Leiden University

Advances in Data Mining

Assignment 2: Predict future sales

Group 44

Authors:
Louis Siebenaler (s3211126)
Theofilus Hobba Pramono (s3613291)
Peter Breslin (s3228282)

October 24, 2023

1 Introduction

In this second assignment to the Advances in Data Mining Course, we are tasked with a time-series forecasting problem where we must accurately predict the future sales of a software company. This follows the Kaggle competition: Predict Future Sales, which provides datasets consisting of daily sales records from the Russian software firm 1C Company. We entered this competition under the team name `ADM_Group44` for the purposes of this assignment. The goal of the competition is to predict the total sales for every product and store in the next month, providing the opportunity to apply our data mining skills to real-world data which is usually not optimized for machine learning algorithms in its base state. Understanding and augmenting the data will be key components to addressing this problem, which is why extensive exploratory data analysis, data processing, and feature engineering is undertook. We also explore different approaches to the construction of our models, as choosing suitable time-series forecasting algorithms is a critical factor to consider.

2 Exploratory Data Analysis

Before applying machine learning algorithms to the data for predicting future sales, we must gain some intuition about the data itself. This is achieved by conducting exploratory data analysis which allows us to determine underlying trends, correlations and possible errors in the data.

2.1 General properties, outliers, errors

The training set is a collection of 2935849 sales for 22170 items sold in 60 shops across Russia over the course of 34 months, from January 2nd 2013 to October 3rd 2015. The sales are split into 6 columns labelled '`date`' (the sale date in format dd/mm/yyyy), '`date_block_num`' (integer label of the month of sale), '`shop_id`', '`item_id`', '`item_price`', and '`item_cnt_day`' (number of products sold per day). Each row represents the sale count of an item in a specific shop on a specific date.

The range of item price and item count per day (i.e `item_cnt_day`) is given in Figure A1 and Figure A2. We find that both properties exhibit a very wide range of values, with 6 orders of magnitude for item prices and 4 orders of magnitude for item count per day. However, we discern that items with extreme values are obvious outliers, with only 1 item exceeding a prize of 10^5 Russian Rubles and 2 items surpassing 10^3 sales in a day. These items will be ignored during training. Furthermore, we discovered that there exists both a negative item price and multiple negative item counts per day. The fact that there is only one negative item price suggests that it is most likely an error, so we remove this item from the data. In contrast, there are 7356 negative item count per day entries in the data, ranging from -1 to -22. It is unclear how to interpret these entries, as they could correspond to errors, or represent returned items. More details on the analysis of negative values can be found in the Notebook on exploratory data analysis.

We are provided with 3 further data sets, including information on shops, item categories and the items themselves. Russian explanations are provided in the data and hence translations were performed. The shop set consists of 60 rows of 2 columns each representing the shop ID and name. The category set contains 84 rows and 2 columns, with each row containing the name

