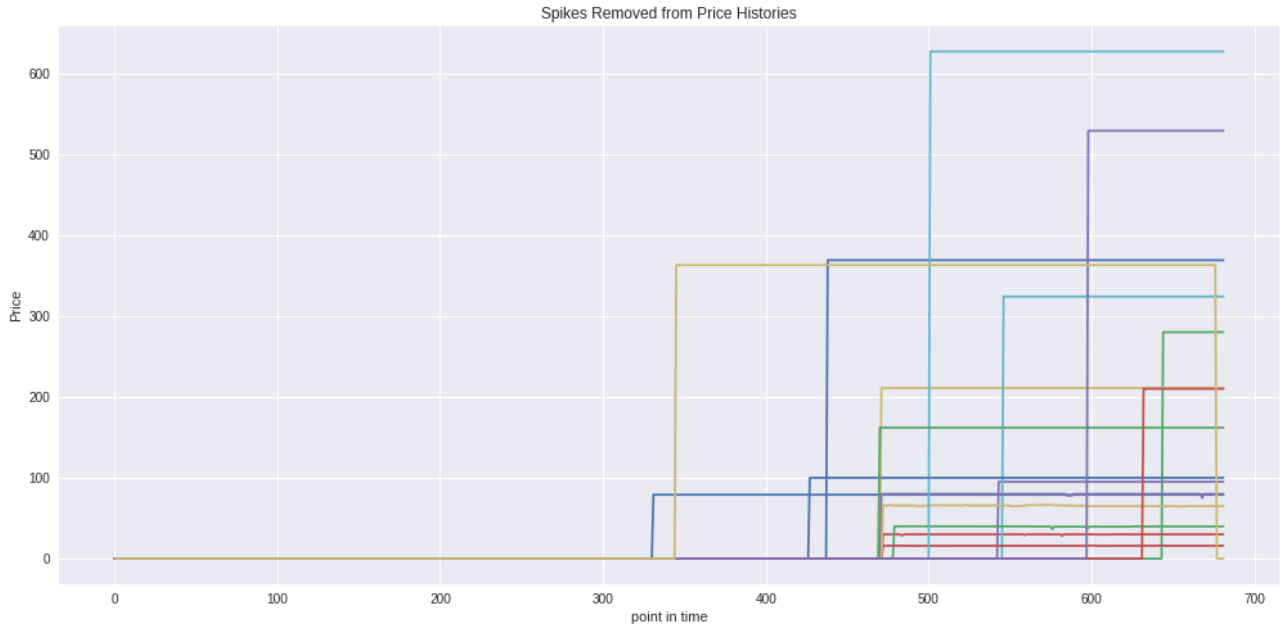


Figure13 Price Histories previously detected as outliers after spike removal

After applying spike removal we get price histories with no spikes as seen in figure 13.
Note that after this transformation no price histories are detected as outliers from our Tukey's test with k=15.

7.2 Generating Instances

To generate our dataset we are defining explicitly how long will the prediction of the price history be and how much information from the past we are requiring, or in other words how long the input sequence will be.

We are hypothesizing that having information of the last two months, or more precisely the last 60 days will be enough information of the trend of the price history of the mobile phone product to be used for predicting the prices in the future.
Input Length: 60

We will be predicting exactly the next 30 days, that is why our regression output should be of 30 instances long. Note that we are not predicting a single day in the future or the average of a few days in the future as this would offer little value to either the customer or the retailer and would come in contrast with our original purpose. Predicting the full range of 30 days will provide us of a

plottable price trend that will reveal whether the mobile phone product price tends to increase or tends to decrease and how much.
Target Length: 30

We have created a dataset generator to generate a series of instances by applying a sliding window to the original price histories, with stride one.

Note that we are not splitting between training and testing set at this point as our aim is to first find an appropriate model for our price prediction task that is able to predict well enough, with the aim of overfitting. Only after overfitting has been observed, it makes sense to optimize our model to generalize on unseen data.

Also, note that by generating instances of a full length of 60+30 = 90 days we are discarding price histories that are shorter than length of 90 days.

From our 781 price histories, 125 sequences have a total length that is smaller than the minimum total length of 90 days.

In figure 14 we see the count-plot of those sequences that we are forced to drop because they contain too little information to fit in our current modeling.

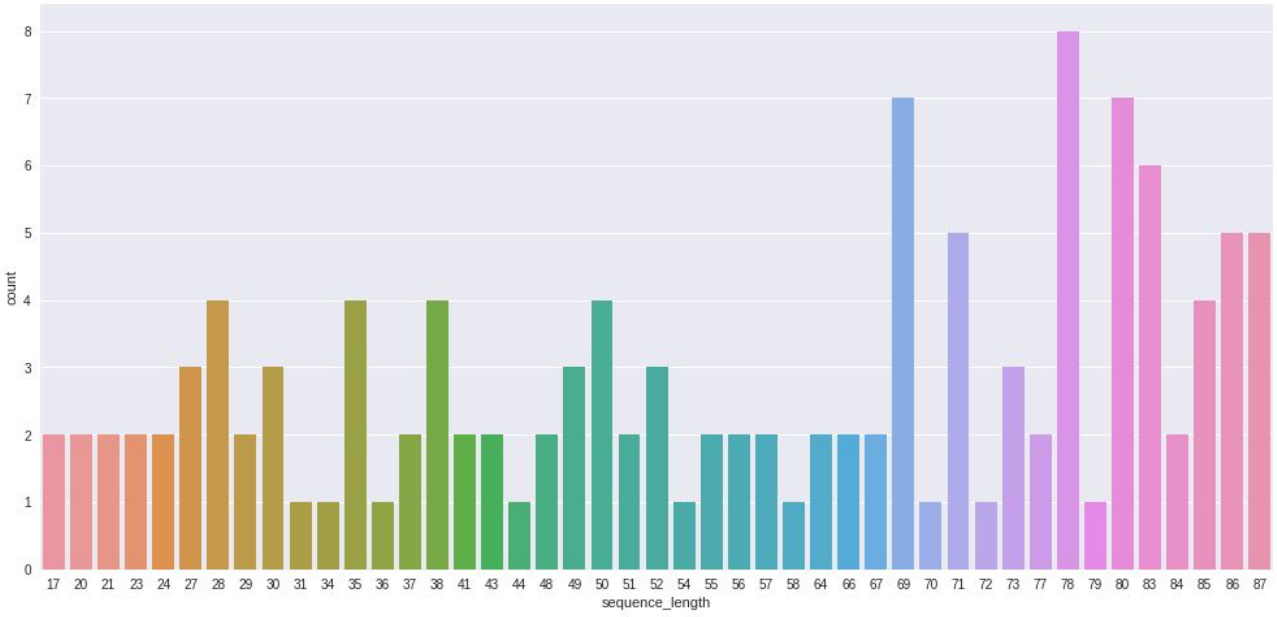


Figure14 Count-Plot of sequences of Price Histories with total length smaller than 90

So for the rest of the price histories that have a length greater or equal to 90 we use a sliding window to generate new instances, from them. Note that these are a total of 781 - 125 = 656 sequences.

For example, if we plot the price history of the mobile phone product with id “10084353”, we get a plot as seen in figure 15.

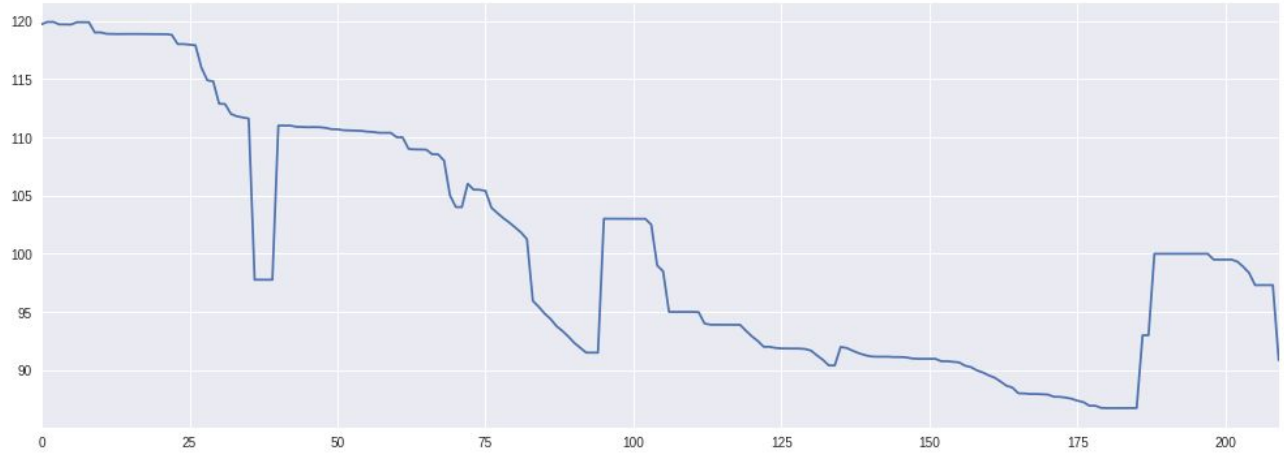


Figure15 Full Price History of Product with id ‘10084353’

Note that sliding a window of width of 90 days slided to a price history of a total of 210 days with a stride of 1 we get a total of:

$N - window\ length$ instances = 210-90 = 120 instances

In figure 16 we see plotted the first and the last of these instances.

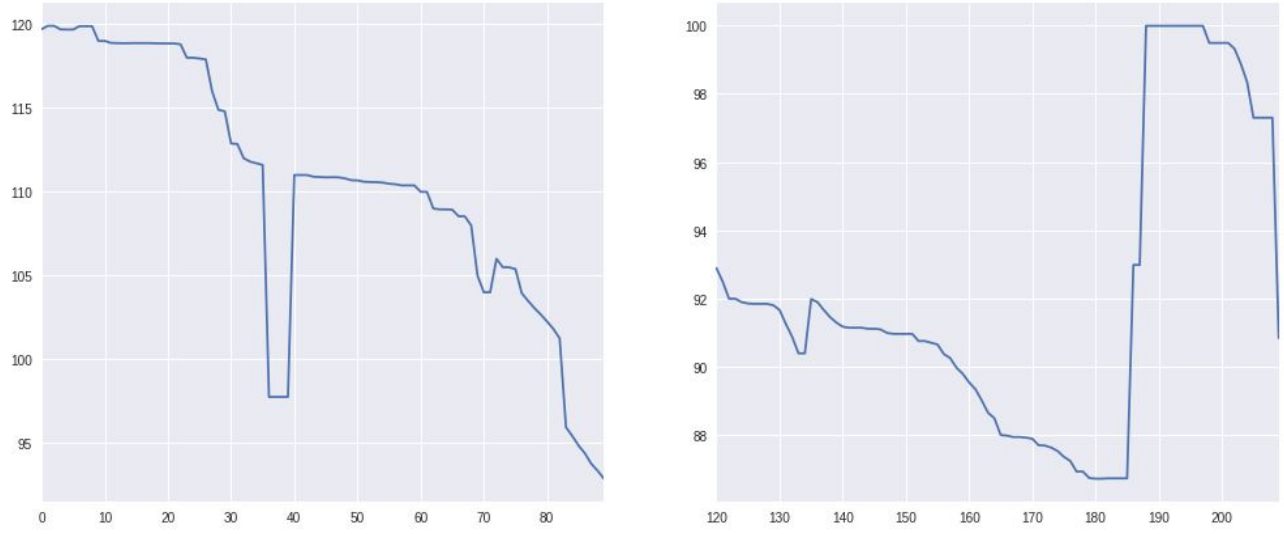


Figure16 First and Last Part of Price History of product with id ‘10084353’ after segmentation with sliding window of width 90 prices

We are using a sliding window because this enables us to generate many more sequences rather than only using the original ones, thus giving the model a greater variety of instances and make it learn better representations and finally generalize better for the forecasting price prediction task in future dates.

7.3 Removing Bias in Price Histories

Since each mobile phone product belongs to a different price category we indeed have each price history to start from a different value. We cannot expect our model to easily learn to work well with all the different price categories which are in fact a bias that can be removed now and added back at the end when we need to reconstruct our original price histories.
Therefore from every sequence we remove the value of the first element in order to make all generated sequences to start from zero. As we see in figure 17, the new graph represents how much the price deviated from the original price as the time goes forward and the bias is removed.
Note also that we are rendering with a different color the first 60 days of the price history from the rest 30 days to denote the inputs and the targets.

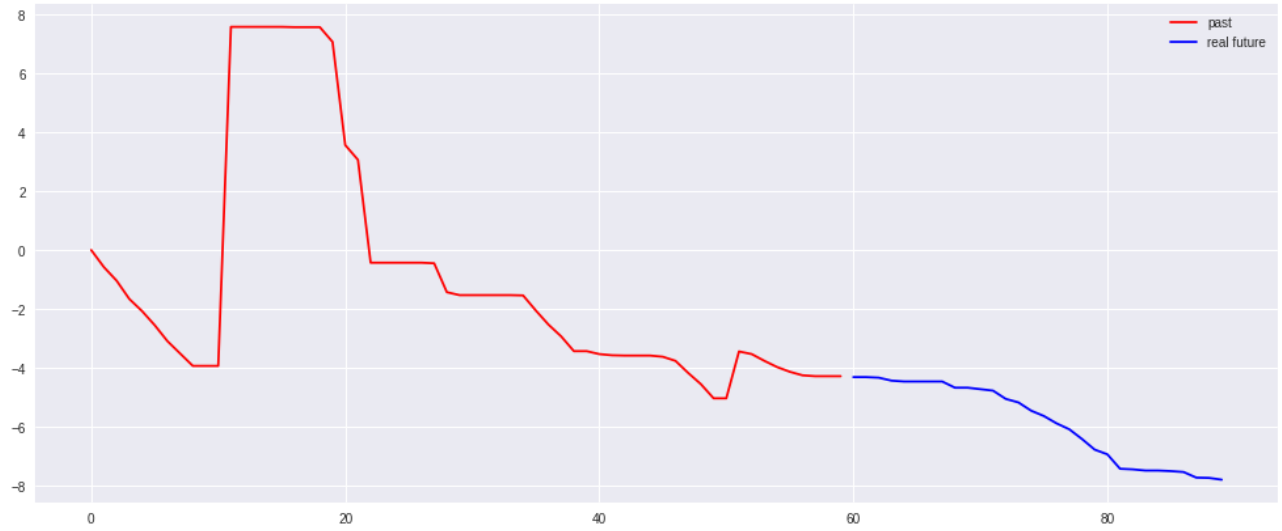


Figure17 Generated Sequence of 90 prices from Price Histories with bias removed. Inputs/Past in red color. Targets/Future in blue color.

7.4 Baseline

Before building our models we need to create a dummy predictor that can give us a baseline of what would be the worst expected prediction if we had not used any machine learning algorithms and we were building the predictor with simple rules.

7.4.1 Dummy Predictor - Linear Trend

One first idea for a dummy predictor is to use Linear Regression to predict the future. This means that we are fitting a line on the part of the sequence that corresponds to the input. Then without having any extra information we simply extrapolate this line to the dates corresponding to the future and we get predictions as displayed in figure 18 and figure 19.

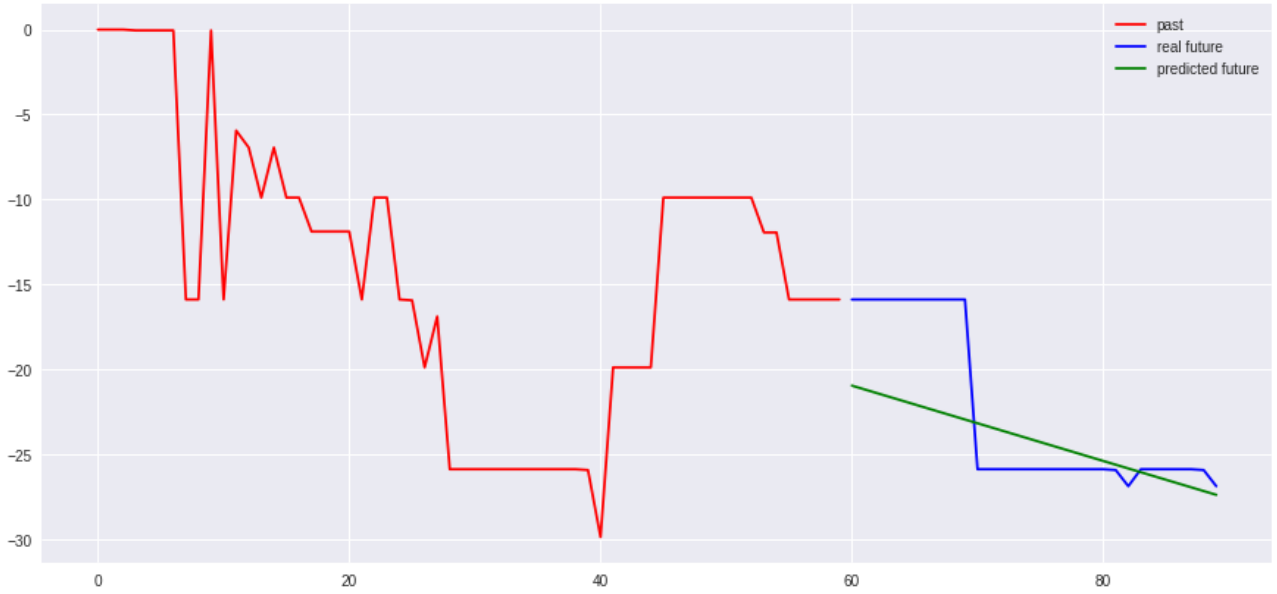


Figure18 Successful case of Linear Future Price Prediction

In figure 18 we notice that the overall trend of the previous two months is useful information for the continuation of the price in the future and thus the predicted linear trend in the future follows the same decreasing trend of our targets.

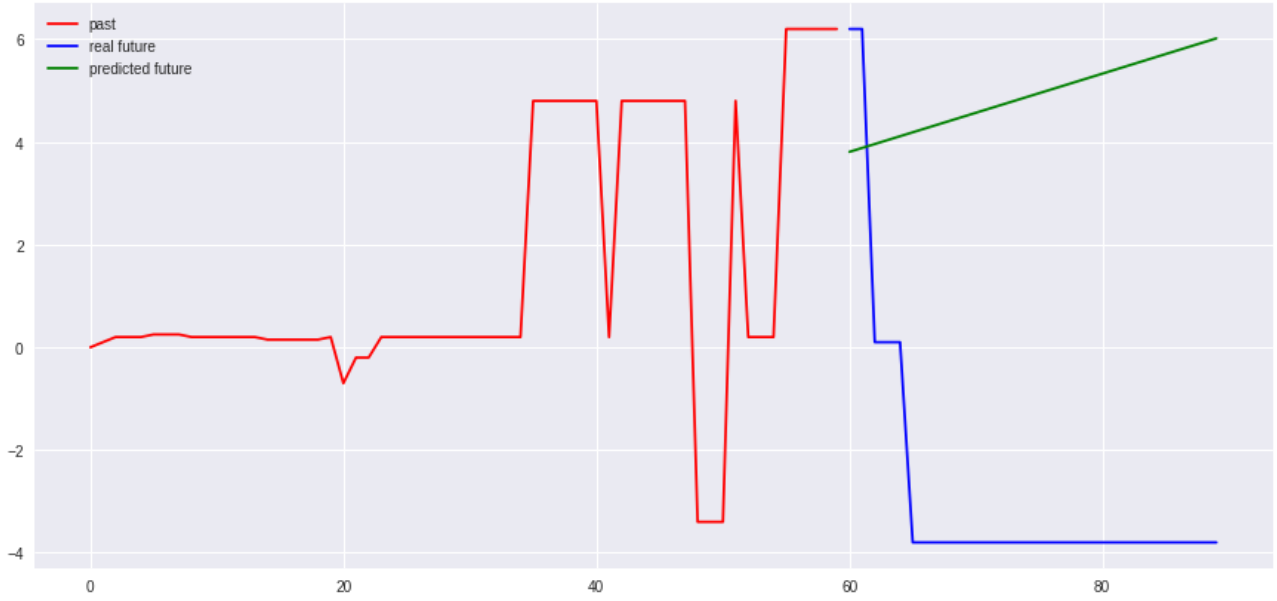


Figure19 Failed case of Linear Future Price Prediction

However, in figure 19, we observe how difficult the problem of price prediction is, since in this case the linear trend is increasing which comes in contrast with what really happens as we notice a couple of sudden drops in the price of the product.

7.4.2 Dummy Predictor - Constant

Another baseline we can take into consideration is to have a predictor which produces always the same constant value as the last seen price. This dummy predictor discards all price history that corresponds to the inputs, except the last value. It predicts constantly in the future this latest price value. The result of a prediction is displayed in figure 20.

7.4.3 Compare Dummy Predictors

Dummy Predictor	**Linear Regression**	**Constant / Static**
Average Huber Loss	8.24602934	4.78770150
Average DTW Score	249.8402279	153.97031739

Notice that while the constant dummy predictor is a much simpler model than the Linear Regression which takes into account as input the entire input sequence, it is also able to outperform the Linear Regression in both metrics of the Average Huber Loss and Average DTW Score.
This can be explained by effects as the ones observed in figure 19 where the individual loss becomes large because of the big difference between the trend of predictions versus the trend of the targets.

Figure20 Dummy Static Future Price Prediction based on latest Price

From the above analysis, we can derive to the important conclusion that predicting future prices is a very difficult problem and there is no linear relation between the inputs in the past and the future. This is why we have chosen artificial neural networks to confront this nonlinear problem.

Finally we choose to use as our baseline the better of the two dummy predictors which is the constant / static dummy predictor and set as our minimum baseline values the following ones:
Average Huber Loss: 4.78770150
Average DTW Score: 153.97031739

7.5 Choosing a subset of Price Histories

So far we have 656 price histories that can be used for generating instances as we are explaining in the previous section with 60 days as input and 30 days as targets.

In figure 21 we see the distribution of the lengths of our 656 price histories. Note that a lot of price histories have a length of exactly 210. In fact, 95 price histories have this particular length.
We are choosing to keep a subset of our price histories with a length of exactly 210 days.
So from price histories with variable length we are choosing a specific length to work with, at least for our first experiments.
Benefits:

- It will give us the freedom to choose models that do not work with inputs of variable length
- It will speed up our experimentation because a total of 95 instances will be trained faster

We are aware that whichever model gets trained on this limited dataset will not be representative of our entire dataset and we shall build a more advanced model as soon as we get any promising results on training on this subset.

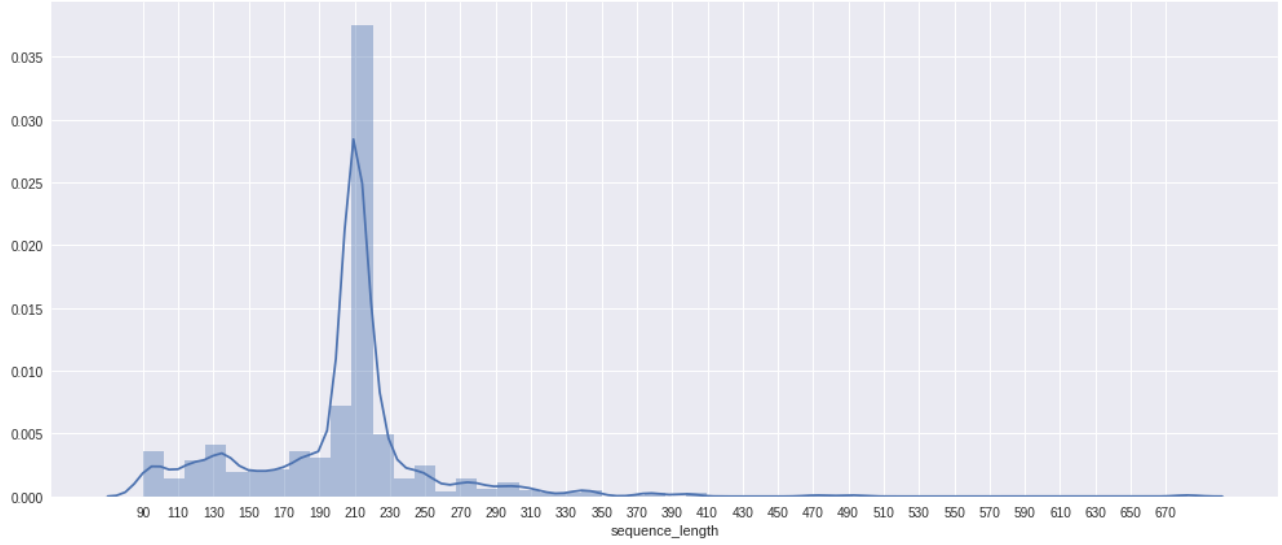


Figure21 Distribution Plot of Lengths Price Histories after removal of mobile products which are outliers

Note that we are considering the our baseline metrics to not be affected by this subsampling of the dataset.

Chapter 8
Recurrent Artificial Neural Networks

RNNs are going to be the basis for our following neural network architectures. We consider them a good candidate for modeling our price prediction in the future because RNNs have two main characteristics:

- The recurrent nature of RNNs enables them to capture correlations of data that are ordered in a certain sequence. In other words "recurrent neural networks take advantage of the temporal summarization of the features of the sequence" [9]. The million song dataset for music genre classification of songs of variable lengths is a typical case which demonstrates the suitability of RNNs for sequential data.
- They allow inputs of variable length to be processed by the same model. If there were no recurrence then the input has to be a vector of particular dimensionality which would limit our modeling to only sequences of a certain length.

All neural networks models are built as a Graph with Tensorflow Python package [see Appendix 1].

8.1 Recurrent Neural Network Cells

In this thesis we are going to use two architectures for our Recurrent Neural Network Cells:

1. Basic RNN
2. Gated Recurrent Unit (GRU)

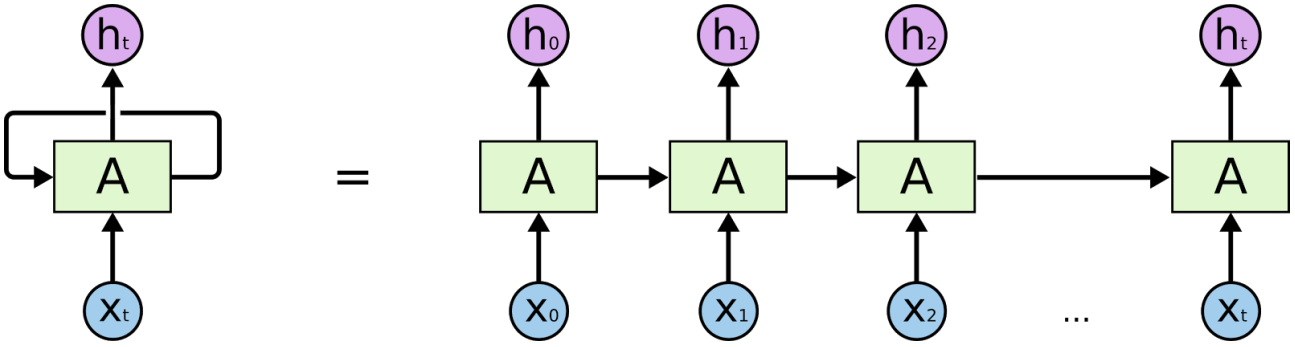
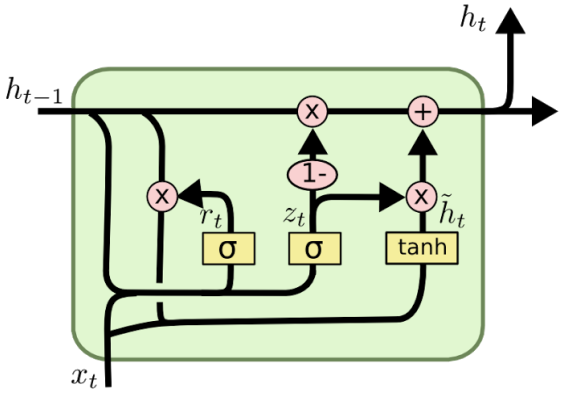


Figure22 Basic RNN cell architecture

A Basic RNN cell is a typical Neural Network Layer which has the ability to feed its output as its input and thus cause recurrence. The layout of a Basic RNN cell is seen in figure 22.

A GRU cell is a “simplified” version of the Long-Short Term Memory (LSTM) cell where the input and forget gate are combined into a single update gate. It also merges the cell state and the hidden state. The layout of the GRU cell can be seen in figure 23.

GRU cell, being simpler than standard LSTM models, because it has less trainable parameters, increases training speed. Moreover it has been shown [11] that GRUs are able to outperform LSTMs in many tasks thus making them a more attractive candidate. Due to those reasons we will not be experimenting with LSTM cells in this thesis.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure23 GRU RNN cell Architecture

8.2 Single RNN Model

As our first experiment we are going to build a very basic Recurrent Neural Network where the price history of the mobile phone product is being fed into the Recurrent Neural Network.
The graph of this model is being displayed in figure 24.
Note that the final state of the RNN is being discarded and we only care about the last 30 outputs of the RNN. This is why there is a gathering node in the graph. The gathering node discards the first 30 outputs and only cares about the last 30 ones. Recall that each one of the 60 inputs produces an output. However, since we are predicting 30 days into the future we keep only the last 30 outputs.

8.2.1 Hyperparameters

Hyperparameter	**Value**
State size	400
Batch size	47

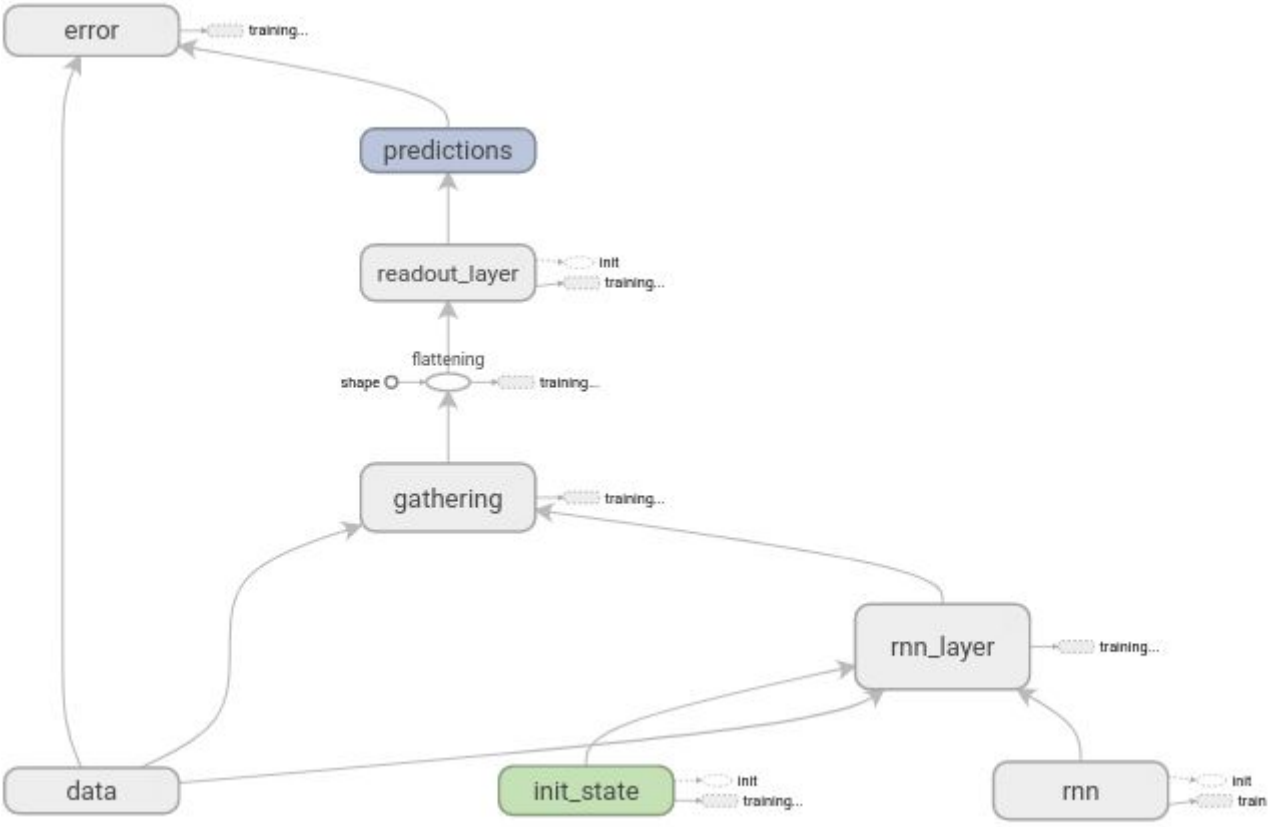


Figure24 Single Recurrent Neural Network Model for Price Prediction

8.2.2 Training

8.2.2.1 Training with Basic RNN cell

First, we are training using a Basic RNN cell with state size of 400 and after 10 epochs we are at a Huber loss of ~7.3 which is much higher than our baseline. We see the progress of this training in figure 25.

