

# Tradeshift Text Classification using two-stage classifiers, Xgboost and scikit-learn

Jiwei Liu and Xueer Chen  
Pittsburgh, PA, USA

## Abstract

This report describes the winning solution of Tradeshift Text Classification hosted on Kaggle.com. The goal is to predict a sequence of probabilities that a piece of text belongs to each of the 33 classes. We adopt a two-stage-classifier approach to exploit the strong inter-dependence of the 33 labels. The training dataset is divided into base data and meta data. **The base data is trained to generate predictions of 32 labels for both meta data and the test data. These predictions are used as meta features, which are combined with raw features to train the meta data to generate the final predictions for the test set.** We use multiple classifiers at the base-stage, including Xgboost, Random Forest, online logistic regression, Stochastic gradient descent and Linear SVC. Xgboost is chosen as the single meta-stage classifier. Our best submission ensembles 14 two-stage models and 12 online models, which gets 0.0043324 private LB (1st place). Our best single two-stage model gets 0.0044595, which could be at the 3rd place. Most design choices and tricks are made based on cross-validation, since it agrees with Leaderboard score to 2nd or even 3rd significant figures.

## Two-stage Models

The procedure of the two stage model is described in the following figure [2]:

```
# training data X_train, y_train, and test data X_test

# split training data into two parts
X_base, X_meta, y_base, y_meta=split(X_train,y_train)

# train base data and predict meta and test data
clf_base.train(X_base,y_base)
X_meta_pred=clf_base.predict(X_meta)
X_test_pred=clf_base.predict(X_test)

# train meta data with raw features and meta features (predictions)
clf_meta.train(column_stack(X_meta,X_meta_pred),y_meta)

# generate the final prediction
pred=clf_meta.predict(column_stack(X_test,X_test_pred))
```

### General procedure of two-stage modeling

The rationale behind two-stage modeling is based on three observations:

1) Interdependence of labels. Despite 1,700,000 training samples, there are only 141 unique combinations of train Labels. Figure 2 depicts the histograms of top 37 frequent combinations of train labels.



```

from sklearn.feature_extraction import DictVectorizer
vec= DictVectorizer()
categorical_feature=[]
for every feature:
    if the feature is a categorical feature:
        train[feature] = map(str, train[feature])
        test[feature] = map(str, test[feature])
        categorical_feature.append(feature)
X_sparse = vec.fit_transform(train[categorical_feature].T.to_dict().values())
X_test_sparse = vec.transform(test[categorical_feature].T.to_dict().values())

```

Figure 1 Encoding Categorical (hashed) Features

## Training data split

Training data need to be split into two parts: base data and meta data. We experimented three different split : 1) random 50-50 split 2) use the 1st half of the training data as base and the 2nd half data as meta. 1) use the 2nd half of the training data as base and the 1st half data as meta. The split is nontrivial. And we made two observations:

1) Some split are superior than others in terms of performance, as shown in table 2.

split\LB	model 1	model 2	model 3
1st half base 2nd half meta	0.0048854	0.0048763	0.0048978
2nd half base 1st half meta	0.0047103	0.0047446	0.0047313

Table 2. Private LB score of similar models but different data split

2) Ensembling models based on different split improves performance. **Our best single model is using 2nd half as base and 1st half as meta.** Our final ensemble includes models of all three splits mentioned above.

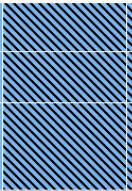
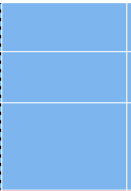
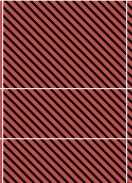
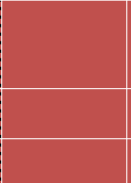




	Id	X_numerical		X_sparse	
<b>b a s e</b>	<b>1</b>				
	<b>2</b>				
	<b>.</b>				
	<b>.</b>				
<b>m e t a</b>	<b>.</b>				
	<b>.</b>				
	<b>.</b>				
	<b>1699999</b>				
	<b>1700000</b>				

Figure 3 data split and Feature engineering

## Base stage classifiers

$X_{numerical}$  and  $X_{sparse}$  are used directly for the base-stage classifiers except for online classifiers, which train raw features directly using the hashing trick without encoding, with forward selection and backward elimination to further boost accuracy. Specifically,  $X_{numerical}$  is trained by Random Forest,  $X_{sparse}$  is trained by SGDClassifier and LinearSVC, all of which are from scikit learn. We also use Xgboost [6] as the base-stage classifier to train all the features by combining  $X_{numerical}$  and  $X_{sparse}$  into one sparse matrix (scipy.sparse.csr\_matrix).

