CSCI 567 - DefinitelySemiPositive

William Choi
Student
University of Southern California
Houston, TX 77098
wschoi@usc.edu

Ahmed Fayed
Student
University of Southern California
Los Angeles, CA 90007
fayed@usc.edu

Alex Winger
Student
University of Southern California
Los Angeles, CA 90007
winger@usc.edu

Abstract

The team's goal was to predict, as accurately as possible, the number of items sold for a list of various items for a future month. The data-set used was obtained from the Kaggle competition Predict Future Sales to train the models used in this report. From the onset, the aim was to attempt implementing multiple models and possibly combine the individual results (ensemble), more methodically, at a later stage. To this end, the team has tackled **four** main model architectures. Namely, simple linear regression, traditional artificial neural-networks, long short-term memory neural networks and XGBoost. The prime prediction performance was obtained using XGBoost with a final score of 0.88253.

1 Introduction

Performing analysis on semi-ordered data containing sales record of various items sold at different electronic shops in Russia, the task was to predict future sales of a list of items at each of the shops for November.

Naturally, plotting the data was the obvious first step to help the team understand what is available to work with. Various ways and perspectives were used to look for trends. The second thought, was to translate the data since it was provided for Russian shops and items from Russian to English. The premise was to see if legibly understanding the data would give additional insight into how best to model. Once the translations were done, it became apparent that English item and shop names could be fashioned into a feature set.

Originally set out with the rough idea of trying different models and combining them more methodically for a final result. The team initiated the implementation of a simple linear regression model with roughly refined 'basic' features. Once a baseline model was functioning, albeit low-scoring, the trajectory proceeded towards developing a traditional artificial neural-network type model, a long short-term memory network, and an XGBoost model in parallel. For this next phase more extensive feature engineering was utilized.

In quick summary, the prime performing model was handed to XGBoost with a final score of 0.88253.

2 Previous work

The first programming assignment of the $CSCI$ 567 $Machine\ Learning$ class, in particular the linear regression portion, was the starting point. Attempting to extract and fit the project training data for a linear regression model provided the team with a solid grasp on the capability and functionality of Python libraries such as Pandas and Numpy, to name a few. The team members also familiarized themselves with navigating through the existing example Kaggle notebooks generously made publicly available by fellow competitors.

There were many Kaggle notebooks to explore, but many were repeats of the top scoring ones. Therefore, the team narrowed down the number of notebooks referenced. The notebooks by Larionov [1] and Yakovlev [2] provided the majority of ideas implemented here, ranging from the pre-processing of the data to how to create the XGBoost model. The notebook by Leclercq [3] was also referenced for ideas on how to organize the data-set and fabricate additional features. In addition to using Kaggle notebooks, blog post and article Agarwal [4] and Wai [5] were tapped for hyper-parameter tuning.

3 Data

Initially, the data-set was given in **six** ".*csv*" files: *"item_categories.csv", "items.csv", "sales_train.csv", "sample_submission.csv", "shops.csv" and "test.csv"*. Therefore, one of the main goals in the data processing phase was to come up with a single Pandas DataFrame representing training features and another one representing training labels allowing for easy integration with various machine learning models.

First, the $Date$ data provided for the item sales was in a format that could not be used for training, so it was converted and broken down into day, month and year columns. The original Russian text did not give much intuition on how to analyse the data-set, so it was translated using the *mtranslate* package. Also, for quicker manipulation of the data-set (and computation), the DataFrames were downcasted to reduce RAM memory consumption.

For linear regression **five** input features were used, namely, *"Month", "Item_Cat", "Shop_ID", "Item_ID",* and *"Item_Price"*. A mean category price was calculated per item category and used if an item price was NAN (not known from previous training data). Pandas was then used for manipulations to get the sold *"item_cnt_month"* (Item Count/month) for each item. Similar to unknown *"Item Prices"*, *"item_cnt_month"* was used as the mean item count of that item's category. The target variable was *"item_cnt_month"*.

