CMPT 741: Data Mining (Fall 2019) – Final Project

Date: 15th December 2019

Predict Future Sales:
Final project for "How to win a data science competition"

Team DataScavengers: Anurag Bejju – abejju@sfu.ca, Savitaa Venkateswaran - savitaav@sfu.ca

1. Problem Statement:

Predicting sales performance is one of the key challenges every business faces. It is important for firms to predict customer demands to offer the right product at the right time and at the right place. The motive behind this challenge is to predict future sales (sales for the upcoming month) for every shop and item pair, given *3 years' historical sales* data ranging between years, *2013 to 2015* (including). This historical data has been generously provided by a Russian software firm named, *1-C company* **[1]** . However, the dataset has sales figures depicting per shop per item per day, in contrast, to the ask of the challenge (to predict monthly sales). The highlight of the dataset presented is that the list of shops and items vary every month (i.e.) the shops and items present in the dataset aren't bound to have sales every single month. Adding on, the item name, item categories, etc., are in Russian- making it harder to interpret at first glance. All these cumulatively add up to the complexity of the challenge.

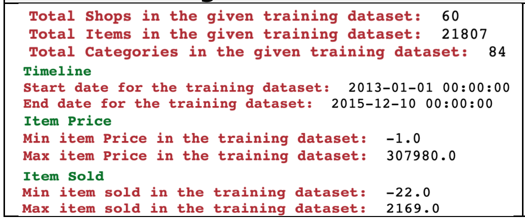
2. Machine Learning Pipeline

We intend to build a model that would provide an accurate a priori prediction about total sales for any given product and store that sells it, on the basis of historical time-series sales data, using machine learning algorithms. The below sections details each and every stage implemented as part of our machine learning pipeline.

2.1. Phase 1: Dataset Identification and Exploratory Data Analysis:

Using the presented data at its earliest stages, we have performed basic data analysis, including plotting sum and mean of *item_cnt_day* for each month to find some patterns, exploring missing values, inspecting test sets, etc. Here are some ETL tasks as well as exploratory data analysis that was performed to engineer meaningful features for our prediction model.

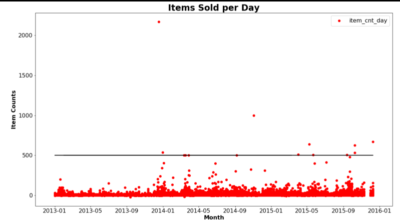
Understanding the Dataset

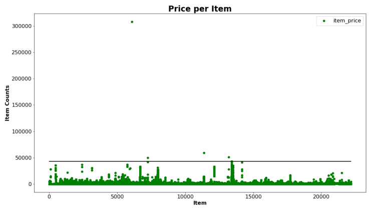


```
 Total Shops in the given training dataset:   60
 Total Items in the given training dataset:   21807
 Total Categories in the given training dataset:   84
Timeline
Start date for the training dataset:  2013-01-01 00:00:00
End date for the training dataset:  2015-12-10 00:00:00
Item Price
Min item Price in the training dataset:   -1.0
Max item Price in the training dataset:   307980.0
Item Sold
Min item sold in the training dataset:   -22.0
Max item sold in the training dataset:   2169.0
```

There are a total of *60 shops*, *21807 items* classified into 84 *categories*. This data is for three years starting from 1*st January 2013* to *10th September 2015*. Talking about min-max, the cheapest item is *-1* and the expensive item was for *307980*. We have also observed that there was a min of *-22* items sold (could be returned) and a maximum of *2169* on a day.

Checking for outliers in price and items sold features





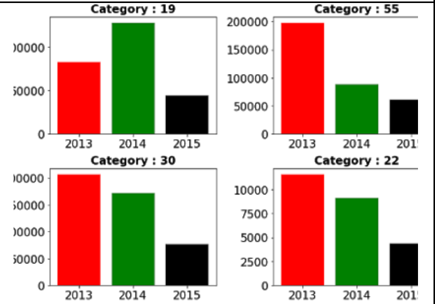
From the above plots, we have decided to remove outliers from data. Our selected range for item *price* is between *0* and *43000* and for *item count* per day is between *0* and *500*.

Apart from removing outliers, tasks like calculation of aggregation feature for each month on shop_id and item_id, shop_id only, item_id only, category_id only were performed. A 33-month history for each shop/item pair was created and standardize numerical features to fit our model. Only for tree-based features, scaling was not performed since they do not affect the model performance. We have used TfidfVectorizer to transform item_name and category_name into vectors and generated mean encoding for all categorical features using the expanding mean method.

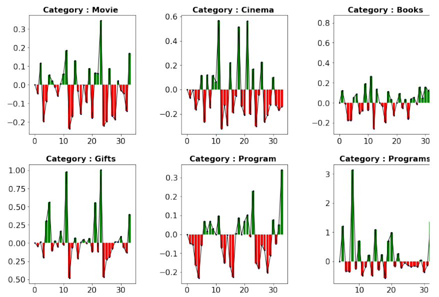
2.2. Phase 2: Exploratory Data Analysis

We have performed the following exploratory data analysis (EDA) in order to analyze the given data sets to better summarize their main characteristics through visual methods. This helped us to gather everything data can tell us beyond the formal modeling or hypothesis testing task.

