CSCI 567 Final Project

Furong Jia
Department of Computer Science
University of Southern California
florajia@usc.edu

Jingmin Wei
Department of Computer Science
University of Southern California
jingminw@usc.edu

Qilin Ye
Department of Computer Science
University of Southern California
yeqilin@usc.edu

Zheyi Zhu
Department of Computer Science
University of Southern California
zheyizhu@usc.edu

Abstract

For this Time Series Forecasting project on Store Sales, we first extracted and created features based on the given data over oil price, holidays, store, transactions, onpromotion, and so on. After trying a set of models, we were able to achieve the highest score using a stack of two separate models. The first model contains 1728 individual models, one for each family and each store. It consists of two streams, one for "school and office supplies" predictions using ExtraTreesBoosting and RandomForestBagging, and another for the predictions of the remaining categories using ridge regression and support vector regression. The second is a LightGBM model which contains 33 models, each model for one family of goods. Eventually, we achieved a best score of 0.37996 on the test data by stacking the results from the two models.

1 Feature Engineering

In this section, we conducted an analysis of the given data to select, combine, and create features relevant to sales prediction. Since our two models work on different subsets of data, they use two separate feature lists containing the respectively derived features. It is worth mentioning that we transform the sales by $\log(1+x)$ due to large data value and accuracy for x close to 0.

1.1 Transactions

Based on Figure 1,a Pearson Correlation of 0.8374 between total sales and transactions indicates a strong positive correlation. Therefore, even though there is no transaction data for the time period of the test set, we can still use the transaction data to engineer time-related periodic features that are correlated with sales.

Through data visualization of the transactions over time using monthly and daily averages, we discover similar patterns over certain months and days. For example, Figure 3 demonstrates that transactions reach the peak during the end of the year; Figure 4 indicates that transactions are prone to be high on Saturdays and people are least willing to shop on Thursdays. Therefore, we create features representing the months and days, which turns out to be beneficial for our prediction of sales. Moreover, since the data displays strong periodic features, CalendarFourier, a tool for time series

Preprint. Under review.

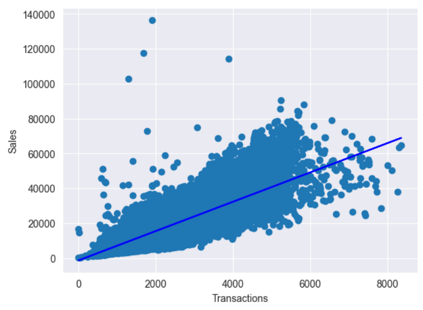


Figure 1: Regression on Transactions and sales

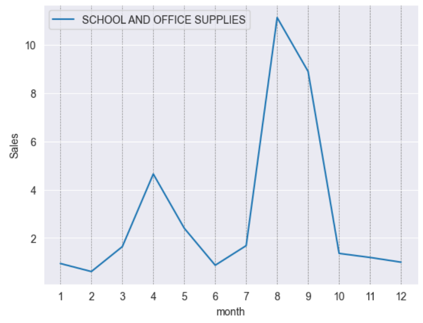


Figure 2: Monthly Patterns of School Supplies

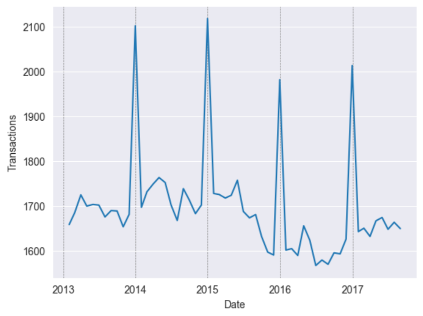


Figure 3: Monthly patterns over transactions

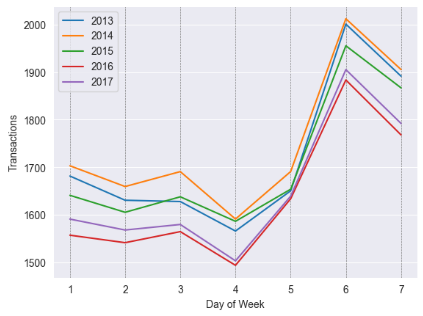


Figure 4: Daily patterns over transactions

forecasting, is applied to our model for creating a repetitive trend that can be applied along with other features.