Once the initial linear regression model was complete, the team performed additional feature engineering for a more robust data-set for use in the XGBoost and neural-network models. To achieve this goal, the team first cleaned the data by dropping outliers, such as items with greater than $1,000$ sales, a price greater than \$$10,000$, and a negative price. For items with negative sales, the team had **three** approaches: deleting these items, doing nothing, and setting the item sales to 0. Setting the item sales to 0 gave the lowest validation set root mean-squared error (RMSE). Shops 0 and 57, 1 and 58, 10 and 11 appear to be the same shop but for different time periods. Shop 40 seems to be an "island" shop of shop 39, so the similar shops were combined to have the same *"Shop_ID"*. From the *"Shop_Name"*, the city each shop was located in was extracted, as well as the type of shop. *"Category_Name"* was also split into a *"Category_type"* and *"Category_Subtype"*. For the items, a *"Name2"* and *"Name3"* were extracted from the *"Item_Name"*. This extra information (e.x. the category information) showed a relation between contrasting items. With the added potential the new features provided, **one** large DataFrame was put together which has rows of each and every (shop, item) combination in each month of sales data, as well as the rows of the test set (which had the *"Month_ID"* set to 34). This new DataFrame was crucial for the next step in processing.

Furthermore, more features were derived by combining many of the current features, such as calculating the average item price per month for each item. From here lag variables were created to use as additional features when making predictions. The team attempted various lag value combinations but determined that using lags of 1, 2, and 3 yielded the best results. Finally, adding some $Date$ features such as the month name each *"Month_ID"* represents and how many days are in that month. By dropping the first few months' worth of data plethora of 0s these rows contained from

the lag variables set was accounted for.
The data-set generated by the processing up to this point was then used to train the XGBoost model and the neural-network models. Attempting to train a traditional neural network with this data-set found submission score of ~1.0 (RMSE), which was quite higher than the ~0.88 (RMSE) achieved by the XGBoost submissions. In order to improve the score, many methods including one-hot encoding the categorical data, introducing longer lag and more were considered, but were quickly walled by memory limitations, especially implementing one-hot encoding of the *item_ids*. Thus the approach was abandoned and the same data-set was used.

4 Experiments

4.1 Learning Algorithms and Models Attempted

4.1.1 Linear Regression *(similar to in-class assignment #1)*

Linear regression was the simplest and first model implemented. This model was attempted before extensive feature engineering was performed on the data, and thus the training features consisted of only five features: *"Month", "Item_Cat", "Shop_ID", "Item_ID", and "Item_Price"*. The provided data-set was split into 80% Training and 20% Validation randomly. The standard SGD algorithm was used to minimize the mean-squared-error (MSE). Python Generator functions were mandated since the data was too huge for $numpy$ to perform matrix multiplication on in one go. Clever techniques were implemented to tie things together. A similar method to the class assignment for tuning to derive the optimal lambda was used. The basic feature set described above was used combined in a single big DataFrame as discussed earlier with English translations.

4.1.2 Neural Network *(using Keras w/ Tensorflow backend)*

The traditional neural-network made use of the data-set generated by the extensive feature engineering in the pre-processing stage with the following features:
'month_id', 'shop_id', 'item_id', 'item_quantity', 'item_seniority', 'shop_type_code', 'city_code', 'item_category_id', 'name2', 'name3', 'category_subtype_code', 'category_type_code', 'monthly_avg_item_quantity_lag_1', 'shop_subtype_monthly_avg_item_quantity_lag_1', 'city_monthly_avg_item_quantity_lag_1', 'item_city_monthly_avg_item_quantity_lag_1', 'delta_price_lag', 'delta_revenue_lag_1', 'month', 'days', 'item_shop_first_sale', 'item_first_sale'.

This extensive data-set was used for XGBoost as well so the features will only be stated here once.

Lagged features (**three** months each) were used namely: *'item_quantity', 'item_monthly_avg_item_quantity', 'shop_monthly_avg_item_quantity', 'shop_item_monthly_avg_item_quantity'* for a total of 34 features.

Neural-networks were good candidates for modelling this data-set, due to the large number of training examples, features and the non-linearity of the trends observed while analyzing the data. Many distinct network configurations were attempted. Combinations of these activation functions: *linear, exponential, hard_sigmoid, sigmoid, tanh, relu, softsign, softplus, softmax, elu*, were tried. Along with varying the depth of the network *(number of layers)*, number of nodes per layer, number of epochs the network was trained over with the data-set, batch-size *(2000, 20000, 70000)*, and different optimizers: *(Adam, Adadelta, Adagard, RMSprop)*. Deep neural-networks were also considered with up to **seven** hidden layers each including distinct or similar activation functions from the set above. The final configuration of a single hidden layer with $1,024$ nodes with bias and a *tanh* as the activation function led to the (traditional neural-network's) prime results. Also facilitated by a *Glorot uniform initializer* for the weight initializations, a *softplus* activation function for the the output layer, as well as an *adam* optimizer. The final prime number of epochs applied on training was 3 with a batch-size of $2,000$.

