Machine Learning Engineer Nanodegree

Capstone Project

Dylan Chu
August 26th, 2018

I. Definition

Project Overview

The project is to build a model to submit predictions to a Kaggle competition called Predict Future Sales. As the name implies, it involves looking at historical sales records to predict future sales counts.

Sales forecasting is a common task undertaken by businesses. I have yet to produce any sales forecasts during my career but I have seen many of these forecasts shown and discussed during quarterly corporate communications meetings. While I would imagine there are statistical tools and methods which could be employed to predict future sales totals, there is probably a great deal of human intuition involved as well. Hence, just as a financial officer or product manager looks over sales amounts in previous periods to estimate future sales amounts, it seems possible that we can feed machine learning models historical sales data and have it predict future sales.

A research paper published in 2013, called “Machine Learning Strategies for Time Series Forecasting,“ pointed out the increased awareness and usage of machine learning methods in the forecasting community. The authors of the paper formulated the validity and assumptions of turning forecasting into a supervised learning problem. As well, they discussed machine learning techniques to use such as approximating the target variable of a datapoint by comparing it to its nearest “neighbour.”

In a more recent research paper titled “Statistical and Machine Learning forecasting methods: Concerns and ways forward,” the researchers compared the accuracy of 8 statistical methods against 10 machine learning ones. They found that statistical methods were generally better at forecasting. However, the researchers also provided some suggestions for improving the usage of machine learning methods on time series data and expressed optimism about the advancement of the field.

Problem Statement

The competion provides all participants with 34 consecutive months of historical sales counts of items by shop. There are 60 different shops and over 22000 items. The challenge is to accurately predict the 35th month of sales counts for a given list of shop and item pairings.

Since the challenge involves predicting the “next step” after being provided a series of “steps”, the strategy is to train a model to do one-step forecasting. We can transform the problem into a supervised learning problem and use a regression algorithm to predict the sales count for the next month using data from previous months.

To do this, I will need to create a training data file with records that can be viewed in the form of *X*, *y*. The set of features, *X*, includes the sales count for a shop and an item for a particular year and month. Let's call this month the *base month*. The target variable, *y*, is the sales count for the month following the base month. Let's call this month the *target month*. An example of records with their X and y constituents is shown in the table below.

Features (X)					***Target Variable (y)***
year	month	shop id	item id	sales count for base month	*sales count for target month*
2014	March	55	4302	25	*28*
2014	April	55	4302	28	*26*

Other features that could be included in *X* are sales counts for months prior the base month, information related to the category of the item, and information related to the shop.

After creating a suitable training data file, I will use a supervised learning regression algorithm to train a model to predict the target variable. I will choose from among the many algorithms available for public use. Of course, I will need to tune the hyperparameters for the selected regressor and use them to train the model. The trained model will predict the sales counts for a list of 200000+ shop/item pairings provided by the competition. Using these predictions, I will create a submission file to submit to Kaggle to assess the performance of the model. Kaggle will evaluate the predictions based on the evaluation metric discussed in the next section.

Metrics

As with any Kaggle competition, there is a given evaluation metric that will be used to judge the accuracy of all submissions. The metric for this competition is the root mean squared error (RMSE) which is defined below.

$$RMSE=\sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i-y_i)^2}{N}}$$

This metric sums up the square of the difference between each prediction and the real target value, divides that by the number of predictions before taking the square root of the whole thing. A lower RMSE score is more desirable. It is a reasonable metric for regression problems as predictions that are farther away from their true targets will push the RMSE score more than proportionally higher.

II. Analysis

Data Exploration

Specifically, the competition provides 6 CSV files. One file called *test.csv* has a listing of 214200 shop and item combinations for which participants are tasked with providing predictions. Another file called *sample_submission.csv* shows how the predictions should be formatted before being submitted to Kaggle. There are 2 more files called *shops.csv* and *item_categories.csv* which are just reference files mapping an id to a name for shops and item categories respectively. Another provided file called *items.csv* has 3 fields, mapping an item id to an item name and an item category id.

The most important provided file is a file called *sales_train_v2.csv*. Each record in this file is a daily count for an item at a particular shop. In total, there are 2935849 records and a sample of 5 records from the file is shown below.

date	**date_block_num**	**shop_id**	**item_id**	**item_price**	**item_cnt_day**
02.01.2013	0	59	22514	999.00	1.0
03.01.2013	0	25	2252	899.00	1.0
05.01.2013	0	25	2252	899.00	-1.0
06.01.2013	0	25	2254	1709.05	1.0
15.01.2013	0	25	2555	1099.00	1.0

The *date_block_num* field is a label for a year and month combination. The label “0” represents January 2013, “1” represents February 2013, and so forth. The file has data for *data_block_num* values from 0 to 33.

The *item_cnt_day* field has values in the range from -22 to 2169. There is no description for this field so presumably, this value represents net sales (sales minus returns). This report will use the term *sales count* for the values in this field.