Doing a qualitative check to compare a random prediction with its targets we see that the model starts its prediction from a value near zero while the actual value is near 10. The prediction does not seem to resemble the targets, as seen in figure 26.

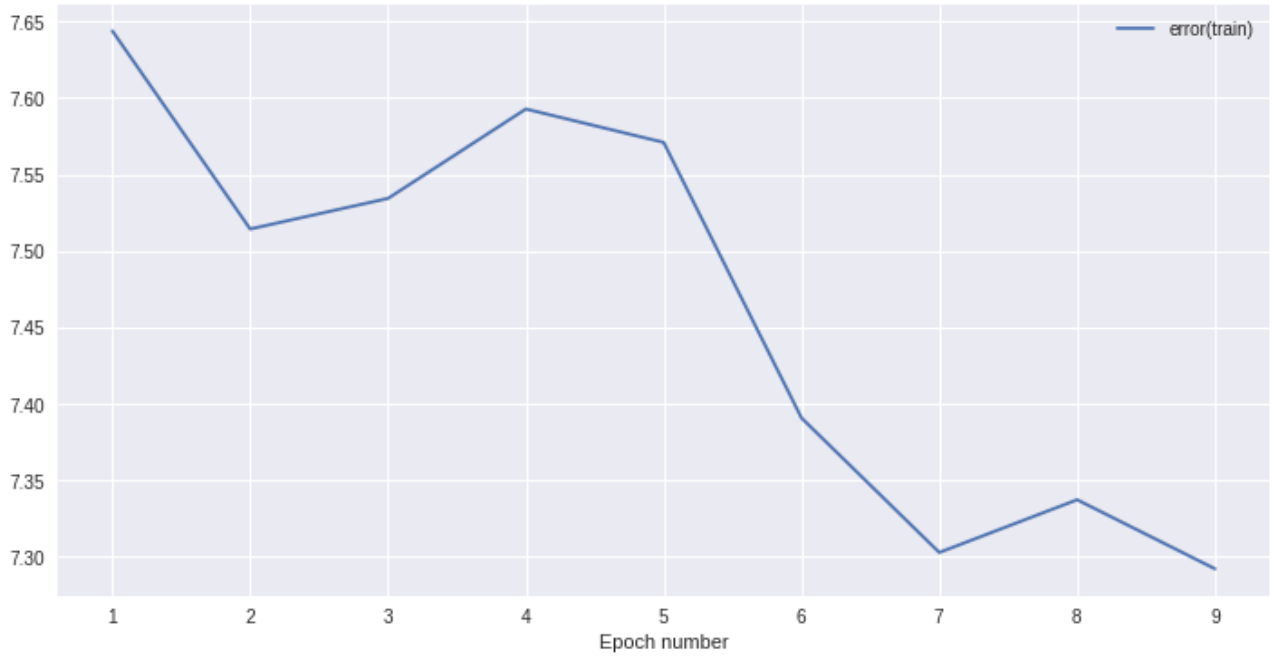


Figure25 Training Single RNN Architecture with Basic RNN cell - state size 400

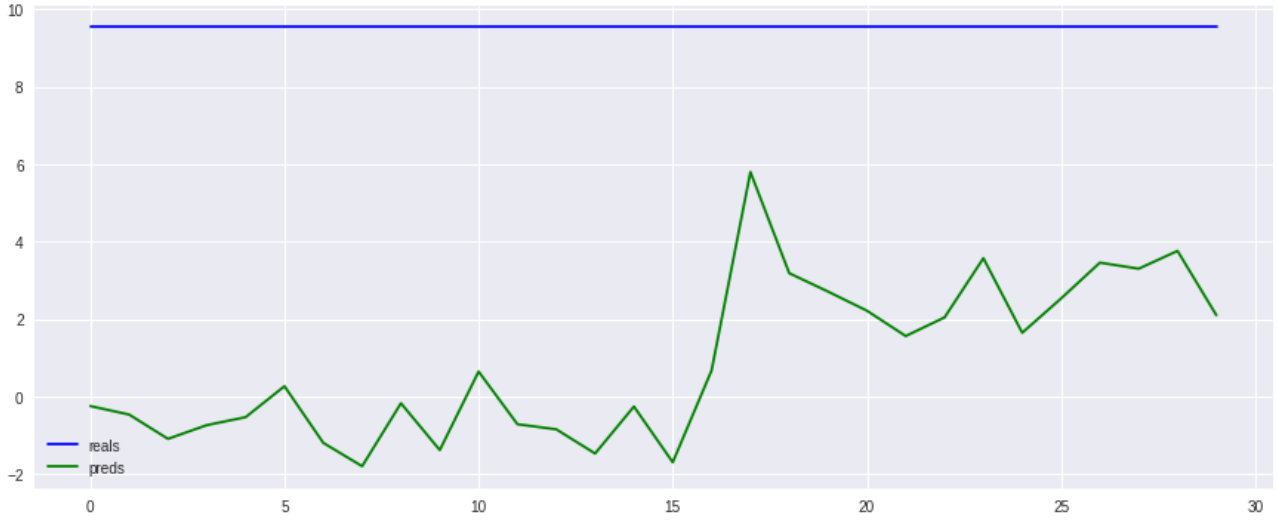


Figure26 Sample Prediction of Single RNN Architecture with Basic RNN cell - state size 400

8.2.2.2 Training with GRU cell for 10 epochs

By simply substituting the Basic RNN cell with a GRU cell of same state size, as we see in figure 27, we get, even from first epoch, a much better performance than Basic RNN and after only 10 epochs we have reached a Huber loss which is below the baseline.

RNN cell	Baseline	Basic RNN cell	GRU cell

Training Huber Loss	4.78770150	7.40530299	3.26558178
Average DTW Score	153.97031739	219.42501921	97.12150262

The conclusion here is that the GRU cell, by having more advanced memory characteristics than the basic RNN, is able to be a better model better for our price prediction task giving values below the baseline.

Note that in figure 27 the training Huber loss is continually decreasing which means we might be able to perform better if we let the model train for more epochs.

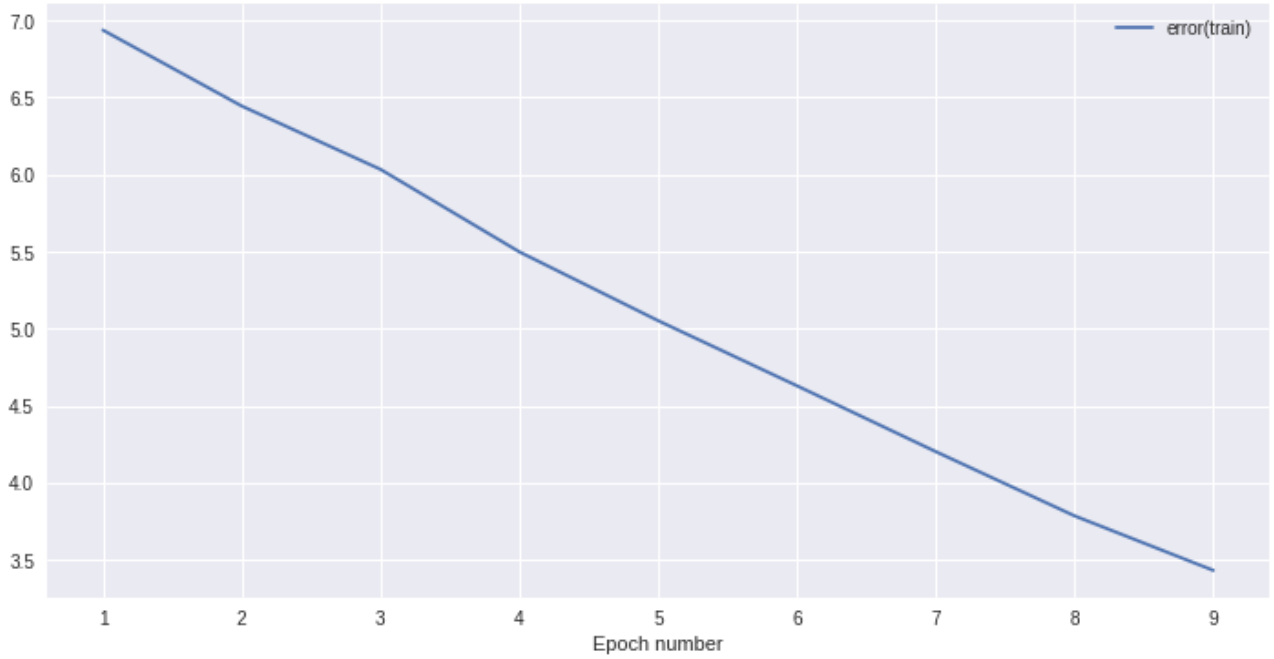


Figure27 Training Huber Loss Single RNN Architecture with GRU cell - state size 400

8.2.2.3 Training with GRU cell for 50 epochs

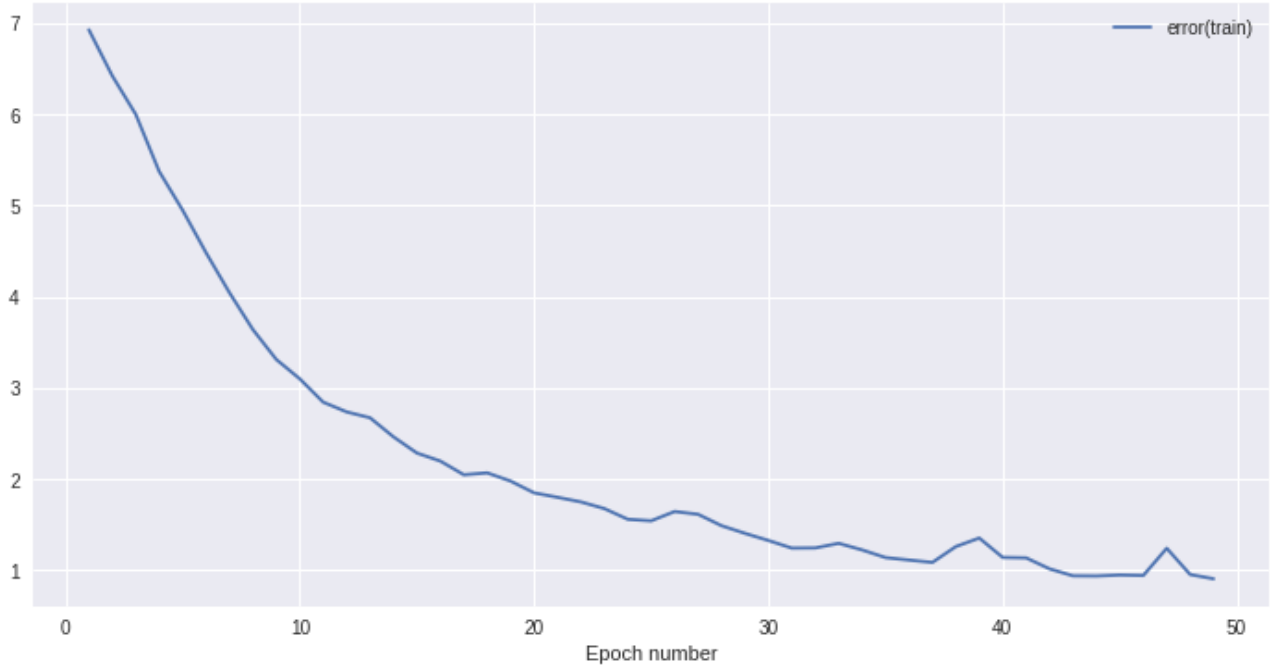


Figure28 Training Huber Loss Single RNN Architecture with GRU cell - state size 400 - 50 epochs

We notice that after 50 epochs the training error has mostly converged.
Average Training Huber Loss: 0.80812264
Average DTW Score: 31.506525364
The conclusion is that both metrics show a significant decrease.

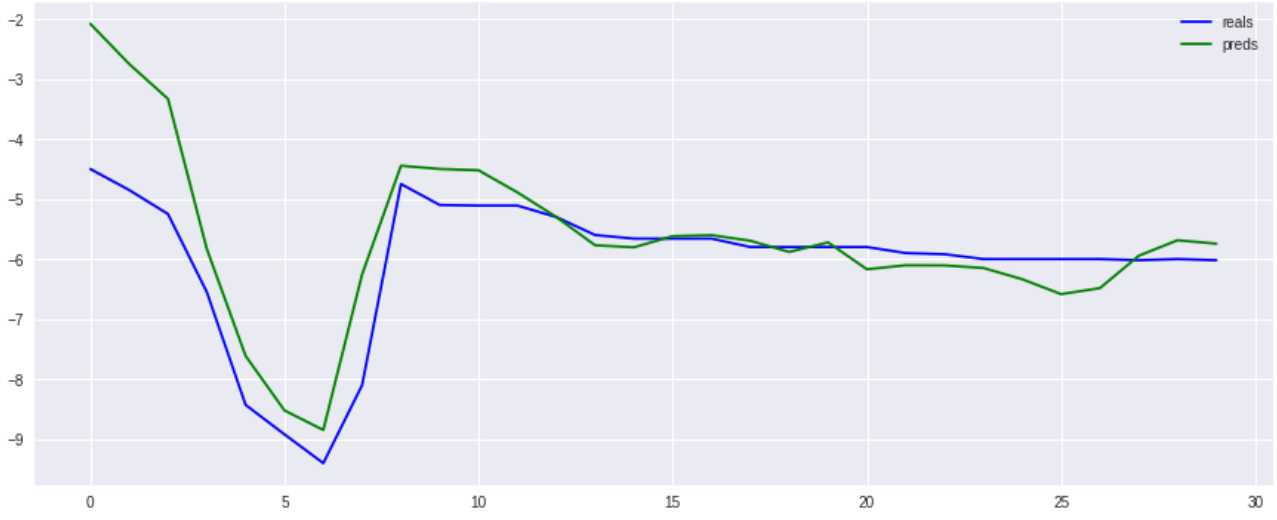


Figure29 Sample Prediction of Single RNN Architecture with GRU cell after 50 training epochs - state size 400

In figure 29 we see qualitatively what it means to have a model that better fits our price prediction task.

If we take a closer look in figure 29 we will notice that the prediction is much better, the further away we are in the future, in comparison to the first few days where the initial prediction is approximately minus 2 while the true value of the target is approximately between minus 4 and minus 5.

This pattern occurs in many instances and it suggests that there is something wrong with our model because it makes more sense for the prediction to be almost perfect at the beginning and worse further in future.

In fact the way we are modelling the prediction task on our single RNN architecture explains why this phenomenon occurs. Notice that when the first prediction is being produced only 30 inputs have been processed. When the second output/prediction has been produced then 30+1 inputs have been processed by the GRU and so on. So at the time that we are about to output the last and final 30th prediction the GRU has already processed the full 60 days of the input.
In other words since GRU has more information available at the end of the predictions it is able to make better predictions at the end rather than the beginning. So this explains our weird behavior of our model and guides us to build a new more suitable model for our price prediction task.

8.3 Sliding Window Model

In this model observations from the prior time steps (t-1 and before) are used to predict the observation at the current time step (t) [12]. In other words:

- we are using all of the 60 days of our input to predict only the next day
- we are stacking the prediction to our available inputs
- Using all the inputs plus the newly predicted value(s) to predict the next day in our forecast
- We repeat until we get 30 days which is our target length

Note in figure 30 that the graph looks a lot similar to our previous graphs but there is a large difference since only one day is being predicted at each training step in contrast to predicting the full length.

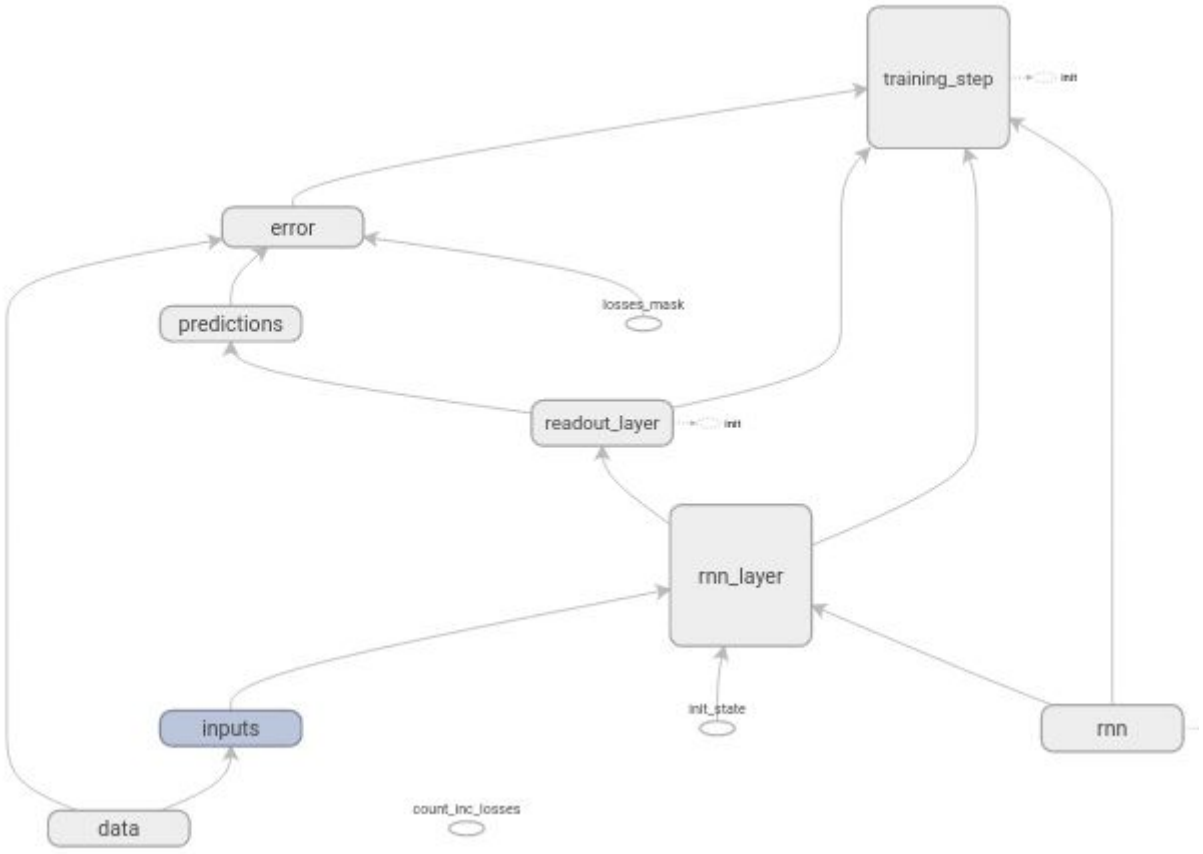


Figure30 Sliding Window Recurrent Neural Network Architecture Predicting one output at a time

8.3.1 Stackable Data Provider

A special data provider is needed for feeding data to the sliding window model because multiple training steps equal to the number of target length, here 30, are used to predict a single instance. More precisely, the inputs being fed to the model start with the original 60 inputs provided by the dataset and on every training step the predicted outputs are being stacked on the inputs and thus the inputs get a range of lengths [60, 90), until all the necessary outputs have been produced for our model. Similarly, the target sequence is not being fed into the model all at once but one at a time.

8.3.2 Hyperparameters

Hyperparameter	**Value**
State size	400
Batch size	47
RNN cell	GRU cell

8.3.3 Training

Training the network for 35 epochs we observe a rather higher Huber loss in comparison to our previous experiments and also contains a lot of oscillation. Moreover, it shows no signs of convergence, rather we observe divergence and an increasing value. This is shown in figure 31.

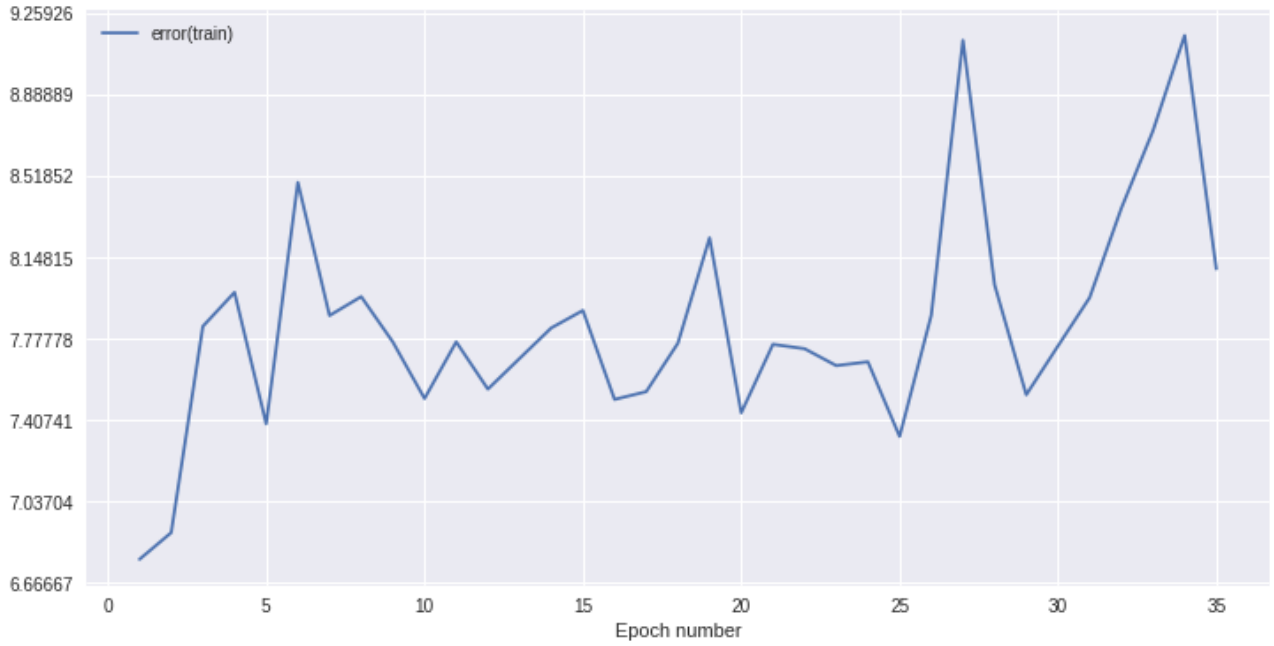


Figure31 Training Huber Loss Sliding Window Model - GRU cell - State size 400 - 35 epochs

8.3.4 Results

Metric	**Baseline**	**Sliding Window Model**
Training Huber Loss	4.78770150	8.33655582
Average DTW Score	153.97031739	258.45237553

The conclusion is that this model underperformed in comparison with our previous experiments and this worse performance can be attributed to a few factors.
Firstly the model is fed a multitude of instances, 30 times more than before, because the regression task is being applied on every one of the 30 days of the future/prediction period for each instance. This makes the generated instances come from a more complex distribution than before.

Another important factor is that the predictions themselves no matter how bad they can be, they contribute to the input for the prediction of the next day. We were expecting that this would make the convergence of the training a much more difficult task and indeed we observe this in figure 31 by large oscillations and divergence of the training error.

Finally, this architecture does not take directly into the account that there is a correlation between the outputs. Recall that each output is being produced separately on every training step. While we take the previous prediction into account to produce the next, this is not enough to build a model that is able to perform well enough for our price prediction task.

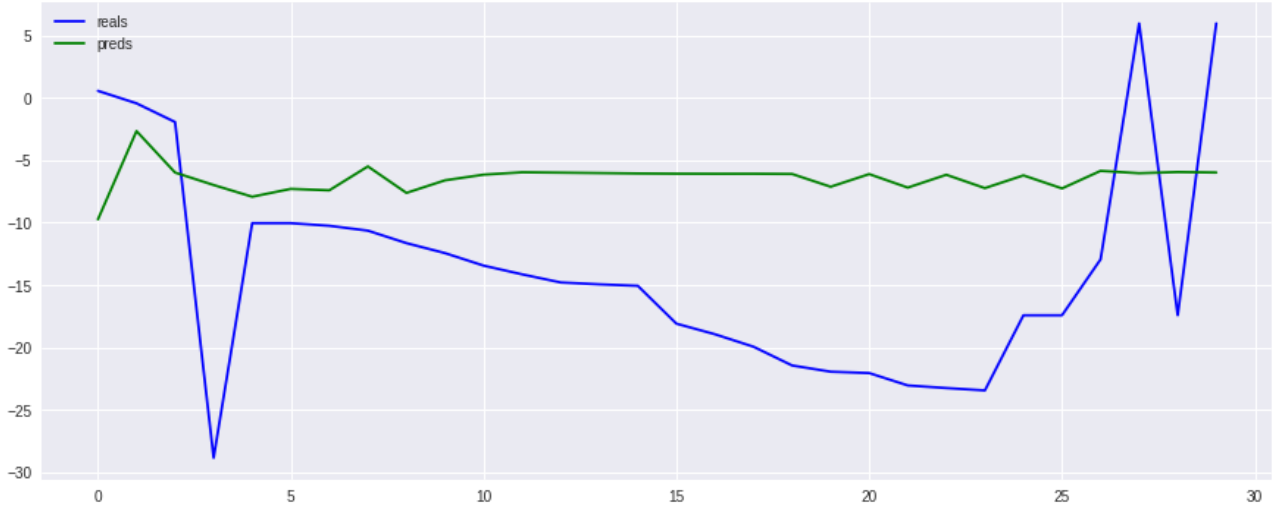


Figure32 Sample Prediction - Sliding Window Model - GRU cell - State Size 400 - after 35 epochs of training

The comparison of our metrics with the baseline and the qualitative review after observing predictions such as the one presented in figure 32 is a strong indication that we need to build a new more suitable model for our price prediction task that takes into account the sequential nature of the outputs.

Chapter 9
Sequence to Sequence Neural Network Modelling

Deep Multi Perceptron Neural Networks (MLPs) have excellent performance for inputs and outputs that can be represented as a vector of fixed length. But their architecture prevents them from mapping an input sequence of variable length to an output sequence of potentially also variable length. For solving such problems sequence-to-sequence neural network architectures have been developed [13]. The sequence-to-sequence models have been famously successful to machine translation tasks. More particularly the inputs sequence represents a phrase in a language and the output sequence should be the same phrase translated to another language. A picture of the main idea is displayed in figure 33.

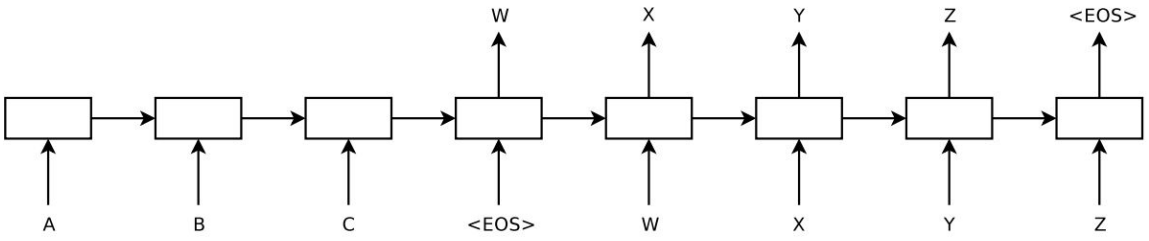


Figure33 Simple Sequence to Sequence Model with Dynamic Unrolling [13]

The above architecture consists of two Recurrent Neural Networks, the first one being the encoder which is being fed with the input and the second one being the decoder which produces the output.

The input RNN is unraveled in figure 33 and it takes three inputs: A, B and C.
Note that the outputs of the encoder RNN are discarded and only the inner state of the encoder is being used as the initial state for the decoder.
Similarly, the decoder is also presented unraveled in figure 33 which produces four outputs: W, X, Y, and Z.
The decoder takes as input what it was predicted in the previous step except the first step where something special has to be used as input.
It is important to note that for the translation task the model works best when a special word which is called, End Of Sequence (EOS) token is being fed as the first input to the decoder and also the decoder is expected to produce an EOS token after the translation of the full phrase has been generated.

9.1 Modelling Price Prediction Task as Sequence to Sequence

We can see our prediction task of being mapped to a sequence to sequence model.
The past, all the price history of the current and previous days of a certain mobile phone product, will be fed as input to the encoder RNN.
The future, all the prices that we want to predict, will correspond to the outputs of the decoder RNN.

By choosing this modeling we are hypothesizing that the past of a certain product contains enough information that will enable us to predict the price trend of the next 30 days.

We are not going to adopt the notion of EOS because this notion derives from the fact that the inputs and outputs in the translation task are words modeled with Word2Vec [14]. In Word2Vec model the words which are originally represented with one-hot encoding are transformed to word embeddings. Namely they become a vector in a common vector space for a particular corpus. Therefore, there is little relation between the case of word embeddings and our price prediction task where the inputs are real numbers and the task is regression and not classification.

Instead of using EOS token, as an initial input for our sequence to sequence model we are going to use the previous price, which corresponds to the last price of the input sequence, which represents the current day, the “now”. Because as we saw from the experiments related to baseline the latest value contains a lot of useful information.
Alternatively, we are going to use a zero vector as the first input to our decoder, by effectively expressing that there is no extra information to be provided and that the decoder has to predict the first output merely by its initial state, which is the final state of the encoder.
We will explicitly define when we use the one versus the other and for which reason.

9.2 Sequence to Sequence Model with Targets as Decoder Inputs

As a first model, we are going to use the dummy version of sequence-to-sequence where the inputs of the decoder are been provided from the targets.
In this architecture instead of having the decoder pass the predicted output as the next input of the decoder, we are directly providing the actual target as the decoder input making the problem much simpler for our model.
The future “is known” on training time but this model will not practically work in a testing situation, on a real scenario because the future will be unknown. We are only conducting the experiment to observe how the model is going to behave for later comparison.

