ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Price prediction and forecasting for items in the online game Path of Exile using Machine and Deep Learning methods

ΤΖΙΒΑΚΗΣ ΔΗΜΗΤΡΙΟΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής, Ε.Μ.Π.

Αθήνα, Μάρτιος 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Price prediction and forecasting for items in the online game Path of Exile using Machine and Deep Learning methods

ΤΖΙΒΑΚΗΣ ΔΗΜΗΤΡΙΟΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής, Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Μαρτίου 2019

.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Στάμου	Δημήτριος Τσουμάκος
Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής, Ιόνιο Πανεπιστήμιο

.....

Τζιβάκης Δημήτριος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

ΠΕΡΙΛΗΨΗ

Η επίτευξη του μεγαλύτερου οικονομικού κέρδους σε ένα online παιχνίδι με τις λιγότερες επενδύσεις στη διάρκεια του χρόνου είναι ίσως μία από τις πιο συχνές ερωτήσεις που έχουν οι νέοι παίκτες.
Απαιτούνται εκτενείς γνώσεις, ειδικά για πολύπλοκα παιχνίδια με οικονομικό περιβάλλον – πχ συναλλαγές - εντός παιχνιδιών. Χρησιμοποιώντας το Path of Exile ως το παιχνίδι της επιλογής μας, έχουμε δημιουργήσει ένα σύνολο δεδομένων με συναλλαγές αντικειμένων που γίνονται μεταξύ των παικτών στο παιχνίδι αξιοποιώντας το ενσωματωμένο API της πλατφόρμας. Εφαρμόζοντας μεθόδους μηχανικής μάθησης στο σύνολο των δεδομένων επιδιώκουμε να βρούμε τρόπους να μειώσουμε τις γνώσεις που απαιτούνται από τους νέους παίκτες, για την απόκτηση αξίας μέσα στο παιχνίδι γρηγορότερα και αποτελεσματικότερα μέσω συναλλαγών αγοράς και πώλησης.
Για το σκοπό αυτό κατεβάσαμε δεδομένα χρονικης διάρκειας 6 μηνών από το API της πλατφόρμας και εφαρμόσαμε τεχνικές ώστε να διαβάσουμε τα δεδομένα, να τα μετατρέψουμε σε ευανάγνωστη μορφή και να τα αποθηκέυσουμε σε μια βάση δεδομένων που βασίζεται σε έγγραφα - δεδομένου ότι τα δεδομένα είναι από τη φύση τους ανάλογη αυτης των εγγράφων. Χρησιμοποιήσαμε κατόπιν στατιστική ανάλυση και γνώση του παιχνιδιού για να σχεδιάσουμε χαρακτηριστικά, τα οποία αργότερα θα βοηθήσουν τα μοντέλα μηχανικής μάθησης να έχουν καλύτερη απόδοση.

Παράγουμε τελικά δύο σύνολα δεδομένων:

• Ένα σύνολο δεδομένων που περιλαμβάνει αντικείμενα που παρουσιάζουν ένα σταθερό σύνολο χαρακτηριστικών στο χρόνο.
Χρησιμοποιούμε τα επαναλαμβανόμενα νευρωνικά δίκτυα με τη μορφή LSTM για την κατασκευή μοντέλων, με σκοπό τη πρόβλεψη των μελλοντικών τιμών των αντικειμένων αυτών.
Τα μοντέλα αυτά, μπορούν αργότερα να χρησιμοποιηθούν για να βοηθήσουν τους παίκτες να αποφασίσουν να αγοράσουν ή να πουλήσουν αντικείμενα σύμφωνα με το πρότυπο πρόβλεψης.
• Ένα σύνολο δεδομένων που περιλαμβάνει στοιχεία που παρουσιάζουν ένα μεταβλητό αριθμό χαρακτηριστικών.
Χρησιμοποιούμε διαφορετικούς αλγορίθμους μηχανικής μάθησης καθώς και μοντέλα τεχνητού νευρικού δικτύου για να προβλέψουμε την τιμή ή να την ταξινομήσουμε σε ομάδες κέντρων τιμών που έχουν νόημα σε έναν παίκτη. Χρησιμοποιώντας αργότερα αυτούς τους εκτιμητές, ένας παίκτης μπορεί να πάρει πληροφορίες σχετικά με την τιμή των αντικειμένων και να αποφασίσει εάν ένα στοιχείο είναι πολύτιμο και πόσο.

Λέξεις- Κλειδιά : Μηχανική Μάθηση, Επαναλαμβανόμενα Νευρωνικά Δίκτυα, LSTM, Στατιστική, Πρόβλεψη, Κατηγοριοποίηση, Προτυποποίηση, Γραμμικη Παλινδρόμηση

ABSTRACT

Making the most profit in an online game with the least time investment is perhaps one of the most asked questions new players have. It requires a vast amount of knowledge especially for complex games with in-game economies. Using Path of Exile as our game of choice, we have built a dataset of in-game item transactions using its built-in API and using machine learning we analyse the dataset to find ways to reduce the knowledge needed for new players to generate currency faster and more efficiently through buy and sell transactions.
After downloading 6 months' worth of data from the API feed, we apply a pipeline in which we read the data, convert it to a readable format using a document based database - since natively the feed schema is highly nested- and use statistics and domain knowledge to engineer features which will later help our machine learning models perform better.
We finally produce two datasets:

- A dataset comprising of items that present a firm set of attributes through time.
  We use Recurrent Neural Networks in the form of LSTM to build models to try and forecast the future prices of these items.
  These models can later be used to help players decide where to buy or sell the items according to our forecast model.

- A dataset comprising of items that present a variable number of features.
  We use different machine learning algorithms as well as Artificial Neural Network models to predict the price of an item or classify it in range price clusters meaningful to a player. A player can later use these estimators, to get information on the price of items and decide if an item is valuable and how much.

Keywords: Machine Learning, Recurrent Neural Networks, RNN, LSTM, Artificial Neural Network, ANN, Linear Regression, Linear classifier, Decision Trees, XGBoost, K - means Clustering, K-nearest Neighbors, Activation functions, Swish Activation Function, sigmoid Activation Function, Softmax Activation Function, Linear Activation Function, Feature Scaling, Feature Engineering, Min-Max Scaler, Standard Scaler, K-fold, Grid Search, Ensemble

CONTENTS

List of Figures

List of Tables

Chapter 1: Overview

1.1 The problem

There have been many games, throughout the years, which involved an in-game economy which flourished through selling and buying items found by the players to the players. Diablo 3 implemented real money trading via a real money auction house, with merchandise being its in-game items. Path of Exile, the game whose data we collected and used, implemented trading with in-game currencies, each of different value. While figuring out the price of an item or if the price of an item is going to change in the future will help you increase your riches, it requires immense knowledge of the game's mechanics and quirks and lots of game time. Considering that Path of Exile has a continuously changing market with more than 172 quadrillion different possible combinations for its items and more than 50 different currency types, makes this a difficult feat.

1.2 The game

In Path of Exile, players have different goals and in the process of achieving those goals, they acquire items with a variable number of features, picking from more than 90 different features for its item category. The items can either be used, sold or just thrown away. Depending on how many features an item has and which are those features, the worth of the item varies. All this takes place in "leagues" as they are called, which are 3-month periods, after which all the items and currency a player has accumulated goes away and they start from the beginning.

The game's items are split to different categories and rarities. Rare items have 2 or more features (in the game the features are called "mods") and names depending on those features, while Unique items have specific features and names. For each item's feature, be it of rare or unique rarity, its features can have different ranges. A feature with a value of 10 on one instance of an item can have a value of 100 in a different instance. The feature's value range depends on the category of the item, on the instance of the item and on the level of the item.

Players in the game mainly use two types of currencies to trade, chaos orbs and exalted orbs. Exalted orbs are worth multiple times the value of a chaos orb (generally the rate is more than 100 chaos orbs to 1 exalted orb). Chaos orbs are the main currency players use to trade. For that reason all prices are converted in chaos orbs using the daily exalted to chaos orbs conversion rate which was data mined from a poe.ninja.

Trading in the game is done using stashes. Each player has a number of stashes which serve the purpose of showcasing the item. Using the game's API, we can capture all the changes made to a stash, using that stashes' unique id. For example, if at a specific point in time a stash with an id Y contains an item X, and later on that stash doesn't contain that item, we can assume item X has been sold.

1.3 Goal

The goal is to be able to predict the prices of rare items within a good error margin or at least classify what price range they belong in as well as forecasting the price of unique items. Predicting the price of a rare item will allow players to decide if they want to throw away that item or try and sell it. The forecasts for unique items will be implemented using observations at the start of a season and will inform the player if the price of the item will rise or drop, allowing him to buy many of the same items for profit or sell his items now so he does not lose money.

Chapter 2: Neural Networks Theory

2.1 Neural Networks

Artificial neural networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input. Artificial neural networks (or ANNs) help us cluster and classify. We can think of them as a clustering and classification layer on top of the data we store and manage. They help to group unlabelled data according to similarities among the example inputs, and they classify data when they have a labelled dataset to train on.

The building unit of the neural networks is called the neuron, and imitates the functionality of the human neuron. Neurons are connected in layers so that one layer can communicate with the others forming a neural network. Every layer, other than the input and the output layers, is called a hidden layer [1]. The output of one layer is fed to the inputs of another layer.

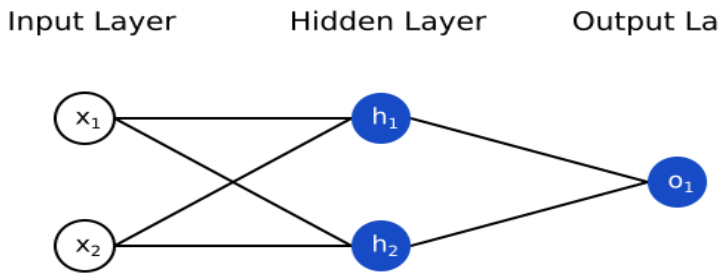


Figure 1: Schematic of Neural Network model

The main goal of the ANN is to “learn” so it can cluster or classify data. Learning is the task of adjusting weights to minimize the error. That is performed by back propagation of error.

According to back propagation, after we initialize a loss function, which is dependent on the output of the last layer, back propagation tries to minimize that function.

By analysing and by carefully picking out the non-linear functions, backpropagation ends up on the dependence of the output layer to the input layer, going through the neural network in reverse order and by computing at each step the derivative of the cost function in relation to the parameter traveled.

Afterwards, by gradient descent or variations, the values of all parameters (which is the weights of each input and the bias) are changed in such a way that the value of the cost function is minimized (depending on its sign and measure derivative in each step).

Then, it tries cluster or classify with the new parameters and the same process is repeated. Finally, the cost function will converge to a minimum value and the system will more often predict the exit.

2.2 Recurrent Neural Networks

In the case of sequential data, such as text or speech signals, in which there are high dependencies between the data features, then really helpful, are the RNNs or Recurrent Neural Networks.

In contrast to RNNs, in a typical Feed-Forward Neural Network like the ANNs we described, the information only moves in one direction, from the input layer, through the hidden layers, to the output layer [2]. The information moves straight through the network. Because of that, the information never touches a node twice. In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously. Therefore a Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

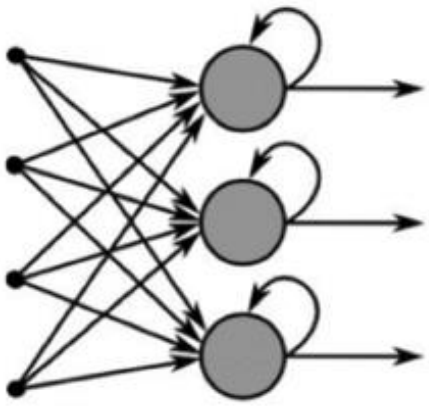


Figure 2 : Recurrent Neural Network

RNN's have two major obstacles, the exploding gradient and the vanishing gradient. The exploding gradient is the assignment of a really high importance to the weights. Vanishing gradient is when the values of a gradient are too small and the model stops learning or takes way too long to learn. LSTM solved both of these issues.

LSTM or Long Short-Term Memory networks are an extension of recurrent neural networks. They are well suited to learn from important experiences, past data points, that consist of long time lags [3].

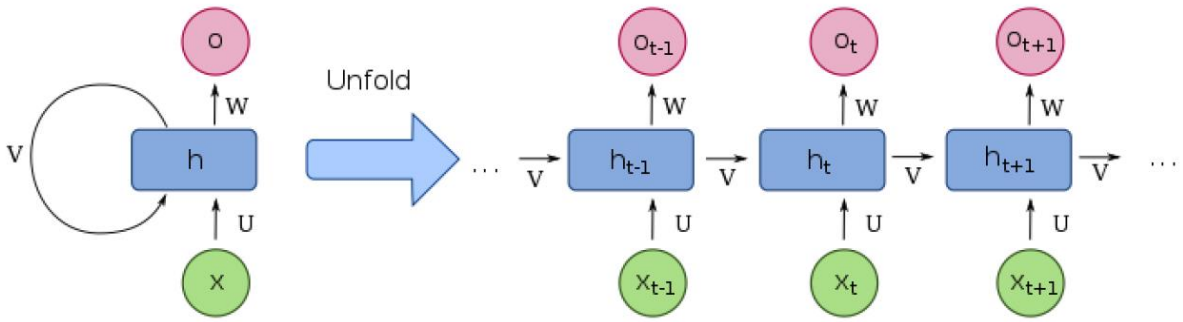


Figure 3: Basic RNN architecture

LSTM consists of many complex cells, with 3 gates each, input, output and forget. A gate is basically a neuron consisting of parameters that can be tweaked and optimized to minimize the cost function just like in the rest of neural networks. This happens with variations of the backpropagation algorithm. Each gate has each own parameters, different activation functions and different inputs and outputs.

A feature of LSTM is that it maintains two internal states and the neuron structure, mainly the forget gate, enables the network to learn what it should "remember" at any stage of the sequence. A common example is language modeling with LSTM

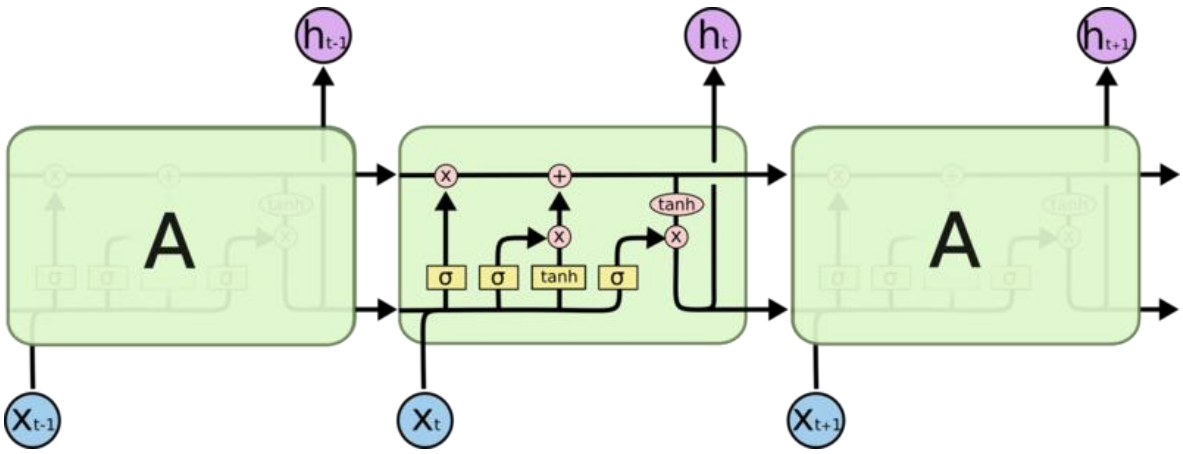


Figure 4 : LTSM chain along with the gates

The input of the cell Xt at time t (for data organized in time sequence) is fed to the various gates (Ft, It, Ot) and based on the parameters of the gates and the states ct, ht determine the output ot and the states of the cells in the next step. At a LSTM layer, the internal states are vectors of length L, so after processing the input, a vector with L attributes appears as the output of the plane. At the output it is possible to also obtain a sequence of the same length as the original, consisting of the outputs Ot [L] for each step of the input sequence. Finally, we are able to get the Ct, Ht states either at each step of the sequence, or only the last state (after passing the whole sequence).

2.3 Arima vs LSTM

All observations in Time Series data have a time stamp associated with them. These observations could be taken at equally spaced points in time or they could be spread out unevenly. Any time series data has two components – trend (how data is increasing or decreasing over time) and seasonality (variations specific to a particular time frame). For time-series data two of the most common analysis approaches are forecasting and pattern and outlier detection [4].

Forecasting [16] time series data has been around for several decades with techniques like ARIMA [8]. Recently Recurrent Neural Networks (LSTM) have been used with much success.

Advantages of using ARIMA :

- Simple to implement, no parameter tuning
- Easier to handle multivariate data
- Quick to run

Advantages of LSTM:

- No pre-requisites (stationarity, no level shifts)
- Can model non-linear function with neural networks
- Needs a lot of data

Since we didn't want to deal with the stationarity, seasonality and trend of our data series and having to convert it specifically for ARIMA, we decided to use LSTM.

Chapter 3: Building the Dataset

3.1 System Architecture

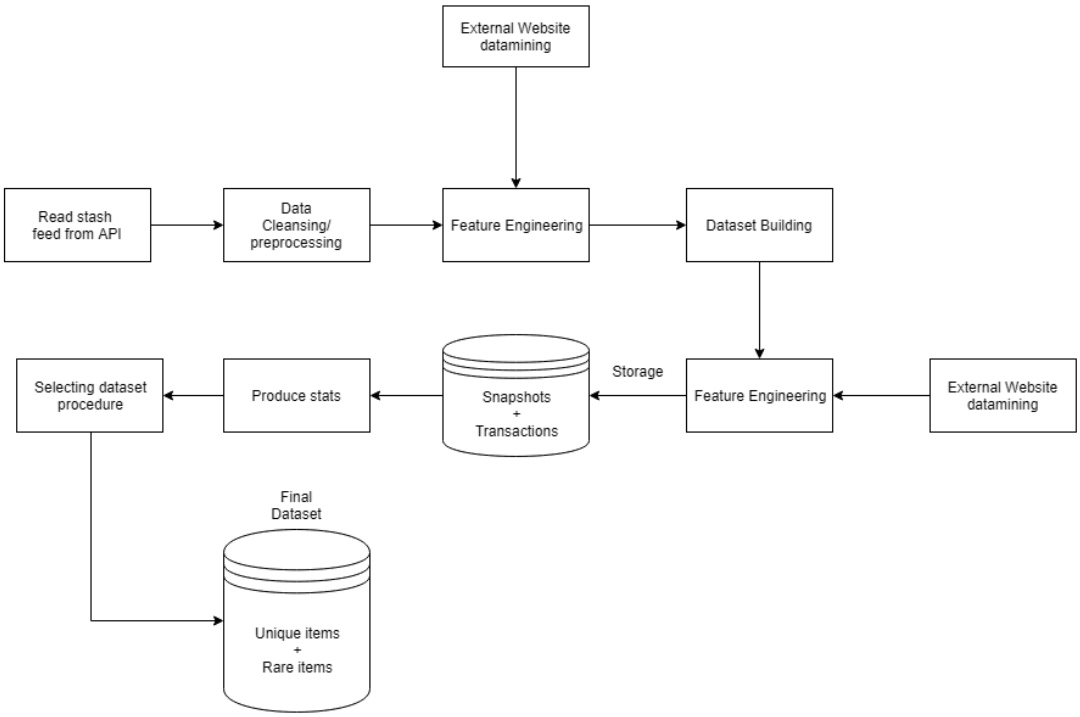


Figure 5: Dataset building system architecture

3.2 API feed

Using the game's API we downloaded data for 2 3-month seasons. That data included snapshot throughout the time of all the player's stashes and the items they have put up for sale. Each stash had a unique id and since the snapshots were historical, each new snapshot of an already seen stash indicated that an item has been removed, and thus sold, or that it had a different price.

3.3 Processing the API feed and generalization of mods

After downloading the API feed, the next step was reading through it and "cleaning" it to a more readable form, as we needed to save the data in a document based database. Another problem was the different mods of the items. It was important to generalize them and turn them to features with ranges. Mods could be either ranged - defined by a range of values - or discrete which meant either an item had that mod or it didn't. Using regex the mods were generalized.

Feature Types	Examples of item features	Generalized feature	Feature as seen in game	{feature in dataset} = {value in dataset}
Ranged	+(12-20) to maximum Life	# to maximum life	+15 to maximum Life	# to maximum life = +15
Ranged	+(20-30)% to Cold Resistance	#% to cold resistance	15% to Cold Resistance	#% to cold resistance = 15
Ranged	(8-12)% Chance to Block	#% chance to block	9% Chance to Block	#% chance to block = 9
Discrete	3% reduced movement speed	#% reduced movement speed	3% reduced movement speed	#% reduced movement speed = 1

Table 1: Feature generalization

Feature Types:

- Ranged: Ranged features can have a range of values.
- Discrete: Discrete features can only have specific values and are only seen with these values. For that reason in our dataset, if this feature appeared in the item, it had a value of 1 but if it did not appear it had a value of 0.

Steps:

1. Data mine all the different mods that could appear for unique and rare items for each different category
2. Generalize the mods using regex and convert them to features/columns for the respective dataset

In the process of reading the API we were also able to engineer additional features:

- date & time : since we were making API calls we were able to save the time and thus have a date and time for each stash feed. Since the API feed downloaded didn't have a date and time we wouldn't be able to treat unique items as a time series problems if we did not create the feature ourselves
- Exalt conversion rate and converted currency to chaos: after acquiring the daily conversion rates for all types of currencies from poe.ninja we were able to convert all currencies to chaos (the main currency used for trading in the game) as well as the daily exalt conversion rate, indicating the inflation of the market with more items in price and amount.

3.4 Dataset building and storing

The aggregation software developed run for 6 months consuming the feed from the game servers.
It concluded in more than 7 TB of uncompressed data in the form of json files.

The feed schema is highly nested so the final storage that would process it should avoid a strict relational schema like RDBMs because the complexity of inserting or updating a record as well as the time consumed would be significant.
MongoDB was chosen as the storage for the final processed feed as it is more suited to host a document schema efficiently.
Using MongoDB native characteristics - document oriented, very fast, really flexible when it comes to field addition and deletion - and its latest transaction capabilities since July 2018 – although immature – the final process of the feed and the dataset was possible [5].

To build the dataset we used 2 types of collections:

- **stashes_snapshot**: we needed a way to figure out which items were being sold and to save the last snapshot of each stash. This was achieved through the stashes_snapshot collection. When reading a stash_feed file, for each stash_id we were processing we were checking the following :
    - Have we seen it again? If not then we had to add the whole stash to the stashes_snapshot
    - If we have seen it again, we had to check each of the items whether it existed before. If not we add it to the snapshot. If an item which previously existed in the stash snapshot now doesn't exist, that means it has been sold. These items were added to the transactions collection
    - By comparing when the item was added, and when it was sold we built the feature days_in_snapshot, indicating how long it took for the item to get sold. Generally items that were correct in price took less than 3 days to get sold, more valuable items took over 3 days but less than 7 days to get sold and anything more than 7 days was more or less an outlier with only a handful of exceptions.
- **transactions:** In this collection we stored the items that were sold. Every time we did that we added the feature "days_in_snapshot" indicating how many days passed before an item was sold. If an item was taken off the stash and added back in, we removed the previous transaction and updated its price.

3.5 Dataset evaluation

Checking the correctness of generalized features was crucial for the dataset quality. Features in the dataset were cross-referenced with the available game features we built. Any features existing in the dataset and not found in the feature set or any errors that occurred during the generalization led to observation deletion.

3.6 Selecting items/categories to analyse

Making the dataset for one season for all items took a lot of time and a lot of computing power so we had to choose specific categories to analyse from the rare items, as well as specific uniques to analyse as time series data. For that reason we assumed that making the database for 7 days for all the items was representative of the whole season and after analysing each item and each category we would be able to choose which ones we would build our dataset with.

3.6.1 Rares

For the rare items, we checked how many sales have happened in those 7 days, how many features there were and their correlation values to the price the item was sold. We also picked categories with good allocation of features and price along all the observations as we didn't want to deal with a dataset skewed highly to specific prices and features. Moreover, we built histograms of the different prices per category before and after removing outliers.

3.6.2 Uniques

For the unique items, we opted for items with good standard deviation and variance of price, as well as 2 types of items : one without features that had correlation > 0.1 to perform univariate forecasting and one with at least 4 features with correlation >0.1 to perform multivariate forecasting.
The uniques we choose should also have an adequate number of transactions to support time-series analysis of 24 hours per day.

item_name	feature	corr_value	no_features	transactions	std
Inpulsa's Broken Heart Sadist Garb	ex_# to maximum life	0.14	9	4505	47.30
Inpulsa's Broken Heart Sadist Garb	ex_#% increased damage if you have shocked an enemy recently	0.32	9	4505	47.30
Inpulsa's Broken Heart Sadist Garb	ex_#% increased effect of shock	0.12	9	4505	47.30
Inpulsa's Broken Heart Sadist Garb	ex_shocked enemies you kill explode, dealing #% of their maximum life as lightning damage which cannot shock	0.16	9	4505	47.30
Inpulsa's Broken Heart Sadist Garb	linked_sockets	0.11	9	4505	47.30
Inpulsa's Broken Heart Sadist Garb	sockets_number	0.27	9	4505	47.30
Windripper Imperial Bow	Attacks per Second	0.21	7	1617	46.46
Windripper Imperial Bow	Critical Strike Chance	0.25	7	1617	46.46
Windripper Imperial Bow	Physical Damage	0.25	7	1617	46.46
Windripper Imperial Bow	ex_#% increased attack speed	0.21	7	1617	46.46
Windripper Imperial Bow	ex_#% increased critical strike chance	0.25	7	1617	46.46
Windripper Imperial Bow	sockets_number	0.23	7	1617	46.46
Loreweave Elegant Ringmail	Armour	0.35	6	1404	281.59
Loreweave Elegant Ringmail	Energy Shield	0.21	6	1404	281.59
Loreweave Elegant Ringmail	Quality	0.35	6	1404	281.59
Loreweave Elegant Ringmail	ex_#% increased elemental damage	0.10	6	1404	281.59
Loreweave Elegant Ringmail	ex_your maximum resistances are #%	0.36	6	1404	281.59
Loreweave Elegant Ringmail	linked_sockets	0.33	6	1404	281.59

Table 2: Uniques selection list (excerpt)

Since the standard deviation of the item's price was skewed heavily by outliers, we removed them after testing different threshold values for IQR and Z-Score methods. Threshold value of 70% gave the best results. If an item had a standard deviation < 5, it was not worth considering since that meant its price didn't change throughout the dataset and thus it wasn't fitted for analysis.
The number of features of an item was also a really good indicator, since it helped to decide which items to choose for multivariate time-series analysis.

Chapter 4: Uniques Dataset Analysis - Forecasting future prices

4.1 The Dataset

For this part of the problem we chose 2 unique items each for a different kind of reason:

- **Tabula Rasa Simple Robe**
  This item had no features with correlation > 0.1 or <-0.1 since the item generally has no mods. For that reason the only feature that contributed to the item's price was the daily exalt conversion rate which indicated the inflation of the market. As we can see the first days of the season, this item is highly valuable and drops drastically as the days go by or in other words as the market inflates and the ex_conv_rate(daily exalt conversion rate) increases and then the price increases again and follows the trend of the daily exalt conversion rate.

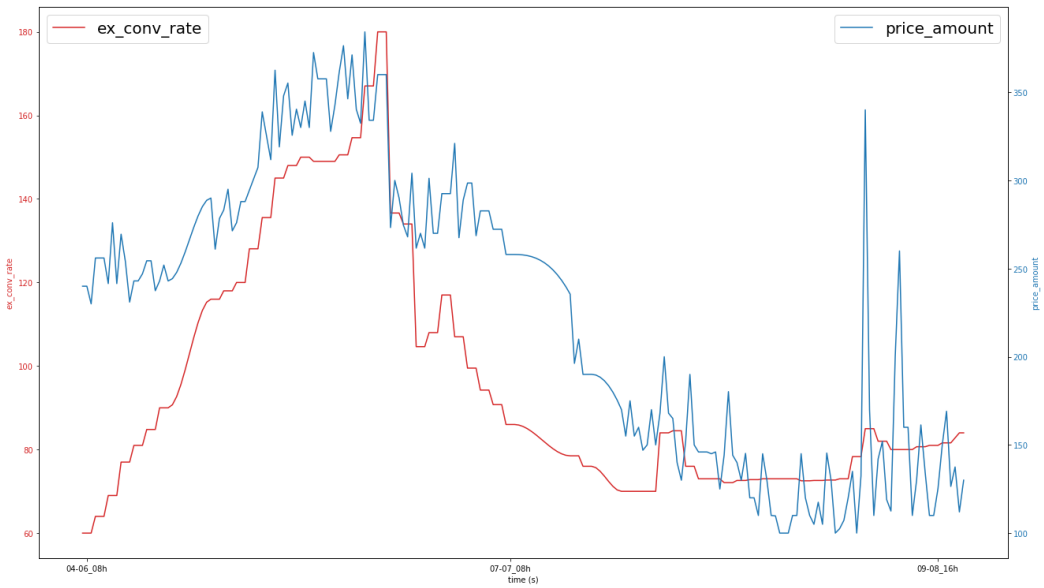


Figure 6: Tabula Rasa price (price_amount) , exchange conversion rate (Ex_conv_rate) plot

- **Windripper Imperial Bow**
  Selected to perform multivariate forecasts on it since it consisted of features with correlation_value > 0.1 and <-0.1. Also, those features didn't vary a lot in their ranges . Lastly, there is a high correlation value between the ex_conv_rate and the price of the item since the item was originally sold in exalts and it was converted to chaos, but also because as the players amass more riches during the season, the expensive items rise in price.

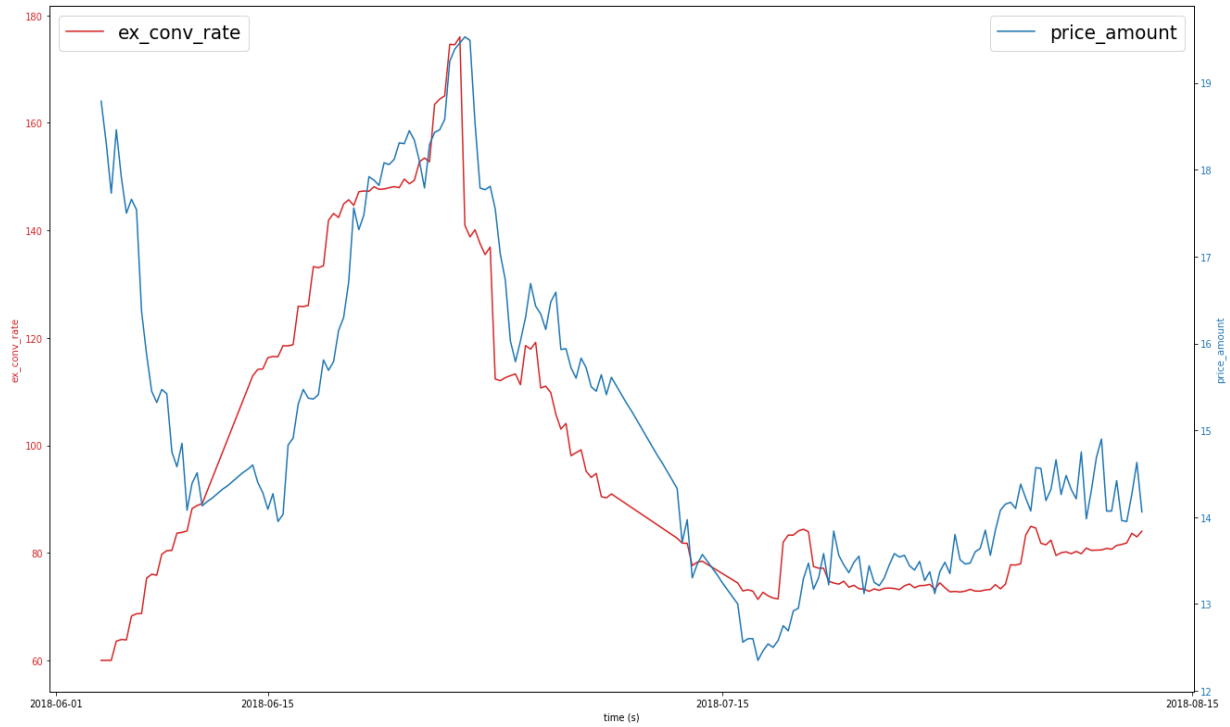


Figure 7: Windripper Imperial Bow price (price_amount) , exchange conversion rate (Ex_conv_rate) plot

For each of the above items, each observation consisted of the following features :

- Generalized features
- Date
- Days_in_snapshot
- Ex_conv_rate
- Price_amount
- Time
- Time_hours

4.2 Outliers

Outliers are observation points, too distant from the other observations. We remove them because they distort the picture of the data we obtain, using statistics and data visualization. We can use boxplots or barplots to visualize them. When our goal is to predict, our models are often improved by ignoring outliers. Here we want to forecast future prices and outliers would skew our line plots in certain time-periods.

To remove outliers from our dataset we used either the IQR method or the Z-Score method and we also removed observations that had days_in_snapshot > 5, since those observations were of low confidence.

The Z-score is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured. If the value of an observation is above a certain threshold, we consider this an outlier. The interquartile range (IQR), also called the midspread or middle 50%, or technically H-spread, is a measure of statistical dispersion, being equal to the difference between upper and lower quartiles, IQR = Q3 − Q1.

Below we see the price_amount for different high quartiles (thresholds) of IQR and Z-score respectively for the items Tabula Rasa and Windripper. As we see Z-score has almost no impact on our dataset and IQR high quartiles of 0.7 to 0.8 give the best results.

