PRML Data Contest

Sumanth Hegde, Subhankar Chakraborty

August 2020

1 Introduction

The challenge was on Employee churn prediction given the past ratings given by an employee and remarks made by that employee about the company collected over time. We are also provided with agreements and disagreements of an employee with other remark entries.

2 Feature Selection

We use a total of 34 features. The company id is a categorical feature, which we convert to a one-hot encoded vector (with 36 entries). Other data such as rating values, although discrete, are treated as continuous for convenience. We use the 10 latest ratings of an employee (if available) and the corresponding rating dates. With the remark data, we found that the length of the remark made had almost no significance. We also used the number of supports and number of oppositions received by an employee for his/her latest remarks, and the number of oppositions given by an employee overall. Wherever there was missing data, we left a `Nan` in its place and let `xgboost` handle it. All in all, our feature vector for each sample is 70 dimensions, with the entries as follows :

- 0-36: company ID
- 37-46: 10 latest ratings by the employee
- 47-56: rating dates corresponding to the last 10 ratings
- 57: average rating of a company
- 58: sum of supports and oppositions given by an employee
- 59: number of (positive) supports given
- 60: number of negative supports/oppositions given
- 61: mean of supports/oppositions given
- 62: sum of supports gotten to last remark
- 63: positive supports to last remark
- 64: negative supports to last remark
- 65: average of supports gotten
- 66: average user rating
- 67: last rating - average user rating
- 68: average company rating - average user rating
- 69: average user rating/average company rating

NOTE: To use make use of the dates, we developed a simple mapping from a date to an integer:

$$\texttt{value} = 10000 \times \texttt{year} + 100 \times \texttt{month} + \texttt{day}$$

This ensures each date gets assigned a unique integer and a unique date can be generated back from each such integer.

3 Data Pre-Processing

After generating the feature vectors as mentioned in section 2, we perform two major pre-processing steps, viz, up-sampling for the companies which have very few samples and ensuring that the ratio of 0's and 1's remains the same after doing so.
NOTE: When we talk about sampling 1's and 0's, it means sampling training data corresponding to people who left their jobs and people who did not leave their jobs respectively.

3.1 Up-sampling for companies with less number of samples

We check for the companies which are under-represented, i.e, have less than a certain `threshold` number of samples. The pseudo-code for each company is mentioned below where `n_samples` represents the number of training samples from that company.

```
diff = max(0, threshold-n_samples)
if diff>0:
    if diff>n_samples:
        generate diff samples with replacement
    else:
        generate diff samples without replacement
    end if
    add generated samples to training data
end if
```

We set `threshold` to 20.

3.2 Ensuring the ratios of 0's and 1's are the same

Once we are done with the up-sampling, we ensure that the ratio of 0's and 1's remains the same as before. The pseudo-code is mentioned below.

```
r = original ratio of 0's and 1's
perform up-sampling
pres_0 = number of 0's now
pres_1 = number of 1's now
ideal_0 = number of 0's required with pres_1 to get ratio r
ideal_1 = number of 1's required with pres_0 to get ratio r
if ideal_1>pres_1:
    sample ideal_1-pres_1 1's and append to training data
else:
    sample ideal_0-pres_0 0's and append to training data
end if
```

We found that we had to add 19 or 20 1's after performing the up-sampling. Since this number is much lesser than the number of 1's present initially, we sampled without replacement.

4 Training and Validation Data split

After we have performed the upsampling and ensured that the ratios are equal, we perform the split for the training and validation data. We sample a fraction of the training data for validation while ensuring the fact that the validation data has the same ratio of 0's and 1's as the training data. We set the value of sampling fraction to be 0.2. The rest is used as our actual training set.

5 Model Selection

We used the optimized gradient boosting algorithm available in the `xgboost` library, for it's superlative performance in a wide range of machine learning problems. The `xgboost` algorithm is also sparsity aware i.e it handles missing values in the train set, which is important in our case since not every feature would be available for every employee. We train an ensemble (20 models) of gradient boosted decision trees, each with a randomized parameter setting. We vary the following parameters :

- `max_depth` : The maximum depth of each base learner.
- `num_parallel_tree` : The number of parallel trees used in each boosting iteration.
- `n_estimators` : Number of boosting rounds.
- `learning_rate` : Learning rate for boosting.
- `subsample` : Subsampling ratio (In each iteration, a fraction of the training data is shown to the base learner based on this value)
- `colsample_bynode` : Feature subsampling at each node of the base learner

Further, to prevent overfitting, we feed only a fraction of the training data to each model. In summary, we trained an ensemble of 20 gradient boosted trees where each base learner was a random forest, with the parameters of each model randomly selected from a fixed list. The list of parameters used were :

- `max_depth` : [3, 4, 5, 6]
- `num_parallel_tree` : [63, 127, 255]
- `n_estimators` : [10, 15, 20, 25]
- `learning_rate` : [0.1, 0.3, 0.5, 1]
- `subsample` : [0.8, 0.9, 1]
- `colsample_bynode` : [0.8, 0.9, 0.6, 1]

The training data for each model is chosen randomly to be either 90% or 99% of the total train set.

