CSCI 567 Project Report

Sai Anuroop Kesanapalli
kesanapa@usc.edu
Department of Computer Science
University of Southern California

Prashanth Ravichandar
pr39760@usc.edu
Department of Computer Science
University of Southern California

Gaurav Aidasani
gaidasan@usc.edu
Department of Computer Science
University of Southern California

Kyle Manke
kmanke@usc.edu
Department of Computer Science
University of Southern California

Abstract

In this report, we will present machine learning techniques to predict store sales in the "Store Sales - Time Series Forecasting" competition [1] on Kaggle.

1 Introduction

For this project, we were asked to forecast 15 days (`2017-08-16` to `2017-08-31`) of store sales for Corporación Favorita , a large Ecuadorian-based grocery retailer. The provided data consisted of the past sales for 33 families of items from 54 stores over the period of `2013-01-01` to `2017-08-15`. Additionally, we were given information on past oil prices, holidays, and store locations to potentially aid our forecast.

For the competition, our forecasts were evaluated using the Root Mean Squared Log Error (RMSLE) metric:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log(1+\hat{y}_i)-\log(1+y_i)\right)^2} \tag{1}$$

Note, that as explored in [2], the error function is asymmetric: under-prediction is penalized more than over-prediction. This can be interpreted as prioritizing over-stocking an item so that potential sales are not lost.

We began by determining a baseline score with a basic linear regression model. (Section 2). With this score in mind, we experimented with decision tree-based models (Section 3). After unimpressive results , we further explored the provided datasets (Section 4) to identify potential trends, seasonal patterns, and features that were relevant to the forecast (Section 5). Finally, we trained various models on our enriched feature set and settled on the one that gave us our best score: a version of an `ExtraTreesRegressor` (Section 6).

2 Linear regression model

For feature preparation, we computed the 7-day moving average of oil prices (`ma_oil`) and forward filled the `NaN` values. We also created a `wd` feature, which served as a boolean indicating `1` for weekdays and 0 for weekends. Further, we mapped all dates to a trend feature with the help of the `DeterministicProcess` library. This helped the model retain the chronological information of the time series. Finally, we trained the linear regression model on this set of features and predicted on the 15 test dates. We clipped the negative values of sales to 0 for realism. By doing so, we obtained a

Kaggle score of 0.45807. This served as our base model and was our submission for Q5 of HW4. Please see notebook `CSCI567_id1.ipynb` in the submitted code for reference.

2.1 Extracting the best out of the base model

We tried to improve the basic model by incorporating various feature transformations [3]. For example, we tried scaling down the sales with `np.log1p`, varying the moving window size for oil prices, and different interpolation methods for missing data.

2.2 Creating a validation set and doing better

To test our model, we took a subset of the training data to serve as a validation set and monitored performance with the RMSLE metric 1. There were various choices for the validation set dates, and we initially opted for the last 15 days of the training dates. By doing so, we observed that there was correspondence between the validation score and the score obtained on Kaggle. We however realized that a better choice of validation dates would be the 16th to the 31st of July since the data did show monthly seasonality.

With our current linear regression framework, we experimented with different hyper-parameters and objective functions. We found that `MultiTaskLassoCV`, with its L1/L2 mixed-norm regularization [4], performed better than standard linear regression. Despite these analyses and minor improvements, we could not reach a submission score below 0.40. Thus, we decided to pivot and explore decision tree-based models.

3 Decision tree based models

3.1 Expanding the set of features made available to the base model

A natural extension was to expand the set of input features, letting the decision trees decide what is important. Doing so involved merging different features present in the various input files. This process presented a significant challenge as some merges resulted in multiple NaN values requiring a meaningful interpolation. Further, we converted all the categorical values to numerical ones.

3.2 Trying different decision tree-based models

We tried a variety of models such as `XGBRegressor`, `DecisionTreeRegressor`, `LGBMRegressor` with different sets of hyper-parameters and objective functions for each. Please see `lightgbm-mergedfeaturedf.ipynb` in the submitted code for reference. Here, however, we found that a näive implementation of decision tree-based models were overfitting the training data and performing worse than our baseline regression model. Before moving ahead, we decided to explore the data further to identify information that we were missing.

