POLITECNICO
MILANO 1863

Data Mining and Text Mining (UIC 583)
Project Report

Daniel Ballesteros Castilla - 898179 Jaimini Boender - 898177
Filippo Calzavara - 898526 Luca Comoretto - 906086
Giada Confortola - 898540

GitHub Repo: @giadaconfo/data-mining-challenge

June 10, 2018

1 Data Visualization and Preprocessing

Visualizing the data helped us discovering important characteristics of the dataset.
We started by using a heat map to get an understanding of correlation between variables. Looking at it, it became very clear that most of the variables related to weather conditions (as $Mean\_Dew\_PointC$, Max_Dew_PointC and Min_Dew_PointC) were highly correlated. More in general, the ones that had all three max, min and mean values (such as temperature, humidity, sea level pressure) were correlated too.

Overall, we discovered that the variable $NumberOfCustomers$ had a very high correlation with $NumberOfSales$ (91% similarity). This was extremely exciting until we realized that it wasn't provided in the dataset to predict. The idea of predicting $NumberOfCustomers$ and use it to predict the $NumberOfSales$ was discarded: the usage of the same dataset would have brought to a lack of independence of the two results. In addition, seen the high correlation, the prediction of sales would have deeply relied on a prediction, leading to a boost of the error.

Next, we identified the variables that contained NaNs. These variables included: $Events$, $CloudCover$, $Max\_Gust\_SpeedKm\_h$, $Max\_VisibilityKm$, $Mean\_VisibilityKm$ and $Min\_VisibilityKm$. NaN values have different meanings depending on the context in the dataset. Because of this, we decided to implement different imputation policies, based on the type of the variable. In particular, we chose that a NaN in '$Max\_Gust\_SpeedKm\_h$' meant that there was no gust on that particular day, thus we substituted it with 0.

Events was a very interesting variable to understand: it had a list of terms to describe what type of weather was present that day (fog, hail, rain, snow, and thunderstorm), but it had an incredibly high number of NaNs (almost 24%) too. The company clarified that they had to be interpreted as no meteorological event (no rain, no thunderstorm, etc) on that day so we split the Event variable into 5 different binary variables: one for each type of event. It not only allowed us to incorporate the NaN meaning, but also made it easier to use the feature when trying to fit to a model. One-hot encoding was also used for the other categorical variables present in the dataset, $StoreType$ and $AssortmentType$.

The number of NaNs was relatively low within $CloudCover$ (7.8% - 41k rows) and 2% at 11k rows for $VisibilityKm$ (it happened that whenever one of the $VisibilityKm$ variables were NaN, then they were all NaN for that store and day). We opted to average the variables in order to give uniformity to the data and avoid losing any rows.

By looking at the univariate distribution of the variables, we also noticed that $WindDirDegrees$ contained a lot of -1 values in it. Considering that the variable should assume values between 0 and 360 we figured that -1 should be treated as a missing value as well. Seen the extremely low meaningfulness with regard to the variable to predict, we simply decided to drop the column.

In order to have a positive distribution of all the Temperature variables, we re-scaled them from Celsius degrees to Kelvin degrees.

Finally, it was decided that the *Date* variable would have been split up into 3; one for day, month and year. The ideas was that perhaps either the day or month could have an effect on the sales. After that, we had finished the imputation of the data and moved on to choosing a model for the data.

We also tried some preprocessing techniques on the set. The most relevant ones were:

- Normalization of all the columns, using a scaler from sklearn, in order to have variables in the same range.
- Principal Component Analysis: we tried PCA with different number of components kept and on different models.

However, we ended up removing those techniques, since the algorithm was not performing better.

2 Model Selection

After trying some simple regression models such as Linear Regression and Lasso, we decided to focus on more complex tools. We used the Python library Sklearn to get all of our models, except XGBoost regression which has its own library.