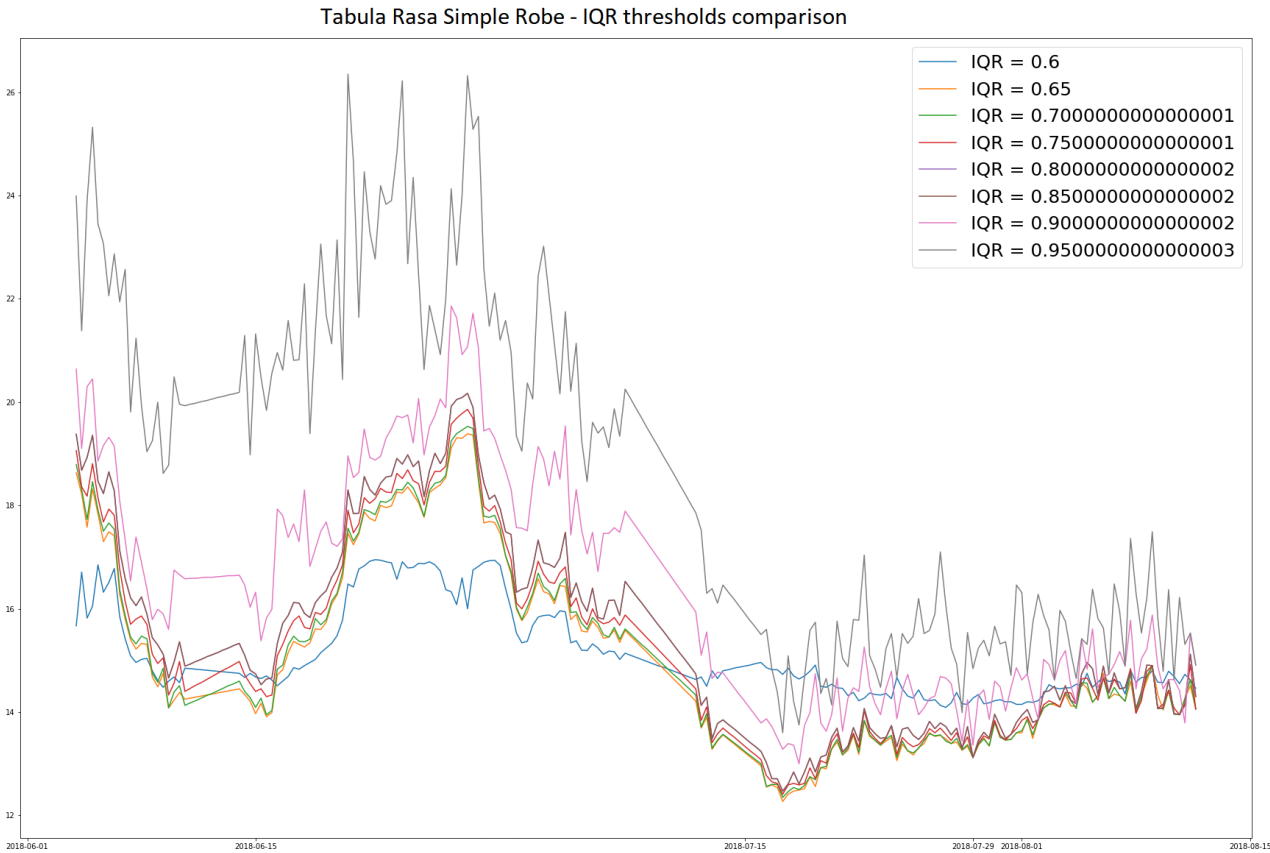


Figure 8: Tabula Rasa Simple Robe – IQR threshold comparison

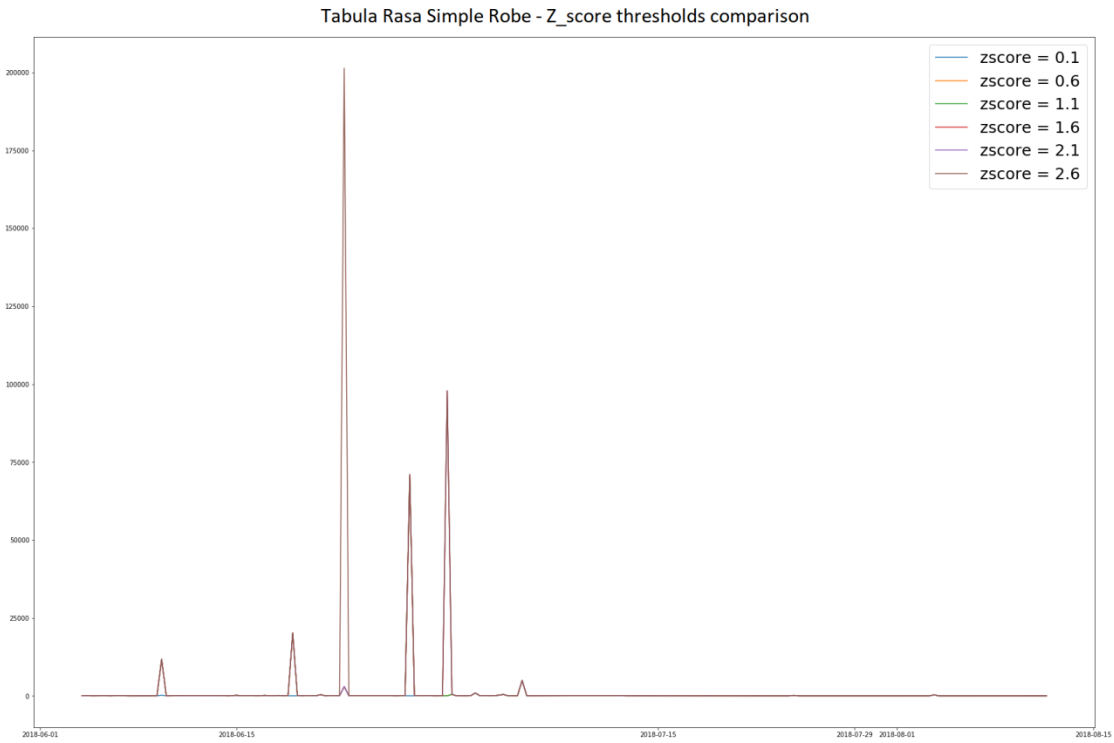


Figure 9: Tabula Rasa Simple Robe Z-score threshold comparison

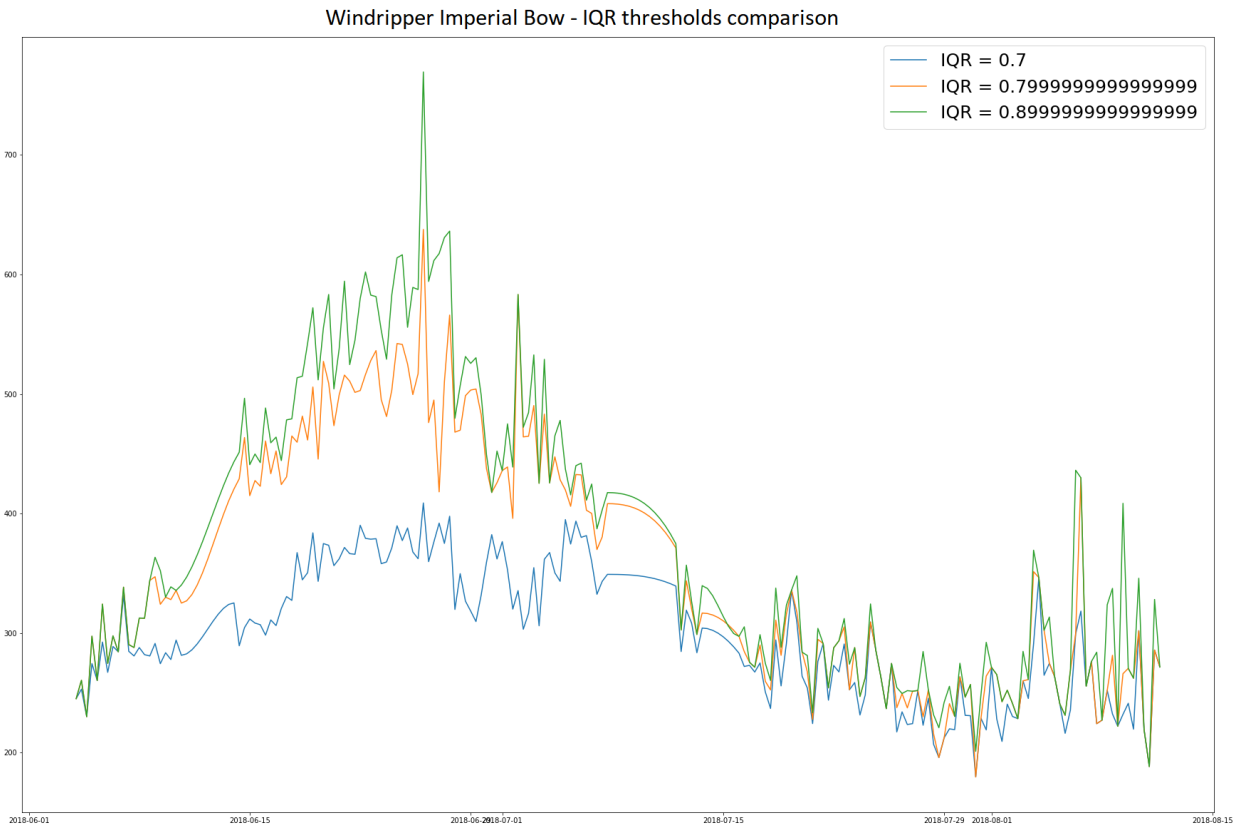


Figure 10: Windripper Imperial Bow – IQR threshold comparison

4.3 Feature Selection

A handful of different techniques were used for feature selection.

4.3.1 Missing Value Ratio

While exploring the dataset we found out that many of the features had a high percentage of missing values. For that reason, we could remove those features since they didn't contribute much information.

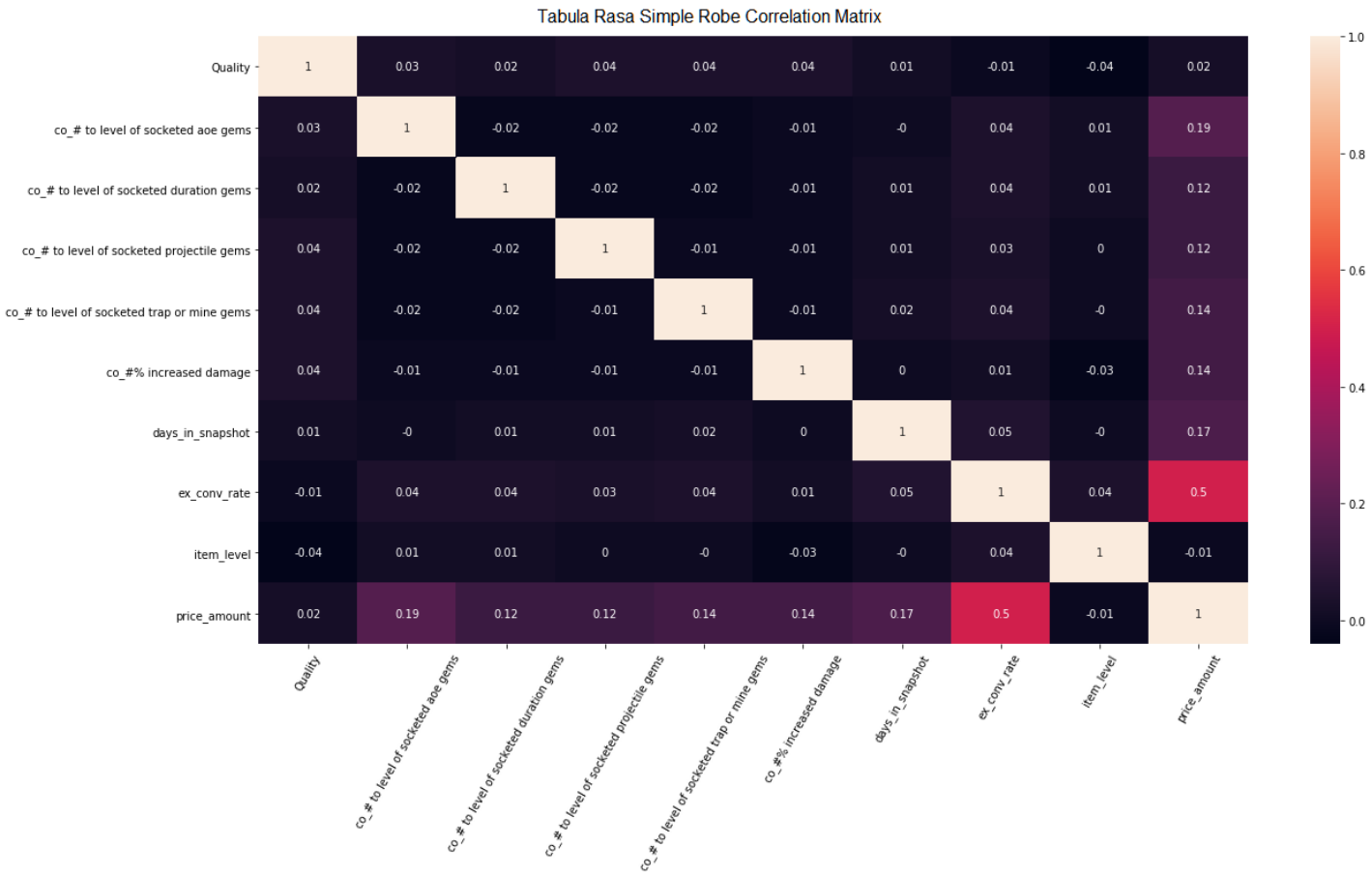
4.3.2 Low Variance Filter

Calculating the variance of each variable, we dropped the variables that had too little to no variance at all, since those variables would not affect the target variable which is the price_amount.

4.3.3 Correlations

Using pandas' 'corr' method to find the cross-correlations of features, we decided to remove features with correlation_value lower than 0.2 and higher than -0.2 to 0. The methods that were used were either Spearman or Kendall which are generally computed on ranks and so depict monotonic relationships, while Pearson was not used since it is on true values and depicts linear relationships [6]. This method wasn't preferred compared to the others.

Figure 11: Tabula Rasa Simple Robe Correlation Matrix



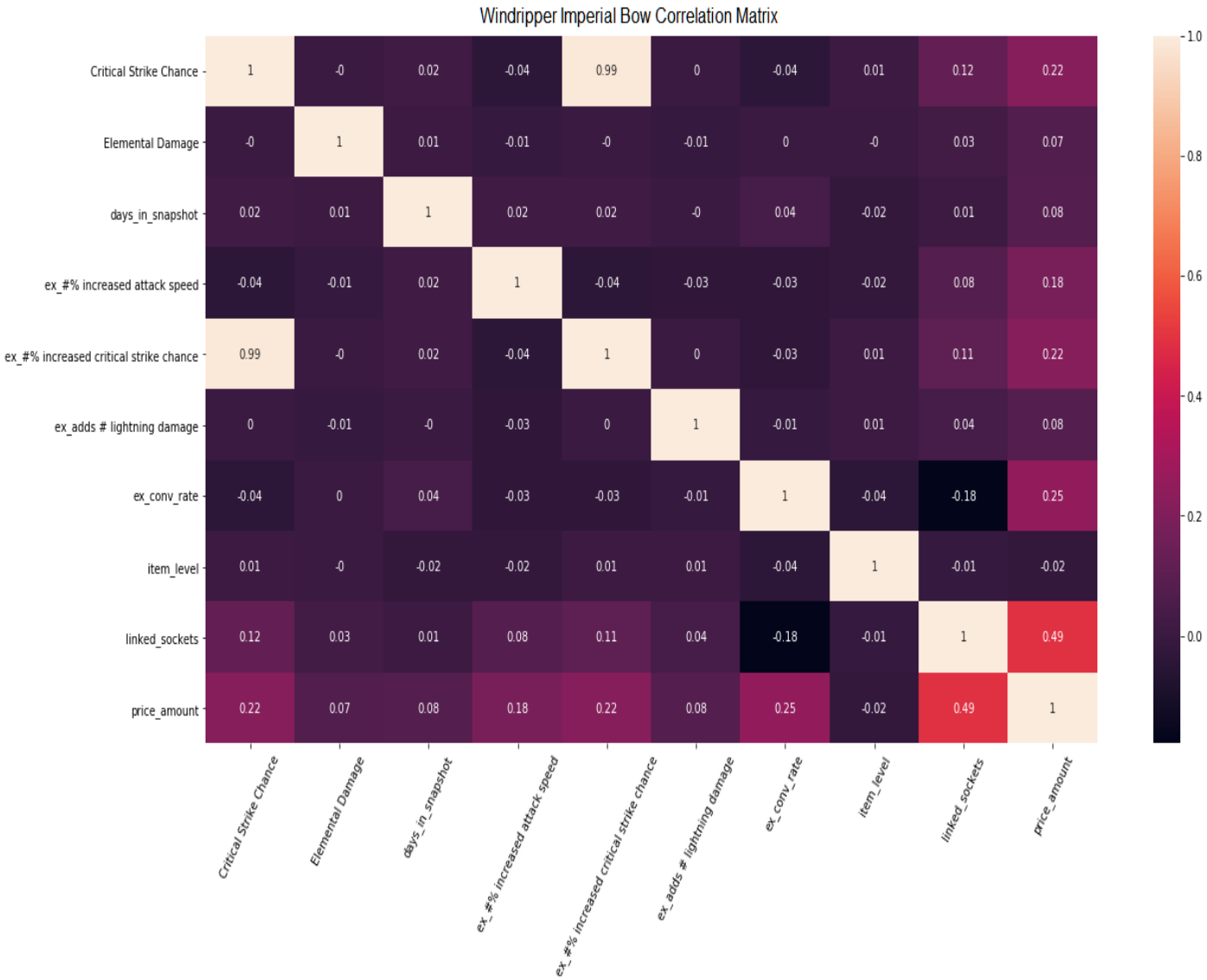


Figure 12: Windripper Imperial Bow Correlation Matrix

4.3.4 Random Forest

Random Forest came packaged with in-built feature importance, so by using it we could find out which features contributed to the target variable.

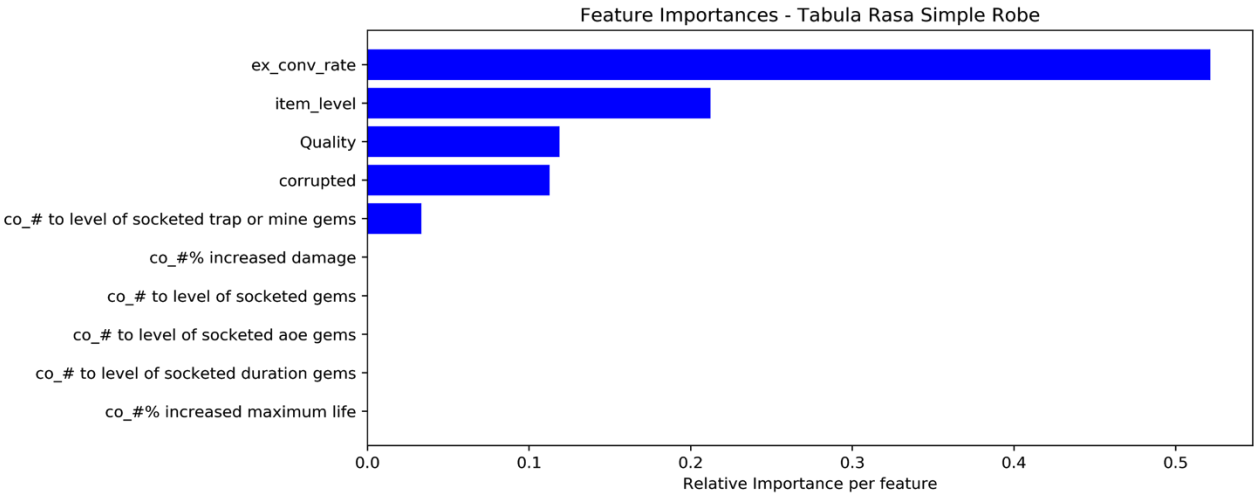


Figure 13: Feature importances – Tabula Rase Simple Rose

There are no features of relative importance except the ex_conv_rate. For that reason we can perform univariate time-series analysis of the item without worrying that we lost information from important features

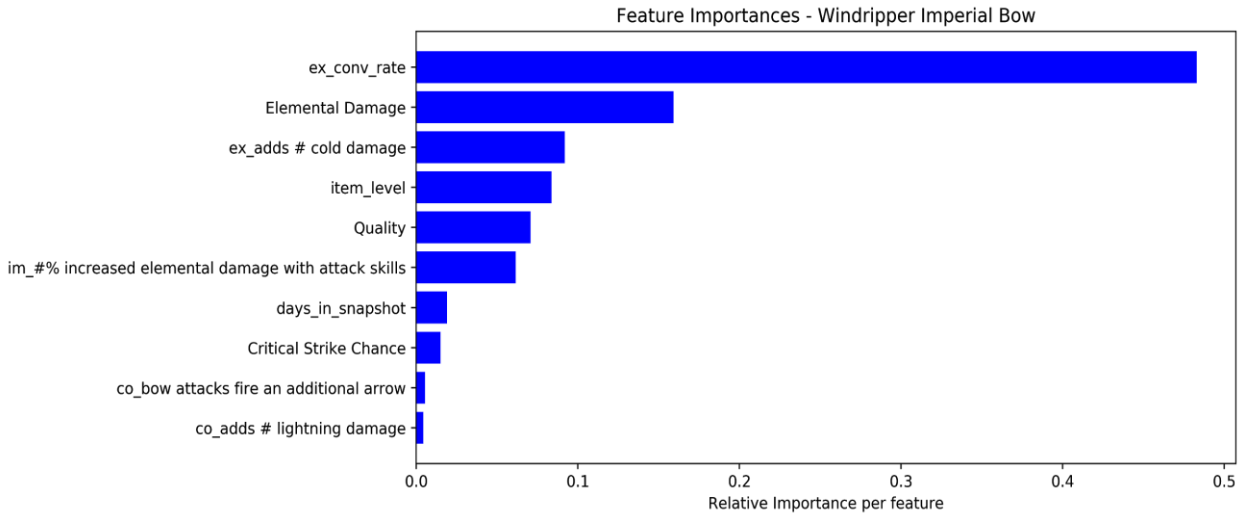


Figure 14: Feature importances – Windripper Imperial Bow

4.3.5 Backward Feature Elimination

- We first take all the n variables present in our dataset and train the model using them
- We then calculate the performance of the model
- Now, we compute the performance of the model after eliminating each variable (n times), i.e., we drop one variable every time and train the model on the remaining n-1 variables
- We identify the variable whose removal has produced the smallest (or no) change in the performance of the model, and then drop that variable
- Repeat this process until no variable can be dropped

To achieve this we used rfe from sklearn.feature_selection library with many algorithms included but not limited to Linear Regression, Logistic Regression, Random Forest, Decision Trees etc. From the above the most consistent results were from Decision Trees.

4.3.6 Forward Feature Elimination

It's the opposite process to Backward Feature Elimination. Instead of eliminating features, we try to find the best features which improve the performance of the model. To achieve this we used f_regression from sklearn.feature_selection library

4.4 Convert irregular time series to regular

Unevenly (or unequally or irregularly) spaced time series is a sequence of observation time and value pairs (tn, Xn) with strictly increasing observation times [7]. As opposed to equally spaced time series, the spacing of observation times is not constant. A common approach to analysing unevenly spaced time series is to transform the data into equally spaced observations using some form of interpolation - most often linear - and then to apply existing methods for equally spaced data.

Initially our data did not have the characteristics of time-series data, that is the observations were not a sequence of equal time segments.

4.4.1 Adding a Time index

Initially our data did not have the characteristics of time-series data, meaning the observations were not a sequence of equal time segments. For that reason our first step was to use the pd.to_datetime method of pandas and by combining our date and time features get a dateTime index. Using that our next step was to break the data down to equal segments by setting up a frequency and interpolating the observations.

4.4.2 Resampling

	Quality	co_# to level of socketed aoe gems	co_# to level of socketed duration gems	co_# to level of socketed projectile gems	co_# to level of socketed trap or mine gems	co_#% increased damage	days_in_snapshot	ex_conv_rate	item_level	price_amount
date_time										
2018-06-04 00:17:00	0.0	0	0	0	0	0.0	0.0	60.00	71.0	18.00000
2018-06-04 00:25:00	0.0	0	0	0	0	0.0	0.0	60.00	75.0	20.00000

Table 3: Adding a time index - Uniques

It can be observed that the date_time index of each observation is in a different time-period or in other words in a different frequency. To turn the dataset into a time-series one we also need to increase the frequency of our samples, such as from minutes to seconds, or decrease it such as from days to months. These processes are called resample. In both cases, data must be invented. Here we chose to downsample from minutes to hours.

4.4.3 Interpolation

In this part we had to solve two problems of irregular data: empty segments that occured after resampling by not having any observations during these time-periods and segments with multiple values that we had to break down to one value. For that reason we had to interpolate our data. Interpolation is a method of constructing new data points within the range of a discrete set of known data points. The processes we followed were filling and flattening.

	Quality	co_# to level of socketed aoe gems	co_# to level of socketed duration gems	co_# to level of socketed projectile gems	co_# to level of socketed trap or mine gems	co_#% increased damage	days_in_snapshot	ex_conv_rate	item_level	price_amount
date_time										
2018-06-04 00:01:00	0.0	0	0	0	0	0.0	0.0	60.0	77.0	19.0
2018-06-04 00:07:00	0.0	0	0	0	0	0.0	0.0	60.0	74.0	22.0
2018-06-04 00:07:00	20.0	0	0	0	0	0.0	0.0	60.0	76.0	19.0
2018-06-04 00:08:00	0.0	0	0	0	0	0.0	0.0	60.0	27.0	20.0
2018-06-04 00:09:00	0.0	0	0	0	0	0.0	0.0	60.0	80.0	20.0

Table 4: Resampling - Uniques

We can inspect that there are multiple observations for the same date_time index and we need to interpolate to a certain frequency and flatten the multiple values.

	Quality	co_# to level of socketed aoe gems	co_# to level of socketed duration gems	co_# to level of socketed projectile gems	co_# to level of socketed trap or mine gems	co_#% increased damage	days_in_snapshot	ex_conv_rate	item_level	price_amount
date_time										
2018-07-13 23:59:00	13.0	0	0	0	1	0.0	3.0	76.00	80.0	14.0000
2018-07-16 00:00:00	20.0	0	0	0	0	0.0	4.0	70.00	76.0	13.0000
2018-07-16 00:00:00	0.0	0	0	0	0	0.0	1.0	70.00	82.0	13.0000

Table 5: Missing values - Uniques

Missing days between 13 of July and 16 of July. Our dataset needs to be filled between these days with default values following a distribution, to be turned into a regular time-series dataset.

Filling

We filled empty segments with either backfilling or forward filling, linearly which followed a linear distribution between the last and first seen values between the missing segments or by using pchip (Piecewise Cubic Hermite Interpolating Polynomial) which is typically used for interpolation of numeric data to obtain a smooth continuous function. We decided to use the pchip method since it was shape preserving and visually pleasing. The key idea was to determine the slopes so that the interpolant does not oscillate too much [10].

Flattening

To have one observation per time unit we had to flatten our data. Flattening observations referring to the same time unit is performed using the mean, the median or the mode of all the values in a segment or with a custom function. For most of the different features we used the median but for the target variable, price_amount, we used the mean since in contrast to the other features, price_amount values varied a lot more. The difference between the two can be seen in the plot below.

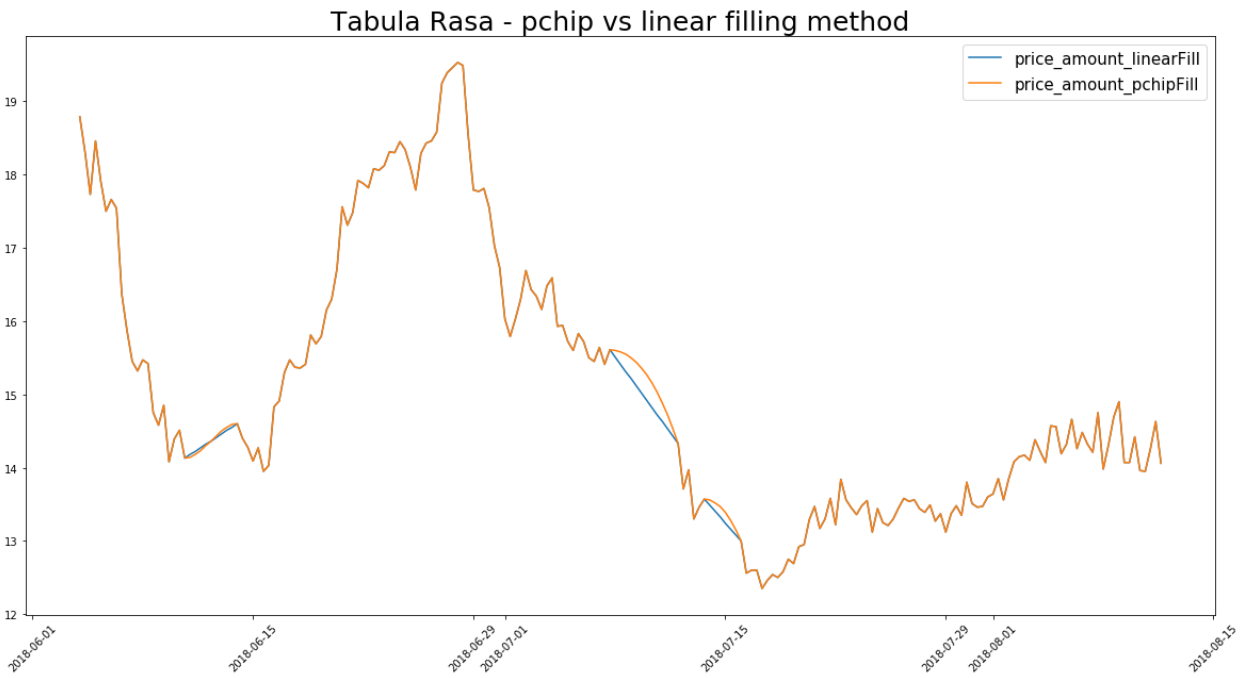


Figure 15 : Tabula Rasa – pchip vs linear filling method

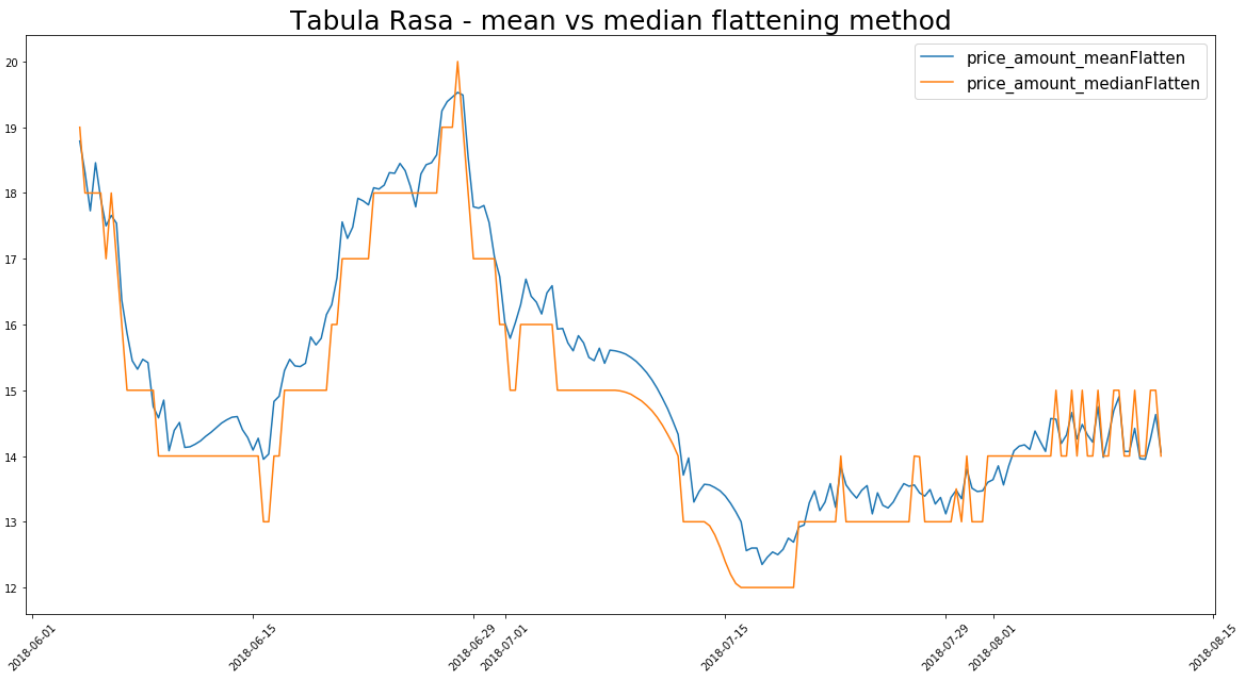


Figure 16 : Tabula Rasa – mean vs median filling method

4.5 Time Series Analysis

For Time Series Forecasting using the Long Short-Term Memory(LSTM) Network we try to provide momentum indicators of the prices of unique items. Before we dive into forecasting and LSTM, there are some important parameters and arguments that need to be explained.

Input Timesteps(Lag)

Traditional neural networks take in a stand-alone data vector each time and have no concept of memory on data. LSTM networks keep a context of memory within their pipeline and thus become powerful at tackling sequential and temporal problems without the issue of the vanishing gradient affecting their performance. The "memory" our network keeps is the lag. For each forecast, we feed a lag of sequential information to our network, for it to learn from.

Forecasting Sequence

If the lag is the input, forecasting sequence is the output. It is the time-period window our model will try to forecast.

Train / Test Split

Just like in ANN's, using LSTM we have to split our dataset to a train set and a test set. There are a few differences though:

- Splitting cannot be on a shuffled dataset. We have to decide beforehand how many time-periods we will use as training.
- Lag cannot be higher than the train dataset.
- Train and test splits are sequential, just like before the split and train split precedes the test split.

Normalization

Since the starting price of an item is different for each 3 month season, we decided to take each n-sided window of training /testing data and normalize it to reflect percentage changes from the start of that window.

Epochs

An epoch is simply one forward pass and one backward pass of all the training examples. Training for more epochs makes our model better but also more prone to overfitting.

Batch Size

Batch size is the number of training examples in one forward/backward pass. The higher the batch size, the more memory space we need. It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize.

Dropout

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.

During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer look like and be treated like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different “view” of the configured layer. For example, a Dropout layer with a rate of 0.2 has a 20% chance to drop each neuron.

Loss Functions

A loss function is used to optimize the parameter values in a neural network model. Loss functions map a set of parameter values for the network onto a scalar value that indicates how well those parameter accomplish the task the network is intended to do. It is essentially a mathematical way of measuring how wrong our predictions are. Loss is that measure.

Optimizer

During the training process, we change the parameters of our model to try and minimize the loss function and make better, more accurate predictions/forecasts. Optimizers tie together the loss function and the model parameters by updating the model in response to the output of the loss function [11].

Activation Function

The activation function of a node defines the output of that node, or "neuron," given an input or set of inputs. This output is then used as input for the next node and so on until a desired solution to the original problem is found. If we do not apply an Activation function then the output signal would simply be a simple linear function.

HyperParameters

A hyperparameter is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training. Different model training algorithms require different hyperparameters . These hyperparameters are going to be optimized later on using a method called Grid Search. Our hyperparameters here were:

- Input timesteps
- Forecasted sequence
- Train / test split
- Epochs
- IQR high quantile threshold
- Loss
- Optimizer
- Learning rate for SGD
- Neurons of different LSTM and Dense layers
- Activation Function
- Dropout rate

4.6 Univariate Unistep vs Univariate Multistep Time-Series Analysis using LSTM

As previously mentioned, we used the item “Tabula Rasa” because it did not consist of important features that could affect its price except the daily conversion rate of exalts. On it, we perform 2 types of analyses, unistep and multistep. Unistep, or point by point, is the prediction of a single time-period ahead of time, plotting this prediction and then taking the next window along with the full testing data and predicting the next point along once again.

The multistep prediction will be done in two parts. The first will consist of a full sequence prediction, by initializing a training window and training our model on it. The model then predicts the next point and we shift the window by one time-period to the right, just like the point by point method. The difference is we then predict using the data that we predicted in the prior prediction. In the second prediction we have one predicted data point, in the third 2 predicted data points and so forth [9]. After we predict points equal to the input sequence, our next prediction consists of only predicted data points. This allows us to use the model to forecast many time steps ahead, but as it is predicting on predictions which can then in turn be based on predictions this will result in increased error rate of the predictions the further ahead we predict.

The second part is a multi-sequence prediction. This is a blend of the full sequence prediction in the sense that it still initializes the testing window with test data, predicts the next point over that and makes a new window with the next point. However, once it reaches a point where the input window is made up fully of past predictions it stops, shifts forward one full window length, resets the window with the true test data, and starts the process again. In essence this gives multiple trend-like predictions over the test data in order to analyse how well the model can pick up future momentum trends [17].

All the processes were created using the Tensorflow and Keras Libraries as well sklearns multiple libraries[12].

