Data Mining Assignment III

Fang Zhou

1. Introduction

The goal in this assignment is to build a classifier based on the feature vector produced in assignment 1. In this assignment, I will evaluate and compare among three different classification algorithms: K-nearest-neighbors algorithm, decision tree algorithm, and naive Bayesian algorithm. I will evaluate the algorithms for the online cost, offline cost, and the accuracy with different distance metrics. I also analyze and conclude the results.

2. Methods

2.1 Article Feature

In this assignment, I classify documents based on the TF-IDF features, and the word frequency. TF-IDF, as I introduced in previous assignment, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. TF-IDF's value range is from 0 to 1. Word frequency is collect the number of one word appears in the article. It is also a very important feature for an article. I use the most important 1000 words in the whole corpus based on the TF-IDF value in the corpus, and collect the 1000 words' TF-IDF value as the feature vector of each article.

2.2 Data Pre-processing

However, the vectors collected in the last assignment don't fit well for this assignment. For example, some articles may miss the topic part so that it cannot be classified into any categories. I pre-process the data following the rules shown below:

- If an article does not have the body part or its body part is empty, then skip this article.
- If an article does not have the topic part or its topic part is empty, then skip this article.
- If an article has more than one topic, then random pick up one topic for this article.

After this processing, the number of articles decreases to 8654.

Each article is bonded with only one topic. This processing makes the future processing easy and correct. Figure I shows the distribution of topics after preprocessing. It should be noted that I remove the topics with the number less than 30 and sum them up into one categories, others, in Figure I, because of the limited space.

As we can see, there are two main topics in the corpus after processing. They are "earn" and "acq", which occupies 43.16% and 24.56%. The input is not very uniform so that it may lead the prediction obviates to the two most topics.

2.3 Classification Algorithms

In this assignment, I choose K-nearest-neighbors (KNN) algorithm, decision tree algorithm, and naive Bayesian algorithm to classify the document. For KNN, I use Manhattan distance, Euclidean distance, and cosine distance. For naive Bayesian, I choose Gaussian naive Bayesian algorithm, which means each feature is considered fitting in a Gaussian distribution.

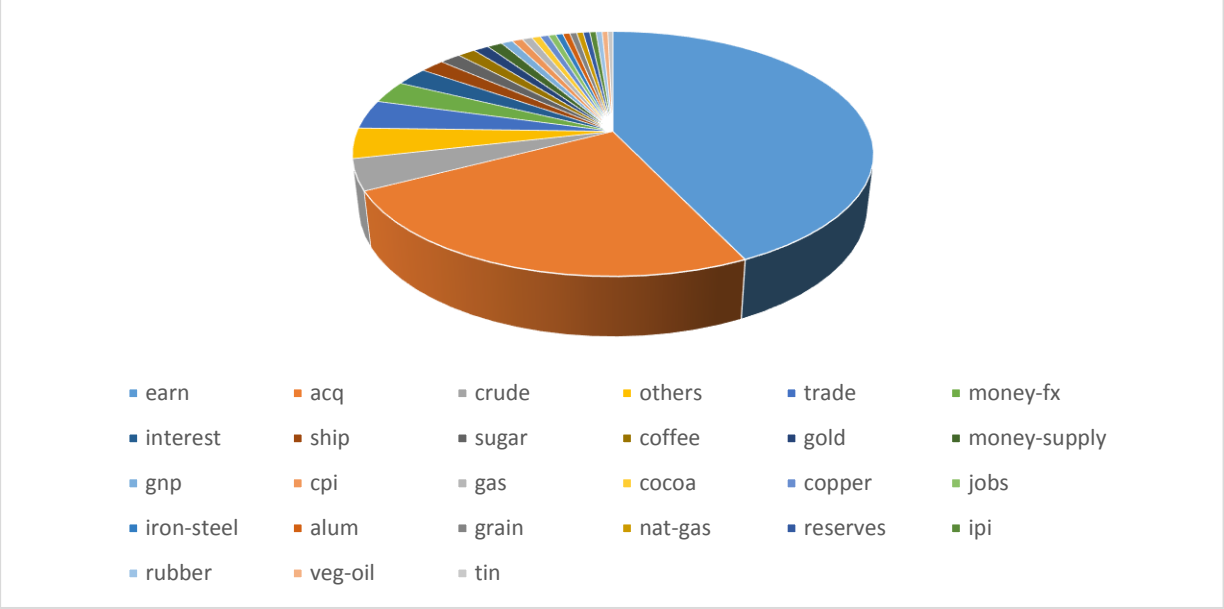

Figure I: The distribution of topics

The equations of these distances and the meanings of linkages are the same with the ones we discussed in the class.

There are a lot of parameters which can be configured in these three algorithms, I will show these parameters in Section 3.2.

3. Experiments

3.1 Testbed

I test the program on a single machine. The machine has a 2.4 GHz 6-core Intel Xeon CPU E5-2630, 24 GB memory and one 500 GB Western Digital SATA hard drives.

3.2 Experiments Details

KNN algorithm has two parameters for the input. The first one is the value of K, and the second one is the distance metric. For K, the intuitive value is the total number of articles divided by the number of distinct topics. For example, for my input, the K value should be 8654/65=130. However, our input is not uniform, I decide to try different values near or far from the intuitive value, such as 65, 260, and so on.

Naive Bayesian algorithm need to decide the type of each feature. In this algorithm, I assume each feature is continuous and fits a Gaussian distribution. This assumption is very useful and a common one used in naive Bayesian.

For decision tree, I use gini and entropy as the criterion to decide which feature should be used to split the current set. I don't limit the tree depth so that the depth tree can be very deep. I make sure that the leaf node can only have one classification.