4.1.3 LSTM Neural Network *(using Keras w/ Tensorflow backend)*

Another neural network seriously considered was *Long Short-Term Memory Neural Networks*, because of the temporal nature of the problem at hand. LSTM required a bit more manipulation of data on top of the data used for traditional neural networks and XGBoost that, unfortunately, led to excessive memory usage that crashed the Kaggle notebook. Since LSTM's recurrent nature meant that it would keep a history of previous inputs, the lagged features were dropped. However, one-hot encoding recommended for neural networks for categories exceeded the amount of physical memory, especially when encoding the more than $21,000$ *item_ids*.

4.1.4 XGBoost

Two different XGBoost approaches provided the best results. XGBoost is a gradient boosting algorithm which focuses on execution speed and model performance. Gradient boosting creates new models to predict the errors of previous models and then combines them to get the end result. It uses the gradient descent algorithm to minimize the loss for new models. For both approaches, the parameters used for the models were the maximum depth of the tree, the minimum sum of instance weights needed in a child, the step size shrinkage, the sub-sample ratio of the training instance, the sub-sample ratio of columns for each tree, the tree method, and the number of gradient boosted trees. To tune the hyper-parameters, the Hyperopt library in Python was applied to set a range of values to tune and then returned the set of parameters which minimized RMSE on the validation set.
For the first approach, the entire validation set during the hyper-parameter tuning found the **ten** best parameter sets for the models. This actually resulted in **ten** slightly different models, which then made predictions on the entire test set using each one and took the average of the results. For the second approach, it was realized that the entries in the test set could be split into **three** separate categories: items with no prior sales in any store, items with no prior sales in the shop but have been sold in other shops before, and items which have been sold in the shop before. It was decided to split the validation and test sets according to this grouping, creating **three** separate XGBoost models for each group. Following the same steps as above to create each model except only using items in the particular group for the validation and test sets. For example, if a model for the items that had been sold in the store before was desired, only the item/shop pair where the item had been sold in the store before would be used for the validation and test set. This new validation set allowed tuning the hyper-parameters and then making predictions on the new test set. Once all predictions from each model were available the predictions were combined into one for the final predictions. This approach was only slightly worse than the first approach and possible explanations for this phenomena will be discussed in the Results section (5) below.

4.2 Evaluation

4.2.1 Linear Regression and Neural Networks

Data corresponding to Month 33 was set as the validation set *(corresponding to Oct 2015)* since it was the latest available data and the test month was set as Month 34 which, naturally, was the prediction goal. No cross-validation was used. The target variable or evaluation metric was the RMSE score returned through Kaggle submission.

4.2.2 XGBoost

For all of the XGBoost models, the large DataFrame, discussed in the Data section (3) above, was split into a training set, validation set, and testing set. The training set consisted of all the entries with *"Month_ID"* less than 33. The validation set was all the entries with *"Month_ID"* equal to 33 *(corresponding to October 2015)* and the testing set was all of the entries with *"Month_ID"* equal to 34 *(corresponding to November 2015).* The target variable using the validation set to train the model and tuning the hyper-parameters was the root-mean-squared-error (RMSE) for predicting the validation set.

4.3 Setup

Languages:
Python

Packages:
hyperopt, itertools, pdb, sklearn, numpy, pandas, math, xgboost, time, pickle, re, keras and matplotlib,

random, datetime, and mtranslate.

Note: Linear Regression model availed regular CPU-training, a GPU-Accelerated (using a Nvidia GTX 1050Ti CUDA-enabled graphics card) Docker container running Tensorflow-gpu 1.15 was used for training the Keras Neural-Networks, and for the XGBoost model, use of the "gpu_hist" option for the tree_method parameter allowed GPU-Accelerated capabilities.
GPU-Acceleration is desirable drastically reducing the computation time to train the model and perform hyper-parameter tuning.

5 Results

5.1 Linear Regression

The prime Linear Regression submission scored an RMSE of 1.35 and was the first submission.