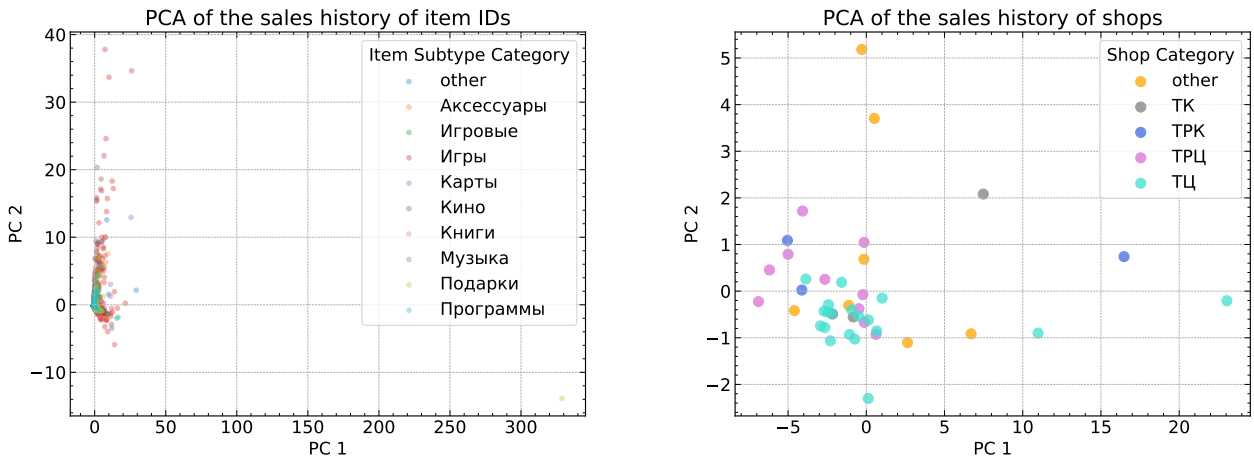

Figure 1: *Left-panel*: The principal component analysis (PCA) of the sales history of item IDs. Each data point is color coded according to the item subtype category. *Right-panel*: shows the PCA of shop sales history. Each data point corresponds to a unique shop ID and is color coded according to the shop category.

and integer label of a single item category. The item set consists of 22170 rows of 3 columns, each representing the name, item ID, and category ID. We performed cleaning and processing to overcome inconsistencies in the string-type variables and extracted additional information from the name columns, this is further explained in Section 3.1. This processing leads to the discovery of potential shop duplicates based on the similarity of their names. These duplicated shop ID pairs are 0 and 57, 1 and 58, and 10 and 11. Furthermore, through the processing we found potential item duplicates in the data. Section 3.1 outlines how these are treated.

To further understand the data and discern properties of use for our forecasting models, we performed Principal Component Analysis (PCA) on certain time-series properties. The *left-panel* in Figure 1 shows the PCA of the sales history of item IDs. Precisely, we have determined the evolution of the total monthly sales of an item ID (i.e summed over all shops), and reduced its dimensionality from 35 (there are 35 months in the training set) to 2. The color coding is according to the item subtype, an additional feature extracted from the category dataset that represents the second part of the category name. We observe that most item IDs are clustered, although the clustering appears stronger within each subtype category. This implies that items in the same subtype category have similar sales histories, and thus item category properties should be useful for forecasting. Furthermore, we observe some data points with large principal components indicating item IDs with unusually high sales histories. The *right-panel* in Figure 1 shows the PCA of the sales history of shops, with color coding according to the shop category. We observe clear clustering, most notably for the turquoise data points (i.e. shop category ТЦ) which suggests that shops in the same category have similar sales behavior. Hence, the shop category parameter should be useful for the forecasting task.

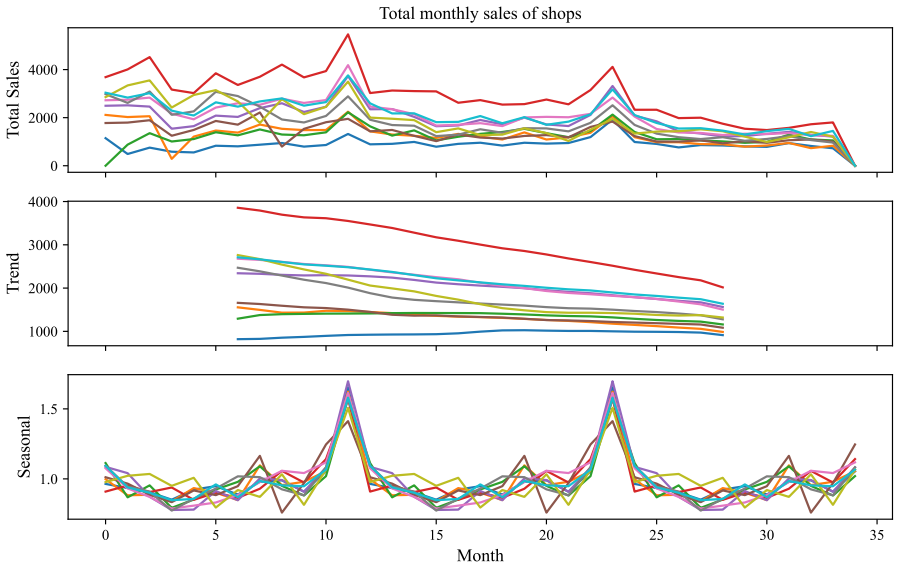

Figure 2: Main components of several shop sales histories in the dataset. Each color represents a unique shop ID. *Upper-panel* shows the sales history as a function of time, *middle-panel* displays the trend in the time series, and the *lower-panel* gives the seasonal component. We find that most shops exhibit a seasonal periodicity of 12 months.