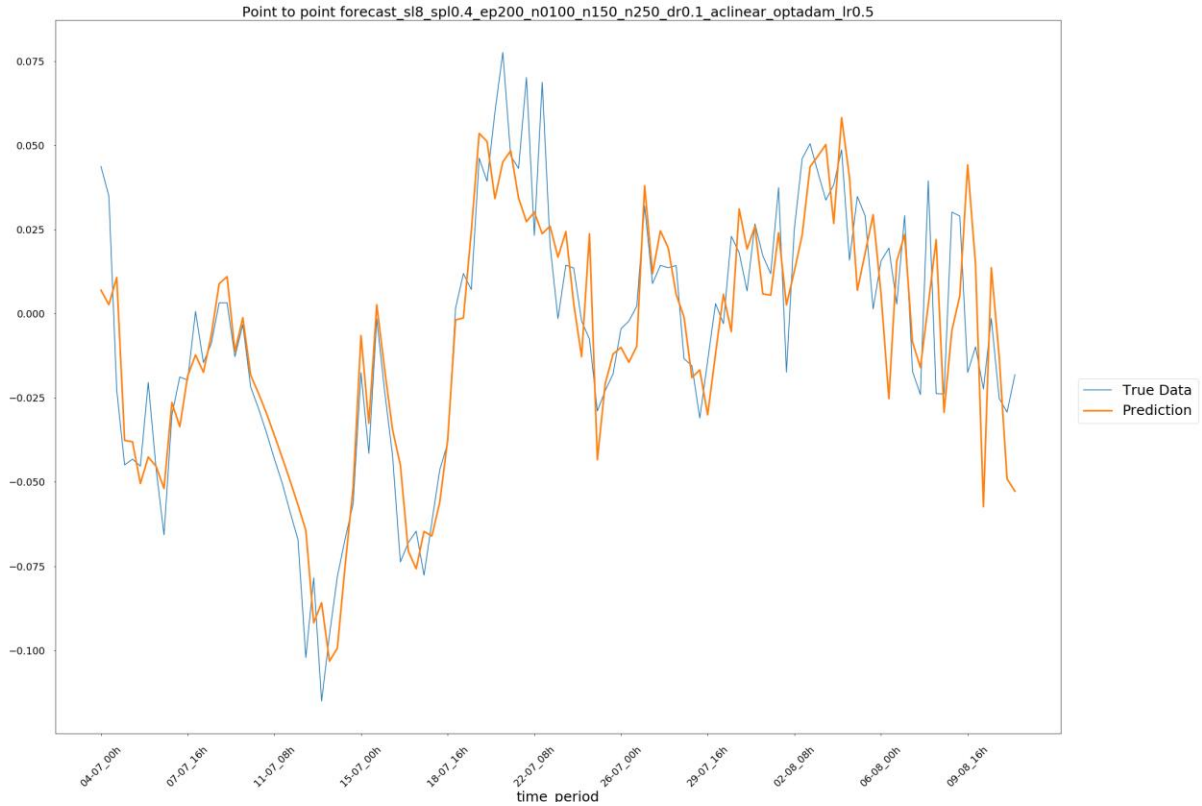
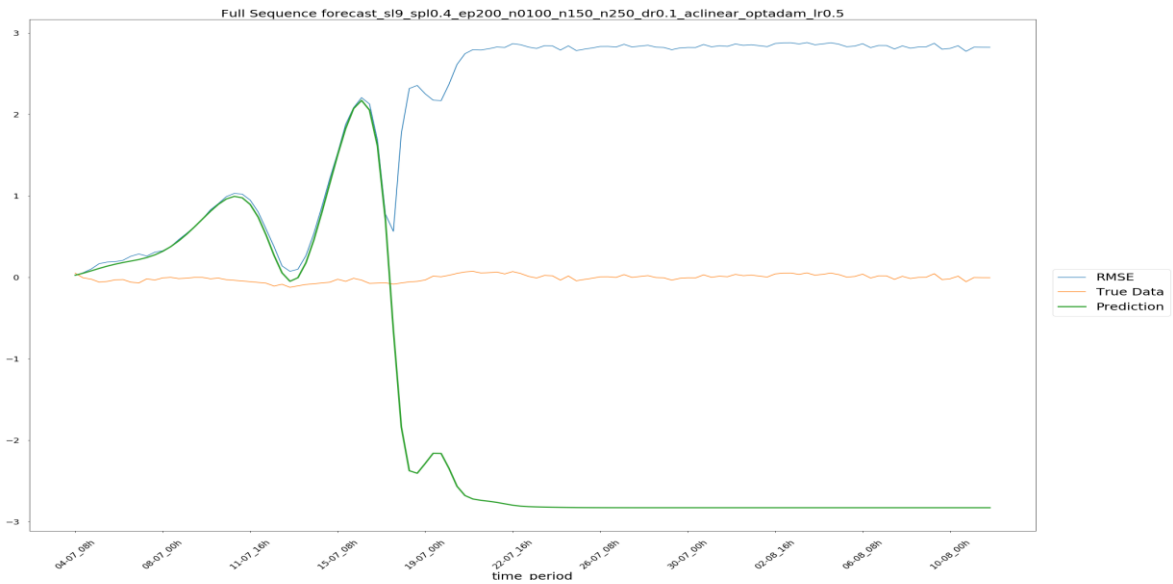


Figure 17 : Point to point forecast sequence length 9 frequency periods, 200 epochs, split 0.4, dropout 0.10, network neurons 100,150,250,activation function linear, optimizer adam, learning rate 0.5

With RMSE = 0.01855 the point to point forecasting is a pretty accurate representation of what is going to happen in the following time-period.

In contrast, our full sequence forecasting below tries to predict what will happen in the first time-periods but afterwards, building solely on other predictions, fails to forecast and the RMSE increases rAPIdly.

Figure 18: Full sequence forecast sequence length 9 frequency periods, 200 epochs, split 0.4, dropout 0.1, network neurons 100,150,250,activation function linear, optimizer Adam, learning rate 0.5



Lastly, we can see the Multistep forecast plot. With a split of 0.4 (or roughly 30 days) it can predict trends (and sometimes the amplitude of trends) for a good majority of the time-periods. While it is not perfect, it is a good indication of the usefulness of the model.

Figure 19: Multisteps forecast sequence length 6 frequency periods, 200 epochs, split 0.4, dropout 0.2, network neurons 100,150,activation function linear, optimizer Adam, learning rate 0.5

Grid Search

To improve our results we have to tune our hyperparameters. For that reason we used Grid search. Grid search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions. Grid search builds a model for every combination of hyperparameters specified and evaluates each model. For point to point predictions, we calculated the RMSE of predictions for each model and picked the one with the lowest RMSE, whereas for full sequence predictions all our results were bad and building on multiple errors didn't give a good representation of future prices.

The parameters we optimized and the ranges we optimized them on can be seen below. It is important to note that using a sgd optimizer gave far worse results than adam for any learning rate we initialized it on and for that reason we stopped the test early since it would double the size of our permutations.

```
grid = ParameterGrid({"sequence_length": [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
                      "train_test_split": [0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8],
                      "epochs": [100,200],
                     "optimizer":['adam','sgd'],
                     "learning_rate":[0.01,0.05,0.1,0.2],
                     "layer0_neurons":[100,200],
                     "layer1_neurons":[50,100],
                     "dropout_rate":[0.1,0.2,0.3,0.4],
                     "layer3_activation_function":['linear','sigmoid']
                     })
```

sequence_length	train_test_split	epochs	neurons0	neurons1	dropout_rate	activation_function	optimizer	learning_rate	rmse	frequency
8	0.4	200	100	50	0.2	linear	adam	0.5	0.013552	12H
9	0.4	200	100	50	0.3	linear	adam	0.5	0.013576	12H
9	0.4	200	100	50	0.1	linear	adam	0.5	0.013616	12H
8	0.7	200	100	50	0.3	linear	adam	0.5	0.013646	12H
9	0.4	300	100	50	0.2	linear	adam	0.5	0.013676	12H
13	0.4	200	100	50	0.1	linear	adam	0.5	0.322730	24H
13	0.4	300	100	50	0.2	linear	adam	0.5	0.329392	24H
13	0.4	300	100	50	0.3	linear	adam	0.5	0.348312	24H
13	0.4	300	100	50	0.1	linear	adam	0.5	0.350861	24H

Table 6: Best and worst RMSE results after GridSearch optimization - Uniques

For multistep forecasts we could not compare different sequence lengths, and depending on the train and test split each forecasted trend was on a different time-window and had a different length. For that reason the metrics rmse, mae or mape were not representative of the performance of the model and its ability to forecast trends. The decision for the best model or models was done purely by observing each different result plot.

Generally for splits higher than 0.5, the model overfitted and the trends did not have positive or negative peaks. Raising the dropout rate or adding more dropout layers did not change that.

For sequence length higher than 10, the predictions had a big error as each next prediction-point was based on more predictions-points.

One sweet spot for 8 hours and 12 hours resampling frequency was the sequence length of 6. Our model could almost always predict if the price of the item would rise , fall or sudden changes as well as the amplitude of those trends.

One parameter that was also really important was the activation function. Since we normalized our variables and predicted change percentages, any activation function that could not produce negative results could not be used. From tanh and linear activation functions, the linear gave the best results.

Testing optimizers, sgd could not be used, even after testing from a learning rate of 0.01 to 1, since it did not produce any results. Adam was the only one that gave good results.

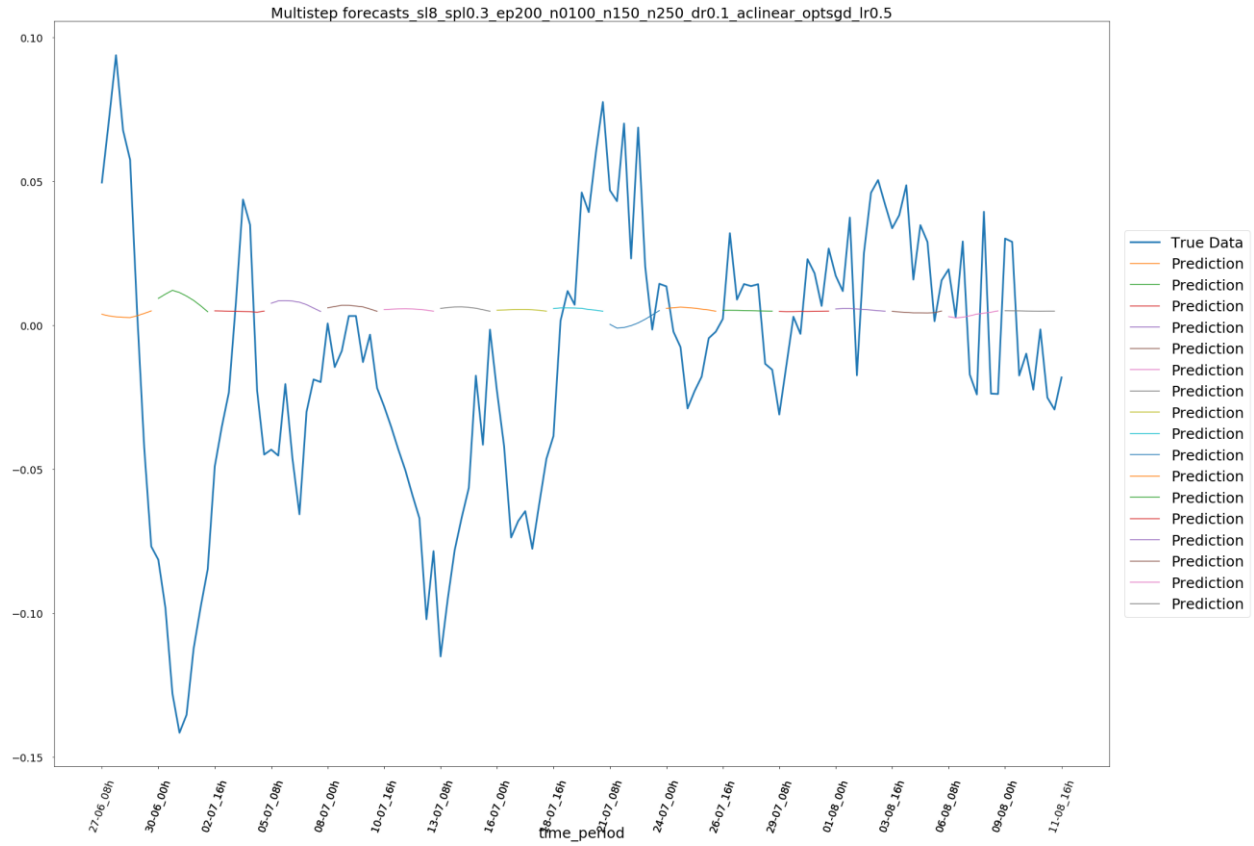


Figure 20: Multistep forecasts sequence length 8 frequency periods, 200 epochs, split 0.3, dropout 0.1, network neurons 100,150,250,activation function linear, optimizer sgd,learning rate 0.5

Lastly epochs and the neurons of each layer were really important to be kept in low numbers, 200 epochs maximum and 100 and 50 neurons in first and second layer, since the more epochs or more neurons a layer had the easier the model overfitted. That means that our forecasts could not have any peaks be it negative or positive.

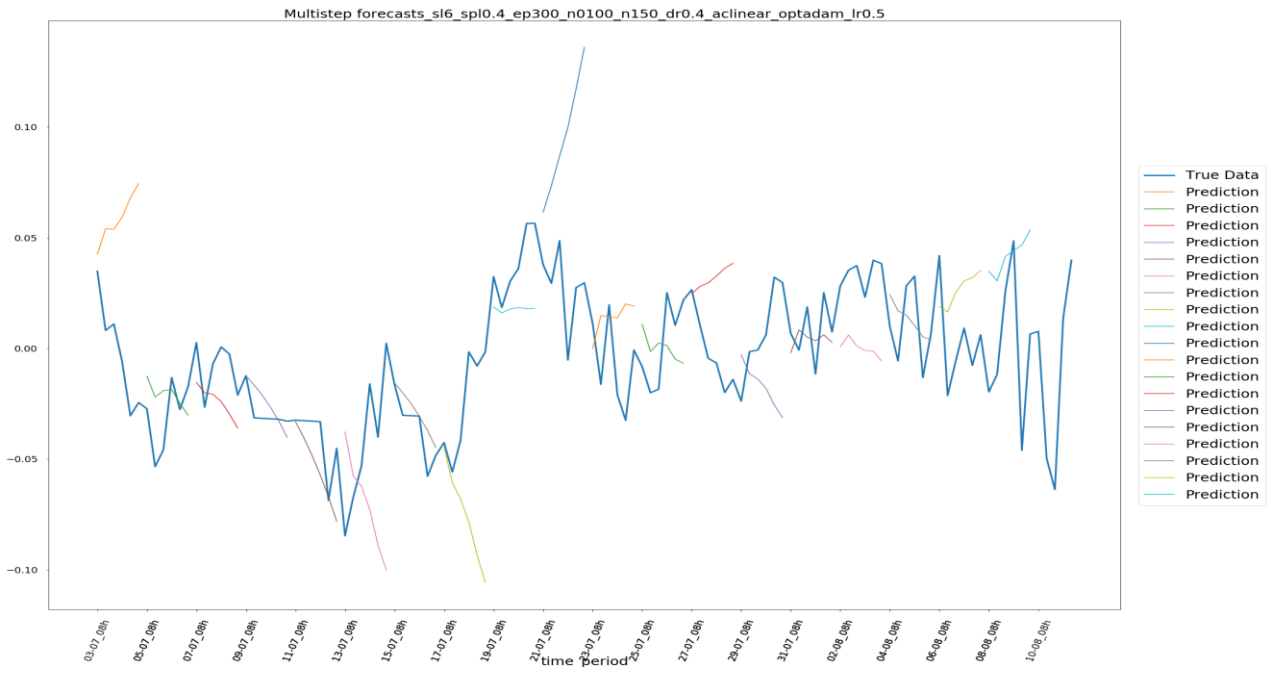


Figure 21: Multistep forecasts sequence length 6 frequency periods, 200 epochs, split 0.4, dropout 0.2, network neurons 100,150,activation function linear, optimizer Adam, learning rate 0.5

One parameter that played a big role was the rate of the dropout layer as well as how many dropout layers there were in our neural network. Using 2 dropout layers of 0.1 rate and 0.2 rate gave us the best results. It lowered the amplitude of the forecasted trends and it enabled the model to predict more price troughs.

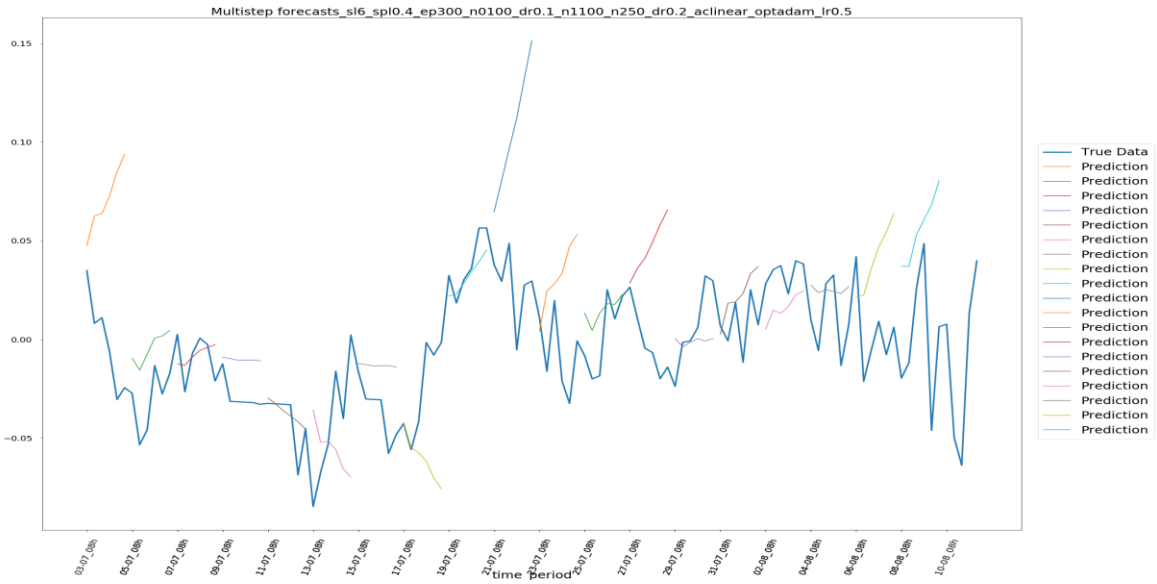


Figure 22: Multistep forecasts sequence length 6 frequency periods, 300 epochs, split 0.4, network 100 dropout 0.1,100,250 dr 0.2,activation function linear, optimizer Adam, learning rate 0.5

4.7 Multivariate Unistep vs Multivariate Multistep Time-Series Analysis using LSTM

In multivariate time-series analysis, we had to choose an item with features that followed certain rules:

- They had to have a correlation of 0.2 or higher
- They had to appear in a big percentage of observations.

Following the second rule any feature which followed the format “co_” had to be removed as seen below.

Feature name	Appearance percentage
co_#% increased movement speed	0
co_bow attacks fire an additional arrow	0.112734
co_adds # fire damage	0.371084
co_#% increased critical strike chance	0.422754
co_socketed gems are supported by level # onslaught	0.432148
co_#% chance to gain a frenzy charge on kill	0.44624
co_socketed gems are supported by level # faster projectiles	0.44624
co_#% increased physical damage	0.450937
co_socketed gems are supported by level # blind	0.474423
co_adds # cold damage	0.483818
co_adds # lightning damage	0.526093
co_#% increased attack speed	0.540185
co_adds # physical damage	0.624736
co_#% increased elemental damage with attack skills	3.231716

Table 7: ‘co_’ features and their appearance percentage - Uniques

Following the univariate example, we first made point-to-point predictions and then tried to forecast multiple trends. This time we experimented with different features to showcase how removing features could affect the overall information gain, which we represented by calculating the RMSE value [14].

We run the same model using 5, 6, 7, 8 and 9 features.
The optimal neural network consisted of 3 LSTM layers with 100, 50 and 50 neurons respectively as well as 1 dropout layer set on 0.3 dropout rate. The last layer was a dense layer with a linear activation function.
The hyperparameters and the ranges we experimented with, as well as the results, can be seen below.

```
{"sequence_length": [6, 7, 8, 9, 10, 11, 12, 13, 14],
 "train_test_split": [0.3,0.4,0.5],
 "epochs": [100,200,300],
"optimizer":['adam'],
"layer_0.neurons":[100,200,300],
"layer_1.neurons":[50,100,150],
"layer_2.neurons":[50,100,150],
"layer_3.rate":[0.1, 0.2, 0.3],
"layer_4.activation_function":['linear']
})
```

Table 8: Results grid after fitting models produced using hyperparameters – Uniques

dropout_rate	epochs	learning_rate	neurons0	neurons1	neurons2	no_of_features	optimizer	rmse	sequence_length	train_test_split
0.3	100	0.5	100	50	50	9	adam	0.111247	6	0.4
0.2	100	0.5	100	50	50	9	adam	0.111601	6	0.4
0.2	200	0.5	100	50	50	5	adam	0.112929	6	0.3
				. . .						
0.3	300	0.5	100	50	50	5	adam	0.206421	10	0.3
0.2	300	0.5	100	50	50	6	adam	0.208711	14	0.3
0.3	300	0.5	100	50	50	6	adam	0.217636	14	0.3
0.2	300	0.5	100	50	50	5	adam	0.235387	11	0.3
0.1	300	0.5	100	50	50	5	adam	0.293465	11	0.3

As seen above, the results we got were better (smaller RMSE) when more features were used, since there was less information loss. Also shorter sequence length resulted in smaller rmse and fewer epochs resulted in less overfitting and therefore a better rmse value.

Figure 23 : Point to point forecast sequence length 6 frequency periods, 100 epochs, split 0.4, dropout 0.3, network neurons 100,150,250,activation function linear, optimizer adam, learning rate 0.5

The same can be observed for multistep forecasting [15]. When the train/test split is kept at 0.4 it seems to be the optimal value for predicting trends at low sequence lengths. As for epochs, with 100 or less epochs, the model can fairly accurately forecast trends. Also, a dropout rate of 0.2 or 0.3 allows the model to predict sharper troughs. Neurons didn't play that much of a role but at the higher values we predicted less peaks.

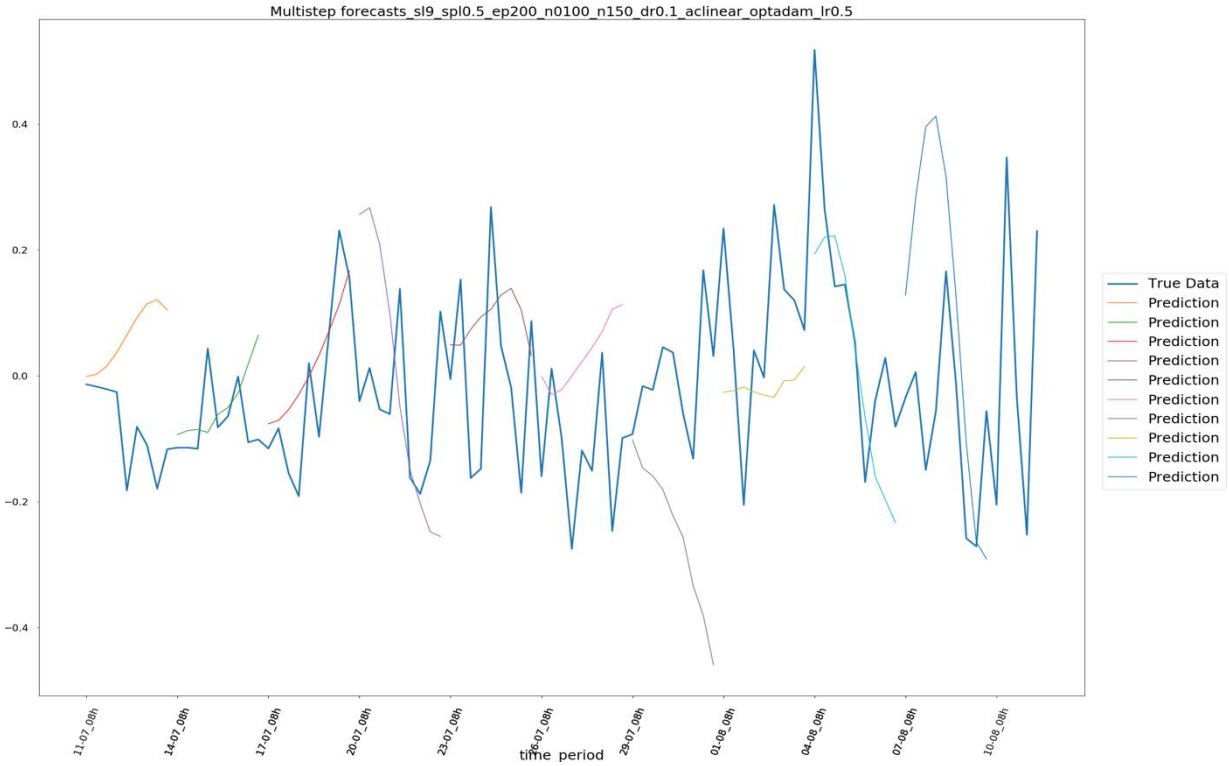


Figure 24: Multistep forecasts sequence length 9 frequency periods, 200 epochs,split 0.5, network 100,250 dr 0.1,activation function linear,optimizer adam,learning rate 0.5

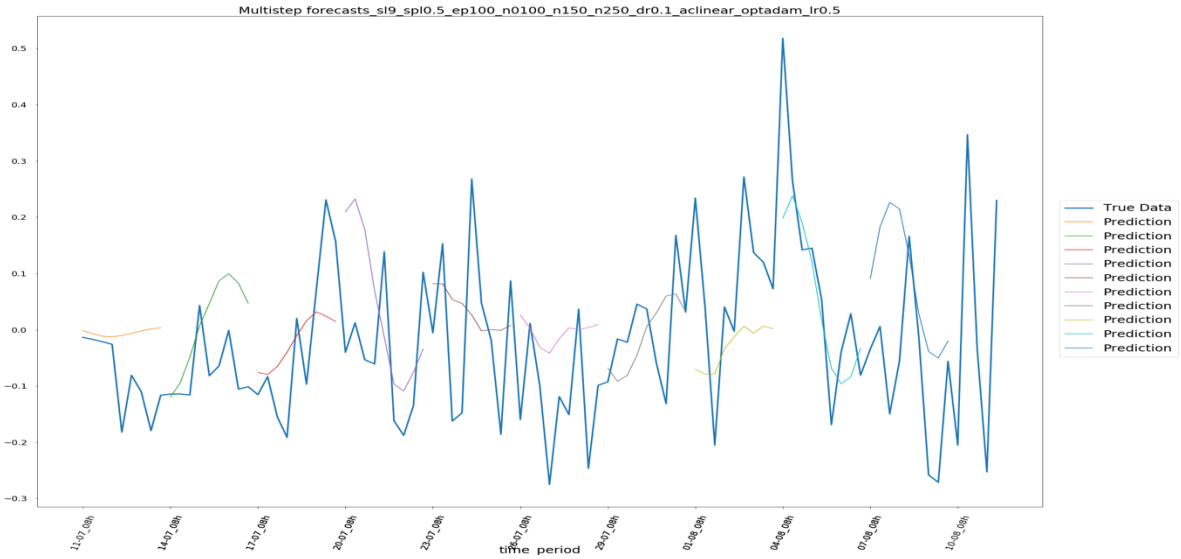


Figure 25: Multistep forecasts sequence length 9 frequency periods, 100 epochs,split 0.5, network 100,250 dr 0.1,activation function linear,optimizer adam,learning rate 0.5

Overall, as seen also in the plots above for 5 and 9 features respectively, having more features, allowed our model to forecast better trends and their amplitudes [18].

4.8 Conclusion

While the process of forecasting seemed difficult at first, we managed to get really good point to point forecasts for unique items as well as good multistep forecasts. Although the evaluation process of multistep forecasting was not optimal, we could find specific hyperparameters which worked best by observing our plots and draw conclusion from those.

Having complete observations for each unique item in the game, we could build models for each one and then automate the process of forecasting their prices. One idea for picking out the best model after hyperparameter optimization would be to evaluate each model according to a certain purpose e.g. predicting market returns to trade with and using as metric the bottomline profit and loss (PnL).

Chapter 5: Rares Dataset Analysis - Price prediction

5.1 General

Observing the rare items at first glance, the problem was the variable number of features for each observation. Considering that each rare item, which meant each observation, could not have more than 12 features and the number of observations was around 580.000 (before removing outliers), it became apparent that the number of different combinations our 96+ features made, far exceeded our observations. Some combinations did not appear at all and some other appeared only once.

We tried to tackle the problem a number of different ways, either as a regression problem by using different algorithms or different neural networks or by doing the same thing but as a classification problem by bucketing the different prices of items.

5.2 The Dataset

For the rare items, we chose to analyse the "Body Armour" category. The dataset consisted of:

- League, rarity, item_category : these columns had to do with categorizing all the different datasets from different seasons. Here they don't matter so we drop them
- Generalized features
- Date , time , time_hours : we did not treat this dataset as a time-series one, therefore we dropped those columns
- Days_in_snapshot : How many days it took for the item to get sold since it was listed as available for sale
- Ex_conv_rate : Daily exalt to chaos conversion rate
- Price_amount : this is the target variable. It is the price we are trying to predict

One major problem with the dataset was the variable number of features for each observation. Each observation, which means each item, cannot have more than a certain number of generalized features. That meant that each of these features could not appear in all the observations. Some appeared a lot (for example 'ex_# to maximum life" appears in 75.27% of our observations) and some other features appear a lot less (for example "ex_gain onslaught for # seconds when hit" appear in 0.001891% of our observations)[19].

Armour	50.422927
Energy Shield	46.267913
Evasion Rating	42.519067
ex_#% increased stun and block recovery	42.501362
ex_#% to cold resistance	40.985666
ex_#% to fire resistance	40.960399
ex_#% to lightning resistance	40.260821
⋮	
co_#% reduced fire damage taken	0.013063
ex_#% chance to avoid cold damage when hit	0.011860
ex_#% chance to avoid fire damage when hit	0.005328
ex_#% chance to dodge spell hits	0.005157
co_#% to all maximum resistances	0.004469
ex_#% increased area of effect	0.002235
ex_gain onslaught for # seconds when hit	0.001891

Table 9: Features in Rares dataset and their appearance percentage

There were a couple of strategies we implemented and tested to overcome this problem:

- Filling the missing values [13] with a mean, median or mode value : Since some features only appear in low priced items or high priced items, and considering that some of the features had a different weight of appearance in the game (which means that it is more rare to see these ones in one of the items) filling out missing values didn't help with the accuracy of our model, on the contrary since our dataset consisted of almost 60% of observations with price_amount < 2, it made all other prices more difficult to predict while leaving the accuracy for low prices almost the same.

- Removing features with appearance lower than a certain percentage threshold: this again did not help our model to make better predictions since features which were rarer sometimes meant an increase in price. Removing those features lowered our prediction accuracy for items with high prices.

In the end, it proved better to not remove any pre-existing features since generally the features with low appearance were connected with higher prices.

Dependent variable y (price_amount) distribution

Another problem was the number of observations associated with each different price_amount. Bucketing our observations to different price_amount ranges gave us the following results.

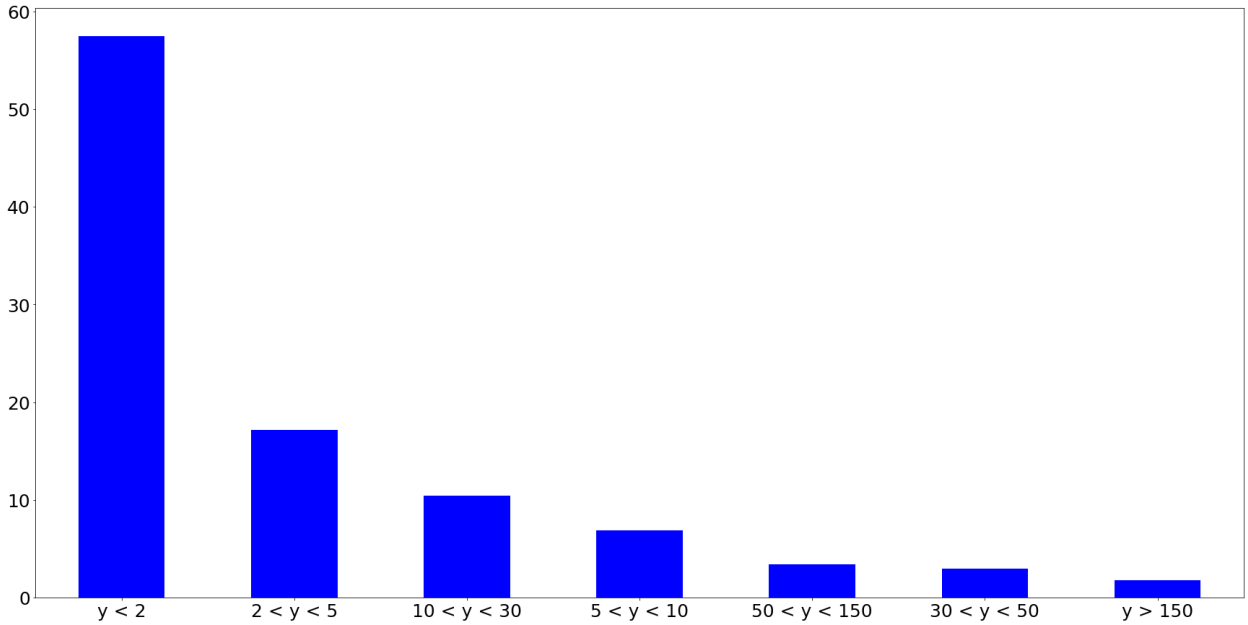


Figure 26: Bucketing price_amount (y) bar plot - Rares

It is clear that we have a lot more observations with price_amount < 2. For that reason one idea was to convert the target variable of each observation, price_amount, to follow a Gaussian distribution using base log. Converting the price_amount returns the following distribution:

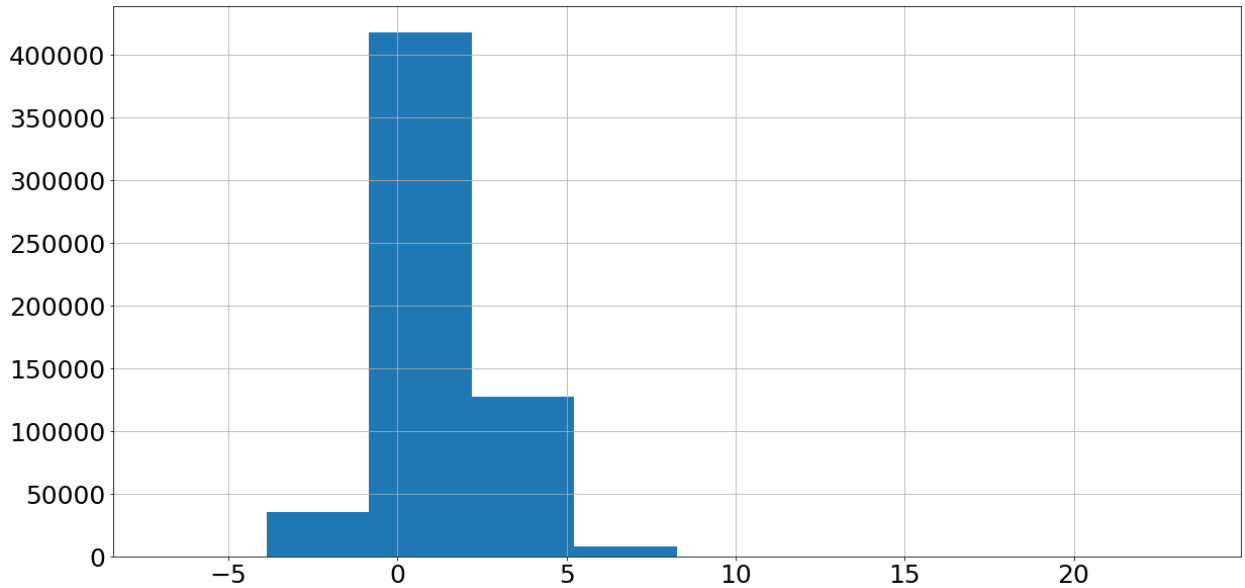


Figure 27: Price_amount (y) Gaussian distribution - Rares

After running a basic neural network, before taking the natural logarithm of the values and after, we get the following results:

Before np.log

	<20%	<30%	<40%	<50%	>50%	total%
(0,1]	2132	936	890	1000	16744	14.136946
(1,5]	1451	730	838	805	2113	36.735725
(5,10]	308	165	269	302	1490	18.666140
(10,30]	1334	529	434	352	1883	41.107679
(30,50]	376	181	157	164	479	41.046426
(50,150]	424	190	211	147	631	38.303182
(150,nan]	75	31	30	35	109	37.857143

After np.log

	<20%	<30%	<40%	<50%	>50%	<50% error
(0,1]	6204	2012	1472	995	6586	61.862297
(1,5]	1712	1001	1271	1509	4886	52.924174
(5,10]	93	58	77	103	669	33.100000
(10,30]	1568	640	483	392	2362	56.620753
(30,50]	565	265	255	225	596	68.730325
(50,150]	412	245	244	199	536	67.237164
(150,nan]	39	34	27	44	167	46.302251

Table 10: y bucketing before and after applying np.log() transformation - Rares

5.3 Feature Engineering

A big part of the problem was using domain knowledge of the data to create new features. This was fundamental to getting better results for our algorithms. Those features were:

- #_of_ele_resistances :
- #_of_resistances
- total_ele_resistance
- total_resistance

Partial Dependence

The partial dependence plot [20] shows the marginal effect one or two features have on the predicted outcome of a machine learning model. A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonous or more complex.