5.2 Neural Networks

The top performing Neural-Network submission scored a RMSE of 0.98 with the help of feature lagging, 34 distinct input features, 1-layer (not counting input/output layers) and a near-optimal (under these conditions) number of nodes in that singular-layer network of $1,024$. Increasing the depth of the neural-net was not beneficial. It would decrease prediction score against the validation which the team constructed from Month 33 *(corresponding to Oct 2015)*. A couple of layers deep and a phenomenon commonly known as the *"Vanishing Gradient Problem"* arises. The specific depth would change depending on the activation function used per layer respectively, but with numerous training runs the *"elu"* activation function was found to the best performing on the single-layer architecture along with a *"softplus"* activation function on the output layer. Researching articles online proved that the *"Adam"* optimizer was one of the best to use and was confirmed to converge the error to minimums quicker and faster. Naturally a MSE was used for the optimizer target since this is the closest available to the RMSE Kaggle scores submissions with.
For LSTMs, many notebooks reported a score around 1.01 using basic features. Since the XGBoost model was already scoring around 0.88, there needed to be something truly distinguishing about new LSTM models to produce results usable in an ensemble in conjunction with the XGBoost model. Thus, the idea was to train with a bigger feature set, which led to excessive memory usage and ultimately no viable LSTM model.

5.3 XGBoost

In the models section above, we discussed two different approaches which used XGBoost. For the first approach, results showed a RMSE of 0.88253 and for the second approach, a RMSE of 0.88273. It is conjectured that the second approach had about the same RMSE due to the use of the same feature set for each model and therefore resulting in **three** models which when combined are highly similar to the single model in the first approach. In other words, final predictions were essentially implementing their own form of boosting by taking the **three** *"weaker"* models to result in the same predictions as the stronger model.

When training the model during the first approach, the achieved validation RMSE of ~$.875$ was especially similar to that of the test RMSE. This indicates that there is no over-fitting in the model. It is also conjectured that there is some difference between the validation RMSE and the test RMSE due to the fact that the validation RMSE is calculated using the entire validation set, whereas the test RMSE is calculated only on 35% of the test set. Thus, there is a possibility that the test RMSE might be lower than what Kaggle is currently reporting. For the second approach, the validation RMSE varied from model to model, with the model for the brand new items resulting in a RMSE ~1.6, the model for the items new to a specific store resulting in a RMSE of ~0.6 and the model for the items sold in the store before resulting in a RMSE of ~0.85. These values indicate that the RMSE for the entire validation set is consistent with that of the first approach because of how the items are separated (~10% as *brand new*, 40% as *new to that shop* and 50% as *sold in that shop before*).

6 Conclusion

Learning how to pre-process/clean a data-set and how that is, arguably, one of the most important steps in achieving a good predictor was a definite surprise. Moreover, implementing Linear Regression, Time-Series, Neural-Networks w/ Keras, and Xgboost on real-world data-sets provided practical experience and familiarity with widely-used libraries and workflows in the Machine Learning community. The lagging of columns/features in the data-set led to great boosts in prediction performance. This makes sense since this problem is temporal in nature.

As the results in this report show, more feature engineering leads to a significantly better feature set for improving the final score achieved by each model. In addition to working on better feature engineering, more insight into which features are best for each model was crucial. For instance, the model with the *brand new items* used all of the fabricated feature set to train the model, but there were features like the *item_cnt* lags which will always be 0 for all of these items. This will influence the results and therefore give a larger RMSE. Final and best for last, creating an ensemble of different models seems incredibly promising. As seen in the presentations, fellow teams from the class were able to find great success by creating an ensemble of XGBoost and LGBM.

References

[1] Denis Larionov. Feature Engineering, xgboost. Online, 2018. URL `https://www.kaggle.com/dlarionov/feature-engineering-xgboost`.

[2] Konstantin Yakovlev. 1st place solution - Part 1 - "Hands on Data". Online, 2019. URL `https://www.kaggle.com/kyakovlev/1st-place-solution-part-1-hands-on-data`.

[3] Tristan Leclercq. Predict future sales - full solution xgboost. Online, 2020. URL `https://www.kaggle.com/tristanleclercq/predict-future-sales-full-solution-xgboost`.

[4] Rahul Agarwal. Using xgboost for time series prediction tasks. Online, 2017. URL `https://mlwhiz.com/blog/2017/12/26/win_a_data_science_competition/`.

[5] Wai. An example of hyperparameter optimization on xgboost, lightgbm and catboost using hyperopt. Online, 2019. URL `https://towardsdatascience.com/an-example-of-hyperparameter-optimization-on-xgboost-lightgbm-and-catboost-using-hyperopt-12bc41a271e`.