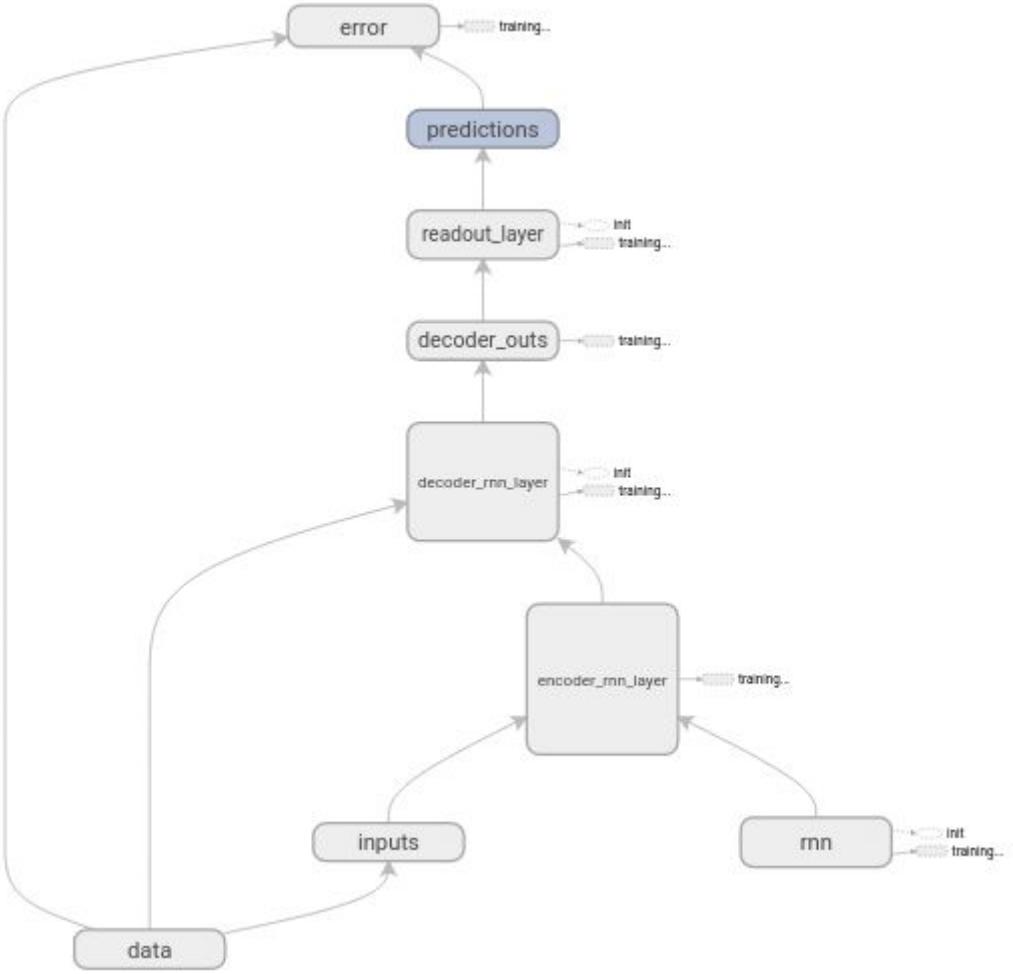


Figure34 Graph of Sequence to Sequence Model with Targets as Decoder Inputs

9.2.1 Hyperparameters

Hyperparameter	**Value**
Encoder/Decoder State size	400
Batch size	47
RNN cell	GRU cell
Optimizer / Learning Rate	Adam [21] / 1e-3

9.2.2 Training

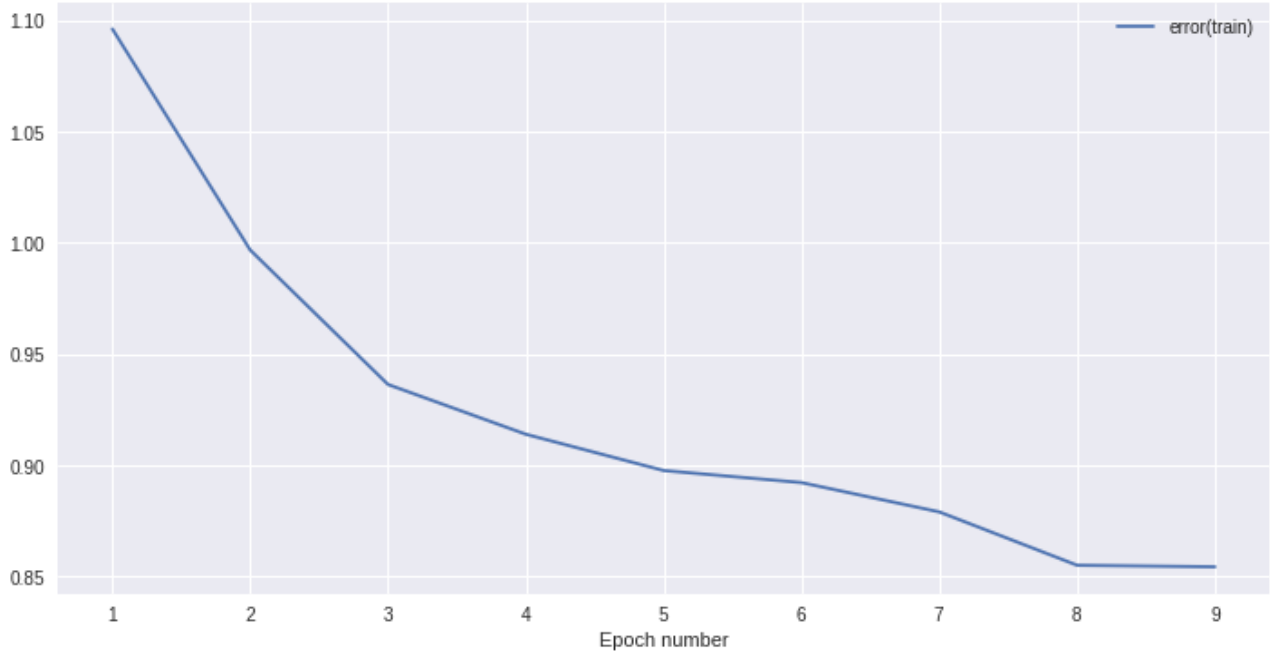


Figure35 Training Huber Loss - Sequence to Sequence Model with Targets as Decoder Inputs

9.2.3 Results

Metric	**Baseline**	**Sequence to Sequence Model with Targets as Decoder Inputs**
Training Huber Loss	4.78770150	0.82026691
Average DTW Score	153.97031739	14.33995578

As we see in figure 35 the training error is much lower than the baseline which is characterized as a fairly good performance of our model as it was expected.

On the other hand, when we plot our predictions, as seen in figure 36, we are able to give, qualitatively, a reason of why the performance is that good. The fact is that our model, and especially the decoder, having all this extra information about the previous predictions it uses it to build a rather dummy model. The optimization of the neural network results in a model which provides as outputs the decoder input. Which means that the decoder becomes a transparent layer, or the identity function, which simply provides as outputs whatever inputs come in.
In terms of minimizing the error, the easiest thing to do is use the previous day which here is known to the decoder. Therefore the resulting model is not expected to be able to generalize.

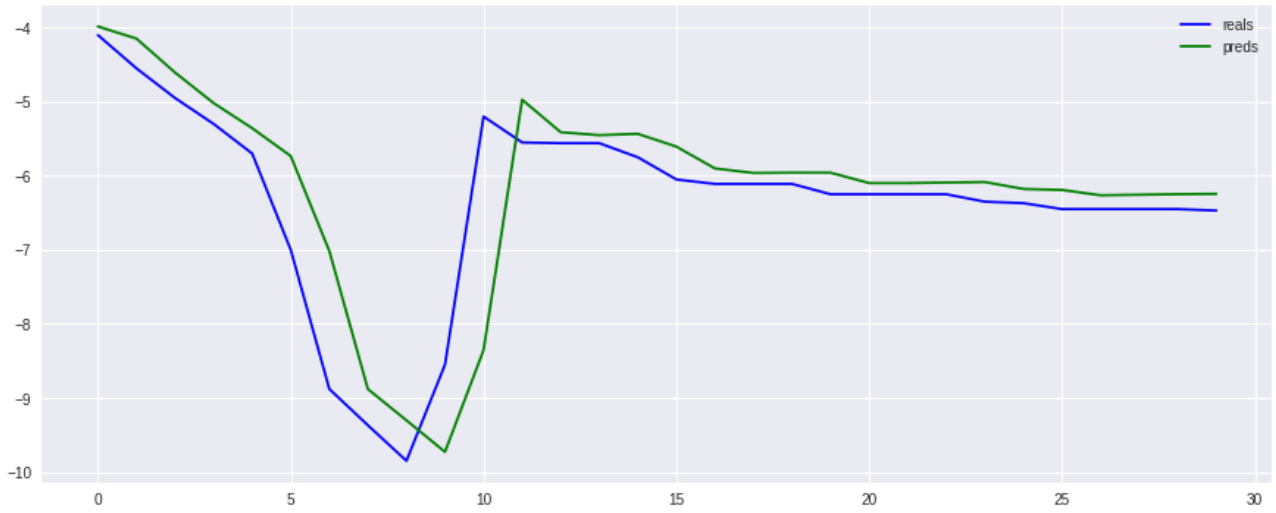


Figure36 Sample prediction - Sequence to Sequence Model with Targets as Decoder Inputs

9.3 Simple Sequence to Sequence Model

We see in figure 37 that the decoder is not receiving any input from the data meaning that the targets are not provided to the decoder as information anymore.

9.3.1 Hyperparameters

Hyperparameter	**Value**
Encoder/Decoder State size	400
Batch size	47
RNN cell	GRU cell
Optimizer	Adam [21]

9.3.2 Cross Validation

The hyperparameter that we seek to optimize is the Learning Rate.
We are going to take advantage of Gaussian Process Bayesian Optimization function gp_minimize [see Appendix 1] to find the optimal Learning Rate.

9.3.2.1 Cross Validation Params

Number of splits: 5-folds
Number of Random Initializations: 5
Number of Bayesian Optimization Calls: 10
Range of Allowed Values of Learning Rate: [1e-5, 1e-1] with log-uniform distribution

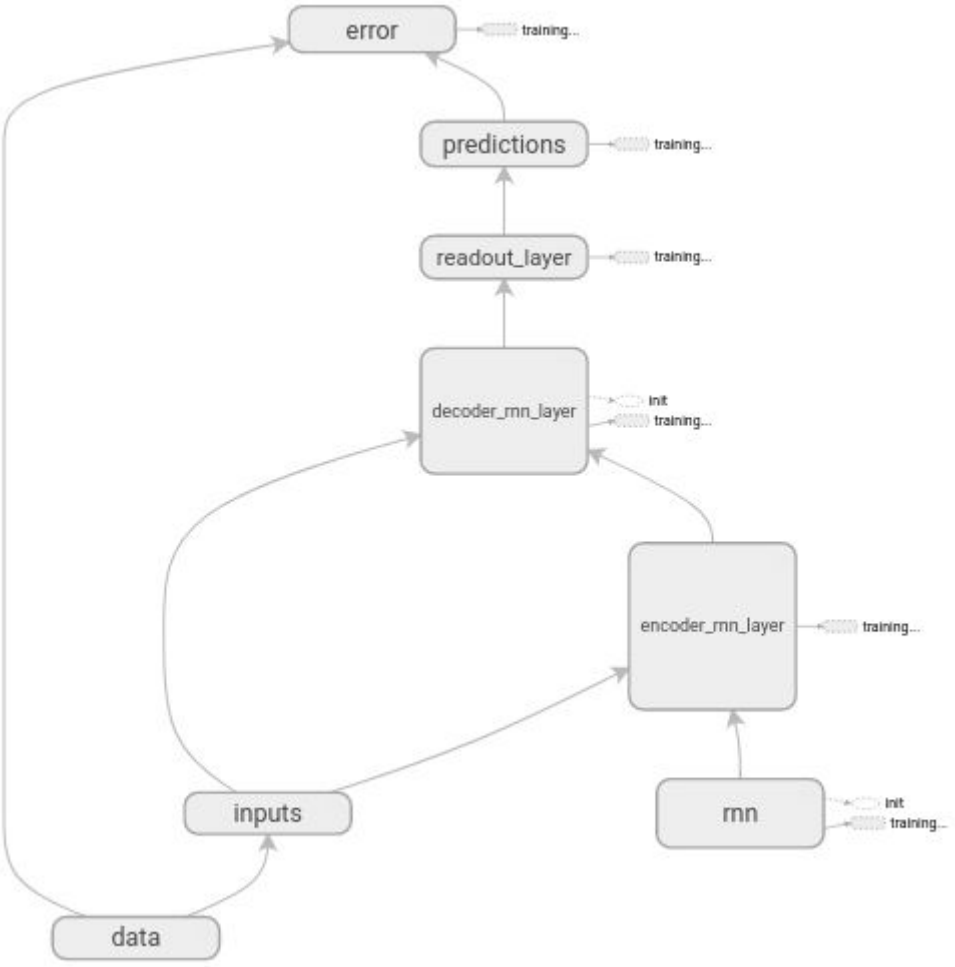


Figure37 Graph of Simple Sequence to Sequence Model for Price Prediction

After executing the cross validation we get the Convergence Plot as seen in figure 38.
Notice that convergence occurs within the first four calls which could be interpreted in different ways.
We could consider that the learning rate is not playing a big role on the performance of the training which would be rather unlikely since in many problems involving neural networks has been shown [15] that the learning rate hyperparameter plays a significant role to the variance of the training error.
Taking this into consideration we must take this result as an indication that our current model is not able to perform on the very difficult price prediction problem.

In figure 39 we see the cross validation score, which here is the average Huber loss for each one of the five trainings of the neural network for each one of the learning rates that were either randomly initialized or chosen by the Bayesian Optimization Gaussian Process. Notice that apart from a few experiments that showed much larger CV score the rest show a similar performance and a smooth training error curve.

Optimal Learning Rate: 0.0026946

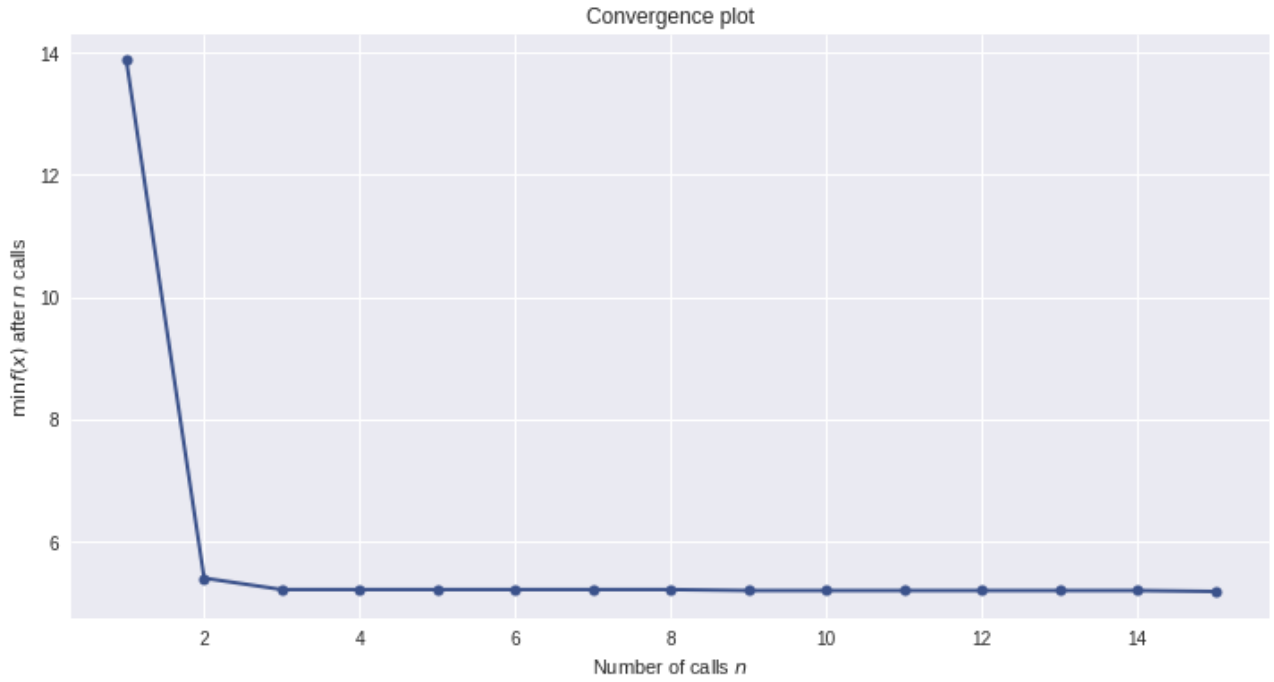


Figure38 Convergence Plot - GP Bayesian Optimization of Learning Rate - Simple Sequence to Sequence Model

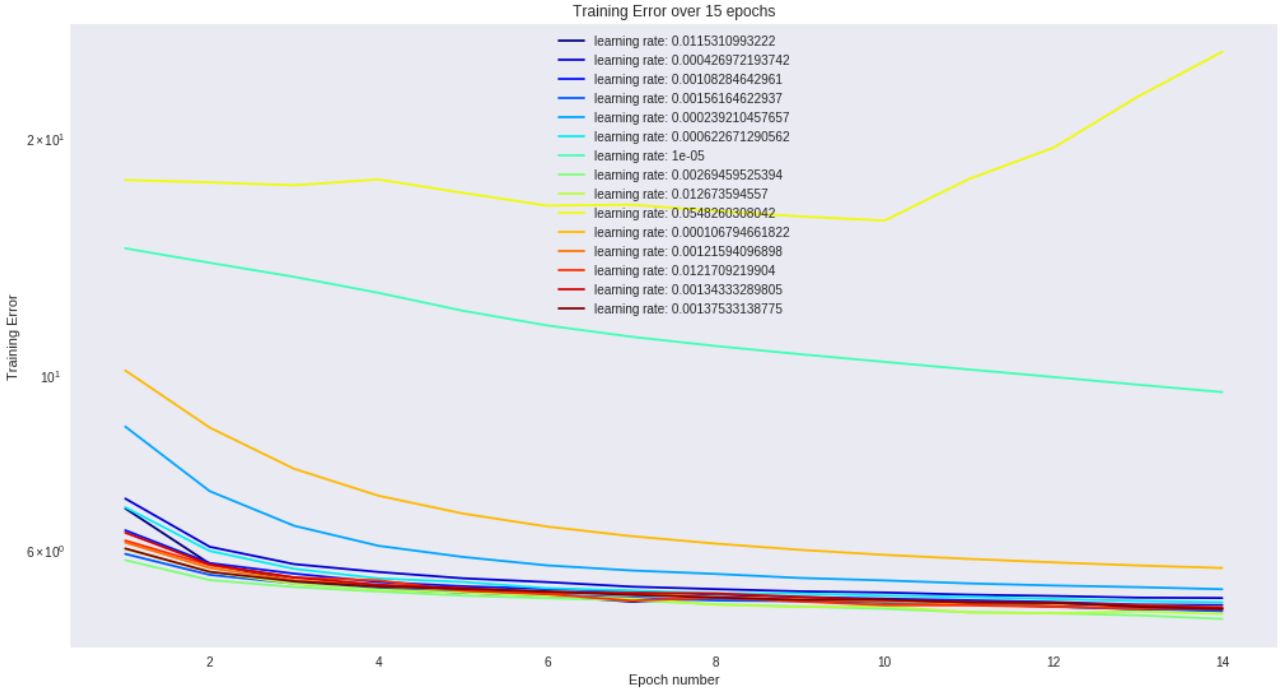


Figure39 Average Training Error (Cross Validation Score) for each learning rate - GP Bayesian Optimization - Simple Sequence to Sequence Model - Log Scale of y-axis

9.3.3 Training

So by using the optimal learning rate and keeping the rest of the hyperparameters intact, we are training our optimized model for 200 epochs to let it converge and get the minimum possible training error as seen in figure 40.

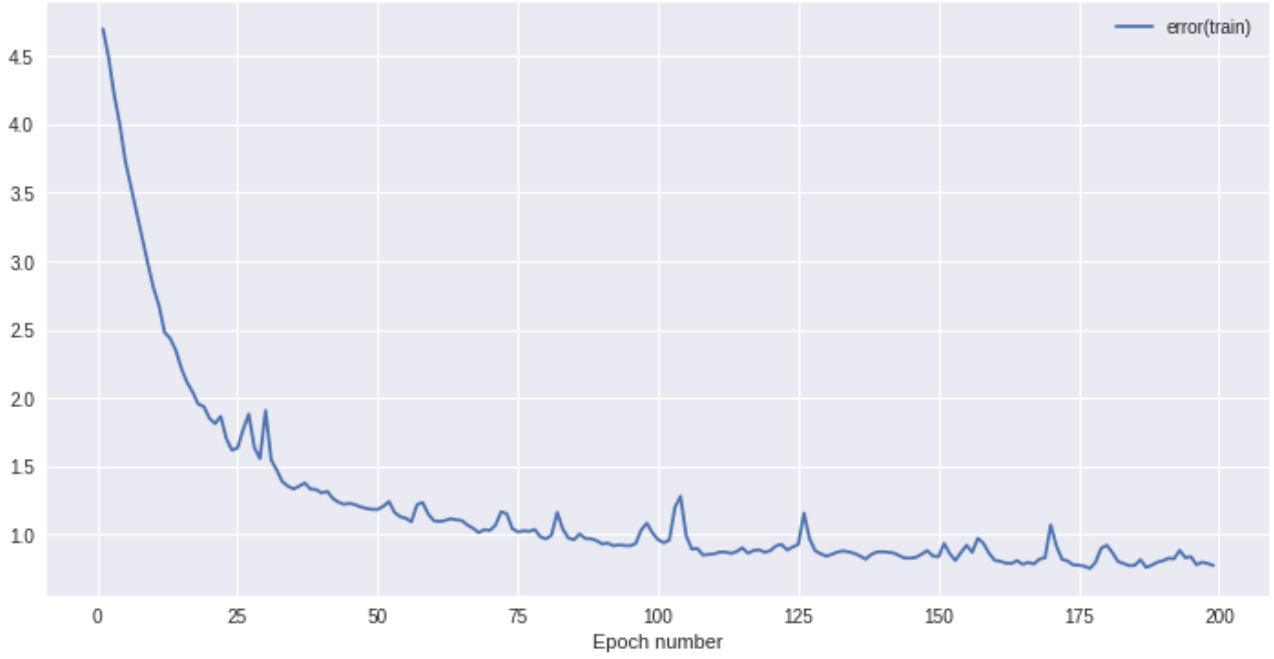


Figure40 Training Huber Loss - Simple Sequence to Sequence Model with Optimized Learning Rate

9.3.4 Results

Metric	**Baseline**	**Sequence to Sequence Model with Targets as Decoder Inputs**	**Simple Sequence to Sequence Model for Price Prediction**
Training Huber Loss	4.78770150	0.82026691	0.703325796
Average DTW Score	153.97031739	14.33995578	24.855095998

In the results table above, we are comparing the current model with our baseline and with our previous model.
Achieving better results in both metrics when comparing our current model with baseline translates into a good enough model.

The most interesting conclusion comes when comparing the previous model, where the decoder inputs were fed with the true values, with the current model that it can only be based on the predictions to produce the final 30 days output. The current model after 200 epochs of training

managed to achieve a lower training Huber loss than the previous model. However, the Average DTW Score of the previous model is superior to that of the current model. This means that even if the optimizer managed to bring the parameters of the neural network to a state that the training loss is low the final result does not resemble the original target sequences, as well as the previous model, could. This is visible in figure 41 where we see a sample of a prediction. We notice that the predicted output sequence is able to "follow" the trend of the target sequence but in a more smooth way by missing a lot of the fine detail of sudden changes and edges that the target sequence consists of.

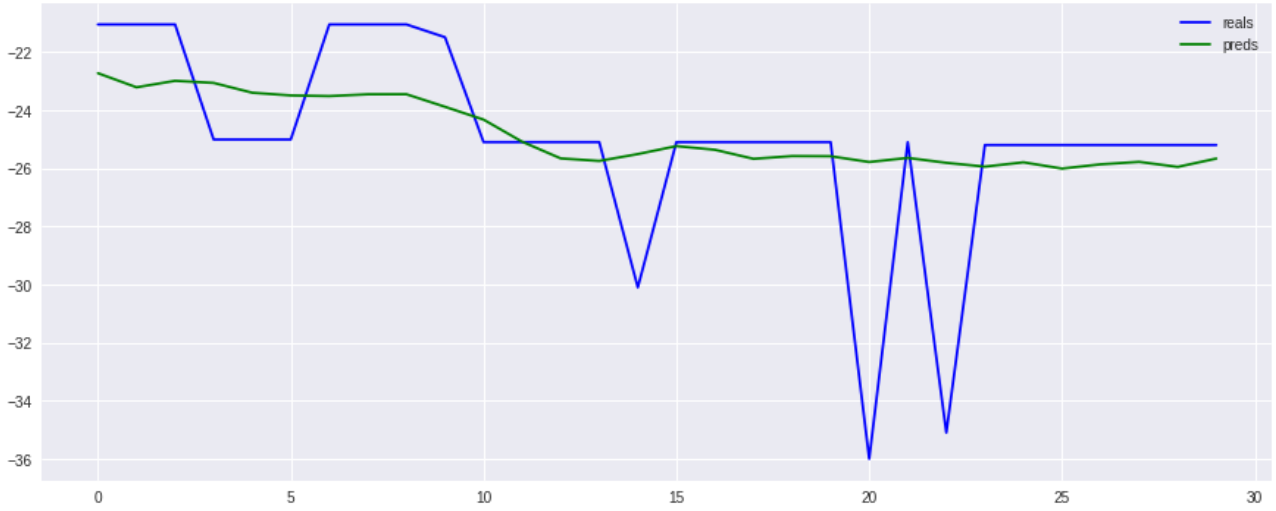


Figure41 Sample Prediction from Simple Sequence to Sequence Model with optimal learning rate after 200 epochs of training

9.4 Conclusions

So far we have managed to build a model that performed better than the baseline and produced outputs that could resemble the targets, especially in terms of following the trend.
However, these models succeeded in the restricted set of only 95 price histories which all share the same length of sequence, 210 days.
The above conditions have made the training easier for the model because:

- the price histories could share a few similarities since the products already existed in the market for some time
- We did not train against really old models since the minimum date of these sequences is 11th Nov 2016
- There is not a large variety of patterns since there are only 95 price histories in comparison to the total number of price histories

We had a chance to experiment faster with this subset but we cannot derive any finite conclusions. We need to train a single model for the entirety of our price histories before claiming that modelling with sequence to sequence neural networks can forecast the price of mobile phone products.

Chapter 10
Advanced Sequence to Sequence Models

Taking into account the conclusion of the previous chapter, we are going to be using the dataset that was created from the full range of price histories and train our new models on that. No further preprocessing is being made. Notice that the baseline does not need to be recalculated because the baseline metrics were already calculated for the generated sequences that correspond to the full range of the price histories.

10.1 Sequence to Sequence Model with Batch Normalization, Dropout and Dynamic Decoder Inputs

We are building a more advanced model which contains a few extra layers which have been shown to improve the performance of neural networks. We see the full graph of our model in figure 42.

As the inputs flow through the neural network the weights and biases are adjusted by the optimizer making them too big or too small. This is called internal covariate shift and by normalizing the data in each layer we are minimizing this issue from occurring [16]. Therefore batch normalization helps the learning process because the weights do not experience large and noisy fluctuations. With improved learning capabilities the neural network is usually able to achieve lower overall error in fewer epochs. Our model does not contain lots of layers and because we are doing regression task we should not put a batch normalization layer at the end since this would prevent outputs of being scaled as much as it is necessary. We are adding a single batch normalization layer on top of the input layer as seen in the “batched_inputs” node in the graph in figure 42 and thus we are expecting only a slight increase in the learning performance.
We have set a boolean parameter to control whether the batch normalization layer is enabled or bypassed.

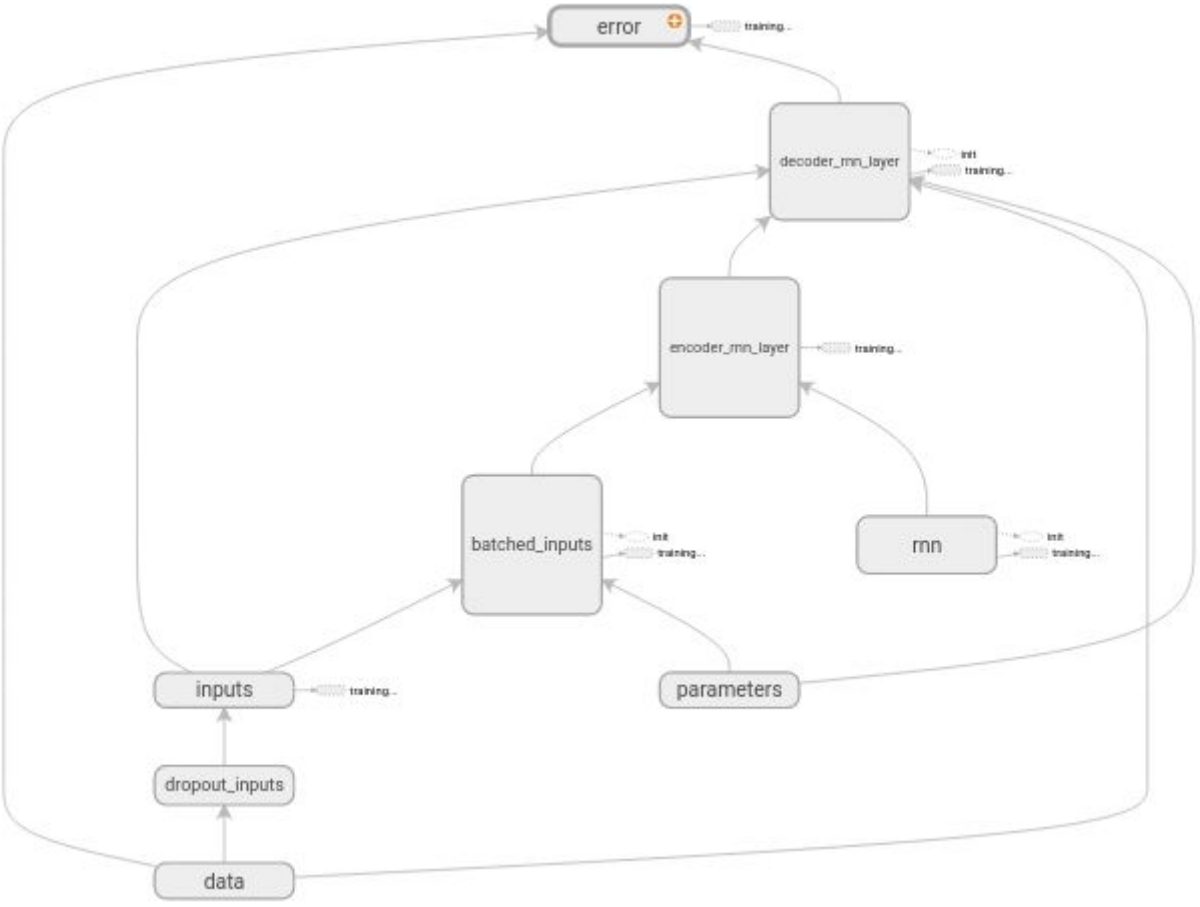


Figure42 Graph of Sequence to Sequence Model with Batch Normalization, Dropout and Dynamic Decoder Inputs

Originally Dropout [17] is used to avoid unnecessary co-adaptation between neurons when start training the entire neural network altogether. This allows Dropout to simulate approximately an exponential number of neural networks. We are adding Dropout in our model as a means of giving variations of our inputs to the model. The model should be robust enough to be able to predict the same targets when the inputs contain "salt-n-pepper" noise which is the effect that dropout has to the input sequence.

Note that by setting the Keep Dropout Input Probability equal to 1.0 = 100% we are effectively disabling the dropout layer.

Moreover, we have enabled to dynamically control the inputs of the decoder. We have created a flexible model where the decoder inputs are given by the following pseudo-formula:

$$decoder\ input = (target * alpha) + (prediction * (1 - alpha))$$

Where alpha acts as a fraction/percentage and is taking values in the [0, 1] range.

Alpha represents the percentage that we are allowing the target to be used as the next decoder input. The rest of the percentage is given to the prediction.

Finally, we have generated a second version of preprocessed targets to help our model. We have removed the bias by subtracting from the target sequence the last value of the input sequence. This transformation will not result in all targets starting from exactly zero but given the fact that price histories are continuous values we are expecting that the initial value of the targets will be

near to zero. The need for this transformation originates from observing the prediction graphs of previous experiments such as in figure 32 and figure 41. We are expecting that by having removed this bias from each target the model will more easily predict the initial days of the output sequence.

10.1.1 Hyperparameters

Hyperparameter	**Value**
Encoder/Decoder State size	400
Batch size	50
RNN cell	GRU cell
Optimizer	Adam [21]
Learning Rate	0.0026946

Note that we are using the optimal Learning Rate from the previous model as an initial value. Dropout Probability Input has been chosen relatively high because if it is too low then most of the values of the input sequence will be zero and thus we will be asking our model to be trained on very different inputs than the ones we actually have. Note that alpha parameter for controlling the decoder input percentage is used here as a form of pretraining. We are expressing to the optimizer that the decoder inputs should ideally be equal to the targets and then we gradually remove this information from the decoder to explore whether this pretraining has helped the new predictions.

