HackerRank Machine Learning CodeSprint:
Predict Email Opens

Name: Caio Taniguchi
Email: caiotaniguchi@gmail.com
Date: September 4th, 2016

Introduction

The problem to solve in the *HackerRank: Predict Email Opens* challenge is to try to predict whether a HackerRank user will open a target email or not. To model the problem, the competitors are provided a dataset which contains information about the user, the email and the user's reaction to the email (the latter just available on the training data).

Data Exploration and Preprocessing

- Preliminary exploration

The data provided is split in two datasets, one containing the training data and the other containing the test that that we are trying to predict. The test set contains ~200k cases and 48 features, while the training set contains ~490k cases and 6 additional features, including the target variable.

One fact that immediately stands out is that most features are some sort of a user's action counts, with different timeframes, ranging from 1-day to 365-days. Two problems that could appear (and were later confirmed) are high correlation between predictors, since we are counting the same actions in different timeframes and the actions themselves might be correlated, and lack of data variability in the 1-day features.

After a quick inspection, it was found that 12 of the 48 features had its majority class correspond to more than 99% of the cases in the test set. This led to them being discarded from the final model. The discarded features are listed in table 1.

contest_login_count_1_days	contest_participation_count_1_days
ipn_read_1_days	ipn_count_1_days
submissions_count_contest_1_days	submissions_count_1_days
submissions_count_master_1_days	ipn_read_7_days
contest_login_count_7_days	contest_participation_count_7_days
mail_type	forum_expert_count

Table 1. Removed features due to lack of variability

Regarding the correlation issue, inspecting a correlation matrix plot shows that many of the count features have at least one feature that it is highly correlated with. This seemed a good spot to try some dimensionality reduction with principal component analysis (PCA), which I explore next.

- **Principal Component Analysis**

I begin the PCA exploration with a cumulative sum of the explained variance by number of principal components, which shows a smoother curve than I anticipated. Generally, I would cut the number of components at around 13, which would retain approximately 95% of the explained variance. However, since the competition evaluates only the F1 score, I decide to keep 17 components, retaining about 99% of the explained variance. At least that was what I thought in the first moment. Using the raw data worked better for my choice of learning model, so I did not use PCA at all.

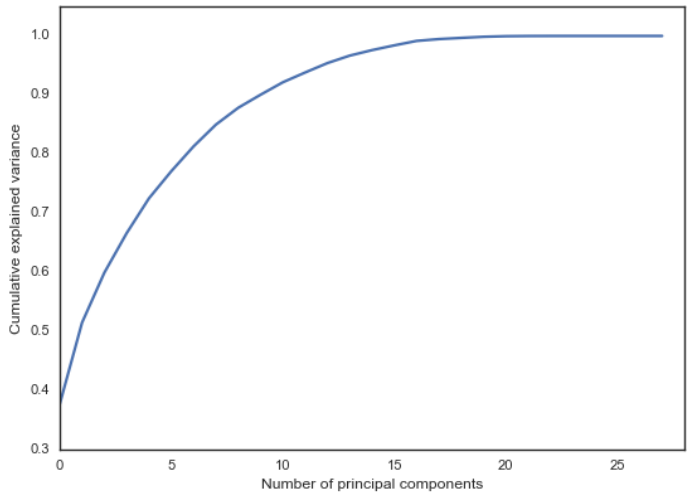

Figure 1. Cumulative explained variance by principal components

I move on to plot the the first 2 principal components. As expected, there is no clear separation for the target cases, so no improvement should be observed during the modeling phase in comparison to the raw features.

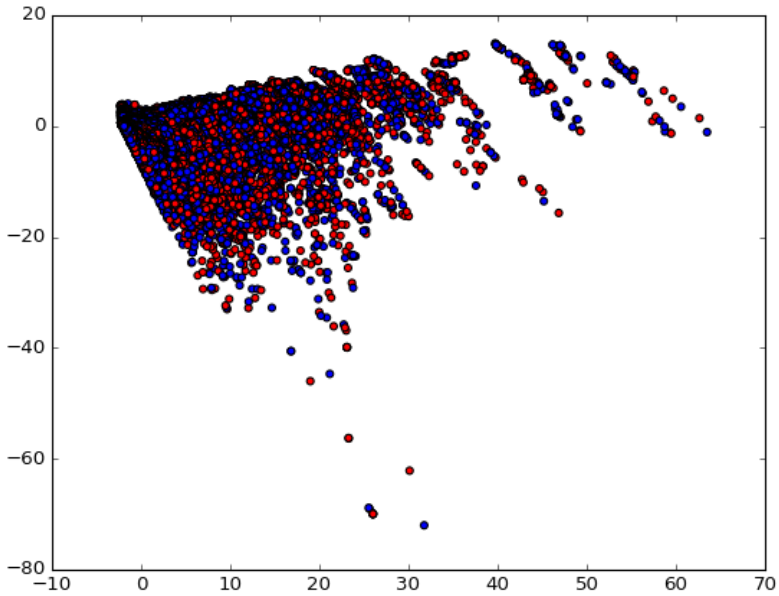

Figure 2. First two principal components plot

- Data Cleaning

One “quirk” that I found about the data was that despite the fact that most features had very similar distributions, the *mail_category* had severely different distributions between training and test sets.

The most frequent mail category from the training set, which is 15, is completely absent from the test set, so I pondered how it should be handled, whether to do nothing, remove the feature or throw away the cases entirely. During the modeling phase it turned out that *mail_category* wasn’t too influential, but it still had some significance, so I kept it.