2.2 Autocorrelations

From the available data set it is possible to create a variety of time-series related to the monthly sale count of item-shop pairs. Examples of this are the evolution of the total monthly sales of a shop or item respectively, the average sales per month, or how item prices evolve over time. In this part, we investigate trends and seasonality patterns in various time-series and measure their autocorrelation, in order to assess whether they are useful for our task at hand. Figure 2 shows the evolution of the total monthly sales in a number of shops as a function of time (*upper-panel*), the existing trend in these time series (*middle-panel*) and their seasonal component (*lower-panel*). From this, it is clear that many shops exhibit a similar temporal behaviour in their monthly sales, since most shops in Figure 2 exhibits a decreasing trend. Furthermore, we discern that shops have a clear seasonal component with a period of 12 months. Precisely, a large increase in sales around month 12 and 24 exists which is around Christmas season. We measured the autocorrelation of this time-series for each existing shop ID in the dataset to determine if past values of the series carry some information about the future behaviour. We find that for a lag size up to 4 months, the majority of shops have an autocorrelation value in their total monthly sales exceeding the 68% confidence level. This is shown in Figure A3, where the distribution of the lag sizes of the autocorrelation values, for which the 68% confidence interval is exceeded, is given. Hence, we conclude that up to 4 months in the past, there still exists meaningful information to predict the following month. Additionally, we observe that lag sizes of 12 and 24 months exhibit high autocorrelation values. As a result, 12 and 24 months in the past could also be useful to make a prediction of the sales in the following month.

For the total monthly sales of a specific item ID, summed over all shops, we are unable to find

clear trends and seasonal components which are present in a large number of item IDs. The analysis of this time-series is more challenging given the more than 20000 unique items in the dataset. Nevertheless, by measuring the autocorrelation function for the total monthly sales of each item, we are able to find that up to a lag size of 3 months, a significant proportion of time series have autocorrelation values exceeding the 68% confidence interval. Hence, we conclude that it is reasonable to consider the past 3 months of total sales of an item to perform our predictions.

Lastly, we have analysed the autocorrelation of the monthly sales of an item in a shop, that is the time-series which we want to model in this work. For this we find that up to a lag size of 3 months, a significant proportion of time series have autocorrelation values exceeding the 68% confidence interval.

2.3 Test Set

The test set of this competition consists of a total of 214200 item-shop pairs for which we want to predict the monthly sales of the 35th month. Upon further analysis of the test set, we find that it consists of three groups of items. The first group corresponds to item-shop pairs which are present in the training set and as a result we have a sales history of them. This accounts for 52% of the total test set. The second group are items which are present in the training set, however the item-shop pair does not exist in the training set. 40% of the test set belongs to this group. Lastly, the remaining 8% of items have no sales history at all given their absence from the training set. These could be new products or a result of errors in the training set.

Clearly, the forecasting task for each item group differs. For item-shop pairs which have a sale history, it makes sense to use item-specific lag features, or price history to perform predictions. This method should be extendable to the second item group, however for items which have no sales history a different approach should be used. One approach would be to use categorical features or shop properties for the prediction, instead of item-specific time-series features.

3 Method

This section includes the concept of feature engineering as the main tool to feed relevant data into the training of our models. We also outline the data cleaning we perform (e.g. clipping and removal of outliers), which properties we consider, which lag features, and so on. Finally, we briefly describe and outline the models we chose to use and how we evaluated them.

3.1 Data Preparation

Before training models to perform forecasting, we prepare the training data to an optimal format. This involves removing/handling errors in the data, duplicates or obvious outliers. The first step consists of removing the shops with ID 9 and 20, given that they are not present in the test set. Although this reduces the training data and we loose potential item-specific time-series features, it produces better results. Next, we remove obvious item outliers, which exhibit a monthly sale count exceeding 1000 and price above 300000 Russian rubles. Furthermore, as outlined in Section

2.1, there exists monthly item sales with negative values. We interpret these as products returned to a shop and hence set their count value to 0. We find that this yields overall better results than taking the absolute value of negative counts.