With the problem involving the prediction of monthly sales counts, it is important to analyze the aggregation of these daily sales count records based on unique groupings of *date_block_num*, *shop_id*, and *item_id*. There are 1609124 records after aggregation. The range of values for monthly sales count is now from -22 to 2253. With a mean of only 2.27 and a standard deviation of about 8.65, the distribution of the monthly sales counts is highly skewed to the right. As well, there are 915 records where the monthly sales count is negative, accounting for 0.05% of all monthly sales counts records.

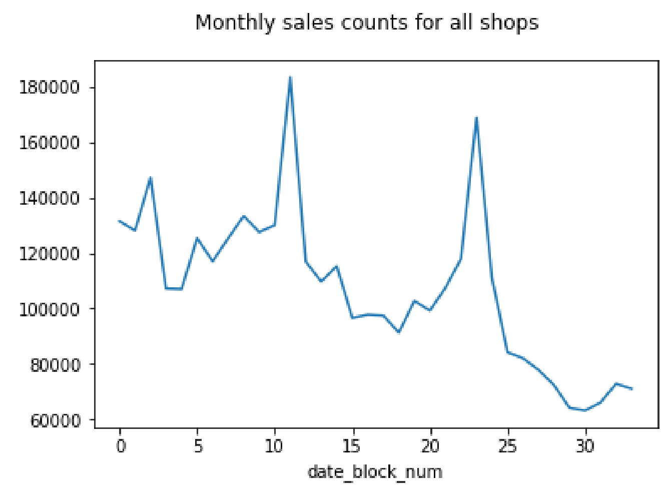
The next thing that I wanted to look at was whether there was any correlation between the sales counts for a base month and the months prior to it. Namely, I wanted to analyze the sales counts among the following set of months:

- a base month (*t*)
- the month prior to the base month (*t-1*)
- 2 months prior to the base month (*t-2*)
- 3 months prior to the base month (*t-3*)
- half a year prior to the base month (*t-6*)
- a year prior to the base month (*t-12*)

A label (i.e. *t-1*) for each month is given in the parenthesis above. I used a scatter matrix to view the data relationship between these months. A discussion of this visualization is given in the next section.

I also wanted to explore whether the shops could be grouped into different clusters. This could be a more useful piece of data to train a model with than the shop id. I further aggregated the monthly sales count to get monthly sales counts by shop. I used the Gaussian Mixture Model clustering algorithm on the monthly sales counts by shop data and found that it did not find any useful way to cluster the shops.

I did find seasonal sales patterns when looking at a monthly sales count for all the shops combined.

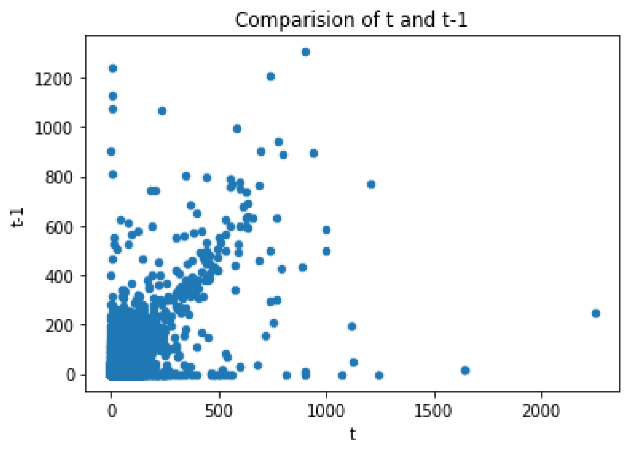


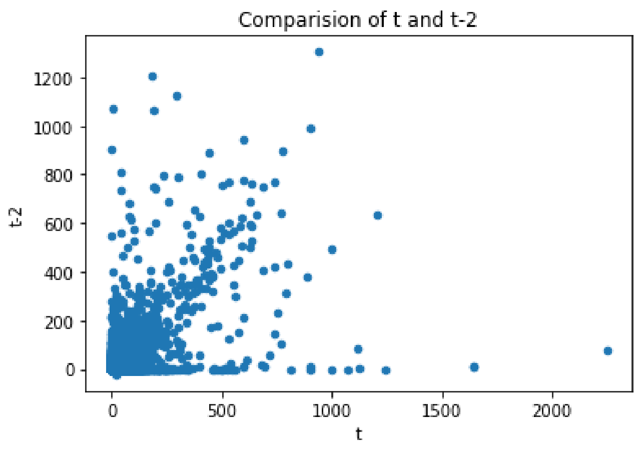
It looks like sales shoot up in December and drop precipitously in January. In addition, the summer months seem to be lowest selling months for the year. Another apparent trend is the gradual decline in total sales over the training period.

