Kaggle | Corporación Favorita Grocery Sales Forecasting

CZ4041 Machine Learning

Eugene Yeo	U1621627G
Xie Zesheng	U1621162D
Tu Anqi	U1622399F
Wu Qinxin	U1522835L
Yang Can	U1520582J

Content

Executive Summary

This report proposes a machine learning model for sales forecasting and evaluation of the model performance for Corporacion Favorita. Data size, extra features and are the main challenges faced. Due to the large data size, and limited computation power the team has, only data from year 2017 is sampled. One model will be built for each day of the week together with different combination of features. Four models has been selected, including linear regression, decision tree, lightgbm and neural network.

Models are trained with 6 different feature settings and their performance calculated on private leaderboard. It is observed that model trained with only lag features perform the best for both private and public Leaderboard and the weighted score. To find the best model setting, experiments are conducted on the same model with different model settings, but with the same data setting which exclude all supplementary data.

For all models, selected hyperparameters are tuned by cross validation. The final tuned model is then trained on the combined data of both train and validation set. LightGBM performs the best among all the model, ranks us at 71st out of 1675 on the private leaderboard. Its NWRMSLE scores for the private leaderboard are also consistently slightly higher than the ones of public leaderboard, and much lower compared to the ones of validation set.

To further improve the overall performance of our model, stacking is implemented with different types of models which are capable to learn some part of the problem, but not the whole space of the problem, and then use each base-level model to build an intermediate prediction. With stacking, the ranking is improved to the 39th.

However, due to the large data size and limited computational power the team has, only sales records from 2017 onwards are used. Performance of the models could be further improved if given higher computation power and time to train more models on more data.

1. Introduction

1.1 Business Background

Machine learning forecasting is attracting an essential role in several significant data initiatives today. For the grocery stores the machine learning forecasting can be applied to aid improve customer engagement, produce demand forecasts, improve productivity and expand new sales channels etc.

Corporación Favorita is a large Ecuadorian-based grocery retailer which operates hundreds of supermarkets, with over 200,000 different products on their shelves. They currently rely on subjective forecasting methods with very little data to back them up and very little automation to execute plans. Realizing the importance of machine learning forecasting, Corporación Favorita has challenged the Kaggle community to build a model that more accurately forecasts product sales to see how machine learning could help with the demand and stock forecasts.

1.2 Performance Measurement

Supervised learning vs Unsupervised, target variable

Within the field machine learning, there are two main types of tasks: supervised and unsupervised. The main difference between the two learning method is that supervised learning is based on the prior knowledge of what the output should be for the sample input. In this project, the supervised learning is implemented as the goal is to predict the product sales base on several feature inputs.

Time series continuous

Time series forecasting is an important area of machine learning involving time component. The time forecasting involves taking models fit on historical data and using them to predict future observations.

NWRMSLE

The submission on Kaggle is evaluated on the Normalized Weighted Root Mean Squared Logarithmic Error (NWRMSLE), calculated as follows:

$$NWRMSLE = \sqrt{\frac{\sum_{i=1}^{n} w_i (\ln(\hat{y}_i + 1) - \ln(y_i + 1))^2}{\sum_{i=1}^{n} w_i}}$$

where for row i, $\hat{y}_i$ is the predicted unit_sales of an item and y_i is the actual unit_sales; `n` is the total number of rows in the test set.

The weights,ω_i, can be found in the `items.csv` file. Perishable items are given a weight of `1.25` where all other items are given a weight of `1.00`.

This metric is suitable when predicting values across a large range of orders of magnitudes. It avoids penalizing large differences in prediction when both the predicted and the true number are large: predicting 5 when the true value is 50 is penalized more than predicting 500 when the true value is 545

1.3 Challenges

Data Size
The training data contains 12,549,704 observations, including dates, store and item information, whether that item was being promoted, as well as the unit sales from 2013 to 2017. As a result, only the data from 2017 is selected for further experiments and it still takes a long time to train each model.

Extra Features
There are additional files providing supplementary information, including the city, state, type, and cluster of stores, family, class, and perishable of items, transaction counts, daily oil price of Ecuador (Ecuador is an oil-dependent country and it's economical health is highly vulnerable to shocks in oil prices) and holiday event. In this project, the effects of including or excluding these data and how to use them is explored.

Time series
To use time series forecasting, it needs more time to reframe the data and engineer useful features and capture the time pattern such as weekday trend.

1.4 Team Structure

Name	**Role**
Tu Anqi	Team Leader
Eugene Yeo	Team Member
Xie Zesheng	Team Member
Wu Qinxin	Team Member
Yang Can	Team Member

2. Proposed Solution

2.1 Tools

Several open source Python libraries are used in this project. **Pandas**, a Python library for data manipulation and analysis, is used to preprocess the data and engineer features. **Scikit Learn**, a Python machine learning library, provides solid implementations of a range of machine learning algorithms and several useful helper functions. **LightGBM** is a Python library implementing the gradient boosting framework that uses tree based learning algorithms. **TensorFlow** is a free and open-source software library for dataflow and differentiable programming across a range of tasks, and is used for implementing neural networks. **Hypopt** is a Python package for grid searching optimal hyper-parameters using a validation set.

2.2 Data Preprocessing

Due to the large size of the training data and limited computation power the team has, it is decided to only sample data in 2017 from the training date to extract features and construct samples for building the prediction model. The sampled data is then preprocessed by data cleaning and feature engineering which will be discussed in details later. As time series forecasting problems must be reframed as supervised learning problems before machine learning can be used, the most important part of the feature engineering is to get the lag features of observed variables - use some methods to calculate the statistics of some targets for different keys in different time windows. Supplementary data provided such as holidays and oil price also have to be processed to extract useful features for the training set.

It is required to predict the unit sales of each item in each store in the test set from 2017/08/16 (Wednesday) to 2017/08/31 (Thursday). To reflect the same distribution of weekdays and same relationship between training dataset and test dataset, data from 2017/07/26 (Wednesday) to 2017/08/10 (Thursday) is extracted as the validation set used for tuning hyperparameters, which will be discussed further later. 7 sets of the same pattern of 16 days from Wednesday to Thursday before the validation set time range are extracted to form the training dataset, with the first set from 2017/05/24 (Wednesday) to 2017/06/08 (Thursday) and the last set form 2017-07-05 (Wednesday) to 2017/07/20 (Thursday).

2.3 Modelling Methodology

Benchmark
To set a benchmark for the model performance, forecasts are made with the persistence model, which uses the observation from the prior time step (t-1) to predict the observation at the current time step (t). The first benchmark uses the sales of most recent day (2017/08/15) for each combination of store and item to predict for all the 16 days, getting a weighted score of 0.74022. Another benchmark uses the sales of the most recent same weekday for each combination of store and item. For example, the sales of 2017/08/09 is used to predict the sales for 2017/08/16, 2017/08/23 and 2017/08/30. This model gets a better weighted score of

0.72954, probably due to that it captures the weekday trend. With machine learning, it is expected to achieve a lower NWRMSLE than the benchmark forecasts.