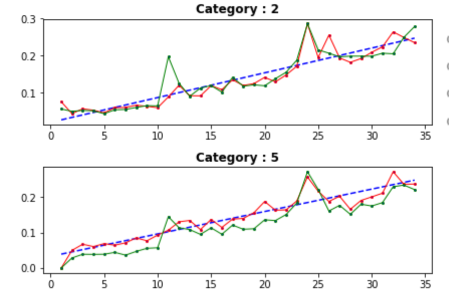
Yearly Item_Count Trends



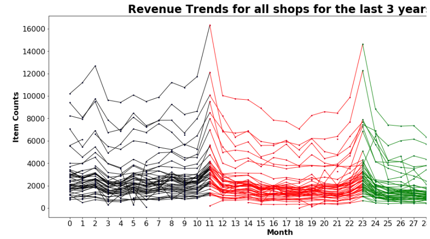
Monthly Revenue Trends



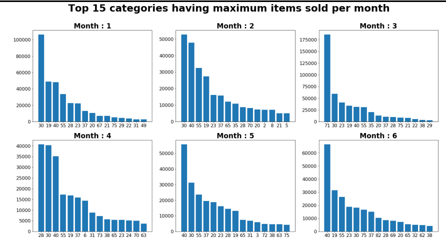
Revenue and Item count trends



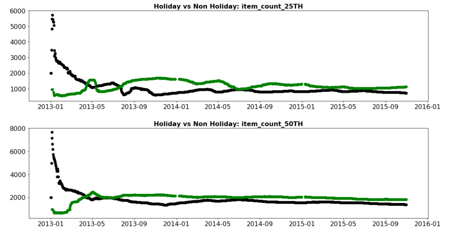
Revenue Trends for all shops



Top Items per Month



25th, 50th and 75th percentiles



Important takeaways:

We have performed a wide range of EDA tasks and have chosen a few really interesting findings to elaborate more. Since we are dealing with time-series data, we have classified our analysis into 3 main areas *[2]*:

Trend components: We have affectively used plots like Yearly Item_Count Trends, Monthly Revenue Trends as well as Revenue and Item count trends to analyze long-term non-seasonal patterns in the data, such an upward or downward linear trend in sales per month.

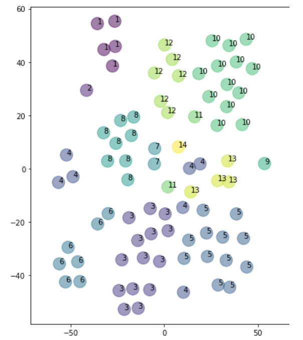
Seasonal components: Plots like Revenue Trends for all Shops and Top Items per Month helped us understand periodic patterns in the time-series data. For the given dataset, which entails predicting monthly sales values, there is likely going to be a seasonal component associated with it.

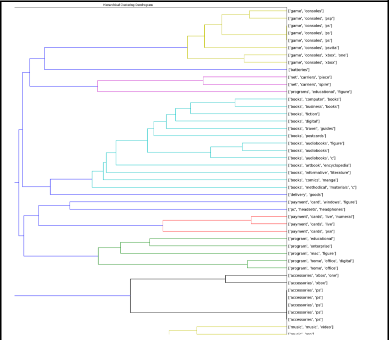
Examples of seasonal components in our data include spikes in sales for months 11 and 23 or a significant rise in items sold for 71 categories in the month of March.

Remainder components: Here we have tried to capture any changes in the data not captured in the trend or seasonal components. Things like 25th, 50th and 75th percentile for item count or clustering (elaborated in the next section) really helped us in generating unconventional but effective features in the next phase.

Clustering Techniques

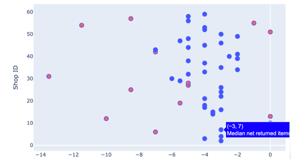
Agglomerative Hierarchical Clustering

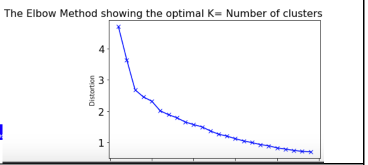


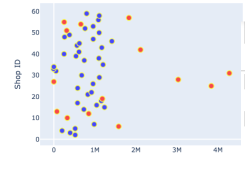


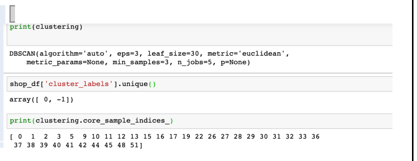
During our assessment, we found that the categories used can further be reduced to 17 through *text-based Agglomerative Hierarchical Clustering*. We have converted each word into a 2-dimensional vector space and used cosine similarity to calculate the distance.

DBSCAN









```
print(clustering)

DBSCAN(algorithm='auto', eps=3, leaf_size=30, metric='euclidean',
    metric_params=None, min_samples=3, n_jobs=5, p=None)

shop_df['cluster_labels'].unique()

array([ 0, -1])

print(clustering.core_sample_indices_)

[ 0  1  2  3  5  9 10 11 12 13 15 16 17 19 22 26 27 28 29 30 31 32 33 36
 37 38 39 40 41 42 44 45 48 51]
```

DBSCAN *[3]* helped us establish a common trend in relation to the buy/sell of different items between shops. We observed that there isn't a very close association between the shops but there is indeed a range within which the sales seem to happen.

2.3. Phase 3: Feature Engineering

Once we have preprocessed our data and analyzed various aspects/ dimensions of it, we have tried to generate a bunch of features that would successfully support our model. While constructing features, since it's a time series problem we tried to include historical features, as well as time window features (i.e mean, median, std, etc). Initially, we created new categorical features [*year, month* and *day*] [4] from date feature. Then based on our observation from our EDA, there was periodicity in the number of monthly product sales, so generating l*ag features* were found to be beneficial. These features were created for items, shops, categories, type, subtype, the city with 1,2,3,6,12 intervals [5]. Intervals were chosen based on their impact on our target variable. We also created derived features like revenue, months since the last sale for a particular shop, item pair and, similarly the first sale for a particular shop item pair to improve our model. Finally, the monthly product sales were clipped to (0,20) as suggested in the competition guidelines.