For all the models, we first split the data into Train and Test set and performed a 5 fold cross validation on the train set to obtain the best combination of parameters. We kept 20% of the data as a test set even if performing cross validation, to be sure that the model predicted was effective also on data never seen from the model. Since the training was done only on 80% of the entire set, the evaluation is likely to assume values slightly pessimistic compared to the real ones. Nonetheless, the final model, used for the prediction delivered, was built on the entire dataset, thus not influencing the real error of our algorithm.

The assignment of the project was to predict the monthly sales for each store. However, the train dataset provided contained information per single day. For this reason, we decided to build a model that predicts the sales of days separately and sum the results per month in the end. For a comparison between models, we considered the evaluation error given, together with the R2 of the predictions per day and the one of the predictions summed per month. Obviously, the latter resulted to be slightly higher compared to the first one, since summing the predictions over balanced the errors made for the individual days.

2.1 Comparisons

Other than XGB, the technique that better performed on the data was Random Forest. We considered two variants of this ensemble:

- **Random Forest Regressor on the entire Train Set**: this method was the one, together with XGBoost, that performed the best on the data. The lowest error obtained with random forest was by considering a number of tree estimators equal to 30, end the value of the error was 0.04982 on the test set.
- **Personalized Model for each store**: based on the idea that two stores may be really different one from another, we decided to attempt to build a single Random Forest Regressor for each store in the dataset. Although the results obtained were slightly better, compared to the general model, we realized that the latter would be more reliable, since the dataset of the single stores is small. Moreover, a technique like the one described would not provide a model in case of new stores opened in the future.

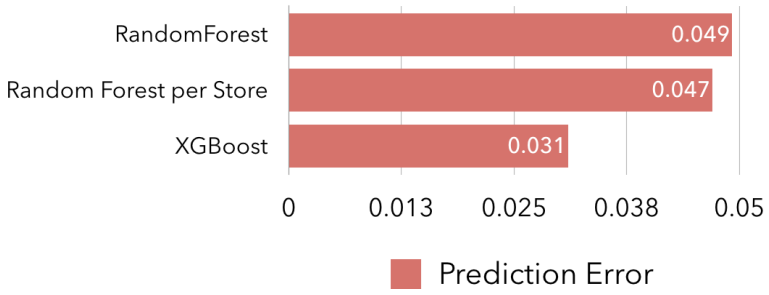

Figure 1: Evaluation Errors on our Best Models

2.2 eXtreme Gradient Boosting

The Extreme Gradient Boosting Regressor performed the best on the dataset. XG Boost implements a scalable gradient boosting for classification or regression trees. This technique clearly outperforms many other methods, winning multiple Kaggle competitions. This method approximates tree learning using quantile sketch. It is an optimized method that could run the tree construction in parallel. The method was used not only for the final performance but also for its execution speed once it was optimized.

2.2.1 Feature Importance

An XGBoost model provides the possibility to check the importance of variables in the dataset, based on the model built. As can be seen in the plot, the most important feature is the StoreID. This means that XGB weights the fact that shop can be different (similarly to what we supposed when we tried to build personalized models per store).
In general, the results reflected what we supposed at the beginning. Data related to the weather are not as important as other information, such as the distance with the nearest competitor and data related to time.

2.2.2 Parameters Tuning

For the Random regression and XGB, the parameters tuning phase was crucial to find the combination that gives the lowest error. To do so, we had to understand the meaning of the parameters that the method takes as input. Those that we considered the most relevant for our problem are the following:

- n_estimators: number of boosted trees to fit
- max_depth: number of tree depth for base learners.

Furthermore, Sklearn provides the GridSearchCV function, which is helpful to try different combinations of the parameters using Cross Validation. However, the function returns the best combination of the parameters based on a standard error measure, such as R2. Since we wanted our results to be optimized based on the error function given by the company, we implemented a personalized grid search function that returns the best combination of parameters according to the given metric. For the XGB case, the pairs considered and their relative error with 5 folds cross validation on the train set were the ones shown in table 1.