The hyper-parameters are fine-tuned through cross-validation and grid search. The detailed description of these classifiers can be found at [7], [8], [9] and [10].

The use of the Xgboost training all the features, `column_stack(X_numerical, X_sparse)`, at the base layer is very important because it is the only classifier that learns the interaction of hashed features ( $X_{sparse}$ ) and non-hashed features ( $X_{numerical}$ ). This is mainly an implementation issue. SGDClassifier and SVC can train `scipy.sparse.csr_matrix` directly but they have weak performance. Random Forest have better performance but it cannot train `scipy.sparse.csr_matrix` directly. Xgboost can take `scipy.sparse.csr_matrix` directly and it gives a better performance than Random Forest. This is the trick that helps us make the 1st place.

	with xgb training all features	without xgb training all features
private LB	0.0044595	0.0047103

Table 3 The improvement of xgb training all features at base stage

## Meta Stage Classifier

The predictions of the base stage classifiers are used as meta features, such as  $X_{meta\_rf}$ ,  $X_{meta\_svc}$ ,  $X_{meta\_sgd}$  and  $X_{meta\_xgb}$ . These features are combined (column-stacked), with the numerical features or sparse features or both, of the meta data. We tried all options and all of them contribute to our final ensemble. The meta stage classifier is also critical. Through cross validation we found Xgboost is superior than Random Forest in terms of both accuracy and speed. We use Xgboost as the only classifier at the meta stage. The hyper-parameter of Xgboost can be fine-tuned to improve performance. More detail about Xgboost is introduced later.

## Use full-dataset-predictions as the meta features of Test data

Here is another trick that we found very useful to further boost score. By default the two stage model only use base data (part of the whole training data) to generate meta features for the test data. However, we can “cheat” by using all the training data to predict the test data.

```
# train base data to predict meta data
clf_base.train(X_base,y_base)
X_meta_pred=clf_base.predict(X_meta)

# train all data to predict test data
# the predictions are used as meta features of test data
clf_base.train(X_train,y_train)
X_test_pred=clf_base.predict(X_test)
```

In this way, the meta features of the test set become more informative, hence generating more accurate final prediction. We can take an even more aggressive step. We can use the predictions of the ensemble of some similar models as the meta features of the test set. As mentioned at the beginning, we always use cross validation to evaluate these techniques and they turn out to be effective. This trick is actually using the meta stage classifier to learn the best ensemble of each single models.

In summary, based on the score of our submissions, the effectiveness of there tricks are ordered from most effective to least effective.

- 1) base stage feature engineering and classifier tuning**
- 2) meta stage classifier tuning**
- 3) use full-dataset-predictions as the meta features of Test data**
- 4) different Train data splits**

We have trained a few two-stage models which are different in these 4 aspects and more importantly they ensemble very well to boost the final score.

## Online models

Our online model is based on [11]. We mainly did three kinds of optimizations:

- 1) include more interactions. It is observed that interactions of hashed features are especially effective. We use cv and forward selection to add more features.
- 2) Linear model is really sensitive to bad features. We throw away dozens of numerical features, through cv and backward elimination.
- 3) Online training with real-time generated meta features. For each new training sample, it predicts twice: the first prediction is just like the baseline benchmark and generate 33 predicted labels. then these 33 labels are taken as meta features together with raw features and predict again. The second prediction is the final prediction for this sample, based on which the weights are updated, including weights for both raw features and meta features. The weights of base features use the same adaptive learning rate as in baseline, while the weights of meta features use constant learning rate. (Also decided by cv)

This generates our best single online model, which gets 0.0057688 private LB. Ensembling 12 online models, it gets 0.0056272 LB.