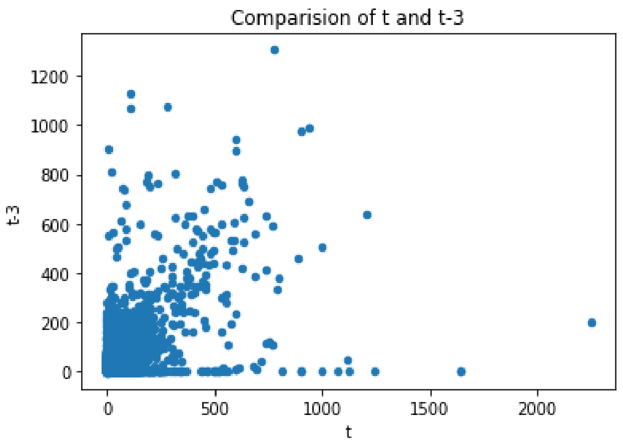
As well, I believe the category of the item is a good piece of info to feed into the model. There are 84 categories and I went through a similar exercise to attempt to extract categories “types”. I merged in the category id for each item and then aggregated the monthly sales counts by shop and item to arrive at monthly sales counts by category. Unfortunately, I could not find any clusters of categories that were useful. All the steps that I went through in the analysis of the data is shown in the *data_exploration.ipynb* notebook.

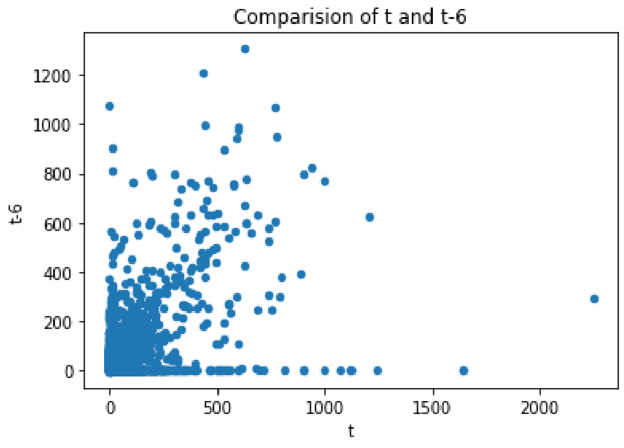
Exploratory Visualization

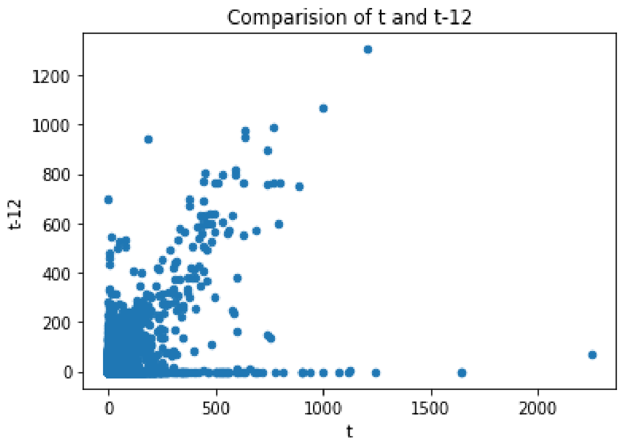
As mentioned in the previous section, I wanted to explore any correlation between the sales counts of a base month and its prior months. The scatter plots below map the sales counts for a base month (*t*) with the sales count for one of: the previous month (*t-1*), two months ago (*t-2*), three months prior (*t-3*), 6 months before (*t-6*) or a year ago (t12).











Firstly, from looking at all the plots, for sales counts in *t* over 300, there seems to be some correlation with the sales counts for the compared month. This relationship becomes less evident as the time distance between the two months increases but it becomes clearer when the months are a year apart. Also, again for sales counts in *t* over 300, the sales counts for the base month is generally less than the sales count for the earlier month. This is most evident again when a year separate the compared months.

For sales counts below 300, there seems to be no correlation as shown by the almost solid block of points from 0 to about 300 on both the horizontal and vertical axes. There appears to be a lot of variance in the sales counts of the items from month to month.

Another interesting observation is the number of points where the sales count is zero for the prior month. This number increases as the distance between the prior month and the base month gets larger. This suggest that new products are being introduced monthly.

Additionally, from looking at the sale counts in *t* in each plot, there are two obvious outlier points in *t* where the sales count is greater than 1500. These two points do not appear on the vertical axis in the plot for *t* and *t-1*. This means that these two points are for records that have a max value of 33 for the *date_block_num* field. I do not need to specifically remove these two data points as all the records with a max value for *date_block_num* field will not be used for training.

Overall, it seems like there are some patterns (but also a lot of variance) in regards to how the sales counts for items changes from month to month. Hopefully, a trained model can discover some form of order among the chaos.

Algorithms and Techniques

The overall process to solving the problem involves 2 major tasks. The first task involves preprocessing the data to generate a suitable training data file. The second task is to use a regression algorithm to train a model with the training file. A step for the second task is to generate a submission file for Kaggle to evaluate the performance of the model.

My intention is to use Python language for all the coding tasks involved with the project. I will create one Jupyter notebook for each of the major tasks.

