Prediction of *Future Sales*

STAT5302 Kaggle Assignment

Members:

Harshit Jain <jain0149@umn.edu>
Zachary Levonian <levon003@umn.edu>
Fatemeh Nosrat <nosra002@umn.edu>
Shriya Rai <rai00016@umn.edu>
Siwen Wang <wang7610@umn.edu>

Guided by:

Prof. Jie Ding
Yiqing Shen

1. Abstract

We completed data exploration for data, ANOVA table to select the best feature and made 10 submissions to predict the future sales using time series data for the Kaggle competition [3]. The models we tried include 2 linear models, 3 gradient boost models, a deep learning model, and 2 random forest models. The best submission was Gradient Boost Model with item_id and shop_id as predictors with Kaggle score of 1.49034. We merged items_categories with sales data using item ID. We have made our code available on Github.

2. Background/Problem Statement:

The company we are looking at is "one of the largest Russian software companies" - 1C Company. Given the historical sales data from January 2013 to October 2015, we want to construct a prediction model that best forecasts the sales for the products and shops for November 2015 [3]. In other words, our suggested model not only fits well with the existing training data set, but also effectively fits with the test data set. In the following sections, we will discuss our preliminary study of the data, the selection of features, and the selection of model approaches.

3. Preliminary Data Study

The data was given to us in the following files:

- The training set (sales_train.csv)
- The test set (test.csv)

- items.csv - supplemental information about the items/products.
- item_categories.csv - supplemental information about the items categories.
- shops.csv- supplemental information about the shops.

The training dataset has daily historical data from January 2013 to October 2015. The test dataset is used to test our models and forecast the dales for shops and items for November 2015. The "item_cnt_month" variable is our response for the prediction while all other variables are evaluated for use as predictors.

3.1 Data Dictionary

Variable	Definition	Categorical/ Continuous Variable
ID	an Id that represents a (Shop, Item) tuple within the test set	Categorical
shop_id	Unique identifier of a shop	Categorical
item_id	Unique identifier of a product	Categorical
item_category_id	Unique identifier of item category	Categorical
item_cnt_day	number of products sold	Continuous
item_price	current price of an item	Continuous
date	date in format dd/mm/yyyy	Categorical
date_block_num	consecutive month number	Categorical
item_name	name of item	n/a

shop_name	name of shop	n/a
item_category_name	name of item category	n/a

** Categorical variables are treated as factors

3.2 Exploring the data

This dataset is significantly more complex than any other we have discussed in class. In particular, the data describes both item and shop entities, with the vast majority of shop/item pairs not appearing in the data. And while the desired output is a month-level sales prediction, all of the provided data provides sales info at the day level. Below, we walk through aspects of the data with some guiding visualizations, all of which are also available in the provided repository.

