COMP 6721 Project Report

Yilin Li[1] and Yuhua Jiang[2]

[1] 40083064 yilinli0524@gmail.com

[2] 40083453 joeyyy1967@gmail.com

1. Introduction and technical details

1.1 Program background

The purpose of this project is to extract the data and to build the model from the provided dataset and then using ML Classifier to test dataset. The original dataset is a CSV file of Hacker News fetched from kaggle. As mentioned in the project description, Hacker News is a popular technology site, where user-submitted stories (known as "posts") are voted and commented upon. The site is extremely popular in technology and start-up circles. The top posts can attract hundreds of thousands of visitors. For more information, this dataset contains Hacker News posts from 2018 to 2019-06 which include more than 2,500,000 posts. We are using all the posts belong to 2018 as the training set to build the model and posts belong to 2019 as the testing set.

1.2 Development environment

The project is written in the Python3.0 programming language, implemented in the Pycharm compiler. And, it is able to run on Jupyter book as the requirement in the description.

1.3 Technical implementation

The core technology we used to implement this project is several methods from *nltk* library. For example, word_tokenize, pos_tag, wordnet, extra... Apart from that. when extracting certain columns that contain useful data from the CSV file, we used the *read_csv* method from the *panda* library. For more detailed implementations, we tokenized and cleaned the dataset to remove all the unrelated data as the first step by *nltk.regexp_tokenize* method. Besides, each retained word is subjected to part of speech restoration by lemmatization using *WordNetLemmatizer* method in *nltk.stem* library with the pattern designed by ourselves.One of the most useful sources of information for WordNetLemmatizer is part-of-speech tags. This is one of the motivations for performing part-of-speech tagging in our information extraction system[2]. In this particular step, to be able to combine two nouns with special meaning together as a single token, we tagged every word with part with speech and analyzed it with bigram step in the program. Then, both the training data model with the frequencies and conditional probabilities and the testing dataset are built. It is also worth mentioning here, based on the request, we modify the smoothing parameter as well as the cleaning strength to analyze how do the side effect changes the final result. And in the testing phase, the program uses a *Naïve Bayes Classifier* to classify testing datasets in their likely class. In addition, to be able to plot the performance in line chart, we used *matplotlib.pyplot* library. Being able to set the limit time, the project references the *time* library and clocks the tasks separately to control the time of each one.

2. Results and analysis of the baseline experiment (exp.#1)

2.1 Brief description

In task 1(build the training model), we cleaned the original dataset by removing all the unrecognized characters and unrelated words; then, we tokenized each title into words when restoring its part of speech and also combining two meaningful nouns together; at last, we used the last data posted in 2018 to build the training model.

In task 2 (testing), we used the model built in task one, calculated the conditional probability of Naive Bayes for each title in the testing set (created in 2019). After we got an estimated type, compared it with the original type to get the accuracy, recall, precision, F1-measure and confusion matrix.

2.2 Main Data storage structure

Dictionary with List. The project a dictionary data structure to store the name of each type and the related data. More specifically, the key is a string of the name of the type and the value is a list contains two numbers, the first one is the total number of this type appears in the training set and the second one is the total number of the vocabulary appears in this. The final result of each type showing below:

```
{'story': [254222, 1718187], 'ask_hn': [12509, 109459], 'show_hn': [10225, 74998], 'poll': [25, 197]}
```

Fig. 1. the dictionary structure of classes

Dictionary with List containing Numpy Array. The dictionary data structure used by the program to store all the vocabulary in the training set and its frequency as well as a probability in each type. The key is a string of the current word. The value stores a list contains two Numpy array with the same size as the number of types, the first one store frequencies this word appeared in each type as the same order as the typeList and the second one contains all probabilities in the same order.

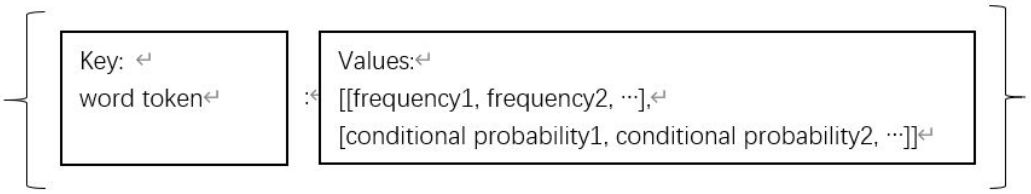

Fig. 2. the dictionary structure of vocabulary

2.3 Detailed implementation

Clean Data: Firstly, we tokenized each word as a token, created a "Regular Expression" to analyze if a word will be useful for the training model as well as the testing set, then, we rewrote each title. The program uses nltk.regexp_tokenize() method to retain data that matches this pattern. The detailed RE information shows as following:

```
pattern = r"""(?x)                    # set flag to allow verbose regexps
              (?:[A-Za-z]\.)+
             |(?:[$#a-zA-Z]+)*[0-9]+(?:[-.,/0-9]+)*['a-zA-Z]+(?:[-.,/0-9]+)*(?:[a-zA-Z]+)*
             |[$#A-Za-z]+[0-9]+(?:[.,/0-9]+)*
             |[a-zA-Z]+(?:[-'_/&][a-zA-Z]+)*(?:[+])*
          """
```