For preprocessing the data and creating the training data file, I will use the standard Python libraries such as numpy, pandas and mathplotlib. Firstly, I will load the data in the provided daily sales records file into a dataframe and use pandas functions to aggregate the records. Other pandas functions will be used to merge in other pieces of data (such as the item_category_id), to append the sales counts for prior months to each record and to replace the *date_block_num* field with a year and a month.

When training the model, I will experiment with using different subsets of all features available in the training data file. I will functions in the scikit-learn library to split up the records in the training data file and use 80% of the records as the training set and 20% of the records as the validation set.

The regression algorithms that I intend to train the model will be from existing Python packages. As mentioned earlier, I will evaluate a bunch of regressors first before picking one regressor to continue with. The candidates include the basic regressors (LinearRegressor, DecisionTreeRegressor), the regressors in the sklearn.ensemble module (AdaBoostRegressor, BaggingRegressor, ExtraTreesRegressor, GradientBoostRegressor, RandomForestRegressor) as well as two boosting algorithms (XGBoost, LightGBM) recommended by the reviewer of my project proposal. For this quick and dirty evaluation phase, I will stick with the default parameters for each regressor.

A widely-used algorithm for Kaggle competitions over the years is Random Forest (RandomForestRegressor). This approach uses a collection (“ensemble”) of weak decision trees and the outputs of these weak learners are averaged (in the case of regression problems) to get the final predictions. Each separate decision tree is trained on a set of records and features that are randomly selected.

The algorithm that has had recent success in Kaggle competitions is XGBoost. This is a gradient boosting decision tree algorithm. XGBoost also uses a collection of trees but it differs from Random Forest in that the trees are generated sequentially. Successive trees added to the ensemble are trained to minimize the error of the predictions of the existing trees in the collection.

An algorithm gaining popularity is LightGBM. Like XGBoost, it is a gradient boosting decision tree algorithm. It initially differed from XGBoost because its decision trees were trained by growing leaf-wise. XGBoost initially use level-wise growth, meaning the trees were balanced. However, XGBoost has since adopted leaf-wise growth so its trees could be deep and narrow as well. LightGBM can also handle categorical features inherently while XGBoost requires some form of label encoding. Another difference is in how the algorithms decide to split data. XGBoost uses the pre-sorted algorithm (split after sorting the values of a feature) and the histogram-based algorithm (split after putting features into bins).

LightGBM uses the gradients of the datapoints as a manner of filtering them before applying a split. As well, LightGBM bundles features that never nonzero simultaneously to reduce the dimensionality of the input space. These differences may be why LightGBM generally trains faster than XGBoost.

The training and validation RMSE scores for the predictions made by each regressor will determine which regressor I will ultimately use. To help find the optimal hyperparameters for this regressor, I will use GridSearchCV, again from the scikit-learn library. Then I will the early stopping option to train the model with the optimal hyperparameters (perhaps with some minor manual adjustments). Afterwards, I will use the trained model to make the requested predictions and submit the predictions to Kaggle to evaluate the performance of the model.

Benchmark

The benchmark model I will use as a comparision is from another participant in the Kaggle competition. A user called TrietChau created a kernel to guide beginners on how to tackle the challenge [2].

This person transformed the data into time-series data and used a LSTM network to train the model. A prominent idea in this user's solution is to train the benchmark model using the sequence of data from from July to November of 2013 and July to November of 2014. Then the benchmark model will be given data from July to October of 2015 and asked to predict sales counts for November 2015.

The benchmark model generated predictions that received a Kaggle score of 1.25081 (RMSE).

III. Methodology

Data Preprocessing

As mentioned earlier, I want to create a training data file with a structure of *X*, *y* where *y* is a monthly sales count so that a supervised learning regression algorithm can make predictions that are monthly sales counts. I started by loading the provided daily sales records file into a pandas dataframe and used the groupby function from the pandas library to aggregate the daily records for each *date_block_num / shop_id / item_id* combination. The new file has only these fields:

date_block_num	*shop_id*	*item_id*	*item_cnt_day*

where the *item_cnt_day* field contains the monthly sales count for each *date_block_num*, *shop_id*, and *item_id* grouping. I renamed the *item_cnt_day* field to *t* to indicate that this is the sales count for the base month. Next, I created a function to help me append sales counts from other months to each record in the training file. The exact steps I took are documented in the *data_preparation.ipynb* Jupyter notebook. With the use of this created function, I appended 5 fields to the training file so that it now has these fields:

date_block_num	*shop_id*	*item_id*	*t*	*t-1*	*t-2*	*t-5*	*t-11*	*t+1*

The *t-1* field is the sales count for the same shop and item but where the date block number is one less than the value in that field in the current record. In other words, it represents the sales count for the month prior to the base month for the respective shop and item. Similarly, the *t-2* field is the sales count from

2 months ago. The *t-5* and *t-11* are sales counts from 5 and 11 months ago. However, these counts are 6 months and a year ago from the month for the target variable. The *t+1* field is the sales count for the month following the base month.