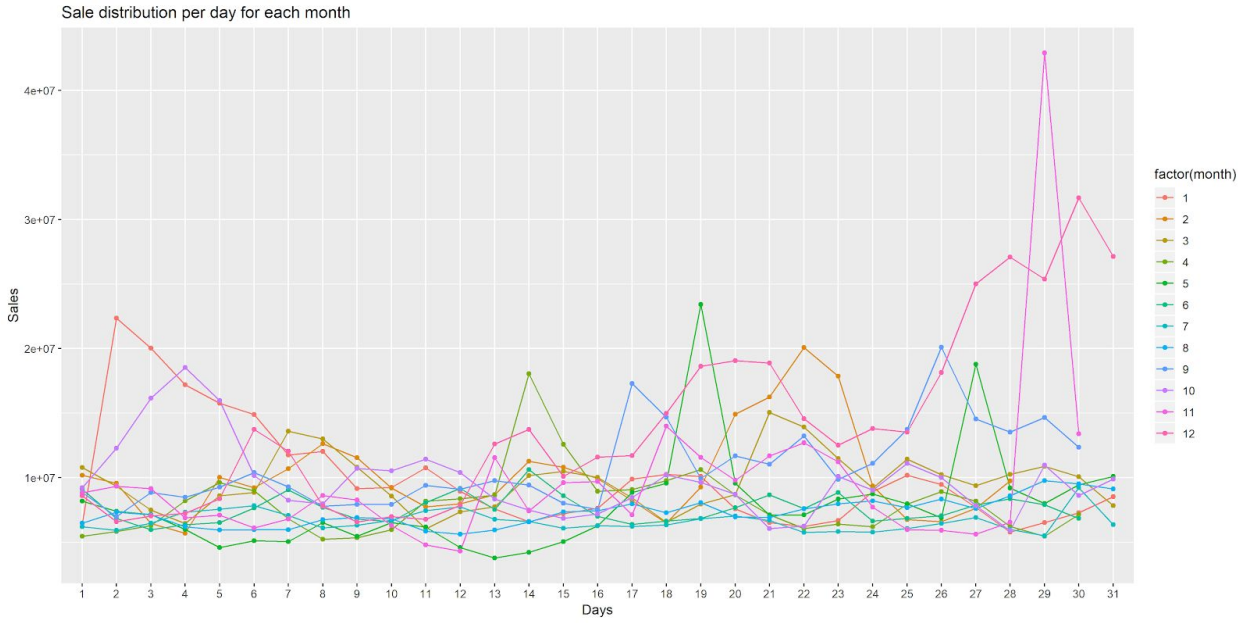


Figure : Distribution of Total Sales for each day of every month

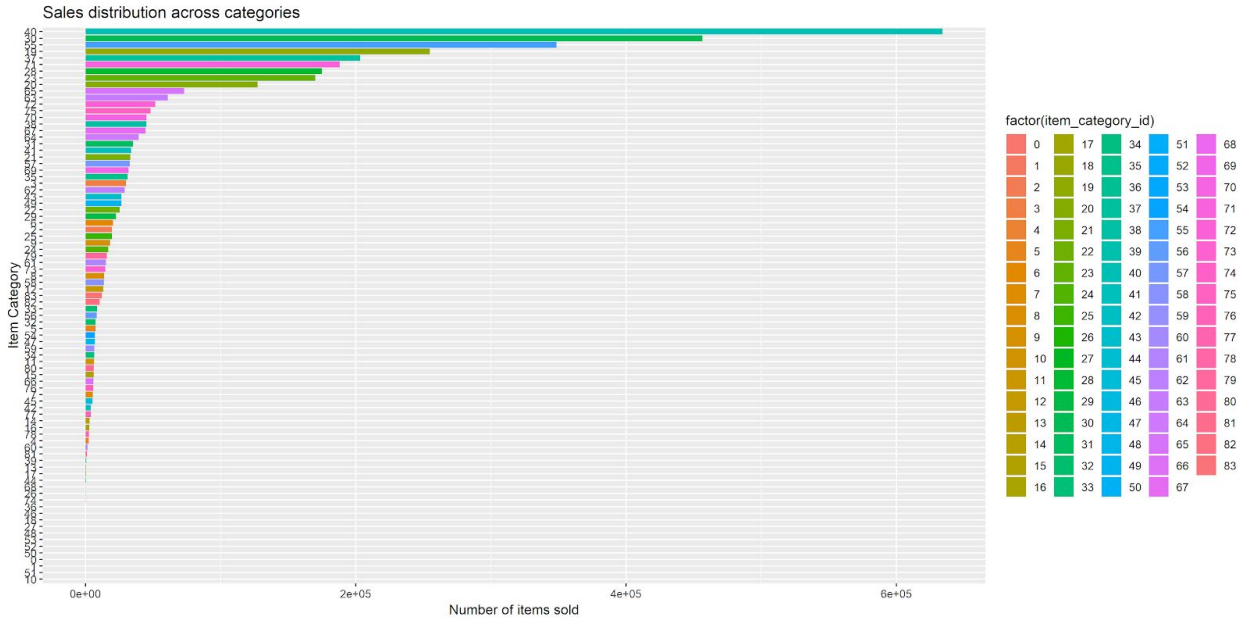
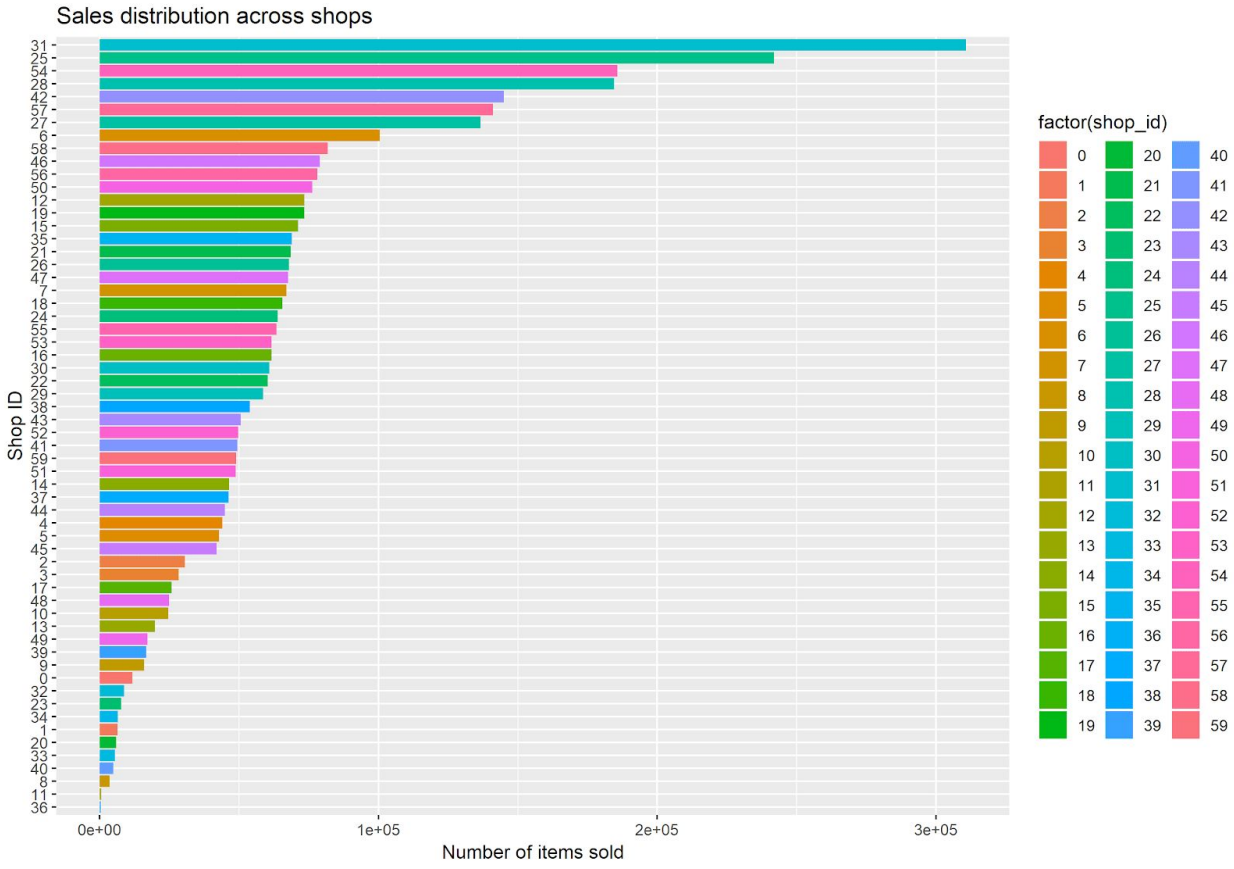