One Model Per Day
While it is possible to fit a model for each item in each store, there might be problems due to having only a few hundred data points for each model, which contains insufficient information. As machine learning models tend to improve with more data, it is decided to concatenate the records for all items and stores and train a single model instead of training a model for each combination of store and item. Since 16 days of sales are to be predicted, one model will be trained for each day. All 16 models will have same independent variables, but different target variable.

Feature Selection
It is uncertain whether the addition of those supplementary data will improve the model performance. Thus, different combination of features (inclusion or exclusion of different source of supplementary data) can be experimented with the same model which must also be trained with the same settings. The best combination of features will then be selected through the evaluation and comparison of those models' performances.

Algorithm Selection
The comparable performance of models varies with different machine learning problem. In other words, the best algorithm for each machine learning problem is different. Bearing this in mind, several different type of algorithms are selected, from the easiest linear regression and decision tree to the most complicated neural network, in order to study the performance of different algorithm in solving this forecasting problem. Besides, in theory, the ensemble methods, such as boosting, can combine a group of "weak learners" to form a "strong learner", thus resulting in better performance. Therefore, a tree-based gradient boosting algorithm implemented by LightGBM is also selected, to study the effectiveness of ensembling.

Model Fine-tuning
Hyperparameter are parameter values to be set before the learning process begins and affect the model's performance largely. Different model training algorithms require different hyperparameters. The optimized value of hyperparameters also vary with different machine learning problem. The method of cross validation can be used to find the optimal combination of hyperparameters. For each specified combination of hyperparameter, the model will be trained on the train set and then validated on the validation set. The hyperparameter combination that leads to best validation score will then be used for the final model. This process can also be described as fine-tune the model. For each algorithm, the fine-tuned model will then be trained on the combination of train set and validation set to predict for the test set.

Model Stacking
To apply another ensemble method, stacking, a meta-level model can be trained to combine the outputs of all those first-level models as mentioned above.

Reproducibility
All models are trained with the seed of 2019 for reproducibility.

3. Data Preprocessing

3.1 Data Cleaning

As the dataset clearly reflects the real situations in the grocery industry, it is also clearly subjected to a lot of real world noises, including incorrectly stated values and missing values. Thus, data cleaning is performed before the modelling.

Missing Zero Sales
It is stated by the data provider that the training data does not include rows for items that had zero unit_sales for a store and date combination. Zero sales are definitely useful and important information for building the prediction model and should be included. A item should be considered to have zero sale in a store on a certain date, if and only if it was available in that store on that day but no one bought it. However, as there is no information as to whether or not the item was in stock for the store on the date, it is assumed that an item was in stock in the item for all dates as long as it had been sold in that store. Thus, for each existing store and item combination in the dataset, zero sale records are added to dates that do not have a sale record of such store and item combination.

Unseen Items
Another problem explicitly indicated by the data provider is that there are a few items seen in the training data that are not seen in the test data and a small number of items in the test data not contained in the training data. More specifically, the train set and test set have 3841 common items, but the train set contains 177 items not unseen in the test set while the test set contains 60 items unseen in the train set. This means the model trained on the training data will predict the sales for some items that do not need to be predicted, and the model will not be able to predict the sales for some items that need to be be predicted. Due to its small portion, it is decided to just fill 0 for the sales of those items appeared in the test set but unseen in the train set. Items seen in the train set but not in the test set are still kept as they are still useful information when training the prediction model.

Log1p transformed, performance measure
Because of the NWRMSLE evaluation metric, customized scorer function must be used for the optimization algorithm of models. Another technique is to clip all unit sales to be non negative, then transform them with log1p, so that the built-in objective (loss) function RMSE which is available from most algorithms will be the same as the the evaluation metric, despite the weighted RMSE is still required to be calculated afterwards. Log1p is used instead of log because the log transform cannot be applied to 0 unit sales and log1p is also able to match the evaluation metrics. The predicted result of the test set will be reversely transformed by expm1 (exp(x) - 1 of x), before submission.

Missing OnPromotion

There is a small proportion of missing values for the promotion information. Since the majority of the records have no promotion, it is assumed that those records with missing promotion information all also have no promotion. Thus, they are filled with False which is the majority value of the OnPromotion variable. This category is then converted to number, with True as 1 and False as 0.

Missing Oil Price
The oil price has several missing values and also not available for weekends. Thus, dates of weekends are added first, then all missing oil price are filled with the average of the oil prices from previous and next days.

3.2 Feature Engineering

The essence of this time series prediction problem is the engineering of useful lag features. Supplementary information such as holidays and oil price also need to be processed to create extra features for the training set.

Sales
The mean of the sales for different previous time periods (1, 2, 3, 7, 14, 30, 60, 120 days) before the days to be predicted are calculated. More mean sales features are also calculated with different ways of splitting those time periods, such as with or without promotion, and by each weekdays. Days to first and last sale are also computed.

Promotion (past and future)
Count of promotion days for different previous time periods are calculated. Days to first and last promotion are computed. The promotion values for the 16 days to be predicted are also included.

Oil Price
As Ecuador is an oil-dependent country, oil price is expected to affect its economic confidence and consumers' purchasing power significantly. Thus, the daily, weekly and monthly oil price differences for each of the 16 days are calculated.

Mean Encoding
For categorical columns of item and store information, mean encoding is used, instead of label encoding which is more suitable for ordinal variables and one hot encoding which will increase the number of column drastically due to the large counts of items and stores. The leave-one-out (LOO) scheme is used here, by encoding each categorical variable value with the mean of those target variable (unit_sale) belonging to that value. For the test set, the mean of target variable per value for all records in train set is used. This scheme minimizes data leakage by preventing the target variable (unit_sale) to be predicted from being calculated into the encoded feature. Apart from single categorical variables such as item number, item family, store number and store type, the interaction of those item and store variables are also considered, by processing the mean encoding for each combination, such as item number/store number, item number/store city, item family/store number and so on. To select the most meaningful combinations, the

correlation between each feature and the target (unit_sale) is calculated and only combinations that gives a correlation higher that 0.25 are preserved. After calculation, the mean encoded features of item_nbr, family, class and store_nbr are preserved. The detailed correlations for each single categorical variable and each categorical variable combination is documented in Appendix 1.

Holiday
Holidays can be local (for a city), regional (for all cities in a state) and national (for all cities in the country). Thus, the city and state information of each store is combined with the holiday information to process the holiday for each store. Then, for each record, whether it was in a holiday and the days to previous holiday and next holiday are calculated according to the store number. There are also categorical features for each holiday such as area range and the description (which has 103 unique values). To eliminate noisy and useless information, only those appeared in the duration of testset are preserved while others all converted to ‘others’.

Payday
Since wages in the public sector are paid every two weeks on the 15th and last day of the month, it is assumed that the grocery sales could be affected by this. Thus, for each record, both the days to previous payday and to next pay day are calculated according to the date.

3.3 Normalization

Normalization is performance for all continuous variables with the StandardScaler helper object from sklearn library, which transform each feature such that its distribution will have a mean value 0 and standard deviation of 1. Standardization of a dataset is a common requirement for many machine learning models which might behave badly if the individual features do not look like standard normally distributed data. It also makes all variables have common scale.