For the next step, I augmented the training data with features related to the categories of the items. Using the provided *items.csv* file, I merged in the item_category_id field for each item. I had hoped to use a field called "category type" instead of using just the id but as indicated earlier, I was not able to find good defining clusters for the different categories.

To attempt to add some more useful data to the training data, I decided to use monthly sales counts of the categories at the different stores. Perhaps the model could use that data instead to identify categories that are hot sellers at some shops but are stable sellers at other shops.

Utilizing the *item_category_id* field, I grouped the monthly sales count by category and shop and stored the totals in another dataframe. Using these derived totals, I appended monthly sales counts for the respective item category in each record for the base month, for the prior month and for a year ago. The training file, with the *item_category_id* field renamed to *categ_id*, now has these additional fields:

...	*categ_id*	*categ_t*	*categ_t-1*	*categ_t-11*	...

Continuing on, I aggregate the monthly sales counts for shop and category to arrive at monthly sales counts for each shop. Following the same formula for items and categories, I added in sales counts from the base month and for prior months for the respective shop in each record in the training file. Upon completion, the training file has 3 more fields:

...	*shop_t*	*shop_t-1*	*shop_t-11*	...

Next, I replaced the date_block_num field with 2 separate fields; one is for the year and the month of the corresponding date_block_num value. These 2 fields should help the model to learn some of the cyclical sales patterns better than a date block number.

This now gave me all the features and the target variable that I wanted in the training file. However, there were invalid records in this file. Any record for the year of 2013 had an incorrect value in the *t-12* field. As such, I removed these records early on in the data preparation process. As well, each record with a year of 2015 and a month value of 9 (corresponding to a date block number with the max value of 33) could have a wrong value in the *t+1* field. These records were removed from the training file as well.

There are also 397 records where the *t+1* field had a negative value. I decided to leave these records in the training file so I can experiment with training the model with by either including or excluding these records.

Implementation

With a suitable training file, I went about evaluating potential algorithms that I could use to train a model. As indicated earlier, I tested regression algorithms that are publicly available Python packages. For this evaluation phase, I decided to use a small subset of the following features: *year*, *month*, *shop_id*, *item_category_id*, *item_id* and *t*. Using a 80/20 split of the training records into a training set and a validation set, I initialized each regressor with just its default parameters and evaluated its performance on both sets. The function to evaluate the performance is the following simple function:

```python
from sklearn.metrics import mean_squared_error

#calculates the root mean squared error
def calc_RMSE(actuals, predictions):
    return np.sqrt(mean_squared_error(actuals, predictions))
```

The training and validate RMSE for each regressor can be seen in the Jupyter notebook called *regressor_evaluation.ipynb* and are duplicated in the table below:

Python Package	**Regressor**	**Training RMSE**	**Validation RMSE**
sklearn.linear_model	LinearRegression	6.65214	6.39019
sklearn.tree	DecisionTreeRegressor	0.0000	6.06272
sklearn.ensemble	AdaBoostRegressor	4.92357	6.28409
sklearn.ensemble	BaggingRegressor	2.63199	5.43849
sklearn.ensemble	ExtraTreesRegressor	0.00000	5.52825
sklearn.ensemble	GradientBoostingRegressor	4.70939	5.65826
sklearn.ensemble	RandomForestRegressor	2.62890	5.41731
xgboost	xgb.XGBRegressor	4.78971	5.64767
lightgbm.sklearn	LGBMRegressor	4.40905	5.19356

I decided to use LGBMRegressor (Light GBM) because it had the lowest validation score. Admittedly, this is a not a very objective manner to pick a winner but there were other considerations that went into this decision. I had read that a number of Kaggle competition winners had used XGBoost and I read Light GBM gave comparable results but took less time to train with. This was important because I was completing my project using my desktop CPU.

Moving forward after choosing Light GBM, I used grid search from the scikit-learn library to help me find optimal hyperparameters. I focused on these parameters: max_depth, num_leaves, n_estimators and learning_rate. Soon, I realized that the optimal learning rate (learning_rate) was 0.1 and that changing the number of weak trees (n_estimators) from a value of 150 did not change the RMSE so I focused on adjusting just the maximum depth of the tree (max_depth) and the minimum number of leaves (num_leaves).

Because using grid search took a very long time, I started resorting to finding optimal hyperparametres by manually tweaking the hyperparameters and training a model with early stopping rounds (with the number of rounds set to 5). To avoid overfitting, I experimented manually with the reg_lambda parameter and came to the conclusion that a value of 0.0001 was always optimal.

With each trained model, I generated the testing data file based on the provided test file for the Kaggle compeition. This allowed me to generate the predictions required for the competition. As indicated by the rules of the competition, the true predictions were clipped to the range of 0 to 20 so I adjusted the predictions accordingly. I merged these final predictions with the ID fields from the provided test file to create a properly formatted submission file. The code that performs the steps described above are in the Jupyter notebook called *training.ipynb*.

After receiving a RMSE score from Kaggle, I proceeded to vary the set of features that I would use to train a model. I also experimented with including and excluding training data records with a negative value for the target variable. I took these actions in hopes of finding a solution that gave a lower score from Kaggle.

