Lab 1 Report: Email Spam Filter

Sui Huang

March 9, 2018

1 Print Top Ten Features

Top 10 features are:
language, free, remove, linguistic, university, money, our, click, business, market

The top features are selected by:
1.Calculate information gain for each feature.
2.Sort the information gain, pick up the top-N index as top-N features index.
3.Extract feature name and print by its index.

2 Print Precision and Recall Tables

Precision Table

Number of Featrues	10	100	1000
Multinomial NB with TF	0.851852	0.959184	1.0
Multinomial NB with BF	0.888889	0.977778	1.0
Bernoulli NB with BF	0.869565	0.939394	1.0

Recall Table

Number of Featrues	10	100	1000
Multinomial NB with TF	0.938776	0.959184	0.938776
Multinomial NB with BF	0.816327	0.897959	0.938776
Bernoulli NB with BF	0.816327	0.632653	0.612245

3 Design a Support Vector Machine (SVM) based spam filter

Methodology description:
Use Binary Features, 100 top featrues are selected by calculating information gain. I use K-Fold to achieve cross validation, the training set is split into 10 folds. Nine for training and one for validation. After training and validate on the training set, I fit the testing set to the trained SVM and calculate the result.

Choosing Parameters:
Kernel = "sigmoid". C=10 because I want to have a smaller margin hyperplane to avoid misclassify. Verbose = 10 to see some indirect results in jupyter notebook command line.

SVM Result:
Accuracy = 0.979254
Precision = 0.985364
Recall = 0.888575

```
from sklearn.model_selection import KFold
from sklearn.metrics import precision_recall_fscore_support
from sklearn.svm import SVC

top_feature, drop_feature, name_feature = feature_selection(100, x_train)

x_train_bf, x_test_bf = binary_feature(ls_train)
x_train_tf, x_test_tf = term_frequency(ls_train)

count_vect = CountVectorizer()
x_train = count_vect.fit_transform(ls_train.data)

nfold = 10
kf = KFold(n_splits=nfold,shuffle=True)

acc = []
pre = []
rec = []
svc = SVC(kernel="sigmoid", C=10, verbose=10)

for train, test in kf.split(x_train_bf):
    Xtr = x_train_bf[train,:]
    Xts = x_train_bf[test,:]
    ytr = ls_train.target[train]
    yts = ls_train.target[test]

    svc.fit(Xtr,ytr)
    yhat = svc.predict(Xts)

    acci = np.mean(yhat == yts)
    prei,reci = pre_rec(yts,yhat)

    acc.append(acci)
    pre.append(prei)
    rec.append(reci)

acc_mean = np.mean(acc)
pre_mean = np.mean(pre)
rec_mean = np.mean(rec)
```

4 Results of Evaluation Set

Codes are under Spam Filter NB notebook.

Result: [0 0 1 1]

5 Extra Credit

5.1 Methodology Followed in Implementation

I tried to follow the instruction in paper: Adversarial Classification. The problem of computing MCC should be a NP problem and uses dynamic programming in the paper. I did not figure out the dynamic code for this problem so I used iternation and for-loops. Since the training data set has only 10 features, the problem can be solved quickly.

My method of attack is to find out which feature's appearence have big influences the probability of an email being classified to legit. And add this kind of featrues to fool the spam filter. And my mythod of opdating is to add edited spam emails to the original training set, then use cross validation to train the classifier again. Then the classifier will be updated.

1.Import data from original training dataset, transform the data to binary feature with 10 top ig selected features.

2.Calculate 'contribution'(log odds) for each feature. For each feature in all ten features, compute log odds for the ith feature,them compute the log odds of the ith feature added to all spam emails. If the new log odds is bigger than the original one, which means adding the ith feature would result in classifing spam to legit. Keep a record of this kind of feature and its cost.

3.Generate new vector space: Use gray code to generate a new vector space which contains all the possible combaniation of features from 2.

4.Pick up ADD-WORDS: Try each combination in 3 and looking for the lowest cost, that each spam email in the test set gets classified as legitimate by the baseline NB classifier. Which means precision = 0.

5.Perform the attack and print the result.

6.Adding new edited spam emails to the original training set.

7.Train the filter again. Print the result.

5.2 False Negative rate of the baseline NB classifier

Before modification: False Negative Rate Before Modification = 0.183673.

After modification: False Negative Rate After Modification = 1.000000.

Minimum unit cost: Average unit cost among all spam emails = 2.97959183673.

5.3 False Negative and False Positive rate of the updated NB classifier

False Negative Rate 0.0248928926089

False Positive Rate 0.0101858419634