I run each test for 10 times and calculate the average value of them. All the results are the average value without special notifications.

3.3 Implementation

I do not implement these algorithms by myself. I choose use scikit-learn library to run the evaluation. I spent some time to learn how to use related APIs in scikit-learn. I also have to define my-own distance metric because some metrics are not provided in the basic scikit-learn library. I think it is a good start for me to try to use the mature third party library.

4. Results

In this section, I will show the scalability and accurate results of each classification algorithms.

4.1 Scalability

First, I will show you the offline cost for the three algorithms. I use two kinds of decoupling. One is 8:2, the other is 6:4. In the first decoupling, I randomly shuffle the whole data set, then randomly pick up 80% or 60% as the train set, 20% or 40% as the test set, respectively.

For KNN algorithm, I choose K= 35, 70, 133, 260 to test the scalability so that we can easily to see if the performance changed with the different K values. I also use different distance metrics to test the scalability. The results are shown in table I, II, III, and IV.

Table I: Offline cost (millisecond) for KNN algorithm with 8:2 decoupling

K	euclidean	manhattan	cosine
35	0.093640003	0.084898	0.180156
70	0.094043313	0.085227	0.186948
133	0.094627518	0.085697	0.187843
260	0.095230898	0.085911	0.187621

Table II: Offline cost (millisecond) for KNN algorithm with 6:4 decoupling

K	euclidean	manhattan	cosine
35	0.084513463	0.083908	0.181098
70	0.084768091	0.083964	0.180835
133	0.084796295	0.084031	0.183655
260	0.084891567	0.084365	0.182581

As we can see, for KNN algorithm, the offline cost has an increasing trend in both 8:2 and 6:4 decoupling cases. The reason is K's value is increased meanwhile. If K has the larger value, then the computation should increase at the same time. So the offline cost increases with the increasing K value. Another interesting observation is the cosine offline cost is twice larger than euclidean and manhattan offline

cost. At first, I feel very confused about this. After checking the scikit-learn document, I understand that the cosine distance is a function implemented by myself and other two distances are built-in functions. This means that my cosine computation function has to be interpreted in the runtime. However, other two distance functions have been compiled into the related Python library. This is why the cosine offline cost is higher than the other two. If we can add cosine function to a built-in distance function, I think the cosine offline results should be similar with the other two.

Table III: Online cost (millisecond) for KNN algorithm with 8:2 decoupling

K	euclidean	manhattan	cosine
35	0.003537343	0.00392	0.038752
70	0.004009654	0.00394	0.042089
133	0.004022112	0.00395	0.043209
260	0.004044797	0.003958	0.042999

Table IV: Online cost (millisecond) for KNN algorithm with 6:4 decoupling

K	euclidean	manhattan	cosine
35	0.006542159	0.006422	0.069121
70	0.006558416	0.006426	0.070634
133	0.00660273	0.006475	0.070778
260	0.006649657	0.006503	0.069985

KNN's online cost has the similar trend with its offline cost. The cost becomes larger when the K value is increased. But the growth rate is not very high.

Below are the tables of offline and online cost for naive Bayesian algorithm and decision tree algorithm.

Table V: Offline and online cost (millisecond) for Bayesian algorithm

Decoupling	Offline	Online
8:2	0.0000368	0.000328
6:4	0.0000368	0.000374

Table VI: Offline and online cost (millisecond) for decision tree algorithm

Decoupling	Offline	Online
8:2	0.0007550	0.0000296
6:4	0.0007757	0.0000293

As we can see, the offline and online cost for each record are very similar no matter which decoupling mode we use and which algorithm we use. The situation is a little different from the KNN algorithm. In KNN, if K is very large, then the cost will increase meanwhile. If we use a fix K value in KNN, the online and offline cost becomes similar with naive Bayesian and decision tree algorithms. It should be noted that naive Bayesian algorithm spends more time on training and less time on testing. The opposite case

is decision tree algorithm. It spends more time on training and once the model is created, the online testing is very efficient. Both of these two algorithms spends less time than KNN algorithm.

4.2 Accuracy

In this section, I will use several tables to show the accuracy rate of each algorithm.

Table VII: Accuracy for KNN algorithms with 8:2 decoupling

K	euclidean	manhattan	cosine
35	37.7%	41.8%	41.8%
70	39.3%	42.3%	41.7%
133	42.5%	43.5%	43.0%
260	42.5%	42.4%	42.3%

Table VIII: Accuracy for KNN algorithms with 6:4 decoupling

K	euclidean	manhattan	cosine
35	39.9%	43.0%	43.3%
70	41.9%	43.4%	43.8%
133	43.7%	43.4%	43.9%
260	43.2%	43.3%	43.1%

Table IX: Accuracy for Bayesian and Decision Tree Algorithms

Method with decoupling	Accuracy
Tree-8:2	25.01%
Tree-6:4	26.72%
Bayes-8:2	8.84%
Bayes-6:4	11.18%

As we can see, the most accurate algorithm is KNN algorithm with 6:4 decoupling and cosine distance. Naive Bayesian algorithm has the worst accuracy, compared to the other two algorithms.

5. Discussions

It is very complicated and difficult to explain the phenomenon we have tested and observed. However, we can conclude some results we have observed:

- KNN algorithm spends more time on training and testing, compared to the other two algorithms. However, it can provide the best accuracy.
- Decision tree algorithm is a good balance between the accuracy and the scalability. It spends less time to create the model than KNN algorithm. It has the best testing speed. The accuracy is also not bad.
- Although naive Bayesian has the worst accuracy, it is the most efficient one to create the model. If we need to create our model as soon as possible, naïve Bayesian is a good choice.