During analysis of the training data, it also discovered that certain family of goods can be largely affected by time features, one example being “SCHOOL AND OFFICE SUPPLIES.” As shown in Figure 2, the number of school supply sales significantly surge in late summer. Correspondingly, we add an additional feature indicating the school seasons. It is revealed in the later that our first model also uses a separate stream for prediction over the "SCHOOL AND OFFICE SUPPLIES" category.

Additionally, as mentioned by the hint, wage day can have some impact on sales. Therefore, we also include the wage day inside our feature selection.

1.2 Oils

Oil prices are intimately tied to shopping incentives, further leading to changes in sales. As demonstrated in Homework 4, the use of moving average of the oil prices of a certain time period can help to get a stable indication and therefore be beneficial for sales prediction. We further include weekly averages, biweekly averages, and monthly averages to gather more complete features.

On the other hand, as an important aspect of time series forecasting, lag features are included as well. The partial Autocorrelation graph (Figure 5) shows that only lags of the first few days have significant correlations. Therefore, we apply the first three lags to retrieve the useful lag features from previous periods.

1.3 Holidays and Stores

According to the provided holiday data, we combined regional features as well are types to modify the previous workday feature. We simply determine if a given date is a holiday based on the date and its relevant geographical information.

As mentioned in 1.1, certain time periods lead to significant sales changes, so we extract them as single features. One such example is Christmas. Another is the 2016 Ecuador earthquake.

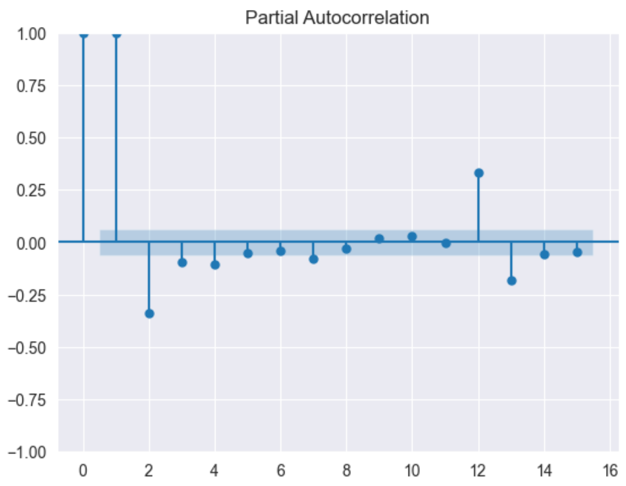


Figure 5: oil price lag pacf

1.4 On Promotion

We have information on promotions from the given data. To leverage this information, we include weekly moving averages and monthly averages in our feature list.

1.5 Pre-prediction as New Features

Some product families are highly correlated with other product families. For example, people who buy seafood are more likely to buy dairy as well. To capture this correlation and leverage it, we trained a linear model on all training data regardless of their families or stores, and we then used the prediction as new features to make our final sales prediction.

1.6 Zero Prediction

Zero predictions are used to assign a prediction of 0 if the previous data continues to gain a result of 0. This is understood as some stores may stop selling products after some time or never sell some categories of products. Here, we choose a period of 15 from 2017-08-01 to 2017-08-15.

1.7 Difference of Features for the Two Models

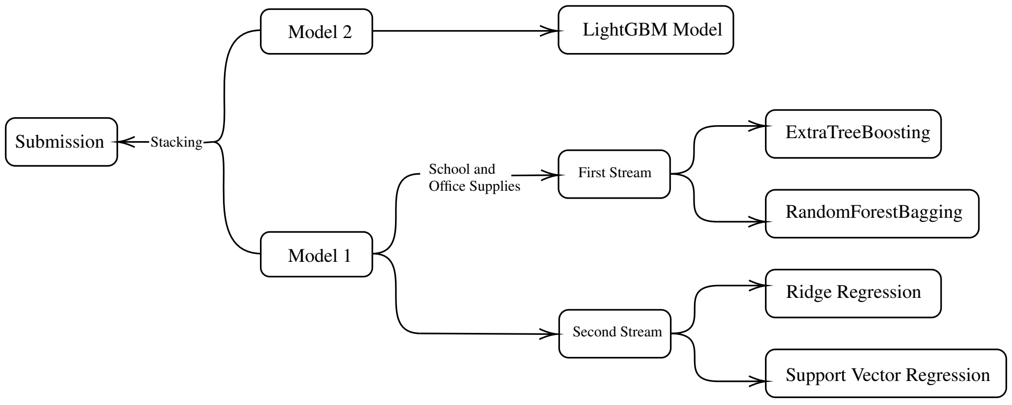
When experimenting with our first model, we did not see significant improvement when using a long period of training data, so we chose a shorter period from 2017-04-01 to 2017-08-15 in our training data to reduce computing and memory consumption. Due to this change, some features, like earthquakes, are eliminated from the feature list.