When making a prediction, the Kaggle competition requires to clip the final target values to a range of $[0, 20]$ monthly sales. Hence we decide to clip the monthly item sales in the training set to the same range, in order to prevent that aggregated features for training are not distorted due to potential outliers. As explained in Section 2.1, there are potential shop duplicates in the data which we grouped to one ID. Those shops are IDs 0 and 57, 1 and 58, and finally 10 and 11. For each shop, we determine its city through its given name and label encode this property. Furthermore, we label encode the shop categories.

For the item categories, we are given names which consist of two parts; the first representing the type of product (e.g. videogame) and the second giving a more specific description (e.g. PS4 or PS3). Based on this, we create further data related to item categories, where we label encode the first part of the category name (defined as type code) and additionally produce a subtype category based on the second part of the name (defined as subtype code), which again is label encoded. Similarly for item names, there exists a structure which allows us to subdivide it into two parts which we label encode. It should be noted, that the preparation and correction of item names as well as categories has been adopted from other publicly available notebooks. After correcting item names, we notice that there exists possible item duplicates in the training data. For potential duplicates, we check if their item IDs are present in the test set. If both their IDs exist in the test set, we decide to leave the data unchanged. When only one of the IDs is in the test set, we assign the other item to this ID. By doing so, we reduce the cases of items in the test set for which we have no sales history to 7% (i.e by 630 items, see Section 2.3). If none of the IDs are in the test set, we assign both items to the same ID.

After performing these steps, we produce a matrix where each row represents monthly sales of an item-shop pair of the cleaned data. Furthermore, item names, item categories, cities, and shop categories are added to each row.

3.2 Feature Engineering

When building a model to predict a certain property over time, it can be useful to extract additional features from the data to use as input in the training. This may capture important information that might not be apparent from the data without having done so. Feature engineering refers to this process of creating new features from existing data. This technique also includes feature selection (i.e. only using features deemed most relevant to the problem) and feature combination to improve their quality. Hence, feature engineering is used to provide the best and most relevant input data to the model, thereby increasing performance and improving predictions.

For time-series forecasting problems, the concept of lag features are often used. These represent values of properties at earlier time-steps with the assumption that prior information about these features may influence their future, helping to capture otherwise hidden trends and patterns in the data. Lag features are created by shifting the values of relevant properties by a certain number

of time-steps. In our work, we created a number of new features inspired by examples of feature engineering in publicly available Kaggle notebooks. We further expanded on the features by lagging their values and adding the resulting lag features to our input. Table A1 gives a summary of all the features used. Most lag values (`date_shop_avg_item_cnt`, `date_avg_item_cnt`, `item_cnt_month` and `delta_price`) are motivated by the analysis in Section 2.2.

When creating lag features for time-series forecasting, it is necessary to discard some rows of the data because the lag features are constructed by shifting the values by a certain number of steps. For example, if you create a lag feature by shifting the time-series by one time-step, you will lose the first row of data because there is no prior time-step to use for the calculation of the lag feature. The loss of data from the shifting process means there are a number of gaps (i.e. NaN values) in the dataset which we choose to set to zero. Furthermore, we choose to exclude data corresponding to the earliest date block from the training of the model because this does not contain any useful data about the lag features. The rows in this block would be almost all set to zero values since we do not have information about the months prior to this, hence it would be counterproductive to include such data into the training.

3.3 Models

While a number of different machine learning models were explored in this project, we decided to focus on two in particular: XGBoost (Chen & Guestrin 2016) and LightGBM (Ke et al. 2017). These are both well suited for regression tasks such as our time series forecasting problem. Both models rely on gradient boosting, a machine learning technique for improving model predictions using an ensemble of decision trees. Please refer to Section B for a description of these concepts.

XGBoost improves upon simple gradient boosting by introducing regularization terms to reduce overfitting. Typically, XGBoost uses a regularized (e.g. L1 and L2) objective function that penalizes the loss function. During training, new trees iteratively predict the errors (or *residuals*) of previous weak learners and combine their outputs until convergence. XGBoost also employs a selection of hyperparameters and further improvements to the gradient boosting algorithm. While the structure of LightGBM is similar to that of XGBoost, a key difference pertains to its decision making. For LightGBM, the decision tree algorithm uses a leaf-wise (vertical) approach in contrast to the standard level-wise (horizontal) approach used by XGBoost. For a description of these different tree-growth strategies, please see Section B .