Figure : Number of items sold for each category. Some category seem to have zero sales.

Figure : Number of items sold by each shop



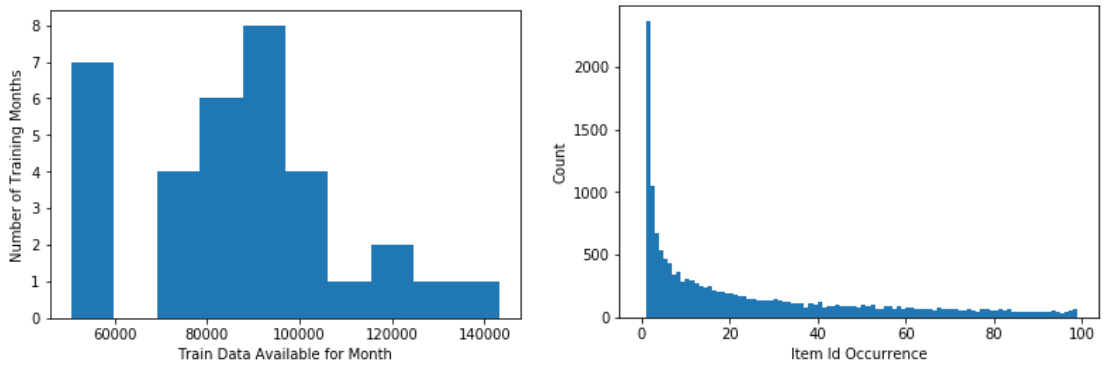
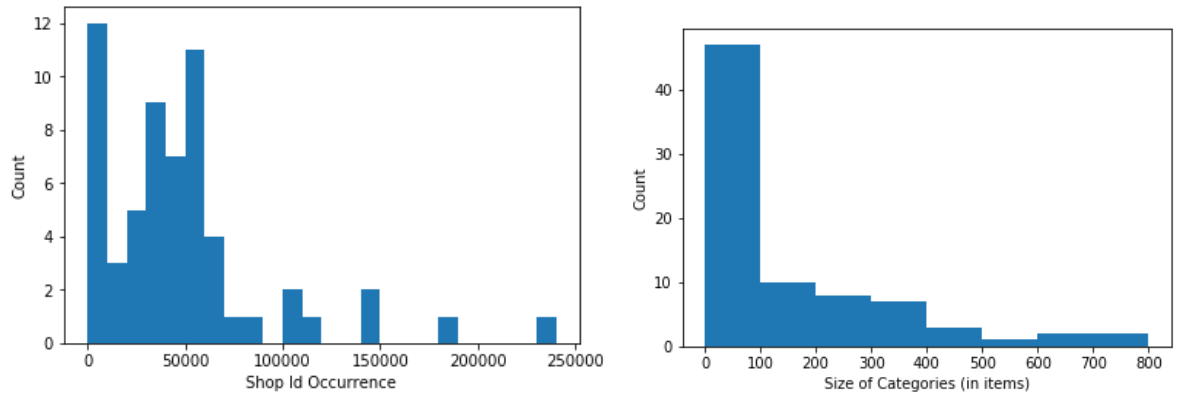


Figure: On the left, we visualize the training data across months. The amount of data available per month varies significantly. This variation may reflect natural variation in monthly sales or a bias in the sampling process. On the right, we observe that the majority of items appear in the dataset only once or twice; only a small percentage of items appears more often. Only 21.7% of the item ids in the training data also appear in the test data.