3.4 Label Encoding

The sklearn and tensorflow libraries requires all the data to be numeric for the model training. Thus, label encoding is performed on the categorical data represented in string format, including 'family', 'city', 'state' and 'type', with the LabelEncoder helper object from sklearn library, which to encode the columns with numeric value between 0 and n_classes-1.

4. Feature Selection

With the lag features remaining constant, the feature settings varies by the inclusion or exclusion of features related to mean encoding, holiday, oil and payday. The results of 6 linear regression models trained with 6 different feature settings are shown below, ranked in descending order of the performance on Private Leaderboard. The weighted score is calculated as the root of the weighted average of the square of private score (69%) and the square of public score (31%). As mentioned in Kaggle - this public leaderboard is calculated with approximately 31% of the test data while the private one is based on the other 69%.

Surprisingly, the model trained with only lag features perform the best for both private and public Leaderboard and the weighted score. This trend is assumed to be the same for all other models. Thus, it is decided to exclude all the supplementary features.

lag_ feature	**mean_ encoding**	**holiday**	**oil**	**pay_day**	**Validation Score**	**Private Score**	**Public Score**	**Weighted Score**
TRUE	FALSE	FALSE	FALSE	FALSE	0.61161	**0.53281**	**0.52072**	**0.52909**
TRUE	FALSE	TRUE	FALSE	FALSE	0.61286	0.53273	0.52126	0.5292
TRUE	TRUE	FALSE	FALSE	FALSE	0.61141	0.53297	0.52077	0.52922
TRUE	TRUE	TRUE	TRUE	TRUE	0.61496	0.53351	0.52099	0.52966
TRUE	FALSE	FALSE	TRUE	FALSE	0.61552	0.53373	0.52104	0.52983
TRUE	FALSE	FALSE	FALSE	TRUE	**0.61064**	0.5327	0.52785	0.5312

However, it is also observed that the score for the validation dataset is noticeably higher than that of the testset. Besides, the ranking of the performance of the models with different feature settings also differs from that of the testset. The score for the validation dataset is the best when there the features related to payday are added to the lag features, followed by the one trained with lag featured and mean encoding features. The performance of the model trained with no supplementary features ranks the third. This might due to that the sales during the selected days (2017/07/26 to 2017/08/10) used for our validation dataset behaves quite different from the sales during those days (2017/08/16 to 2017/08/31) of the test set.

This reveals a challenge of most machine learning problem - finding the best validation set that could best represent the test set, thus hopefully reflects the final performance. The rules of Kaggle competition also results in another dilemma - whether to trust the performance of our own validation set or the public leaderboard score. It is uncertain which of the validation set score or the public leaderboard score is a better indicator of the private leaderboard score - determiner of the final ranking.

Since we have access to all of these scores, the performance of models as reflected by the private leaderboard is used to determine the best feature combination - only lag features.

5. Model Fine-tuning

Algorithms can be trained with different model settings. To find the best model setting, experiments are conducted on the same model with different model settings, but with the same data setting which exclude all supplementary data.

5.1 Linear Regression

The linear regression model can be built with or without recursive feature elimination, which can be achieved by the estimator RFECV of Scikit Learn that performs feature ranking with recursive feature elimination and cross-validated selection of the best number of features. As shown below, for both private and public leaderboard, the linear regression model built with full features perform slightly better than the one built with feature elimination. However, the observation is the reverse for out validation set. Again, the private leaderboard result will be prioritized. Thus, the final linear regression model will be built without feature elimination.

LR Mode	lag_ feature	mean_ encoding	holiday	oil	pay_ day	Validatio n Score	Private Score	Public Score	Weighted Score
FULL	TRUE	FALSE	FALSE	FALSE	FALSE	0.61161	0.53281	0.52072	0.52909
RFECV	TRUE	FALSE	FALSE	FALSE	FALSE	0.61149	0.53284	0.52074	0.52912

5.2 Decision Tree

The decision tree model can be trained with or without pruning. Setting the minimum number of samples required to split an internal node as 2, the decision tree is allowed to grow to the max, achieving a weighted score of 0.75678. Pruning can be performed by controlling the value of of the minimum number of samples required, which is optimized though cross validation. As expected, the fully grown decision tree is overfitted and the pruned decision tree performs much better, as shown below. The tree without pruning obtained higher error than the one with pruning, for all private leaderboard, public leaderboard and validation set.

Tree Mode	lag_ feature	mean_ encoding	holiday	oil	pay_ day	Validatio n Score	Private Score	Public Score	Weighted Score
Pruned	TRUE	FALSE	FALSE	FALSE	FALSE	0.61381	0.53617	0.52812	0.53369
Full	TRUE	FALSE	FALSE	FALSE	FALSE	0.86252	0.76146	0.74625	0.75678

The hyperparameter *min_samples_split* in the DecisionTreeClassifier estimator of Scikit Learn sets the minimum number of samples required to split an internal node. For each of the 16 day, 5 decision trees models are trained with different *min_samples_split* ([2000, 2500, 3000, 3500, 4000]) on the train set, with all models evaluated on the validation set and the one with lowest NWRMSLE retained. As shown below, the optimized *min_samples_split* is 3500 for most days.

Day	0	1	2	3	4	5	6	7	8
min_samples_split	3500	3500	2500	3500	3000	3500	3500	3500	3500
Day	8	9	10	11	12	13	14	15	
min_samples_split	3500	3500	3500	3500	3500	3500	3500	3500	

5.3 LightGBM

The gradient boosting model is implemented by the LightGBM library which can train a tree-based gradient boosting model. One important hyperparameter of lightgbm is the *num_boost_round* which sets determined the total number of boosting iterations when training the model. The maximum rounds is set to be 500 while the validation dataset is used to monitor the model's training process and force early stopping when there are signs of overfitting, ie. the score of the validation set stops improving any more.

Another useful hyperparameter for LightGBM is *weight,* which allows the setting of weights of training data. This is quite important because in this competition, the perishable items are given a weight of 1.25 where all other items are given a weight of 1. Thus, the weight of each observation is also passed in to the model during the training through this hyperparameter, based on the item's perishable status indicated in the supplementary data for item.

The LightGBM library can train a tree-based gradient boosting model with or without specifying the categorical features. Thus, experiments are conducted to compare the effects of specifying the categorical features ('item_nbr', 'family', 'class', 'store_nbr', 'city', 'state', 'type', 'cluster', 'perishable') when constructing the model. As shown below, the lightgbm performs better when not specifying the categorical features.

lgb_cat	**lag_ feature**	**mean_ encoding**	**holiday**	**oil**	**pay_ day**	**Validatio n Score**	**Private Score**	**Public Score**	**Weighted Score**
FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	0.59659	0.51948	0.51364	0.51768
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	0.60429	0.52959	0.51797	0.52602

5.4 Neural Network

Apart from the models above, we also implemented a Neural Network model. We used a Sequential model with seven densely connected hidden layers and an output layer that returns a single, continuous value. In the hidden layers, we used Rectified

Linear Unit (relu) Activation Function, and batch normalization to enables faster and more stable training of the model, it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behaviour of the gradients, allowing for faster training. After the batch normalization, we also added in Dropout to the layers, it randomly dropping out nodes during training and offers a very computationally cheap and remarkably effective regularization method to reduce overfitting and generalization errors. As for the output layer, we used Linear Activation Function.