10.1.2 Training without normalized targets

First we are training our model by disabling all the extra implemented features and without normalized targets.

10.1.2.1 Extra Hyperparameters

Hyperparameter	**Value**
Keep Probability of Dropout Input	100%
Initial Decoder Input	Previous Day - Last Input
Targets with Removed Bias	No
Alpha (Decoder Input Control Percentage)	100% for epoch 1 75% for epoch 2 50% for epoch 3 25% for epoch 4 0% for all other epochs

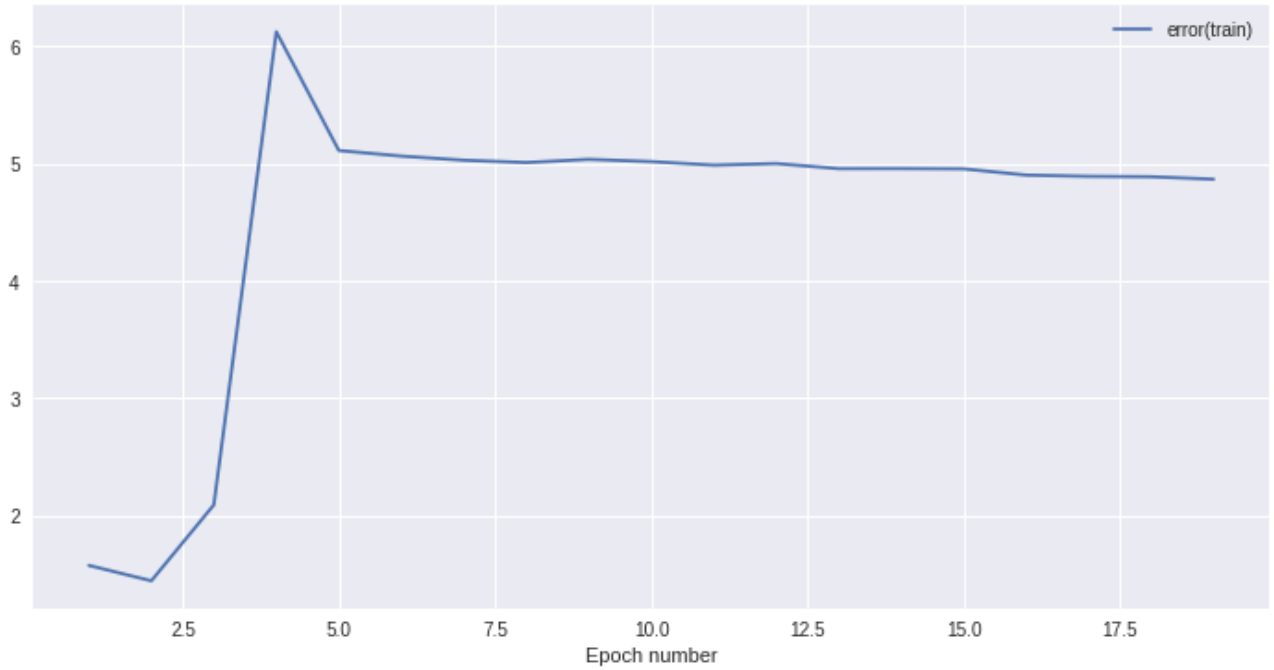


Figure43 Training Huber Loss - Advanced Sequence to Sequence with Decoder Input Control for Pretraining

As we see in figure 43 the training error becomes smaller than baseline when the decoder is fed with the true targets but as the epochs go along and at epoch five the decoder is fed with its own predictions the training error becomes quickly larger than the baseline and the training is very slow. At the end of training, we still have not reached below the baseline.
This is a strong indication that creating a single neural network model to be a predictor of any kind of mobile phone product, in other words, to work for all price histories might be much harder than the previous experiments have indicated.

10.1.3 Training including normalized targets and zero initial input for the decoder

We are using the normalized targets that we created by removing their bias. So in response to that, we are setting the initial input of the decoder to zero. We are expecting that the decoder having as first input a vector of zeros will be able, from the initial state alone, to start with a better prediction.

Note that we are also letting the pretraining with the decoder inputs being fed from targets to last as long as half of the total number of training epochs.

10.1.3.1 Extra Hyperparameters

Hyperparameter	Value
Keep Probability of Dropout Input	100%
Initial Decoder Input	Vector of Zeros
Targets with Removed Bias	Yes
Alpha (Decoder Input Control Percentage)	100% for epoch 1 and linear progression to 0% for half of the epochs 0% for all other epochs

In figure 44 we are getting a similar behavior as the in the previous experiment. The more epochs for extra pretraining did not help, the training is low and final training Huber loss is above baseline.

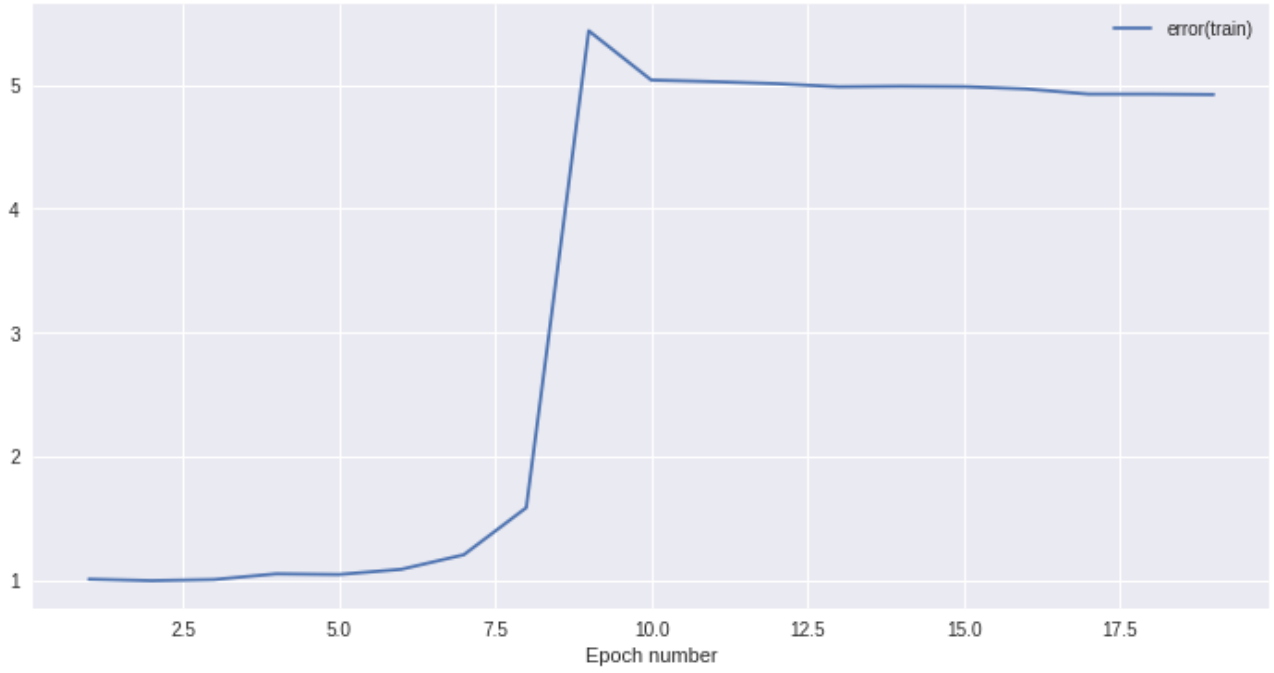


Figure44 Training Huber Loss - Advanced Sequence to Sequence with Decoder Input Control for Pretraining and Normalized Targets

10.1.4 Training including Batch Normalization and Dropout

We are adding Batch Normalization and Dropout to our model to see if we will get any improved performance.

As we see in figure 45, the performance is similar to the previous experiments and training is even slower or even static after epoch 20. It seems that the added layers have been more detrimental rather than helpful to our model.

10.1.4.1 Extra Hyperparameters

Hyperparameter	Value
Keep Probability of Dropout Input	70%
Initial Decoder Input	Vector of Zeros
Targets with Removed Bias	Yes

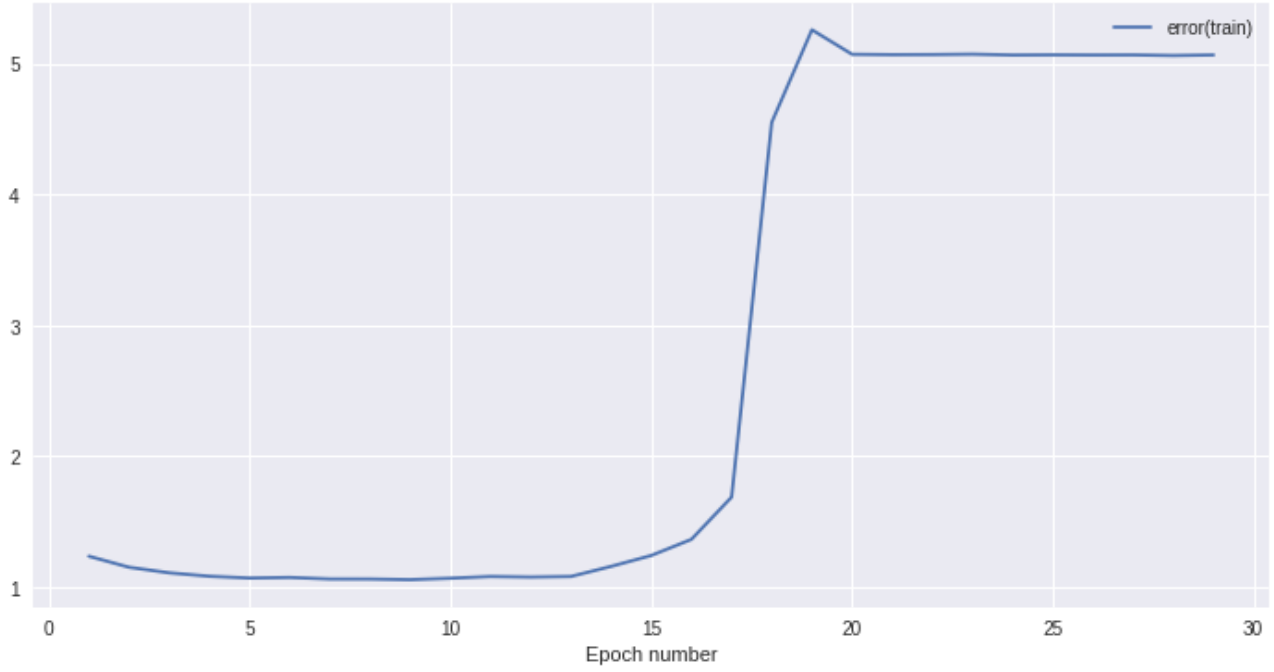


Figure45 Training Huber Loss - Advanced Sequence to Sequence with Decoder Input Control for Pretraining, Normalized Targets, Batch Normalization and Dropout

10.1.5 Results

Metric	Baseline	Training without normalized targets	Training including normalized targets and zero initial input for the decoder	Training including Batch Normalization and Dropout
Training Huber Loss	4.78770150	4.81190285	4.96664515	5.0690297985
Average DTW Score	153.97031739	153.1493629	158.37255693	162.57913656

From the results table, we note that the evaluation metrics were getting worse as we introduced the new implementations to the model.
The most important conclusion though is that for all the models the metrics are higher than the baseline meaning that the models were not able to be trained for the current hyperparameters for 20 epochs. This means that including all the price histories makes the problem of building a single model for price prediction, for all mobile phone products, much harder. Next we are researching:

- Building a more advanced model with more parameters
- Providing additional information to the model to help the training process
- Preprocessing the inputs to help the training of the neural network

10.2 Normalizing Scale of Price Histories

Apart from removing the bias from our price histories in order to help the training of our model we are going to remove the scale as well. Since all price histories consist of prices we are creating a single attribute, a long vector, which consists from all of our prices in all of our price histories. We calculate the standard deviation of this long vector.

Standard Deviation of all Price Histories: 186.5695327494079

We divide all prices of price histories with the standard deviation to remove scale.

10.3 New Baseline of Normalized Prices

We generate new baseline metrics using again our dummy predictor which produces always the same constant value as the last seen price.

Average Huber Loss	0.005683492766
Average DTW Score	1.131402364

In figure 46 we see the distribution of the Huber loss for our dummy predictor. Notice that because some prices do not change at all, or change insignificantly, the dummy predictor is able to provide a good prediction sequence for such prices, near zero. On the other hand, there are a few cases where the dummy prediction is not suitable for highly dynamic prices and therefore we see a long tail of the distribution in figure 46.

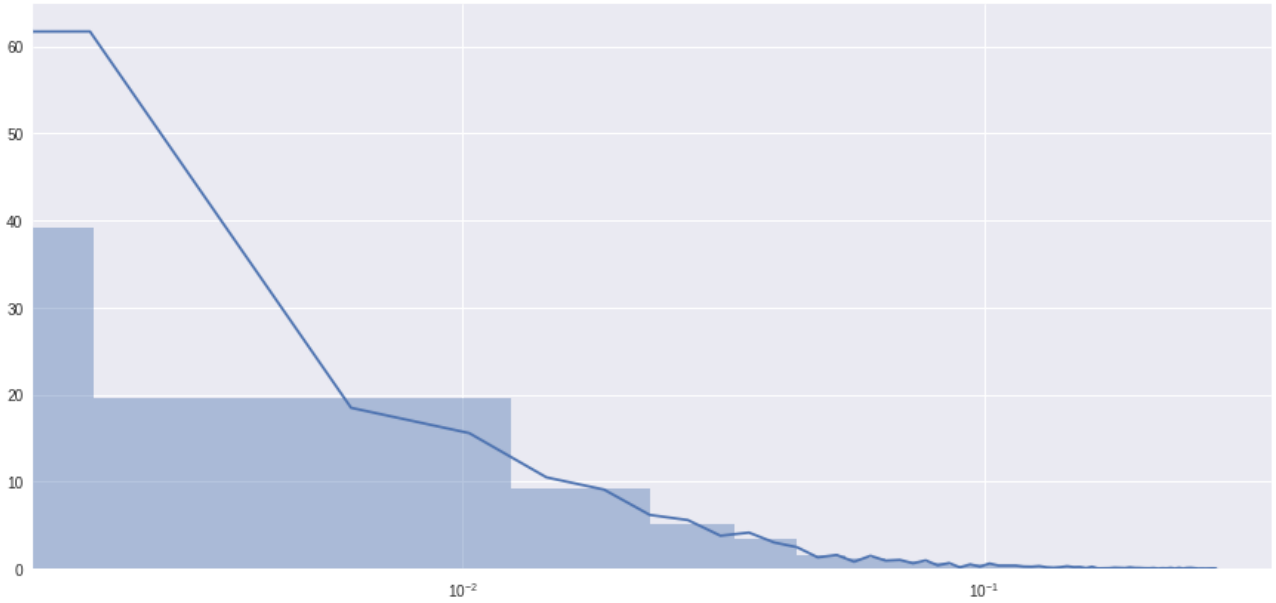


Figure46 Huber Loss Distribution of Dummy Predictor on Scale-Normalized Price Histories

10.4 Sequence to Sequence Model with Batch Normalization, Dropout, Dynamic Decoder Inputs and Normalized Price Histories

We are using the same model we used in the previous experiment but we are training with the normalized price histories. Our expectation is that the model will not have to accommodate the different scales of the price histories and that it will perform better.
Note that the graph of the model has not changed.

10.4.1 Hyperparameters

Hyperparameter	**Value**
Batch size	64
RNN cell	GRU cell
Optimizer	Adam [21]
Alpha (Decoder Input Control Percentage)	100% for epoch 1 0% for all other epochs

Note that the decoder inputs will be given by the targets as a form of very short pretraining of only one epoch and the rest of the epochs are used for actual training where the decoder only relies on its own predictions as inputs.

10.4.2 Cross Validation

We are going to use Bayesian Optimization to optimize important hyperparameters altogether by using a multidimensional Gaussian Process for the space that all of these hyperparameters lie on.

10.4.2.1 Hyper Parameters for Optimization

Hyperparameter	**Minimum Value**	**Maximum Value**	**Prior Distribution**
Keep Probability of Dropout Input	0.5	1.0	uniform
Number of Units of GRU cell	50	500	uniform
Learning Rate	1e-6	1e-3	log-uniform

10.4.2.2 Cross Validation Params

Number of splits	5
Number of Random Initializations	2
Number of Bayesian Optimization Calls	15
Epochs	15

The objective function is returning the value of the average DTW score on the validation set.
In figure 47, we see that the optimization has converged within the first eight calls.
In the table below we see the optimal values of the hyperparameters as are derived from the Gaussian Process Bayesian Optimization.

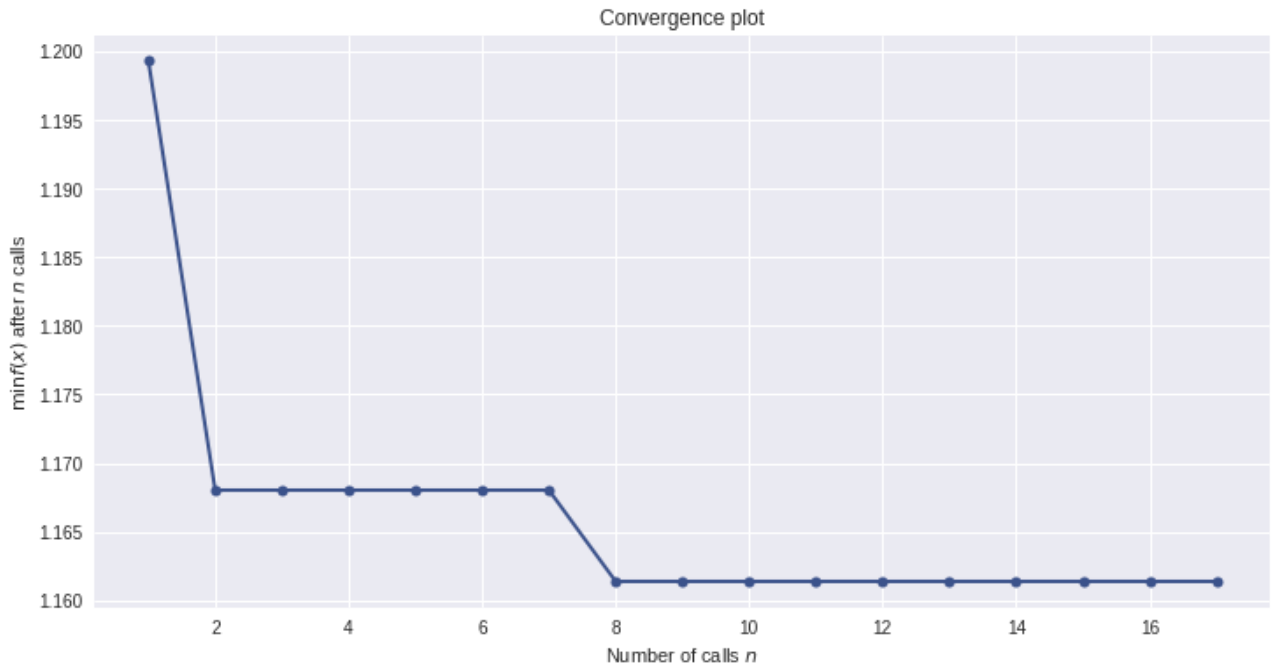


Figure47 Convergence - GP Bayesian Optimization - Learning Rate, Units of GRU cell and Keep Probability of Dropout Input - Sequence to Sequence Model with Normalized Prices

10.4.2.3 Optimized Hyperparameters

Hyperparameter	**Optimal Value**
Keep Probability of Dropout Input	0.624880 ~=62.5%
Number of Units of GRU cell	500
Learning Rate	1e-3

It is interesting to note that the optimal number of units of the GRU cell is equal to the maximum number of the GRU units we had set as an upper boundary to the optimization process. This is an extra indication that our model is inadequate for being a single model for the very difficult price prediction task without making it conclusive.

10.4.3 Training

Using the optimal hyperparameters and keeping the rest of the hyperparameters intact, we are training our optimized model for 100 epochs to let it converge and get the minimum possible training error as seen in figure 48.

10.4.4 Results

Metric	Baseline	Sequence to Sequence Model with Batch Normalization, Dropout, Dynamic Decoder Inputs and Normalized Price Histories
Training Huber Loss	0.005683492766	0.00320502
Average DTW Score	1.131402364	1.153435198

An immediate conclusion from the results of this training is that the training Huber loss is below the baseline while the average DTW score is above the baseline.
The disadvantages of using a separate yet differentiable evaluation metric from the evaluation metric that we really try to optimize are highlighted in the current experiment.
The optimizer brought the training Huber loss to a low value while the predicted sequences have characteristics similar to the sample prediction in figure 49.
The prediction sequence smoothly follows the general trend of the prediction. The prediction at best is a very low resolution, of the actual price sequence. Therefore it makes sense that the average DTW score is above the baseline.

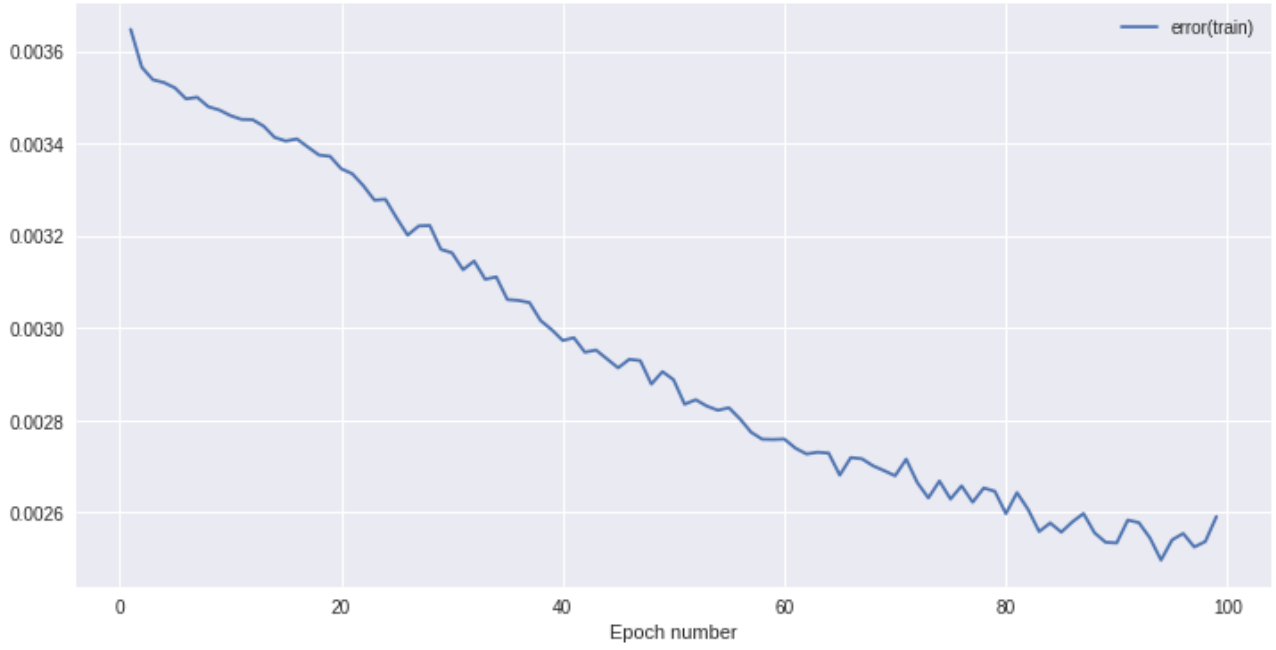


Figure48 Training Huber Loss - Sequence to Sequence Model with Optimal Hyperparameters - Normalized Price Histories

.

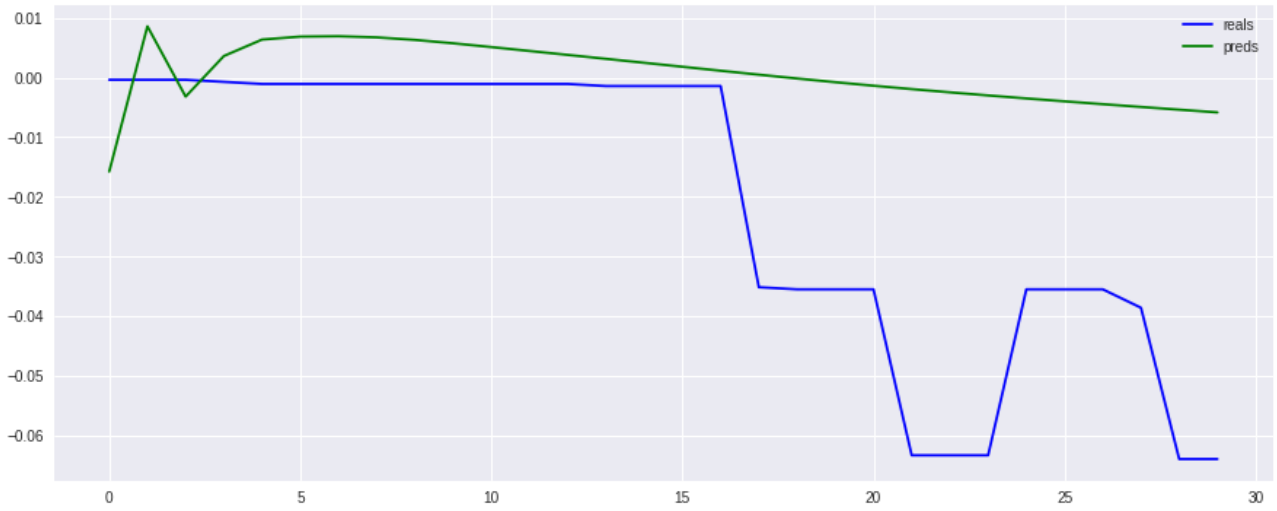


Figure49 Sample Prediction - Sequence to Sequence Model with Optimal Hyperparameters - Normalized Price Histories

10.5 Sequence to Sequence Price History and Static Mobile Attributes Model

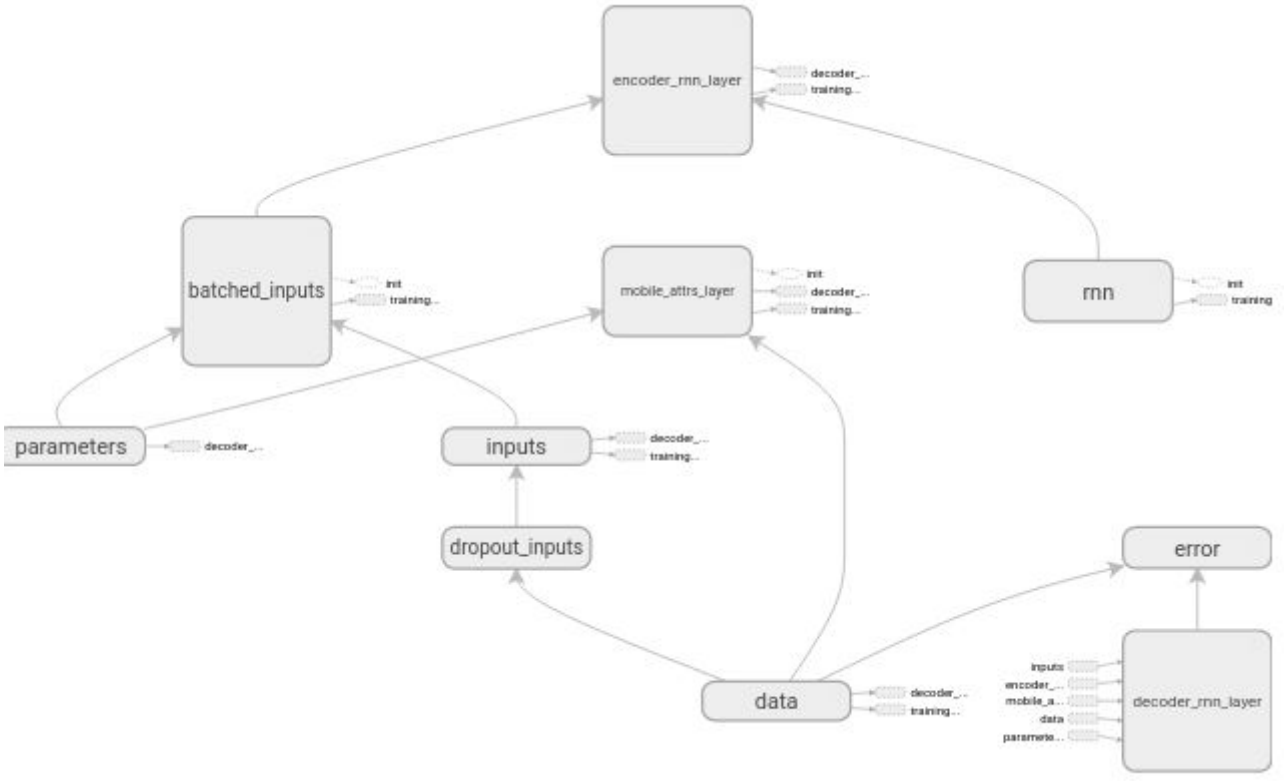


Figure50 Graph of Sequence to Sequence Price History and Static Mobile Attributes Model

So far we have used as the only information passed to our model the price histories ignoring all the rest of the attributes that are available for each mobile phone product.
To add the static mobile attributes in our model, we are adding three MLP layers.
The first MLP layer is the one visible at the middle of the graph in figure 50 and its role is to create more meaningful features out of the raw static mobile attributes which are going to be helpful to our very difficult problem of price prediction.
The output of the decoder is being processed by an MLP Layer as well. There is also a Dropout layer being added between the decoder outputs and this MLP layer.
The outputs of the above two layers are concatenated and used as input to a final affine layer which produces the prediction.
By providing extra information we are enabling our model to differentiate its prediction based also on the static mobile attributes. We recall that these attributes contained useful information regarding the price as it was proved by learning a ridge regression model.

10.5.1 Hyperparameters

Hyperparameter	**Value**
Batch size	100
RNN cell	GRU cell
Optimizer	Adam [21]
Alpha (Decoder Input Control Percentage)	100% for epoch 1 0% for all other epochs

10.5.2 Cross Validation

We are going to use Bayesian Optimization to optimize important hyperparameters altogether by using a multidimensional Gaussian Process for the space that all of these hyperparameters lie on.

10.5.2.1 Hyper Parameters for Optimization

Hyperparameter	**Minimum Value**	**Maximum Value**	**Prior Distribution**
Decoder Dropout Keep Probability	0.2	1.0	uniform
Number of Units of GRU cell	300	700	uniform
Learning Rate	1e-6	1e-2	log-uniform
Decoder Output Hidden MLP Layer Dimensionality	100	500	uniform

Mobile Static Attributes MLP Layer Dimensionality	200	600	uniform

Due to the results of the previous experiment we are expanding the maximum number of possible GRU units from 500 to 700 to cover for a more complex state that could be required.
For the same reason, the maximum possible learning rate is increased from 1e-3 to 1e-2.
Note that the range of the dimensionalities of the MLP layers is a rough guess as we do not have any prior knowledge of what could be required.

10.5.2.2 Cross Validation Params

Number of splits	5
Number of Random Initializations	2
Number of Bayesian Optimization Calls	15
Epochs	15

The objective function is returning the value of the average DTW score on the validation set.

In figure 51 we see that the optimization has converged within the first nine calls.