Categorical Features	Lag Features [1,2,3,6,12]	Derived Features
date_block_num, shop_id, item_id, item_cnt_month, item_category_id, city_code, type_code, subtype_code, month, day	*MEA item count per month and per item* *MEA item count per month per shop* *MEA item cnt per month per item category and per shop* *MEA item cnt per month per item type/subtype per shop* *MEA item count per month per city code* *MEA item count per month per item type/subtype* *Delta Price and Revenue Lag*	*item_shop_last_sale,* *item_shop_first_sale,* *item_last_sale,* *item_first_sale* *is_holiday_season*

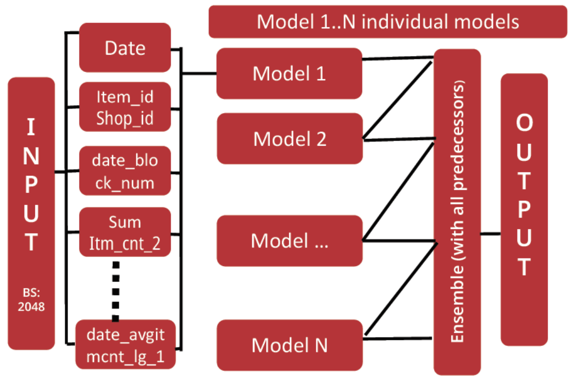
2.4. Phase 4: Model Building

In this section, we will explore 3 state-of-the-art methods (*Prophet, Sales Ranking LSTM* and *Fast ES-RNN*) and compare it with our naïve approach (*XGBoost*).

2.4.1. <u>Naïve Approach: XGBoost</u>

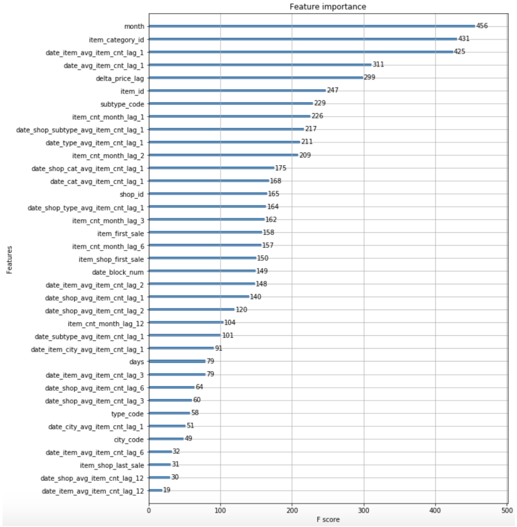
XGBoost (Extreme Gradient Boosting) is an extremely optimized gradient boosting algorithm through parallel processing, tree pruning, handling missing values and regularization to tackle overfitting (or) bias. It's an open-source library with an underlying C++ codebase combined with a Python interface sitting on top, making it a powerful yet easy to implement an ensemble learning technique. [6].

2.4.1.1. Architecture

Design	XGBoost Algorithm
 Model 1..N individual models Date Item_id Shop_id date_block_num Sum Itm_cnt_2 date_avgit mcnt_lg_1 INPUT BS: 2048 Model 1 Model 2 Model ... Model N Ensemble (with all predecessors) OUTPUT	**Algorithm 1:** XGboost algorithm **Data:** Dataset and hyperparameters Initialize $f_0(x)$; **for** $k = 1, 2, \ldots, M$ **do** Calculate $g_k = \frac{\partial L(y,f)}{\partial f}$; Calculate $h_k = \frac{\partial^2 L(y,f)}{\partial f^2}$; Determine the structure by choosing splits with maximized gain $\mathbf{A} = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right]$; Determine the leaf weights $w^* = -\frac{G}{H}$; Determine the base learner $\hat{b}(x) = \sum_{j=1}^{T} wI$; Add trees $f_k(x) = f_{k-1}(x) + \hat{b}(x)$; **end** **Result:** $f(x) = \sum_{k=0}^{M} f_k(x)$
Boosting is technically an ensemble learning method in which, many models are combined together to obtain the final prediction. Boosting trains models in succession, with each new model being trained to correct the errors made by the previous ones. Models are added sequentially until no further improvements can be made. The advantage of this iterative approach is that the new models being added are focused on correcting the mistakes which were caused by other models. Gradient Boosting is a type of boosting where the weights for wrong outcomes are not incremented, instead, it tries to optimize the loss function of the previous learner by adding a new model that adds weak learners in order to reduce the loss function. The main idea here is to overcome the errors in the previous learner's predictions.	

2.4.1.2. Model Assessment

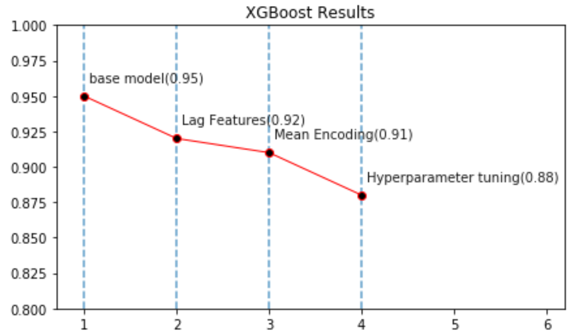
Since XGBoost is a sure shot performer, we used it as our baseline aka benchmark approach to comparatively evaluate/study the other state-of-the-art methods that we had also implemented. Moreover, being an ensemble learning technique (a group of models working in succession to learn to correct its predecessors mistakes) with a distributed framework support, it helped us understand the importance- different features (statistical features, time-based features, lag features based on various attributes from the dataset, holiday based attributes, categorical features, price, and revenue-based features, etc,.) had on the dataset and helped us get an intuition about the same. We can say so because due to its high-end optimization, it consumed a lot fewer resources and time- thus, making it an efficient way to search through the feature space for good tuning parameters (as well as effective features). Also, we made sure that each feature added to the set had a prominent effect on the final prediction by using statistical methods such as performing F-test and checking for P-values. Finally, our cut-off strategy was to let the model keep training until there is no decline in RMSE for 10 consecutive epochs.