For this project, we experimented with a range of different tuning parameters for our XGBoost and LightGBM models as well as combining their results through a model blending approach. Blending is an ensemble technique to combine different models using a so-called 'meta-model' that is trained on the outputs of the base models (i.e. XGBoost and LightGBM in our case). Blending differs from stacking in that the meta-model is not fit on out-of-fold predictions made by the base models (e.g. from k-fold cross-validation), but instead it is fit on predictions made on a holdout dataset (i.e. the validation set). Generally, the predictions made by the base models on the validation sets are fed into the meta-model as input for training. However, we decided to train the meta-model on the entire training set rather than the validation set only as the best results were obtained in this

Table 1: Main Parameters of the XGBoost and LightGBM models performing best on the test set. Here, η corresponds to the learning rate. Other notable parameters of LightGBM are the minimum data in a leaf which is set to 10 and a feature fraction of 0.7.

Model	Max. depth	Num. of leaves	Min. child weight	Column sample by tree	Subsample	η
XGBoost	10	1000	0.5	0.8	0.8	0.1
LightGBM	no limit	1100	10^{-3}	1.0	1.0	0.04

way. The final predictions are then computed using the test predictions made by the base models as input into the meta-model.

3.4 Evaluation Procedure

For evaluating each model, the holdout validation method was used. When the dataset was split to make the training and test set, a further split was made to create the validation set. The validation set is used to evaluate the initial performance of the model during the training process. The hyperparameters of each model are tuned on the validation set which helps the model better generalize to unseen data. The training set is then employed to exclusively train the model while the test set is used to evaluate the final performance of the tuned and trained model. It should be noted that the test sets contained data corresponding to the final date block only (i.e. month 34). We only use this data as predictions for the following month because it is the most complete, meaning it does not contain any gaps from the shifting process when constructing the lag features (since this data has not been shifted a time-step into the future). The validation sets contain data form the date block before this, month 33, since we expect the sales in this month to closely match the behaviour of that in the test set. Finally, the root mean squared error was the metric used to evaluate the performance of each model, a standard approach in the case of regression models.

4 Results

We trained our models using the processed data presented in Section 3.1 along with the engineered features given in Table A1. For each model we employ a early stopping condition of 20 training epochs, based on the *RMSE* of the validation set. From this, we create a XGBoost model which achieves a *RMSE* score of 0.88567 on the test set. The parameters used for this model are summarized in Table 1. We are able to further improve our results by implementing the LightGBM model. Using this structure we achieve our best *RMSE* score of 0.85116, which is in the top 5% of the Kaggle leaderboard. It's main parameters are given in Table 1. One advantage of LightGBM is the possibility to designate categorical features that the model should focus on during training. We choose these properties to be `item_category_id`, `month`, `shop_id` and `shop_city`, which are all non-item specific features and hence are important for the sales predictions of items with no sales history in the training set (see Section 2.3). Furthermore, we find that by deviating from the number of lag features stated in Table A1, the predictions on the test set are worse. This reflects the significance of suitable feature engineering which was enabled by the autocorrelations of several time series studied in Section 2.2.

For our blending model, we experimented with different regressors in the meta-model (to combine the base models) and found the best results using an ordinary least squares linear regressor. Using the same input features and parameters as those used in our XGBoost and LightGBM models, the blending model obtained a *RMSE* score of 0.86121. Interestingly, the LightGBM model outperforms this. A possible reason why the blending model performs worse could relate to a lack of diversity between both of the base models. Since XGBoost and LightGBM operate in a similar way, some of the mistakes they make could be similar, and the blending model could be acting to reinforce each other's errors. Another reason could be that the blending model introduces an added complexity which may make it prone to overfitting, leading to results that are less accurate than any of the individual models.