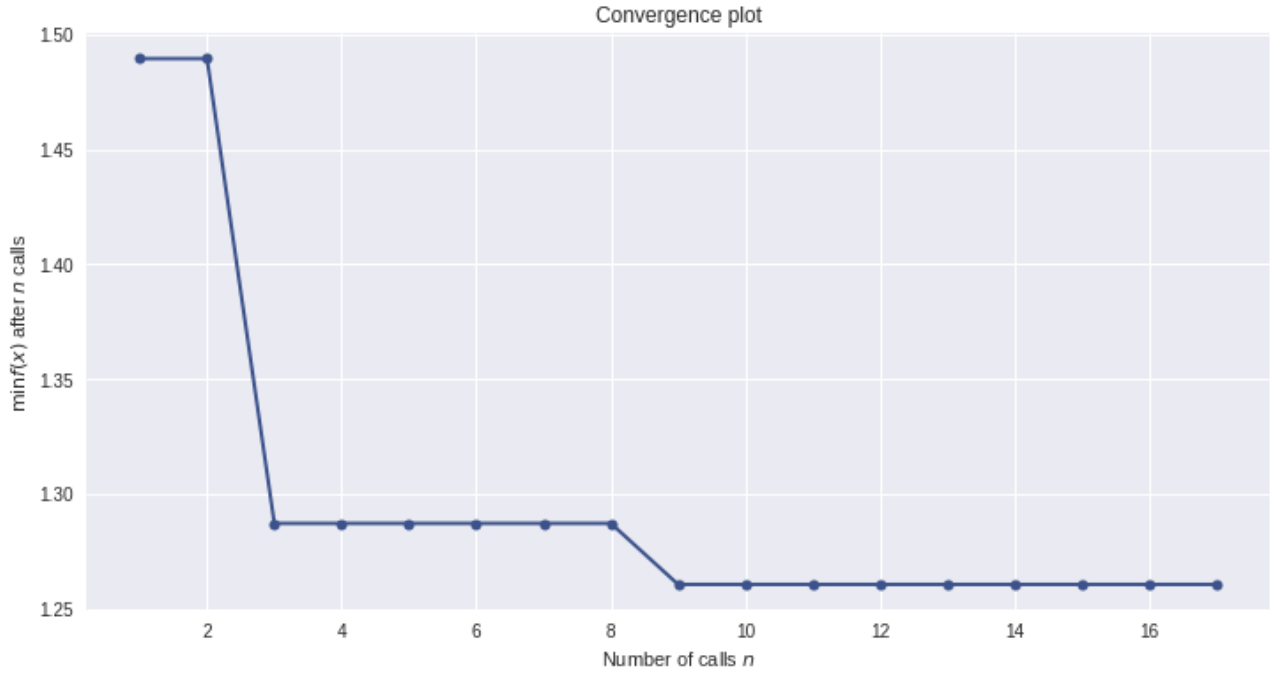


Figure51 Convergence - GP Bayesian Optimization - Sequence to Sequence Price History and Static Mobile Attributes Model

10.5.2.3 Optimized Hyperparameters

The optimization here has resulted that the best dropout keep probability for the decoder output layer is the lowest available probability we provided as boundaries to the bayesian optimization.

This effectively cancels out most of the information coming from the decoder and makes the price prediction depending heavily on the static mobile attributes.

Hyperparameter	**Optimal Value**
Decoder Output Dropout Keep Probability	0.2 = 20%
Number of Units of GRU cell	650
Learning Rate	0.00081953012
Decoder Output Hidden MLP Layer Dimensionality	320
Mobile Static Attributes MLP Layer Dimensionality	515

10.5.4 Training

Using the optimal hyperparameters and keeping the rest of the hyperparameters intact, we are training our optimized model for 100 epochs to let it converge and get the minimum possible training error as seen in figure 52.

10.5.5 Results

In the results table we are including the model of the previous experiment for comparison reasons.

Metric	**Baseline**	**Sequence to Sequence Model with Batch Normalization, Dropout, Dynamic Decoder Inputs and Normalized Price Histories**	**Sequence to Sequence Price History and Static Mobile Attributes Model**
Training Huber Loss	0.00568349	0.00320502	0.00314844
Average DTW Score	1.13140236	1.15343520	1.17044361

We have the same phenomenon of the previous experiment where the training Huber loss is smaller than baseline but the average DTW score is larger, only amplified. Indeed, the training Huber loss is slightly smaller than the previous experiment but the average DTW score is significantly larger.

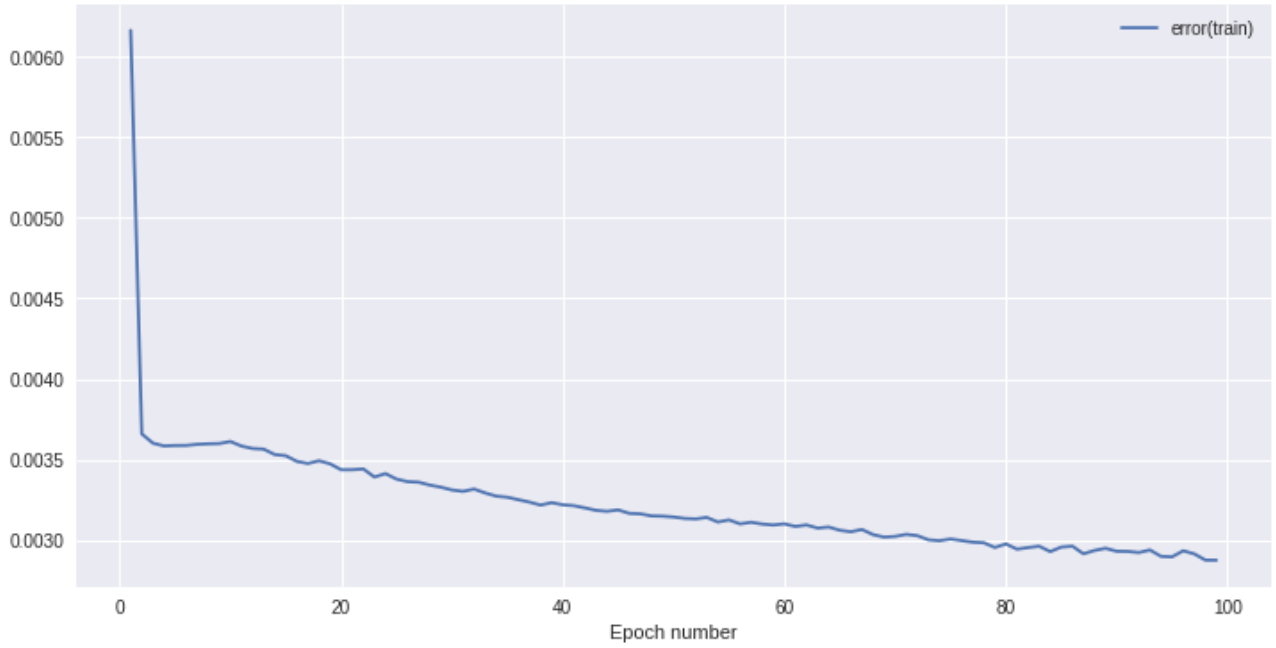


Figure52 Training Huber Loss - Sequence to Sequence Price History and Static Mobile Attributes Model

From the figure 52 we observe that the training loss has not converged but the learning rate is too slow.
Adding the mobile static attributes added a significant amount of complexity to the model without offering any added value to our price prediction task.

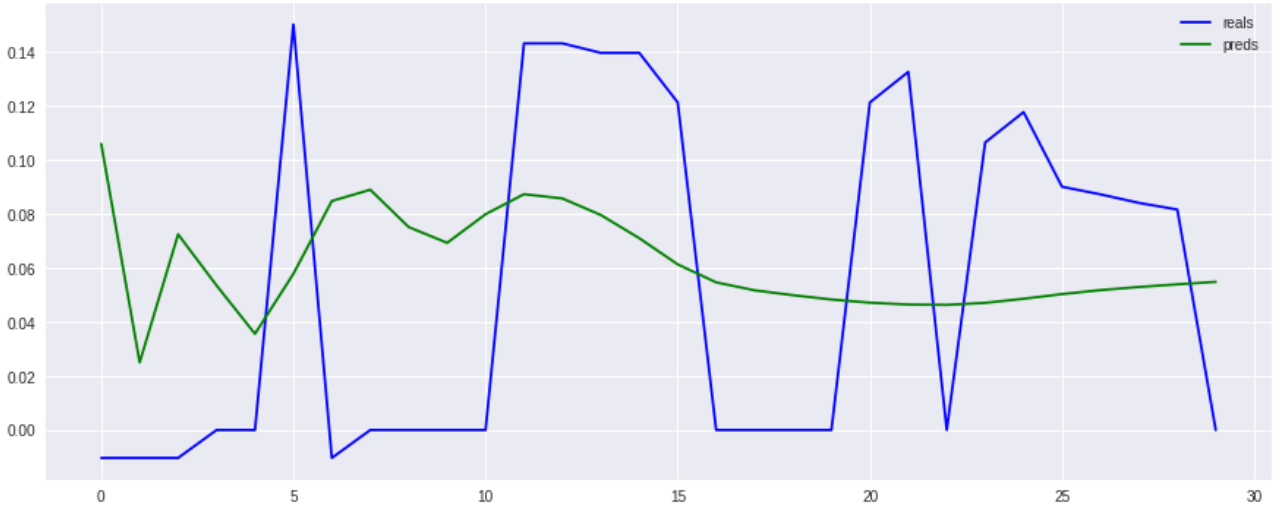


Figure53 Sample Prediction - Sequence to Sequence Price History and Static Mobile Attributes Model

By observing a sample prediction in figure 53 we reach to similar conclusions as to the previous experiment. The price prediction task is a difficult problem and predicting the sudden changes that we observe in the figure might be related to information that is missing from the inputs, which is the past.

10.6 Sequence to Sequence Price History and Static Mobile Attributes Model with Date Information

One more information that we possess as domain knowledge about our price histories is their position in time. More specifically we have at our disposal the exact date of every price of our price histories on a daily frequency. Feeding this information to our current model would enable it, for example to learn any daily or weekly or monthly periodicities of any patterns related to the price prediction task. We are taking advantage of all available information that could be extracted from a date which are Month, Day of the Month, Day of the Week, Year, Day of the Year and Week of the Year. As an example, if we convert the date 21st March 2017 to our vector of six values we get:

- Month: 3
- Day of Month: 21
- Day of the Week: 2
- Year: 2017
- Day of the Year: 80
- Week of the Year: 12

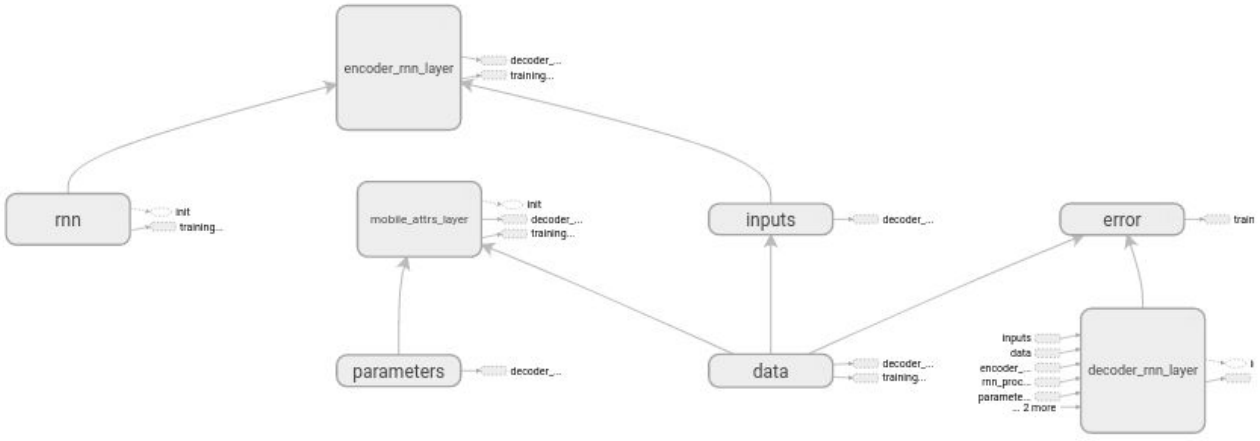


Figure54 Graph of Sequence to Sequence Price History and Static Mobile Attributes Model with Date Information

Note in figure 54 that the graph of the current model is exactly the same as the graph of the previous model and the only difference is of the dimensionality of the information that is fed inside the encoder and the decoder. We are passing the date information as a six-dimensional vector to every input both the input of the encoder and the corresponding dates of the decoder. No other change in the architecture is necessary.

10.6.1 Hyperparameters

Hyperparameter	Value
Batch size	300
RNN cell	GRU cell
Optimizer	Adam [21]
Alpha (Decoder Input Control Percentage)	100% for epoch 1 0% for all other epochs

10.6.2 Cross Validation

We are going to use Bayesian Optimization to optimize important hyperparameters altogether by using a multidimensional Gaussian Process for the space that all of these hyperparameters lie on.

10.6.2.1 Hyper Parameters for Optimization

Hyperparameter	Minimum Value	Maximum Value	Prior Distribution
Readout Input Dropout Keep Probability	0.5	1.0	uniform
Decoder Output Dropout Keep Probability	0.5	1.0	uniform
Number of Units of GRU cell	300	600	uniform
Learning Rate	1e-6	1e-2	log-uniform
Decoder Output Hidden MLP Layer Dimensionality	100	300	uniform
Mobile Static Attributes MLP Layer Dimensionality	100	300	uniform

Due to the results of the bayesian optimization of previous experiments we are setting the maximum available dimensionality for the Decoder Output and the Mobile Static Attributes to a maximum dimension of 300.
In addition we are increasing the minimum available keep probability for all Dropout layers, because we might tempt the optimization of the model to turn off a part of the graph by setting this value too low while on an intuitive level all of the available information should be helpful.

10.6.2.2 Cross Validation Params

Number of splits	3
Number of Random Initializations	3
Number of Bayesian Optimization Calls	17
Epochs	10

The objective function is returning the value of the DTW score on the validation set.

It is interesting to notice in figure 47, figure 51 and figure 55 that the number of calls that are required for convergence are steadily increasing. This is because as the model gets more complex, the more its number of hyperparameters increase and the more difficult is to found an optimal place in the hyper parameter space. In figure 47 we needed eight calls, in figure 51 we needed nine calls and in figure 55 we are needing ten calls.

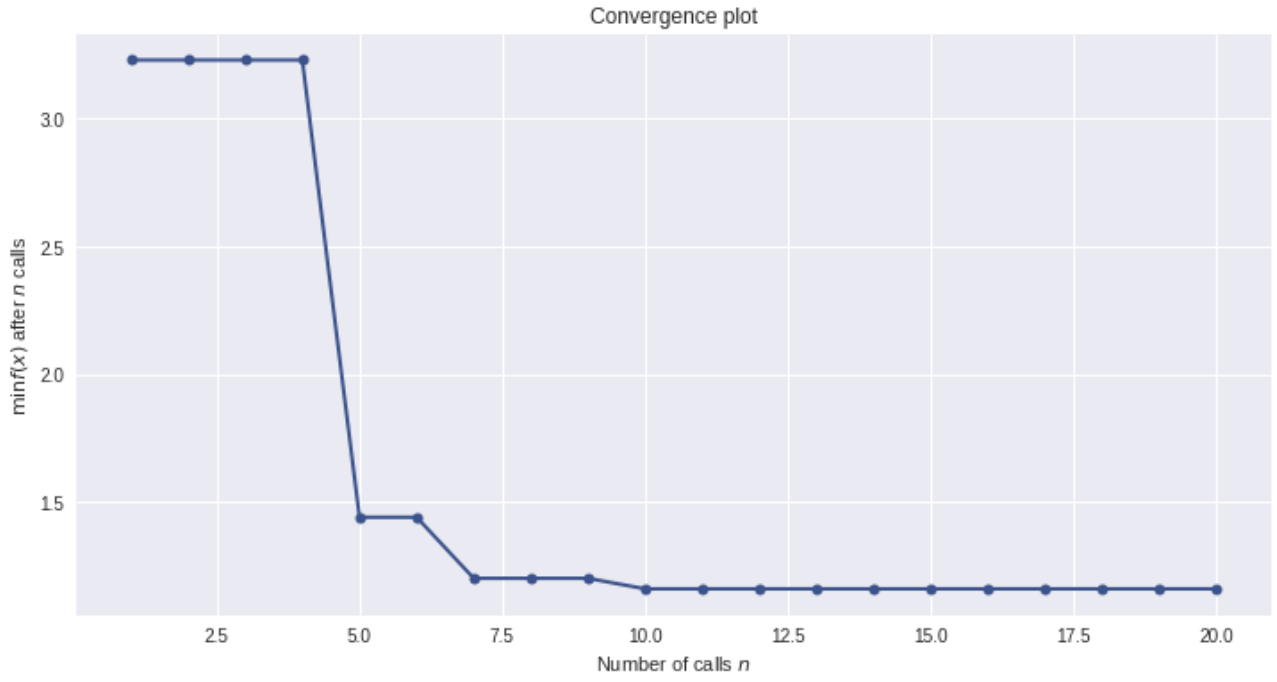


Figure55 Convergence - GP Bayesian Optimization - Sequence to Sequence Price History and Static Mobile Attributes Model with Date Information

10.6.2.3 Optimized Hyperparameters

Hyperparameter	Optimal Value
Readout Input Dropout Keep Probability	1.0 = 100%
Decoder Output Dropout Keep Probability	1.0 = 100%
Number of Units of GRU cell	415
Learning Rate	1e-2
Decoder Output Hidden MLP Layer Dimensionality	168
Mobile Static Attributes MLP Layer Dimensionality	182

The optimization here has resulted in the maximum available values of the learning rate that we provided as boundaries to the Bayesian optimization. This is attributed to the fact that every call of the Bayesian optimization lasted for only 10 epochs meaning that the calls, where the learning rate was small, they were bound to be of failure.

Moreover the optimization resulted to Keep Probability equal to 100% for both Dropout Layers which effectively disables them. This is also expected since the introduction of the dropout effect on a layer increases the exploration of the possible states to eventually result in a better state in the error space but it also requires more epochs of training.

10.6.3 Training

Using the optimal hyperparameters and keeping the rest of the hyperparameters intact, we are training our optimized model for 100 epochs to get the minimum possible training error as seen in figure 56.

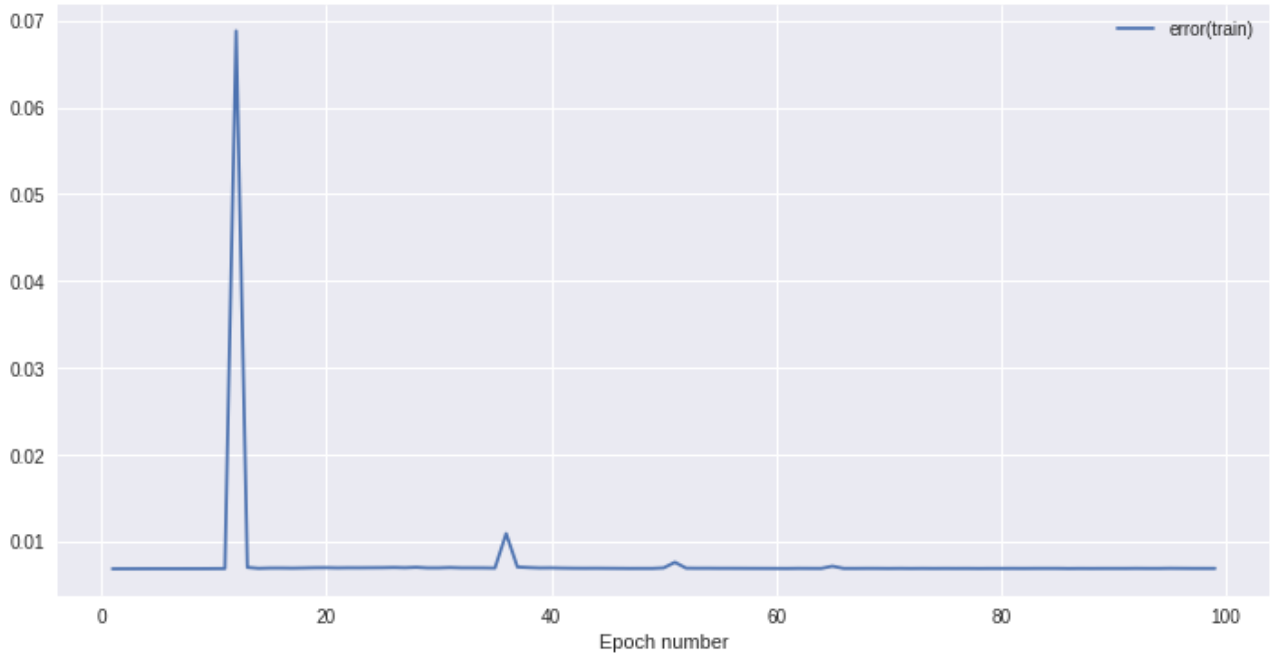


Figure56 Training Huber Loss - Sequence to Sequence Price History and Static Mobile Attributes Model with Date Information

10.6.4 Results

In the results table we are including the model of the previous experiment for comparison reasons.

Metric	**Baseline**	**Sequence to Sequence Model with Batch Normalization, Dropout, Dynamic Decoder Inputs and Normalized Price Histories**	**Sequence to Sequence Price History and Static Mobile Attributes Model**	**Sequence to Sequence Price History and Static Mobile Attributes Model with Date Information**
Training Huber Loss	0.00568349	0.00320502	0.00314844	0.00361336
Average DTW Score	1.13140236	1.15343520	1.17044361	1.21531176

From figure 57 we observe that the training loss plot includes some spikes which are caused by the high learning rate but eventually manages to converge at a training loss which is lower than the baseline.

The average DTW score of this experiment is higher than the baseline and higher than the two other models. Indeed, by plotting several predictions against their targets we get the view that we expect, and a representative plot of them is seen in figure 57. All of the predictions are straight lines which represent the average of the prediction and therefore this qualitative analysis agrees with the metrics and concludes that this model has performed worse than the previous ones.

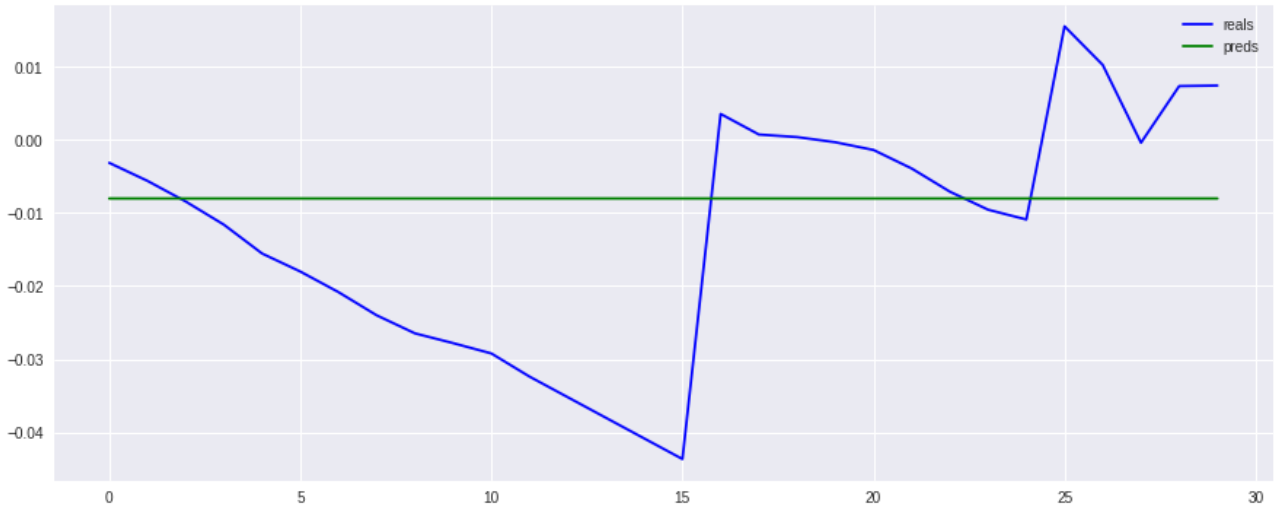


Figure57 Sample Prediction - Sequence to Sequence Price History and Static Mobile Attributes Model

10.7 Conclusions

From the experiments and models developed in this chapter we explored the behavior of advanced neural network models that took advantage of the price histories, the static mobile attributes and the date information implied for each price in a price history.
The more complex the model became the more computationally expensive became and the more difficult it was to be optimized.
Comparing the results from the training of the advanced models in this chapter with the more simple sequence to sequence model that was tested against a subset of our price history there is a strong indication that we should try and simplify the problem by breaking the dataset into more homogenous pieces. In other words, our current alternative is to avoid building a single model to learn all of the price histories for all of the mobile phone products but we rather split the price histories by clustering.
Moreover, our price histories are actually time series of prices and should be preprocessed before being used as input to a neural network model.
The models we have explored have failed to produce an adequately sufficient model using the normalized raw data of the price histories.
Note that the problem of price prediction is very difficult and there might be little useful information from the past that indicates the progression of the price in the future. However more exploration is necessary until we could make this statement conclusive.

Chapter 11
Price History Clustering

We have seen that the price prediction is very difficult and thus it might be difficult to create a single model that can be trained to become a predictor for all of the mobile phone products of our dataset. One way to improve our model is to give it an easier task and we will achieve that by segmenting our dataset, thus giving the model a smaller subset of instances to be trained on. But we cannot do this by simply splitting the dataset into smaller pieces, we also need to get the new groups to share some characteristics. This is where the method of unsupervised clustering will be beneficial.

It is important of clustering the price histories because we care of instances that behave similarly in the market. We hypothesize that by clustering mobile phone products using their static attributes will not be helpful for the price prediction task.

11.1 K-Means Clustering with DTW Distance

Our first method of clustering is using K-means clustering algorithm on price histories.
We are using the Dynamic Time Warping (DTW) Distance as a distance metric for comparing the similarities between two price histories.

11.1.1 Normalization

Note that before we apply our clustering algorithm we are applying the Max-Min Scaler to all of the price histories so that they are normalized. Max-Min Scale standardization subtracts the minimum value and divides by the range. The formula is given below:

$$Minmax\ scaled\ X = \frac{X - X_{min}}{X_{max} - X_{min}}$$

This is important to prevent the DTW distance metric to judge the similarity of two price histories based on their bias or scale.

11.1.2 Algorithm

The K-means clustering algorithm with DTW distance metric is summarized in the following steps:

1. Choose Centroids by randomly sampling from the time series
2. For all of our price histories calculate their DTW distance from all of the centroids
3. Assign each price history to the nearest centroid
4. Recalculate the centroids of our Clusters by using the average values of all of the price histories that are assigned to the same cluster

a. Note that we cannot mathematically reverse the DTW distance in order to find a centroid that would have an equal distance between all of the price histories in the same cluster

5. Comparing the previous with the newly generated centroids check if the maximum distance of all pairs is smaller than a tolerance limit and stop accordingly
6. Repeat the process from step 2 for a maximum number of iterations.

Note that K-Means algorithm usually is deployed along with the Euclidean distance because it guarantees convergence. However using DTW distance convergence is not guaranteed and we are using a maximum number of iterations to stop the algorithm even if no convergence has occurred.

One important limitation of our algorithm is that it only works with time series of a specified length. The DTW distance can be calculated for time series of variable length but we have currently no way to generate a centroid by taking the average values of price histories of variable length.

11.1.3 Evaluation

We need an objective way to measure the quality of our clustering and this is why we will calculate the Silhouette Score [22] after each clustering. The distances will be the precomputed DTW distances of all the possible pairs of time series. Note that Silhouette Score can be used with any distance metric.
The Silhouette score is implemented in scikit-learn python package [see Appendix 1] and it works as follows:
We first find the average distance between one price history and all the rest of the price histories that belong to the same cluster and name it as Average Distance of Own Cluster.
We then find the average distance between this price history and all price histories that belong to the nearest cluster and name it as Average Distance of Nearest Cluster.
The silhouette coefficient is defined as the difference between the latter and the former average distance divided by the greater of the two (max(A,B)):

$$Silhouette\ Coefficient = \frac{Average\ Distance\ of\ Nearest\ Cluster - Average\ Distance\ of\ Own\ Cluster}{max\ (Average\ Distance\ of\ Nearest\ Cluster,\ Average\ Distance\ of\ Own\ Cluster)}$$

We evaluate the clustering coefficient of each price history and from this we can obtain the overall average clustering coefficient, which is the Silhouette score.
Note that the Silhouette score will take values within the range [-1, 1] where -1 is the worst possible clustering and 1 means perfect separation of the clusters.

Intuitively we are trying to measure the space between clusters. If cluster cohesion is good, namely the Average Distance of Own Cluster is small, and cluster separation is also good, namely the Average Distance of Nearest Cluster is large, then the numerator of the Silhouette coefficient will be large and the score will be bigger and thus better.

11.1.4 Experiments

Due to our limitation to work on price histories of fixed length we are limiting our clustering experiments to the price histories that are of the same length of 210 days and are the most frequent on our dataset. By including the outliers we have a total of 112 time-series to work on.

11.1.4.1 Hyperparameters

DTW Distance Tolerance Threshold: 0.1
Maximum number of iterations: 100

In Appendix 2 we see the final centroids after running our clustering algorithm for this range of k values: {2, 3, 4, 5, 6, 7}

11.1.5 Results

Number of Clusters - K factor in K-means clustering	Silhouette Score	Number of Clusters - K factor in K-means clustering	Silhouette Score
2	0.364836794412	5	0.246270968418
3	0.346282905256	6	0.241415247676
4	0.308568252108	7	0.212728285197

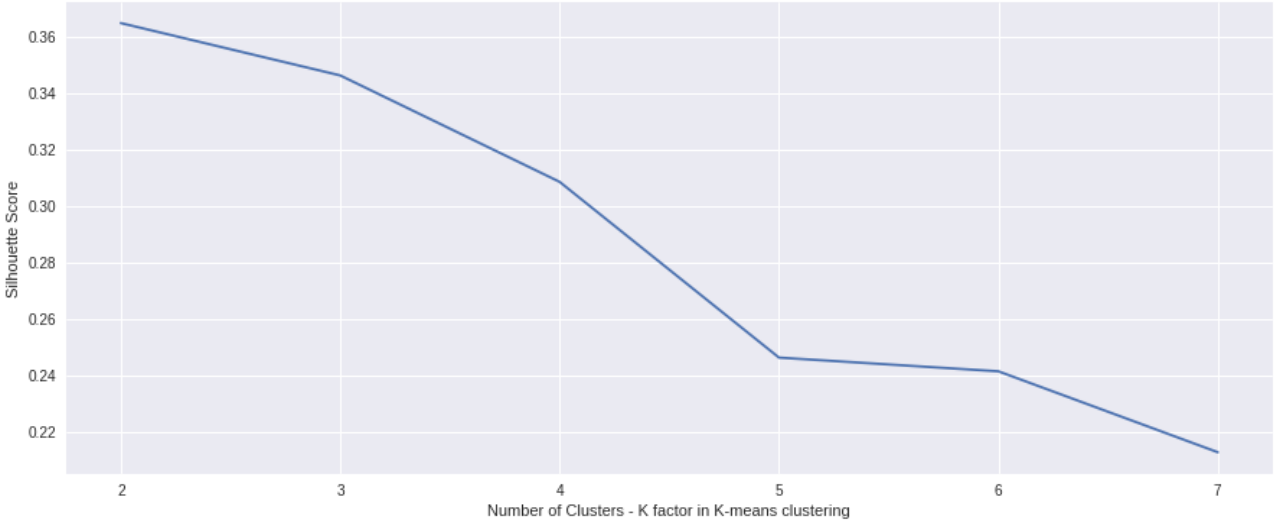


Figure58 Progression of Silhouette Score as the number of clusters increase

11.1.6 Conclusions

We conclude that our clustering algorithm is not suitable for this price history dataset as the price histories seem to follow complex patterns which leads to clusters that overlap each other.

The only clustering that we can attribute meaning by looking at it, is when clustering for k=2 clusters. We see that the resulting centroids are concentrated around price histories which in general either have an increasing or a decreasing tendency.

However we see that the silhouette score is monotonically decreasing when the number of clusters are being increased and thus we cannot trust our current algorithm of producing useful clusters.

11.2 Dimensionality Reduction of Time Series with Recurrent Neural Network Autoencoder

Neural Network Autoencoders are using a stack of MLP layers of decreasing dimensionality mirrored by the same stack with increasing dimensionality to train a model to represent the inputs as closely as possible with a lower dimension vector [23].
We are expanding this idea by introducing recurrent neural networks in order to be able and reduce a time series, such as our price histories which are of variable length, to a fixed vector of low dimensionality.