4 Visualzing trends in data

4.1 Sales per family across years

From Figure 1, it was immediately evident that the same model would not work for each family. Some families of products had a weekly trend, others exhibited monthly seasonality, whereas some, such as `SCHOOL AND OFFICE SUPPLIES` experienced relatively low sales except for peaks around April and August of every year. Clearly, we were going to need models that could capture the different trends and seasonal patterns of each family.

4.2 Periodograms

To solidify our knowledge from the previous yearly trend plots, we plotted periodograms for each product family, as seen in Figure 1. From the periodograms, we could better identify seasonal patterns. For example, the periodogram for the `AUTOMOTIVE` family shows a strong weekly pattern that corresponds to what the respective yearly graph shows.

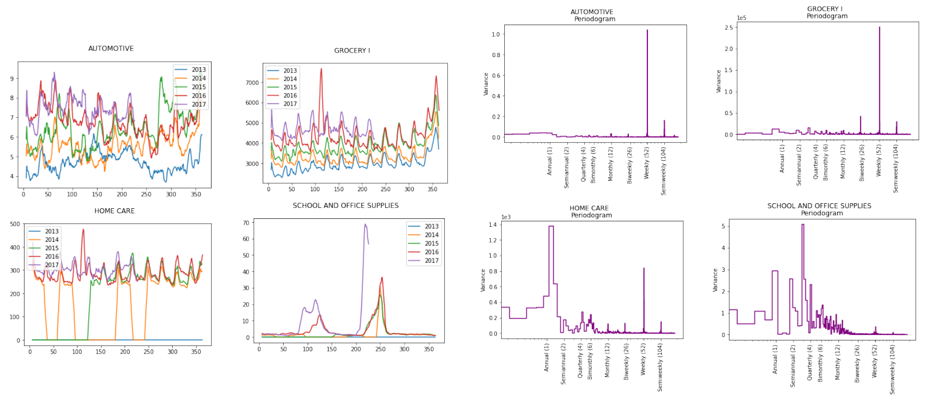
Figure 1: Sales Trend and Periodograms

4.3 Day of month and day of week trends

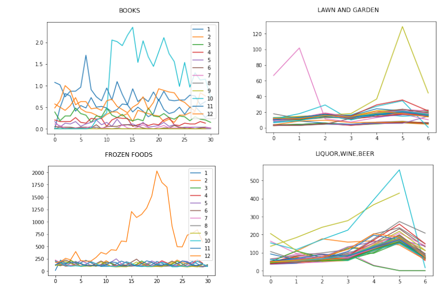
Figure 2: Day of month and day of week trends

As seen in Figure 2, we plotted the average sales for each family over the weeks and months of the year. This allowed us to observe if the day of the week or the day of the month was visually correlated to the sales. For example, one can observe that for the `FROZEN FOODS` family, sales peak during the month of December just before Christmas. Also, the sales of `LIQUOR,WINE,BEER` tend to increase during the weekends.

5 Extracting more features after data analyses

5.1 Time and frequency trends

We concluded from the analyses in the previous section that we had to extract more features that could capture seasonal trends. We did so by adding the following features described below (some of which are eponymous and thus the explanation is skipped) – `year`, `quarter`, `day`, `weekoftheyear`, `dayofweek`, `isweekend`, `newyear`, `startschool`: identified dates for when school typically starts; `wageday`: identified the dates for which wages are paid: the 15th and end-of-the month; `earthquake_Aprin2016_effect`: captured dates affected by the earthquake in 2016 ; `onpromotion_week_avg`, `onpromotion_biweek_avg`, `onpromotion_1_month_avg`, `onpromotion_2_month_avg`: captured dates on which some product families at certain stores were on promotion; `fourierA`, `fourierM`, `fourierW`: fourier transforms to capture seasonal data in the annual (`freq='A'`), monthly (`freq='M'`) and weekly (`freq='W'`) frequency domains.

5.2 Holiday corrections and lags

On further analysis, and from inputs obtained from publicly available notebooks [5], [6], [7], it was evident that adding a feature to account for stores closing or opening during the training period was necessary. Additionally, we realized that features must be included to identify public holidays as noted in `holidays.csv`. These holidays had a large impact on the sales for many product families.