As clearly seen, our created feature called *‘month’* which is a modified ‘*date_num_block*’ attribute from the original dataset attribute has the greatest effect on our prediction and it makes complete sense as we are trying to use the dataset to predict a sales figure for a month forward. Secondly, our time-based lag features which we had used with XGBoost seems to have helped the model performs well on this time-series data again making it intuitive that the lags helped the model understand the pattern before an event or occurrence which in-turn helped XGBoost generalize certainly well despite the odds of the dataset (outliers, missing values, differently ranging values, never-before-seen patterns in data, etc,.). Similarly, features related to shops, items, prices, and its associated lag features have seemed to be of tremendous help to our model in making the final predictions. Again this seems fair enough as we are trying to predict sales for a particular shop-item pair.

Surprisingly, *’subtype_code’* a newly found and added a feature by us using the results from our clustering approach (we figured out the names of shops had cities in it)- has been highly useful in making final predictions. We had embedded this feature with the idea that demand shall vary geographically (like, Venice's sales may be different from Moscow). Finally, we can observe that shorter lag periods of 1,3,6 months have done well over 12-month lags. This again makes sense as sales (it's time-dependent) does not have a definite pattern which follows through years, it's affected by various natural reasons that may be outside our control as well.

2.4.1.3. Results



As a base model, we considered features (like, without incorporating lag based attributes) and were able to attain a test accuracy (on the Kaggle test set) of 0.95. Later, we enhanced our feature set with a list of holidays and added time, price, shops, and items based lag features, which got us a slight improvement in the accuracy to about 0.92. Since our data has been modified to be grouped on a monthly basis (short-term prediction), we substantially added lag features for up to 12 months but not more.

Further, we added some additional features like *‘month’*, *‘subtype_code’* (obtained from our clustering results), features based on revenue and as well performed extensive hyper-parameter tuning by searching through the entire feature space. Also, we removed and tweaked certain features based on statistical evaluation techniques to make sure our feature set had only features that really made an impact on the final prediction result. This move helped us get a significant boost in our RMSE score- our current leaderboard score of ***0.899.***

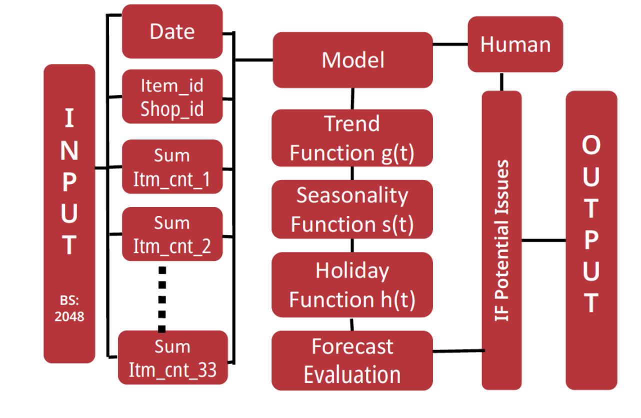
2.4.1.4. Verdict

Our best test accuracy was obtained (Kaggle, evaluation metric: RMSE) with XGBoost. It was an easy yet extremely powerful modeling approach that had great offerings pertaining to time-series predictions. Being an extremely flexible model (a simple model is all you have and everything else is up to you to design), it gave us greater independence in detailing through the entire feature space. It gave us greater autonomy in deciding features and coming up with an appropriate model design for the challenge in hand.

2.4.2. Prophet

Typically, when working with a forecasting model, we intended to be able to tune the parameters with regard to the specific problem at hand. For instance, the first input parameters to automated *ARIMA*, are the maximum orders of the differencing, the auto-regressive components, and the moving average components. Since it's a daunting task for an analyst to know the right way to adjust these orders it becomes a hard problem to resolve and scale. *Facebook* developed an open-source project called *Prophet*, *[7]* a forecasting tool available in both Python and R. It provides intuitive parameters that are easy to tune. Even someone who lacks deep expertise in time-series forecasting models can use this to generate meaningful predictions for a variety of problems in business scenarios.

2.4.2.1. Architecture

<table>
<tr><th>Design</th><th>Prophet Equation</th></tr>
<tr><td>
</td><td>

$$\boldsymbol{y}(\boldsymbol{t}) = \boldsymbol{g}(\boldsymbol{t}) + \boldsymbol{s}(\boldsymbol{t}) + \boldsymbol{h}(\boldsymbol{t}) + \in_{\boldsymbol{t}}$$

$\boldsymbol{g}(\boldsymbol{t})$ is the trend function which models non-periodic changes in the value of the time series
$\boldsymbol{s}(\boldsymbol{t})$ represents periodic changes
$\boldsymbol{h}(\boldsymbol{t})$ represents the effects of holidays.