The vast majority of item categories contains fewer than 100 items, but the majority of items (50.2%) are contained in the five largest item categories.

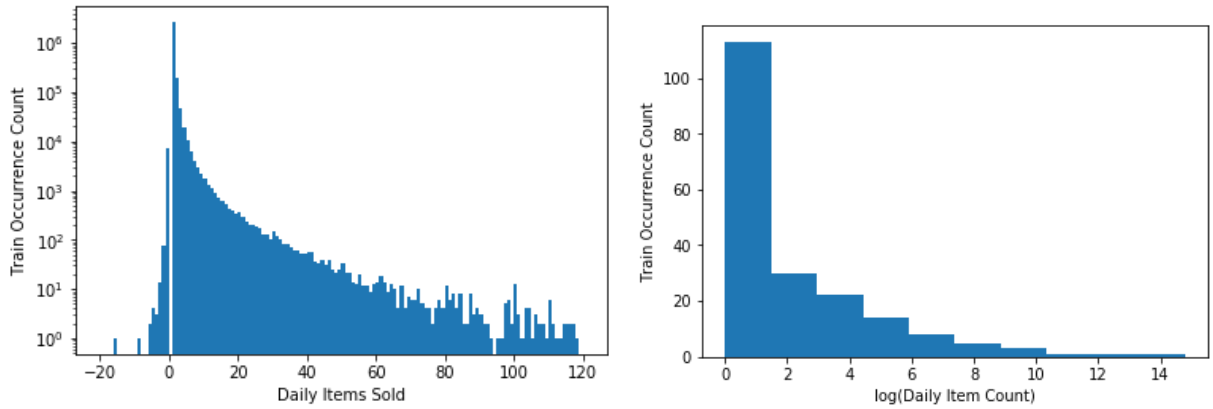


Figure: On the left, the distribution of item_cnt_day. Note that zeros are not included in the dataset, and that a significant number of negative values occur. It is not clear if these values represent data errors or bear some other meaning, e.g. cancelled orders. On the right, the counts of the item_cnt_day; the range of item_cnt_day spans multiple orders of magnitude.

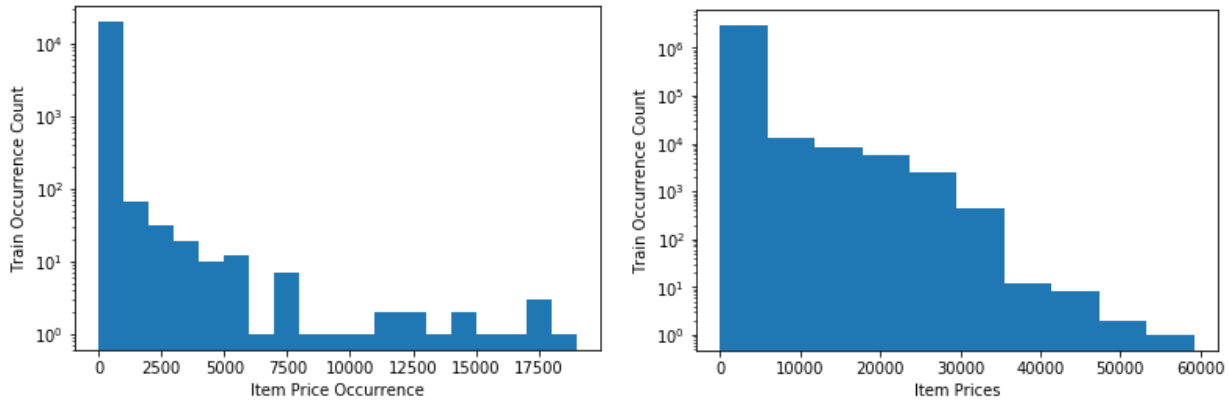
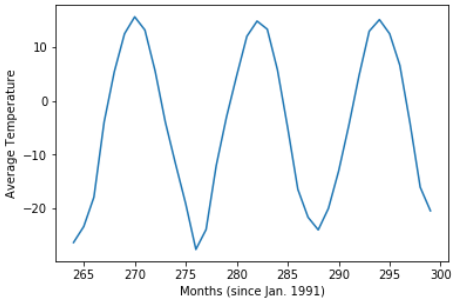


Figure: On the left, the counts of unique item prices; most item price levels occur only once, but some appear thousands of times. On the right, the distribution of item prices shows that most prices are < 10000. An outlier price (> 100000) has been omitted.

