Kaggle Caterpillar Competition Write-up

Team Paf! (tlorieul & Axel de Romblay)

Competition presentation

Try to predict the unitary price of tubes: regression problem.
The evaluation score is RMLSE.
Train / test split based on “random” sampling of tube_assembly_id (rather nearly one out of two sampling than random).
More details: https://www.kaggle.com/c/caterpillar-tube-pricing

Features engineering

We spent around 70% of our time on this which restricted what we were able to do on the model learning part. This part consisted mainly on encoding properly the different categorical features, adding new features, trying some features selection, visualizing features' importance and above all trying to make sense to what we have done.

About the target:

- log transformation: most of regression algorithms optimize the RMSE or the MSE. Thus, we have to apply a log transform on the target (x -> log(x+1)).
- we tried other transformations like power transforms (such as x -> x^1/16 as suggested in the forum), etc. and blended the models but it was not useful.

Data given as a relational database.

- orders features:
  - supplier: dummified (much better than a simple one-hot-encoder)
  - tube_assembly_id (kept the associated number: “TA-1624” -> 1624)
  - quote_date: kept year, day (per year) and day_of_week
  - annual_usage, min_order_quantity, bracket_pricing, quantity, diff_quantity = quantity-min_order_quantity
- tubes features:
  - material_id: not dummified (kept the associated number)
  - diameter, wall, length, num_bends, bend_radius

    - merged end_a, end_x with tube_end_form + added diff_end (end_a == end_x)
    - number of specs
- components features:
    - bill_of_materials merged with components: component_id, component_type_id, weights of components, quantities (4 per component thus 4*8 features)
    - we added: total weight, total quantity of components and number of different components. These features were very valuables.

What did not work:

- dummifying day_of_week, months
- volume, density, etc...
- removing tube_assembly_id: the order of the ids is important !!
- keeping all the specs
- orientation features, pitch features, form features (and other particular component features)

Each time we wanted to test a new feature, we trusted our score on a correct CV (generally with 2 folds made on tube_assembly_id) + score on LB

What we could have done to improve our score after:

- adding mean, min and max of components' weights
- counts of similar tube_assembly_id (+per supplier etc…)

*Finally, we ended it up with 131 features. Our train and test dataframes were very sparse (dummified supplier + 4*8 features on components with 0 as encoded missing values).*

Models tuning

Approximately 30% of the work. We used the hyperopt package. First we launched the optimization process on computer with a lot of cores that we had available before limiting the search space and using a regular computer. The objective function given to hyperopt used cross_val_predict before manually computing the error instead of directly calling cross_val_score. Indeed, the latter computes the scores of every folds which did not contain the same number of samples and thus a weighted average was needed to compensate this but it is incompatible with RMLSE (because of the square-root).

The problem we faced:

- Lack of data : there were not enough data to build an accurate CV. One solution was to increase the number of folds by slicing on tube assembly id (for instance: TA-0001 and TA-0002 in the train set and TA-0003 in the validation set). But it does not respect rigorously the train/test split. Another solution was to predict in a “cross_val_predict” way: by doing this we can get a more accurate prediction but we do not have any information on the standard deviation between folds. We kept this approach but also kept in mind that we would need some bagged/ensembled models to decrease overfit.

Single models:

- linear models, knn Regressor, etc...
- random forest : gives poor results
- extra trees : gives better results but still not enough good compared with XGBoost
- AdaBoosted decision trees (+ idem bagging)
- “bagged” XGBoost (more like an average of 15 tuned XGBoost with different random seeds). Gives pretty good score (on LB: **0.2260** with a single tuned XGBoost and **0.2160** with 15 averaged XGBoost)

We also tried to build a model using an explicit relationship between prediction and quantity as: **unitary_price = unitary_cost + fixed_cost / quantity (*)**
This requires to fit approximately 5000 regressions to get unitary_cost and fixed_cost on “identical” training data (same features except quantity). We fitted regression only where we had at least 2 points (we tried 3 and more but did not improve the model) and where min_order_quantity == 0 as we could not completely rely on bracket_pricing as was revealed in the forum. In the end, we kept about 25000 rows and 5000 when dropping duplicates. Then, we fitted 2 tuned “averaged” XGBoost on targets unitary_cost and fixed_cost and predicted it on the test set also only where min_order_quantity ==0 and where we have at least 2 “identical” data (also about 25000 samples). Finally we use our formula (*).

The CV score was worst. But when blended (0.80/0.20) with the “normal” method (ie without using the explicit formula), it gives better results. Thus we blended the two different approaches on the 25000 test samples (min_order_quantity == 0 and at least 2 “identical” test data) and on the 7000 others, we kept the “normal” approach.

We got **0.2145** on LB.

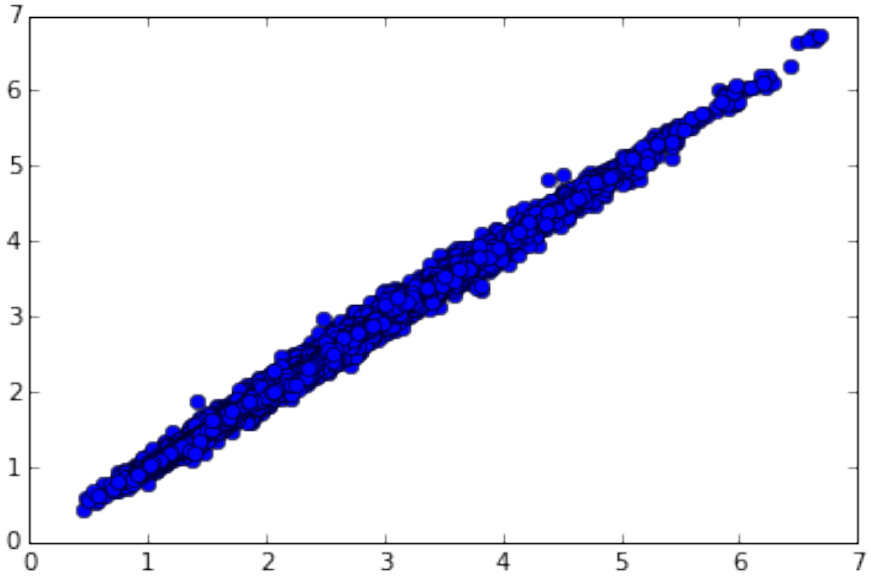
Ensembled models:

- First stacked model:
    - 15% of 15 AdaBoosted XGBoost
    - 10% of 15 BaggingRegressor with base estimator = XGBoost
    - 75% of different tuned XGBoosts : 3 different max_depth.

- Second stacked model:
    - layer 1: RF + Adaboost (loss=linear) + Adaboost (loss=square) + XGBoost + ET
    - layer 2: linear regression

    - note: learning done using 3 folds to learn a “generalizable” linear regression (mean of the 3) before relearning the first layer on all the data

- ‘blind’ weighted averages on the submitted files: each time we submitted, we plotted the predicted prices of our best submission (on Y-axis) and of the submitted predictions (on X-axis). We also computed pearson correlation coefficient. If the 2 csv files were “enough” decorrelated and gave “good” results, we blended and gained.



above : log-prices of the best predictions and the submitted predictions

Conclusion

Finally, we ended the competition with 108 submissions and **ranked 15th on the private LB with 0.208742** (and 16th on the public LB with 0.214021). We wish we could merge with a team to compare approaches and rank top 10 but we did not. It was a great experience and we tried our best ! :)