Refinement

Without changing the regressor or adding new features, it seemed that my options for improving the accuracy of the predictions were to continue to tweak the hyperparameters or to train the model with a better subset of features among all those available in the training file I created.

My strategy was to initially pick a small subset of features. After finding good hyperparameters for that subset, training the model with what I believe were optimal hyperparameters and using the model to submit predictions to get a RMSE score, I would repeat the whole process with a subset with more features.

The initial subset only had 6 features. Next, I added more features for the sales counts of items for months prior to the base month. After that, I added features related to the shop and then expanded this set of features with data related to the item category.

I found that the score did not get consistentlyt get better as more features got added. For the last subset that I trained with, I went back to the best performing subset so far and took away what I thought was not an useful feature (the *tm5* field). Training with this last subset gave the best Kaggle score.

Next, I experimented with excluding records with negative values for the target variable from the training data. I repeated the earlier rocess of going from a small subset of features to larger subsets but this time with all records with negative targets removed.

The training sets and their contents, along with their training and validation scores are given in the table below.

Training Set Name	**Features in Set**	**Training RMSE**	**Validation RMSE**
A	*year*, *month*, *shop_id*, *categ_id*, *item_id*, *t*	4.21059	5.01942
B	*year*, *month*, *shop_id*, *categ_id*, *t*, *t-1*, *t-2*, *t-5*, *t-11*	3.84225	4.89484
C	*year*, *month*, *shop_t*, *shop_t-1*, *shop_t-11*, *categ_id*, *t*, *t-1*, *t-2*, *t-5*, *t-11*	3.72874	4.80330
D	*shop_t*, *shop_t-1*, *shop_t-11*, *categ_t*, *categ_t-1*, *categ_t-11*, *item_id*, *t*, *t-1*, *t-2*, *t-5*, *t-11*	3.96213	5.12538
E	*shop_id*, *categ_id*, *t*, *t-1*, *t-2*, *t-11*	4.08296	5.01376
A’	(same as A but with records with negative values for the target variable removed)	4.70118	3.65165
B’	(same as B but with records with negative values for the target variable removed)	4.23762	3.73627
C’	(same as C but with records with negative values for the target variable removed)	4.25635	3.74826
D’	(same as D but with records with negative values for the target variable removed)	4.12965	3.64634

E′	(same as E but with records with negative values for the target variable removed)	4.43170	3.69191

It should be noted that the training and validation RMSE scores did not correlate with the received Kaggle score. As well, the training RMSE scores are lower when the records with negative targets were included in the training data. However, the validation RMSE scores are lower when these records are excluded.

IV. Results

Model Evaluation and Validation

The RMSE scores from Kaggle for each of the different feature sets that I used to train a model are given below.

Training Set Name	**Hyperparameters** (*n_estimators*: 150, *learning_rate*: 0.1, *reg_lambda*: 0.0001)		**Kaggle Score**
	max_depth	*num_leaves*	
A	75	1250	1.11473
B	75	1250	1.02806
C	80	1500	1.03606
D	80	1500	1.09144
E	75	1250	1.02798
A′	75	1250	1.08309
B′	75	1250	1.02998
C′	80	1500	1.06994
D′	80	1500	1.06488
E′	**75**	**1250**	**1.02736**

The model that predicted best was trained with training set E′. The hyperparameters for this model should be optimal because moving up and down from their values eventually gave a worse validation RMSE score. The values for the hyperparameters are the same as the ones for the previously best performing subset (E) so it gives me a bit confidence in this regard as well.

The final model seems like an adequate solution as the currently leading solution for the competition has a RMSE score of 0.82205. It seems to generalize well as the score came from predictions on unseen data. As well, it appears to be somewhat robust. The same model trained with records with negative targets received a RMSE score that is just 0.00062 more.

In terms of whether results from the model can be trusted, I feel that they cannot be. However, my reasons are more subjective than substantive. It starts with the difficulty in validating the results.

One difficulty in verifying the results is that there are shops and items that are in the provided test file but are not in the training data. For these pairings, I could not compare the training data with their predictions to determine whether the predictions are reasonable.

As well, I am unsure about whether the provided data is factual or just generated for the purpose of this competition. The total sales count for some of the shops (i.e. -1) and the counts for the date_block_num seem to indicate that some parts of the data was changed to add more of a challenge for the participants.

Additionally, the organizers clipped the true targets (predictions) to be in the range of from 0 to 20. During the training process, I did not confine the targets and saw RMSE scores above 3. The mean for the monthly counts by shop and item was just 1.24 so being 3 off the true target does not inspire much confidence.

Finally, the best score I achieved for the competition is actually 1.02570. However, this score came for predictions that were generated with incorrect labeled data.

Justification

My final solution received a Kaggle score of 1.02736. Currently, this is good for position 547 (out of 1418 submissions) on the public leaderboard of the competition. It is lower than the score of 1.25011 received by the benchmark model which is currently in position 1003 on the leaderboard. This means that my final solution is about 17.8% more accurate than the benchmark model.