After we built the model, we implemented Checkpoint and Early Stopping to our model. We were training a huge model, and it takes hours to train, with Checkpoint it allows us to continue training, resume on failure. We save the model weights each time an improvement is observed during training. While having the best model weights saved, we could simply load the best weights from the previous experiment and make predictions based on the model. Besides the Checkpoint, Early Stopping also helps us in deciding the best number for the epochs in the model. During the training of the model, Early Stopping evaluates the network performance on the test set at the end of each epoch, if the network outperforms the previous best model, it then saves a copy of the network at the current epoch. Finally, it takes as the final model with the best accuracy on the test set.

As the training log and diagrams shows in Appendix 3 for day 1 prediction, the model stops at 14th epoch, because the loss and MSE is not improving anymore in the following epoch.

The table below shows the number of epochs, loss and mse for Day 1 to Day 16.

Day	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Epochs	14	14	10	14	17	7	12	6
Loss	0.3447	0.3625	0.3802	0.3983	0.4057	0.3999	0.3837	0.3795
MSE	0.3446	0.3607	0.3796	0.3972	0.4049	0.3997	0.3827	0.3799

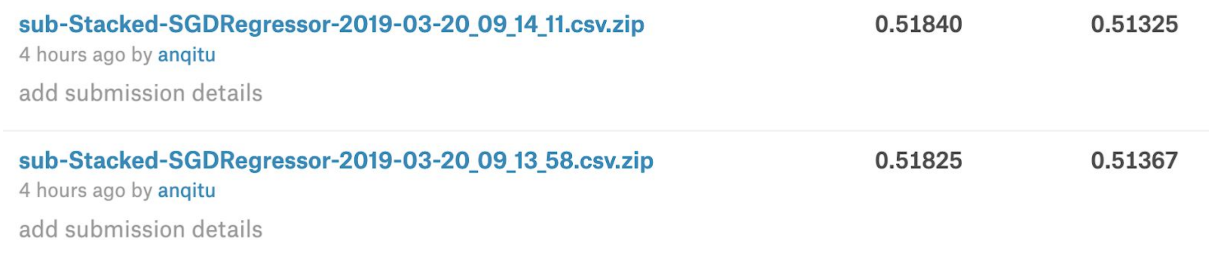
Day	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Epochs	17	9	14	18	16	18	16	10
Loss	0.3828	0.3924	0.4199	0.4295	0.4113	0.3954	0.3836	0.3937
MSE	0.3815	0.3923	0.4195	0.4294	0.4095	0.3947	0.3844	0.3925

6. Performance Comparison Across Models

For all models, selected hyperparameters are tuned by cross validation. The final tuned model is then trained on the combined data of both train and validation set. Only lag features of observed variables are included in the data since the supplementary data has been proved to be detrimental to the model's performance. The runtime of training each model is also recorded. (All models are run on a idle MacBook Pro with a 2.5 GHz Intel Core i7 processor and 16GB RAM.)

The screenshots of the score of best model on both Private and Public Leaderboard for each algorithm are documented below, in the order of descending performance.

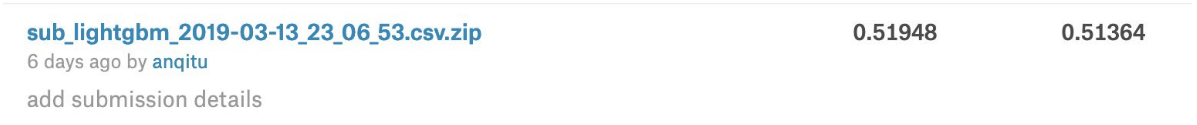
1. Stacked SGDRegressor



sub-Stacked-SGDRegressor-2019-03-20_09_14_11.csv.zip 0.51840 0.51325
4 hours ago by anqitu
add submission details

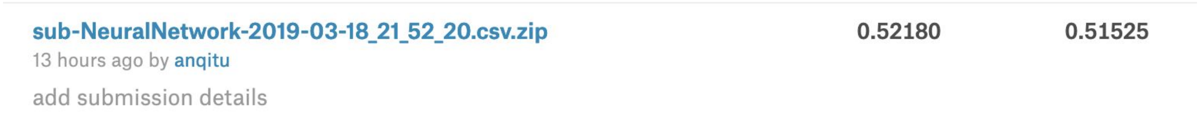
sub-Stacked-SGDRegressor-2019-03-20_09_13_58.csv.zip 0.51825 0.51367
4 hours ago by anqitu
add submission details

2. LightGBM



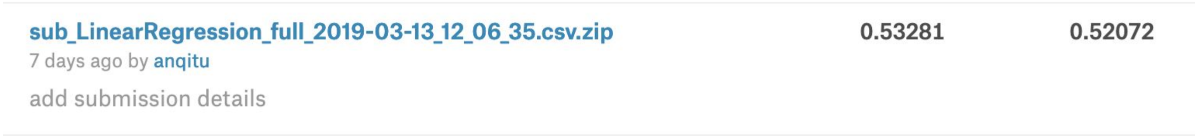
sub_lightgbm_2019-03-13_23_06_53.csv.zip 0.51948 0.51364
6 days ago by anqitu
add submission details

3. Neural Network



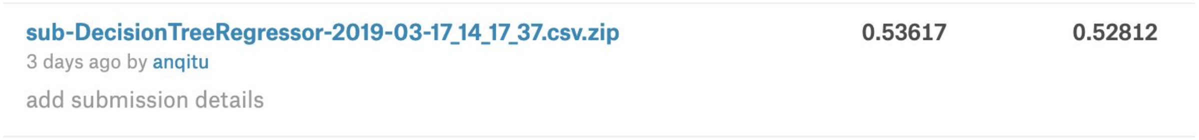
sub-NeuralNetwork-2019-03-18_21_52_20.csv.zip 0.52180 0.51525
13 hours ago by anqitu
add submission details

4. Linear Regression



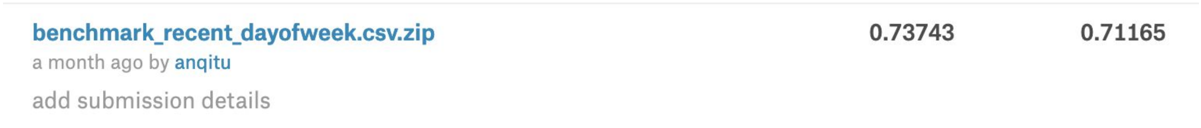
sub_LinearRegression_full_2019-03-13_12_06_35.csv.zip 0.53281 0.52072
7 days ago by anqitu
add submission details