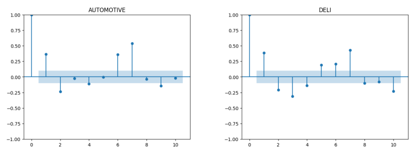

Figure 3: PACF plots for two families

Our research also led us to believe that lag features may be important to include. For example, with `dcoilwtico`, oil prices from the day, month, or even two months before may serve as predictors for sales and the current economy of Ecuador. Important lag features can be identified through Partial Autocorrelation Function (PACF) plots [3]. Through these plots, as seen in Figure 3, we found that the current sales of an item were strongly correlated to sales values from 7 days, 14 days, a month, and a year before. However, this was not surprising since we had already identified these seasonal patterns. However, we never managed to truly utilize these features meaningfully and obtain a large score improvement. We suspect that more preprocessing might be required to resolve this.

6 Final model, results, shortcomings, and alternate attempts

6.1 Final model and results

From Sections 2 and 3, we observed that though the linear regression models worked well as the base model, given the number of features and the precision with which they were defined, we believed that tree-based models were the way to proceed. In the implementation of our final model, we worked with `ExtraTreesRegressor`, a class implementing a meta-estimator that fits a number of randomized decision trees (Extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [8]. To induce further generalization, we wrapped this base learner `ExtraTreesRegressor` into `BaggingRegressor`, which is an ensemble meta-estimator that fits base learners on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form a final prediction [9]. We also carried forward our initial features mentioned in Sections 2 and 3 into our model. Our final model obtained a submission score of 0.39884.

6.2 Shortcomings

On checking the training phase RMSLE scores for various families, we found that some families, `LINGERIE`; `LIQUOR,WINE,BEER`; `SCHOOL AND OFFICE SUPPLIES` in particular, were scoring higher than the rest, negatively affecting our final submission score. This is most likely due to these families exhibiting a unique trend or behaving more randomly than the other families. We tried to solve this issue by using other base learners for these specific families, but since each run of our model took anywhere between 4 to 26 hours depending on the input hyper-parameters, (`extra_trees_n_estimators` and `bagging_n_estimators`), we were severely bottle-necked by the training time.

6.3 Alternate attempts

6.3.1 Using best sales as a feature

We used the best value of sales predicted so far as a feature in our model and retrained it with lower values of the aforementioned two hyper-parameters, so that our runtime came under 2 hours. However, a side-effect was that this approach biased the new model to latch on to the predicted sales of the old model, thereby preventing it from learning better model parameters in general.

6.3.2 Re-scaling the predictions

Also, we observed that our model was routinely over-predicting. To combat this, we down-scaled our predictions by 1% to slightly improve our score on the validation set. We also tried to perform a dynamic scaling of the predictions on the test set, based on the ratio of the actual to predicted sales on the validation set. However, this was not successful.

6.3.3 Residuals, Hybrid Models and Linear Trees

As ensemble methods are very popular, we tried averaging the best of our results and submitted the averaged predictions, but the score obtained was not an improvement over the individual results. Therefore, we decided to try a few hybrid models: models where one learns the overall trends and the other attempts to learn the residual errors [3]. For example, we could train a refined linear regression model to capture linear trends and seasonal information. Then, a decision tree can be used to learn the error or information that was missed by the first. However, we never managed to get our hybrid models to perform sufficiently. Perhaps using another function class or training the secondary model with a different subset of the data would have helped. We further tried using linear trees - decision trees with linear regression functions acting on the leaves. However, all of these required extended hyper-parameter tuning, which demanded more computing power (CPU and RAM) and time - so we could not pursue them fully.

6.3.4 Feature Enrichment

Further, we tried enriching some of the features using Upgini: a popular framework that searches public databases for relevant information to be included in your model [10]. The enriched features were then added to the families that were under-performing with a training error over > 0.2. However, this strategy did not turn out to be useful since the features provided by Upgini overlapped with up to 98% of the original data.