Fig. 3. the patterns to clean up data

There are four kinds of patterns to extract useful data. In the cleaning data phase, our objective of pattern matching is to keep the English words, "-", "_", number with units. The first pattern can retain the pure English words including the abbreviated words such as the "U.S.A". The second and third patterns are used to reserve the meaningful numbers such as "27-year-old", "34HZ", "Version9.0". In our opinion, pure numbers are meaningless to the model, and they do not provide valid information for type classification. So, we only keep the numbers that are connected to the letters (that is, numbers with units). Besides, the last pattern will preserve the words with some specified symbols such as "top-high", "ask_hn", "c++", "M&A".

After that, each retained word is subjected to part of speech restoration by lemmatization. Pronouns are reduced to singular nouns, and verbs with tenses are reduced to verb prototypes. This operation better merges the thesaurus, making the characteristics of each word more obvious.

Then, we tokenized each word as a token after restored its part of the speech. To be able to connect two related words together as a single token, we treat two connected proper nouns as a single token and turn it as the lowercase words.

In addition, we filtered out words of length 1 or length greater than 17 in this phase. Because from our perspective, a single letter cannot provide too much effective information like the simple numbers, and has little effect on classification. In addition, words that are too long in length make pattern matching take too long. A situation in which a long fuzzy word appears to cause a program to die when actually running a large data set. And considering the actual data considerations, we set the critical value to 17.

In summary, we decided to remove these two kinds of words with these length. And this step is before the synthesis of proper nouns, so the synthesized phrase is not limited by the token length.

Build Model: In this phase, the program builds both the training model and the testing dataset. First of all, the program obtains all types of classes and builds the vocabulary structure based on the number of classes. By reading all cleaned titles, if this title created at the 2018 year, the new token will be added into vocabulary and its frequency of post type of this title will be added once. At the same time, this type will record the number of its titles and the number of its words in the training dataset. Otherwise, this title will be recorded in the testing dataset.

Smoothing: The objective of this phase is to calculate the conditional probabilities of all types of all tokens in vocabulary. The program sets the value of smooth to 0.5 to ensure that each word has a probability of occurrence and uses logs to avoid the numerical underflow of the final score results. The formula of the conditional probability without log states as following:

$$P(\text{token}|\text{class}) = ((\text{frequency of token in this class}) + \text{smooth } 0.5) / (\text{total number of tokens in this class} + \text{size of vocabulary} * \text{smooth } 0.5) \quad (1)$$

Sort Vocabulary: The keys (tokens) of the whole vocabulary dictionary are sorted by alphabetical order.

Testing phase: The model built before is used to calculate the final score and test a Naïve Bayes Classifier to classify titles in their likely class in the testing dataset according to the highest score of different classes. Comparing the class given by classifier with its original class, if they are in the same class then the result will show as right. And the formula of score calculation states as following:

$$\text{Score}(\text{one class}) = \log(P(\text{one class})) + \Sigma \log(P(\text{token}|\text{class\_i})) \quad (2)$$

2.4 Result Analysis

Recalls/Accuracy, Precision, F1 measure:

```
Recalls/Accuracy, Precision, F1 measure:
 [[0.9965, 0.99515, 0.99572], [0.96333, 0.94872, 0.9548], [0.90883, 0.95766, 0.93655], [0.0, 0.0, 0.0], [0.99201, 0, 0]]
```

Fig. 4. the screenshot of recalls/accuracy, precision and F1-measure in exp.#1

Table 1. Recalls/accuracy, precision and F1-measure table

	story	ask_hn	show_hn	poll
Recall	99.65%	96.333%	90.883%	0
Precision	99.515%	94.872%	95.766%	0
F1-measure (β=0.85)	99.572%	95.480%	93.655%	0
Accuracy	99.201%			

Confusion matrix:

```
Confusion Matrix:
 [[126408, 247, 197, 0, 126852], [200, 5254, 0, 0, 5454], [411, 36, 4456, 0, 4903], [5, 1, 0, 0, 6]]
```

Fig. 5. the screenshot of confusion matrix in exp.#1

Table 2. Confusion matrix table

correct class (that should have been assigned)	classes assigned by the learner				
	story	ask_hn	show_hn	poll	total
story	126408	247	197	0	126852
ask_hn	200	5254	0	0	5454

show_hn	411	36	4456	0	4903
poll	5	1	0	0	6

Based on the above result we got, the overall accuracy of the classifier reached 99.201%, and we take this test results of this classifier as acceptable. Specifically, the recall, precision, and F1-measure of the "story" class in the four classes are the highest. But the value of the poll class is always 0. We believe that this result is related to the amount of training data for each class. In this project, the amount of training data for the "story" class reached more than 250,000, which were 20 times, 25 times, and 10,000 times the "ask_hn", "show_hn", and "poll" classes, respectively. From the above Table#1 data, the accuracy of the four categories also decreases in order according to the decrease in the amount of training data. The classification accuracy of the "poll" class is 0 because its training data set is too small, and insufficient training results in large deviations in test results. According to Table#2, the number of titles misjudged as "story" is the largest. We consider the reason is the large training set of the "story" class, and its proportion of high-frequency tokens in the vocabulary is also large. This makes the test results calculated by Naïve Bayes Classifier more prone to the "story" class. Therefore, the larger the amount of data in the training phase, the better the classifier's performance will be and the more accurate the classification will be.

