Xiaomi Logistics Service Quality Prediction

1 Background

1.1 Business background

With the gradual increase in the categories of goods sold in Xiaomi Mall, the gradual increase in the coverage of Xiaomi's logistics network, and the continued impact of the epidemic, online shopping is playing an increasingly important role in our daily lives. When purchasing products online at Xiaomi Mall and using Xiaomi logistics services, we usually decide whether to purchase products by browsing the service reviews. However, the quality of reviews on Xiaomi's logistics services is currently uneven, and there are even malicious praise or malicious negative reviews, which seriously affects the customer's shopping experience. Therefore, the prediction of review quality has become one of the indispensable topics for Xiaomi Logistics. If the quality of service reviews can be automatically evaluated, it will be possible to avoid showing low-quality reviews based on the predicted results. In this case, we will predict the review quality in the real scene of Xiaomi Logistics based on the integrated learning method.

1.2 Concept of Net Promoter Score

NPS (Net Promoter Score), also known as Net Promoter Score, or word of mouth, is an index that measures the likelihood that a certain customer will recommend a certain company or service to others. It is the most popular customer loyalty analysis indicator, focusing on how customer reputation affects business growth. By closely tracking the Net Promoter Score, companies can make themselves more successful.

The net recommendation value is equal to the percentage of recommenders minus the percentage of critics.
Net Promoter Value (NPS) = (Number of Recommenders/Total Number of Samples) × 100%-(Number of Detractors/Total Number of Samples) × 100%

Determining your Net Promoter Score is straightforward: ask your customers a question-"Will you be willing to recommend the "company name" to your friends or colleagues?"
According to the level of willingness to recommend, let customers score between 0-10, and then you build 3 categories of customer loyalty based on the score:

- Referrers (scores between 9-10): People with fanatical loyalty who will continue to buy and recommend to others.
- Passive (score between 7-8): Overall satisfaction but not fanatical, will consider other competitors' products.
- Detractor (score between 0-6): Dissatisfied with the use or not loyal to your company. The logic of the NPS calculation formula is that recommenders will continue to buy and recommend to others to accelerate your growth, while critics can destroy your reputation and prevent you from growing in a negative word of mouth.

NPS scores above 50% are considered good. If the NPS score is between 70-80%, it proves that your company has a group of highly loyal and good customers. The survey shows that the NPS value of most companies still fluctuates between 5-10%.

2 Service quality prediction model

We extracted reviews on Xiaomi logistics services from the official website of Xiaomi Mall, including 57,039 comments in the training set and 11,208 comments in the test set. Based on the above data, through the realization of two integrated learning algorithms (Bagging and AdaBoost.M1), to build the Xiaomi logistics service quality prediction model. Among them, the base classifier uses SVM and decision tree. Therefore, a total of four sets of results need to be compared, and AUC is used as the evaluation index:

- Bagging + SVM
- Bagging + decision tree
- AdaBoost.M1 + SVM
- AdaBoost.M1 + decision tree

2.1 Data extraction

- Extract data from the test set and training set of the model

```
import pandas as pd
import numpy as np

train_df = pd.read_csv(' /Users/chan/Documents/data/train.csv', sep='\t')
test_df = pd.read_csv(' /Users/chan/Documents/data/test.csv', sep='\t')
len(train_df), len(test_df)
```

Output: (57039, 11208)

- View the first 5 rows of data

```
train_df.head()
```

	reviewerID	asin	reviewText	overall	votes_up	votes_all	label
0	7885	3901	First off, allow me to correct a common mistak...	5.0	6	7	0
1	52087	47978	I am really troubled by this Phone and Compute...	3.0	99	134	0
2	5701	3667	A near-perfect package of a downright glo...	4.0	14	14	1
3	47191	40892	Keep your expectations low. Really really low...	1.0	4	7	0
4	40957	15367	"they dont make em like this no more..."well.....	5.0	3	6	0

```
test_df.head()
```

	Id	reviewerID	asin	reviewText	overall
0	0	82947	37386	I REALLY wanted this series but I am in SHOCK ...	1.0
1	1	10154	23543	I have to say that this is a work of art for m...	4.0
2	2	5789	5724	Mi 12 is certainly the most controversal smar...	3.0
3	3	9198	5909	I love this earphones...? Well, of course i...	5.0
4	4	33252	21214	Even though I previously bought the Redmi Ser...	5.0

2.2 Feature extraction

Process text features by extracting "Excellent" and put together the total score features

```
import scipy
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer

# tf/idf Processing text features
word_model = TfidfVectorizer(stop_words='Excellent')
train_X = word_model.fit_transform(train_df['reviewText'])
test_X = word_model.transform(test_df['reviewText'])

# Combine the total score feature
train_X = scipy.sparse.hstack([train_X, train_df['overall'].values.reshape((-1, 1)) / 5])
test_X = scipy.sparse.hstack([test_X, test_df['overall'].values.reshape((-1, 1)) / 5])
```