Using partial dependence we analysed the dependence of each feature's values to the target variable and created two new features, no_of_good_features and no_of_bad_features. According to domain knowledge, an item that has a number of good features above a certain threshold is considered valuable, and an item that has a number of bad features above a certain threshold is considered worthless. After engineering these 2 new features we use partial dependence again to figure out if our assumption was right in the first place.

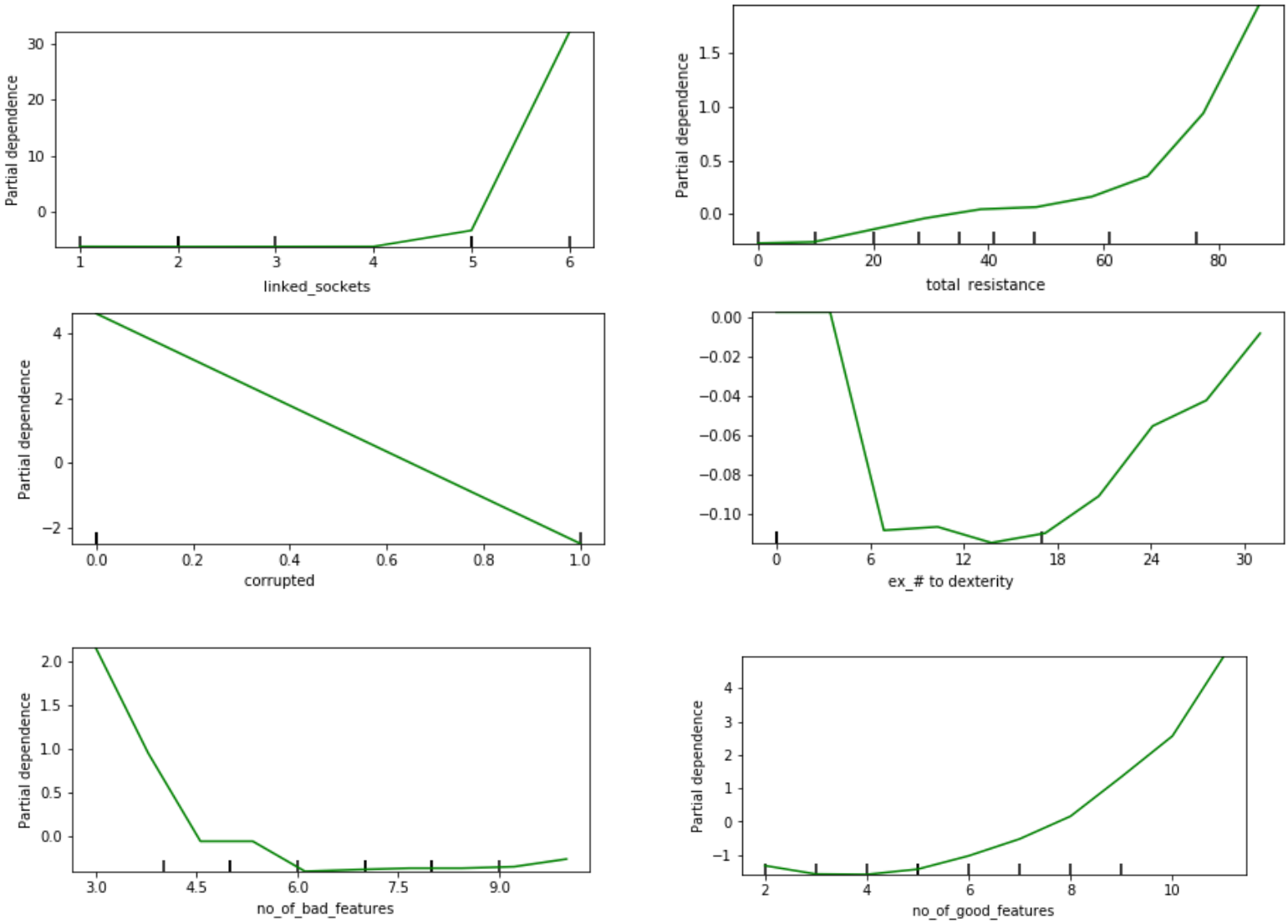


Figure 28: Features partial dependency plots

5.4 Outliers

As before for outliers we used IQR and Z-Score as our main methods, as well as days_in_snapshot < 10. From partial dependence plots we also saw that the higher the days_in_snapshot feature the higher the price. This made sense since the more expensive the item, the more difficult it is to be sold. We experimented with different IQR thresholds and Z-scores but at the end of the day only IQR gave us noteworthy results. Still, by inspecting which observations were considered outliers and running our machine learning processes both with and without removing outliers, we came to the conclusion that leaving the dataset as is, gives us better results overall.

	price >150
with outliers	0.65%
without outliers	1.80%

Table 11: Outliers existence percentage in Rares dataset (over 150 price)

As we can see by removing outliers, we remove almost ⅓ of our observations with price_amount > 150. This lowered our accuracy of predicting prices above that threshold sometimes even by half.

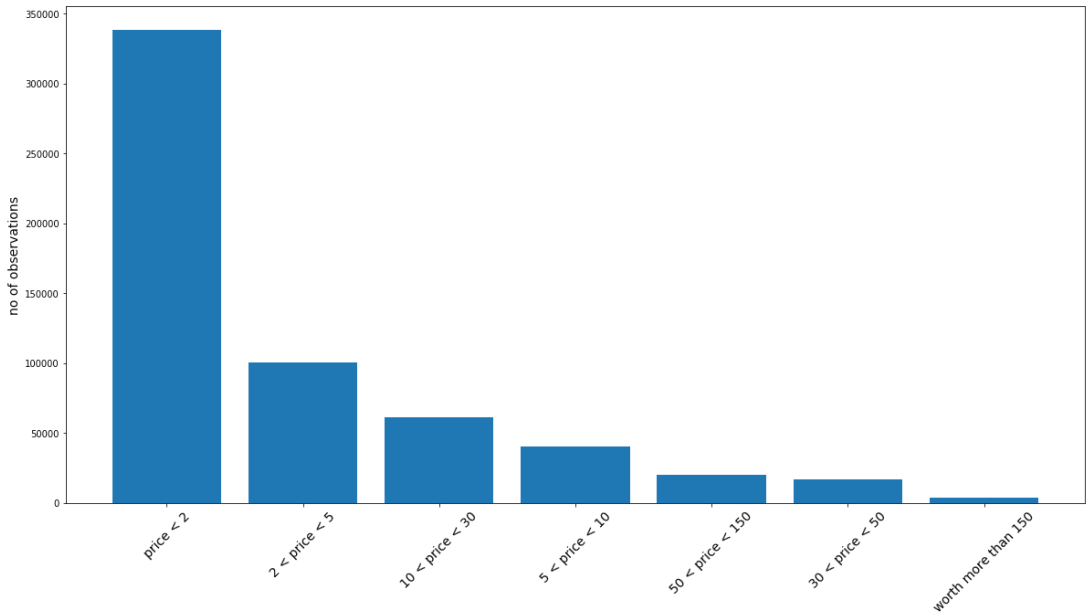
Using one of our neural networks, after running the model with and without outliers we got an accuracy of 60.2% for prediction items with price_amount > 150 with outliers while only 50.7% without outliers.

5.5 Approach - Regression vs Classification

Our initial goal was to build a regression model which could price items within a relatively small margin from their actual values. Without using the logarithmic values of the price_amount our results were not as good, since we weren't able to price any items with an accuracy of 42% or above. After the logarithmic transformation our results got better but were still lackluster in some cases.

This gave us the idea to bucket our prices in price bins and instead of trying to build a regression model, to try and classify in which bin each item belonged to. Using a dynamic algorithm, which used the minimum percentage each price bin should have and the maximum range from the lower and the higher value of the bin as parameters, we broke down the price to segments as seen in the plot below.

Figure 29: y (price_amount) bucketing in Rares dataset



5.6 Theory

5.6.1 Machine learning algorithms

Apart from building neural network models and training them, we also tested basic machine learning algorithms such as linear regression, k-means clustering etc :

Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable [21].

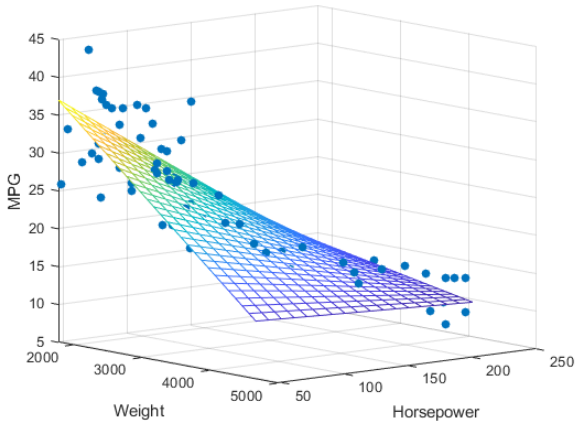


Figure 30: Linear Regression

In linear regression, the relationships are modelled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.
Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

Linear classifier

In the field of machine learning [22], the goal of statistical classification is to use an object's characteristics to identify which class (or group) it belongs to. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector. Such classifiers work well for practical problems such as document classification, and more generally for problems with many variables (features), reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use.

Decision Trees

Decision trees build regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree corresponds to the best predictor called root node.

XGBoost

XGBoost stands for eXtreme Gradient Boosting. It is an implementation of gradient boosting machines. The XGBoost library implements the gradient boosting decision tree algorithm. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines.

Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then they are added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

K - Means Clustering

A cluster refers to a collection of data points aggregated together because of certain similarities.

We'll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The 'means' in the K-means refers to averaging of the data; that is, finding the centroid [23].

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids It halts creating and optimizing clusters when either:

- The centroids have stabilized—there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

K-nearest Neighbors

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

K-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification.

5.6.2 Activation functions

Activation functions are really important for a Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable. They introduce non-linear properties to our Network. Their main purpose is to convert an input signal of a node in a A-NN to an output signal. That output signal now is used as an input in the next layer in the stack.

Specifically in A-NN we do the sum of products of inputs(X) and their corresponding Weights(W) and apply an Activation function f(x) to it to get the output of that layer and feed it as an input to the next layer.

If we do not apply an Activation function then the output signal would simply be a simple linear function. A linear function is just a polynomial of one degree. Linear equations may be easy to solve but they are limited in their complexity and have less power to learn complex function mappings from data.

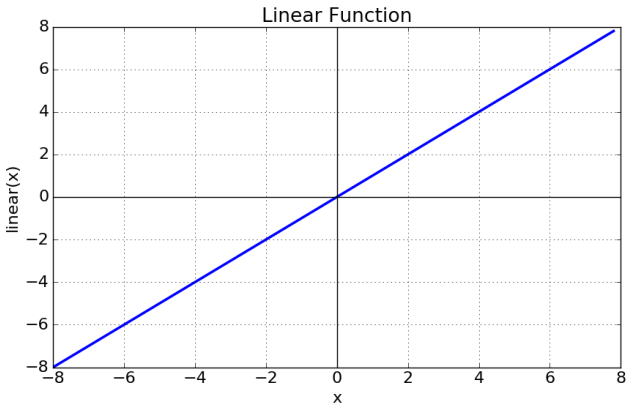


Figure 31: Linear Activation Function

Non-linear functions are those which have degree more than one and they have a curvature when we plot a Non-Linear function. Hence, we need to apply an Activation function f(x) so as to make the network more powerful and add to it the ability to learn something complex and complicated from data and represent non-linear complex arbitrary functional mappings between inputs and outputs

Except from the **Linear** activation function, we also experimented with the following ones:

Swish

With the function f(x) = x · sigmoid(x),
the swish activation function has been observed to work better than ReLU on deeper models across a number of challenging data sets.

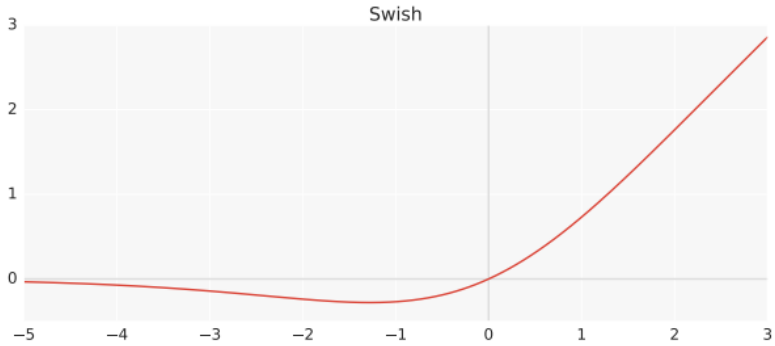


Figure 32: Swish Activation Function

Swish is a smooth, non-monotonic function that consistently matches or outperforms ReLU on deep networks applied to a variety of challenging domains such as Image classification and Machine translation. It is unbounded above and bounded below & it is the non-monotonic attribute that actually creates the difference. With self-gating, it requires just a scalar input whereas in multi-gating scenario, it would require multiple two-scalar input.

Sigmoid

t is a activation function of form f(x) = 1 / 1 + exp(-x) . Its Range is between 0 and 1. It is a S—shaped curve. It is easy to understand and apply but it has major reasons which have made it fall out of popularity -

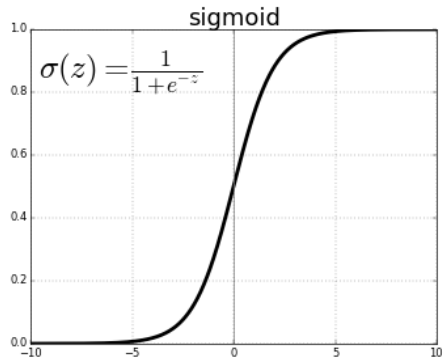


Figure 32: Sigmoid Activation Function

- Vanishing gradient problem
- Its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.

Softmax

The above functions are not suitable for classification problems, and for that we needed a new function. The softmax function is a more generalized logistic activation(sigmoid) function which is used for multiclass classification. If the problem was on binary classification the sigmoid function would work just as well.

5.7 Feature scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

Here we had features like ‘ex_# to maximum life” with ranges from 1 to 130+ and others like “ex_#% to fire resistance” with ranges from 1 to 50. For scaling we experimented with 2 different scalers: Min-max scaler and Standard Scaler.

Min-Max Scaler

In this we subtract the Minimum from all values – thereby marking a scale from Min to Max. Then divide it by the difference between Min and Max. The result is that our values will go from zero to 1. This is quite acceptable in cases where we are not concerned about the standardisation along the variance axes. e.g. image processing or neural networks expecting values between 0 to 1.

The downside however is that because we have now bounded the range from 0 to 1, we will have lower standard deviations and it suppresses the effect of outliers.

Standard Scaler

We standardize features by removing the mean and scaling to unit variance.
The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples or zero if with_mean=False, and s is the standard deviation of the training samples or one if with_std=False.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

Comparing the two with a neural network consisting of 2 dropout layers of 0.2 rate, 4 hidden layers with 400 neurons and a Relu activation function, a linear activation function on the output layer, 30 epochs and a batch size of 10000 we got the following results which helped us decide to drop the Standard scaler and use the Min Max Scaler :

Standard Scaler							Min Max Scaler						
	<20%	<30%	<40%	<50%	>50%	<50% error		<20%	<30%	<40%	<50%	>50%	<50% error
(0,2]	7926	3829	3585	2498	11006	61.843018	(0,2]	5873	3655	2935	1915	7227	66.549410
(2,5]	1246	776	871	1134	4187	49.026053	(2,5]	746	475	555	769	3509	42.038322
(5,10]	122	102	119	158	836	37.471952	(5,10]	77	65	92	96	709	31.761309
(10,30]	1839	809	624	451	3086	54.677633	(10,30]	1627	563	477	312	2324	56.175750
(30,50]	425	262	299	301	977	56.846290	(30,50]	508	261	244	261	727	63.668166
(50,150]	249	190	244	290	891	52.199571	(50,150]	270	176	231	229	703	56.308266
(150,nan]	52	46	73	104	682	28.735632	(150,nan]	71	48	69	100	570	33.566434

Table 12: Predicted errors using Standard and MinMax Scalers on Rares dataset

5.8 K-fold

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference of the model, such as k=10 becoming 10-fold cross-validation.

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group:
- Take the group as a hold out or test data set
- Take the remaining groups as a training data set
- Fit a model on the training set and evaluate it on the test set
- Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

In our problem we experimented with k=5 and k=10 and decided to use k=10 since it gave us better results, probably because the more the training set, the more different combinations our model trained on.

5.9 Grid Search

Before we start our hyperparameter tuning using GridSearch we had to choose our ranges, different activation functions, optimizers etc. Also, we had to choose how many hidden layers and dropout layers we were going to use.

We experimented with the following neural networks :

- 1 hidden layer
- 2 hidden layers
- 2 hidden layers 1 dropout
- 2 hidden layers 2 dropouts
- 3 hidden layers 2 dropouts
- 4 hidden layers 2 dropouts
- 5 hidden layers 2 dropouts

From the above, using 2 hidden layers and 2 dropouts gave us the best performance overall. Using more than 2 hidden layers only increased the time it took to train our model and lowered the epochs needed until our model started to overfit. Comparing the two with regression gave us the following results:

30 epochs, 2 hidden layers, 2 dropout layers 8 minutes 13 seconds training time						
	<20%	<30%	<40%	<50%	>50%	<50% error
(0,2]	7926	3829	3585	2498	11006	61.843018
(2,5]	1246	776	871	1134	4187	49.026053
(5,10]	122	102	119	158	836	37.471952
(10,30]	1839	809	624	451	3086	54.677633
(30,50]	425	262	299	301	977	56.846290
(50,150]	249	190	244	290	891	52.199571
(150,nan]	52	46	73	104	682	28.735632

30 epochs, 5 hidden layers, 2 dropout layers 15 minutes 50 seconds training time						
	<20%	<30%	<40%	<50%	>50%	<50% error
(0,2]	8916	4354	3532	2586	9493	67.130640
(2,5]	866	579	760	1057	4946	39.741715
(5,10]	153	61	85	112	914	31.018868
(10,30]	1213	760	806	631	3330	50.593472
(30,50]	229	198	252	305	1250	44.046553
(50,150]	80	78	120	221	1399	26.290832
(150,nan]	27	21	38	72	845	15.752742

Table 13: Grid Search Result grid, best models – Rares

As we can see, while our predictions for price_amount < 5 are better, every other price range accuracy is worse.

5.10 Pipelines (scikit-learn)

A typical machine learning task generally involves data preparation to varying degrees.
After a dataset is cleaned up from a potentially initial state of massive disarray however, there are still several less intensive yet no less important transformative data preprocessing steps such as feature scaling, dimensionality reduction etc.

A typical scenario is to string a number of transformations together and ultimately finish off with an estimator of some sort.

Scikit-learn Pipelines is used here to build such a pipeline of scaling and chaining an estimator.
This method provides many of the same advantages that decoupling does in software development.

Advantages include:

- Flexibility: Units of computation are easy to replace. If you discover a better implementation for one chunk, you can replace it without changing the rest of the system.

- Scalability: Each bit of computation is exposed via a common interface. If any part becomes a bottleneck, you can scale that component independently. Common scaling techniques might involve a load balancer or additional backends.

- Extensibility: when the system is divided into meaningful pieces it creates natural points of extension for new functionality.

In our case, we stringed together in the same Pipeline the feature scaling, the machine learning process and the grid search with k-fold cross validation with k=10.

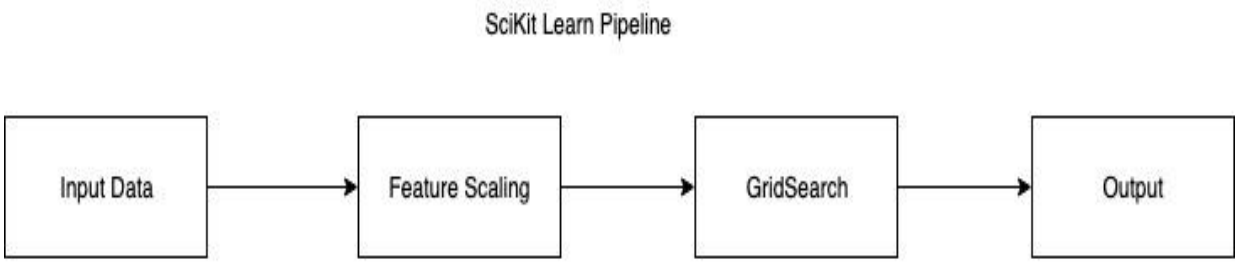


Figure 34: SciKit learn pipeline

For regression the process looked like this:

```
estimator =
KerasRegressor(build_fn=build_model,verbose=1,epochs=epochs,batch_size=batch_size)

estimators = []
estimators.append(('standardize',MinMaxScaler(feature_range = (0, 1)) ))
estimators.append(('mlp',estimator))
pipeline = Pipeline(estimators)

kfold = KFold(n_splits=10, random_state=seed)

gridsearch_parameters = configs['gridsearch_model']
```

```
grid_search = GridSearchCV(estimator = pipeline,
                           param_grid = gridsearch_parameters,
                           scoring='neg_mean_absolute_error',cv=kfold)

grid_search = grid_search.fit(X,y)

#to get the best parameters
print(grid_search.best_params_)
```

5.11 Evaluation Metrics

Analysing the problem with regression algorithms, the metrics we could use were mean squared error, mean absolute error or mean absolute percentage error. Even though all 3 could be used to measure the accuracy for continuous variables, none of the above really helped us determine why our model was wrong in our predictions, which price_bucket had the best accuracy and how that accuracy changed with different hyperparameters.

For that reason we coded our own accuracy measurement for regression, calculating the error percentage of each of our predictions. If our prediction was less than 40% of the real price, then we counted the prediction as correct for the price bucket of the real price. That way we calculated the prediction accuracy of each price bucket.

Analysing the problem using classification, we not only calculated the overall accuracy of each model using sklearn's built in metrics library, but we also calculated the overall accuracy of each price bucket.

5.12 Results

Calculating the prediction accuracy for our regression and classification models after grid search we get the following diagram.

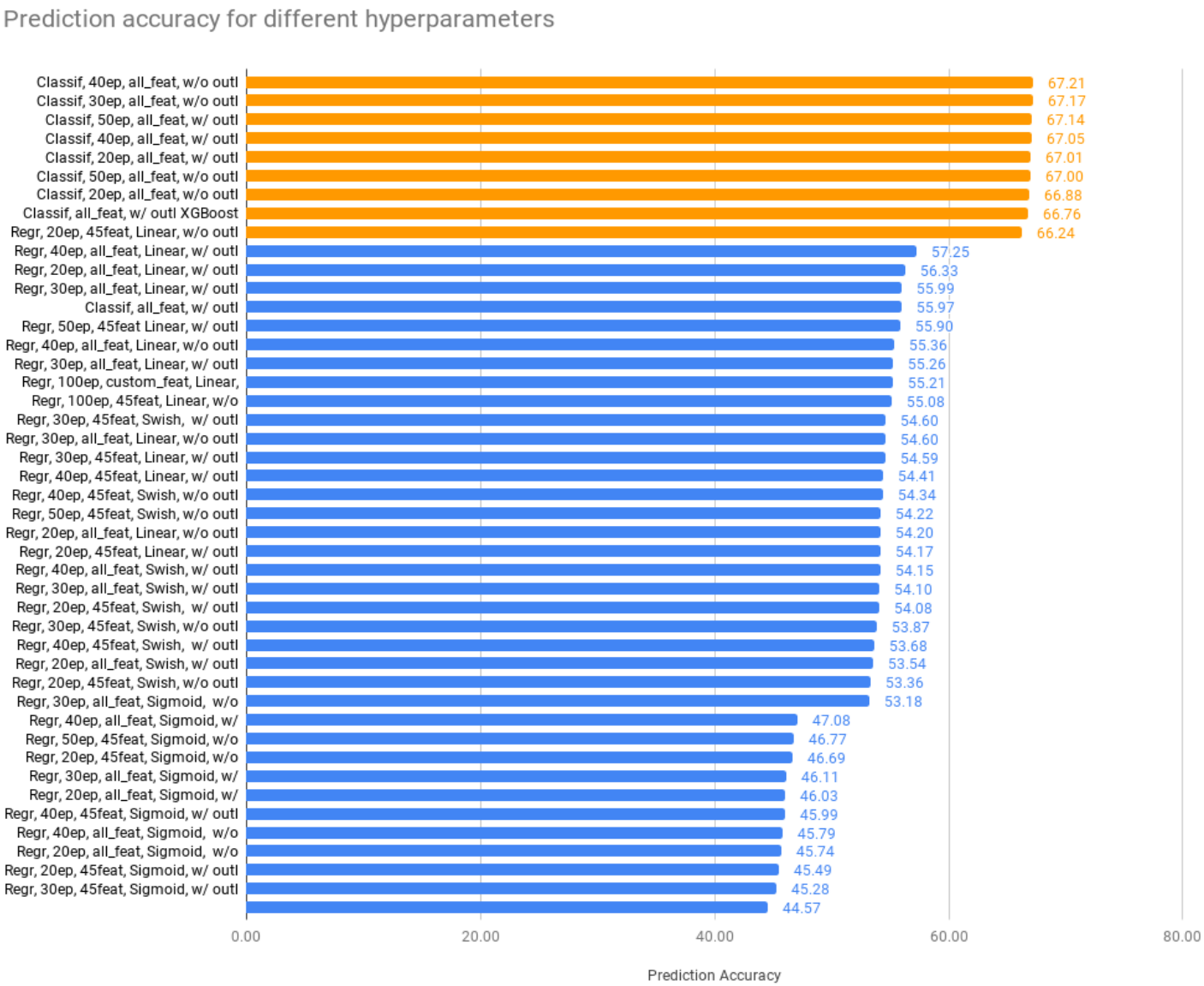


Figure 35: Prediction accuracy using models generated from hyperparameters technique - Rares

We can see that each classification model gives higher prediction accuracy than any other model. XGBoost has a 66.76% accuracy which is pretty significant for an out of the box with no tuning algorithm. The “middle” accuracy models from 57% to 53% consist of Decision Trees for classification and all the other regression models with Linear and Swish activation functions. Only one model with a Sigmoid activation function has 53% accuracy and all the others are below 47%. In fact, without this exception, every model with a Sigmoid activation function underperforms.

From the above diagram we could say that we should just use a classification approach to our problem and just select the model with the best accuracy. Although this would be correct for our overall accuracy, we get no information for the accuracy per price bucket. Considering that our dataset does

not have a good distribution of price ranges, consisting 60% of price ranges from 0 to 2, it could very well over perform for these price ranges and underperform for others.

For this reason, we need to take a closer look to the accuracies per price bucket, breaking the problem down to 3 different cases: the high overall accuracy range, the middle and the low.

High prediction accuracy models

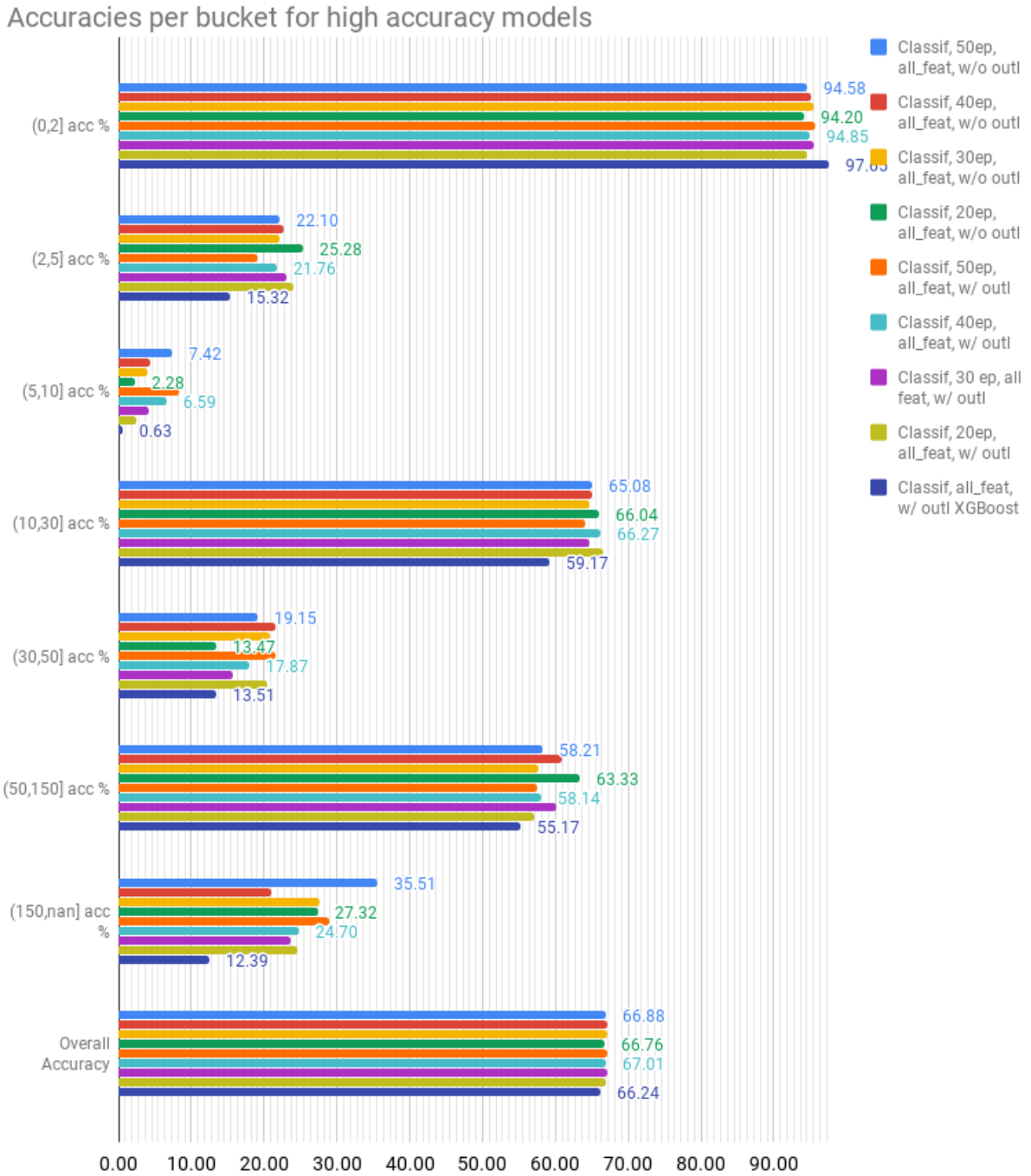


Figure 36: Accuracies per bucket for high accuracy models - Rares

As suspected, our model performs exceptionally well for the (0, 2] price range with a prediction accuracy of over 95% using XGBoost. Observing the other price ranges we get a decent prediction accuracy of 65%+ for items on the range of (50,150] as well as on the range of (10, 30]. For items on the price range of 150 and over the only model that comes close to a 35% is the classifier with 50 epochs, using all features and without outliers.
For the ranges of (2, 5] and (30, 50] our models are bad having a 20-22% accuracy. But for the range of (5, 10] no model can even come close to a 10% accuracy. If we used that model to classify the price of an item, we could be mostly correct for 3 of the price ranges, but be absolutely wrong for the other 4. This confirms our suspicions that relying solely on the overall prediction accuracy of a model would not give us the optimal results.

Middle prediction accuracy models

Taking a look at each price range individually we can draw the following conclusions:

- For the (0, 2] price range we observe that each regression model has a lower prediction accuracy from the only classifier in our group, which used Decision Trees, but also from the other classifiers in our previous diagram. The best model from regression is the one using a Swish activation function, running for 30 epochs without outliers and using 45 features after feature selection.
- For the (2, 5] price range we get an almost 53% prediction accuracy when running regression with 45 features, for 50 epochs, using a Linear activation function and without removing outliers. Just like before the model that performs the worse is the classification model. Using a Swish activation function we get an accuracy of 41% approximately or worse with 20 epochs dipping to a 30% accuracy. This is a lot better than just using a classification model, which had 25% accuracy. In other words, using regression instead of classification we double our accuracy for this price range
- For the (5, 10] price range our accuracies are not as bad as before, but we still cannot predict with a higher than 40% accuracy if an item belongs to this price range. Using a Swish activation function we get an accuracy of over 36%. We also observe that not removing outliers lowers our prediction accuracy, probably because these outliers were observations on the higher price ranges but had combinations of items that belonged to a lower price range.
- For the (10, 30] price range using regression we get a lower accuracy by almost 10%. Whatever the hyperparameters, the accuracies were really close, with the exception of activation functions with Linear being the to one give us the best results (from 54.7 to 56.6%)
- For the (30,50] price range we see an increase in accuracy of almost 3 times from the highest classification algorithm being the Decision Trees one with an accuracy of 26.7%. With regression we get an accuracy of 68.7% using a Linear activation function with 50%, with Swish being a close second with an accuracy of 68.6%. The increase in epochs after the value of 50 doesn't increase our accuracy since we see our model with 100 epochs and a Linear activation function having a lower accuracy.
- For the (50,150] price range our regression models are as good as our classification ones and even better, since using a Swish activation function with 50epochs and without outliers, gives a 68.88% accuracy and with a Linear activation function we have a 67.2% accuracy.

- For the 150 and over price range for 50 epochs using a Swish activation function we get 51.87% accuracy whereas using classification we got a 35.51% accuracy.

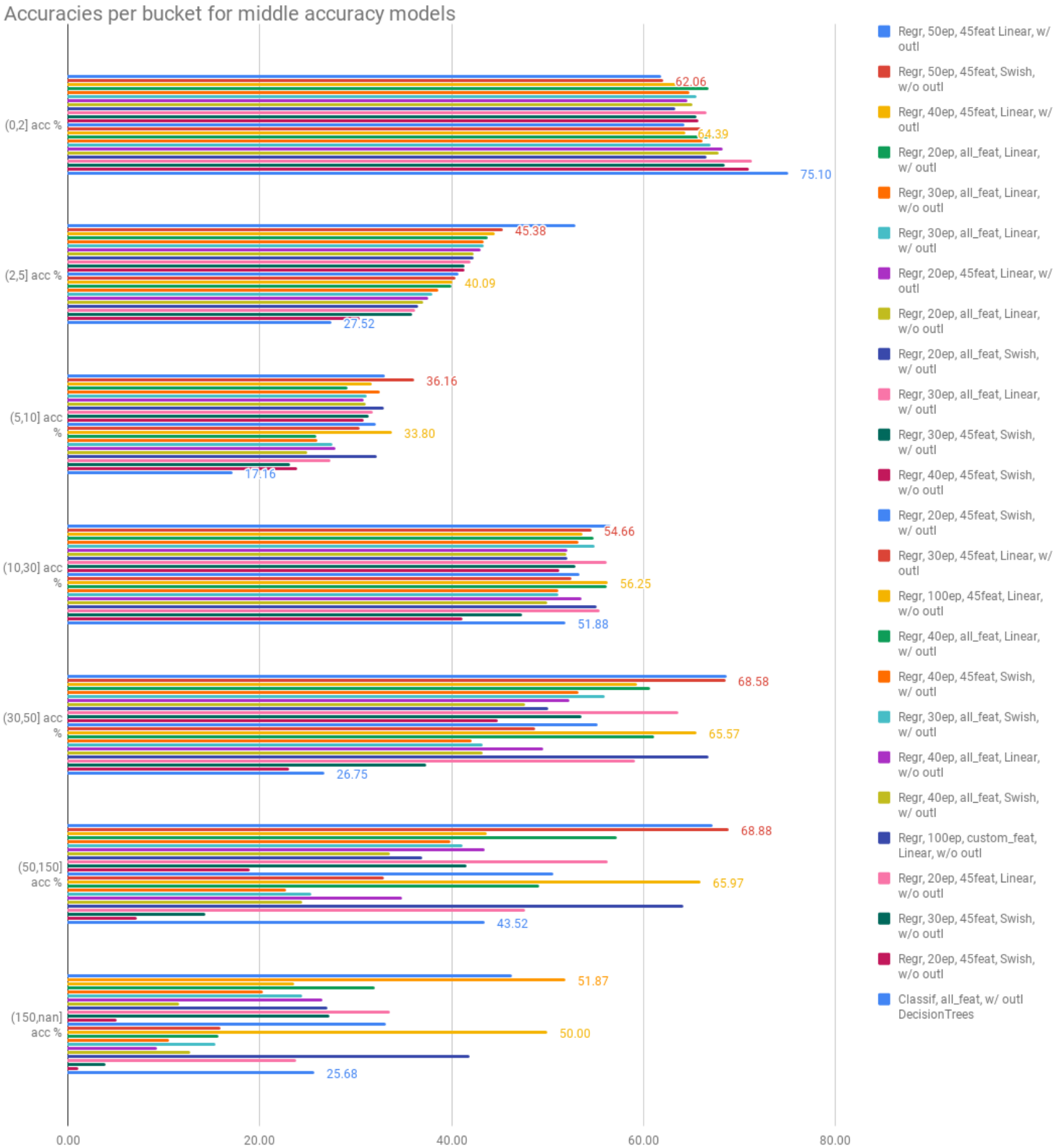


Figure 37: Accuracies per bucket for high accuracy models - Rares

Low prediction accuracy models

As we can see the reason our last models had such low overall prediction accuracy, was because they could not predict any item from 5 to over 150 price. That is because of the nature of the sigmoid function which, as previously mentioned, cannot output negative numbers. Even though a price of an item cannot have negative values, we used base logarithmic function on all prices to get a better distribution and that resulted in negative target variables and that is why using sigmoid activation function we cannot make predictions for prices higher than 5.