5. Decision Tree



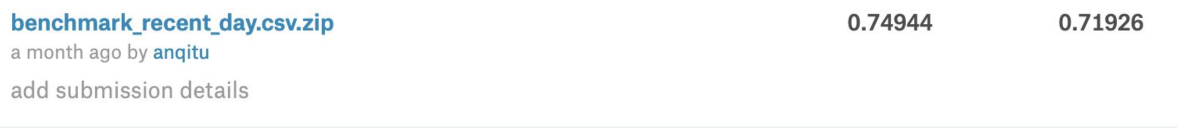
sub-DecisionTreeRegressor-2019-03-17_14_17_37.csv.zip 0.53617 0.52812
3 days ago by anqitu
add submission details

6. Benchmark by Most Recent Day of Week



benchmark_recent_dayofweek.csv.zip 0.73743 0.71165
a month ago by anqitu
add submission details

7. Benchmark by Most Recent Day



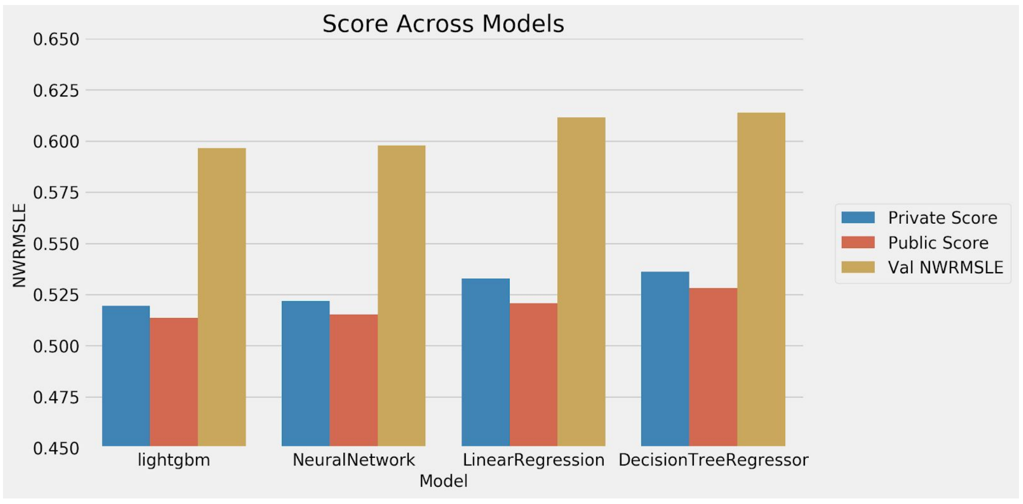
benchmark_recent_day.csv.zip 0.74944 0.71926

a month ago by anqitu

add submission details

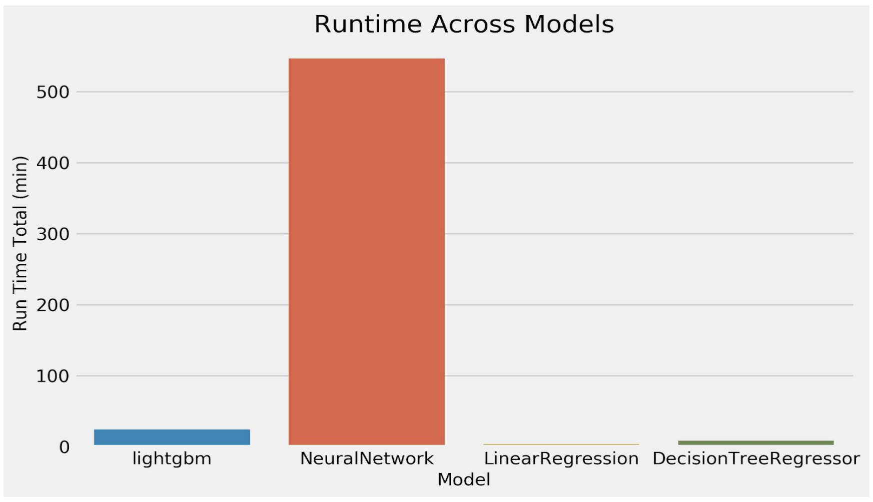
Here is a summary of the performance of each model:

	LightGBM	Neural Network	Linear Regression	Decision Tree
Private Score	**0.51948**	0.5218	0.53281	0.53617
Public Score	**0.51364**	0.51525	0.52072	0.52812
Weighted Score	**0.51768**	0.51978	0.52909	0.53369
Validation Score	**0.59659**	0.59790	0.61161	0.61381
Run Time (min)	24.43861	546.31673	**3.99731**	8.54229



As shown in the plot above, LightGBM, the tree-based gradient boosting algorithm, performs the best, followed by neural network, decision tree and linear regression. This ranking is consistent for all validation set, private leaderboard and public leaderboard. The NWRMSLE scores for the private leaderboard are also consistently slightly higher than the ones of public leaderboard, and much lower compared to the ones of validation set.

Besides, though both being tree-based, LightGBM is proven to be more powerful than decision tree, due to the ensembling technique implemented with gradient boosting.

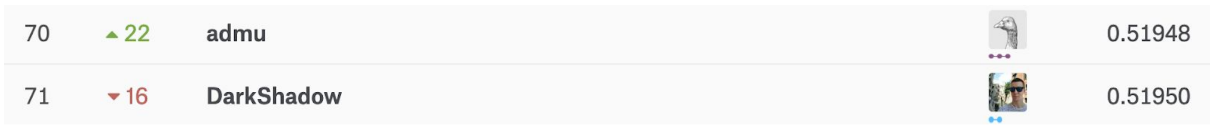


Regarding the runtime, neural network algorithm requires the longest training time, which though could be largely reduced if GPU is used. Notwithstanding, this still reflects the high computational power demanded by the neural network model, as compared to other algorithms.

The trade-off between performance and runtime is also observed by comparing both the model performance and tuntime. Putting neural network as an outlier aside, though LightGBM gets the lowest error and achieves the best performance, its runtime is also the longest. Most importantly, the LightGBM library is known for its fast training speed. The time would be much longer if other libraries implementing tree-based gradient boosting are used, such as XGBoost. Thus, the general trend is that the better performance always requires longer training time.

However, the reverse is not necessarily true - models with longer training time has no guarantee to result in better result. This is shown by the neural network which has a much longer training time, but slightly worse result than LightGBM, and the decision tree, which has a much longer training time, but worse result than linear regression.

In this case, despite the tradeoff between performance and training time, the runtime difference is not that huge enough to reject LightGBM, especially in such a data competition where every minimal score improvement could mean a huge jump in the ranking. This best performed model LightGBM places us as the 71st out of 1675 on the private leaderboard as shown in the screenshot below.