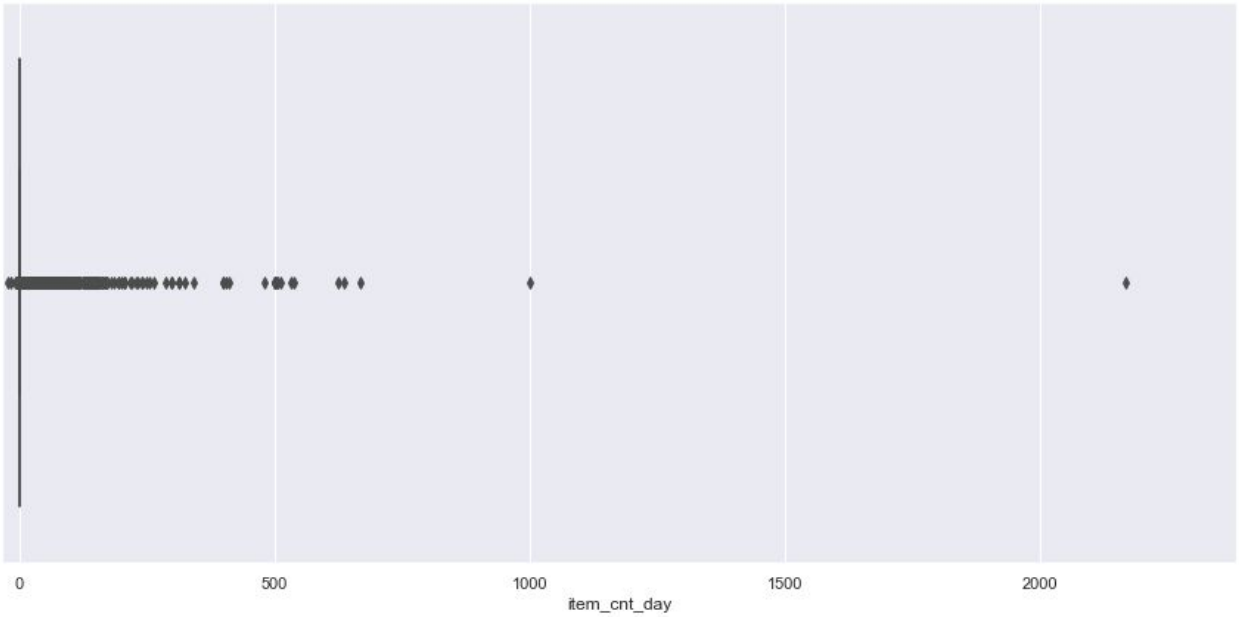
Weather Data



Because sales are affected by the weather [8], it seems reasonable that we could increase our ability to predict sales counts by incorporating weather data into our model. We pull data from the Worldbank on the average monthly temperature in Russia, where the tech company 1C is based [9]. For the item_cnt_month models described below, we use this weather data as a continuous variable that may help predict shop's sales of particular items. The inclusion of this data is largely a proof-of-concept for merging in and using external data to improve results; we do not have enough information in this competition about the type of sales to have any intuition if weather data is actually useful.

Outliers

The item_price is negative in one case, which isn't right and we replaced it by the median of item_price. Also the value of "item_cnt_day" is very high for one case. We also removed that value as well. We can see from the boxplot that the value of item_cnt_day is negative and there is a possible outlier whose value is greater than 2000.



4. Data Leakages

We don't want to train the data which will not be present at future. Because the data we are dealing with is time series data, there are high chances of Data leakages.

Therefore, we remove the extra data from the training data which is not present in testing data. We do that over the ID fields, "shop_id" and "item_id". Therefore we only use that data that appear on the test set.

5. Feature Engineering:

Rolling window based features

Because we are dealing with time series data, it becomes obvious to use sliding window for our features. We can calculate summary statistics across the values in the sliding window and include these as features in our dataset. We have used the sliding window of 3 months over the summary statistics of min, max, mean and std and created new columns for the item_cnt_min, item_cnt_max, item_cnt_mean and item_cnt_std.

Lag Based features

Lag features are the classical way that time series forecasting problems are transformed into supervised learning problems. The simplest approach is to predict the value at the next time (t+1)

given the value at the previous time (t-1). We use this approach again for "item_cnt" with individual shifts of 1, 2 and 3 which gives us three new features item_cnt_shifted_1, item_cnt_shifted_2 and item_cnt_shifted_3.