5 Conclusion

We have developed several models to predict the future sales of one of the largest Russian software firms. From the exploratory data analysis, we were able to identify both useful time dependent and independent features to train our models. Furthermore, we discovered obvious outliers in the data and were able to correct for potential shop and item duplicates. One of the most important discoveries is related to the different groups of items in the test set, each of which present a different forecasting task. In our models, we attempt to overcome this by including in our training set both item-specific time-series features, which are important for predictions of items with a sales history, and non-item specific properties, which describe items without a sales history. Using this approach, we are able to produce a LightGBM model which has a $RMSE = 0.85116$ on the test set, which is in the top 5% of the Kaggle leaderboard. Hence, this model could be a useful tool for firms to optimally manage their item storage. An alternative approach to possibly improve our results is to train multiple models. One is trained on non-item specific data only and hence is specialized in predictions without item sale histories, while the other model uses item-specific features during training and is stronger in predicting items with a sales history. The final idea would be to ensemble both models to produce better results. It is important to emphasize that our models are unable to handle sudden changes in the economical situation of the world, similarly to what happened during the COVID-19 pandemic and Russian invasion in the Ukraine. To overcome this, we need to monitor sales in real time and use this data to adjust the predictions to the current situation. Alternatively, we may add parameters to our models which quantify the danger of sudden economical collapses and hence modify the predictions. This probably presents a very challenging task which may require collaboration with domain experts.

References

Chen, T., & Guestrin, C. 2016, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16 (New York, NY, USA: Association for Computing Machinery), 785–794, doi: 10.1145/2939672.2939785

Ke, G., Meng, Q., Finley, T., et al. 2017, in Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17 (Red Hook, NY, USA: Curran Associates Inc.), 3149–3157, doi: 10.5555/3294996.3295074

A Additional figures

Figure A1: Plot of range of all unique `item_price` values within the train data set.

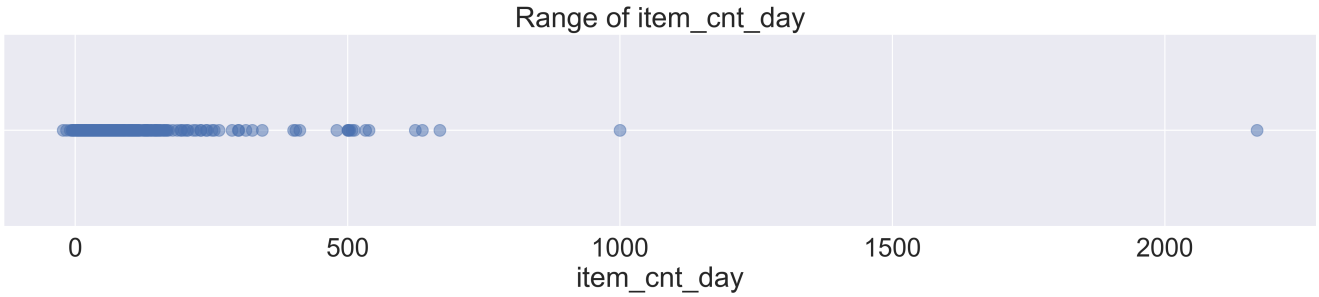

Figure A2: Plot of range of all unique `item_cnt_day` values within the train data set.

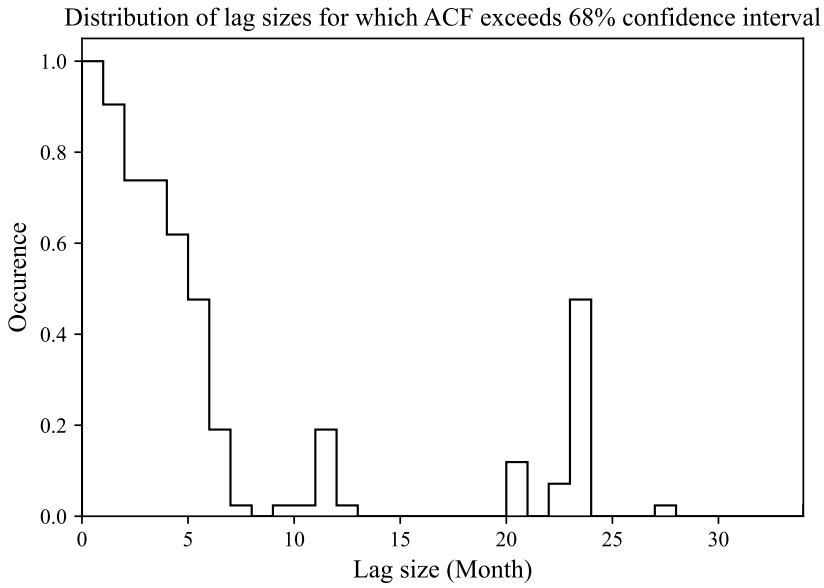

Figure A3: Distribution of lag sizes of autocorrelation values for the shop sales histories exceeding the 68% confidence interval.