The error term is any idiosyncratic changes which are not accommodated by the model; later we will make the parametric assumption that it is normally distributed.</td></tr>
<tr><td colspan="2">This model is very similar to the <i>generalized additive model (GAM)</i>, a class of regression models with potentially non-linear smoothers applied to the regressors. In this case, Prophet uses only time as a regressor and models seasonality as an additive component mimicking the exponential smoothing method. It also allows multiplicative seasonality, where the seasonal effect is a factor that multiplies g(t), can be accomplished through a log transform. This model decomposes easily and accommodates new components as necessary.</td></tr>
</table>

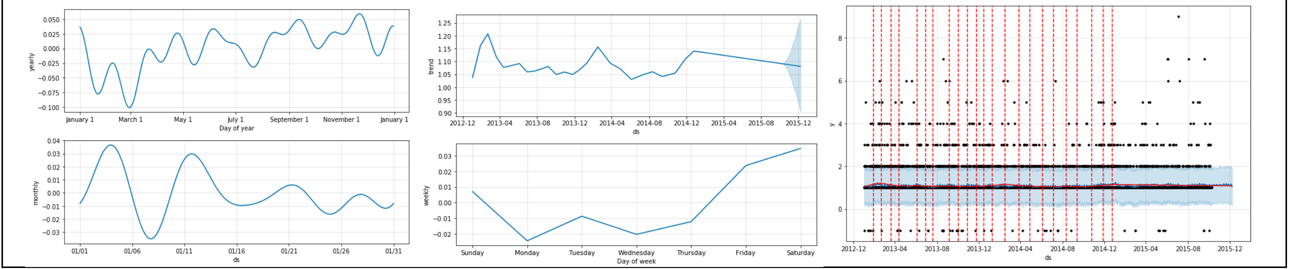
2.4.2.2. Model Assessment

We have decided to work with the prophet for this project as it can be implemented for a general-purpose time series model that considers *piecewise trends, multiple seasonality* (day of the week, day of the year, month, etc.), as well as *floating holiday*s. Prophet frames its time series forecasting problem as a curve-fitting exercise and the dependent variable is a sum of three components: *growth, periodic seasonality,* and *holidays*.

Based on our EDA, features having nonlinear growth uses a logistic growth model with time-varying carrying capacity and those having linear growth, uses a simple piecewise constant function. We also observed several changepoints in trends for features like revenue and item_count, and prophets address it with its changepoints functionality that uses a vector of rate adjustments, each corresponding to a specific point in time. This rate adjustment variable is modeled using a Laplace distribution with location parameter of 0.

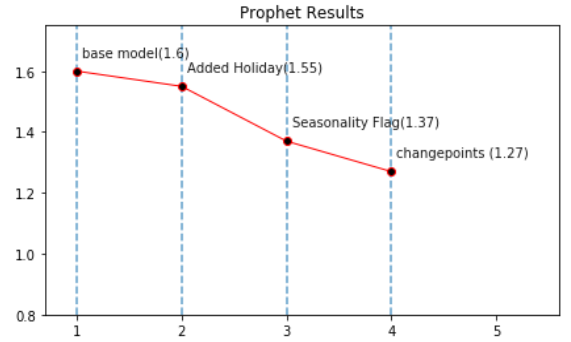
We have played around with changepoints by providing 33rd month as validation set or by adjusting the scale parameter associated with the Laplace distribution. Finally, we have passed the web scrapped Russian holidays list for the last three years into the indicator function of our model that takes 1 on holidays and multiplies by a normal smoothing prior. In the end, we observed that this model uses periodic seasonality with a standard Fourier series and the seasonal component is smoothed with a normal prior.

Prophet Analysis



We were very impressed by the level of sophistication it has provided us by effectively utilizing periodic, seasonal and holiday-specific features. As you can see in the above graphs, the model very well understands the trends and generalizes well to seasonal changes. You can also witness the same peaks observed in our EDA considered by the model. Based on these simple, interpretable results for the components (date, year, month) of the time series decomposition, we can safely say that it is a great approach to deal with sales data.

2.4.2.3. Results



Initially, we have started with the basic model without *seasonality, additional features* or *changepoints* and received a test score of *1.6* on it. Then we passed a list of holidays and got a slight but not significant improvement in the accuracy. Since our data is grouped on a monthly basis, the holiday effect gets diminished a little and the prophet cant generalizes well. Later on, we added some additional features like item_id, shop_id and turned on the monthly seasonality flag resulting in a significant boost (-17%) in our RMSE score. Finally, we adjusted the scale parameter associated with the Laplace distribution and got a final score of *1.27* for our test set.

2.4.2.4. Verdict

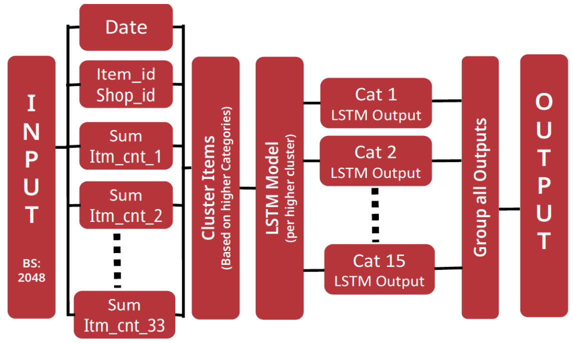
Prophet was a very simple yet effective modeling technique that had some great functionalities catered to time-series predictions. It has done an amazing job of assessing seasonality and trends and generalized well on the whole. We were not able to achieve a better result compared to our naïve approach (XGBoost) as the prophet performs poorly if there are high irregularities in seasonality data. Also, Exogenous variables affecting the time series are not taken into consideration by the model. Finally, it's extremely difficult for the prophet to consider some categorical or derived features as part of the model building process.

2.4.3. Sales Demand Forecast in E-commerce using a Long Short-Term Memory Neural Network Methodology

This *Long Short-Term Memory network (LSTM)* model developed by researchers at *Walmart* **[8]**, exploits the non-linear demand relationships available on an E-commerce platform using features like sales demand and sales sparsity to group products. Since most of the current state-of-the-art techniques are typically univariate methods, which produce forecasts considering only the historical sales data of a single product, this model caters to data with large numbers of the shop – item combinations, in which the sales demand patterns can be correlated to build a unified model. They have suggested several product grouping strategies (which we used) to supplement the LSTM learning schemes, in situations where sales patterns in a product portfolio are disparate.

