Otto Group Product Classification

Machine Learning M2 AIC

Members:
VU Trong Bach
NGO HO Anh Khoa
Divya GROVER
Mahmut CAVDAR

I. INTRODUCTION

The objective of Otto Group Product Classification Challenge is to build a model which is able to distinguish between 9 categories provided. The dataset is organized in folders containing training and testing data for products selected randomly. The training set is comprised of 61.878 labeled products (Class_1, Class_2, ..., Class_9) and have 93 numerical features ranging from 0 to 352. Our work is to predict 144.368 products of the testing dataset (test.csv) in the format predefined (sampleSubmission.csv). The four different approaches-algorithms are used for this task and their results are compared.

II. NAIVE BAYES

Treated as Baseline algorithm. Crude Naive Bayes gives score of 11.0 on private leaderboard. Naive Bayes with probability calibration and cross-validation improves the score to 1.01, compared to the best score of 0.382. Spectral clusters for the training data suggests:

1. That almost all samples cluster into one group suggests that the features are implicitly not very discriminative.

2. The data is not very geometrically separable, hence boosting algorithms might perform better than any other decision-surface methods.

III. DECISION TREE - EXTRA TREE

A decision tree predicts a target value by asking a sequence of questions. In our implementation firstly we started by using pure classifier of sklearn without any configuration. For evaluation of classification model we separate training set into two subsets: real training data(%90) and test set(%10). Loss value of our initial implementation is -9.77 ($\pm$ 0.23). In the first phase, we tune 'max_depth' parameter by doing cross-validation for several possible values of that parameter and set the value of 'max_depth' = 10. Our loss value decrease significantly to -1.17 ($\pm$ 0.05). Secondly we try to decrease number of 'max_feature'. Best loss value is -1.16 (+/- 0.06) with 90 features. Also we try to tune the other parameters like 'class_weight','min_samples_split',etc. But we can not improve our final score significantly.

Secondly ExtraTree and Calibrated Classifier are used for comparison. We use default configuration except feature size. In our experience, 'max_features' value is 73. The result of cross-validation is decreased to -0.55 ($\pm$ 0.02).

IV. LINEARSVC

This section describes Support Vector Machine for the classification task. Based on Scikit-learn algorithm cheat-sheet, **LinearSVC** is first choice for multi-class classification with the labeled data having less than 100.000 samples.

LinearSVC with linear kernel uses "**one-vs-the-rest**" strategy for multi-class classification. As regards the data, its features and its class are unbalanced. In fact, the sum of each features have a large range from 0 to 180.000, which means that data should be pre-processed. However, the testing set logloss result of two methods StandardScaler (0.76530) and maxabs_scale [-1,1] (0.76744) are worse than **no-scale** (0.73934). Moreover, the good features for classification are selected by

ExtraTreesClassifier but it does not improve the final score. In addition, Class_2 and Class_6 have more than 14000 samples, followed by Class_3 and Class_8 with about 8000 samples and the rest have less than 5000 samples. For this reason, parameter **class_weight** is set *balanced*.

Other parameters of LinearSVC selected by **GridSearchCV** with **10-Fold**. This LinearSVC estimator is used in **CalibratedClassifierCV** for probability calibration with **method** = *isotonic* (*isotonic* with logloss 0.66940 is better than *sigmoid* with 0.77200), which decreases significantly the logloss of testing set. In brief, the best logloss found for LinearSVC is **0.66940** and its parameters:

- LinearSVC(penalty='l2', loss=squared_hinge, tol=0.0001, C=1.0, class_weight='balanced', max_iter=1000)
- CalibratedClassifierCV(method='isotonic', cv=10)

V. PARAMETER TUNING IN GRADIENT BOOSTING

10 gradient boosting machines (GBM) were each components of a 10-fold cross-validation.

We essentially broke the Kaggle training data into 10 folds and used each of these folds as a validation set, and the others as training. This produced 10 gradient boosting machines.
We then used an NxM coefficient matrix to blend each of these together. Where N is the number of models, M is the number of features. In this case it was a 10x9 grid. This matrix weighted each of the 10 model's predictive power in each of the 9 categories.

These coefficients were a straight probability calculation from the confusion matrix of each of the 10 models. This allowed each model to potentially specialize in each of the 9 categories.
We spent considerable time tuning my GBM. We used Nelder-Mead searches to optimize my hyper-parameter vector. We ultimately settled on the following parameters:
param['objective'] = 'multi:softprob', param['eta'] = 0.125, param['max_depth'] = 4, param['eval_metric'] = 'mlogloss', param['silent'] = 1, param['nthread'] = 10, param['num_class'] = 9, param['subsample'] = 0.7, param['colsample_bylevel'] = 0.12, param['colsample_bytree'] = 1.0.

VI. CONCLUSION

We conclude that no matter the choice of algorithm used, cross-validation plays a very important role in improving the predictive power of a given model. For this dataset, due high-dimensionality, space-partitioning algorithms like random-forests aka ExtraTreesClassifier tend to perform better. Our best log loss value is 0,577.