2.3 Ensemble algorithm implementation

- SVM and decision tree algorithm call, and probability correction

```
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.calibration import CalibratedClassifierCV

def construct_clf(clf_name):
    clf = None
    if clf_name == 'SVM':
        clf = svm.LinearSVC()
    elif clf_name == 'DTree' :
        clf = DecisionTreeClassifier(max_depth=10, class_weight='balanced')
    elif clf_name == 'NB' :
        clf = BernoulliNB()
    clf = CalibratedClassifierCV(clf, cv=2, method='sigmoid')  # Probability correction
    return clf
```

- Bagging algorithm implementation:

```
class Bagging(object):
    def __init__(self, clf, num_iter):
        self.clf = clf  # Classifier object
        self.num_iter = num_iter  # Number of bagging classifiers

    def fit_predict(self, X, Y, test_X):
```

```
        result = np.zeros(test_X.shape[0])  # Record the prediction results of the test set
        train_idx = np.arange(len(Y))
        for i in range(self.num_iter):
            sample_idx = np.random.choice(train_idx, size=len(Y), replace=True)  # Bootstrap
            sample_train_X = X[sample_idx]
            sample_train_Y = Y[sample_idx]
            self.clf.fit(sample_train_X, sample_train_Y)
            print('Model {:>2d} finish!'.format(i))
            predict_proba = self.clf.predict_proba(test_X)[:, 1]
            result += predict_proba  # Accumulate the predicted probabilities of different classifiers
        result /= self.num_iter  # Take the average (vote)
        return result
```

- AdaBoost.M1 algorithm implementation：

```
class AdaBoostM1(object):
    def __init__(self, clf, num_iter):
        self.clf = clf  # Classifier object
        self.num_iter = num_iter  # Number of iterations

    def fit_predict(self, X, Y, test_X):
        result_lst, beta_lst = list(), list()  # Record the prediction results and voting weight of each iteration
        num_samples = len(Y)
        weight = np.ones(num_samples)  # Sample weight, note that the sum should be num_samples
        for i in range(self.num_iter):
            self.clf.fit(X, Y, sample_weight=weight)  # Weighted fit
            print('Model {:<2d} finish!'.format(i))
            train_predict = self.clf.predict(X)  # Training set prediction results
            error_flag = train_predict != Y  # Predict the location of the error
            error = weight[error_flag].sum() / num_samples  # Calculation error rate
            if error > 0.5:
                break
            beta = error / (1 - error)
            weight *= (1.0 - error_flag) * beta + error_flag  # Adjust the weight, multiply beta by the correct position, and the wrong position is still the original
```

```
            weight /= weight.sum() / num_samples  # Normalize so that the weight sum is equal to
num_samples
            beta_lst.append(beta)
            predict_proba = self.clf.predict_proba(test_X)[:, 1]
            result_lst.append(predict_proba)
        beta_lst = np.log(1 / np.array(beta_lst))
        beta_lst /= beta_lst.sum()  # Normalized voting weight
        print('\nVote Weight:\n', beta_lst)
        result = (np.array(result_lst) * beta_lst[:, None]).sum(0)  # Weighted sum of prediction
results for each round
        return result
```

2.4 Test and generate results

Test the following 4 groups of prediction models and generate results, and finally get voting weights.

- Bagging + SVM
- Bagging + decision tree
- AdaBoost.M1 + SVM
- AdaBoost.M1 + decision tree

```
np.random.seed(0)
clf = construct_clf('SVM')  # DTree, SVM, NB
# runner = Bagging(clf, 10)
runner = AdaBoostM1(clf, 10)
y_predict = runner.fit_predict(train_X.tocsr(), train_df['label'], test_X.tocsr())
```

```
Model 0  finish!
Model 1  finish!
Model 2  finish!
Model 3  finish!

Vote Weight:
 [0.47013022 0.35834806 0.17152172]
```

- Create submission documents and derive AUC evaluation indicators

```
# Generate submission file
result_df = pd.DataFrame()
result_df['Id'] = test_df['Id'].values
result_df['Predicted'] = y_predict
result_df.to_csv('./result.csv', index=False)
```

Method	Base	+Bagging	+AdaBoost.M1
DTree	0.74	0.77	0.76
SVM	0.78	0.81	0.81

3 Conclusion

From the above training results, it can be concluded that the prediction accuracy rate of using SVM+Bagging or SVM+AdaBoost.M1 is high, and this model can be used to continue to predict the quality of Xiaomi logistics service.