## Best single model

Our best single model is a two stage model, which

- 1) uses 4 classifiers at base stage: random forest for numerical features, SGDClassifier for sparse features, online logistic for all features and Xgboost for all features. SVC is not used since it is the weakest base classifier and to save some memroy.
- 2) at meta stage, Xgboost is fine-tuned to select a group of parameters.
- 3) we add the ensemble of predictions of 14 online models as the meta feature for the test set. All these models are trained using full data rather than the base data.
- 4) we also use a Xgboost to train full data, the `column_stack(X_numerical, X_sparse)`, to generate more accurate meta features for the test data set.
- 5) we use the 2nd half training data as base and the 1st half training data as meta, because this split gives a higher CV score.

The specifics of the base classifiers

- `svm = LinearSVC(C=0.17)`
- `sgd=SGDClassifier(loss='log',alpha=0.000001,n_iter=100)`
- `rf = RandomForestClassifier(n_estimators=200, n_jobs=16, min_samples_leaf = 10,`

`random_state=1,bootstrap=False,criterion='entropy',min_samples_split=5,verbose=1)`

- `xgb=xgb_classifier(eta=0.3,min_child_weight=6,depth=100,num_round=40,threads=16)`

The specifics of the meta classifiers

- `xgb=xgb_classifier(eta=0.09,min_child_weight=6,depth=18,num_round=120,threads=16)`

It takes roughly 48 hours to generate this single solution from scratch. We used a 8 core 32 GB server.

## Final ensemble

Our best solution is a weighted average of 14 two-stage models, 13 online models and 2 simple one stage models. The weights is adjusted manually based on the simple intuition, models with similar scores should have similar weights. For example we have 4 two stage models with the same data split, part2 as meta and part1 as base. They only differ in the parameters of the Xgboost. Therefore they have the same weight in the final ensemble, despite that their score is slightly different. If two models have very different score and very different structure, like two stage model and online model, we simply tweak the weights manually and follow the leaderboard score.

## Dependencies

- Python 2.7
- pypy 2.4.0
- Scikit learn-0.15.2
- numpy 1.7.1
- scipy 0.11.0
- Xgboost 0.3

## Code description

The code is organized as a group of scripts.

run.py: the main script

run\_online.py: the script that initiates all the online models

pre-ensemble.py: ensemble the predictions of all the online models as the meta features for further training.

pre-processing-base.py: prepare data for the base stage classifier

pre-processing-meta-xxx.py: prepared data for the meta stage classifier, and for different data split

xgb\_meta\_xxx\_predict: train the meta stage classifiers and predict, for different data split

other\_model.py: we develop another three simple models to break the tie in the last day. Two of them are single stage Xboost model that train the raw features directly, with different parameters of the classifier. The third one is also a two stage model, however rather than using numerical features at the meta stage, it uses both numerical features and sparse features as well as all useful meta features, however due to the time limit, we only use it to train y33.

ensemble.py: This is the final ensemble that generates the best solution we got.

## How to generate a solution

- Set Up all the dependencies
- change the data dir in run.py
- change the xgboost wrapper path in ./src/xgb\_classifier.py
- python run.py

The best single solution:

xgb-part1-d18-e0.09-min6-tree120-xgb\_base.csv private LB 0.0044595

The best ensemble solution:

best-solution.csv private LB 0.0043324 (1st place)

## Reference

- [1] Tradeshift Text Classification: data description, <http://www.kaggle.com/c/tradeshift-text-classification/data>
- [2] Tradeshift Text Classification: Benchmark with sklearn  
<http://www.kaggle.com/c/tradeshift-text-classification/forums/t/10629/benchmark-with-sklearn>
- [3] [http://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.DictVectorizer.html](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html)
- [4] Tradeshift Text Classification: does y33 decides everything?  
<http://www.kaggle.com/c/tradeshift-text-classification/forums/t/10521/does-y33-almost-decides-everything>
- [5] scikit learn **scikit-learn.org**
- [6] Xgboost <https://github.com/tqchen/xgboost>
- [7] sklearn svc: <http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
- [8] sklearn SGDClassifier:  
[http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.SGDClassifier.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html)
- [9] sklearn RandomForest:  
<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- [10] Xgboost parameters: <https://github.com/tqchen/xgboost/wiki/Parameters>
- [11] Online model  
<http://www.kaggle.com/c/tradeshift-text-classification/forums/t/10537/beat-the-benchmark-with-less-than-400mb-of-memory>
- [12] Tradeshift: interaction of features  
<https://www.kaggle.com/c/tradeshift-text-classification/forums/t/10537/beat-the-benchmark-with-less-than-400mb-of-memory/55685#post55685>