6. Selected Model:

Model Description	Kaggle Score	Section
Gradient Boost Model with item_id and shop_id as predictors	1.49034	6.1
Linear model with item_category_id as predictor	1.52507	7.4
Linear model with item_id and shop_id as predictors	1.55635	7.4
Random Forest with multiple predictors	2.14936	7.6
item_cnt_month fast.ai Deep Learning model	3.15374	7.1
item_cnt_month Gradient Boosting model	5.65900	7.2
Gradient Boost Model with item_category_id	4.83507	7.5
item_cnt_month Random Forest model (Overfit)	12.54525	7.3

Table: Models discussed in the subsequent sections.

6.1 Best Model: Naive Kaggle baseline

None of our models performed better than the naive baseline provided by Kaggle, which scores 1.23646 RMSE while predicting 0.5 for all monthly counts. In general, we suspect this may be due to (1) fundamental lack of understanding of the provided data and (2) handling of "missing" data. Information provided about the provided data is minimal, and we have no sense of the data completeness. We only have entries for items with non-zero sales, and we chose not to treat "missing" items as having zero sales at that store for that day/month. It seems likely that this was a costly omission, as evidence suggests that the true test result should contain many zeros. Additional discussion in section 7.3.

6.2 Selected Model : Gradient Boost Model with item_id and shop_id

We modeled gradient boosting regressor model in R using gbm package. The regressor treated all variables as continuous, and included the items and the shops. The model generates 5000 trees and the shrinkage parameter lambda=0.01which is sort of **learning rate**. The interaction depth d is 3 which is the total splits we want to do. So here each tree is a small tree with only 3 splits. The score we got on kaggle for this model was 1.49034.

7. Alternative Models

7.1 item_cnt_month fast.ai Deep Learning model

We trained a deep learning model based on the fast.ai library and deep learning tutorial [4]. The Jupyter Notebook that produced this model can be viewed easily in nbviewer [2] and in the GitHub repository associated with this submission. This model was the last of three models (discussed subsequently) that we trained to predict item_cnt_month directly, rather than to predict at the daily level. To produce ground truth values for item_cnt_month from the provided training data, we aggregate all shop/item combinations by month, taking item_cnt_month as the summation of an item's item_cnt_day in that month. A key concern with this approach is that we have no knowledge about the completeness of the historical sales data; any missing data could result in a significant bias towards over or under-estimating monthly sales counts at the item or store level. Nevertheless, without more information about the data, this approach seems reasonable.

To train the model, we treat month, year, item id, shop id, and item category id as categorical variables, and the average monthly temperature as a continuous variable. After transformation, we produce a training set of 1,609,124 month/shop/item entries. Discussion of the exact model specification is unnecessary for this document, but the model consists of five primary layers: an embedding layer for the categorical features, three linear layers (two with batch normalization), and a final linear layer to produce the regression output. This model seems to have avoided overfitting: while the validation error is 5.16, the Kaggle test error is an appropriately lower 3.15. The figure below shows the models outputs alongside the true values for the validation set; generally, the model has learned to predict the distribution of the true values.

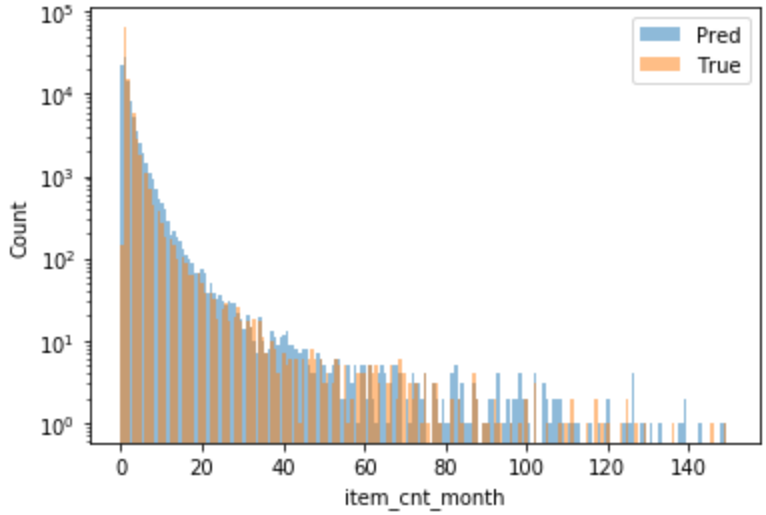


Figure: Distribution of true item_cnt_month values alongside the predicted values.

7.2 item_cnt_month Gradient Boosting model

We trained a gradient boosting regressor model in Python. The regressor treated all variables as continuous, and included the items, the shops, the item categories, the month, and the year. The gradient boosting model makes decisions about how many variable splits to make in an iteratively constructed tree. All hyperparameters for the model can be seen in [2]. We conducted a parameter search over the maximum depth of the tree (see Table below), finding that a higher depth resulted in a higher validation accuracy. (The final three months of the training data [batch ids 31-33] were held out as a validation set.) However, we can see that the validation loss is much higher than the training loss which is an indicator that this model is overfit. The test RMSE (from Kaggle) for this model was 12.56, which is in line with the validation RMSE. We also trained a full version of the model with all of the training data, but this model actually performed worse on the Kaggle test set, scoring 7.94 RMSE.