Given the approach that I used to solve the problem, I do not think I can further tune the model but I do feel that I might be able to achieve an ever slightly lower score by going through more permutations of different features to use to train the model. However, to make a significant improvement in the score, it appears that I would have to tackle the problem with a different approach.

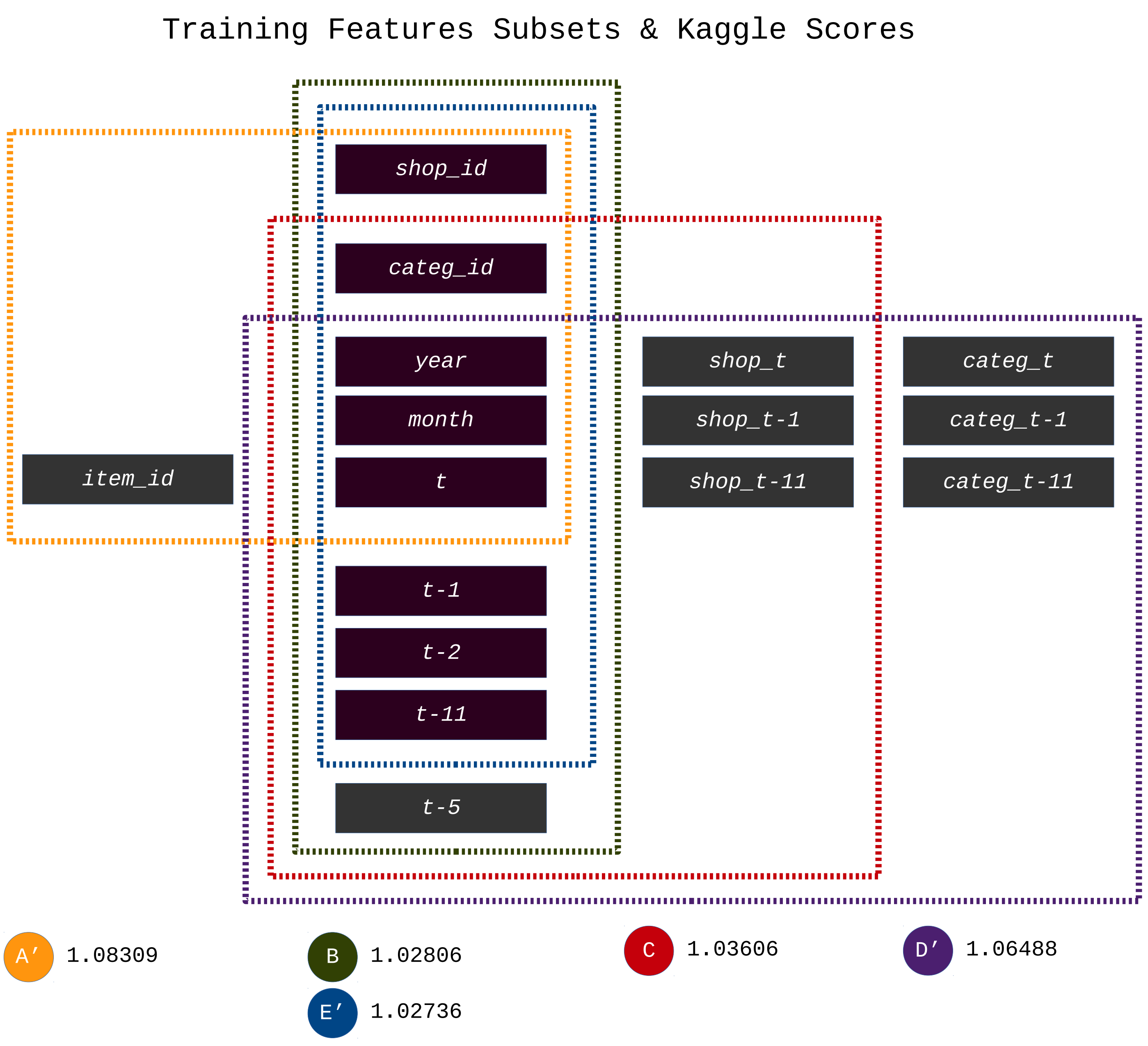
The Kaggle score for my final solution suggests that it is significant enough to have solved the problem. A RMSE score of slightly over 1 means that on average, the predictions were off by 1. Of course, the rules for the competition make the the problem different from the task of sale forecasting in the real world.

The competition povided a limited amount of training data to start off with. For a real world problem, I would use features about the shops such as where it is located, its hours of operations, and the population of its neighbourhood. For categories, I would also provide labels that might help a model identify the sales cycle of its items. For example, I would use labels for the “seasonality” (i.e. winter, summer) of categories to differentiate between sales patterns for snow clearing equipment and for patio furniture.

The competition also clipped the true targets. Real life is not as forgiving. Therefore, the solution is signifcant enough to solve the problem but only within the context of the competition.