As can be seen, the best results were obtained by considering a number of estimators equal to 1000 and a maximum depth of 10. To be sure of the results, we built the model on the entire train set and tested it on the test set we kept in the beginning. In the end, we obtained an error of 0.03180534. The R2 for the prediction of number of Sales on a single day was 0.95718332, while the R2 computed for predictions over month was, as expected, marginally better: 0.99225055.

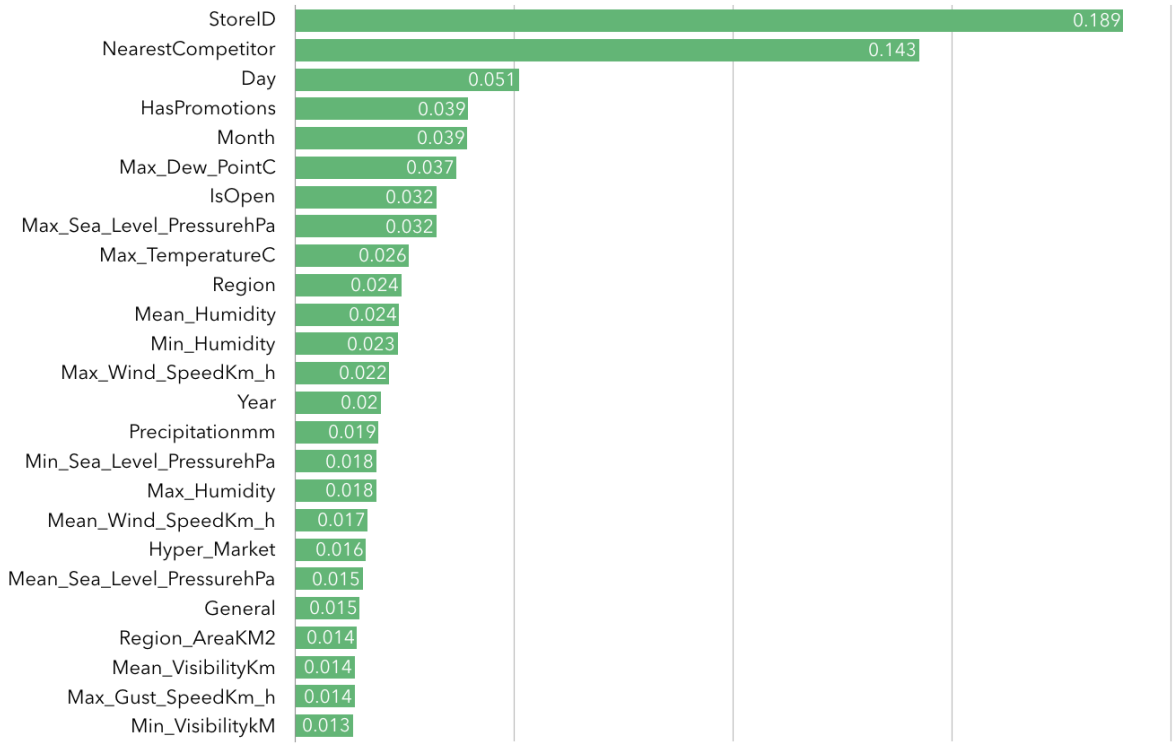

Figure 2: Feature Importance in XGBoost

`n_estimators`	`max_depth`	Error
100	12	0.0517605230850261
300	20	0.0464168866411036
500	10	0.0383303268230849
500	15	0.0398226223984374
500	18	0.0440484633553197
700	20	0.0463971260849973
1000	10	**0.03610537433254**
1000	15	0.0397935821046312

Table 1: Results of the Parameters Tuning of XGBoost with 5-fold cross validation

3 Conclusions

We believe to have achieved a robust prediction model that could be improved given more time and computation power. Thus, we are confident that XGBoost results could be further improved by optimizing the learning parameters used in the model.
[1]

[1]

[1] XGBoost. `https://xgboost.readthedocs.io/en/latest/model.html`

[2] scikit learn docs: `http://scikit-learn.org/stable/`