2.4.3.1. Architecture

Design



LSTM Equation

$$
\begin{aligned}
i &= \sigma(x_t U^i + s_{t-1} W^i) \\
f &= \sigma(x_t U^f + s_{t-1} W^f) \\
o &= \sigma(x_t U^o + s_{t-1} W^o) \\
g &= \tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i \\
s_t &= \tanh(c_t) \circ o \\
y &= softmax(V s_t)
\end{aligned}
$$

i : input gate, how much of the new information will be let through the memory cell. f : forget gate, responsible for information should be thrown away from memory cell. o : output gate, how much of the information will be passed to expose to the next time.

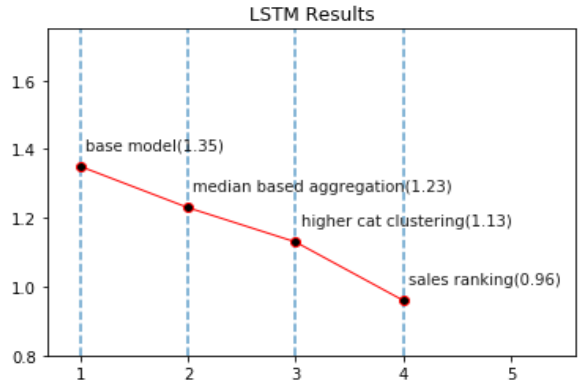
LSTMs are an extension of RNNs that have the ability to learn long-term dependencies in a sequence, overcoming the limitations of vanilla RNNs. In this paper, they have used a special variant of LSTMs, known as "*LSTM with peephole connections*" that requires the LSTM input and forget gates to incorporate the previous state of the LSTM memory cell. Instead of feeding in a single observation at a time, they have used the Moving Window procedure to create an array of lagged values as the input data for the LSTM. This essentially relaxes the LSTM memory regulation and allows the network to learn directly from them.

2.4.3.2. Model Assessment

We have decided to work with this paper as the Walmart dataset was very similar to our Kaggle competition dataset as well as the approach used was unique and thoughtful. Since the business environment in E-commerce is highly dynamic and often volatile, which is largely caused by holiday effects, low product-sales conversion rates, this behavior resonates with 1C company data as well. As a result, we can see the same set of challenges (eg: highly non-stationary historical data, irregular sales patterns, sparse sales data, highly intermittent sales) faced specifically in this domain to estimate demand data of a particular product. One more interesting similarity was the inherent hierarchical structure present for item categories with each subgroup of the hierarchy containing similar items. We have used this hierarchical structure to naturally group products in which the sales patterns can be correlated.

According to *[9]* and the professor's feedback, employing a time series grouping strategy can improve the LSTM performance in situations where time series are disparate. Therefore, we decided to work with two product grouping mechanisms: the first being, the target products grouped based on sales ranking and sales sparsity (the percentage of zero sales) and the second being, the target products grouped based on higher categories (done through text-based hierarchical clustering). Both these approaches were implemented and the paper and fit perfectly with our cause.

2.4.3.3. Results



Initially, we have started with the basic model using the whole dataset and aggregating by median and received an RMSE test score of *1.23* on it. With our multiple time series data containing large quantities of related and similar information that can be grouped, this approach allowed us to exploit the similarities between them. By accounting for this notion, the overall accuracy was drastically improved. Grouping based on a higher category resulted in a score of *1.13* whereas grouping based on sales ranking and sparsity resulted in an exceptional score of *0.96* (22% decrease). This was possible because of the grouping methodology that clusters seasonal (low sales ranking high sparsity) and frequently bought (high sales low sparsity) separately.

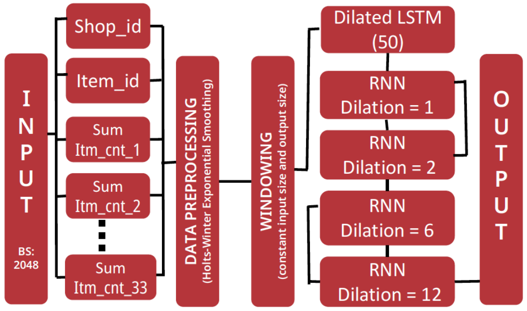
2.4.3.4. Verdict

Overall, this approach was a great learning ground to implement and explore various state-of-the-art methods associated with time-series data w.r.t neural networks. Specifically, it showed us a new technique to use clustering in combination with LSTM to get better and accurate predictions. This hasn't performed as good as our naïve model due to a lot of sparse and uneven data present for each shop/item pair. Since this drastically affects the overall performance of LSTM, we couldn't better this score.

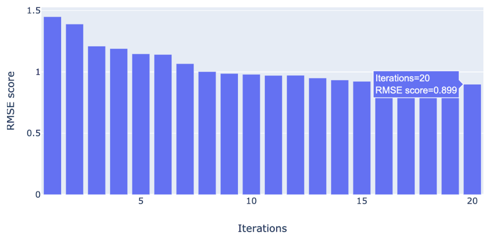
2.4.4. Fast ES-RNN

Fast ES-RNN (Exponential Smoothing-Recurrent Neural Networks) ***[10]*** was designed to compete in the M4: Time Series Forecasting competition held in the year 2018 and originally designed by folks from *Uber*. It was designed to make state-of-the-art forecasting fast, accessible, and generalizable. ES-RNN is a hybrid between classical state-space forecasting models and modern RNNs. Crucially, ES-RNN implementation requires per-time series parameters. By vectorizing the original implementation and porting the algorithm to a GPU (python interface and PyTorch computation), the authors (r*evised GPU implementation by folks from CMU, USA*) achieve up to 322x training speedup depending on the batch size with similar results to the original submission.