6 Validation

Since we are taking an ensemble of 20 models, each of whose parameters are chosen at random, we do not have any validation to perform over hyper-parameters. (The list of values to use was chosen based on simple heuristics) However, we need to decide how to predict 1 and 0 from the output of the 20 models. The naive approach of taking the majority does not make sense for two major reasons

- The classes are highly imbalanced. We have almost 5 times the number of 0's compared to the number of 1's
- The loss matrix is not symmetric, i.e, we have much more to lose by predicting 0 when the true value is 1 than the other way round.

Hence, we try different `threshold` values ranging from 0 till 20. If at least `threshold` models predict 1, we predict 1 else we predict 0. The `threshold` is decided depending on what threshold gives the best weighted accuracy on the validation set. We used a `threshold` value of 6.

7 Analysis

In this section we check the feature importance of our models, try and interpret them, analyse what went wrong in the Kaggle data contest and hypothesize what we could have done to prevent it.

7.1 Feature Importance

Each `xgboost` model has an attribute called `feature_importances_` which gives an importance for each feature on a scale of 0 to 1. The sum of importances over all features in a model is 1. Since we are taking 20 models, we take the average importance of a feature over the 20 models. The top 5 most important features are mentioned below.

- Whether the employee belongs to company `phcvroct` or not.
- The date corresponding to the last rating.
- Number of negative supports given.
- Negative supports to last remark
- Whether an employee belongs to company `yodaczsb` or not.

On looking back at the training data, the following interpretations can be made.

- An employee belonging to company `phcvroct` or `yodaczsb` has a high chance of leaving. The number of people who left the two aforementioned companies are 230 out of 471 and 139 out of 178 respectively. Both the ratios are much higher than the average ratio of people leaving, which is 594 out of 3526.
- If a person has opposed a lot of his/her colleagues' remarks or his/her remarks have gotten a lot of opposes by colleagues, it means he/she is is disagreement with a lot of his co-workers and is thus more likely to leave.
- The last rating date being the second most important feature can be interpreted as being indicative of some degradation/improvement in the company environment over time.

7.2 Mistakes Made in the Data Contest

We got a relatively high score of **0.89597** on the public dataset. However, on the private dataset we scored only a modest **0.84396**. Our highest score on the public dataset was pretty close to the score we got on our validation set. We suspect what might have happened is that the private dataset had more samples from the companies which were under-represented in the training set while the public dataset did not have too many points from the under-represented companies. We think our validation set also had a distribution of companies similar to the one present on the public leader-board as the scores were close.

7.3 What we could have done better

Drawing from our hypotheses regarding what went wrong in the data contest, here are a few things we think we could have done to do better.

7.3.1 Handling under-represented companies

Up-sampling more heavily from the under-represented companies would have probably helped. 20 was probably too small a number to up-sample to. This would also mean we would need to choose the training and validation split more carefully. Since some under-represented companies have less than 10 samples, there is a possibility that a random selection of the validation set has all the samples from those companies in the validation set, rendering the process of more heavy up-sampling meaningless. We would need to ensure that we sample the validation set in such a way that all companies are represented in the training set and then up-sample the training set. The validation set having repeated samples can bias our results.

7.3.2 Boosting

Given how some companies are so under-represented, boosting should theoretically help. In fact, we did try two slightly different versions of boosting. It is important to understand what are we referring to boosting here as each `xgboost` model is a gradient boosted random forest. We keep the overall structure same as before, i.e, an ensemble of 20 "models" but instead of 20 `xgboost` models, each "model" is a collection `n_models_in_boost` `xgboost` models. Each of the `n_models_in_boost` models is trained using a naive boosting algorithm with $\beta = 3$. A pseudo-code is mentioned below.

```
for i = 1:20
    choose parameters
    train model[i, 1]
    boost mistake samples beta times
    for j = 2:n_models_in_boost:
        initialize xgb model with chosen parameters
        train model[i,j] on boosted samples
        save model
        calculate mistakes by model[i, j] on train set
        boost mistake samples beta times
    end for
    save models
end for
```

The performance on the private set for the models chosen is mentioned in Figure 1. As it is apparent, the performance on the public dataset is not that exceptional. Selecting among these models is tricky since one does not know the private test statistics and we incur a high cost for even a small number of errors on 1's.

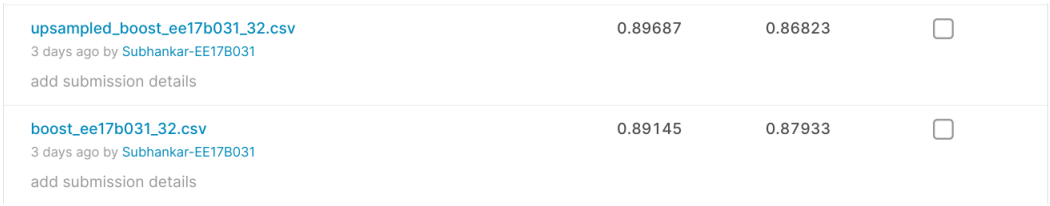

Figure 1: Performance of the two boosted models. `upsampled_boost_ee17b031_32.csv` is resulting prediction when we performed heavy up-sampling to make the number of 0's and 1's equal and then trained the model. `boost_ee17b031_32.csv` is the resulting prediction when we run the aforementioned algorithm on the training set as it is while performing the company-wise upsampling.