Since Darts is a more suitable tool for time series forecasting by allowing us to leverage past covariates and future covariates, we basically deal with similar features and stack them to either past covariates or future covariates. When dealing with the feature preprocessing for the LightGBM model, some further details (e.g. earthquakes, store locations combined with regional information) are also included. This provides the model with more features to depend on. Holiday information, promotion information by family, and oil information are taken in as future covariates, while transactions are used as past covariates since there is no relevant transaction information for the time period of the test set.

Table 1: Models and results

Algorithm	Result
Linear Regression	0.592
Ridge Regression	0.557
Support Vector Regression	0.531
Gaussian Process Regression	1.0+
Multi-layer perceptron	1.0+
XGBoost	0.423
LightGBM	0.410
Random Forest	0.397
Ridge + SVR (RS)	0.414
RS + ExtraTreesBagging + RFBagging	0.395
RS + XGBoost + RFBagging	0.396
RS + LGBM + RFBagging	0.443
Darts + LightGBM	0.384
Ridge+SVR+ETbag+RFbag+Darts(LightGBM)	0.380

2 Models and Results



2.1 Model 1: Ensemble method

In this section, we will go through all the models we syyrmpyrf, including single algorithm models, ensemble learning, combined pipeline, Darts, and stacking ensemble method. The result for different models is shown in Table 1. The best score we achieved is 0.37996.

2.1.1 Single Model

On our first few tries, we set different relative weights for each sample, based on the exponential date gap between every training datapoint and end date, which can reflect the time series relatively.

For each product family, we've explored different single machine learning models to train provided by the scikit-learn toolkit, including linear regression, kernel ridge regression, support vector regression(SVR), simple multi-layer perceptron regression(neural network), gaussian process regression(GPR), etc. All of them have different performances, but none of their accuracies is good enough.

Linear regression has a 0.592 RMSLE error.

In ridge regression, we use regularization with α = 0.80 and the Dot-Product kernel, where $k(x, x') = \sigma^2 + x \cdot x'$ and σ controls the inhomogeneity of the kernel. and the model's error reaches 0.557.

In support vector regression, we use RBF kernel with l_2 penalty ($C = 0.2$), where kernel

$$k(x, x') = \exp\Big(-\frac{||x - x'||_2^2}{2\sigma^2}\Big)$$

and the error is 0.531.

In gaussian process regression, we still use RBF kernel to train the model, and it has more than 1.0 error on the test set.

In a simple neural network, we set 3 layers with 128-256-128, adam optimizer, and 0.0001 regularization. It has more than 1.0 error on the test set, too. GPR and MLP cannot take relative weight into account, thus they both have bad performance.

All these single-algorithms are not robust enough to make some precise predictions.

2.1.2 Ensemble Learning

The imprecise predictions of the model mentioned above lead us to introduce some ensemble learning algorithms to fit the data, including XGBoost, LightGBM, and Random Forest. In this part, we've also utilized a parameter search strategy to fine-tune the model and get better results.

First, we use boosting algorithm and tried XGBoost with default settings (n_estimators=100). The error is around 0.434. The 10% improvement proves our optimizing direction is correct. Then we use the grid search method to fine-tune the parameter and get a better score(0.423). We also tried LightGBM with default settings and it reaches a score of 0.410.

We also use the bagging algorithm and introduced Random Forest (RF) model with the default settings (`n_estimators=100, min_samples_split=2, min_samples_leaf=1, max_depth=None, max_leaf_nodes=None`). The error is around 0.434. The grid search method is also used to find the relative optimal parameter. With the parameter setting (`n_estimators=160, min_samples_split=5, min_samples_leaf=9, max_depth=10000, max_leaf_nodes=4700`), the error reaches 0.397.