2.4.4.1. Architecture

<table>
<tr><th>Design</th><th>Holts-Winter exponential smoothing</th></tr>
<tr><td>
</td><td>$$\widehat{y_{t+h}} = l_t b_t^h s_{t-m+h_m}$$ where l is state variable for level, b is a state variable for trend and s is a multiplicative seasonality coefficient. Since the Holt-Winters and the RNN methods were merged to create the final model. $y_{\widehat{t+1\ldots t+h}} = RNN(X_t) * l_t \ * s_{t+1\ldots t+h}$ Where X_t is a vector of normalized, de-seasonalized features..</td></tr>
<tr><td colspan="2">Fast ES-RNN is a deep learning technique that employs <i>stacked dilated LSTMs</i> (LSTMs with skip connections) and has been explicitly designed to cater to the needs of time-series prediction with generalized designs that can predict monthly, quarterly and yearly forecasts. The advantage of using Dilated LSTMs over vanilla LSTMs is that it greatly increases computational efficiency and allows the network to remember information from earlier time instances. Due to the prevalence of time series forecasting is crucial in multiple domains. Since our challenge focuses on predicting monthly sales figures, we have considered the monthly variation of the Fast ES-RNN model and further, tweaked the model to suit our dataset. However, the output is a deseasonalized and normalized value which is not the case of our truth data. Therefore we need to re-seasonalized and de-normalize our outputs, <i>Holts-Winters equations</i> and the parameter estimates we have for each series are used to arrive at the desired output.</td></tr>
</table>

2.4.4.2. Model Assessment

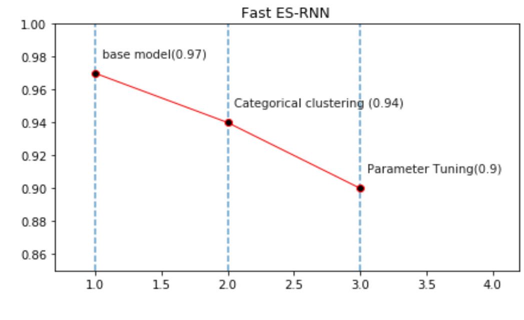


The best thing about ES-RNN is that we are able to tune per-time series parameters, that is for every category (item categories) in the dataset you can potentially train/tune a model separately pertaining to it (originally designed by the authors itself). This is exactly why Fast ES-RNN was quite close in its accuracy to our baseline model as it was able to model uniquely item-shop pair's sales forecast belonging to different clusters (pairs belonging to the same cluster tend to behave similarly), beating other state-of-the-art models we had implemented.

Adding-on, the M4 competition for which it was originally designed for has striking similarities to our dataset under consideration. For instance, like our challenge is to predict sales for one month forward for an item-shop ID pair (each pair has a time series for itself), the M4 competition too had a month ID based prediction task but asking for 18 months forward-looking model. Further, the M4 dataset also had categories as an important aspect to help club IDs together in-order to help the model intuitively understand that it can club relatively similar time-series patterns together (club IDs that have common patterns together).

This will eliminate the overfitting/bias possibilities of the model. Further, ES-RNN is a hybrid technique that combines the very famous time-series based statistical method of 'Exponential smoothing' and our classic 'Dilated LSTMs'. It is important to note that the RNN and the classical Holts-Winters parameters detailed in the architecture part are jointly trained. The power lies in the co-training of both the per-time series Holt-Winters parameters and the general RNN parameters.

2.4.4.3. Results

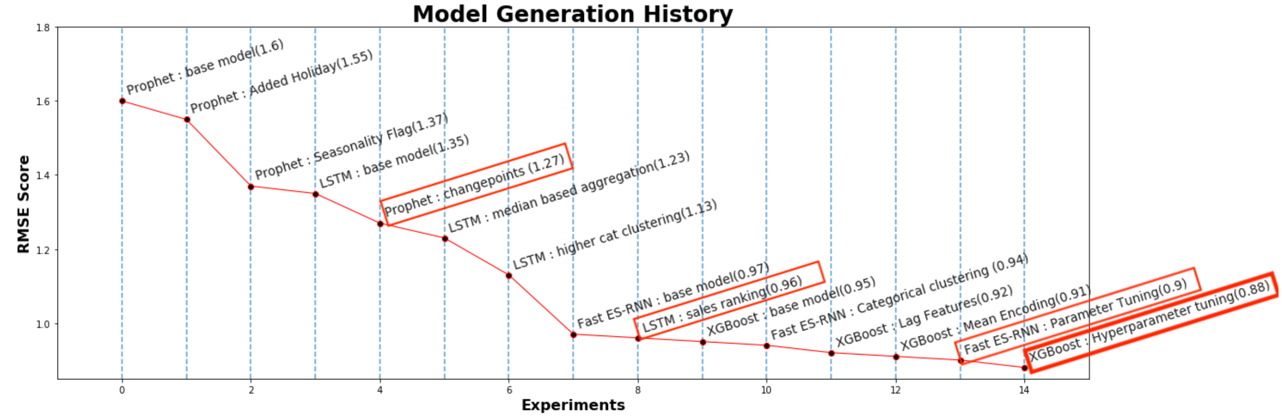


Initially, we started by using the same model as used for M4 competition (no parameters changed) due to lack of time and received a test score of 0.971 on it. Then, we changed the categories for which ES-RNN creates separate time-series predictions (per- time-series tuning) to 12 categories (using clustering results) from the originally 84 item categories in the dataset. Later, we added additional features like item_id, shop_id to the model as well, resulting in a significant boost in the RMSE score. Finally, with the time left in hand, we tried to tune the parameters and obtained a test accuracy of 0.901.