70	▲ 22	admu	0.51948
71	▼ 16	DarkShadow	0.51950

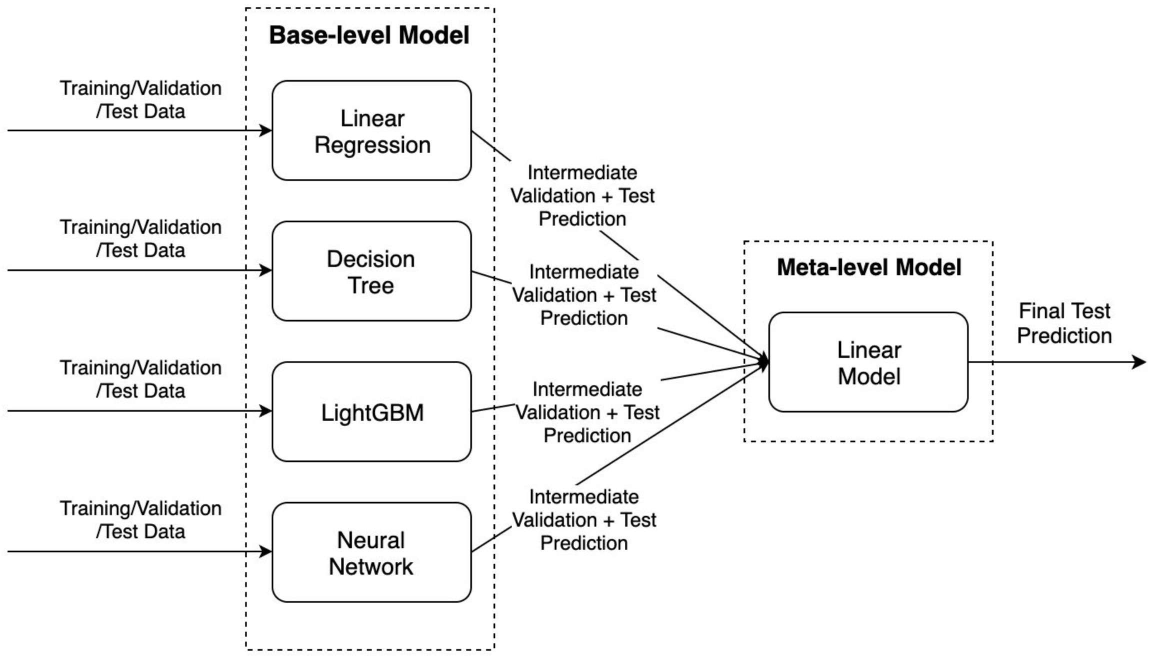
It places us as the 518th out of 1675 on the public leaderboard as shown in the screenshot below.

518	**YutingGui**	0.51361
519	**Tomohiko Itano**	0.51369

7. Effectiveness of Stacking

The idea of stacking is to tackle the machine learning problem with different types of models which are capable to learn some part of the problem, but not the whole space of the problem, and then use each base-level model to build an intermediate prediction. A final meta-level model will learn from the intermediate predictions for the same target. This could potentially improve the overall performance, thus building a model which is better than any individual intermediate model.

To study the effectiveness of stacking in this machine learning problem, a linear regression model is built with all best predictions by each model for the validation set, and then used to predict for the test set. As shown in the diagram below, after all the four base-models are fine-tuned with the cross validation method, they are trained on the training set to produce the intermediate predictions for the validation set, and then trained on the combination of train and validation set to produce the intermediate predictions for the test set. The meta-level model, is trained on all the intermediate predictions of validation set for the same target, and then used to produce the final prediction for the test set with all the intermediate predictions of test set.

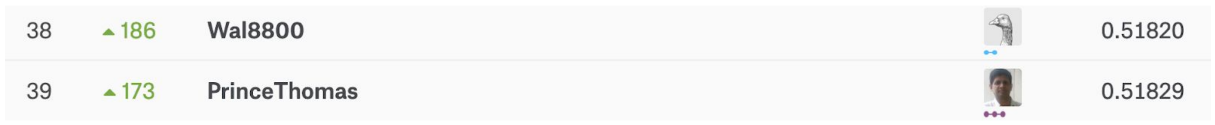

Base-level Model
Training/Validation /Test Data
Linear Regression
Training/Validation /Test Data
Decision Tree
Training/Validation /Test Data
LightGBM
Training/Validation /Test Data
Neural Network
Intermediate Validation + Test Prediction
Intermediate Validation + Test Prediction
Intermediate Validation + Test Prediction
Intermediate Validation + Test Prediction
Meta-level Model
Linear Model
Final Test Prediction

Two linear model algorithm, ordinary least squares linear regression (implemented via the LinearRegression class of Scikit Learn) and stochastic gradient descent linear regression (implemented via the SGDRegressor class of Scikit Learn), are trained on the previously attained validation intermediate predictions and used to predict for the testset with the testset intermediate predictions. The coefficient of the linear models can also be considered as the weight of each intermediate base-level model.

Initially, all four predictions are used to train the meta-model. The SGDRegressor attains lower error and thus better performance. The negative coefficient of linear regression intermediate prediction reflects that its prediction might be so worse compared to others that it could actually worsens the performance of the stacked model. Thus, another two meta-models are trained on only the predictions by LightGBM, Neural Network and Decision Tree. Due to the small weight of Decision Tree, two more meta-models are trained on only the predictions by LightGBM and Neural Network. The SGDRegressor consistently achieves better performance than LinearRegression.

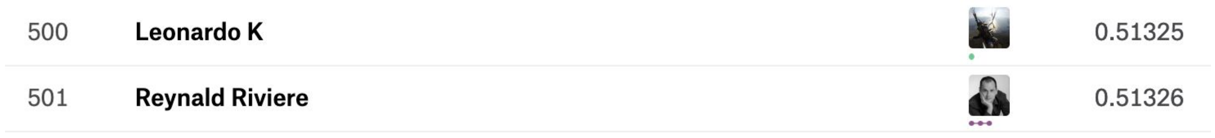
	LightGBM	**Neural Network**	**Decision Tree**	**Linear Regression**	**Private Score**	**Public Score**
LinearRegression	0.60353	0.46152	0.04723	-0.09199	0.51898	0.51479
SGDRegressor	0.58888	0.47543	0.04861	-0.09851	**0.51825**	0.51367
LinearRegression	0.56217	0.44064	0.01828	NA	0.51907	0.51423
SGDRegressor	0.54604	0.45397	0.01790	NA	0.51840	**0.51325**
LinearRegression	0.57739	0.44362	NA	NA	0.51904	0.51424
SGDRegressor	0.56340	0.45667	NA	NA	0.51848	0.51347

As discussed in previous section, the best performed single model LightGBM attains a private score of 0.51948 and public score of 0.51364. Here, the best stacking model, the SGDRegressor trained on all four intermediate models' predictions, obtains a lower error of 0.51825 on the private leaderboard, ef. However, the same model perform the worst on the public leaderboard. It is also weird that the weight of the linear regression prediction is negative, but it still helps improve the model's performance slightly, as reflected by the private leaderboard scores.



38	▲186	Wal8800	0.51820
39	▲173	PrinceThomas	0.51829

Though less important than the private leaderboard, the SGDRegressor trained on the intermediate models' predictions excluding linear regression obtains the lowest error of 0.51325 on the public leaderboard, pushing our ranking from the 518th to the 500th.



500	Leonardo K	0.51325
501	Reynald Riviere	0.51326

8. Limitation and Recommendation

Larger Training Data Size
Given the limited computing resources and the large dataset provided, only data from year 2017 is used. Thus, it is recommended to Include data before 2017 which allows the models to learn from more data, thus potentially further improving the performance.