2.1.3 Combined Pipeline

We still want to optimize our model and make some progress on the performance. Thus, we combine some models used before and make some pipelines to construct a new structure.

We constructed a new structure. It has 2 streams, the first stream is for 'SCHOOL AND OFFICE SUPPLIES' predictions, and the other is for the remaining features. Each stream has some pipelines. All these pipelines are combined by average voting regression as the output of each stream.

We've also used a fixed seed to generate reproducible results in the model since we noticed that without a fixed seed, the model has fluctuating results.

For the first stream, since the category 'SCHOOL AND OFFICE SUPPLIES' has low correlations with other families and its sales increase significantly in the school season, we use decision tree-based methods to predict it. We've tried different ensemble learning combinations, such as LGBM + RFBagging, ExtraTreesBagging+RFBagging, and XGBoost+RFBagging. All methods use similar parameter settings, and each combination has a different performance. The best is the ExtraTrees-Bagging+RFBagging with 0.395 scores, while the XGBoost+RFBagging is 0.396 and the LGBM + RFBagging is 0.443.

For the second stream, we used the combination of ridge regression and support vector regression. The ridge regression is with normalization and α = 0.80 regularization, the same as we set before. The SVR is realized by RBF kernel with l_2 penalty ($C = 0.2$), also the same as we set before.

2.2 Model 2: Darts Model

We'd like to further improve the model's performance. In the previous step, the only thing we focus on "time series" is to set different relative weights for each sample based on the date gap. In this case, we may lose some chronological information. Thus, we utilize Darts Python Library to assist with time series modeling. Darts library can distinguish past and future covariates. Past covariates are time series whose past values are known at the prediction time, while Future covariates are time series whose future values are known at the prediction time. Different time series defined by Darts for forecasting make the prediction more precise.

During the feature engineering step, we use TimesSeries Class to create a bunch of time series from some preprocessed dataframes. Most feature extractions are similar, despite that there is some difference existing in handling the codes due to the different nature of the two models.

Then we stack all the processed time series together and use some ML algorithms to fit the time series data. Same as we've done before, we tried various ML models, and found out that the gradient boosting method - LightGBM has the best performance. We specify the model's lagging value for time series prediction. We set lags value = 64 (lagged target values used to predict the next time step); lags_future_covariates = (14, 1) (number of lagged future_covariates values used to predict the next time step); lag_past_covariates = $-16 \sim -31$ (number of lagged past_covariates values used to predict the next time step). And the model reaches 0.384 error, which means our optimization direction is correct.

2.3 Stacking Ensemble Learning

At last, we would like to combine the result of the combined pipeline and Darts time series prediction. Intuitively, we would like to use the stacking ensemble learning method to fit the result. Stacking is often used to train different learners to learn the same data, then use another model or method to stack results from those previous learners together. These methods include arithmetic operations like $\min$ or $\max$, as well as weak ML models such logistic, SVM, and decision trees. In our model, we simply take the mean to stack the separated result from the combined pipeline and Dart. The final score we obtained is around 0.380.

2.4 Some Other Thoughts

Some models we tried had better validation errors but still got higher errors in the Kaggle competition test set. It is possible because hyperparameters are tuned for the validation set.

There are some other improvements we can try in the future, such as adding some more features(whether the store is a new store), finding the outliers in the data and removing them before training.

We tried autoML(Autogluon) before we experimented with models on our own. It automatically ensembles and stacks Catboost, XGboost, LightGBM, ExtraTrees, Random Forest, Neural Network, K-Nearest Neighbor. The best score it achieved is 0.401. The reason why its score is not as good as the models we tried is still a mystery. A probable reason we can think of is overfitting. It's a good question for further exploring.

3 How to run the code

Since we use Darts, the kernel might crush on your computer, so use Google Colab to run **csci567_id3.ipynb**. It takes about 30 mins to finish. The result is in submission.csv.

References

We got some insights from the following Kaggle notebooks.

`https://www.kaggle.com/code/ekrembayar/store-sales-ts-forecasting-a-comprehensive-guide/notebook#4.-Oil-Price`

`https://www.kaggle.com/code/romaupgini/guide-external-data-features-for-multivariatets`

`https://www.kaggle.com/code/ferdinandberr/darts-forecasting-deep-learning-global-models`

`https://www.kaggle.com/code/markbquant/stacking-upgini-darts`