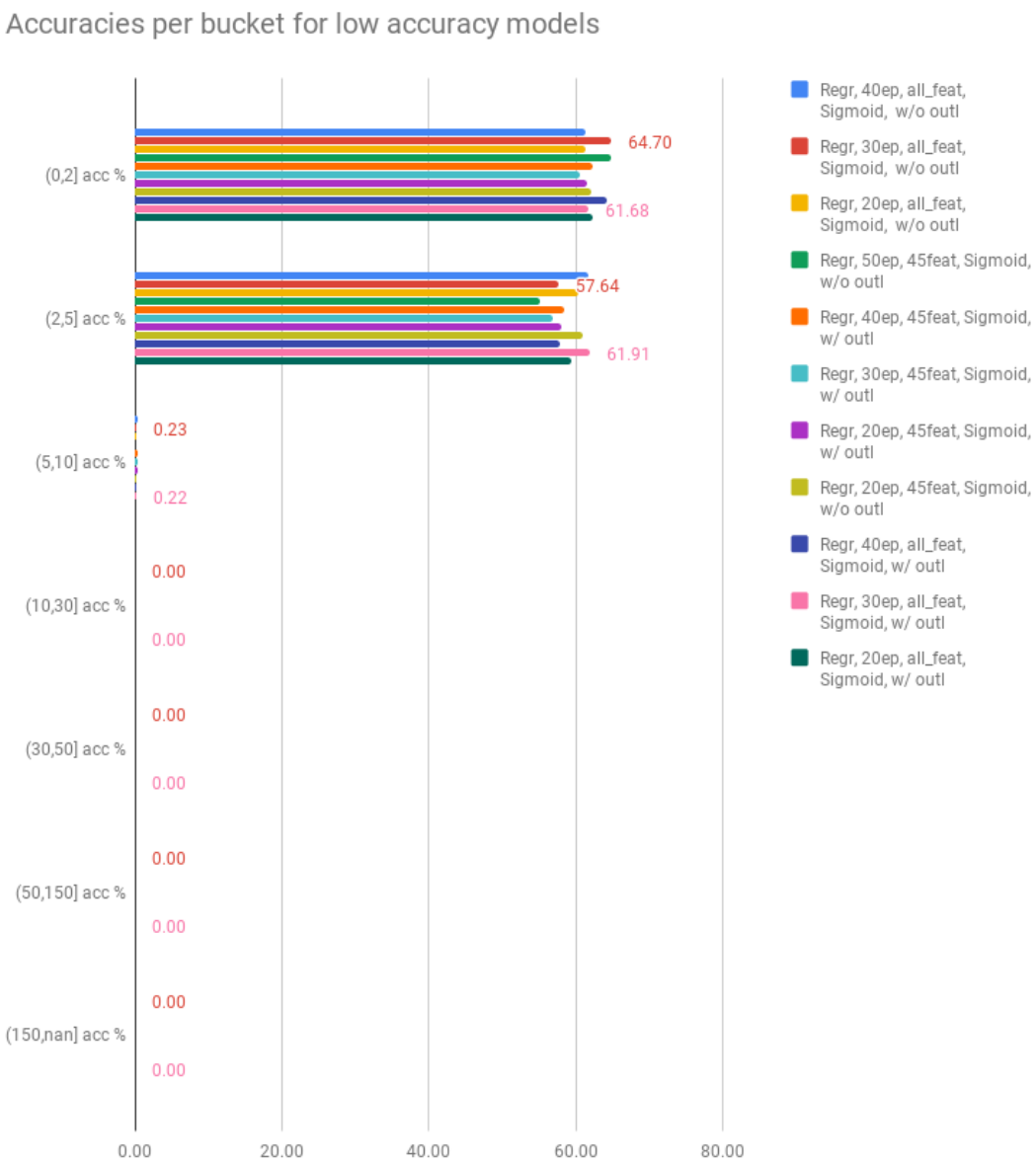


Figure 38: Accuracies per bucket for low accuracy models - Rares

After analysing the low, medium and high prediction accuracy models we see that each category does a good job predicting a price range but not all of the ranges. This becomes even more apparent when we observe the best models per price range in this last diagram.

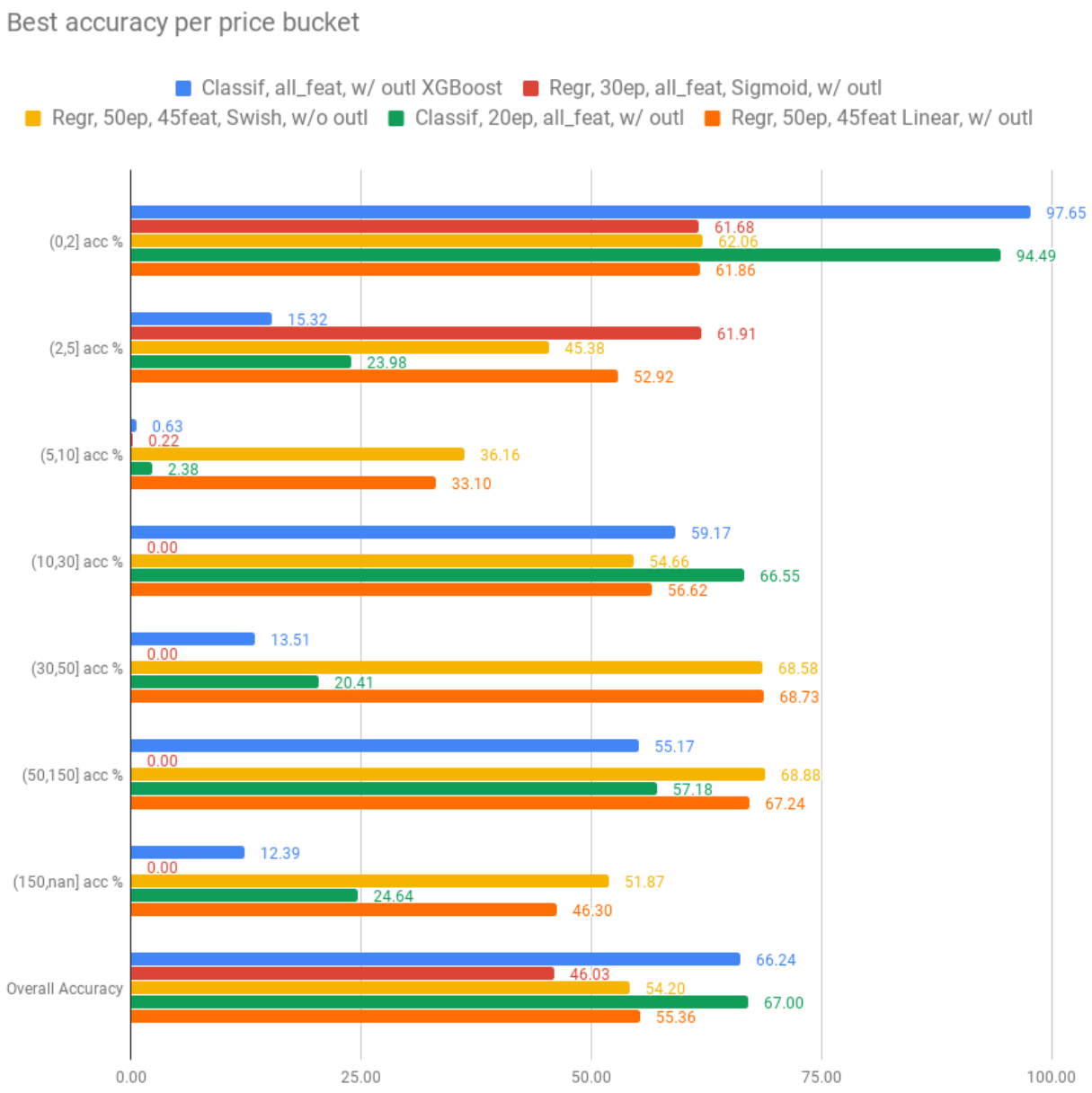


Figure 39: Best accuracy per price bucket - Rares

It is clear that classification models are really good for the ranges (0,2] , (10,30] and (50,150], regression models with a sigmoid activation function for the range of (2,5], regression models with either swish or linear activation functions for the range of (30,50] and regression model using a Swish activation function for the range of 150 and over. For the price range of (5, 10] there are no models that can give us a good prediction accuracy.

5.13 Ensemble

Ensemble methods that train multiple learners and then combine them for use, are a kind of state-of-the art learning approach. It is well known that an ensemble is usually significantly more accurate than a single learner [24].
Ensemble methods have already achieved great success in many real-world tasks like face detection and in the Xbox Kinect, a random forest-based skeleton tracking algorithm which allows people to interact with games freely without game controllers.

5.13.1 Combine Model Predictions into Ensemble Predictions

The three most popular methods for combining the predictions from different models are:

- **Bagging**. Building multiple models (typically of the same type) from different subsamples of the training dataset.
- **Boosting**. Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.
- **Voting**. Building multiple models (typically of differing types) and simple statistics (like calculating the mean) are used to combine predictions [26].

5.13.2 Bagging Algorithms

Bootstrap Aggregation or bagging involves taking multiple samples from your training dataset (with replacement) and training a model for each sample.
The final output prediction is averaged across the predictions of all of the sub-models.

Such algorithms are Bagged Decision Trees, Random Forest, Extra Trees

5.13.3 Boosting Algorithms

Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence.

Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction.
Such algorithms are AdaBoost, Stochastic Gradient Boosting.

5.13.4 Voting Ensemble

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms.

It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap the models and average the predictions of the sub-models when asked to make predictions for new data [25].

The predictions of the sub-models can be weighted [27] , but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from submodels, but this is called stacking (stacked aggregation) and is currently not provided in scikit-learn.

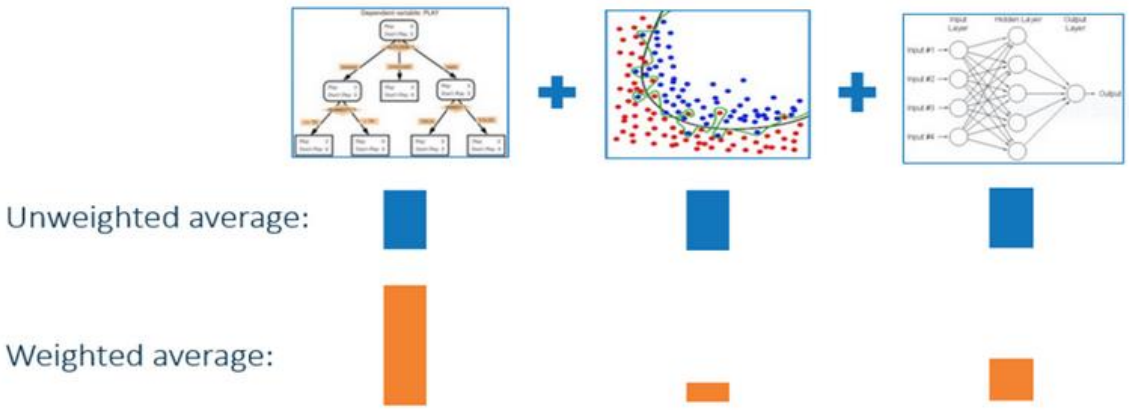


Figure 40: Averaging predictions to form ensemble models.

Stacked Generalization or stacking is an ensemble technique that uses a new model to learn how to best combine the predictions from two or more models trained on your dataset.

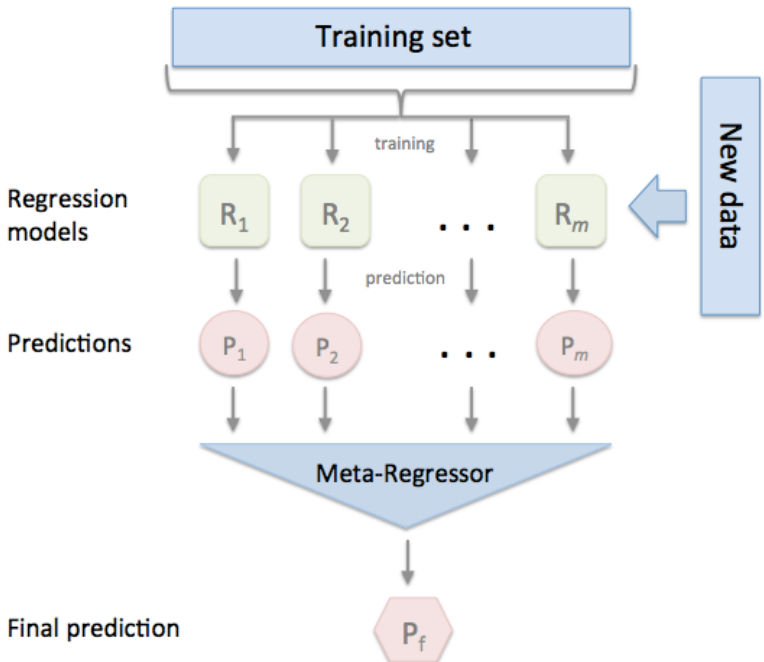


Figure 41: stacking concept

5.13.5 Ensemble approach for POE Rares Regression results

Observing our analysis and performance results on Rares dataset we see that the algorithms used do not offer the best outcome for all price buckets.

This is an indication that by applying ensemble methods we might get a better performance score.

We used voting with weighted prediction.

The sub-learners used are:

Sub - learner	Better results at segment	Prediction Accuracy
ANN with activation function 'swish', 50 epochs, 45 features, without outliers	(30,50 (50,150 (150,nan] (5,10]	55.38%
ANN with activation function 'swish', 30 epochs, 45 features, without outliers	(0,2]	53.76%
ANN with activation function 'sigmoid', 30 epochs, 45 features, without outliers	(2,5]	47.44%
ANN with activation function 'linear', 100 epochs, 45 features, without outliers	(10,30]	55.41%

Table: 14 Sub – learners configuration participating in the ensemble predictor

We fitted the models and concatenated their predictions in a list.

We used the method minimize to find the predictor's weights and the ensemble score applying the function log_loss() .

The function log loss is the following:

```
def log_loss_func(weights):
    ''' scipy minimize will pass the weights as a numpy array '''
    final_prediction = 0
    for weight, prediction in zip(weights, predictions):
            final_prediction += weight*prediction
```

return mean_squared_error(y_test, final_prediction)

and is based on mean squared root.

The ensemble gave us a score of 57.12 % accuracy improving the previous performance score of 55.41% which was the best score out of the 4 models we used.

This first approach is the simplest possible since we have better results when we apply classification algorithms to the Rares dataset.
Producing an ensemble with the best classification results or even deploying a stacked aggregation of 2 levels would possibly return better results.

5.14 Conclusion

The dataset is comprised of 580.000 observations hosting a variable number of features from 90+ to 510+ in combinations of 11maximum. As already shown in the diagrams some combinations between these features are rather rare and have a small presence in the dataset. As a result the algorithms do not have any way to learn and the performance drops.

Even though none of our algorithms was good at predicting all of the items' prices, breaking the price in ranges and observing those price ranges, gave us insight on which algorithms performed best on each one. After tackling the problem from a classification perspective, we were able to build a model that would accurately predict if an item was valuable or not. With a performance score of 97.65% in the segment of (0, 2] the model is pretty confident that this is a not valuable item.

For the regression part of the problem, using 4 different regression models and via ensemble learning, we were also able to improve our overall accuracy. Tweaking our parameters and finding better average weights for our models or trying different models could give us even better results. Finally, using ensembling on classification models could also help with improving our overall accuracy.

After observing our results we came to the conclusion that one big improvement on our algorithms' performance, would be the addition of more data. This would result not only in more different combinations but also more observations per price range. There are a number of ways we could add more data:

Reading the API stash feed from the start of the game

This would include more than 2.5 years of transactions but it would also introduce a number of problems since the API does not have time properties which we introduced ourselves and it would mean it is even more difficult to remove faulty observations. It would also introduce problems related to the storage of the dataset and the computing power needed to build the dataset and would imply a completely new distributed architecture.

Boosting the dataset with our own samples

Sampling around the value of the features, and creating a bunch of similar samples as well as using our domain knowledge to create more observations. This would certainly introduce bias to our models but it could potentially improve our performance.

Improving the current dataset using user feedback

This would be rather difficult to implement, but we could deploy our best model, create software around it to be used directly from players in the game and then get feedback directly from the players for the validity of our predictions. That way we could determine which combinations of features create problems and introduce initial feature weights to our problem as well as get more data directly from the players.

Chapter 6: Future Work

The project may have proved to be harder than initially expected and the performance of our models is certainly not the best for the Rare items, but there are still a lot of ideas we could test in the future that may increase the credibility of the dataset or the performance of our models.

Fighting fake data

- When building our dataset we decided to create a number of features to use later, one of them being the days it took for an item to be sold. Later, we decided that we should have tracked the hours it took for an item to be sold or even minutes since, according to domain knowledge, an item that takes less than 30 minutes to be sold, has a high chance of being fake data.

- In the dataset there is a number of features that are not used.
  One of them is the username of the player that made the transaction.
  We could classify players in categories expressing the level of confidence we estimate not only from the dataset itself but from other external sources too.

Legitimate outlier values

- In certain cases outlier values are completely legitimate.
  One such case is when an item in Unique items has a spike in value when a streamer uses it while playing.
  We could use causal inference to estimate the deviation from the normal plot and forecast the price correctly

- Another case is when a certain rare combination takes effect when the game applies an attribute to the item.

Natural Language processing on game forums and reddit

For our Unique items analysis, we could use natural language processing on game forums or reddit, to determine positive player language on certain items and introduce new features in our dataset which would enable us to better predict trends of items.
Reddit and forums also is a very rich source of various players' guilds that try to scam novice players to sell cheap. This could help classify certain players as Untrusty and thus the items they list for sale are considered low on price, or other players as Novice and thus their items are not to be considered high in confidence regarding to their price.

Moreover, being able to determine the "streamer effect", a phrase which in the gaming world means that if a famous person uses certain items these items would raise in price, will also

help predict trends of those items.

Meta items

In the world of Path of Exile, the “meta” is referred to the best items in a given time in the game. These “meta” items are generally more expensive. Data mining the most used items from external websites would help us determine which are these items and engineer a new feature which would represent the popularity of an item.
These items, if they are rare, also contain certain features which would help us initialize the weight of those features for our machine learning models.

Building models for individual price ranges

Another idea to be implemented is to build binary classification models, one for each price range. To do that, for a model of each price range we would categorize the price of each observation as 0 or 1 if it wasn’t or was in that price range and train each model with all the observations. Then we would use ensembling to predict the price range of an item.

Bibliography

[1] Ian Goodfellow and Yoshua Bengio and Aaron Courville : *Deep Learning, An MIT Press book*

[2] *Recurrent neural network. 2019, Wikimedia Foundation. https://en.wikipedia.org/wiki/Recurrent_neural_network*

[3] Zhao, Z., Chen, W., Wu, X., Chen, P. C. Y., Liu: *LSTM network: a deep learning approach for short-term traffic forecast. IET Intelligent Transport Systems, 11(2): 68–75.*

[4] Jae Hyuk Han*:* *Comparing Models for Time Series Analysis, University of Pennsylvania. 2018*

[5] David Paper*:* *Data Science Fundamentals for Python and MongoDB Apress 2018*

[6] Isabelle Guyon, Andre Elisseeff *:* *An Introduction to Variable and Feature Selection, Journal of Machine Learning Research 3 (2003) 1157-1182*

[7] Andreas Eckner*:* *Some Properties of Operators for Unevenly Spaced Time Series_ 2017*

[8] DOUGLAS C. MONTGOMERY,CHERYL L. JENNINGS,MURAT KULAHCI*:* *Introduction to Time Series, Analysis and Forecasting, Wiley 2008*

[9] Shamsul Masum, Ying Liu and John Chiverton*:* *Multi-step Time Series Forecasting of Electric Load using Machine Learning Models*

[10] Mathieu Lepot , Jean-Baptiste Aubin and François H.L.R. Clemens*:* *Interpolation in Time Series: An Introductive Overview of Existing Methods, Their Performance Criteria and Uncertainty Assessment, 2017*

[11] ***The Python Deep Learning library*** *. Keras Documentation. https://keras.io/*

[12] ***TensorFlow****. https://www.tensorflow.org/.*

[13] G. David Garson*:* *Missing Values Analysis and Data Imputation, Statistical Associates Publishers; 2015*

[14] Sigurd Øyen *:* *Forecasting Multivariate Time Series Data Using Neural Networks, Master of Science in Cybernetics and Robotics, 2018*

[15] Souhaib Ben Taieb,Gianluca Bontempi, Amir Atiya - Antti Sorjamaa: *A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition, Expert Systems with Applications 39(8)*

[16] Forecasting: Principles and Practice. *Rob J Hyndman and George Athanasopoulos. Monash University, Australia*

[17] Tenko Raykov George A. Marcoulides:*An Introduction to Applied Multivariate Analysis, 2008 by Taylor & Francis Group, LLC*

[18] Winnie Wing-Yi Chan : *A Survey on Multivariate Data Visualization, Department of Computer Science and Engineering, Hong Kong University of Science and Technology,2006*

[19] Verónica Bolón-Canedo • Noelia Sánchez-Maroño :
Feature Selection for High-Dimensional Data, Springer 2015

[20] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin: *Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of individual Conditional Expectation, 2014*

[21] Miroslav Kubat: *An Introduction to Machine Learning, Springer 2015*

[22] Simon Haykin: *Neural Networks and Learning Machines, McMaster University, Prentice Hall 1999*

[23] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani:*An Introduction to Statistical Learning with Applications in R, Springer 2013*

[24] Cha Zhang , Yunqian Ma : *Ensemble Machine Learning, Methods and Applications, Springer 2012*

[25] *Ensemble Selection from Library of Models. Proceedings of ICML' 04*

[26] Z.-H. Zhou: *Ensemble Methods, Foundations and Algorithms. . CRC Press 2012*

[27] Ludmila Kuncheva, Juan J. Rodríguez: *A weighted voting framework for classifiers ensembles. Knowledge and Information*

Περιεχόμενα Εκτεταμένης Περίληψης

Εκτεταμένη Περίληψη

Κεφάλαιο 1 :Εισαγωγή

1.1 Το πρόβλημα

Για παιχνίδια με οικονομίες εντός παχνιδιού οι οποίες ακμάζουν μέσω της αγοράς και της πώλησης αντικειμένων, η γνώση για τα αντικείμενα του παιχνιδιού είναι καταλύτης για να αυξήσει ένας νέος παίκτης το πλούτο του. Για το project μας χρησιμοποιήσαμε το παιχνίδι Path of Exile, το οποίο έχει μια συνεχώς μεταβαλλόμενη αγορά με άντικείμενα που αποτελούνται από πάνω απο 172 τετράκης εκατομμύρια διαφορετικούς συνδυασμούς χαρακτηριστηκών.

1.2 Το παιχνίδι

Στο παιχνίδι Path of Exile, οι παίκτες έχουν διαφορετικούς στόχους και για την επίτευξή τους αποκτούν αντικείμενα. Τα αντικείμενα χωρίζονται σε Rare και Unique. Τα Unique αντικείμενα έχουν συγκεκριμένους τύπους χαρακτηριστικών και αλλάζουν μόνο οι τιμές τους. Για τα Rare, τα χαρακτηριστικά αυτά μπορούν να είναι συνδυασμοί από 6 μέχρι 11 από μία κατηγορία χαρακτηριστικών και ο συνδυασμός τους αλλά και οι τιμές του κάθε χαρακτηριστικού ορίζει τη τιμή του αντικειμένου.

Η διαπραγμάτευση στο παιχνίδι γίνεται με τη χρήση stashes. Τα stashes παίζουν το ρόλο βιτρίνας στην οποία απεικονίζει ένας παίχτης τα αντικείμενα που θέλει να πωλήσει. Χρησιμοποιώντας το ενσωματωμένο API του παιχνιδιού, κατασκευάζουμε ένα σύνολο δεδομένων το οποίο αποτελείται από τις πωλήσεις που έγιναν στο παιχνίδι

1.3 Στόχος.

Οι στόχοι μας είναι δύο ανάλογα το τύπο αντικειμένου. Για τα Unique αντικείμενα θα παρακολουθήσουμε την τιμή τους και θα προσπαθήσουμε να προβλέψουμε την αλλαγή της στο μέλλον ώστε ο παίχτης να αποφασίσει αν θα αγοράσει το αντικείμενο με σκοπό το κέρδος επειδή η τιμή του θα ανέβει ή θα το πουλήσει για να αποφύγει τη ζημία επειδή η τιμή του θα πέσει. Για τα Rare αντικείμενα, σκοπός είναι η εκτίμηση της τιμής τους ανάλογα τα χαρακτηριστικά από τα οποία αποτελούνται και χρησιμοποιώντας αυτή τη πληροφορία ο παίκτης να αποφασίσει αν θέλει να πουλήσει ή όχι το αντικείμενο.

Κεφάλαιο 2: Ανάλυση συνόλου δεδομένων Unique – Πρόβλεψη μελλοντικών τιμών

2.1 Το σύνολο δεδομένων

Για το κομμάτι αυτό του προβλήματος διαλέξαμε 2 unique αντικείμενα, το κάθε ένα για διαφορετικό λόγο :

- **Tabula Rasa Simple Robe**
  Το αντικείμενο αυτό δεν ειναι χαρακτηριστικά με συσχετισμό με τη τιμή μεγαλύτερο του 0.1 ή μικρότερο του -0.1 εκτός από την ημερήσια ισοτιμία μετατροπής των 2 κυριότερων συναλλαγμάτων του παιχνιδιού, η οποία χαρακτηρίζει τον πληθωρισμό της αγοράς. Τις πρώτες μέρες της κάθε σεζόν το αντικείμενο αυτό ειναι πολύ πολύτιμο και η τιμή του πέφτει δραματικά καθώς περνούν οι μέρες ή με άλλα λόγια καθώς αυξάνεται η πληθωρισμός στην αγορά.

- **Windripper Imperial Bow**
  Αυτό το αντικείμενο χρησιμοποιήθηκε με σκοπό την multivariate πρόγνωση τιμών αφού αποτελείται από χαρακτηριστικά τα οποία έχουν συσχετισμό με τη τιμή μεγαλύτερο του 0.1 ή μικρότερο του -0.1. Τέλος, όπως και στο προηγούμενο αντικίμενο υπάρχει μεγάλος συσχετισμός με τη τιμή του χαρακτηριστικού που εξηγεί τη από την ημερήσια ισοτιμία μετατροπής των 2 κυριότερων συναλλαγμάτων.

2.2 Outliers

Για την αφαίρεση των outliers, πειραματιστήκαμε με δύο τεχνικές , την IQR και τη Z-score. Από τις δύο τεχνικές, μόνο η τεχνική IQR μπορούσε να αντιμετωπίσει τους outliers και δοκιμάζοντας διαφορετικές τιμές της καταλήξαμε σε threshold=0.7 για το αντικείμενα Tabula Rasa Simple Robe και threshold = 0.8 για το αντικείμενο Windripper Imperial Bow.

Επίσης χρησιμοποιήσαμε τον περιορισμό days_in_snapshot < 5 το οποίο χαρακτηριστικό δηλώνει πόσες μέρες πήρε στο συγκεκριμένο αντικείμενο να πωληθεί. Αυτό έγινε διότι οι πωλήσεις μετά από 5 ημέρες είναι χαμηλής εμπιστοσύνης.

2.3 Feature Selection

Για την επιλογή χαρακτηριστικών για ανάλυση χρησιμοποιήθηκαν αρκετές διαφορετικές τεχνικές. Οι κυριότερες ήταν :

Correlations

Χρησιμοποιήσαμε τη μέθοδο ‘corr’ της βιβλιοθήκης pandas για να αφαιρέσουμε κάθε χαρακτηριστικό με μικρό συσχετισμό με τη τιμή. Για τη περίπτωση του Tabula Rasa Simple Robe αντικειμένου δεν υπήρχε κάποιο τέτοιο χαρακτηριστικό ενώ για τη περίπτωση του

Windripper Imperial Bow, κρατήσαμε μόνο τα χαρακτηριστικά που είχαν συσχετισμό μεγαλύτερο του 0.1 και μικρότερο του -0.1.

Random Forest

Η μέθοδος Random Forest έχει ενσωματωμένη μέθοδο που αναλύει τη σημασία κάθε χαρακτηριστικού οποτε χρησιμοποιώντας τη πήραμε ως αποτέλεσμα τη σημασία που έχει το κάθε χαρακτηριστικό για τη τιμή του αντικειμένου

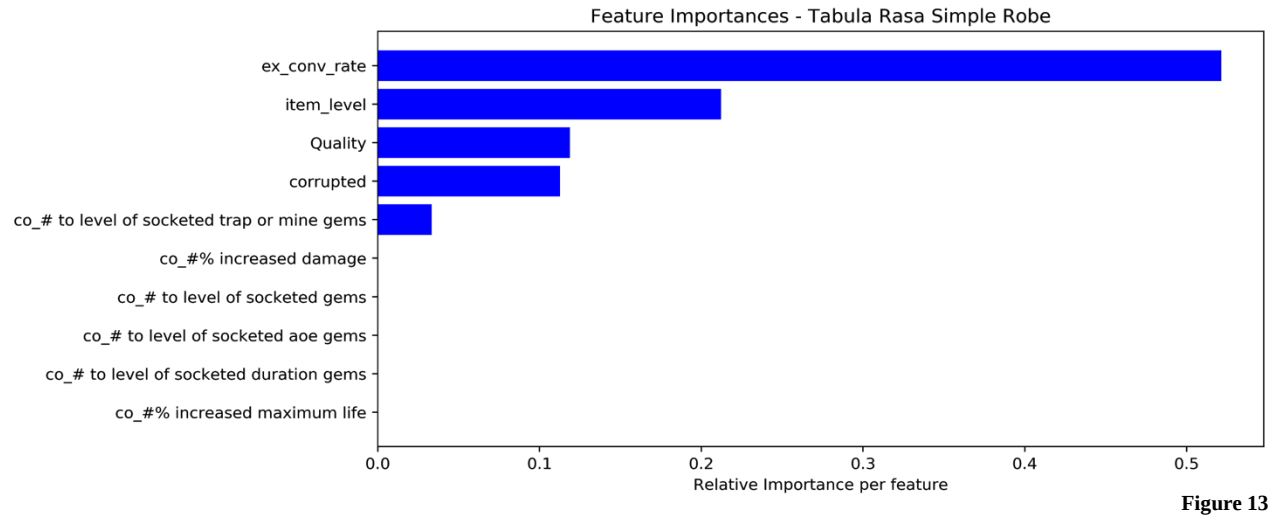


Figure 13

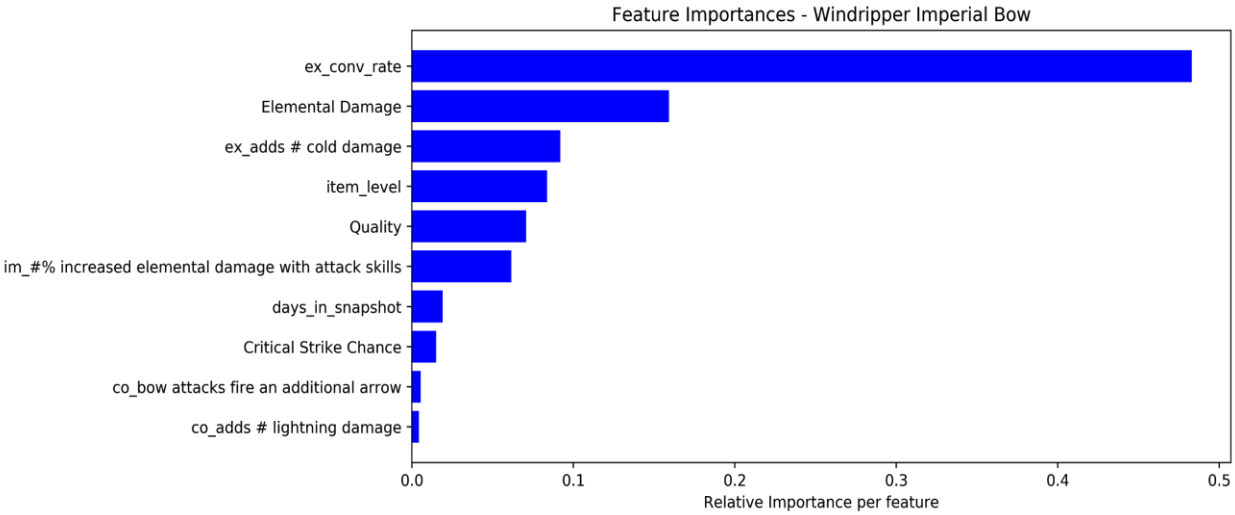


Figure 14

Backwards Feature Elimination

- Αρχικά λαμβάνουμε όλες τις μεταβλητές n που υπάρχουν στο σύνολο δεδομένων μας και εκπαιδεύουμε το μοντέλο χρησιμοποιώντας αυτά
- Στη συνέχεια, υπολογίζουμε την απόδοση του μοντέλου
- Τώρα, υπολογίζουμε την απόδοση του μοντέλου αφού εξαλείψουμε κάθε μεταβλητή (n φορές), δηλαδή αφαιρούμε μία μεταβλητή κάθε φορά και εκπαιδεύουμε το μοντέλο στις υπόλοιπες n-1μεταβλητές
- Προσδιορίζουμε τη μεταβλητή της οποίας η αφαίρεση έχει δημιουργήσει τη μικρότερη (ή όχι) αλλαγή στην απόδοση του μοντέλου και στη συνέχεια την απομάκρυνση της μεταβλητής αυτής
- Επαναλάβετε αυτή τη διαδικασία μέχρι να μην αφαιρεθεί καμία μεταβλητή

Για να το πετύχουμε, χρησιμοποιήσαμε rfe από τη βιβλιοθήκη sklearn.feature_selection με πολλούς αλγορίθμους που συμπεριλάμβαναν, αλλά δεν περιορίζονταν σε Linear Regression, Logistic Regression, Random Forest, Decision Trees κλπ. Από τα παραπάνω τα πιο σταθερά αποτελέσματα ήταν από τα Decision Trees.

2.4 Μετατροπή ανόμοιων(άνισων ή ακανόνιστων) χρονοσειρών σε όμοιων

Ανόμοιες (ή άνισες ή ακανόνιστες) χρονοσειρές είναι μία ακολουθία ζευγών χρόνου και τιμής παρατηρήσεως (tn, Xn) με αυστηρά αυξανόμενους χρόνους παρατήρησης [7]. Σε αντίθεση με τις εξίσου διαχωρισμένες χρονικές σειρές, η απόσταση των χρόνων παρατήρησης δεν είναι σταθερή. Μία κοινή προσέγγιση για την ανάλυση των χρονικά ανεπαρκώς διαχωρισμένων συνόλων είναι η μετατροπή των δεδομένων σε εξίσου διαχωρισμένες παρατηρήσεις με τη χρήση κάποιας μορφής παρεμβολής - συνήθως γραμμικής - και στη συνέχεια εφαρμογής των υπαρχουσών μεθόδων για εξίσου διαχωρισμένα δεδομένα.

Αρχικά τα δεδομένα μας δεν είχαν τα χαρακτηριστικά των χρονοσειρών δεδομένων, δηλαδή οι παρατηρήσεις δεν ήταν μια ακολουθία ίσων χρονικών τμημάτων.