2.4.4.4. Verdict

Fast ES-RNN was an extremely effective and powerful modeling technique for this particular dataset, that had every great functionality necessary to do well on time-series predictions. It has done an amazing job of generalizing well on the whole. However, we were not able to achieve a better result compared to our naïve approach (XGBoost) as we had little time to tune the hyper-parameters of the model. Despite its improved performance using GPUs, considering we had 84 categories (too many categories for it to perform well) of items originally and 12 categories later after clustering- the model took quite a substantial amount of time to train. Finally, being a deep learning model with several stacks of LSTMs, it's extremely data-hungry and prone to bias. That's were tuning parameters to the right extent plays a major role. Also, another factor that can help improve the model performance is finding a good window size to truncate the series. Since ES-RNN uses windowing with exponential smoothing and our dataset's traits will definitely vary from M4, we need to do some analysis and benchmarking to find the apt window for our dataset.

2.5. Model Selection



After all the experimentation is over, we have settled on the best data preprocessing, model architecture, and learning parameters. During the course of this project, we have explored 3 state-of-the-art methods (*Prophet, Sales Ranking LSTM and Fast ES-RNN*) and compared it with our naïve approach (*XGBoost*). For each model, we have done a thorough study of each component and modeled it according to the needs of this competition. We have not only implemented each of these models but tried to individually tune it wit various methods suggested by the authors and improve the overall test RMSE score of it.

Model	RMSE-VE	RMSE-TE	Ranking
XGBoost (Naïve approach)	0.88	0.89	1
Fast ES-RNN	0.943	0.9	2
Sales Demand Forecast in E-commerce	0.95	0.96	3
Prophet	1.13	1.27	4

After exploring and fine-tuning each model, we have concluded that the naïve approach used by many fellow Kaggle competitors has emerged as the clear winner. With this model, we have got a test RMSE score of **0.89** and have reached an overall rank of ***353 (o.f. 5137)*** placing us in the **top 6%** of the competition. Following right behind, we have achieved an RMSE score of ***0.9*** for *Fast ES-RNN* and ***0.96*** for *Sales Demand Forecast in the E-commerce LSTM* model. Since both of them play on deep learning model with LSTMs, it's extremely data-hungry and prone to bias. It requires us to invest a lot of time to tune parameters to the right extent. Finally, Facebook's *prophet* scored ***1.27*** after exploring all the features it has to offer. This has performed relatively poor as it does not consider exogenous variables as well as some categorical or derived features.

3. Conclusion and Future Work

We have researched and implemented on the whole 4 widely different models to 'Predict Future Sales' a month forward. As part of the project, we have followed the entire machine learning pipeline and performed tasks like *data preprocessing, exploratory data analysis, feature engineering, model generation, evaluation, and selection*. We have also done a comparative study amongst the 4 models, of which the XGBoost (naïve approach) technique emerged as the winning model. Even Though XGBoost outperformed its counterparts (all three state-of-the-art models), Fast ES-RNN and stacked LSTM approaches showed great potential for this dataset in particular. Especially, we felt with Fast ES-RNN the gap was becoming narrower. With further tuning of its hyper-parameters (given the time) and adding some reliable features to the model, we strongly believe ES-RNN can surpass our baseline model. Also employing a time series grouping strategy in conjunction with LSTM is a novice idea and can be used in any other time series forecasting problem.

As an extension of our work done with our 4 algorithms, we would like to explore more statistical methods like combining our approaches with *ARIMA*, *SARIMA*, etc, especially with our XGBoost approach. We would like to do so as these statistical methods shall help enhance the performance of the starting weak learners (for XGBoost) themselves as they will be able to better understand the time-based variations in the dataset and would be armed in handling never-before-seen patterns with cleverer approaches (better generalization to the dataset). Also, given the time we can also further try different combinations of tuning parameters to boost its results alongside looking for enhanced features that could help make the prediction more accurate. This same shall apply for our three state-of-the-art models as well. Finally, delving deeper into understanding the feature space to try adding item/price based features to our deep learning approaches and see if that can provide them with more information that would be helpful for the prediction task.

4. Acknowledgement

We would like to express our deepest regards to *Dr. Martin Ester - Professor of Computing Science – SFU* for taking the time to provide great insights and *extending help with many aspects of this project. I would also like to thank our course TA,* Zahra Zohrevand for her constant support throughout the term as well as advice for this project. This has been a great learning experience and would be really beneficial in our careers as data scientists.

5. References

[1] https://www.kaggle.com/c/competitive-data-science-predict-future-sales/overview
[2] Introduction to Statistical Analysis of Time Series - Richard A. Davis - Department of Statistics -
[3] A density-based algorithm for discovering clusters in large spatial databases with noise
by Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu , 1996
[4] Predict Future Sales using Ensembled Random Forests - Yuwei Zhang,Xin Wu,Chenyang Gu, Yueqi Xie less
[5] https://zhuanlan.zhihu.com/p/83425772
[6] XGBoost: A Scalable Tree Boosting System - Tianqi Chen, Carlos Guestrin - University of
[7] Prophet: Forecasting at Scale - Sean J. Taylor & Benjamin Letham - Facebook, Menlo Park, US
[8] LSTM: Sales Demand Forecast in E-commerce using a LSTM Neural Network MethodologyKasun Bandara, Peibei Shi, Christoph Bergmeir, Hansika Hewamalage, Quoc Tran and Brian Seaman -2019
[9] Bandara, K., Bergmeir, C.& Smyl, S., 2017. Forecasting Across Time Series
Databases Using RNNs on Groups of Similar Series: A Clustering Approach. arXiv
[10] Fast ES-RNN: A GPU Implementation of the ES-RNN Algorithm authored by Andrew Redd, Kaung Khin, Aldo Marini from Carnegie Mellon University. This paper is based on the original work of Slawek Smyl from Uber on hybrid Exponential Smoothing-Recurrent Neural Networks (ES-RNN), which was the winning model of M4 forecasting competition held in the year 2018.