11.2.1 Architecture

The overview of our Sequence-to-Sequence Autoencoder architecture is given firstly by an encoder RNN whose final state is reduced to a predefined dimensionality, depending on the dimensionality reduction that we want to accomplish. The reduced final state of the encoder is then used as the initial state of the decoder. Both the encoder and the decoder are fed with the date information that corresponds to each day of the price history.
The encoder is also fed with the actual price history while the output of the decoder aims to regenerate this price history as closely as possible.
So the output is generated by the decoder whose only information about the price history being fed to the input, is given by this, reduced in dimensionality, state of the encoder.
Due to the above fact and due to the symmetric of the architecture we hypothesize that this reduced state should be representative of the price history.

More particularly as seen in figure 59 our RNN Autoencoder consists of multiple layers. There are two RNN layers to be used for the encoder. The outputs of the first RNN layer are used a inputs to the second, hidden, RNN layer of the encoder. The outputs of the second RNN layer of the encoder get discarded and we only keep the final state of this layer as the final state of the encoder.

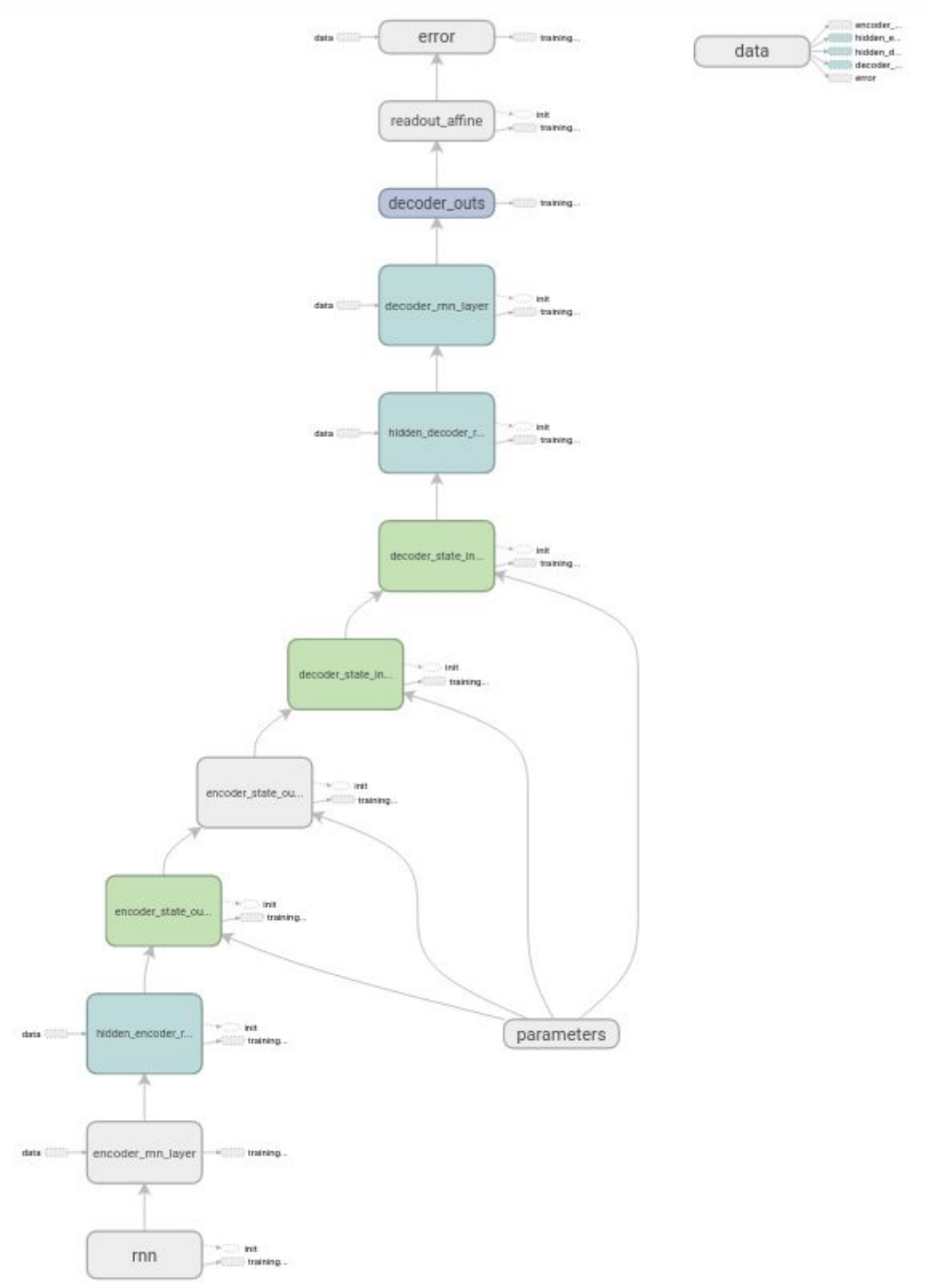


Figure59 Graph of Recurrent Neural Network Autoencoder

Next, the final state of the encoder is being reduced using a regular autoencoder architecture[23], of two layers, whose output is being used as the initial state of the decoder. Similarly, the decoder has two RNN layers as the encoder and its inputs are only the domain knowledge that we have for the

time series which is the date of each day. Finally, there is an affine layer to convert the dimensionality of the output of the decoder to one in order to get the final output sequence.
The main benefit of this architecture in comparison to the k-means clustering is that we are allowed to use price histories of variable length and represent them in as a low dimensional vector of fixed dimensionality without any restrictions. Before we were forced to either trim long price histories and discard short ones or use a subset of them. Therefore now we can train the autoencoder on the entirety of our price histories.

11.2.3 Training

11.2.3.1 Hyperparameters

Hyperparameter	**Value**
Batch Size	13
RNN cell	GRU
RNN number of units	250
Dimensionality of Hidden MLP Layer	101
Learning Rate	1e-4
Optimizer	Adam

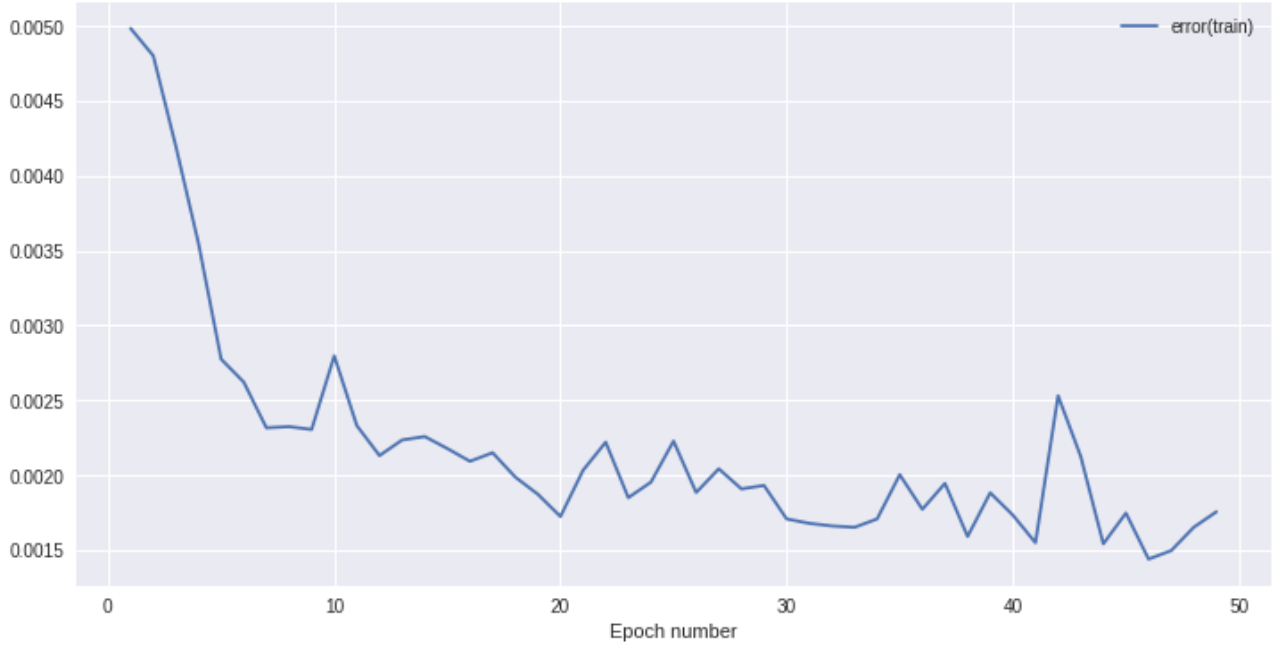


Figure60 Training Huber Loss - RNN Autoencoder

The training error is being given by the Huber loss between the output and the targets, which here are the same as the input. We are training for 50 epochs and the result of this training is rendered in figure 60. We observe that there is a lot of oscillation in the training error but there has been convergence after 50 epochs.

11.2.4 Results

Average Huber Loss	0.00978253
Average DTW Score	10.98620926

It is important that the output of the autoencoder represents the inputs as closely as possible and therefore the most important metric here is the average DTW Score and we also need to do a qualitative check on the actual predictions and see how much they approximate the inputs.

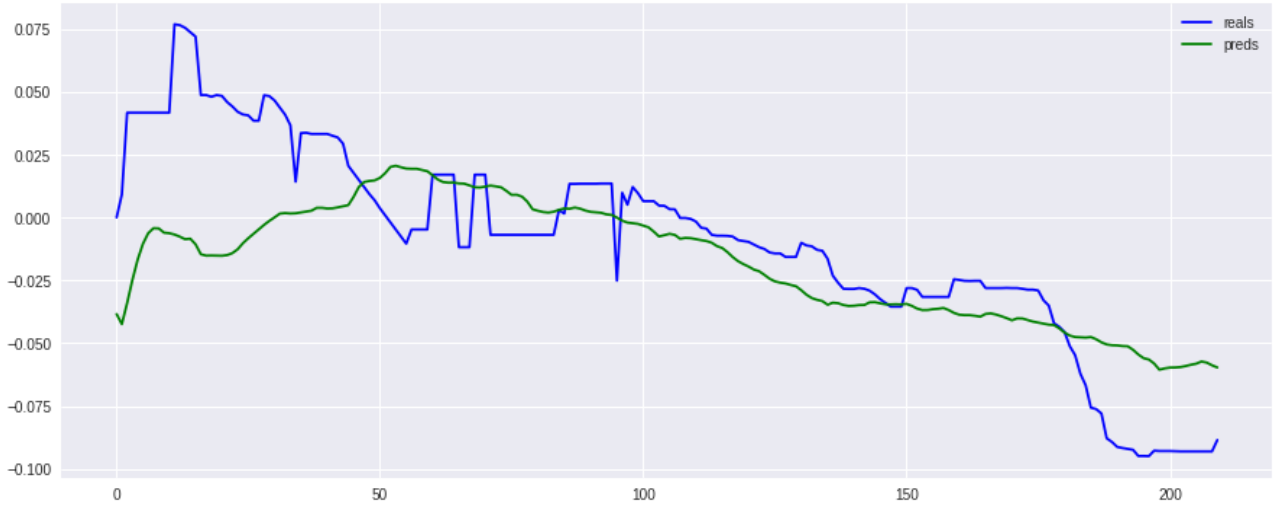


Figure61 Sample Output - RNN Autoencoder

As we see in figure 61 the output of our RNN autoencoder is a low-resolution, low-fidelity approximation of the input but it can still be considered to be representative of the price history.

11.2.5 2D Dimensionality Reduction

Before executing any clustering programmatically it is important to plot the fixed vector of the intermediate layer of our RNN Autoencoder to a dimensionality of two. This will allow us to represent each one of our price histories on a 2D plane and thus be able to plot them and check for any patterns visually.

After plotting each price history on a two-dimensional plane we get figure 62 where we cannot observe, by eye, any clusters to form but it is important to note that the price histories lie on a one dimensional manifold. In other words, we can split this manifold into pieces and get separate clusters of the dataset that should share similar characteristics.

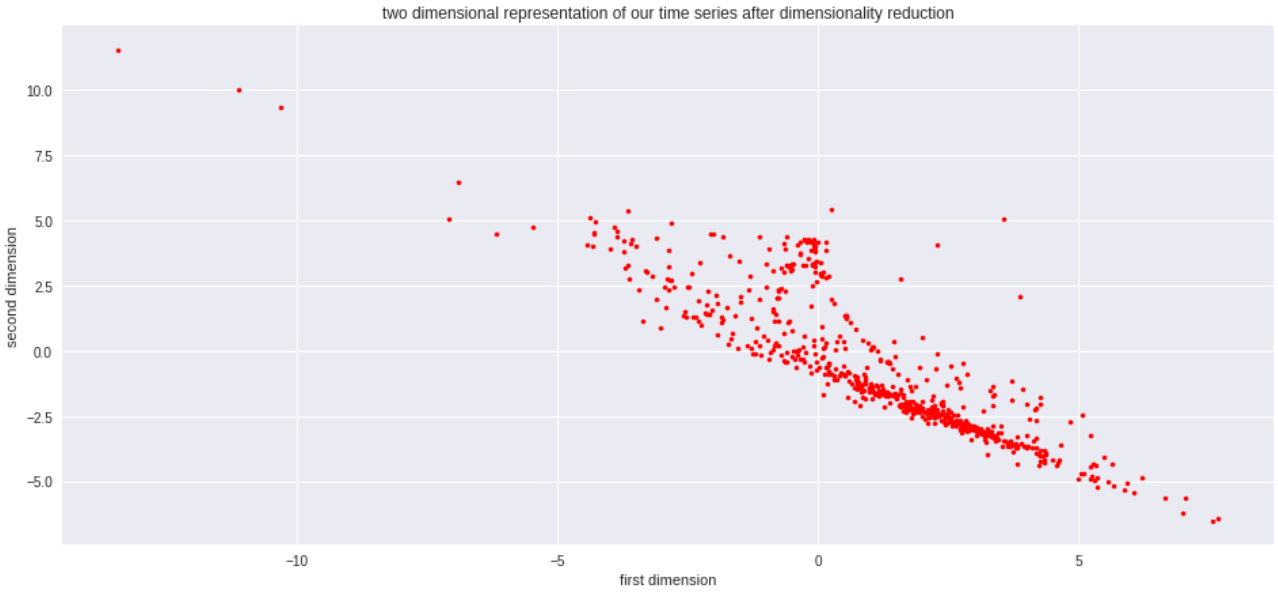


Figure62 Two Dimensional Representation of Price Histories - RNN Autoencoder

11.2.6 Clustering

Using the above representation to apply the K-means clustering algorithm will be expecting to find better clusters and significantly increased speed performance of clustering since we are using Euclidean distance which is computationally less expensive than the DTW distance.

We are clustering by setting the number of clusters in the range [2, 19] and for each execution, we are calculating the silhouette score. The results are shown in figure 63.

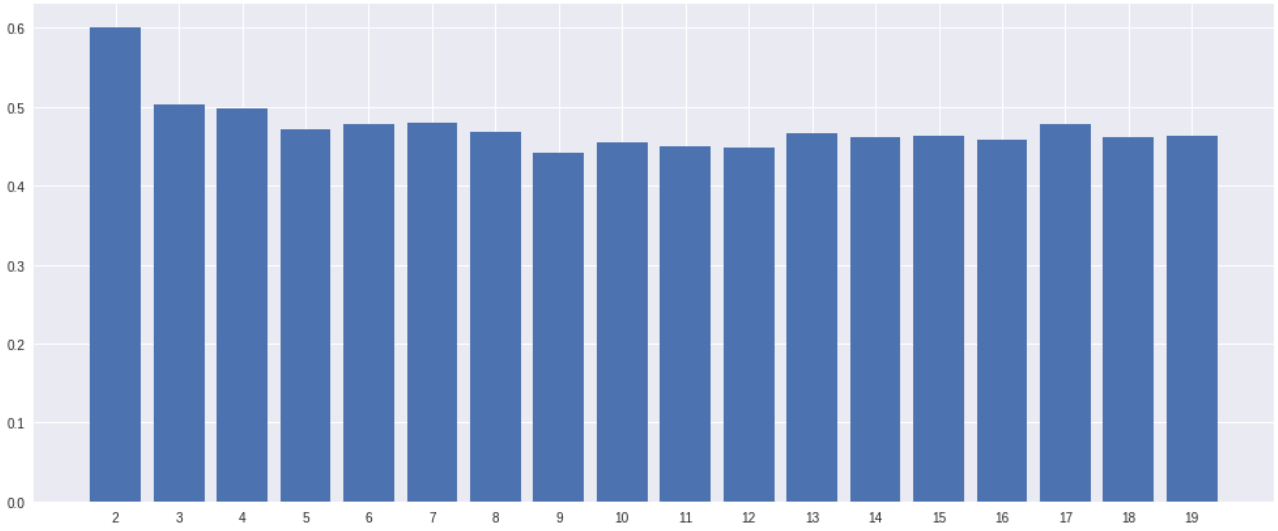


Figure63 Silhouette Score for clustering of 2D representation of price histories - RNN autoencoder

One important comparison with the previous method of clustering is that the new clusters, for all cases, have a much higher Silhouette score than the best Silhouette score we calculated in the previous method of clustering.

As an example, we are plotting the kernel density estimate of each one of the clusters, after applying k-means clustering with k=7, in figure 64.

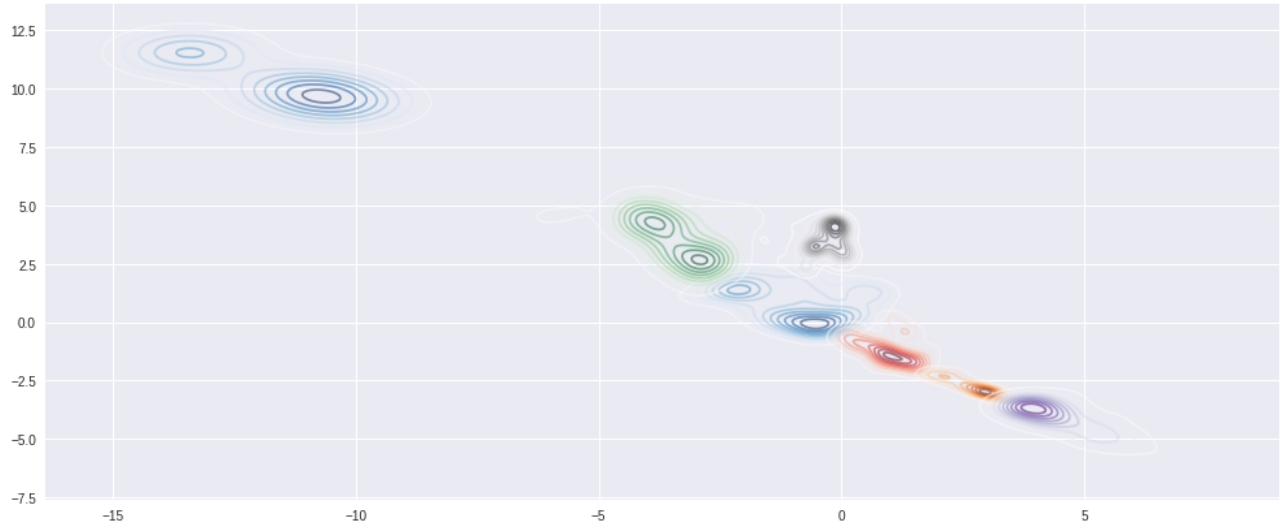


Figure64 Kernel Density Estimate of each cluster - K-means Clustering - k=7

The optimal number of clusters is depended on the particular task. For our price prediction task, we could build different models for each cluster and evaluate the performance for the instances that correspond to each model under a common evaluation metric in order to be able and express which number of clusters is best for the price prediction task.

Chapter 12
Price History Analysis and Stationarity

As we have seen in previous chapters our models developed so far have not managed to give a prediction 30 days ahead in the future with a high fidelity or being significantly better than our baseline.
We hypothesize that this is because we have executed only a very basic preprocessing of our original raw dataset which effectively transfers the responsibility of learning useful features completely to the artificial neural network. One way to mediate this phenomenon is to explore further the statistical characteristics of our price histories and apply transformations that will help our models learn more easily.

12.1 Detrended Fluctuation Analysis

Detrended Fluctuation Analysis is used in time-series analysis as a method to determine the statistical self-similarity of a signal. It can also be defined as a method for scaling the long-term auto-correlation of non-stationary signals. The scaling exponent, known as the Hurst exponent gives an idea of the correlation [24].

Here we are applying Detrended Fluctuation Analysis on all of our price histories as provided by the *dfa* function of the *nolds* python module [see Appendix 1]. This function returns the estimated alpha for the Hurst parameter which determines if the price history is stationary or not. If alpha is smaller than one then the price history is stationary, while if alpha is larger than one the price history is nonstationary.

In figure 65 we see the distribution of the alpha parameter after detrended fluctuation analysis of our price histories. Indeed, we verify our original expectation that most price histories are nonstationary.

12.2 Correlation of Raw Price Histories

So far we have considered that a price history of one mobile phone product does not influence, does not carry information, for the price histories of other mobile phone products. By taking the Pearson correlation of one price history against the rest of the price histories and repeating this process for all price histories we generate the heatmap that we see in figure 66.
Note that price histories are of variable length and also starting from different positions on the time axis. When calculating the correlation of two price histories we are taking into account only the

overlap of these two price histories in time. In other words, only the days which are common among two price histories are taken into account when calculating the correlation. There are two expected effects of this. Firstly, price histories that do not overlap at all have zero correlation. Secondly, price histories which have short length might seem to have a high correlation with most other price histories but this is just an illusion as the high correlation is only valid for this short period of overlap between them.

The heatmap, in figure 66, indicates strong correlations between the price histories but this correlation is impacted by within-time-series dependence. This is called spurious correlation [25]. In Spurious Correlation the price histories appear correlated but the correlation itself is not meaningful. The whole notion of considering association between price histories without taking into account the within-time-series dependence is wrong because there might be no actual cross-price-histories relation and the apparent association is a mere illusion.

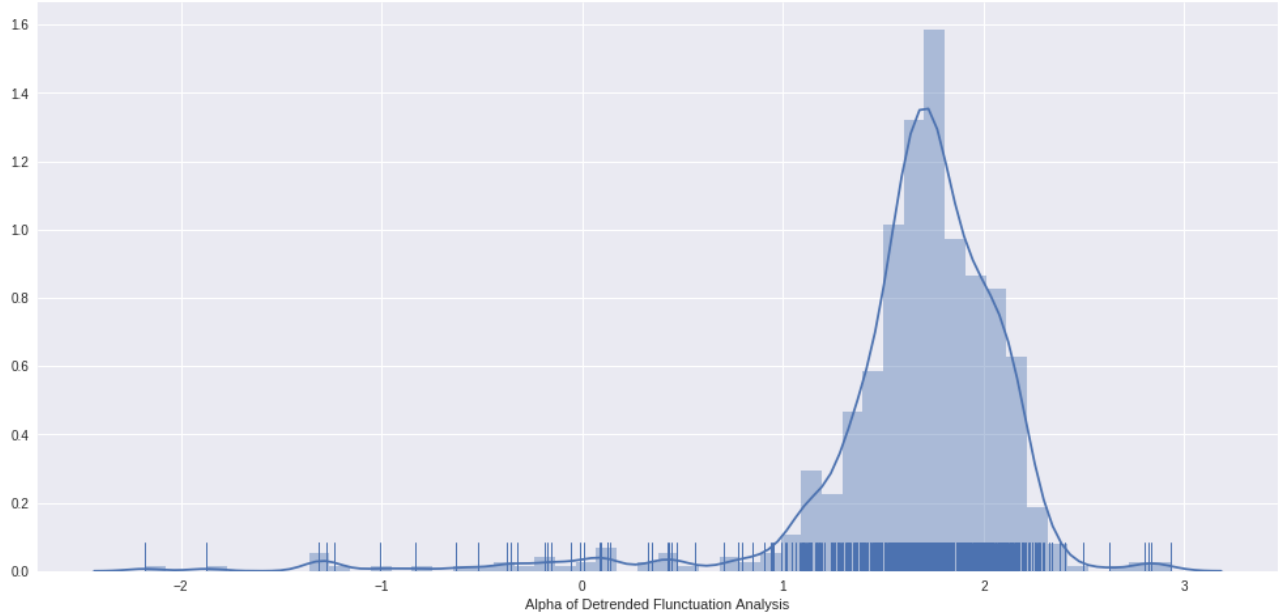


Figure65 Distribution of Alpha of Hurst Parameter after Detrended Fluctuation Analysis of our Price Histories

We have to deal properly with the issue that makes the price histories auto-dependent and auto-correlated before applying Pearson Correlation Properly.

12.3 Making Price Histories Stationary

Our metric for testing if our price histories are stationary is the Augmented Dickey Fuller Test, which is implemented in Statsmodels Python package [see Appendix 1].
The threshold for the p-value of the Dickey Fuller Test is 5% = 0.05.
After applying the Dickey Fuller Test on all of our raw price histories we get a distribution for the p-value as seen in figure 67. As expected we observe that most of our price histories are non-stationary and above the threshold of the p-value of 0.05.

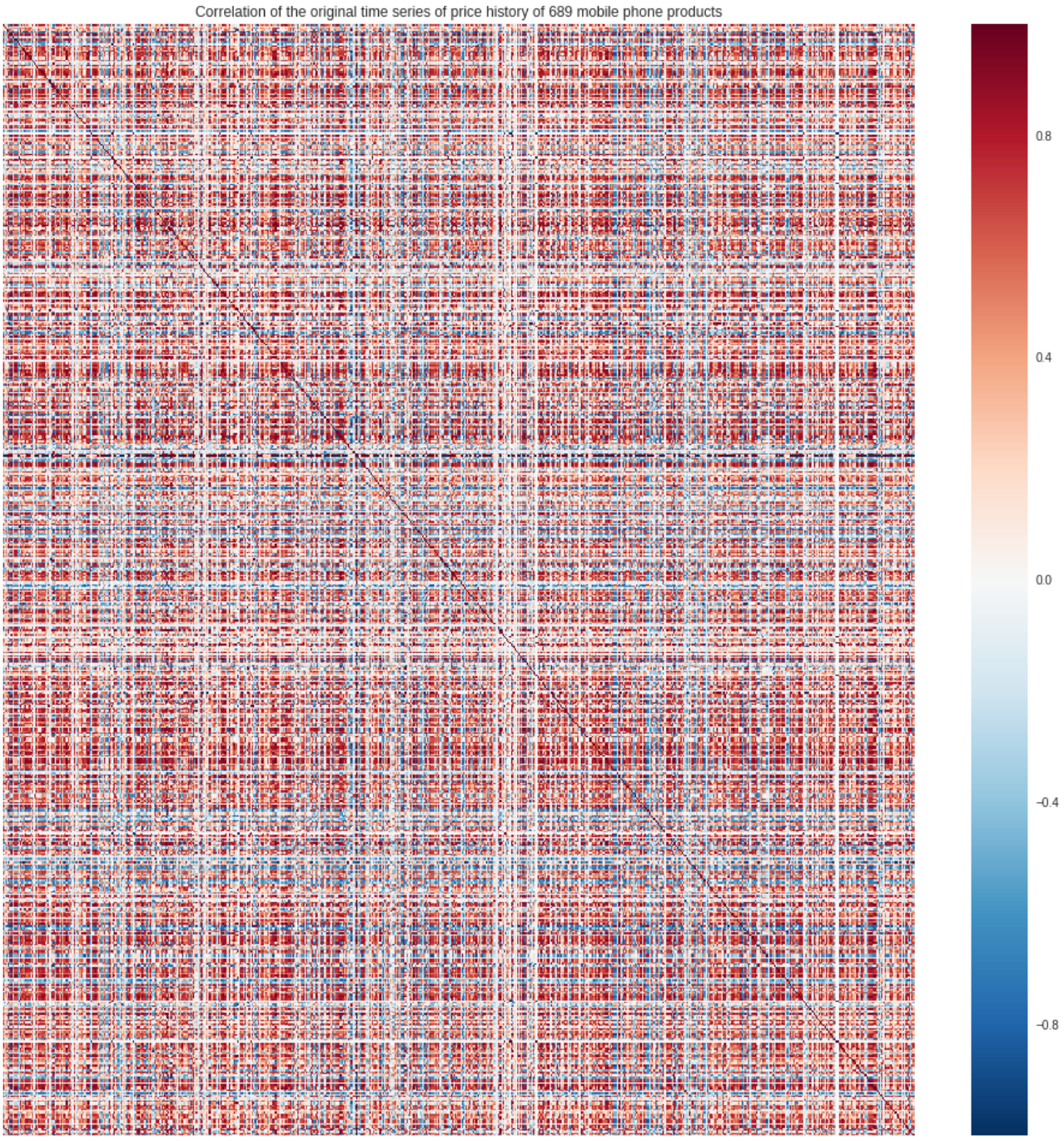


Figure66 Heatmap - Pearson Correlation - Raw Price Histories

12.3.1 Differencing Price Histories

There are several transformations to explore that would make our price histories stationery, such as subtracting the rolling mean, but we need to choose a transformation that can be reversed. We need a transformation that would enable us to reconstruct our original price histories. One commonly used such transformation is differencing. A sample of this transformation can be seen in figure 68. We notice that by differentiating the average value of the sequence becomes near zero and the statistical properties of the sequence seem to be preserved along the sequence.