B Models: Further information

As mentioned in Section 3.3, gradient boosting uses an ensemble of decision trees in its operation. The decision tree algorithm consists of a hierarchically structured sequence of tests or conditions that follow the shape of a tree. A tree is divided into nodes which begin at the 'root' and extends downwards to the 'leaf' nodes, where each leaf is connected by a condition which dictates which direction the algorithm will take. The value of the final leaf is the prediction of the decision tree and the set of leaves the algorithm took to reach this prediction is known as the inference path. In gradient boosting, new trees (known as *boosted* trees) are created to fix any errors made in prior trees which were 'weak learners'. The name is coined from the gradient descent algorithm, which is used to minimize the loss function as the model trains. Trees are continuously added until no further improvements can be made to the model. In our problem case, the weak learners of a decision tree ensemble are called the regression trees. In order to optimize the output of each regression tree, the objective function must be minimized.

As introduced in Section 3.3, a key difference between the XGBoost and LightGBM models pertain to their decision making. For LightGBM, the decision tree algorithm uses a leaf-wise (vertical) approach in contrast to the standard level-wise (horizontal) approach used by XGBoost. These strategies define how the nodes are split (i.e. it dictates the order in which the tree is expanded) and hence determines the direction the algorithm will take from the root to the final solution. In the level-wise approach, the best possible node to split is found and split down one level, resulting in a symmetric tree where the depth is extended. LightGBM's leaf-wise approach works to locate the leaves that will maximize information gain and hence minimize the loss by the most. These leaves are then split in the same level, resulting in an asymmetrical tree. Compared to level-wise, this strategy is considerably faster and results in a greater loss reduction and higher accuracy. In general, the leaf-wise approach performs better but is susceptible to overfitting on smaller datasets, hence the level-wise approach is considered more appropriate when the dataset is small.

Table A1: The features used as input into our models. These consist of the features in the dataset deemed most relevant as well as additional features and lag features that were engineered from the original data.

Feature	**Description**
date_block_num	a consecutive month number
shop_id	unique identifier of a shop
item_id	unique identifier of an item
item_cnt_month	# of items sold per month
item_category_id	unique identifier of item category
name2	secondary item name extracted from the original name
name3	tertiary item name extracted from the original name
type_code	label encoding of the first part of the category name
subtype_code	label encoding of the second part of the category name
shop_category	category of items sold by shop
shop_city	city where shop is located
shop_cluster	k-means clustering applied to shop categories
item_cnt_month_lag_1	# items sold per month lagged 1 month
item_cnt_month_lag_2	# items sold per month lagged 2 months
item_cnt_month_lag_3	# items sold per month lagged 3 months
date_avg_item_cnt_lag_1	avg. # items sold per month lagged 1 month
date_item_avg_item_cnt_lag_1	avg. # items sold per month/item lagged 1 month
date_item_avg_item_cnt_lag_2	avg. # items sold per month/item lagged 2 months
date_item_avg_item_cnt_lag_3	avg. # items sold per month/item lagged 3 months
date_shop_avg_item_cnt_lag_1	avg. # items sold per month/shop lagged 1 month
date_shop_avg_item_cnt_lag_2	avg. # items sold per month/shop lagged 2 months
date_shop_avg_item_cnt_lag_3	avg. # items sold per month/shop lagged 3 months
date_shop_avg_item_cnt_lag_4	avg. # items sold per month/shop lagged 4 months
date_shop_subtype_avg_item_cnt_lag_1	avg. # items sold per month/shop/item/subtype lagged 1 month
date_city_avg_item_cnt_lag_1	avg. # items sold per month/city lagged 1 month
date_item_city_avg_item_cnt_lag_1	avg. # items sold per month/city/item lagged 1 month
delta_price_lag_1	how current month avg. price relates to global avg, lagged 1 month
delta_price_lag_2	how current month avg. price relates to global avg, lagged 2 months
delta_revenue_lag_1	how current month revenue relates to global avg, lagged 1 month
month	month number
days	day number
item_shop_first_sale	month of each shop-item pair's first sale
item_first_sale	month of each item's first sale