6.3.5 Darts, global models and neural networks

We explored some approaches using the Darts [11] library for exploring time series, but could not completely understand the resources, as they employed backtesting and a complex ensemble of models, to effectively forecast for the competition. We also learned that global models using subsets of the 1782 (= 54 * 33) time series could lead to better performance. But there was a trade-off between the number of models and the number of time series features used. We also briefly explored neural network-based approaches - RNNs, LSTMs and Transformers. But the need for hyper-parameter tuning, larger datasets, compute requirements and training time did not outweigh decision tree-based approaches. Also, decision trees were used by many top performers globally as seen in the discussion forums [12].

6.3.6 SARIMAX

Apart from these, we tried implementing the SARIMAX [13] model. SARIMAX is an extension of the ARIMA model but allows for trends and seasonal patterns to be learned [3]. It showed some promise, however, we did not include this model in our final submission notebook as we could not perform an exhaustive parameter search by the deadline. Please see `sarimax.ipynb` in the submitted code for reference.

7 Running the code

You can run the final notebook submitted for this project, `final_submission_id1.ipynb` (Score: 0.39884), either on Kaggle or your local system directly. In case of the latter, please ensure that you have the relevant basic Python libraries installed. Also, kindly note that this notebook may take about 6 hours to run. We have also included the other notebooks we have written at various project stages, for your reference: `CSCI567_id1.ipynb` (Score: 0.45807), `lightgbm-mergedfeaturedf.ipynb` (Score: 0.91712), and `sarimax.ipynb` (Verification Score: 0.88). These too could be run either on Kaggle or your local system.

8 Conclusion

We performed data analysis and explored various models in a systematic way in our approach to predict grocery sales in the "Store Sales - Time Series Forecasting" competition hosted on Kaggle. We also understood the importance of feature engineering, pre-processing, logging metrics and intermediate results, and setting up a training pipeline. Planning and estimating development time, use of modular and reusable code, and efficient collaboration were pivotal to this project.

9 Acknowledgements

We thank Prof. Vatsal Sharan, TA staff of CSCI 567, other teams, and the Kaggle community – who helped us gain a holistic view of the various techniques and models implemented herein this project and understand the concepts with clarity.

References

[1] Alexis Cook et al. Store sales - time series forecasting, 2021. URL https://kaggle.com/competitions/store-sales-time-series-forecasting.

[2] Understanding the metric: RMSLE — kaggle.com. https://www.kaggle.com/code/carlolepelaars/understanding-the-metric-rmsle/notebook, . [Accessed 14-Dec-2022].

[3] Robert Hyndman and George Athanasopoulos. Forecasting: Principals and Practice. https://otexts.com/fpp2/, 2018. [Accessed 10-Dec-2022].

[4] Sklearn.linear_model.multitasklassocv. URL https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.MultiTaskLassoCV.html.

[5] Store Sales TS Forecasting - A Comprehensive Guide — kaggle.com. https://www.kaggle.com/code/ekrembayar/store-sales-ts-forecasting-a-comprehensive-guide/notebook#References, . [Accessed 14-Dec-2022].

[6] Per Family with Store Multioutput — kaggle.com. https://www.kaggle.com/code/mscgeorges/per-family-with-store-multioutput, . [Accessed 14-Dec-2022].

[7] Store Sales: Using the average of the last 16 days — kaggle.com. https://www.kaggle.com/code/carlmcbrideellis/store-sales-using-the-average-of-the-last-16-days, . [Accessed 14-Dec-2022].

[8] sklearn.ensemble.ExtraTreesRegressor — scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html, . [Accessed 14-Dec-2022].

[9] sklearn.ensemble.BaggingRegressor — scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html, . [Accessed 14-Dec-2022].

[10] GitHub - upgini/upgini: Free automated data enrichment library for machine learning $\rightarrow$ searches through thousands of ready-to-use features from public and community shared data sources — github.com. https://github.com/upgini/upgini. [Accessed 14-Dec-2022].

[11] Time Series Made Easy in Python; darts documentation — unit8co.github.io. https://unit8co.github.io/darts/. [Accessed 14-Dec-2022].

[12] What algorithms are most successful on Kaggle? — kaggle.com. https://www.kaggle.com/code/antgoldbloom/what-algorithms-are-most-successful-on-kaggle, . [Accessed 14-Dec-2022].

[13] statsmodels.tsa.statespace.sarimax.SARIMAX; statsmodels — statsmodels.org. https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html. [Accessed 14-Dec-2022].