Team DAB: CS74/174 Homework #3

Machine Learning and Statistical Data Analysis: Winter 2016

Daniel Chen, **A**ndrew Kim, **B**enjamin Packer

Contents

1 Introduction

This project worked on finding a solution to the BNP Paribas Cardif Claims Management Kaggle competition. The competition is to use machine learning algorithms to effectively classify claims with anonymized data into two classes with minimal error:

(a) claims for which approval could be accelerated leading to faster payments

(b) claims for which additional information is required before approval

The purpose of this classification is to allow BNP Paribas Cardif to accelerate its claims process and provide a better service to its customers.

Error on both Kaggle and as presented in this paper is measured through Log Loss:

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$$

N is the number of observations, log is the natural log, y_i is the binary target and p_i is the predicted probability that $y_i = 1$

In the end, our solution is a ensemble of logistic regression, neural networks, and boosted trees. Our best testing error is XXXX, which ranks XXXX on the leaderboard.

2 The Data

The data given is split into training and testing. The sets are the same, except we do not know the expected targets on the testing data (Kaggle retains this information to prevent cheating).

The training set consists of 114,321 examples while the test set contains 166,607. Each example consists of 131 features, 19 which are categorical with the remaining numerical. The features are all anonymized, so we don't know what the data means.

2.1 Analysis of the Data

2.2 One Hot Encoding

We originally converted our categorical features to integer values. This confused our classifiers into believing the features expressed a numerical relationship. By switching over to a one-hot-encoding of our categorical features, we were able to improve performance.

One-hot-encoding splits each feature into n features, where n is the number of unique values that this feature takes on in both the training and testing sets. Each of these resulting features either take on 0 or 1. For each data point, only a single feature (the one which corresponds to the value that the original categorical feature had) has the value 1. This increases the number of features, but allows for correct encoding of categorical data.

One issue with one-hot-encoding was that we had to remove one of the categorical features, v22. This is because this feature had over 1000 unique values, so it increased the number of features to a number which was computation cumbersome. We prefer the empirical increase in performance to the loss of this single feature.

Our original logistic regression without one hot encoding achieved an error of 0.49859. After switching to one-hot-encoding our error significantly to 0.48213.

3 Logistic Regression

4 Boosted Trees

Following suggestions on the forum, we implemented boosted trees using the XGBoost library, which stands for eXtreme gradient boosting. The library is designed an optimized for boosted tree algorithms. We utilized the Python package for this library. The classifier uses an ensemble of trees.

The seven parameters that we optimize are shown below:

max_depth The maximum depth of a tree in the ensemble

min_child_weight Defines the minimum sum of weights of all observations required in a child

gamma The minimum loss reduction required to make a split

colsample_bytree The fraction of columns to be randomly sampled for each tree

subsample The fraction of observations to be randomly samples for each tree

eta The learning rate

num_rounds The number of rounds

4.1 XGBoost Parameter Tuning

The parameters were tuned using two-fold cross validation. After playing around with tuning the parameters manually, we ran two large sets of parameters in an attempt to reduce error and understand the empirical relationships between the parameters.

4.1.1 Tuning 1

The first optimization iterated through all 120 combinations of these parameters:

```
max_depths = [2, 3, 4, 5, 6]
min_child_weights = [1]
gammas = [0, 1]
colsample_bytrees = [0.5, 1]
subsamples = [0.5, 1]
rounds_and_eta = [(20, 0.3), (50, 0.1), (100, 0.05)]
```

With this, our best results are shown below:

error	runtime	minchildweight	subsample	eta	colsamplebytree	max depth	gamma
0.468638593	347.661603	1	1	0.05	0.5	6	0
0.468680062	346.063396	1	1	0.05	0.5	6	1
0.468846706	177.2962441	1	1	0.1	0.5	6	1
0.468898392	178.4002779	1	1	0.1	0.5	6	0
0.469002588	661.6131201	1	1	0.05	1	6	1

The testing error found using the best parameters from the tuning was: 0.46853

4.1.2 Tuning 2

Our second optimization iterated through all 24 combinations of these parameters:

```
max_depths = [6, 8, 10]
min_child_weights = [1, 2]
gammas = [0]
colsample_bytrees = [0.5, 1]
subsamples = [1]
rounds_and_eta = [(200, 0.05), (300, 0.01)]
```

With this, our best results are shown below:

error	runtime	minchildweight	subsample	eta	colsamplebytree	max depth	gamma
0.464182985	1191.265073	1	1	0.05	0.5	10	0
0.464419596	1154.138998	2	1	0.05	0.5	10	0
0.46442771	959.313591	2	1	0.05	0.5	8	0
0.464540668	1020.961268	1	1	0.05	0.5	8	0
0.466287292	760.0540562	2	1	0.05	0.5	6	0

The testing error found using the best parameters from this tuning was: 0.47856

Since this testing error is greater than the testing error achieved from the first tuning, there is evidence that somewhere between the first tuning and the second tuning we began to overfit the training data.

4.1.3 Final Selection of Parameters to be Ensembled

Although we did not end up using the optimal results from our tuning, it provided valuable insights to how we can use the parameters to achieve better results. Ultimately the results from our tuning was leading us to areas where this kind of brute force technique towards parameter tuning would become too computationally expensive, as tuning 2 already took nearly 12 hours to run.

Mostly we spent time modifying eta, max_depth, and colsample_bytree.

Ultimately, the parameters that we came up with were:

```
max_depths = 16
min_child_weights = 1
gammas = 0
colsample_bytrees = 0.68
subsamples = 0.5
num_round = 1800
eta = 0.01
```

This gave us a testing error of 0.45695. The runtime was approximately 9773 seconds (approx. 3 hours).

5 Neural Networks

6 Ensemble

7 Cross Validation

8 Result

9 Responsibilities

9.1 Daniel Chen

Daniel implemented the One Hot Encoding, which improved the performance of our classifiers by encoding the categorical data correctly. In addition, he implemented the boosted trees using XGBoost, and also ran the parameter tuning for that classifier. He assisted with the implementation of cross validation, especially K-Fold. For all of these contributions, he wrote up the corresponding sections in this report. In addition to those sections, he wrote up the introduction and data sections.

9.2 Andrew Kim

9.3 Benjamin Packer