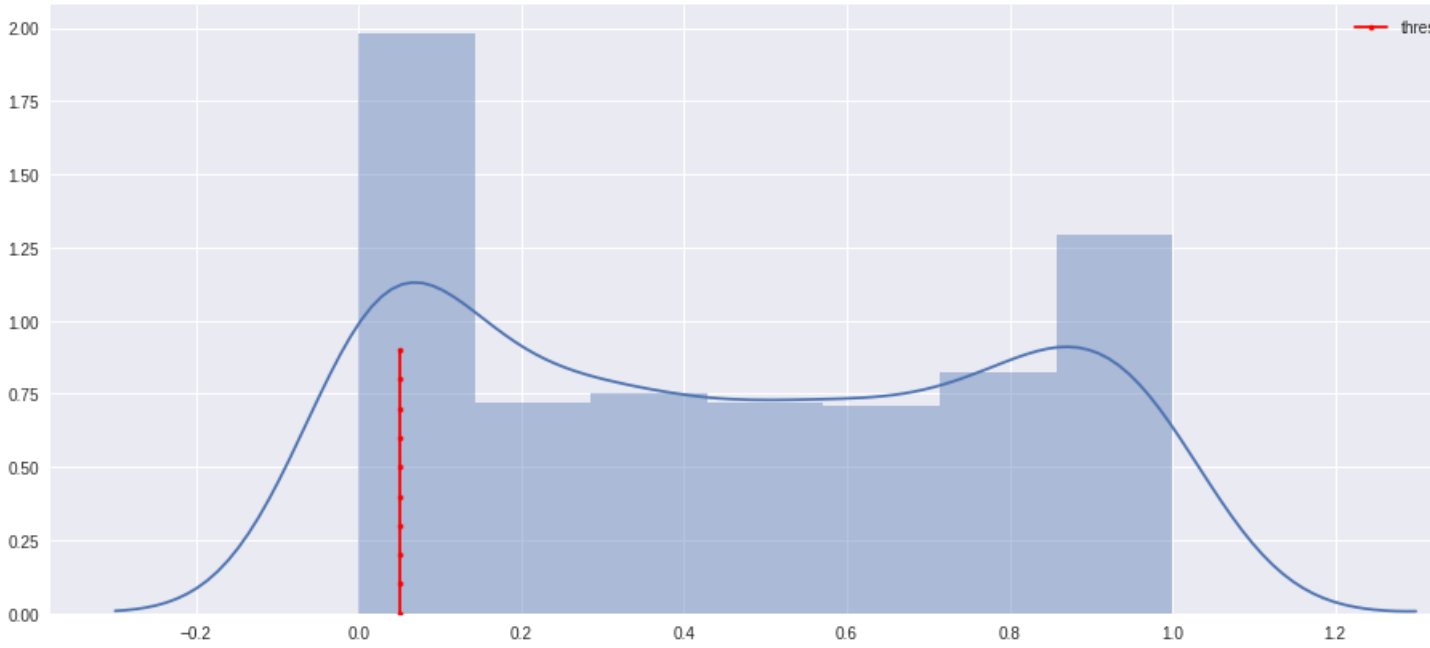


Figure67 Distribution p-value of Augmented Dickey Fuller Test - Raw Price Histories

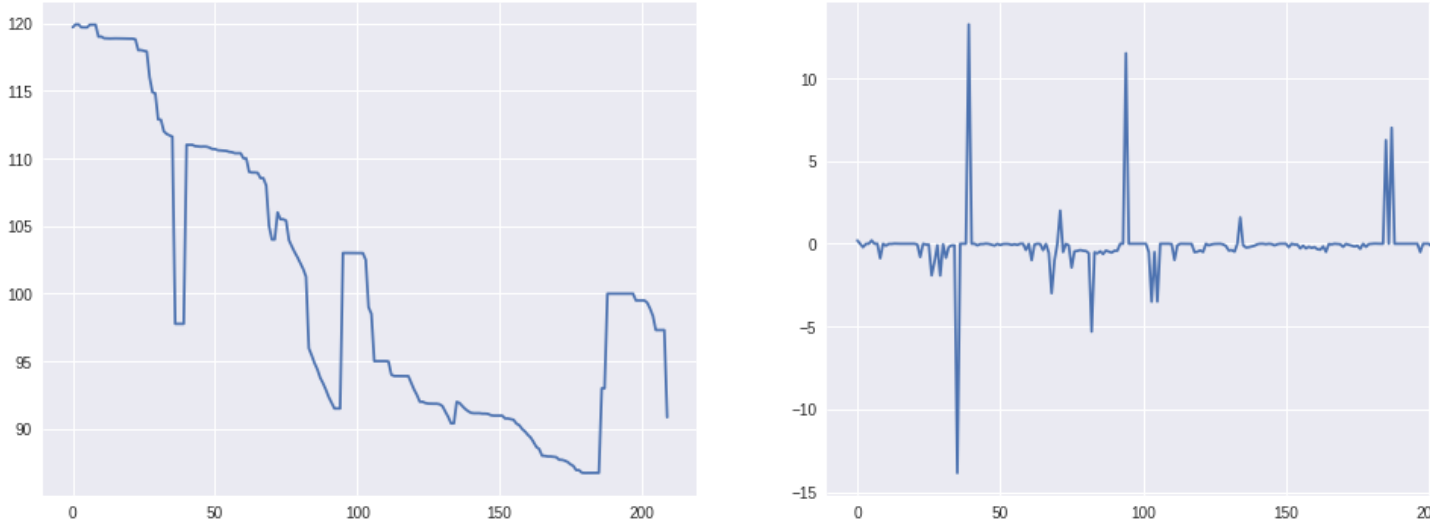


Figure68 Sample of Differentiation of a Price History

After differencing all of our price histories we repeat the Augmented Dickey Fuller Test for all of the differentiated price histories and the distribution of the p-values of the test are shown in figure 69.

We note that the vast majority of price histories after differentiation have p-values, of the Augmented Dickey Fuller Test, which are below the threshold and thus can be considered stationary.

This transformation can be used as a preprocessing step of our dataset in order to make the learning process more easy for our machine learning models.

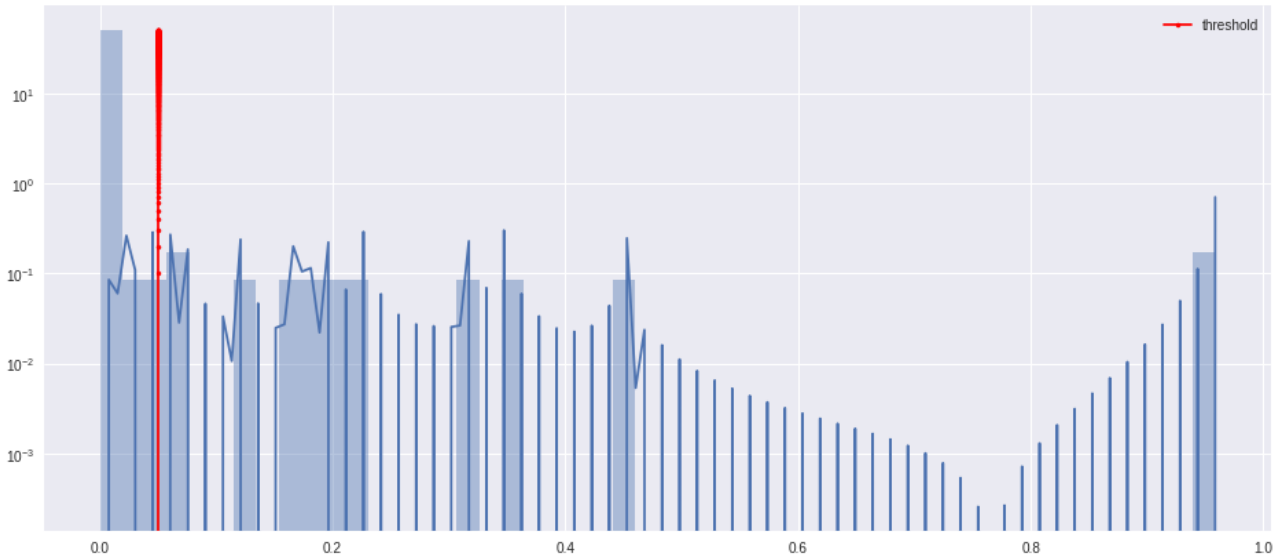


Figure69 Distribution p-value of Augmented Dickey Fuller Test - Price Histories after differencing - Log-scale of y-axis

12.3.2 Correlation of Differentiated Price Histories

By differentiating we have transformed our price histories to stationary time series, thus removing the within-time-series dependencies. We now calculate again the Pearson correlation coefficient for our differentiated price histories and the results are shown in figure 71.

Note that after removing the within-time-series dependencies most of the correlations are near zero producing a white-coloured heatmap. This can be verified by plotting the distribution of all the correlations, as it is seen in figure 70. We conclude that there are no significant correlations among our price histories without introducing any lagging to the calculation.

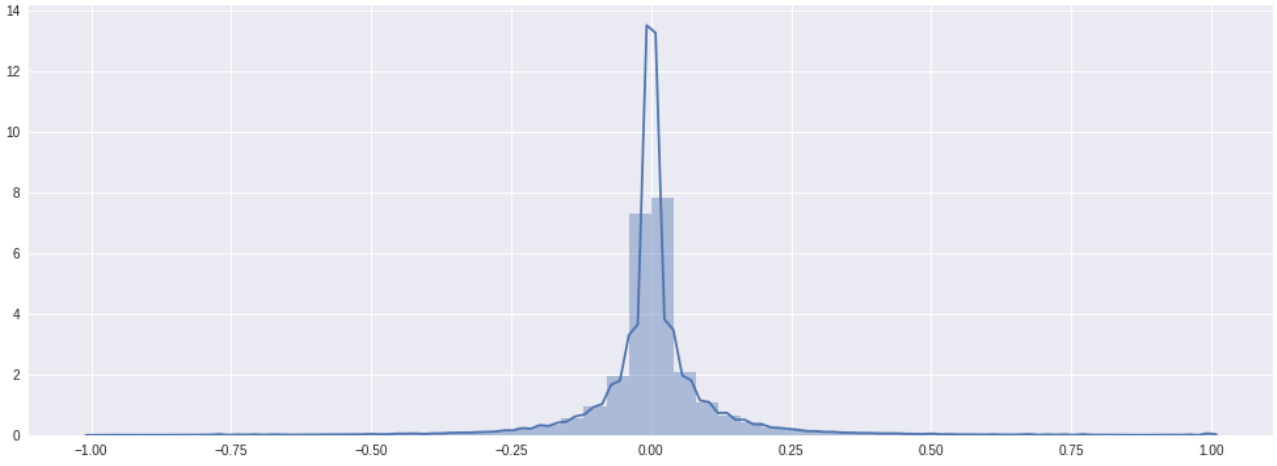


Figure70 Distribution of Correlations of Price Histories after Differentiation

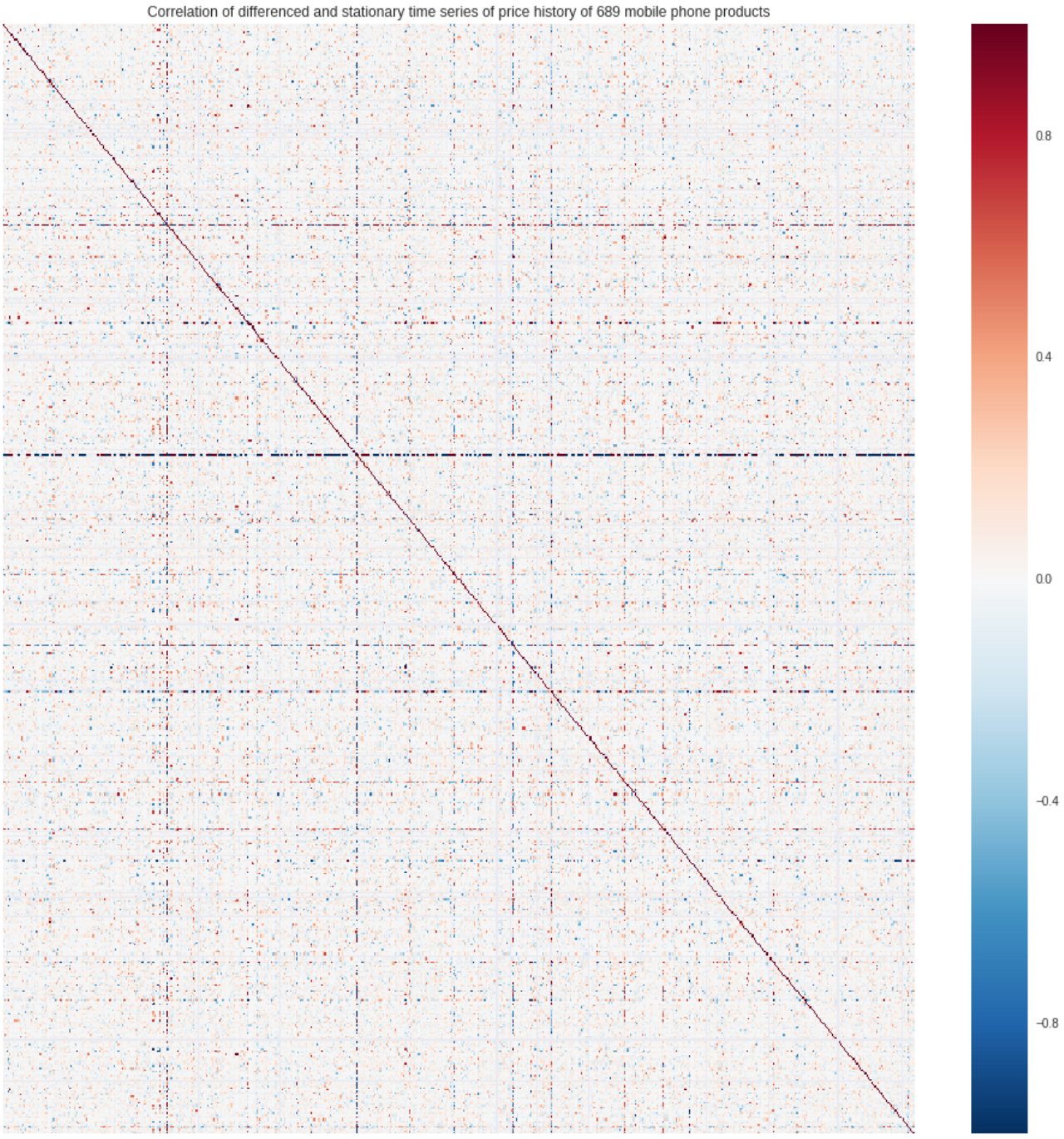


Figure71 Heatmap - Pearson Correlation - Differentiated Price Histories

12.3.3 Cross Correlations of Differentiated Price Histories

We can hypothesize that the price changes of a mobile phone product can affect the changes of another mobile phone product in the future. This means that we should use a lag when calculating the correlation of price histories. This lag can be as big as we like depending on our domain knowledge. This is called cross-correlation and will enable us to see if there are correlations among our price histories that are more subtle and shifted in time.

Due to the dynamic nature of the price histories of our mobile phone products we are hypothesizing that up to a full week of 7 days can be expected that a price history can affect another price history. Our implementation takes all the possible day-lags in the range [-7, 7]. Note that we are including also negative lags because each time that we calculate a correlation we do not know if the first price history is correlated with the second in the future or it is vice versa.

After calculating the correlation for all of the lags we are keeping the original value of the correlation with the maximum absolute value. This means that a different lag will have been taken into account for calculating the final correlation. However, we do not care of the particular lag here as we want to see if there are correlations at all, and in what extent.

By plotting the heatmap of the cross correlations of our price histories, as seen in figure 73, we immediately observe that the heatmap has a salt-pepper look. In other words and it does not have the white color of the correlation heatmap of figure 71. It is interesting to observe that the distribution of the cross correlations, as seen in figure 72, is not surrounded around zero but indeed there is a significant amount of price histories which are weakly correlated and a few price histories that are strongly correlated. Note that due to the symmetry of the correlations we have a mirrored effect on the distribution plot.

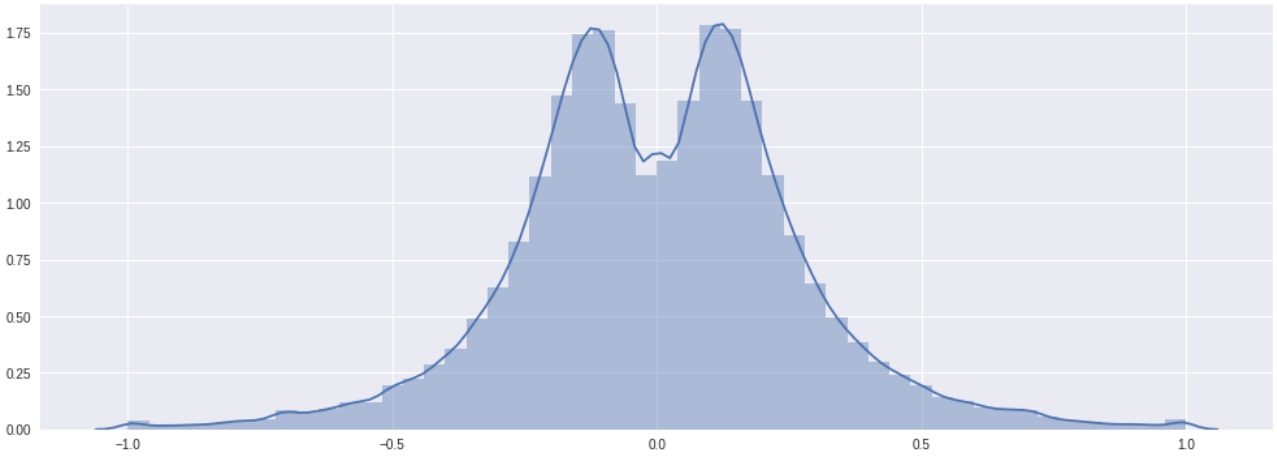


Figure72 Distribution of Cross Correlations of Price Histories after Differentiation

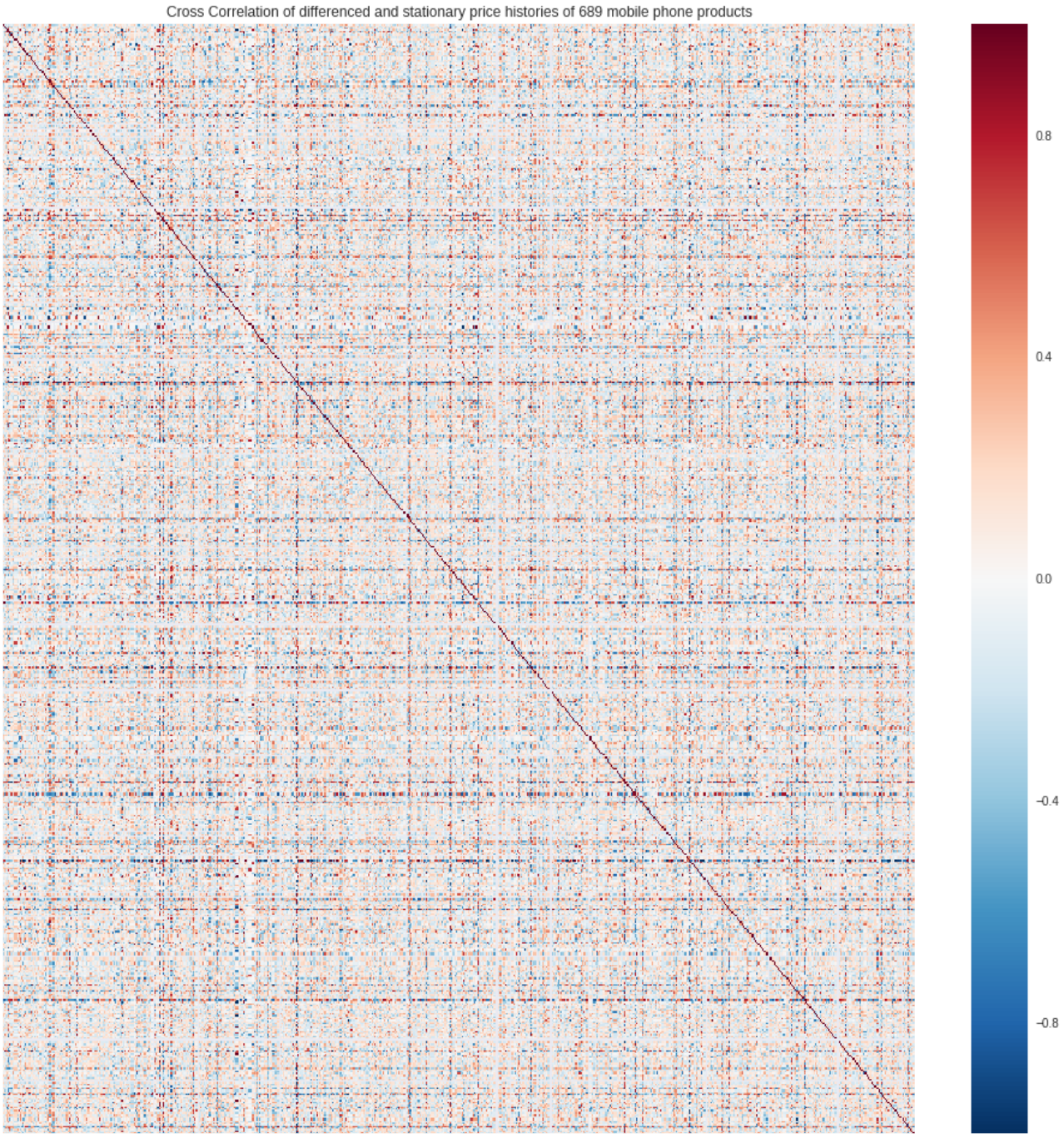


Figure73 Heatmap - Pearson Correlation - Differentiated Price Histories

Chapter 13
Results and Conclusions

13.1 Summary of Results

We are providing a summary of the results of all the experiments with the various models that were trained for our very difficult price prediction task.

All experiments were executed after removing mobile phone products which were classified as outliers and after removing mobile phone products that had a price history length smaller than 90 days. This reduced our mobile phone products from 781 to 656.

13.1.1 Smaller Subset of 95 mobile phone products with 210 days of price history

The first set of experiments were executed by choosing a subset of the 656 mobile phone products. The subset included price histories which had a fixed length of 210 days. 95 mobile phone products shared this characteristic.

13.1.1.1 Results

Model	Training Epochs	Average Huber Loss	Average Dynamic Time Warping Score
Baseline	-	4.78770150	153.97031739
Single RNN Model with Basic RNN cell	10	7.40530299	219.42501921
Single RNN Model with GRU cell	10	3.26558178	97.12150262
Single RNN Model with GRU cell	50	0.80812264	31.50652536
Sliding Window Model	35	8.33655582	258.45237553
Sequence to Sequence Model with Targets as Decoder Inputs	10	0.82026691	14.33995578
Simple Sequence to Sequence Model for Price Prediction	200	0.703325796	24.855095998

13.1.2 Full Dataset

Next set of experiments contains models which are trained against the instances being generated from the full range of 656 mobile phone products.

13.1.2.1 Results - without Price Normalization

Model	Training Epochs	Average Huber Loss	Average Dynamic Time Warping Score
Baseline	-	4.78770150	153.97031739
Advanced Sequence to Sequence Model	20	4.81190285	153.1493629
Advanced Sequence to Sequence Model including normalized targets	20	4.96664515	158.37255693
Advanced Sequence to Sequence Model including normalized targets, Batch Normalization and Dropout	30	5.0690297985	162.57913656

13.1.2.3 Results - Prices Normalized

Model	Training Epochs	Average Huber Loss	Average Dynamic Time Warping Score
Baseline	-	0.00568349	1.13140236
Sequence to Sequence Model with Batch Normalization, Dropout, Dynamic Decoder Inputs and Normalized Price Histories	100	0.00320502	1.15343520
Sequence to Sequence Price History and Static Mobile Attributes Model	100	0.00314844	1.17044361
Sequence to Sequence Price History and Static Mobile Attributes Model with Date Information	100	0.00361336	1.21531176

13.2 Conclusions

In this dissertation, we went from raw product data to building models for future price prediction. We used our domain knowledge to filter the attributes that were related to the price prediction task, filled the values that were missing and proved how these attributes contain useful information for the price of the product. After preprocessing the attributes and the price history of each product, we used these attributes as inputs to trais various architectures based on recurrent neural networks. Through our experiments, we have shown that a low fidelity price prediction for the next 30 days in future is possible. We have shown that the training is easier for a subset of the full dataset meaning that the number of instances is smaller and these instances share common characteristics. This suggests that for the price prediction task multiple models should be trained instead of a single one. The sudden changes of the price of a product in the future could not be predicted by any of our models. One reason for this could be because the underlying causes of these price changes could be unrelated with the past, such as having a special one-time discount for a product. Other reasons could be highly dynamic price changes due to day-to-day competition which again would not be related to the past and only depended on the previous or a few previous days. However, we cannot reach a definite conclusion until further exploration. We have also shown that the price histories are non-stationary and there are ways of converting them to stationary by differencing them which is a reversible conversion. Finally we have shown that the price histories are not completely independent from each other but there are more weak or stronger cross-correlations. This research is important because it provides machine learning practitioners the basis to building price prediction models and it is even more important because it has shown that recurrent neural network architectures being used in other domains can also be effective in price prediction using the same principles. There are three things suggested for future research. Firstly, letting the artificial neural network find all the features from raw data might require a too complex model, thus there is need to apply preprocessing using our domain knowledge, such as calculating the moving average. Secondly, grouping price histories together via a certain criterion, such as price histories that have shown a high cross-correlation could be used collectively as input to a model that is predicting the future price of a certain product. And finally, more appropriate cost functions should be explored in order to express explicitly to the neural network that predicting the sudden changes of a price is equally as important as capturing the general trend. It is not easy to prove that not enough information is hidden, in the collective history of the available information of the products, that enables or not the price prediction task but as Cher Wang, CEO of HTC company, famously noted "It takes humility to realize that we don't know everything, not to rest on our laurels and know that we must keep learning and observing".

Appendix 1
Software Stack

- **Python [https://www.python.org/]**: Many scientific packages are written in Python Programming Language
- **Unirest [http://unirest.io/]**: Lightweight HTTP Request Client Library
- **NumPy [http://www.numpy.org/]**: Python package to perform quickly and easily matrix and array operations
- **Pandas [https://pandas.pydata.org/]**: An open source, BSD-licensed, library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language
- **Scikit Optimization Module [https://scikit-optimize.github.io]**: The experiments of this thesis are using extensively the gp_minimize function which is Gaussian Processes used for Bayesian Optimization of hyperparameters of a model, as explained here [8].
- **Matplotlib [https://matplotlib.org/]**: Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
- **Scikit Learn [http://scikit-learn.org]**: Machine Learning in Python. Simple and efficient tools for data mining and data analysis. Accessible to everybody, and reusable in various contexts. Built on NumPy, SciPy, and matplotlib. Open source, commercially usable - BSD license
- **FastDTW [https://pypi.python.org/pypi/fastdtw]**: Python implementation of FastDTW [10], which is an approximate Dynamic Time Warping (DTW) algorithm that provides optimal or near-optimal alignments with an O(N) time and memory complexity.
- **Tensorflow 1.1 [https://www.tensorflow.org/versions/r1.1/]:** TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.
- **nolds [https://cschoel.github.io/nolds/nolds.html]**: a small numpy-based library that provides an implementation and a learning resource for nonlinear measures for dynamical systems based on one-dimensional time series
- **statsmodels [http://www.statsmodels.org/stable/index.html]**: statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration

All software implemented for this dissertation can be found in this link: https://github.com/pligor/predicting-future-product-prices [27]