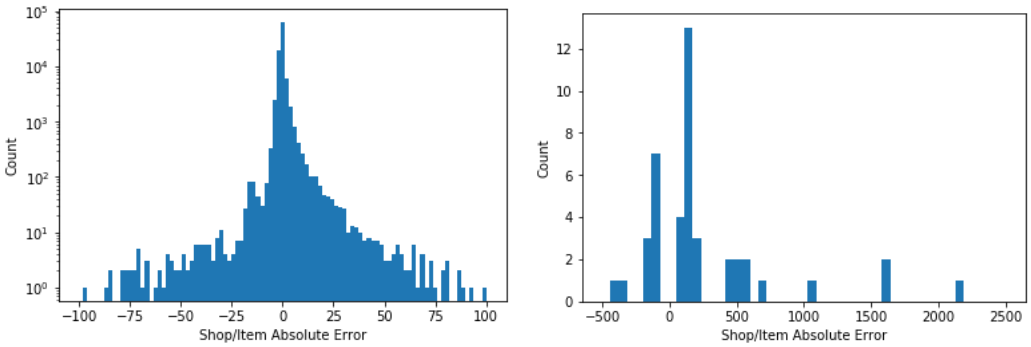
Max Depth	Training RMSE	Validation RMSE
1	8.10	13.60
2	7.09	13.25
3	6.79	13.13
4	6.18	12.87
5	5.66	12.66

Table: Training and Validation RMSE for different values of the max_depth hyperparameter.

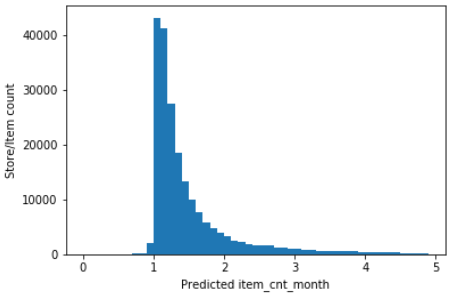
7.3 item_cnt_month Random Forest model

We trained a random forest regressor model in Python. The regressor treated all variables as continuous, and included the items, the shops, the item categories, the month, and the year. All hyperparameters for the model can be seen in [2]. Our RMSE on the training and validation sets indicate that this model also severely overfit to the training data. While the training RMSE was 3.13, the validation and test RMSE was 12.68 and 12.55 respectively.

We conducted an error analysis on the Random Forest's outputs to understand why this model was performing so much worse than the simple Kaggle baseline. Errors are shown in the figure below.



On the left, we visualize the absolute errors (the residuals) for each shop/item combo in the validation set. While these errors are generally centered around zero, the larger number of positive values indicate that this model is systematically underestimating the true sales count. On the right, we visualize only the "big errors": predictions for which the true value and the prediction differ by more than 100. For five shop/item combos, we underestimate the true number of monthly sales by more than 1000!



By looking at the distribution of predictions, we can see that nearly all of our predictions are 1 or more. While we don't have access to the test data's true values, it seems likely that we have introduced significant bias into the model by training only with non-zero sales counts. As the test data asks us to test on a much larger set of shop/item combinations than are reflected in the training data, it seems likely that the majority of the true values for the test set are zeros. This could explain why the RMSE for this and our other models is so much higher than the naive Kaggle baseline.

7.4 Main Effects Model for Different Variables

We used a regular multiple linear regression to explore some of the variables usefulness. The table below describes how adding main effect of different variables affect the RSS and whether there is any significant improvement in RSS with help of ANOVA analysis.

Model	**Variable Selected**	**RSS**	**RSS Improved (Anova table comparison) (Yes/No)**
M0	Only Intercept	20134908	
M1	Added item_id to M0	20129326	Yes
M2	Added item_category_id to M0	19136309	Yes
M3	Added Shop_Id to M0	20134357	Yes
M4	Added day to M0	20131792	Yes
M5	Added weekday to M0	20131513	Yes
M6	Added month to M0	20120007	Yes
M7	Added item_price to M0	20132384	Yes
M8	Added weekdays+month to M2	20116528	No
M9	Added item_id+shop_id to M0	20128667	Yes

**Anova comparison is performed with the original model and the new model with added regressor to the original model unless specified otherwise

M2 model that includes main effects for just item_category_id has the lowest RSS(191363 November 09) and has Multiple R-squared: 0.3984.