More Engineered Features
Create and experiment with more statistics on the lag features of sales and promotion, such as using median, standard deviation and difference. Explore more other ways to use the supplementary data that can potentially improve the model performance.

One hot Encoding
When using mean encoding, we use target classes to encode for our training labels which may leak data about the predictions causing the encoding to become biased. Relying on an average value might cause overfitting and rendering the feature biased. A better way is to use one hot encoding to encode categorical variables.

One hot Encoding takes a column with categorical data which has been label encoded and create another N-1 more columns. Linear regression and Neural Network uses linear combination of independent variables(X) to predict the dependent variable (Y), it will give advantages to the labels with higher index, 3>2>1>0. In all such cases, One Hot Encoding outperforms Mean Encoding as it creates another N-1 more columns and that makes more sense as we have unbiased linear combination now.

Targeted Feature Selection
The feature combination is optimized using experiments with only linear regression for time-saving. The assumption is that the optimized feature combination for linear regression is also the optimized one for all other algorithms. However, this might not always hold. Thus, it is recommended to experiment with all feature combinations with all algorithms to find the best one for each model.

Finer Model Tuning
Only a few selected hyperparameters of some models are tuned. More hyperparameters can be tuned for each model, such as number of layers, learning rate, number of iteration, number of layers for Neural Network. Max_features, min_samples_split, etc for RandomForestRegressor. More parameters tuning could potentially improve the performance of the model.

Also, the selected hyperparameters are tuned with a relatively large interval. For example, the optimal value of minimum samples to split for the decision tree is found from a list of numbers [2000, 2500, 3000, 3500, 4000] through cross validation. A potential way to improve the model performance is to find the optimal hyperparameter by using smaller interval such as [2000, 2250, 2500, 2750, 3000, 3250, 3500, 3750, 4000].

More diversity of Models for Stacking
Increase the number and diversity of models use in stacking can further improve the its performance. Models trained using different machine learning algorithm, hyperparameter settings, training sets and features provides a huge diversity. It is recommended to increase the number of models used in each layers and increase the number of layers to further improve the performance of stacking.

9. Conclusion

This project forecasts grocery stores product sales. Due to the large size of the training data and limited computation power the team has, we only used the sales record from 2017 onwards. We have performed data pre-process on the sales data such as included zero unit sales and adding items that are not in training dataset but exist in testing dataset. We also performed feature engineering such as implement Mean Encoding and adding features like promotion, oil price, holiday and payday. However, after running on the model, the newly added features do not contribute to the final prediction, we removed the newly added features at the end.

Apart from the data preparation, we also built four models to train our dataset which are Linear Regression, Decision Tree, LightGBM and Neural Network. After which, we made use of the stacking which use the four models to build an intermediate prediction, trained on all the intermediate predictions and produce the final prediction for the test set. This methodology was pushed our ranking from the 71st to the 39th in the leaderboard.

However, many optimizations could be done in the future. Future work concerns more engineered features, using of One Hot Encoding, building an advanced model such as LSTM, finer model tuning and have more models for stacking. With all these, the performance could have a vast improvement.

Overall, this project has largely improved our knowledge of machine learning and has been very enriching and exciting on how so many factors can affect the final predictive results.

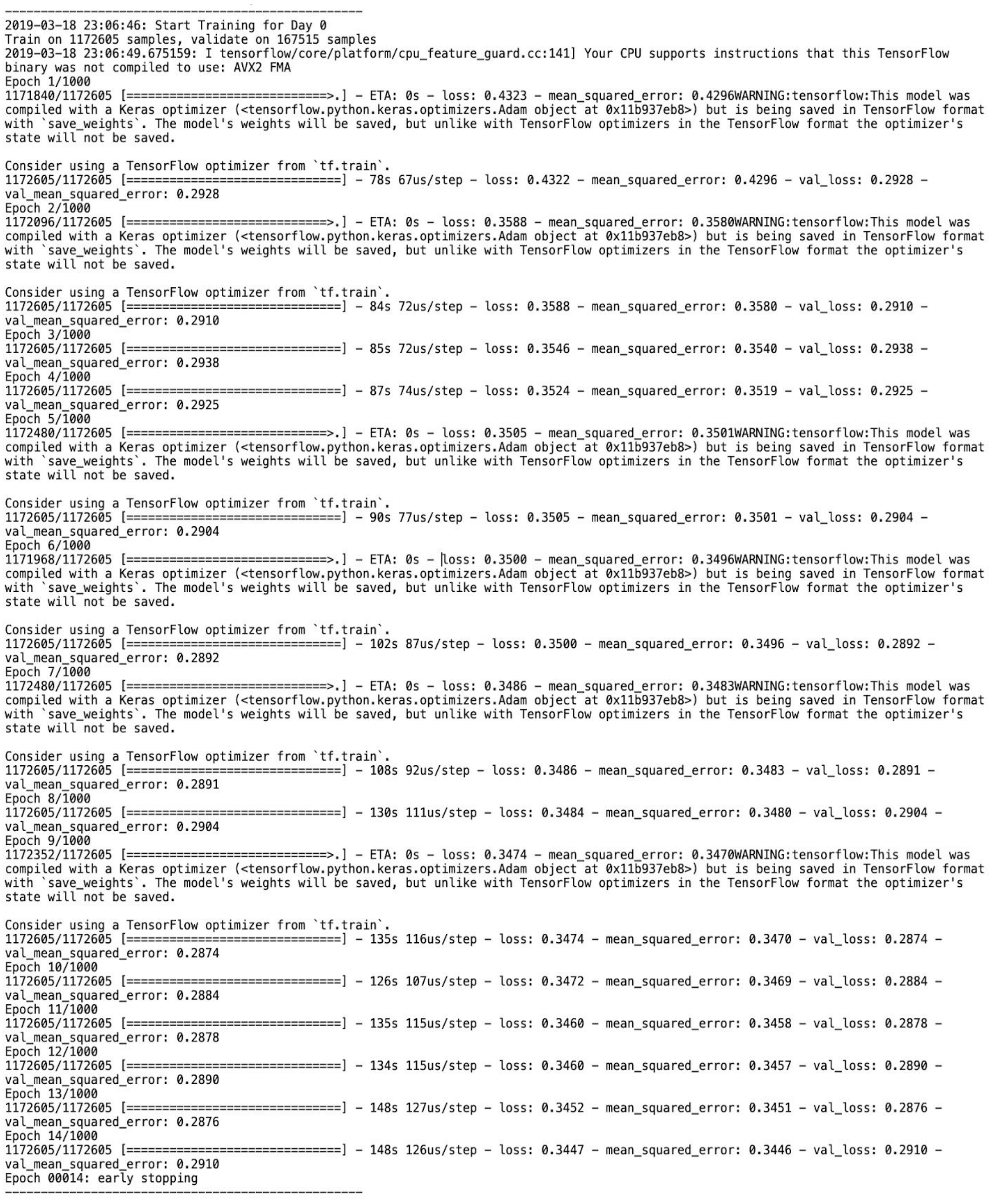
APPENDIX 1: Mean Coding Correlation

The correlation value of each mean encoded column using the leave one out scheme is documented below. Only the first 4 mean encodings with the correlation higher than 0.25 threshold are used.