Appendix 2
Figures of Centroids of K-means Clustering on Price Histories

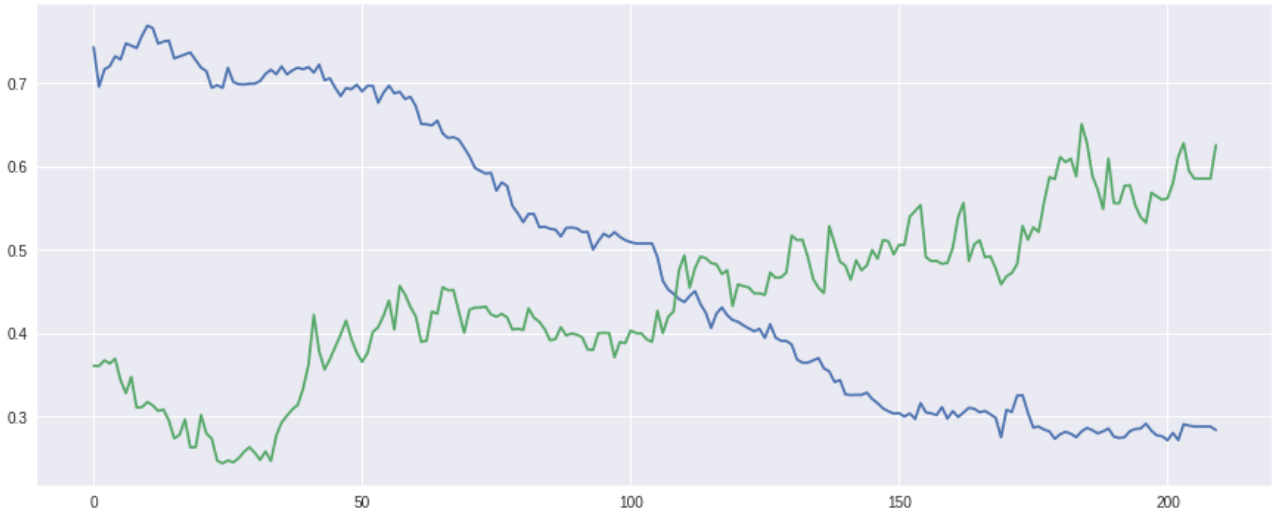


Figure74 K-means Clustering of 112 Price Histories for k=2

Figure75 K-means Clustering of 112 Price Histories for k=3

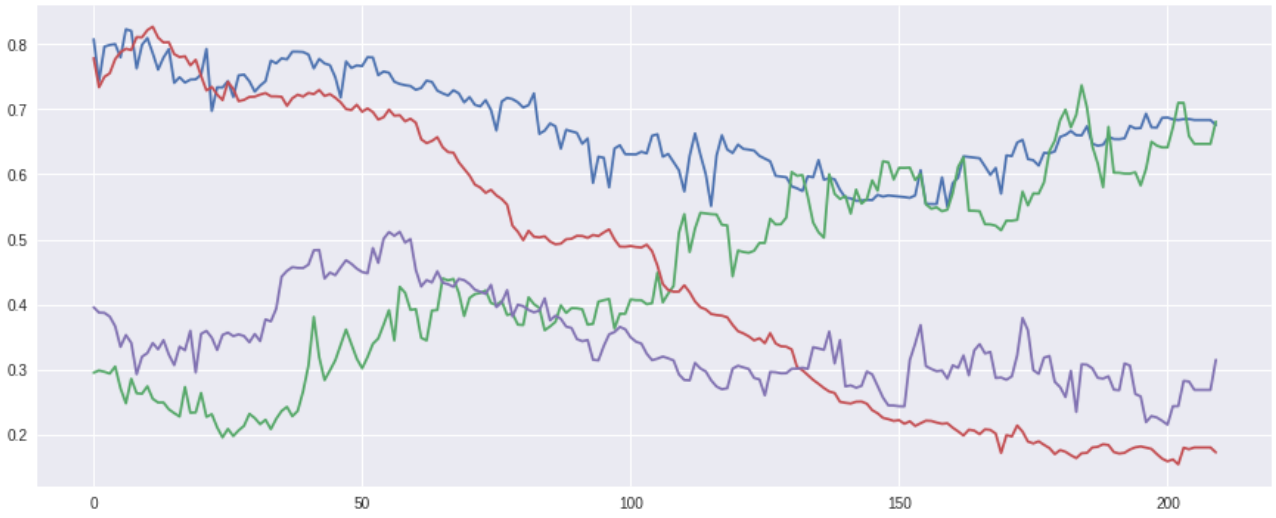


Figure76 K-means Clustering of 112 Price Histories for k=4

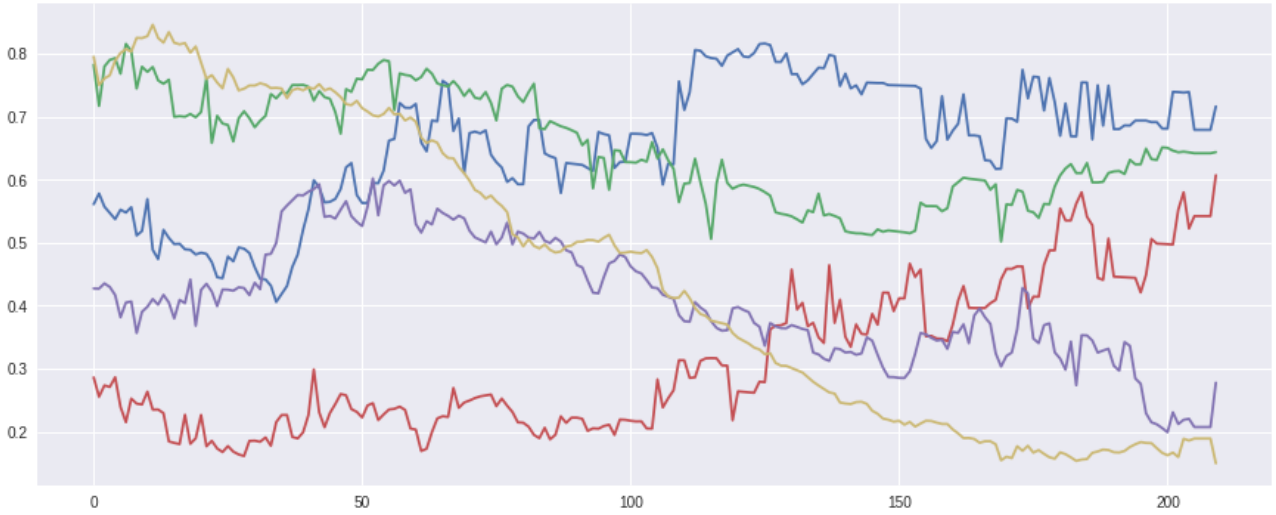


Figure77 K-means Clustering of 112 Price Histories for k=5

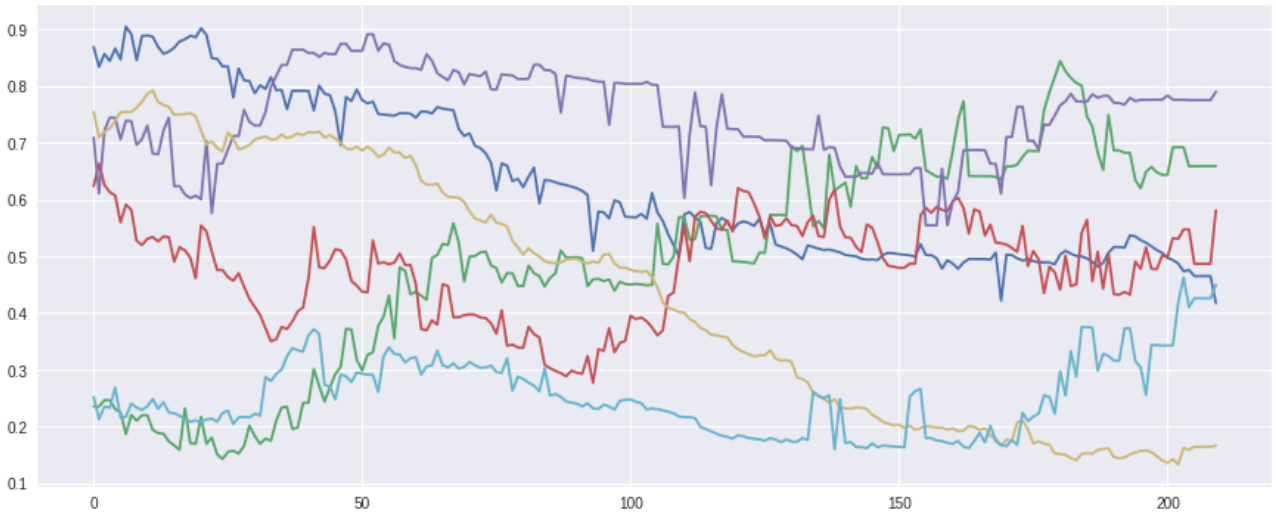


Figure78 K-means Clustering of 112 Price Histories for k=6

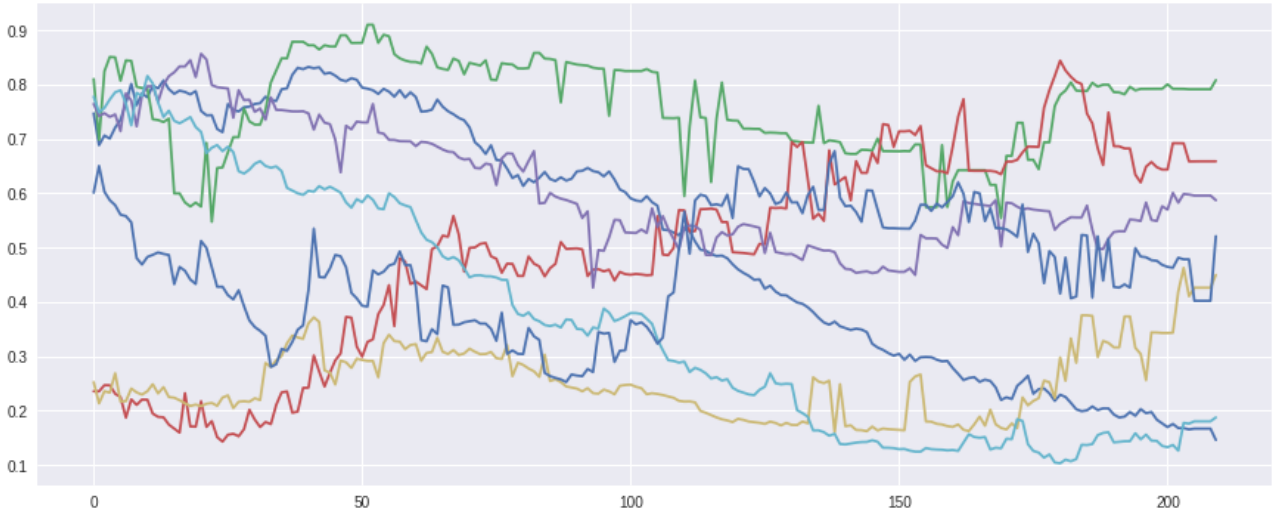


Figure79 K-means Clustering of 112 Price Histories for k=7

Appendix 3

Real-Valued or Integer Attributes

Attribute Name	Units	Sample Values	Relation to Value
Battery Capacity	mAh	3000, 4165, 2100	Mobile phones that last longer are preferred
Camera	Megapixels	8, 13, 5	High resolution pictures are preferred by customers
CPU Power	GHz	1.3, 1.8, 2.15	Larger CPU Power better responsiveness of the mobile phone which gives a better user experience
Diagonal Size	Inches	5.5, 2.4, 5.0	The smaller or larger diagonal size could make the size of the mobile phone more or less practical for a user. No linear relationship is expected here
RAM	GB	2, 3, 0.5	Larger RAM means that the mobile will be able to execute more applications in the background and in general improve the responsiveness of the phone which improves the user experience
Release Year	Year	2016, 2017, 2015	Newer mobile phones are perceived as of higher value
Reviews Count	Count	11, 51, 0	More reviews make average review rating more credible which depending on the rating could decrease or increase the perceived value. Also lots of rating means that the mobile product attracts the interest of the public
Review Score	Score 0 to 5	4, 3.0625, 4.96296	Opinions of people who have purchased
SAR	W / Kg	0.47, 0.98, 0.378	This is an indication of the radiation. Depending on the health concerns of the individual customer a mobile phone with a high SAR could be a deal breaker
Selfie Camera	Megapixels	5, 0.3, 1.3	Higher resolution will produce better pictures which is preferred
Shop Count	Count	38, 28, 2	If a mobile phone product is being sold

			in many electronic shops this could affect the perceived popularity that the customer has for the product
Speaking Autonomy	Hours	14, 21, 60	Customers that have the habit of spending large amounts of their time speaking on the phone will prefer mobile phones with high speaking autonomy
Standby Autonomy	Hours	270, 268, 528	Consuming the minimum amount of energy so that the phone can be used after many hours of being in standby without needing recharging is a preferred experience
Storage	GB	16, 4, 128	Having a large storage for installing many applications and storing many photographs and videos is preferred
Weight	gr	145, 172, 142.6	The feeling of the weight of the mobile phone when holding it on your hand or when the user needs to hold it near his or her ear during a long phone call affects the perceived value

Binary Attributes

Attribute Name	**Relation to Value**
Wireless Charging	Not needing to have a physical connection to charge the mobile phone means no wires are necessary to be carried around nor any erosion of any plugs which is seen as an advantage from the potential customers
Removable Battery	If the battery is removable then it can either be replaced by a larger battery of it can be replaced with a battery of the same capacity when the performance of the original battery decreases. This is seen as an advantage from the potential customer
Card Slot	If the phone allows an extra card to be inserted which expands its storage it is preferred from the potential customers
Double Back Camera	When the mobile phone product has two cameras then it can take stereoscopic shots which can be seen as 3D by wearing special 3D glasses which is an advanced feature that might be essential to the potential customer

Fast Charge	The need of charging an empty battery as quickly as possible is preferred by the potential customers
Flash	Without having a flash the user might be unable to take any high quality photos under low light conditions which will affect the user's preference
Bluetooth Connectivity	Most wireless handsfree devices need bluetooth communication and therefore customers prefer mobile devices that support the Bluetooth protocol
Lightning Connectivity	Lightning Connector is only found on Apple Inc. products and depending on the rest of the Apple devices a user might have this could be a necessary accessory
Mini-Jack Connectivity	For wired handsfree or if a customer wants to listen to music or other kind of audio in a private mode he needs to be able to connect earphones with or without a microphone in his mobile phone product
NFC connectivity	NFC tags and NFC connectivity in general give the ability to the mobile phone product to act as a contactless bank card. The ability to pay without needing to carry your wallet could be seen as an added value by many customers
USB connectivity	To transfer files to and from any modern computer a USB connectivity is the only kind of connectivity that allows it without any extra setup, only by plug-n-play functionality
USB Type C connectivity	USB Type C is a new type of connection that allows wired connection among two devices without the need for alignment before plugging. All previous USB types could be plugged in with only the "right" way. This kind of convenience could be considered as a plus from the potential customer
WiFi connectivity	Since 4G connections are not ubiquitous the customers rely on WiFi connections for a fast and limitless connection to the internet. Customers that use the internet on their mobile phone will usually require WiFi connectivity as well
Natural Keyboard control	Touch screen typing is difficult for some users who prefer a natural keyboard to have a tactile sensing when typing
Touchscreen control	On the other hand there are customers who prefer the real estate of the mobile phone is utilized for display purposes and therefore they prefer the touch-screen which is hidden when no input from the user is required
Dust Resistant protection	Depending on the environment that a user is found the protection from dust might be necessary

Rugged protection	Depending on the environment that a user is found rugged protection from severe strikes or falls might be necessary because most of the mobile phone products on the market are considered as fragile
Water Resistant protection	There could be circumstances like talking in a rainy environment that makes the water resistant feature required
Accelerometer sensor	Being able to measure the acceleration of the phone allows for practical usages such as measuring the steps that a person takes every day
Altimeter sensor	If you are a climber then knowing how high you are above the sea level might be a feature that adds extra value to the mobile phone
Barometer sensor	This is particularly useful to people who have some knowledge in meteorology and that they want to compare the look of the sky or the horizon with the indication of the barometer
Compass sensor	Usually used in conjunction with GPS functionality to show not only the position of the user on the map but also his or her direction. It improves the User Experience
Fingerprint sensor	Allowing authorized access without the need of remembering long and difficult passwords is a feature which saves time and increases security for users of mobile phones with a fingerprint sensor
Gyroscope sensor	This is used a lot in gaming where users are able to control the game by simply stirring their mobile phone in mid-air. This experience might be necessary from the potential customer
Hall sensor	Most mobile phone products with a flip cover use a hall sensor to detect whether the flip cover goes on and off
Heartbeat Meter sensor	This is mandatory for people with heart diseases that are looking for an affordable sensor that does not require extra hardware
Iris Scanner sensor	Another feature of biometric security is the use of the iris as a unique identifier
Light sensor	Sensing the ambient light of the environment allows mobile phones to adjust the brightness of the screen automatically and thus reduce the battery consumption
Oximeter sensor	Users with health concerns that need to monitor the level of oxygen in their blood on a regular basis will find

	convenient the measurement of the oxygen level with laser technology
Proximity sensor	Typical use case of the proximity sensor is when the ear comes in contact with the screen and generates touch events while being on a call. Proximity sensor prevents this bad user experience
Tango sensor	If a potential customer needs to use extensively applications that are related with augmented reality then the tango sensor helps its functionality by using computer vision to detect the position relative to the world around the user without the use of GPS

Categorical Attributes

Attribute Name	**Categories**	**Relation to Value**
Connection Network	2G, 3G, 4G	The potential customer will want to choose mobile phone products that support the latest available protocol especially if he or she want to have a fast access to the internet
Mobile Type	SmartPhone, Feature Phone, for Elders	This attribute separates mobile phone products in three very distinctive use cases. Feature phones are in comparison cheaper phones for basic phone and messaging usage. Mobile phones for elders have special interface that accommodates the needs of people with minimum knowledge of technology and the lack of interest to learn new technologies while they still want to communicate. Lastly, Smartphones are all the rest.
Manufacturer	Here is a long list of manufacturers	There are brands that have fans who are consistently purchase mobile phones from their favorite brand
Operating System	Android, iOS, Windows Phone, Proprietary, RIM BlackBerry	Each operating system provides a different user experience which some users find more comfortable to work with. In addition changing an operating system usually comes with a learning curve and some infrastructure costs such as moving all of your contacts to the new system

Special Attributes

Attribute Name	Special Preprocessing	Relation to Value
CPU cores	The possible values for the CPU cores are: 1, 2, 2+2, 3, 4, 4+2, 4+4, 4+4+2, 5, 7, 8 But since there is a relation between the values it would be misleading to treat it as a categorical attribute. This is why we are treating it as ordinal attribute assigning a positive increasing number to each category: '1': 1, '2': 2, '2+2': 3, '3': 4, '4': 5, '4+2': 6, '4+4': 7, '4+4+2': 8, '5': 9, '7': 10, '8': 11	Users have the experience that more cores could mean more computing power or that it could mean better responsiveness of the mobile phone when there are lots of applications running in the background
SIM	SIM has two possible values: ‘Single’ or ‘Dual’. We are treating it as an ordinal values assigning a positive number equal to the number of sim cards that are allowed: ‘Single’: 1 ‘Dual’: 2	Having two sim cards brings an extra advantage to having only one sim card
Screen Resolution	It comes in a string format having the height and the width concatenated with an ‘x’ as separation symbol. We are splitting the information of height and width to two attributes and we are also multiplying them to create a third attribute which corresponds to the value of the nominal screen resolution in Megapixels units	High screen resolution brings a better visual experience to the user especially in the use cases of seeing an HD image or video or even when reading text
Dimensions	It comes in a string format having the height, width and depth concatenated in a single string and separated with an ‘x’ symbol. We are splitting the string to get the individual values and we are creating the following attributes: ● width ● height ● depth ● area: width x height ● area: height x depth ● area: width x depth ● volume: width x height x depth	The dimensions correspond to the look of the mobile phone product and how it feels holding it at your hands. How thin it is could bring different perceived to different users

Targets

Attribute Name	Units	Sample Values	Explanation
Minimum Price	Currency	99.99, 235.86, 358.74	The minimum price as seen so far
Maximum Price	Currency	151.69, 259.0, 360.0	The maximum price as seen so far

Dropped Attributes

Attribute Name	Sample Values	Explanation
Display Name	‘BlackView A8 Max (16GB)’, ‘Apple iPhone 6 (64GB)’, ‘Samsung Galaxy Trend 2 Lite (4GB)’	Simply for displaying purposes
Name	Pixel (128GB), iPhone 6 (64GB), X6 (8GB)	A more formal description of the name of the product excluding the brand name which is mainly for displaying purposes
Main Image	‘https://b.scdn.gr/images/sku_main_images/008491/8491027/medium_20160303160934_xiaomi_mi_5_pro_64gb.jpeg’	Exploring how the look and feel of a mobile phone product is related to its price using deep neural network techniques to extract useful features from the image is out of the scope of this thesis
Screen Type	‘Retina HD display, ED-backlit widescreen Multi‑Touch display with IPS technology’, 'IPS OGS capacitive screen, 160K Colors’, ‘TFT LCD, 65K colors’	There is a broad range of technical terms that are used to describe the type of the screen of the mobile phone product. There is no easy comparison between the products unless the potential customer is familiar with all the screen technologies. We claim that there is minimum information related to the value of the phone and there is no straightforward way to convert these strings into numerical values
Id	11067464, 7219275, 11343960	simply a unique numerical identification per mobile phone product
Plain Spec Summary	‘Οθόνη: 5.5", RAM: 2 GB, Μνήμη: 16 GB, Κάμερα: 8 MP, Δίκτυο: 4G, Πυρήνες CPU: 4, SIM: Dual, Μπαταρία: 3000 mAh’	Only for displaying purposes is a concatenation in a single string of the most important features of a product
Pn	‘NXT-L09 32GB’, ‘WAS-LX1’,	A code-name used by

	‘C000238’	manufacturers to specify a certain product uniquely even if its name on the market changes. Mainly used as an internal reference
virtual	0	all values are identical across all instances thus they hold zero information
future	0	all values are identical across all instances thus they hold zero information

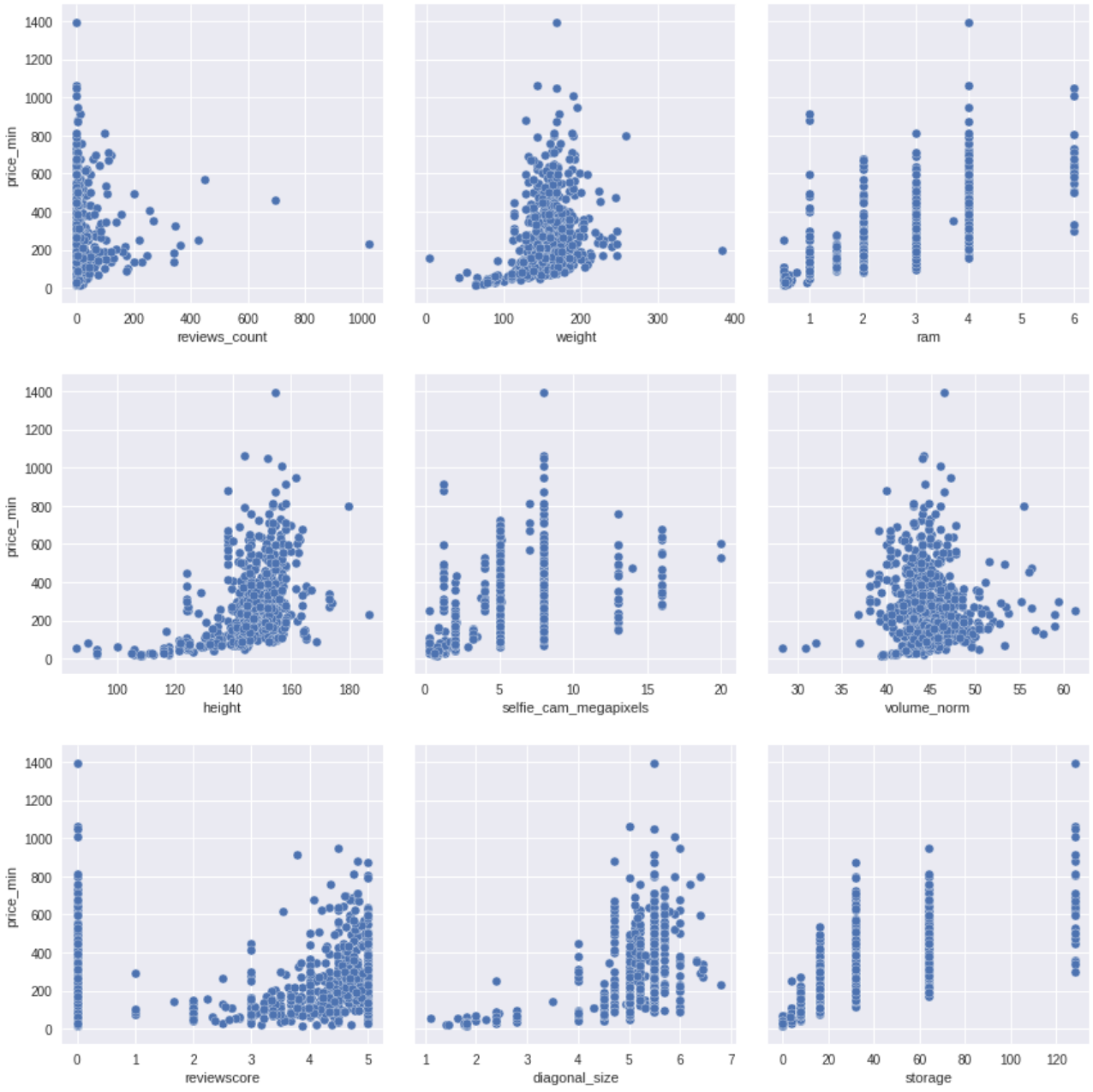
Appendix 4
Best Imputation Algorithm per Attribute

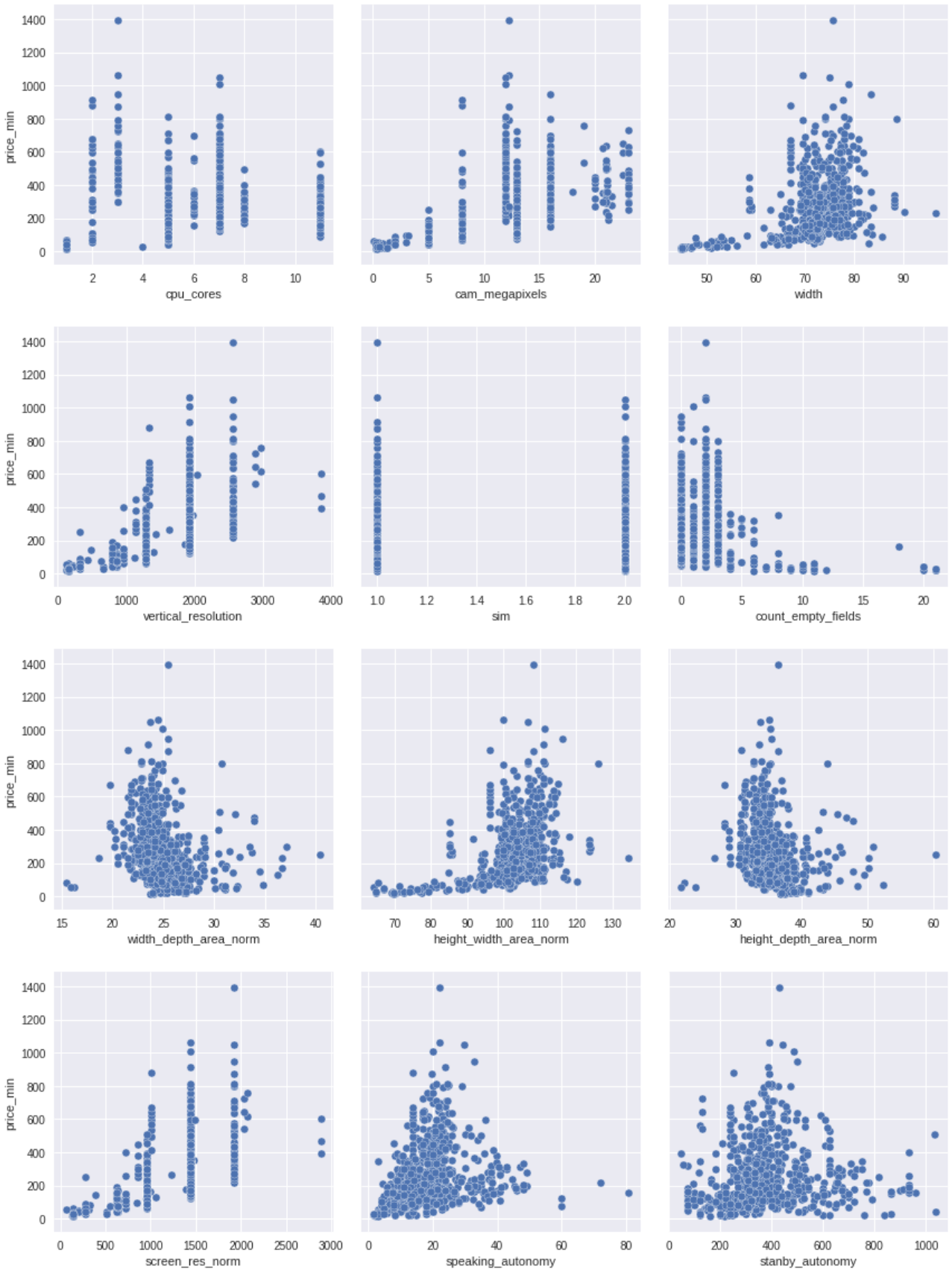
Attribute Name	KNN (k=5)	Soft Impute	Iterative SVD	MICE	Nuclear Norm Minimization	Prior Domain Knowledge
CPU cores	✔					If not smartphone most likely has only one CPU core, otherwise gets rounded to integer
Wireless Charging	✔					As expected all zeros since wireless charging is a rare feature
Removable Battery		✔				rounded to binary value 0 or 1
Card Slot	✔					rounded to binary value 0 or 1
Double Back Camera	✔					rounded to integer
Fast Charge			✔			rounded to binary value 0 or 1
Flash	✔					rounded to binary value 0 or 1
Dust Resistant Protection	✔					rounded to binary value 0 or 1
Rugged Protection	✔					rounded to binary value 0 or 1
Water Resistant Protection	✔					rounded to binary value 0 or 1

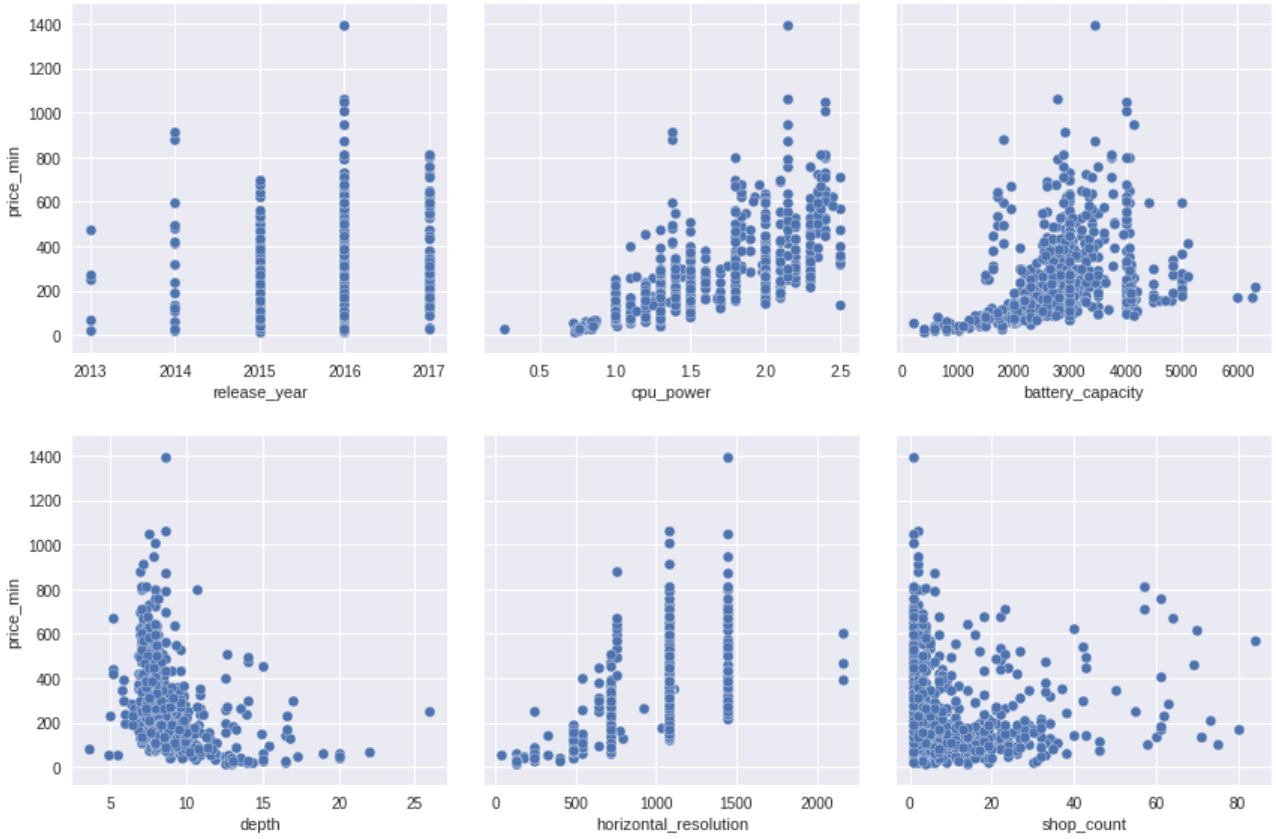
Altimeter Sensor					✔	rounded to binary value 0 or 1
Barometer Sensor					✔	rounded to binary value 0 or 1
Compass Sensor				✔		rounded to binary value 0 or 1
Fingerprint Sensor				✔		rounded to binary value 0 or 1
Gyroscope Sensor					✔	rounded to binary value 0 or 1
Hall Sensor				✔		rounded to binary value 0 or 1
Heartbeat Meter Sensor			✔			rounded to binary value 0 or 1
Iris Scanner Sensor					✔	rounded to binary value 0 or 1
Light Sensor		✔				rounded to binary value 0 or 1
Oximeter Sensor					✔	rounded to binary value 0 or 1
Proximity Sensor					✔	rounded to binary value 0 or 1
Tango Sensor				✔		rounded to binary value 0 or 1
Camera Megapixels	✔					rounded to one decimal
CPU Power		✔				rounded to two decimals
RAM	✔					rounded to two decimals
Release Year				✔		rounded to integer and thresholded to be a valid year number that is not referring to the

						future
Selfie Camera Megapixels	✔					rounded to one decimal
Speaking Autonomy	✔					rounded to one decimal
Standby Autonomy			✔			rounded to integer
Storage			✔			rounded to integer
Weight	✔					rounded to one decimal
Connection Network	✔					rounded to integer
Operating System	✔					rounded to integer
Dimensions			✔			rounded to integers
Horizontal Resolution				✔		rounded to integer
Vertical Resolution				✔		rounded to integer

Appendix 5
Static Real-Valued Attributes of Mobile Phone Products against Price




price_min
cpu_cores
cam_megapixels
width
vertical_resolution
sim
count_empty_fields
width_depth_area_norm
height_width_area_norm
height_depth_area_norm
screen_res_norm
speaking_autonomy
stanby_autonomy


price_min
release_year
cpu_power
battery_capacity
depth
horizontal_resolution
shop_count

Bibliography

1. Weiss, Robert M., and Ajay K. Mehrotra. "Online dynamic pricing: Efficiency, equity and the future of e-commerce." *Va. JL & Tech.* 6 (2001): 1.
2. Mazumder, Rahul, Trevor Hastie, and Robert Tibshirani. "Spectral regularization algorithms for learning large incomplete matrices." *Journal of machine learning research* 11.Aug (2010): 2287-2322.
3. Troyanskaya, Olga, et al. "Missing value estimation methods for DNA microarrays." *Bioinformatics* 17.6 (2001): 520-525.
4. Azur, Melissa J., et al. "Multiple imputation by chained equations: what is it and how does it work?." *International journal of methods in psychiatric research* 20.1 (2011): 40-49.
5. Candès, Emmanuel J., and Benjamin Recht. "Exact matrix completion via convex optimization." *Foundations of Computational mathematics* 9.6 (2009): 717.
6. Tukey, John W. "Exploratory data analysis." (1977): 2.
7. Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest." *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008.
8. Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical bayesian optimization of machine learning algorithms." *Advances in neural information processing systems*. 2012.
9. Choi, Keunwoo, et al. "Convolutional recurrent neural networks for music classification." *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017.
10. Salvador, Stan, and Philip Chan. "Toward accurate dynamic time warping in linear time and space." *Intelligent Data Analysis* 11.5 (2007): 561-580.
11. Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures." *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015.
12. Brownlee, Jason Dr. “Time Series Forecasting with the Long Short-Term Memory Network in Python.” Machine Learning Mastery, Dr. Jason Brownlee, 19 July 2017, machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/.
13. Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *Advances in neural information processing systems*. 2014.
14. Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).
15. Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." *Journal of Machine Learning Research* 13.Feb (2012): 281-305.
16. Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *International Conference on Machine Learning*. 2015.
17. Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580* (2012).

18. Pricefrontier.com. (2017). Amazon Price Prediction Model | PriceFrontier. [online] Available at: http://www.pricefrontier.com/prediction [Accessed 7 Aug. 2017].
19. Connor, Jerome T., R. Douglas Martin, and Les E. Atlas. "Recurrent neural networks and robust time series prediction." IEEE transactions on neural networks 5.2 (1994): 240-254.
20. Afolabi, Mark O., and Olatoyosi Olude. "Predicting stock prices using a hybrid Kohonen self organizing map (SOM)." System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on. IEEE, 2007.
21. Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).
22. Rousseeuw, Peter J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." *Journal of computational and applied mathematics* 20 (1987): 53-65.
23. Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *science* 313.5786 (2006): 504-507.
24. Hardstone, Richard, et al. "Detrended fluctuation analysis: a scale-free view on neuronal oscillations." *Frontiers in physiology* 3 (2012).
25. Banerjee, Anindya, et al. "Co-integration, error correction, and the econometric analysis of non-stationary data." *OUP Catalogue* (1993).
26. Devadoss, A. Victor, and T. Antony Alphonnse Ligori. "Stock prediction using artificial neural networks." *International Journal of Data Mining Techniques and Applications* 2 (2013): 283-291.
27. Georgios Pligoropoulos, Predicting Future Product Prices, (2017), GitHub repository, https://github.com/pligor/predicting-future-product-prices