CS6998 Homework 2: Text classifier

Author: Hang Qian

UNI:hq2124

Kaggle username: Bankq

Overview

Our goal is by training a dataset (6398 reviews with positive(1) or negative(0) label) to classify a testset (4265 unlabeled reviews). We choose an approach with naive bayes as basic classifier and bagging-learning with different feature selection. In the partial tests our approach stablized at 22% to 24% error rate. The project is implemented in Python programming language. It should be able to run under any python runtime environment.

Preprocessing

We preprocessed the data by having the words stemmed .

We use Porter algorithm to stem the words. In implementation, user can choose whether having the word stemmed or not.

Feature selection/extraction

Tokenizing

First we tokenized each review into tokens that contains a single word. And by using NLTK English stopwords corpora as stopwords set, we eliminated the common stopwords in English.

Chi-Square test

We implement the chi-square test to select *valuable* features. Its code can be found at `project/api.py:chi_square_test` . The selection makes sense because we found the words like `dull`, `boring` and `beauti-` etc. have high chi-square score, which are also informative in deciding whether the review is positive or negative. However it might be tricky to choose how many features based on chi-square score. We use two approach:

1. User deciding. We pass a ratio `r(0<r<=1)` into the classifier to choose top `r` features based on their chi-square score.

2. Max leap. By calculating where the maximum relative leap in chi-square happens, abandon the smaller ones. This approach will roughly eliminate 60% of the features, which is considered too much in this problem.

Bigrams

Instead of using single words as features, we can also choose to use bigrams of reviews as our features. By choosing the two adjacent words as one feature, we are expecting it to convey more information.

Classifier implementation

Naive Bayes

We use the maximum liklihood as the expectation of the `P(features|label)` and use their sum of logarithm to avoid underflow.

Smoothing We use laplace smoothing to avoid zero probability.

Bagging To help improve the accuracy of the classifier, we choose to use several (5 by default) naive bayes classifiers(with different selected features) to bagging-learning. And the final label based on the vote result.

The implementation of the classifier can be found at `project/api.py` .

Discussion

By submitting classifying results to the Kaggle system, we found that stemming and bagging improved the performance while eliminating stopwords and doing chi-square test didn't improve the performance but did reduce the computing time. The lack of training data is main reason of this tradeoff. To get a high-accuracy result, we can't afford reducing much features because we don't have enough confidence to decide which one is not relevent. The performance of the classifier depends on the feature selection and the best result come from choosing all the features. Due to it's a offline classifier, if we stick on the the training data we have, we recommend to fully use all the features.

Reference

1. IR Book
2. NLTK manual