Further analysis was also done on another linear model that was M9 (predictors - item_id,shop_id) which had RSS(20128667) and has Multiple R-squared: 0.001059.
This analysis was done to provide contrast to two Gradient boosting models done with predictors as item_category_id and item_id,shop_id respectively. The Gradient boosting model for the latter performed better when compared to the first one. This was interesting as linear model with item_category_id had performed much better than the linear model with item_id,shop_id.

7.5 Gradient Boost Model with item_category_id

We modeled gradient boosting regressor model in R using gbm package. The regressor item_category_id treated as factor .The model generates 2000 trees and the shrinkage parameter lambda=0.01. The interaction depth d is 3. The score we got on kaggle for this model was 4.83507.

7.6 Random Forest Regressor with multiple features

We used random forest regressor for one of our model where we mostly used the derived features that uses the time series based feature extraction techniques like rolling window, lag time features and others. We used the number of estimators to be equal to 50, max depth =7 and random state=0. We got the training rmse of 1.9239633447826523 and Validation rmse of 1.5043044575066464. The features we used for this are: 'shop_id', 'item_id', 'item_cnt', 'transactions', 'year','item_cnt_mean', 'item_cnt_std', 'item_cnt_shifted1', 'item_trend' and 'mean_item_cnt'. We can see that except for the shop_id, item_id and item_cnt, others are the derived features.

7.7 Transformation:

Use library(plyr) to use join function in order to combined data in test and sales_train by item_id in combined_data. Then, library(lubricate) is used to work on the data based on the dates (year, month, day). Use aggregate function to obtain the total items by month. Use library(data.table) for creating a data table (as.data.table is used to combine the data.) shop items in months is summarized in summerized_month/-shop_item.For avoiding negative value for log transformation, 2 is added to the data, but it will be subtracted at the end. Sales_model is the log transformation of the model.[log(item_cnt_month) ~ date_block_num + month + shop_id + item_category_id, data = summarized_month_shop_item). November is not considered in the test data, so we add 11 in order to assign November to the prediction, too. Assign 34 to the date_block_num for November 2015. Again combine data in test and items by item_id to get combined_test. Fit sales_model to prediction.test by used combined_test. Subtract 2 from the prediction.test to reverse the addition. The log transformation of the response is used to fit the model because it is a better model to understand the behavior of the data.

8. Conclusion

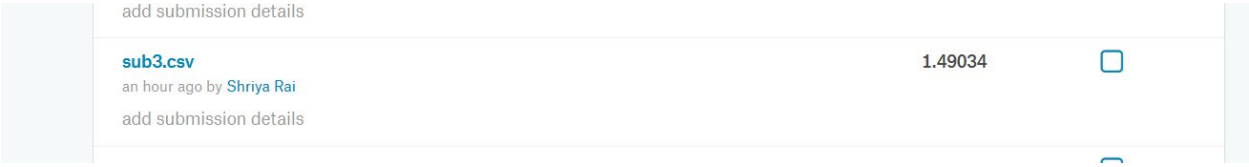
We had the opportunity to try many different models and to experiment with a variety of additional features. However, none of our models performed better than the naive Kaggle baseline. As discussed, this is likely due to a misunderstanding about the nature of the raw data,

as we generally overestimate the true values in our validation set. Nevertheless, we enjoyed developing models for this challenging problem.

References

[1]https://nbviewer.jupyter.org/github/levon003/kaggle-predict-future-sales/blob/master/code/python/data-exploration.ipynb
[2]https://nbviewer.jupyter.org/github/levon003/kaggle-predict-future-sales/blob/master/code/python/fastai-prediction.ipynb
[3] https://www.kaggle.com/c/competitive-data-science-predict-future-sales
[4] https://course.fast.ai/lessons/lesson3.html
[5]https://medium.com/making-sense-of-data/time-series-next-value-prediction-using-regression-over-a-rolling-window-228f0acae363
[6]https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/
[7]https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html
[8]https://www.adwordsrobot.com/en/blog/how-weather-influences-product-sales
[9]http://sdwebx.worldbank.org/climateportal/index.cfm?page=downscaled_data_download&menu=historical

Screenshot of Score



Code

All code and derived data for this project is available in this Github repository:
https://github.com/levon003/kaggle-predict-future-sales

Code is in both R, Python, and Bash, contained in the code/r and code/python and code/bash directories respectively.

A quick description of the primary code files follows:
-code/python/data-exploration.ipynb, viewable at [1] - Contains visualizations of the data and other number crunching
-code/python/fastai-prediction.ipynb, viewable at [2] - Contains the code for the three item_cnt_month models, including the fast.ai Deep Learning model.
-code/bash/gzip_file.sh - Utility for gzipping CSV files for submission to Kaggle
-code/r/DataExploration.Rmd - Unused
- code/r/Prediction_Linear_and_GBM - Anova Table Analysis, Linear models, GBM with item_id,shop_id,GBM with item_category_id
- code/r/PredictSalesVisualisationExploration - Contains visualizations of the data