V. Conclusion

Free-Form Visualization



The diagram above shows the Kaggle scores for different subsets of features used to train the model to make the predictions. It illustrates an important aspect that I learned from tackling this type of problem with the approach that I chose. What I learned was that using the right input is an important consideration.

For each pair of training set (i.e. A and A′), the diagram showed the set that had the best score. Therefore, A′, D′ and E′ did better than A, D, E respectively. Likewise, B and C outperformed B′ and C′ respectively. There was not a clear cut decision for when to include or exclude the training records with negative targets.

As well, in general, a smaller subset outperformed larger ones. The subset of features that earned the best score (E’) had just 8 features. The diagram shows that adding more features to this set (as done with training sets B, C, and D’) made the accuracy of the predictions worse. However, training set A’ has just 6 features but it did relatively worse than the other subsets.

From these observations, I realized that understanding the data is just as important as knowing how to apply algorithms to the data. Being able to choose a choose a “good” set of features and records requires a good working knowledge of the domain of the task and also of the data.

Reflection

This is the first machine learning project that I had to complete without the benefit of starting with a provided framework. It was an extremely valuable exercise to go through because it required me to truly understand all the steps involved in solving the problem.

I first had to come up with an approach which was to convert the problem into a supervised learning regression problem. Next, I had to analyze the data and preprocess it to create a suitable training file for a model that could do one-step forecasting. Then I needed to pick and tune a regressor to train the model. After evaluating the performance of the trained model, I had to scour my brain for ideas to improve the performance.

While the experience of taking a project from idea to solution was great, it was stressful at times. My skills in Python are limited and there was a lot of backtracking. For example, I initally added the sales counts for prior months using the apply function of a pandas dataframe and this method took hours. Once I learned how to use the merge function, the time to accomplish the same thing reduced to minutes. The time pressure made each backtrack doubly frustrating.

As well, after over 20 submissions to Kaggle, I realized that I had mislabeled some of the features for the data that I was using to generate the predictions. I had set the year field to 2015 (which is correct) and the month field to 10. The target month is 10 so the label for the input data should have been 9.

I did not enjoy seeing improvements in the score I received from Kaggle. Overall, my final solution exceeeded my expectation. I had not expected that it would receive a Kaggle score that would be over 10% better than the benchmark model.

Improvement

Judging from the score for the current leader of the Kaggel competition, it is clear that better solutions exist. If I had more time, I would have like to have experimented with the same approach but with other pieces of data, such as the price field. For the sake of time, I completely ignored this field. I briefly looked at the values for this field and decided that it remained the same for an overwhelmingly majority of the items so I dropped it from consideration for the set of available features.

As well, I would like to experiment with the same approach but using another algorithm called CatBoost. I stuck with Light GBM because it was released to the public earlier than CatBoost and pressed for time, I thought there would be more information available for how to tune the hyperparameters for Light GBM than for CatBoost. I did read that CatBoost was more accurate for certain tasks.

To have a solution with a substantial improvement, as I mentioned earlier, I would most likely need to rethink my approach. I do not really know what other approaches to attempt but I am curious about using a LSTM network to tackle this challenge.

VI. References

Kaggle. “Predict Future Sales.” https://www.kaggle.com/c/competitive-data-science-predict-future-sales

Brownlee, Jason. Machine Learning Mastery. “Time Series Forecasting as Supervised Learning.” https://machinelearningmastery.com/time-series-forecasting-supervised-learning/

TrietChau. Kaggle. “A beginner guide for sale data prediction.” https://www.kaggle.com/minhtriet/a-beginner-guide-for-sale-data-prediction

“Getting Started with XGBoost” https://xgboost.readthedocs.io/en/latest/get_started.html

ML Blog Team. Microsoft. “Lessons Learned From Benchmarking Fast Machine Learning Algorithms.” https://blogs.technet.microsoft.com/machinelearning/2017/07/25/lessons-learned-benchmarking-fast-machine-learning-algorithms/

Mandot, Pushkar. “What is LightGBM, How to implement it? How to fine tune the parameters?” https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

Swalin, Alvira. “CatBoost vs. Light GBM v. XGBoost” https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Bontempi, G., Le Borgne, Y. and Ben Taieb S. “Machine Learning Strategies for Time Series Forecasting.” https://www.researchgate.net/publication/236941795_Machine_Learning_Strategies_for_Time_Series_Forecasting

Makridakis, S., Spiliotis E., and Assimakopoulos V. "Statistical and Machine Learning forecasting methods: Concerns and ways forward." https://doi.org/10.1371/journal.pone.0194889

Barnwal, Manish Kumar. "Random Forests explained intuitively" https://www.datasciencecentral.com/profiles/blogs/random-forests-explained-intuitively

KDnuggets. "XGBoost, a Top Machine Learning Method on Kaggle, Explained" https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html

keitakurita. Machine Learning Explained. "LightGBM and XGBoost Explained" http://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/