Moving on to missing data, the datasets were fairly clean in this regard. Most data missing were from exclusive training data features. Since they were not used in the model, I did not perform any kind of imputation. There were two other features that had missing data, *last_online* and *hacker_timezone*. Since the amount of missing cases was so small, I simply imputed the median and the mode respectively.

- **Feature Engineering**

Feature engineering turned out to be crucial, as it usually is for prediction modeling competitions. For datasets that represent interactions between actors, in this case the user and an email, embedding past outcome feedback as feature tend to carry a lot of information and for this challenge it is no different.

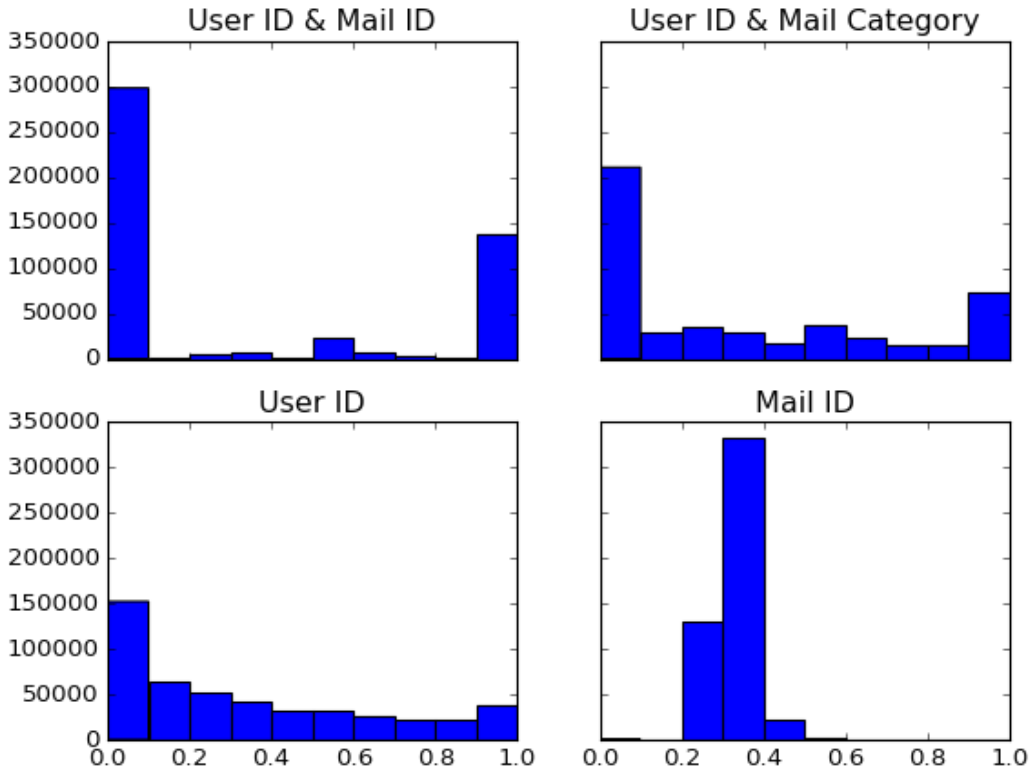

Figure 3. Value distribution of the mean of user, mail, and mail category combinations

As illustrated in figure 3, there is a lot of value in grouping by the user, email, and email category and extracting the mean. There is a caveat, however. Results for grouping by user and email are exceptionally good, but only because there are very few repetitions in user-email pairs. Adding this feature to the model results in major overfitting, turning the predictions useless. I ended up including only the mean by user and mean by email, which gave a great score boost.

Another aspect that I tried exploring were the time features, transforming raw timestamp to hours, days, days of the week and months. I experienced varying degrees of success with these features. None were nearly as relevant as the grouped means, but brought some improvement nonetheless. Tried many other combinations between the time features, but they also yielded marginally better results at best.

Predictive Modeling

Since the problem at hand is a classification problem with labels available and there was no obvious linear separation as shown by the PCA diagram, I chose to use XGBoost as the learning model. XGBoost is an implementation of gradient boosting with built-in regularization and has seen a lot of success in machine learning competitions, in particular for classification problems. One key feature of XGBoost is that it allows the user to choose its evaluation metric, for which the model optimizes to. Although the evaluation metric for the challenge F1 score is not available on XGBoost, AUC is a good enough substitute as it is also a kind of balanced accuracy metric.

For the validation framework I chose to use 1/10 of the training set as a validation set. Not as reliable as using cross-validation, but since I was using means of the target as features, it simplified the validation framework and also allowed faster iterations while optimizing the hyperparameters of the model.

One thing that I noticed but have not explored further was that the training and test datasets were easily separable, meaning that the i.i.d. assumption on which many learning algorithms rely on, including XGBoost, is violated. This led to a fairly difficult and frustrating task to evaluate and optimize the model, with the feedback from the public leaderboard being completely different from the feedback of the local validation. The difference between scores was around 0.10~0.12.

Conclusion and Results

The major breakthrough was the use of the means of the target variable. Other than that, I couldn't find anything to improve the model significantly, be it new features, transformations, or hyperparameter optimization. The choice of evaluation metric proved to be very challenging, as it depends not only on the learning algorithm but also in the threshold optimization to give the final output. I ended up exhaustive search on the same validation set to optimize it, but I wonder if there wasn't a better way to choose it.

For future competitions I would like to explore non-linear dimensionality reduction techniques like t-SNE, which could lead to better results. Model ensembling is also something that could be applied but wasn't due to the tight timeframe of the competition.