Column	Correlation
item_nbr	**0.57304**
family	**0.27621**
class	**0.36346**
store_nbr	**0.26744**
city	0.18541
state	0.18163
type	0.17441
cluster	0.21086
item_nbr&store_nbr	0.00143
item_nbr&city	0.0144
item_nbr&state	0.01574
item_nbr&type	0.00371
item_nbr&cluster	0.00052
family&store_nbr	0.00584
family&city	0.04849
family&state	0.03801
family&type	-0.02432
family&cluster	0.03154
class&store_nbr	0.00522
class&city	0.04011
class&state	0.03406
class&type	0.01558
class&cluster	0.0111

APPENDIX 2: Neural Network Prediction on Day 1

The figures below shows the training process for Day 1 purchasing and sales forecasting and the changes of loss and Mean Square Error (mse).



```
--------------------------------------------------
2019-03-18 23:06:46: Start Training for Day 0
Train on 1172605 samples, validate on 167515 samples
2019-03-18 23:06:49.675159: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow
binary was not compiled to use: AVX2 FMA
Epoch 1/1000
1171840/1172605 [============================>.] - ETA: 0s - loss: 0.4323 - mean_squared_error: 0.4296WARNING:tensorflow:This model was
compiled with a Keras optimizer (<tensorflow.python.keras.optimizers.Adam object at 0x11b937eb8>) but is being saved in TensorFlow format
with `save_weights`. The model's weights will be saved, but unlike with TensorFlow optimizers in the TensorFlow format the optimizer's
state will not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
1172605/1172605 [==============================] - 78s 67us/step - loss: 0.4322 - mean_squared_error: 0.4296 - val_loss: 0.2928 -
val_mean_squared_error: 0.2928
Epoch 2/1000
1172096/1172605 [============================>.] - ETA: 0s - loss: 0.3588 - mean_squared_error: 0.3580WARNING:tensorflow:This model was
compiled with a Keras optimizer (<tensorflow.python.keras.optimizers.Adam object at 0x11b937eb8>) but is being saved in TensorFlow format
with `save_weights`. The model's weights will be saved, but unlike with TensorFlow optimizers in the TensorFlow format the optimizer's
state will not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
1172605/1172605 [==============================] - 84s 72us/step - loss: 0.3588 - mean_squared_error: 0.3580 - val_loss: 0.2910 -
val_mean_squared_error: 0.2910
Epoch 3/1000
1172605/1172605 [==============================] - 85s 72us/step - loss: 0.3546 - mean_squared_error: 0.3540 - val_loss: 0.2938 -
val_mean_squared_error: 0.2938
Epoch 4/1000
1172605/1172605 [==============================] - 87s 74us/step - loss: 0.3524 - mean_squared_error: 0.3519 - val_loss: 0.2925 -
val_mean_squared_error: 0.2925
Epoch 5/1000
1172480/1172605 [============================>.] - ETA: 0s - loss: 0.3505 - mean_squared_error: 0.3501WARNING:tensorflow:This model was
compiled with a Keras optimizer (<tensorflow.python.keras.optimizers.Adam object at 0x11b937eb8>) but is being saved in TensorFlow format
with `save_weights`. The model's weights will be saved, but unlike with TensorFlow optimizers in the TensorFlow format the optimizer's
state will not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
1172605/1172605 [==============================] - 90s 77us/step - loss: 0.3505 - mean_squared_error: 0.3501 - val_loss: 0.2904 -
val_mean_squared_error: 0.2904
Epoch 6/1000
1171968/1172605 [============================>.] - ETA: 0s - loss: 0.3500 - mean_squared_error: 0.3496WARNING:tensorflow:This model was
compiled with a Keras optimizer (<tensorflow.python.keras.optimizers.Adam object at 0x11b937eb8>) but is being saved in TensorFlow format
with `save_weights`. The model's weights will be saved, but unlike with TensorFlow optimizers in the TensorFlow format the optimizer's
state will not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
1172605/1172605 [==============================] - 102s 87us/step - loss: 0.3500 - mean_squared_error: 0.3496 - val_loss: 0.2892 -
val_mean_squared_error: 0.2892
Epoch 7/1000
1172480/1172605 [============================>.] - ETA: 0s - loss: 0.3486 - mean_squared_error: 0.3483WARNING:tensorflow:This model was
compiled with a Keras optimizer (<tensorflow.python.keras.optimizers.Adam object at 0x11b937eb8>) but is being saved in TensorFlow format
with `save_weights`. The model's weights will be saved, but unlike with TensorFlow optimizers in the TensorFlow format the optimizer's
state will not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
1172605/1172605 [==============================] - 108s 92us/step - loss: 0.3486 - mean_squared_error: 0.3483 - val_loss: 0.2891 -
val_mean_squared_error: 0.2891
Epoch 8/1000
1172605/1172605 [==============================] - 130s 111us/step - loss: 0.3484 - mean_squared_error: 0.3480 - val_loss: 0.2904 -
val_mean_squared_error: 0.2904
Epoch 9/1000
1172352/1172605 [============================>.] - ETA: 0s - loss: 0.3474 - mean_squared_error: 0.3470WARNING:tensorflow:This model was
compiled with a Keras optimizer (<tensorflow.python.keras.optimizers.Adam object at 0x11b937eb8>) but is being saved in TensorFlow format
with `save_weights`. The model's weights will be saved, but unlike with TensorFlow optimizers in the TensorFlow format the optimizer's
state will not be saved.

Consider using a TensorFlow optimizer from `tf.train`.
1172605/1172605 [==============================] - 135s 116us/step - loss: 0.3474 - mean_squared_error: 0.3470 - val_loss: 0.2874 -
val_mean_squared_error: 0.2874
Epoch 10/1000
1172605/1172605 [==============================] - 126s 107us/step - loss: 0.3472 - mean_squared_error: 0.3469 - val_loss: 0.2884 -
val_mean_squared_error: 0.2884
Epoch 11/1000
1172605/1172605 [==============================] - 135s 115us/step - loss: 0.3460 - mean_squared_error: 0.3458 - val_loss: 0.2878 -
val_mean_squared_error: 0.2878
Epoch 12/1000
1172605/1172605 [==============================] - 134s 115us/step - loss: 0.3460 - mean_squared_error: 0.3457 - val_loss: 0.2890 -
val_mean_squared_error: 0.2890
Epoch 13/1000
1172605/1172605 [==============================] - 148s 127us/step - loss: 0.3452 - mean_squared_error: 0.3451 - val_loss: 0.2876 -
val_mean_squared_error: 0.2876
Epoch 14/1000
1172605/1172605 [==============================] - 148s 126us/step - loss: 0.3447 - mean_squared_error: 0.3446 - val_loss: 0.2910 -
val_mean_squared_error: 0.2910
Epoch 00014: early stopping
--------------------------------------------------
```