Αφού εισαγάγαμε ένα δείκτη datetime ο οποίος αντιπροσώπευε την ώρα και τη μέρα της πώλησης, σπάσαμε τα δεδομένα μας σε ίσα τμήματα, ρυθμίζοντας μια συχνότητα και παρεμβάλλοντας τις παρατηρήσεις.

Μπορεί να παρατηρηθεί ότι ο δείκτης date_time κάθε παρατήρησης βρίσκεται σε διαφορετική χρονική περίοδο ή με άλλα λόγια σε διαφορετική συχνότητα. Για να μετατρέψουμε το σύνολο δεδομένων σε μια σειρά χρονοσειρών, πρέπει επίσης να αυξήσουμε τη συχνότητα των δειγμάτων μας, όπως από λεπτά σε δευτερόλεπτα, ή να το μειώσουμε για παράδειγμα από μέρες σε μήνες. Αυτές οι διαδικασίες ονομάζονται επαναδειγματοληψία. Και στις δύο περιπτώσεις, τα δεδομένα πρέπει να εφευρεθούν. Εδώ επιλέξαμε να μειώσουμε τα δείγματα από λεπτά σε ώρες.

2.4.1 Interpolation

Σε αυτό το μέρος είχαμε να λύσουμε δύο προβλήματα των ακανόνιστων δεδομένων: κενά τμήματα που εμφανίστηκαν μετά την αναδειγματοληψία, χωρίς να έχουν παρατηρήσεις κατά τη διάρκεια αυτών των χρονικών περιόδων και τμήματα με πολλαπλές τιμές που έπρεπε να καταλάβουμε σε μία τιμή. Γι 'αυτό έπρεπε να παρεμβάλλουμε τα δεδομένα μας. Η παρεμβολή είναι μια μέθοδος κατασκευής νέων σημείων δεδομένων εντός της περιοχής ενός διακριτού συνόλου γνωστών σημείων δεδομένων. Οι διαδικασίες που ακολουθήσαμε ήταν πλήρωση και ισοπέδωση.

Flattening

Στη περίπτωση των πολλαπλών τιμών στην ίδια χρονική περίοδο πρέπει να τις ισοπεδώσουμε. Αυτή η διαδικασία ονομάζεται Flattening. Για να πετύχουμε το σκοπό μας χρησιμοποιήσαμε είτε το μέσο αριθμό, το διάμεσο είτε το mode. Για τα περισσότερα χαρακτηριστικά χρησιμοποιήσαμε το διάμεσο αλλά για τη τιμή χρησιμοποιήσαμε τον μέσο όρο, δεδομένου ότι σε αντίθεση με τα άλλα χαρακτηριστικά, οι τιμές τιμών τιμής κυμαίνονταν πολύ περισσότερο. Η διαφορά μεταξύ των δύο μπορεί να φανεί στο παρακάτω γράφημα.

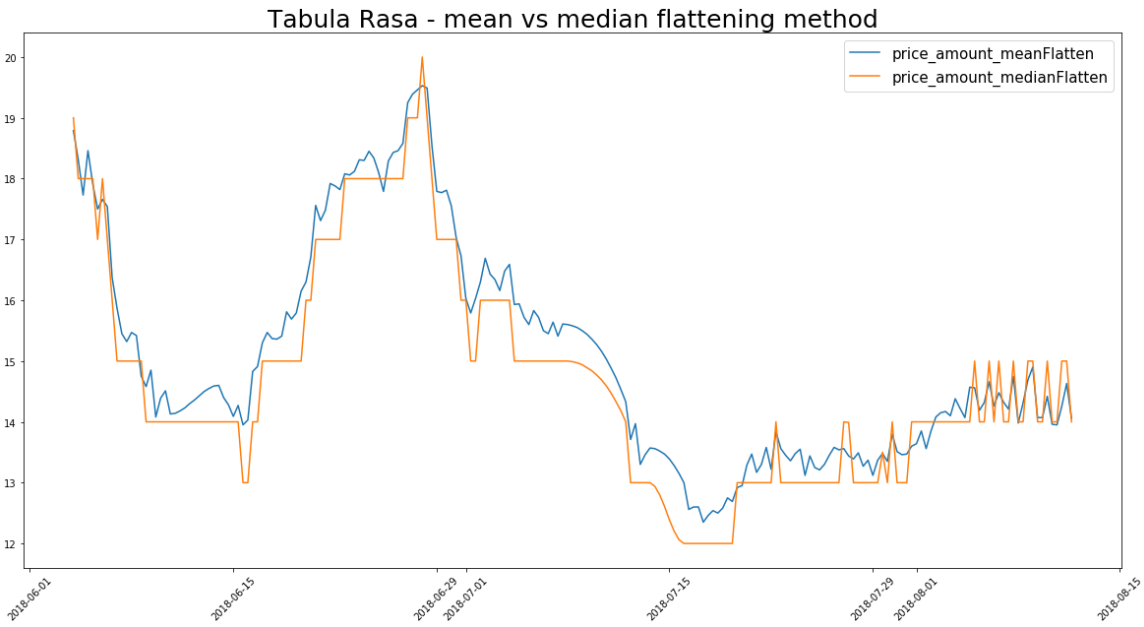


Figure 16

Filling

Στη περίπτωση που λείπουν χρονικές περίοδοι πρέπει να τις συμπληρώσουμε με προεπιλεγμένες τιμές ώστε να μετατρέψουμε τα δεδομένα μας σε σύνολο δεδομένων χρονοσειρών. Αυτό έγινε είτε με με μέθοδο backfilling, forward filling, γραμμικά χρησιμοποιώντας γραμμική κατανομή είτε με τη μέθοδο pchip που συνήθως χρησιμοποιείται για παρεμβολή αριθμητικών δεδομένων μια ομαλή συνεχής λειτουργία. Αποφασίσαμε να χρησιμοποιήσουμε τη μέθοδο pchip, δεδομένου ότι ήταν διατηρημένο και οπτικά ευχάριστο.

Η βασική ιδέα ήταν να προσδιοριστούν οι κλίσεις έτσι ώστε ο παρεμβαλλόμενος να μην ταλαντεύεται πάρα πολύ [10].

2.5 Ανάλυση Χρονικών Σειρών

Για την πρόβλεψη χρονοσειρών με τη χρήση του δικτύου LSTM, προσπαθούμε να παρέχουμε δείκτες δυναμικής των τιμών των μοναδικών αντικειμένων. Σε αυτό το σημείο υπάρχουν σημαντικές παράμετροι, τη σημασία των οποίων πρέπει να εξηγήσουμε:

Χρόνος εισόδου (Lag / Input Timesteps)

Τα παραδοσιακά νευρωνικά δίκτυα παίρνουν κάθε φορά έναν αυτόνομο φορέα δεδομένων και δεν έχουν έννοια της μνήμης στα δεδομένα. Τα δίκτυα LSTM διατηρούν ένα πλαίσιο μνήμης μέσα στον αγωγό τους και έτσι καθίστανται ισχυρά στην αντιμετώπιση των διαδοχικών και χρονικών προβλημάτων χωρίς το ζήτημα του διακύμανσης της διαφυγής που επηρεάζει την απόδοσή τους. Η "μνήμη" που διατηρεί το δίκτυό μας είναι η καθυστέρηση. Για κάθε πρόβλεψη, τροφοδοτούμε μια καθυστέρηση διαδοχικών πληροφοριών στο δίκτυό μας, για να μάθουμε από αυτό.

Ακολουθία πρόβλεψης (Forecasting Sequence)

Εάν η καθυστέρηση είναι η είσοδος, η ακολουθία πρόβλεψης είναι η έξοδος. Πρόκειται για το παράθυρο χρονικής περιόδου που το μοντέλο μας θα προσπαθήσει να προβλέψει.

Διαχωρισμούς σετ εκπαίδευσης / δοκιμής

Ακριβώς όπως και στα ANN, χρησιμοποιώντας το LSTM, πρέπει να χωρίσουμε το σύνολο δεδομένων μας σε ένα σετ εκπαίδευσης και ένα σετ δοκιμών. Υπάρχουν ωστόσο μερικές διαφορές:

● Ο διαχωρισμός σε ένα σύνολο δεδομένων δεν πρέπει να ανακατεύεται. Πρέπει να αποφασίσουμε εκ των προτέρων πόσες χρονικές περιόδους θα χρησιμοποιήσουμε ως εκπαίδευση.

● Το Lag δεν μπορεί να είναι υψηλότερο από το σύνολο δεδομένων εκπαίδευσης.

● Οι διαχωρισμοί εκπαίδευσης και δοκιμών είναι διαδοχικοί και το σετ εκπαίδευσης είναι πριν το σετ δοκιμής.

Ομαλοποίηση (Normalization)

Δεδομένου ότι η τιμή εκκίνησης ενός στοιχείου είναι διαφορετική για κάθε εποχή 3 μηνών, αποφασίσαμε να λάβουμε κάθε παράθυρο δεδομένων εκπαίδευσης / δοκιμών και να το κανονικοποιήσουμε ώστε να αντικατοπτρίζει τις ποσοστιαίες αλλαγές από την αρχή αυτού του παραθύρου.

Εποχές (Epochs)

Μια εποχή είναι απλά ένα πέρασμα προς τα εμπρός και ένα προς τα πίσω πέρασμα όλων των παραδειγμάτων εκπαίδευσης. Η εκπαίδευση για περισσότερες εποχές καθιστά το μοντέλο μας καλύτερο αλλά και πιο επιρρεπές σε υπερφόρτωση(overfitting).

Μέγεθος παρτίδας (Batch Size)

Το μέγεθος παρτίδας είναι ο αριθμός των εκπαιδευτικών παραδειγμάτων σε ένα μπροστινό / οπίσθιο πέρασμα. Όσο υψηλότερο είναι το μέγεθος της παρτίδας, τόσο περισσότερος χώρος μνήμης χρειάζεται. Έχει παρατηρηθεί στην πράξη ότι όταν χρησιμοποιείται μια μεγαλύτερη παρτίδα υπάρχει σημαντική υποβάθμιση στην ποιότητα του μοντέλου.

Απόσυρση (Dropout)

Η απόρριψη είναι μια μέθοδος τακτοποίησης που προσεγγίζει την κατάρτιση ενός μεγάλου αριθμού νευρωνικών δικτύων με διαφορετικές παράλληλες αρχιτεκτονικές. Κατά τη διάρκεια της εκπαίδευσης, κάποιοι τύποι εξόδων στρώματος αγνοούνται τυχαία ή "αποχωρούν. Για παράδειγμα, ένα στρώμα Dropout με ρυθμό 0,2 έχει 20% πιθανότητα να ρίξει κάθε νευρώνα.

Συναρτήσεις Απώλειας(Loss Functions)

Μια λειτουργία απώλειας χρησιμοποιείται για τη βελτιστοποίηση των τιμών παραμέτρων σε ένα μοντέλο νευρωνικού δικτύου. Οι λειτουργίες απώλειας αντιστοιχούν σε ένα σύνολο τιμών παραμέτρων για το δίκτυο σε μια κλιμακωτή τιμή που υποδεικνύει πόσο καλά η παράμετρος αυτή ολοκληρώνει την εργασία που το δίκτυο προορίζεται να κάνει. Είναι ουσιαστικά ένας μαθηματικός τρόπος μέτρησης πόσο λανθασμένες είναι οι προβλέψεις μας. Απώλεια είναι αυτό το μέτρο.

Βελτιστοποιητής (Optimizer)

Κατά τη διάρκεια της διαδικασίας κατάρτισης, αλλάζουμε τις παραμέτρους του μοντέλου μας για να προσπαθήσουμε να ελαχιστοποιήσουμε τη λειτουργία απώλειας και να κάνουμε καλύτερες, ακριβέστερες προβλέψεις. Οι βελτιστοποιητές συνδυάζουν τη συνάρτηση απώλειας και τις παραμέτρους του μοντέλου [11].

Συνάρτηση ενεργοποίησης

Η συνάρτηση ενεργοποίησης ενός κόμβου καθορίζει την έξοδο αυτού του κόμβου ή "νευρώνα", δεδομένης μιας εισόδου ή συνόλου εισόδων. Αυτή η έξοδος χρησιμοποιείται στη συνέχεια ως είσοδος για τον επόμενο κόμβο και ούτω καθεξής έως ότου βρεθεί η επιθυμητή λύση στο αρχικό πρόβλημα. Αν δεν εφαρμόσουμε μια συνάρτηση ενεργοποίησης τότε το σήμα εξόδου θα ήταν απλά μια απλή γραμμική συνάρτηση.

HyperParameters

Ένα hyperparameter είναι μια παράμετρος της οποίας η τιμή έχει οριστεί πριν αρχίσει η διαδικασία εκμάθησης. Αντίθετα, οι τιμές άλλων παραμέτρων προέρχονται από την

εκπαίδευση. Διαφορετικοί αλγόριθμοι κατάρτισης μοντέλων απαιτούν διαφορετικές hyperparameters οι οποίες θα βελτιστοποιηθούν αργότερα χρησιμοποιώντας μια μέθοδο που ονομάζεται Grid Search. Οι υπερπ hyperparameters μας ήταν εδώ

- Χρονοδιακόπτες εισόδου
- Διαχωρισμός τρένου / δοκιμής
- Κατώφλι υψηλού ποσοτικού ορίου IQR
- Προβλεπόμενη ακολουθία
- Εποχές
- Απώλεια
- Ποσοστό εκμάθησης για SGD και Dense
- Νευρώνες διαφορετικών στρωμάτων LSTM
- Συνάρτηση ενεργοποίησης (activation function)
- Ποσοστό απόρριψης(dropout rate)
- Optimizer

2.6 Univariate Unistep εναντίον Univariate Multistep χρησιμοποιώντας LSTM

Όπως αναφέρθηκε προηγουμένως, χρησιμοποιήσαμε το αντικείμενο "Tabula Rasa" επειδή δεν αποτελούνταν από σημαντικά χαρακτηριστικά που θα μπορούσαν να επηρεάσουν την τιμή του εκτός από την ημερήσια ισοτιμία μετατροπής. Σε αυτό, εκτελούμε 2 τύπους αναλύσεων, unistep και multi-step. Το Unistep ή το point-to-point είναι η πρόβλεψη μιας μόνο χρονικής περιόδου μπροστά από το χρόνο, σχεδιάζοντας αυτή την πρόβλεψη και στη συνέχεια παίρνοντας το επόμενο παράθυρο μαζί με τα πλήρη δεδομένα δοκιμών και προβλέποντας το επόμενο σημείο για άλλη μια φορά.

Η πρόβλεψη multistep θα γίνει σε δύο μέρη. Το πρώτο θα αποτελείται από μια πλήρη πρόβλεψη ακολουθίας, αρχικοποιώντας ένα παράθυρο εκπαίδευσης και εκπαιδεύοντας το μοντέλο μας σε αυτό. Το μοντέλο τότε προβλέπει το επόμενο σημείο και μετατοπίζουμε το παράθυρο κατά μία χρονική περίοδο προς τα δεξιά, ακριβώς όπως η μέθοδος point-by-point. Η διαφορά είναι τότε η επόμενη πρόβλεψη γίνεται χρησιμοποιώντας τα δεδομένα που προβλέψαμε στην προηγούμενη πρόβλεψη. Στη δεύτερη πρόβλεψη έχουμε ένα προβλεπόμενο σημείο δεδομένων, στη τρίτη 2 προβλεπόμενα σημεία δεδομένων και ούτω καθεξής [9]. Αφού προβλέψουμε σημεία ίσα με την ακολουθία εισόδου, η επόμενη πρόβλεψή μας αποτελείται μόνο από τα προβλεπόμενα σημεία δεδομένων. Αυτό μας επιτρέπει να χρησιμοποιήσουμε το μοντέλο για να προβλέψουμε πολλά βήματα μπροστά, αλλά καθώς προβλέπει τις προβλέψεις που στη συνέχεια μπορούν να βασιστούν στις προβλέψεις, αυτό θα οδηγήσει σε αυξημένο ποσοστό σφάλματος των μετέπειτα προβλέψεων.

Το δεύτερο μέρος είναι μια πρόβλεψη πολλαπλών ακολουθιών(multistep). Αυτό είναι ένα μείγμα της πρόβλεψης πλήρους ακολουθίας υπό την έννοια ότι αρχικοποιεί το παράθυρο δοκιμής με δεδομένα δοκιμών, προβλέπει το επόμενο σημείο πάνω από αυτό και κάνει ένα νέο παράθυρο με το επόμενο σημείο. Ωστόσο, μόλις φτάσει σε ένα σημείο όπου το παράθυρο εισαγωγής αποτελείται από προηγούμενες προβλέψεις, σταματά, μετακινεί προς

τα εμπρός ένα πλήρες μήκος παραθύρου πρόβλεψης, επαναφέρει το παράθυρο με τα πραγματικά δεδομένα δοκιμής και ξεκινά ξανά τη διαδικασία. Στην ουσία αυτό δίνει πολλαπλές τάσεις όπως οι προβλέψεις πάνω στα δεδομένα των δοκιμών, προκειμένου να αναλυθεί πόσο καλά το μοντέλο μπορεί να πάρει τις τάσεις της μελλοντικής ορμής [17].

Όλες οι διαδικασίες δημιουργήθηκαν χρησιμοποιώντας τις βιβλιοθήκες Tensorflow και Keras καθώς επίσης και πολλαπλές βιβλιοθήκες του sklearn[12].

Figure 17

Με RMSE = 0,01855 η πρόβλεψη από σημείο σε σημείο είναι μια αρκετά ακριβής αναπαράσταση του τι πρόκειται να συμβεί στην επόμενη χρονική περίοδο.

Αντίθετα, η πρόβλεψη πλήρους ακολουθίας μας προσπαθεί να προβλέψει τι θα συμβεί στις πρώτες χρονικές περιόδους, αλλά αργότερα, βασιζόμενη αποκλειστικά σε άλλες προβλέψεις, αποτυγχάνει να προβλέψει και το RMSE αυξάνεται δραματικά.

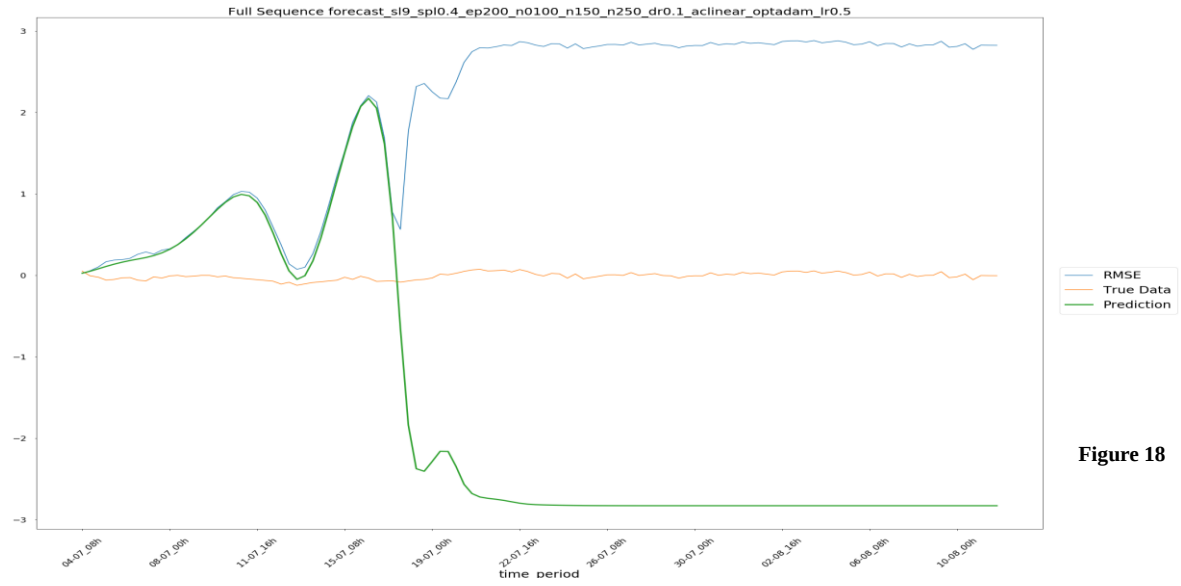


Figure 18

Τέλος, βλέπουμε τη πλοκή πρόγνωσης πολλών σταδίων. Με σετ εκπαίδευσης 0.4 (ή περίπου 30 ημέρες) μπορεί να προβλέψει τις τάσεις (και μερικές φορές το εύρος των τάσεων) για μια μεγάλη πλειοψηφία των χρονικών περιόδων. Αν και δεν είναι τέλειο, αποτελεί καλή ένδειξη της χρησιμότητας του μοντέλου.

Grid Search

Για να βελτιώσουμε τα αποτελέσματά μας, πρέπει να συντονίσουμε τις hyperparameters μας. Για το λόγο αυτό χρησιμοποιήσαμε Grid Search. Η μέθοδος Grid Search χρησιμοποιείται για την εύρεση των βέλτιστων υπερπαραμέτρων ενός μοντέλου που οδηγεί στις πιο «ακριβείς» προβλέψεις. Η αναζήτηση πλέγματος δημιουργεί ένα μοντέλο για κάθε συνδυασμό υπερπαραμέτρων που καθορίζεται και αξιολογεί κάθε μοντέλο. Για προβλέψεις από σημείο σε σημείο, υπολογίσαμε το RMSE των προβλέψεων για κάθε μοντέλο και επιλέξαμε το ένα με το χαμηλότερο RMSE, ενώ για τις προβλέψεις πλήρους ακολουθίας όλα τα αποτελέσματά μας ήταν κακά και βασιζόμενα σε πολλαπλά σφάλματα δεν δίνουν καλή αντιπροσώπευση των μελλοντικών τιμών.

Οι παράμετροι που βελτιστοποιήσαμε και τα εύρη τιμών τους φαίνονται παρακάτω. Είναι σημαντικό να σημειωθεί ότι η χρήση ενός sgd optimizer έδωσε πολύ χειρότερα αποτελέσματα απ 'ό, τι η adam για κάθε learning rate που τον ξεκινήσαμε και για αυτό το λόγο σταματήσαμε τη δοκιμή νωρίς, διότι θα διπλασίαζε το μέγεθος των περιπτώσεών μας.

Για τις προβλέψεις πολλών σταδίων δεν μπορούσαμε να συγκρίνουμε διαφορετικά μήκη

```
grid = ParameterGrid({"sequence_length": [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
                      "train_test_split": [0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8],
                      "epochs": [100,200],
                     "optimizer":['adam','sgd'],
                     "learning_rate":[0.01,0.05,0.1,0.2],
                     "layer0_neurons":[100,200],
                     "layer1_neurons":[50,100],
                     "dropout_rate":[0.1,0.2,0.3,0.4],
                     "layer3_activation_function":['linear','sigmoid']
                     })
```

αλληλουχίας και ανάλογα με το ποσοστό των σετ εκπαίδευσης δοκιμής κάθε προβλεπόμενη τάση ήταν σε διαφορετικό χρονικό παράθυρο και είχε διαφορετικό μήκος. Για το λόγο αυτό, οι μετρήσεις rmse, mae ή mape δεν ήταν αντιπροσωπευτικές της απόδοσης του μοντέλου και της ικανότητάς του να προβλέψει τάσεις. Η απόφαση για το καλύτερο μοντέλο ή μοντέλα έγινε αποκλειστικά με την παρατήρηση κάθε διαφορετικής γραφικής παράστασης αποτελεσμάτων.

Γενικά για διαχωρισμούς υψηλότερους από 0,5, το μοντέλο έκανε "overfit" και οι τάσεις δεν είχαν θετικές ή αρνητικές κορυφές. Η αύξηση του ποσοστού dropout ή η προσθήκη περισσότερων επιπέδων εγκατάλειψης δεν είχε καμία αλλαγή σε αυτό.

Για μήκος αλληλουχίας υψηλότερο από 10, οι προβλέψεις είχαν μεγάλο σφάλμα καθώς κάθε επόμενο σημείο πρόβλεψης βασίστηκε σε περισσότερα σημεία προβλέψεων.

Ένα καλό σημείο για 8 ώρες και 12 ώρες χρονικές περιόδους ήταν το μήκος ακολουθίας 6. Το μοντέλο μας θα μπορούσε σχεδόν πάντα να προβλέψει εάν η τιμή του αντικειμένου θα ανέβαινε, να πέσει ή ξαφνικές αλλαγές, καθώς και το εύρος αυτών των τάσεων.

Μια παράμετρος που ήταν επίσης πολύ σημαντική ήταν η συναρτήσεις ενεργοποίησης. Εφόσον κανονικοποιήσαμε τις μεταβλητές μας και τα προβλεπόμενα ποσοστά αλλαγής, δεν μπορούσε να χρησιμοποιηθούν συναρτήσεις ενεργοποίησης που δεν μπορούσαν να παράγουν αρνητικά αποτελέσματα. Από τις συναρτήσεις tanh και linear activation, η linear έδωσε τα καλύτερα αποτελέσματα.

Τέλος, οι εποχές και οι νευρώνες κάθε στρώματος ήταν πολύ σημαντικό να διατηρηθούν σε μικρούς αριθμούς, 200 εποχές ως μέγιστη τιμή και 100 και 50 νευρώνες στο πρώτο και το δεύτερο στρώμα, καθώς οι περισσότερες εποχές ή περισσότεροι νευρώνες σε ένα στρώμα προκαλούσαν overfitting του μοντέλου. Αυτό σημαίνει ότι οι προβλέψεις μας δεν θα μπορούσαν να έχουν κορυφές είτε αρνητικές είτε θετικές.

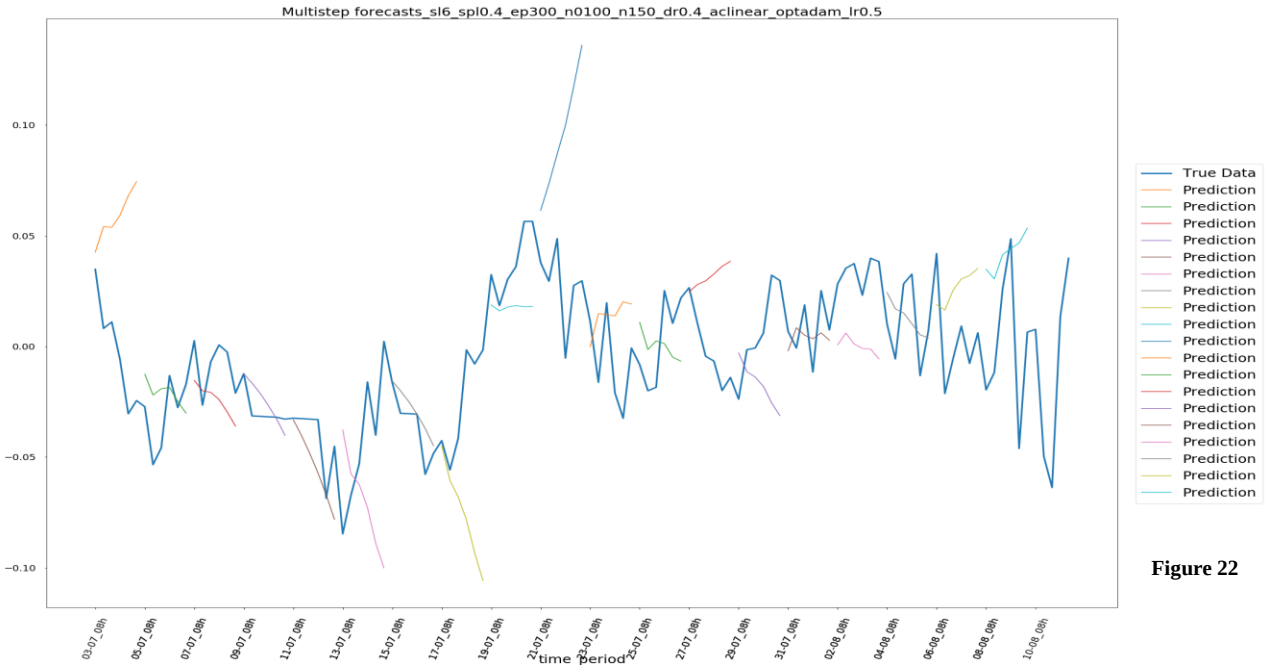


Figure 22

Μια παράμετρος που έπαιξε μεγάλο ρόλο ήταν ο ρυθμός του στρώματος εγκατάλειψης(dropout rate) καθώς και πόσα dropout επίπεδα υπήρχαν στο νευρωνικό μας δίκτυο. Χρησιμοποιώντας 2 επίπεδα dropout της τάξης του 0,1 και 0,2, μας έδωσαν τα καλύτερα αποτελέσματα καθώς μείωσαν το εύρος των προβλεπόμενων τάσεων και επέτρεψαν στο μοντέλο να προβλέψει περισσότερες κορυφές τιμής.

Figure 21

2.7 Multivariate Unistep εναντίων Multivariate Multistep ανάλυση δεδομένων χρονοσειράς με LSTM

Για την ανάλυση multivariate διαλέξαμε χαρακτηριστικά με συσχετισμο με τιμή μεγαλύτερο του 0.2 και που εμφανιζόντουσαν σε μεγάλο ποσοστό. Για αυτό το λόγο κάθε χαρακτηριστικό του τύπου «co_» έπρεπε να αφαιρεθεί.

Ακολουθώντας το παράδειγμα με univariate, κάναμε πρώτα προβλέψεις από σημείο σε σημείο και στη συνέχεια προσπαθήσαμε να προβλέψουμε πολλαπλές τάσεις. Αυτή τη φορά πραγματοποιήσαμε πειραματισμούς με διάφορα χαρακτηριστικά για να δείξουμε πώς η αφαίρεση των χαρακτηριστικών θα μπορούσε να επηρεάσει το συνολικό κέρδος πληροφορίας που αντιπροσωπεύσαμε υπολογίζοντας την τιμή RMSE [14].

Εκτελούμε το ίδιο μοντέλο χρησιμοποιώντας 5, 6, 7, 8 και 9 χαρακτηριστικά.

Το βέλτιστο νευρικό δίκτυο συνίσταται από 3 στρώματα LSTM με 100, 50 και 50 νευρώνες αντιστοίχως, καθώς και 1 στρώμα dropout που ρυθμίστηκε σε 0,3. Το τελευταίο στρώμα ήταν ένα dense στρώμα με γραμμική συνάρτηση ενεργοποίησης.

Οι hyperparameters και τα εύρη τιμών τους που δοκιμάσαμε, καθώς και τα αποτελέσματα, μπορούν να δουν παρακάτω.

Τα αποτελέσματα που πήραμε ήταν καλύτερα (μικρότερα RMSE) όταν χρησιμοποιήθηκαν περισσότερα χαρακτηριστικά, καθώς υπήρχε λιγότερη απώλεια πληροφοριών. Επίσης, μικρότερο μήκος αλληλουχίας είχε ως αποτέλεσμα μικρότερο rmse και λιγότερες εποχές είχαν ως αποτέλεσμα μικρότερο overfitting και κατά συνέπεια καλύτερη τιμή rmse.

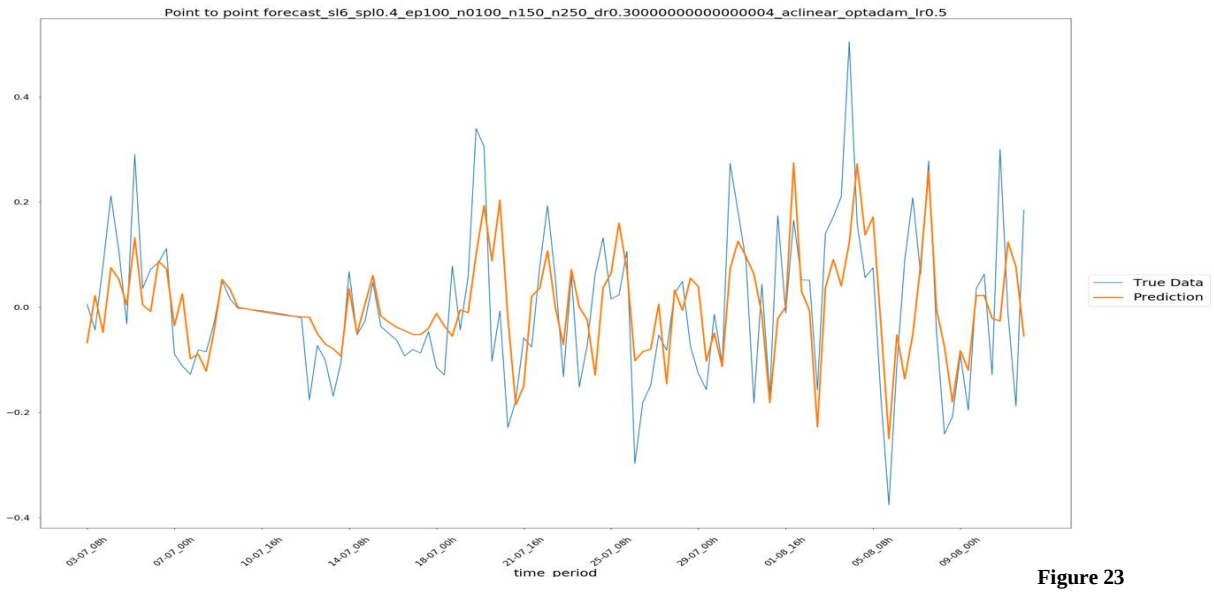


Figure 23

Table 8

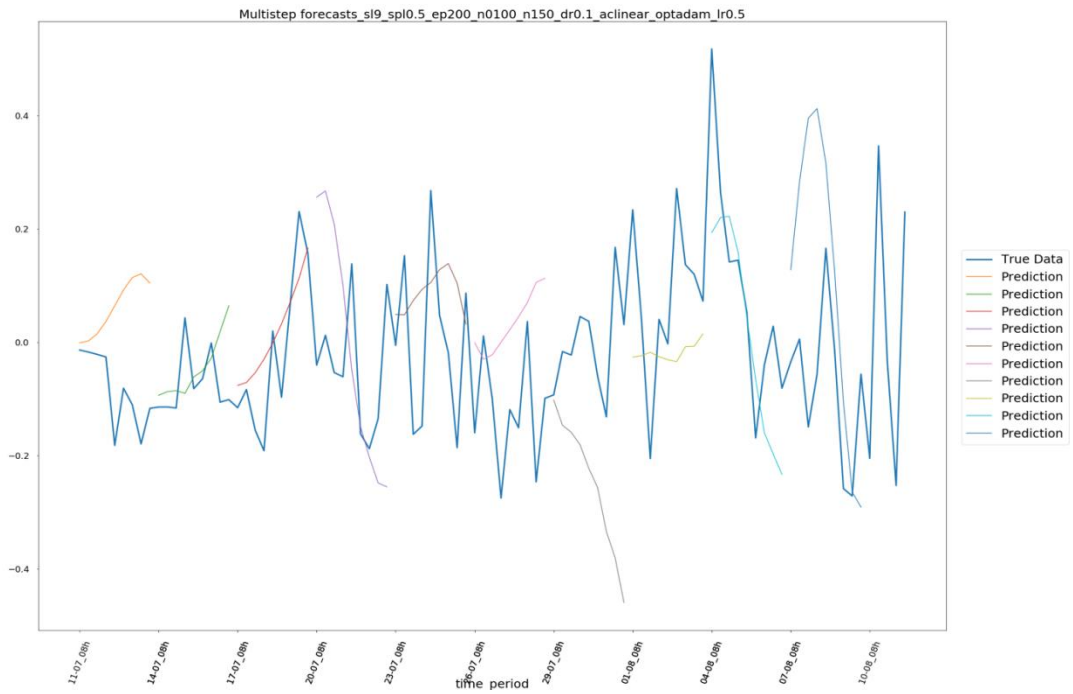
```
{"sequence_length": [6, 7, 8, 9, 10, 11, 12, 13, 14],
 "train_test_split": [0.3,0.4,0.5],
 "epochs": [100,200,300],
"optimizer":['adam'],
"layer_0.neurons":[100,200,300],
"layer_1.neurons":[50,100,150],
"layer_2.neurons":[50,100,150],
"layer_3.rate":[0.1, 0.2, 0.3],
"layer_4.activation_function":['linear']
})
```

Table 8: Results grid after fitting models produced using hyperparameters – Uniques

dropout_rate	epochs	learning_rate	neurons0	neurons1	neurons2	no_of_features	optimizer	rmse	sequence_length	train_test_split
0.3	100	0.5	100	50	50	9	adam	0.111247	6	0.4
0.2	100	0.5	100	50	50	9	adam	0.111601	6	0.4
0.2	200	0.5	100	50	50	5	adam	0.112929	6	0.3
				. . .						
0.3	300	0.5	100	50	50	5	adam	0.206421	10	0.3
0.2	300	0.5	100	50	50	6	adam	0.208711	14	0.3
0.3	300	0.5	100	50	50	6	adam	0.217636	14	0.3
0.2	300	0.5	100	50	50	5	adam	0.235387	11	0.3
0.1	300	0.5	100	50	50	5	adam	0.293465	11	0.3

Το ίδιο μπορεί να παρατηρηθεί για την πρόβλεψη σε πολλά στάδια [15]. Όταν ο διαχωρισμός εκπαίδευσης / δοκιμής διατηρείται σε 0,4 φαίνεται να είναι η βέλτιστη τιμή για την πρόβλεψη τάσεων σε χαμηλά μήκη ακολουθίας. Όσον αφορά τις εποχές, με 100 ή λιγότερες εποχές, το μοντέλο μπορεί να προβλέψει με ακρίβεια τις τάσεις. Επίσης, ο ρυθμός εγκατάλειψης 0,2 ή 0,3 επιτρέπει στο μοντέλο να προβλέπει πιο έντονα κατώτατα όρια. Οι νευρώνες δεν παίζουν τόσο μεγάλο ρόλο, αλλά στις υψηλότερες τιμές προβλέψαμε λιγότερες κορυφές.

Figure 24



Συνολικά, όπως φαίνεται και στις παραπάνω πλοκές, για τα χαρακτηριστικά 5 και 9 αντίστοιχα, έχοντας περισσότερα χαρακτηριστικά, επέτρεψαν στο μοντέλο μας να προβλέψει τις καλύτερες τάσεις και τα πλάτη τους [18].

2.8 Συμπέρασμα

Ενώ η διαδικασία πρόβλεψης φάνηκε αρχικά δύσκολη, καταφέραμε να έχουμε πολύ καλό αποτέλεσμα σε προβλέψεις για μοναδικά στοιχεία καθώς και καλές προβλέψεις πολλών σταδίων. Παρόλο που η διαδικασία αξιολόγησης της πρόβλεψης πολλαπλών σταδίων δεν ήταν βέλτιστη, θα μπορούσαμε να βρούμε συγκεκριμένους υπερπαραμετρικούς παράγοντες οι οποίοι λειτουργούσαν καλύτερα παρατηρώντας τις πλοκές και εξάγοντας τα συμπεράσματά τους.Έχοντας ολοκληρωμένες παρατηρήσεις για κάθε μοναδικό αντικείμενο του παιχνιδιού, θα μπορούσαμε να δημιουργήσουμε μοντέλα για καθένα και στη συνέχεια να αυτοματοποιήσουμε τη διαδικασία πρόβλεψης των τιμών τους. Μία ιδέα για την επιλογή του καλύτερου μοντέλου μετά από βελτιστοποίηση hyperparameters θα ήταν να αξιολογηθεί κάθε μοντέλο σύμφωνα με ένα συγκεκριμένο σκοπό π.χ. την πρόβλεψη των αποδόσεων της αγοράς στο εμπόριο και τη χρήση του κέρδους και της ζημίας (PnL) ως μετρήσεις.

Κεφάλαιο 3: Ανάλυση δεδομένων Rares - Πρόβλεψη τιμών

3.1 Γενικά

Παρατηρώντας τα Rare αντικείμενα με την πρώτη ματιά, το πρόβλημα ήταν ο μεταβλητός αριθμός χαρακτηριστικών για κάθε παρατήρηση. Λαμβάνοντας υπόψη ότι κάθε Rare αντικείμενο, που σημαίνει κάθε παρατήρηση, δεν μπορούσε να έχει περισσότερα από 12 χαρακτηριστικά και ο αριθμός των παρατηρήσεων ήταν περίπου 580.000 (πριν αφαιρεθούν τα αποθέματα), έγινε φανερό ότι ο αριθμός διαφορετικών συνδυασμών των 90+ χαρακτηριστικών μας, ήταν πολύ μεγαλύτερος των παρατηρήσεων. Κάποιοι συνδυασμοί δεν εμφανίστηκαν καθόλου και κάποιοι άλλοι εμφανίστηκαν μόνο μία φορά.

Προσπαθήσαμε να αντιμετωπίσουμε το πρόβλημα με διάφορους τρόπους, είτε ως πρόβλημα regression χρησιμοποιώντας διαφορετικούς αλγορίθμους ή διαφορετικά νευρωνικά δίκτυα είτε κάνοντας το ίδιο πράγμα, αλλά ως πρόβλημα ταξινόμησης, χωρίζοντας τις διαφορετικές τιμές αντικειμένων σε κομμάτια εύρους τιμών.

3.2 Το σύνολο δεδομένων

Ένα σημαντικό πρόβλημα με το σύνολο δεδομένων ήταν ο μεταβλητός αριθμός χαρακτηριστικών για κάθε παρατήρηση. Κάθε παρατήρηση, που σημαίνει κάθε στοιχείο, δεν μπορεί να έχει περισσότερα από ένα ορισμένο αριθμό γενικευμένων χαρακτηριστικών. Αυτό σήμαινε ότι καθένα από αυτά τα χαρακτηριστικά δεν θα μπορούσε να εμφανιστεί σε όλες τις παρατηρήσεις. Ορισμένα εμφανίστηκαν πολλές φορές (για παράδειγμα, το 'ex_ # to maximum life' εμφανίζεται στο 75,27% των παρατηρήσεών μας) και κάποια άλλα χαρακτηριστικά εμφανίζονται πολύ λιγότερα (για παράδειγμα "ex_gain onslaught για # δευτερόλεπτα όταν χτυπήσουν" εμφανίζονται στο 0.001891% [19].

Δοκιμάσαμε διαφορετικές στρατηγικές για να αντιμετωπίσουμε το παραπάνω πρόβλημα. Είτε γεμίσαμε τις τιμές που έλειπαν από καθέ στήλη είτε αφαιρέσαμε χαρακτηριστικά τα

Armour	50.422927
Energy Shield	46.267913
Evasion Rating	42.519067
ex_#% increased stun and block recovery	42.501362
ex_#% to cold resistance	40.985666
ex_#% to fire resistance	40.960399
ex_#% to lightning resistance	40.260821
⋮	
co_#% reduced fire damage taken	0.013063
ex_#% chance to avoid cold damage when hit	0.011860
ex_#% chance to avoid fire damage when hit	0.005328
ex_#% chance to dodge spell hits	0.005157
co_#% to all maximum resistances	0.004469
ex_#% increased area of effect	0.002235
ex_gain onslaught for # seconds when hit	0.001891

Table 9

οποια δεν εμφανίζονταν σε μεγάλο ποσοστό. Τελικά αποδείχτηκε πως αφήνοντας τα χαρακτηριστικά ως έχουν ήταν η καλύτερη στρατηγική.

Dependent variable y (price_amount) distribution

Ένα άλλο πρόβλημα ήταν ο αριθμός των παρατηρήσεων που σχετίζονται με κάθε διαφορετική τιμή. Η συγκέντρωση των παρατηρήσεων μας σε διαφορετικές σειρές τιμών μας έδωσε τα ακόλουθα αποτελέσματα.

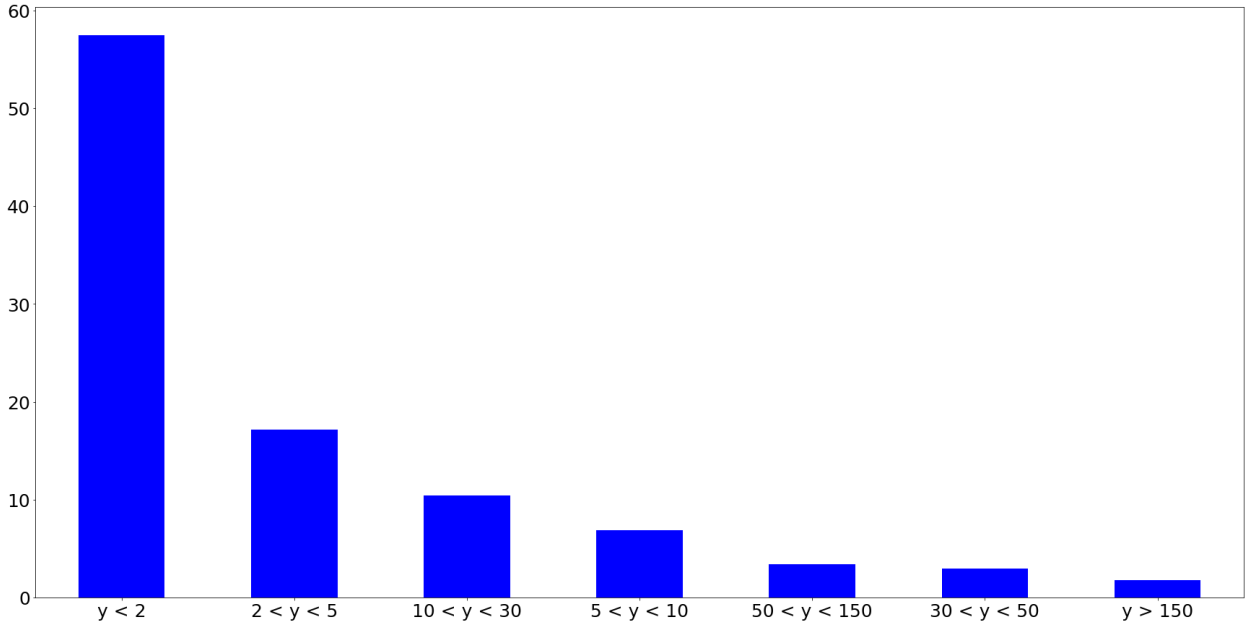


Figure 26

Είναι σαφές ότι έχουμε πολύ περισσότερες παρατηρήσεις με τιμή <2. Για το λόγο αυτό, μια ιδέα ήταν να μετατρέψουμε τη μεταβλητή-στόχο κάθε παρατήρησης, τιμής, για να ακολουθήσουμε μια Gaussian κατανομή χρησιμοποιώντας λογάριθμο. Η μετατροπή της τιμής επιστρέφει την ακόλουθη διανομή:

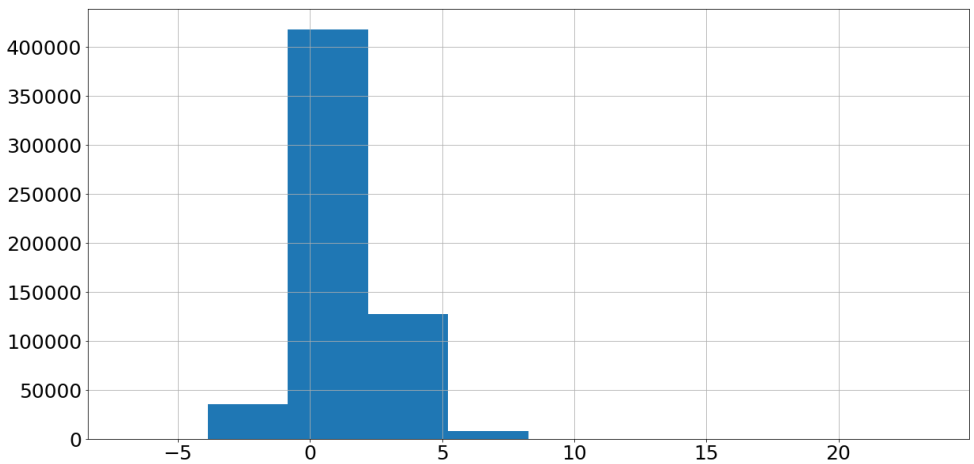


Figure 27

Αφού εκτελέσουμε ένα βασικό νευρωνικό δίκτυο, πριν πάρουμε τον φυσικό λογάριθμο των τιμών και μετά, έχουμε τα ακόλουθα αποτελέσματα:

Before np.log

	<20%	<30%	<40%	<50%	>50%	total%
(0,1]	2132	936	890	1000	16744	14.136946
(1,5]	1451	730	838	805	2113	36.735725
(5,10]	308	165	269	302	1490	18.666140
(10,30]	1334	529	434	352	1883	41.107679
(30,50]	376	181	157	164	479	41.046426
(50,150]	424	190	211	147	631	38.303182
(150,nan]	75	31	30	35	109	37.857143

After np.log

	<20%	<30%	<40%	<50%	>50%	<50% error
(0,1]	6204	2012	1472	995	6586	61.862297
(1,5]	1712	1001	1271	1509	4886	52.924174
(5,10]	93	58	77	103	669	33.100000
(10,30]	1568	640	483	392	2362	56.620753
(30,50]	565	265	255	225	596	68.730325
(50,150]	412	245	244	199	536	67.237164
(150,nan]	39	34	27	44	167	46.302251

Table 10

3.3 Κατασκευή χαρακτηριστικών

Ένα μεγάλο μέρος του προβλήματος ήταν η χρήση των γνώσεων για τα δεδομένα για τη δημιουργία νέων χαρακτηριστικών. Αυτό ήταν θεμελιώδες για την επίτευξη καλύτερων αποτελεσμάτων για τους αλγόριθμους μας. Αυτά τα χαρακτηριστικά ήταν:

- #_of_ele_resistances :
- #_of_resistances
- total_ele_resistance
- total_resistance

Μερική Εξάρτηση (Partial Dependence)
Η γραφική παράσταση μερικής εξάρτησης [20] δείχνει το οριακό αποτέλεσμα που έχουν ένα ή δύο χαρακτηριστικά στην προβλεπόμενη έκβαση ενός μοντέλου μηχανικής μάθησης. Μια γραφική απεικόνιση μερικής εξάρτησης μπορεί να δείξει εάν η σχέση μεταξύ του στόχου και ενός χαρακτηριστικού είναι γραμμική, μονότονη ή πιο περίπλοκη.

Χρησιμοποιώντας μερική εξάρτηση αναλύσαμε την εξάρτηση των τιμών κάθε χαρακτηριστικού από τη μεταβλητή στόχου και δημιουργήσαμε δύο νέα χαρακτηριστικά, no_of_good_features και no_of_bad_features. Σύμφωνα με τη γνώση πάνω στο τομέα, ένα στοιχείο που έχει μια σειρά καλών χαρακτηριστικών πάνω από ένα ορισμένο όριο θεωρείται πολύτιμο και ένα στοιχείο που έχει μια σειρά κακών χαρακτηριστικών πάνω από ένα συγκεκριμένο όριο θεωρείται άχρηστο. Αφού κατασκευάσουμε αυτά τα 2 νέα χαρακτηριστικά, χρησιμοποιούμε και πάλι μερική εξάρτηση για να βεβαιωθούμε πως η αρχική υπόθεσή μας ήταν σωστή.

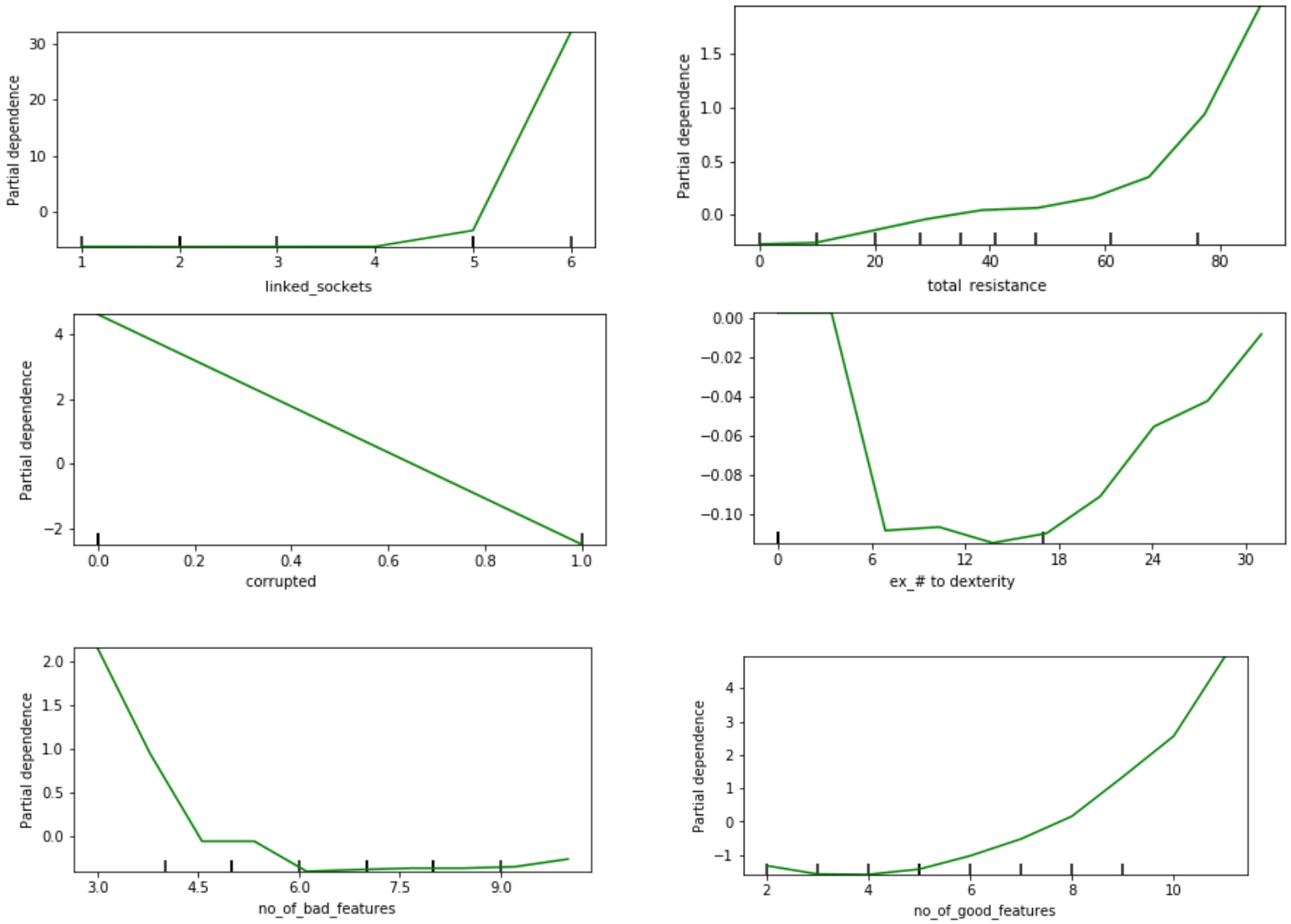


Figure 28

3.4 Outliers

Για τους outliers χρησιμοποιήσαμε IQR και Z-score μεθοδους όπως και πριν. Επειδή αφαιρώντας outliers αφαιρούμε περίπου το 1/3 από τις παρατηρήσεις μας με τιμή μεγαλύτερη του 150 , αποφασίσαμε να αφαιρέσουμε τελικα τις ακραίες τιμές. Χρησιμοποιώντας ένα από τα νευρωνικά μας δίκτυα, έχοντας τρέξει το μοντέλο με και χωρίς outliers, έχουμε ακρίβεια 60,2% για τα στοιχεία πρόβλεψης με τιμή τιμής> 150 με outliers, ενώ μόνο 50,7% χωρις outliers.

	price >150
with outliers	0.65%
without outliers	1.80%

Table 11

3.5 Προσέγγιση – Regression εναντίων Classification

Ο αρχικός μας στόχος ήταν να δημιουργήσουμε ένα μοντέλο regression(regression), το οποίο θα μπορούσε να τιμολογήσει στοιχεία μέσα σε ένα σχετικά μικρό περιθώριο από τις πραγματικές τους τιμές. Χωρίς να χρησιμοποιούμε τις λογαριθμικές αξίες της τιμής τιμής, τα αποτελέσματά μας δεν ήταν τόσο καλά, αφού δεν μπορούσαμε να τιμολογήσουμε στοιχεία με ακρίβεια 42% ή παραπάνω. Μετά το λογαριθμικό μετασχηματισμό τα αποτελέσματά μας βελτιώθηκαν, αλλά σε κάποιες περιπτώσεις ήταν ακόμα πολύ μικρά.

Αυτό μας έδωσε την ιδέα να χωρίσουμε τις τιμές μας σε κάδους τιμών και αντί να προσπαθήσουμε να χτίσουμε ένα μοντέλο regression, να προσπαθήσουμε να ταξινομήσουμε σε ποιο εύρος τιμής κάθε αντικείμενο ανήκε. Χρησιμοποιώντας έναν δυναμικό αλγόριθμο, ο οποίος χρησιμοποίησε το ελάχιστο ποσοστό κάθε εύρους τιμών και το μέγιστο εύρος από τη χαμηλότερη και την υψηλότερη τιμή του εύρους ως παραμέτρους, χωρίσαμε την τιμή σε τμήματα όπως φαίνεται στο παρακάτω γράφημα.

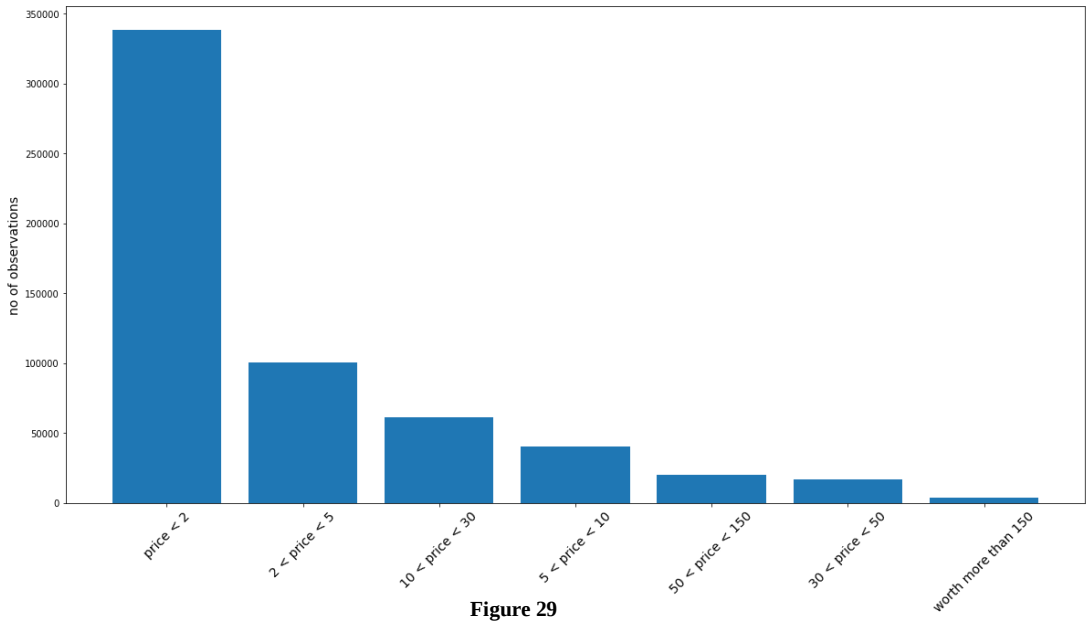


Figure 29

3.6 K-fold

Η διασταυρούμενη επικύρωση είναι μια διαδικασία επαναδειγματοληψίας που χρησιμοποιείται για την αξιολόγηση μοντέλων μηχανικής μάθησης σε ένα περιορισμένο δείγμα δεδομένων. Η διαδικασία έχει μια μόνο παράμετρο που ονομάζεται k που αναφέρεται στον αριθμό των ομάδων στις οποίες πρόκειται να χωριστεί ένα δεδομένο δείγμα δεδομένων. Ως εκ τούτου, η διαδικασία συχνά ονομάζεται k-fold cross-validation.
Στη δική μας περίπτωση όσο μικρότερη η τιμή του k τόσα περισσότερους διαφορετικούς δεν θα έβλεπε το μοντέλο μας και τόσο δυσκολότερο θα ήταν να το εκπαιδεύσουμε. Τελικά χρησιμοποιήσαμε k=10 χωρίζοντας το σύνολο δεδομένων μας σε 90% και 10% κομμάτια εκπαίδευσης και δοκιμών αντίστοιχα από το σύνολο δεδομένων.

3.7 Grid Search

Πριν ξεκινήσουμε το hyperparameter tuning με τη χρήση του GridSearch, έπρεπε να επιλέξουμε τα εύρη μας, τις διάφορες συναρτήσεις ενεργοποίησης, τους βελτιστοποιητές κλπ. Επίσης, έπρεπε να επιλέξουμε πόσα κρυφά στρώματα και επίπεδα αποχώρησης επρόκειτο να χρησιμοποιήσουμε.

Πειραματίσαμε με τα παρακάτω νευρωνικά δίκτυα:
- 1 κρυφό στρώμα
- 2 κρυμμένα στρώματα
- 2 κρυμμένα στρώματα 1 απόρριψη
- 2 κρυμμένα στρώματα 2 dropouts
- 3 κρυμμένα στρώματα 2 dropouts
- 4 κρυμμένα στρώματα 2 dropouts
- 5 κρυμμένα στρώματα 2 dropouts

Από τα παραπάνω, χρησιμοποιώντας 2 κρυμμένα στρώματα και 2 dropouts μας έδωσαν την καλύτερη απόδοση συνολικά. Χρησιμοποιώντας περισσότερα από 2 κρυφά στρώματα αυξήθηκε μόνο ο χρόνος που χρειάστηκε για να εκπαιδεύσουμε το μοντέλο μας και να μειώσουμε τις εποχές που απαιτούνται μέχρι το μοντέλο μας να αρχίσει να υπερισχύει. Η σύγκριση των δύο με μοντέλο regression μας έδωσε τα ακόλουθα αποτελέσματα:

30 epochs, 2 hidden layers, 2 dropout layers 8 minutes 13 seconds training time						
	<20%	<30%	<40%	<50%	>50%	<50% error
(0,2]	7926	3829	3585	2498	11006	61.843018
(2,5]	1246	776	871	1134	4187	49.026053
(5,10]	122	102	119	158	836	37.471952
(10,30]	1839	809	624	451	3086	54.677633
(30,50]	425	262	299	301	977	56.846290
(50,150]	249	190	244	290	891	52.199571
(150,nan]	52	46	73	104	682	28.735632

30 epochs, 5 hidden layers, 2 dropout layers 15 minutes 50 seconds training time						
	<20%	<30%	<40%	<50%	>50%	<50% error
(0,2]	8916	4354	3532	2586	9493	67.130640
(2,5]	866	579	760	1057	4946	39.741715
(5,10]	153	61	85	112	914	31.018868
(10,30]	1213	760	806	631	3330	50.593472
(30,50]	229	198	252	305	1250	44.046553
(50,150]	80	78	120	221	1399	26.290832
(150,nan]	27	21	38	72	845	15.752742

Table 12

3.8 Μετρήσεις αξιολόγησης

Αναλύοντας το πρόβλημα με αλγόριθμους regression, οι μετρήσεις που μπορούσαμε να χρησιμοποιήσουμε ήταν μέσες τιμές τετραγωνικών σφαλμάτων(mse), μέσου απόλυτου σφάλματος(mae) ή μέσου απόλυτου ποσοστού σφάλματος(mape). Παρόλο που και τα 3 θα μπορούσαν να χρησιμοποιηθούν για τη μέτρηση της ακρίβειας για συνεχείς μεταβλητές, κανένα από τα παραπάνω δεν μας βοήθησε να προσδιορίσουμε γιατί το μοντέλο μας ήταν λάθος στις προβλέψεις μας, ποιο εύρος τιμής είχε την καλύτερη ακρίβεια και πώς άλλαξε αυτή η ακρίβεια με διαφορετικούς υπερπαραμετρικούς παράγοντες.

Για το λόγο αυτό κωδικοποιήσαμε τη δική μας μέτρηση ακρίβειας για τα μοντέλα regression, υπολογίζοντας το ποσοστό σφάλματος κάθε πρόβλεψης. Εάν η πρόβλεψή μας ήταν μικρότερη από το 40% της πραγματικής τιμής, τότε υπολογίσαμε την πρόβλεψη ως σωστή για το εύρος τιμής της πραγματικής τιμής. Με αυτόν τον τρόπο υπολογίσαμε την ακρίβεια προβλέψεων κάθε εύρους τιμών.

Αναλύοντας το πρόβλημα με classification, υπολογίσαμε όχι μόνο τη συνολική ακρίβεια κάθε μοντέλου χρησιμοποιώντας την ενσωματωμένη βιβλιοθήκη μέτρησης της sklearn, αλλά υπολογίσαμε επίσης τη συνολική ακρίβεια κάθε εύρους τιμών.

3.9 Αποτελέσματα

Υπολογίζοντας την ακρίβεια της πρόβλεψης για τα μοντέλα regression και classification μετά από Grid Search έχουμε το ακόλουθο διάγραμμα.

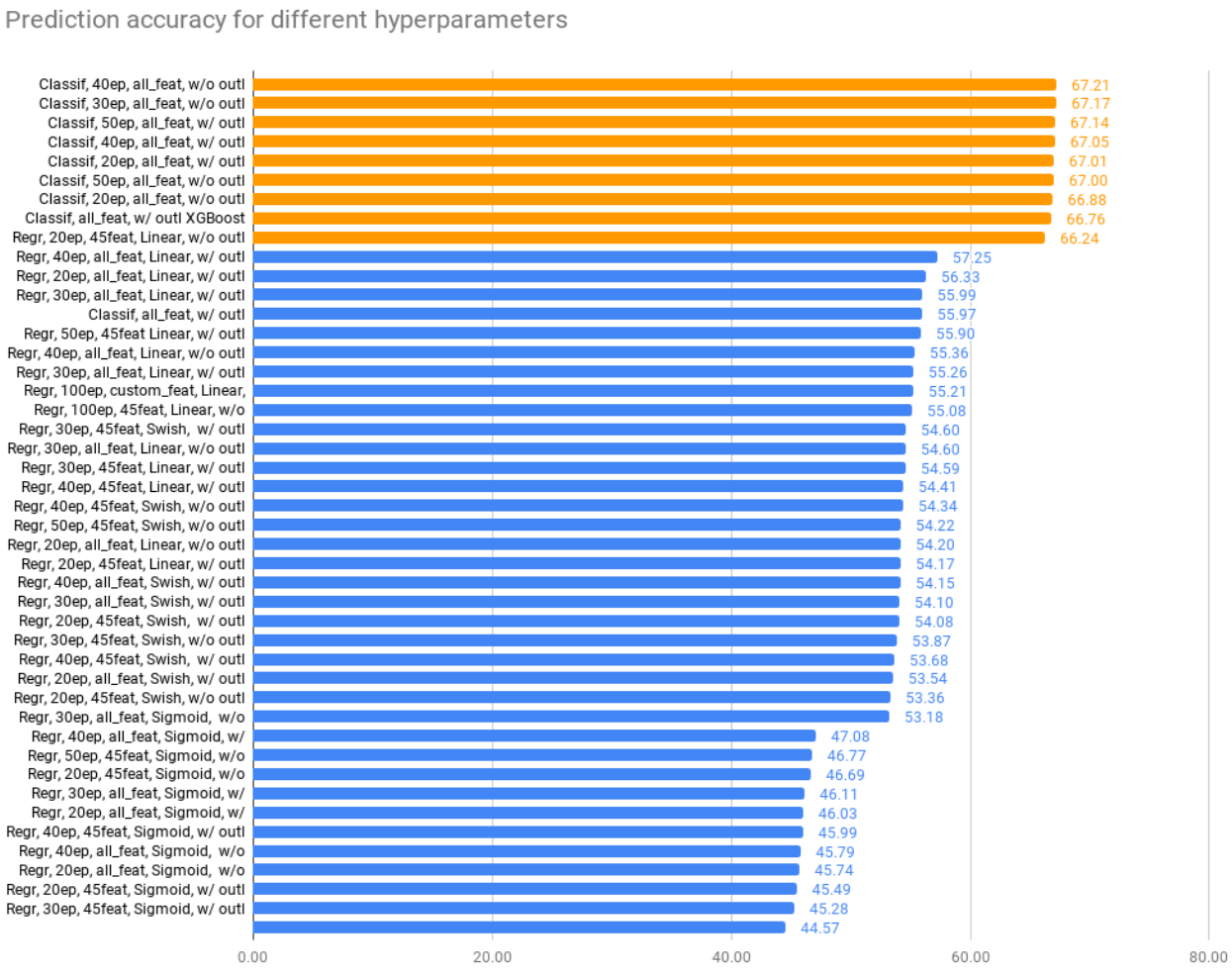


Figure 35

Μπορούμε να δούμε ότι κάθε μοντέλο classification δίνει υψηλότερη ακρίβεια πρόβλεψης από οποιοδήποτε άλλο μοντέλο. Το XGBoost έχει ακρίβεια 66,76%, το οποίο είναι αρκετά σημαντικό για ένα out of the box χωρίς αλγόριθμο συντονισμού μοντέλο. Τα μοντέλα

"μεσαίας" ακρίβειας από 57% έως 53% αποτελούνται από τα Decision Trees για classification και όλα τα άλλα μοντέλα regression με συναρτήσεις ενεργοποίησης Linear και Swish. Μόνο ένα μοντέλο με συνάρτηση ενεργοποίησης Sigmoid έχει ακρίβεια 53% και όλα τα άλλα είναι κάτω από 47%. Στην πραγματικότητα, χωρίς αυτή την εξαίρεση, κάθε μοντέλο με συνάρτηση ενεργοποίησης Sigmoid αποδίδει άσχημα.

Από το παραπάνω διάγραμμα παρατηρούμε ότι πρέπει να χρησιμοποιήσουμε μια προσέγγιση classification στο πρόβλημά μας και να επιλέξουμε το μοντέλο με την καλύτερη ακρίβεια. Αν και αυτό θα ήταν σωστό για τη συνολική ακρίβεια μας, δεν έχουμε πληροφορίες για την ακρίβεια ανά εύρος τιμής. Λαμβάνοντας υπόψη ότι το σύνολο δεδομένων μας δεν έχει καλή κατανομή των τιμών, καθώς το 60% των τιμών κυμαίνεται από 0 έως 2, θα μπορούσε αποδίδει πολύ καλά για αυτά τα εύρη τιμών και να έχει χαμηλότερη απόδοση για τα άλλα. Για το λόγο αυτό, πρέπει να εξετάσουμε προσεκτικά τις ακρίβειες ανά εύρος τιμής, σπάζοντας το πρόβλημα σε 3 διαφορετικές περιπτώσεις: τα μοντέλα πρόβλεψης υψηλής, μεσαίας και χαμηλής ακρίβειας

Μοντέλα πρόβλεψης υψηλής ακρίβειας
Όπως υποπτευθήκαμε, το μοντέλο μας αποδίδει εξαιρετικά καλά για το εύρος τιμών (0, 2) με ακρίβεια πρόβλεψης άνω του 95% χρησιμοποιώντας το XGBoost. Παρατηρώντας τα άλλα εύρη τιμών επιτυγχάνουμε αξιοπρεπή ακρίβεια προβλέψεων 65% 50,150] καθώς και για το φάσμα των τιμών των 10,30 Για τα αντικείμενα που κυμαίνονται μεταξύ 150 και πάνω από το μοναδικό μοντέλο που προσεγγίζει το 35% είναι ο classifier με 50 εποχές, χρησιμοποιώντας όλα τα χαρακτηριστικά και χωρίς outliers.
Για τα εύρη των (2, 5] και (30, 50) τα μοντέλα μας είναι κακά με ακρίβεια 20-22%. χρησιμοποιήσαμε αυτό το μοντέλο για να ταξινομήσουμε την τιμή ενός στοιχείου, θα μπορούσαμε να είμαστε ως επί το πλείστον σωστοί για 3 από τα εύρη τιμών, αλλά να είμαστε απολύτως λάθος για τους άλλα 4. Αυτό επιβεβαιώνει τις υποψίες μας ότι η επιλογή βασιζόμενη αποκλειστικά στη συνολική ακρίβεια πρόβλεψης ενός μοντέλου, δεν μας δίνει τα καλύτερα αποτελέσματα.

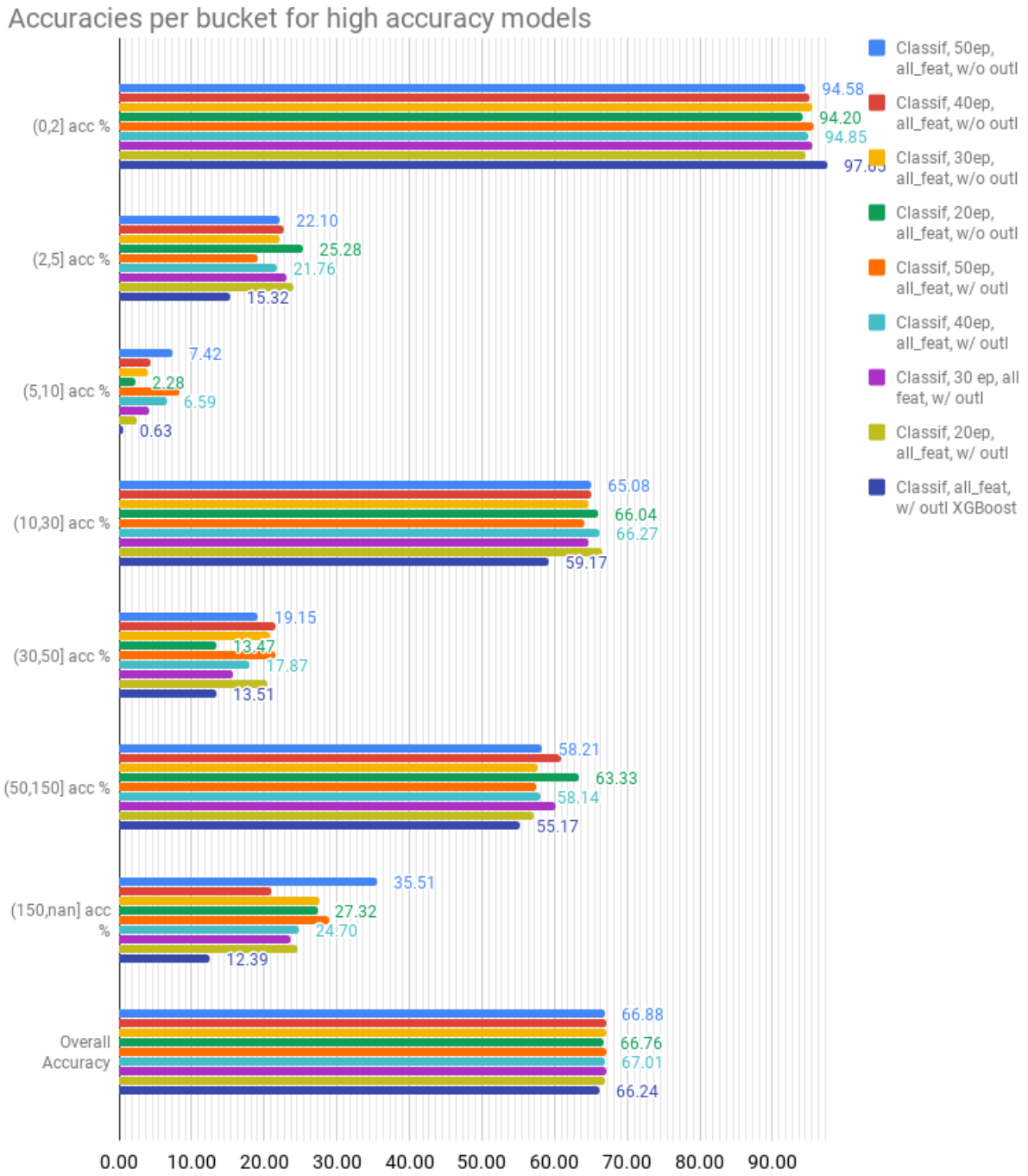


Figure 36

Μοντέλα πρόβλεψης μεσαίας ακρίβειας

Μελετώντας κάθε κλίμακα τιμών ξεχωριστά μπορούμε να καταλήξουμε στα ακόλουθα συμπεράσματα:

- Για το εύρος τιμών (0, 2) παρατηρούμε ότι κάθε μοντέλο regression έχει χαμηλότερη ακρίβεια πρόβλεψης από τον μοναδικό classifier της ομάδας μας, ο οποίος χρησιμοποίησε Decision Trees, αλλά και από τους άλλους classifier στο προηγούμενο διάγραμμα. είναι αυτή που χρησιμοποιεί μια συνάρτηση ενεργοποίησης Swish, που εκτελείται για 30 χρονικές περιόδους χωρίς ακραίες τιμές και χρησιμοποιεί 45 χαρακτηριστικά μετά την επιλογή των χαρακτηριστικών.

- Για το εύρος τιμών (2, 5) επιτυγχάνεται ακρίβεια πρόβλεψης σχεδόν 53% όταν εκτελείται regression με 45 χαρακτηριστικά, για 50 χρονικές περιόδους, χρησιμοποιώντας μια λειτουργία Γραμμικής ενεργοποίησης και χωρίς outliers. Χρησιμοποιώντας μια συνάρτηση ενεργοποίησης Swish, έχουμε μια ακρίβεια 41% περίπου ή χειρότερη, με 20 εποχές που βυθίζονται σε ακρίβεια 30% .Αυτό είναι πολύ καλύτερο από τη χρήση ενός μοντέλου classification που είχε ακρίβεια 25%, με άλλα λόγια, χρησιμοποιώντας regression αντί για classification, διπλασιάζουμε την ακρίβειά μας για αυτό το εύρος τιμών
- Για την τιμή τιμών (5,10) η ακρίβειά μας δεν είναι τόσο κακή όσο πριν, αλλά δεν μπορούμε να προβλέψουμε με ακρίβεια άνω του 40% εάν ένα στοιχείο ανήκει σε αυτό το εύρος τιμών. πάνω από 36%, παρατηρούμε ότι η αφαίρεση των outliers μειώνει την ακρίβεια των προβλέψεών μας, πιθανώς επειδή αυτές οι τιμές ήταν παρατηρήσεις για τα υψηλότερα εύρη τιμών αλλά είχαν συνδυασμούς στοιχείων που ανήκουν σε χαμηλότερο εύρος τιμών.
- Για την κλίμακα τιμών (10,30) με regression, έχουμε μια χαμηλότερη ακρίβεια σχεδόν κατά 10%. Όποια και αν είναι οι υπερπαραμετρικές τιμές, οι ακρίβειες ήταν πολύ κοντά, με εξαίρεση τις συναρτήσεις ενεργοποίησης με Linear οι οποίες μας έδωσαν τα καλύτερα αποτελέσματα (από 54,7 έως 56,6%)
- Για το εύρος τιμών (30,50) παρατηρούμε αύξηση της ακρίβειας σχεδόν 3 φορές από τον υψηλότερο αλγόριθμο ταξινόμησης που είναι η Decision Trees με ακρίβεια 26,7%. Με την regression επιτυγχάνεται ακρίβεια 68,7% με τη χρήση γραμμικής ενεργοποίησης Η αύξηση των εποχών μετά την τιμή των 50 δεν αυξάνει την ακρίβειά μας, αφού βλέπουμε το μοντέλο μας με 100 εποχές και μια συνάρτηση γραμμικής ενεργοποίησης με χαμηλότερη ακρίβεια .
- Για το εύρος τιμών (50,150), τα μοντέλα regression είναι τόσο καλά όσο τα ταξινομητικά μας και ακόμα καλύτερα, αφού η συνάρτηση ενεργοποίησης Swish με 50epochs και χωρίς ακρίβεια δίνει ακρίβεια 68,88% και με γραμμική ενεργοποίηση έχουμε 67,2% ακρίβεια.
- Για το εύρος τιμών των 150 και άνω για 50 εποχές, χρησιμοποιώντας μια λειτουργία ενεργοποίησης Swish, έχουμε 51,87% ακρίβεια ενώ με τη χρήση classification έχουμε ακρίβεια 35,51%.

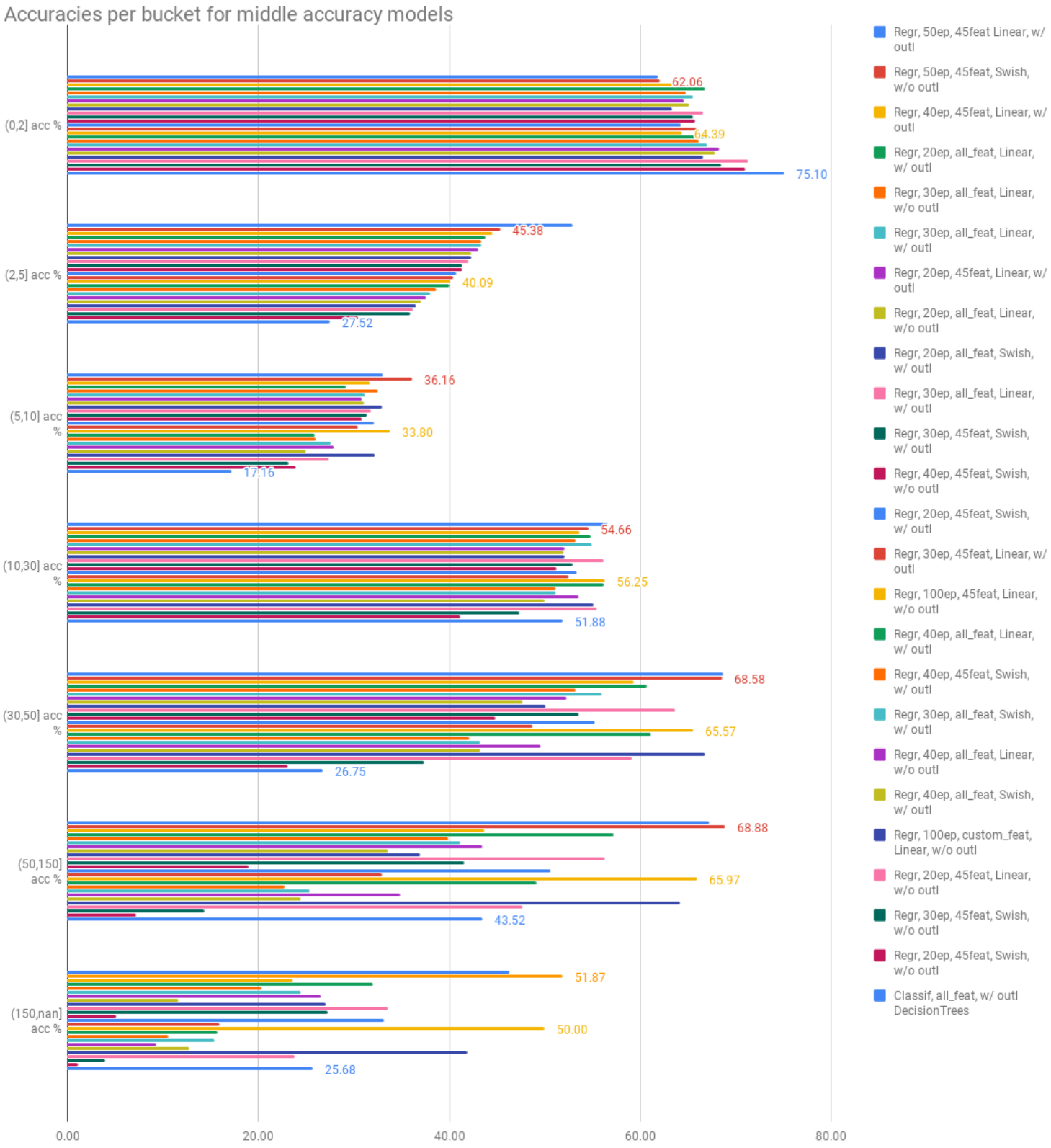

Accuracies per bucket for middle accuracy models
(0,2] acc %
(2,5] acc %
(5,10] acc %
(10,30] acc %
(30,50] acc %
(50,150] acc %
(150,nan] acc %
62.06
64.39
75.10
45.38
40.09
27.52
36.16
33.80
17.16
54.66
56.25
51.88
68.58
65.57
26.75
68.88
65.97
43.52
51.87
50.00
25.68
0.00
20.00
40.00
60.00
80.00
Regr, 50ep, 45feat Linear, w/ outl
Regr, 50ep, 45feat, Swish, w/o outl
Regr, 40ep, 45feat, Linear, w/ outl
Regr, 20ep, all_feat, Linear, w/ outl
Regr, 30ep, all_feat, Linear, w/o outl
Regr, 30ep, all_feat, Linear, w/ outl
Regr, 20ep, 45feat, Linear, w/ outl
Regr, 20ep, all_feat, Linear, w/o outl
Regr, 20ep, all_feat, Swish, w/ outl
Regr, 30ep, all_feat, Linear, w/ outl
Regr, 30ep, 45feat, Swish, w/ outl
Regr, 40ep, 45feat, Swish, w/o outl
Regr, 20ep, 45feat, Swish, w/ outl
Regr, 30ep, 45feat, Linear, w/ outl
Regr, 100ep, 45feat, Linear, w/o outl
Regr, 40ep, all_feat, Linear, w/ outl
Regr, 40ep, 45feat, Swish, w/ outl
Regr, 30ep, all_feat, Swish, w/ outl
Regr, 40ep, all_feat, Linear, w/o outl
Regr, 40ep, all_feat, Swish, w/ outl
Regr, 100ep, custom_feat, Linear, w/o outl
Regr, 20ep, 45feat, Linear, w/o outl
Regr, 30ep, 45feat, Swish, w/o outl
Regr, 20ep, 45feat, Swish, w/o outl
Classif, all_feat, w/ outl DecisionTrees

Figure 37

Μοντέλα πρόγνωσης χαμηλής ακρίβειας

Όπως μπορούμε να δούμε, ο λόγος για τον οποίο τα τελευταία μοντέλα μας είχαν τόσο χαμηλή συνολική ακρίβεια πρόβλεψης, ήταν επειδή δεν μπορούσαν να προβλέψουν κανένα στοιχείο από 5 σε πάνω από 150 τιμές. Αυτό συμβαίνει λόγω της φύσης της συνάρτησης Sigmoid, όπως αναφέρθηκε προηγουμένως, δεν μπορεί να παράγει αρνητικούς αριθμούς. Παρόλο που μια τιμή ενός αντικειμένου δεν μπορεί να έχει αρνητικές τιμές, χρησιμοποιήσαμε βασική λογαριθμική συνάρτηση σε όλες τις τιμές για καλύτερη κατανομή και πήραμε αρνητικά αποτελέσματα και γι 'αυτό με τη χρήση της συνάρτησης ενεργοποίησης sigmoid δεν μπορούμε να κάνουμε προβλέψεις για τιμές υψηλότερες από 5

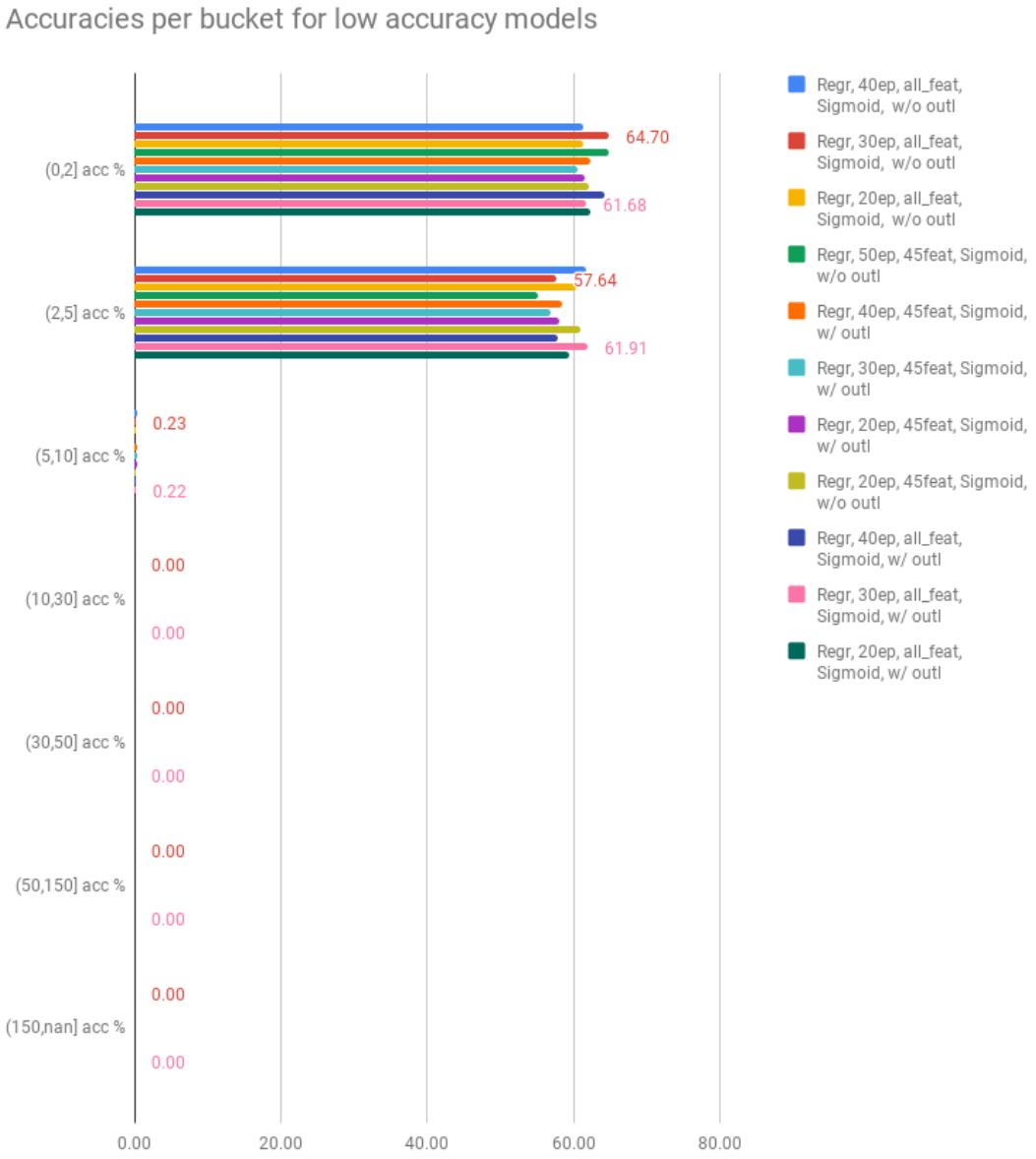


Figure 38

Αφού αναλύσουμε τα μοντέλα χαμηλής, μέσης και υψηλής ακρίβειας πρόβλεψης, βλέπουμε ότι κάθε κατηγορία κάνει καλή δουλειά προβλέποντας ένα εύρος τιμών, αλλά όχι όλα τα εύρη. Αυτό γίνεται ακόμα πιο εμφανές όταν παρατηρούμε τα καλύτερα μοντέλα ανά περιοχή τιμών σε αυτό το τελευταίο διάγραμμα.

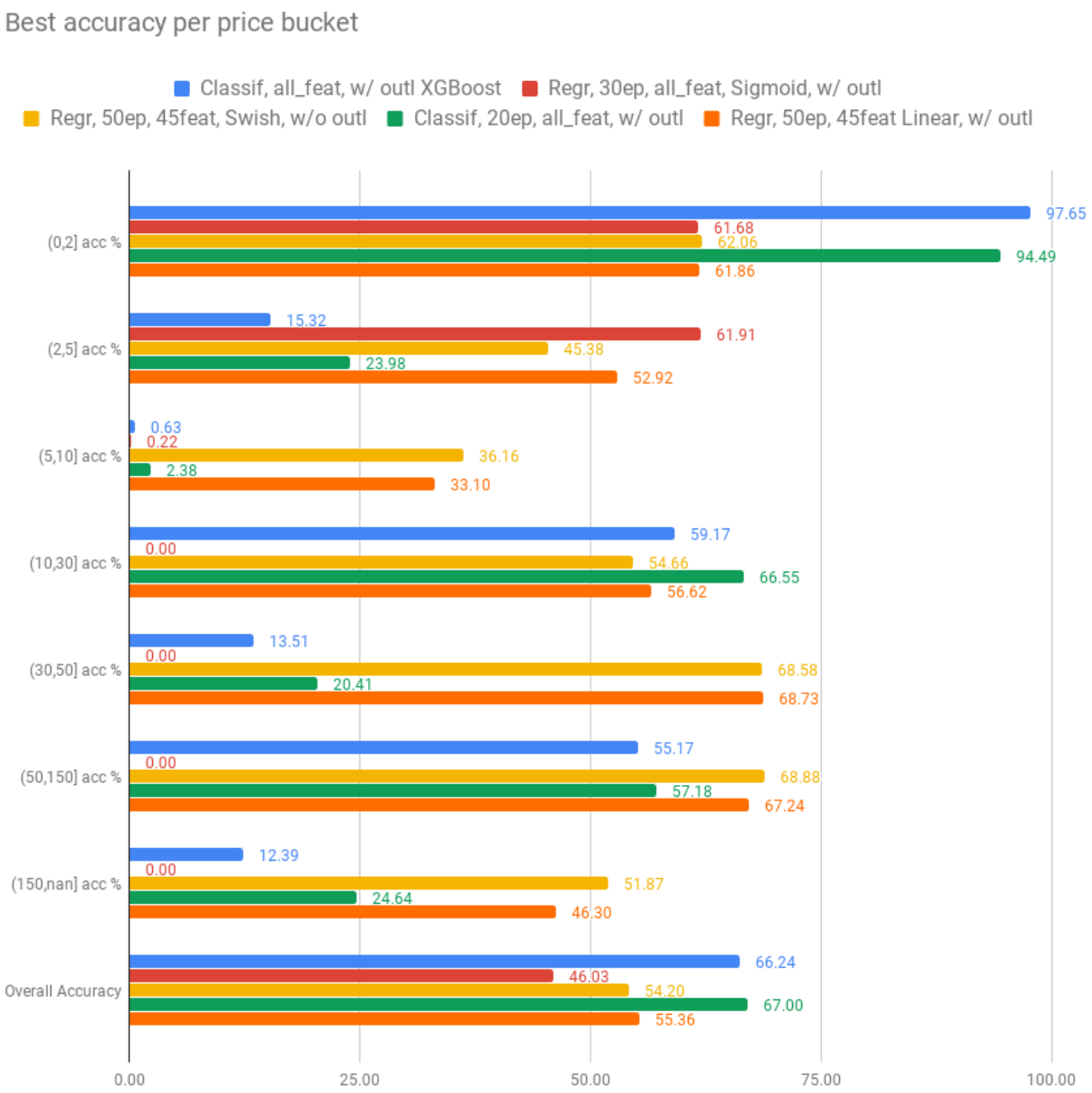


Figure 39

Είναι σαφές ότι τα μοντέλα classification είναι πραγματικά καλά για τις περιοχές (0,2], (10,30) και (50,150), μοντέλα regression με συνάρτηση ενεργοποίησης Sigmoid για το εύρος των (2,5), μοντέλα regression με συνάρτηση γραμμικής ή swish ενεργοποίησης για το εύρος(30,50) και μοντέλα regression χρησιμοποιώντας μια συνάρτηση ενεργοποίησης Swish για την κλίμακα των 150 και άνω. Για το εύρος τιμών των [5,10] δεν υπάρχουν μοντέλα που να μας δίνουν μια καλή προβλέψεις με καλή ακρίβεια.

3.10 Ensemble

Παρατηρώντας τα αποτελέσματα της ανάλυσης και της απόδοσής μας στο σύνολο δεδομένων Rares βλέπουμε ότι οι χρησιμοποιούμενοι αλγόριθμοι δεν προσφέρουν το καλύτερο αποτέλεσμα για όλους τους τύπους τιμών. Αυτή είναι μια ένδειξη ότι εφαρμόζοντας μεθόδους ensemble μπορεί να έχουμε καλύτερη απόδοση. Χρησιμοποιήσαμε ψηφοφορία με σταθμισμένη πρόβλεψη.Τα υπομοντέλα που χρησιμοποιήθηκαν είναι:

Sub - learner	Better results at segment	Prediction Accuracy
ANN with activation function 'swish', 50 epochs, 45 features, without outliers	(30,50 (50,150 (150,nan] (5,10]	55.38%
ANN with activation function 'swish', 30 epochs, 45 features, without outliers	(0,2]	53.76%
ANN with activation function 'sigmoid', 30 epochs, 45 features, without outliers	(2,5]	47.44%
ANN with activation function 'linear', 100 epochs, 45 features, without outliers	(10,30]	55.41%

Table 14

Εκπαιδεύσαμε τα μοντέλα και συγκροτήσαμε τις προβλέψεις τους με διαφορετικό βάρος για το κάθε μοντέλο.

Η μέθοδος ensemble μας έδωσε βαθμολογία ακρίβειας 57,12% βελτιώνοντας την προηγούμενη βαθμολογία απόδοσης 55,41%, η οποία ήταν η καλύτερη βαθμολογία από τα 4 μοντέλα που χρησιμοποιήσαμε.Αυτή η πρώτη προσέγγιση είναι η απλούστερη δυνατή αφού έχουμε καλύτερα αποτελέσματα όταν εφαρμόζουμε αλγόριθμους classification στο σύνολο δεδομένων Rares.Η παραγωγή ενός ensemble μοντέλου με τα καλύτερα αποτελέσματα ταξινόμησης ή ακόμα και η ανάπτυξη μιας στοιβαγμένης συσσωμάτωσης(stacked aggregation) 2 επιπέδων θα επέφερε ενδεχομένως καλύτερα αποτελέσματα.

3.11 Συμπεράσματα

Το σύνολο δεδομένων αποτελείται από 580.000 παρατηρήσεις που φιλοξενούν ένα μεταβλητό αριθμό χαρακτηριστικών από 90+ έως 510+ σε συνδυασμούς των 11. Όπως φαίνεται ήδη στα διαγράμματα, ορισμένοι συνδυασμοί μεταξύ αυτών των χαρακτηριστικών είναι μάλλον σπάνιοι και έχουν μικρή παρουσία στο σύνολο δεδομένων. Ως αποτέλεσμα, οι αλγόριθμοι δεν έχουν κανένα τρόπο να μάθουν και η απόδοση πέφτει.

Παρόλο που κανένας από τους αλγορίθμους μας δεν ήταν καλός στην πρόβλεψη όλων των τιμών των αντικειμένων, τη σπάσιμο της τιμής σε εύρος τιμών και παρατηρώντας αυτά τα εύρη τιμών, μας έδωσε την εικόνα για τους καλύτερους αλγόριθμους σε κάθε ένα από αυτά. Αφού αντιμετωπίσαμε το πρόβλημα με classification, κατορθώσαμε να δημιουργήσουμε ένα

μοντέλο που θα προέβλεπε με ακρίβεια εάν ένα στοιχείο ήταν πολύτιμο ή όχι. Με ένα βαθμό απόδοσης 97,65% στο τμήμα του (0, 2) το μοντέλο είναι αρκετά σίγουρο ότι αντικείμενα σε αυτό το εύρος τιμής δεν ειναι πολύτιμα.

Για το τμήμα regression του προβλήματος, χρησιμοποιώντας 4 διαφορετικά μοντέλα regression και χρησιμοποιώντας ensemble, μπορέσαμε επίσης να βελτιώσουμε τη συνολική ακρίβειά μας. Η προσαρμογή των παραμέτρων μας και η εύρεση καλύτερων μέσων βαρών για τα μοντέλα μας ή η δοκιμή διαφορετικών μοντέλων θα μπορούσαν να μας δώσουν ακόμα καλύτερα αποτελέσματα. Τέλος, η χρήση ensemble σε μοντέλα classification θα μπορούσε επίσης να βοηθήσει στη βελτίωση της συνολικής ακρίβειάς μας.

Αφού παρατηρήσαμε τα αποτελέσματά μας, καταλήξαμε στο συμπέρασμα ότι μια μεγάλη βελτίωση στις επιδόσεις των αλγορίθμων μας θα ήταν η προσθήκη περισσότερων δεδομένων. Αυτό θα είχε ως αποτέλεσμα όχι μόνο διαφορετικούς συνδυασμούς αλλά και περισσότερες παρατηρήσεις ανά εύρος τιμών. Υπάρχουν διάφοροι τρόποι με τους οποίους μπορούμε να προσθέσουμε περισσότερα δεδομένα:

Ανάγνωση της του API feed από την αρχή του παιχνιδιού
Αυτό θα περιλάμβανε πάνω από 2,5 χρόνια συναλλαγών, αλλά θα δημιουργούσε επίσης ορισμένα προβλήματα, καθώς το API δεν έχει τις ιδιότητες του χρόνου που εισήγαμε και αυτό θα σήμαινε ότι είναι ακόμη δυσκολότερο να απομακρυνθούν οι παραληρηματικές παρατηρήσεις. Θα εισήγαγε επίσης προβλήματα σχετικά με την αποθήκευση του συνόλου δεδομένων και την υπολογιστική ισχύ που απαιτείται για την κατασκευή του συνόλου δεδομένων και θα συνεπαγόταν μια εντελώς νέα κατανεμημένη αρχιτεκτονική.

Ενισχύοντας το σύνολο δεδομένων με δικά μας δείγματα
Δειγματοληψία γύρω από την αξία των χαρακτηριστικών και δημιουργώντας μια δέσμη παρόμοιων δειγμάτων καθώς και χρησιμοποιώντας τις γνώσεις του τομέα μας για να δημιουργήσουμε περισσότερες παρατηρήσεις. Αυτό σίγουρα θα εισήγαγε bias στα μοντέλα μας, αλλά θα μπορούσε να βελτιώσει τις επιδόσεις μας.

Βελτίωση του τρέχοντος συνόλου δεδομένων με profiling των χρηστών
Αυτό θα ήταν δύσκολο να εφαρμοστεί, αλλά θα μπορούσαμε να αναπτύξουμε το καλύτερο μοντέλο μας, να δημιουργήσουμε λογισμικό γύρω του για να το χρησιμοποιήσουμε απευθείας από τους παίκτες του παιχνιδιού και στη συνέχεια να λάβουμε άμεσα σχόλια από τους παίκτες για την εγκυρότητα των προβλέψεών μας. Με αυτόν τον τρόπο θα μπορούσαμε να προσδιορίσουμε ποιοι συνδυασμοί χαρακτηριστικών δημιουργούν προβλήματα και εισάγουν αρχικά βάρη χαρακτηριστικών για το πρόβλημά μας, καθώς και να πάρουμε περισσότερα δεδομένα απευθείας από τους παίκτες.

Κεφάλαιο 4: Μελλοντική δουλειά

Το έργο μπορεί να αποδείχθηκε πιο δύσκολο από ό, τι αρχικά αναμενόταν και η απόδοση των μοντέλων μας σίγουρα δεν είναι η καλύτερη για τα Rare αντικείμενα, αλλά υπάρχουν ακόμα πολλές ιδέες που μπορούμε να δοκιμάσουμε στο μέλλον που μπορεί να αυξήσουν την αξιοπιστία του συνόλου δεδομένων ή την απόδοση των μοντέλων μας.

Καταπολέμηση πλαστών δεδομένων

- Κατά τη δημιουργία του συνόλου δεδομένων μας αποφασίσαμε να δημιουργήσουμε μια σειρά από χαρακτηριστικά που θα χρησιμοποιηθούν αργότερα, ένα από τα οποία είναι οι ημέρες που χρειάστηκε για να πωληθεί ένα στοιχείο. Αργότερα, αποφασίσαμε ότι θα έπρεπε να έχουμε εντοπίσει τις ώρες που χρειάστηκε για να πωληθεί ένα αντικείμενο ή ακόμα και λεπτά, αφού, σύμφωνα με τις γνώσεις του τομέα, ένα αντικείμενο που παίρνει λιγότερο από 30 λεπτά για να πουληθεί, έχει μεγάλη πιθανότητα να αποτελεί ψεύτικο δεδομένο.

- Στο σύνολο δεδομένων υπάρχουν ορισμένα χαρακτηριστικά που δεν χρησιμοποιούνται.
  Ένα από αυτά είναι το όνομα χρήστη του παίκτη που πραγματοποίησε τη συναλλαγή. Θα μπορούσαμε να κατηγοριοποιήσουμε παίκτες σε κατηγορίες που εκφράζουν το επίπεδο εμπιστοσύνης που υπολογίζουμε όχι μόνο από το ίδιο το σύνολο δεδομένων αλλά και από άλλες εξωτερικές πηγές.

Δικαιολογημένες αξίες

- Σε ορισμένες περιπτώσεις οι εξωστρεφείς τιμές είναι απολύτως αποδεκτές.
  Μία τέτοια περίπτωση είναι όταν ένα στοιχείο στα Μοναδικά στοιχεία έχει μια ακμή στην αξία, όταν ένας streamer το χρησιμοποιεί καθώς παίζει ζωντανά το παιχνιδι. Θα μπορούσαμε να χρησιμοποιήσουμε causal inference για να υπολογίσουμε την απόκλιση από το κανονική πλοκή και να προβλέψουμε την τιμή σωστά

- Μια άλλη περίπτωση είναι όταν κάποιος σπάνιος συνδυασμός παίρνει αποτελέσματα όταν το παιχνίδι εφαρμόζει ένα χαρακτηριστικό στο αντικείμενο.

Επεξεργασία φυσικής γλώσσας σε φόρουμ παιχνιδιών και reddit

Για την ανάλυση των Unique αντικειμένων μπορούμε να χρησιμοποιήσουμε τη φυσική επεξεργασία γλώσσας σε φόρουμ παιχνιδιών ή το reddit, να καθορίσουμε τη θετική γλώσσα του παίκτη σε ορισμένα αντικείμενα και να εισαγάγουμε νέα στοιχεία στο σύνολο δεδομένων μας, που θα μας επιτρέψουν να προβλέψουμε καλύτερα τις τάσεις των αντικειμένων.
Το Reddit και τα φόρουμ είναι επίσης μια πολύ πλούσια πηγή γνώσεις διαφόρων παικτών που προσπαθούν να απατήσουν αρχάριους παίκτες προτρέποντάς τους να πουλήσουν τα αντικείμενα τους πολύ φθηνά. Αυτό θα μπορούσε να βοηθήσει στην ταξινόμηση ορισμένων παικτών ως Untrusty και συνεπώς τα στοιχεία που πωλούν προς πώληση θεωρούνται χαμηλά

στην τιμή, ή άλλους παίκτες ως Novice και επομένως τα αντικείμενα τους δεν πρέπει να θεωρούνται υψηλά στην εμπιστοσύνη όσον αφορά την τιμή τους.

Επιπλέον,το να είμαι σε θέση να καθορίσουμε πότε υπάρχει «streamer effect», μια φράση η οποία στον κόσμο των παιχνιδιών σημαίνει ότι εάν ένα διάσημο άτομο χρησιμοποιεί ορισμένα στοιχεία τα οποία αυτά τα στοιχεία θα αυξήσουν την τιμή, θα βοηθήσει επίσης να προβλέψουμε τις τάσεις αυτών των στοιχείων.

Μέτα στοιχεία

Στον κόσμο του Path of Exile, το "meta" αναφέρεται στα καλύτερα στοιχεία σε μια δεδομένη στιγμή στο παιχνίδι. Αυτά τα στοιχεία "meta" είναι συνήθως πιο ακριβά. Η εξόρυξη δεδομένων των πιο χρησιμοποιούμενων αντικειμένων από εξωτερικούς ιστότοπους θα μας βοηθήσει να προσδιορίσουμε ποια είναι αυτά τα στοιχεία και να δημιουργήσουμε ένα νέο χαρακτηριστικό που θα αντιπροσωπεύει τη δημοτικότητα ενός στοιχείου.
Αυτά τα στοιχεία, αν είναι σπάνια, περιέχουν επίσης ορισμένα χαρακτηριστικά που θα μας βοηθούσαν να αρχικοποιήσουμε το βάρος αυτών των χαρακτηριστικών για τα μοντέλα μηχανικής μάθησης.

Δημιουργία μοντέλων για μεμονωμένες κατηγορίες τιμών

Μια άλλη ιδέα που πρέπει να εφαρμοστεί είναι να δημιουργηθούν δυαδικά μοντέλα ταξινόμησης, ένα για κάθε εύρος τιμών. Για να γίνει αυτό, για ένα μοντέλο κάθε τιμής τιμών θα ταξινομήσαμε την τιμή κάθε παρατήρησης ως 0 ή 1 αν δεν ήταν ή ήταν σε αυτό το εύρος τιμών και να εκπαιδεύσει κάθε μοντέλο με όλες τις παρατηρήσεις. Στη συνέχεια, θα χρησιμοποιήσαμε ensemble για να προβλέψουμε το εύρος τιμών ενός στοιχείου.