During this task, since there are lots of words in the original data set are not indentible and locatable, instead of removing all of these words, we chose to only identify and store the data we consider valuable. We tried plenty of different ways to clean the dataset in order to just left the meaningful words which are useful when building the model. Unfortunately, the outcomes are not obviously different from each others'.

3. Results and analysis of the stop-word filtering experiment (exp.#2)

3.1 Brief description

In this experiment, we rebuilt the training model after removing all the stop words. Stop word as we know is some words have no meaning like 'the', 'is', 'are'. Theoretically, filtering stop words can raise the functional strength of the training model. There is no universal list of stop words in NLP research, we used a text file which the professor provided on the model.

3.2 Detailed implementation

Remove Stopword based on the cleaned data:

To be able to remove the stop word without affecting the basic cleaned data, we used deep copy function from copy library to copy the exact same data structure from the previous task.

Then, looped every single stop word in the .txt file to check if it appears in the previous cleaned data, if it is not, keep it in the training set to build the model.

Redo Exp. #1 from build model phase

3.3 Result Analysis

Recalls/Accuracy, Precision, F1 measure:

```
Recalls/Accuracy, Precision, F1 measure:
 [[0.99784, 0.99746, 0.99762], [0.98001, 0.97288, 0.97586], [0.95533, 0.97158, 0.9647], [0.0, 0.0, 0.0], [0.99557, 0, 0]]
```

Fig. 6. the screenshot of recalls/accuracy, precision and F1-measure in exp.#2

Table 3. Recalls/accuracy, precision and F1-measure table

	story	ask_hn	show_hn	poll
Recall	99.784%	98.001%	95.533%	0
Precision	99.746%	97.288%	97.158%	0
F1-measure	99.762%	97.586%	96.470%	0

(β=0.85)				
Accuracy	99.557%			

Confusion matrix:

```
Confusion Matrix:
 [[126578, 137, 137, 0, 126852], [109, 5345, 0, 0, 5454], [208, 11, 4684, 0, 4903], [5, 1, 0, 0, 6]]
```

Fig. 7. the screenshot of confusion matrix in exp.#2

Table 4. Confusion matrix table

correct class (that should have been assigned)	classes assigned by the learner				
	story	ask_hn	show_hn	poll	total
story	126578	137	137	0	126852
ask_hn	109	5345	0	0	5454
show_hn	208	11	4684	0	4903
poll	5	1	0	0	6

As can be seen from the above data, removing the stopwords from vocabulary improves the accuracy of the test. Also, for three classes with a large amount of data, removing the stopwords improves their recalls. It can be concluded that the stopwords are some meaningless words and do not have class-related features. After the stop word is deleted, the size of the data set is reduced, and the time for training the model is also reduced. In addition, leaving only fewer and only meaningful words can improve the accuracy of the classification. As a result, removing such interference information will enhance the feature strength in the original classes and optimizes the classifier performance.

Deleting stop words in NLP is not a strict rule, depending on what we are doing. For text categorization, title generation, auto-tag generation tasks, removing or excluding stop words from a given text can focus more on words that define the meaning of the text[1]. In machine translation, language modeling, text summarization, and question and answer (QA) systems, avoid deleting stop words[1]. This project is about text categorization, so deleting the stop words will optimize the results.

4. Results and analysis of the word-length filtering experiment (exp.#3)

4.1 Brief description

In this task, we filtered the original cleaned data to remove certain words, only leave the word in which length less than nine as well as plus than two. As the length of the word has nothing to do with the frequency, we expected the result would not get affected too much.

4.2 Detailed implementation

Remove word with required length based on the cleaned data: The program loops the cleaned data and retains the word whose length is less than nine as well as plus than two.

Redo Exp. #1 from build model phase

4.4 Result Analysis

Recalls/Accuracy, Precision, F1 measure:

```
Recalls/Accuracy, Precision, F1 measure:
 [[0.99131, 0.99948, 0.99604], [0.99707, 0.89959, 0.93806], [0.98511, 0.90213, 0.93517], [0.0, 0.0, 0.0], [0.99128, 0, 0]]
```

Fig. 8. the screenshot of recalls/accuracy, precision and F1-measure in exp.#3

Table 5. Recalls/accuracy, precision and F1-measure table

	story	ask_hn	show_hn	poll
Recall	99.131%	99.707%	98.511%	0
Precision	99.948%	89.959%	90.213%	0
F1-measure (β=0.85)	99.604%	93.806%	93.517%	0
Accuracy	99.128%			

Confusion matrix:

```
Confusion Matrix:
 [[125750, 578, 524, 0, 126852], [16, 5438, 0, 0, 5454], [44, 29, 4830, 0, 4903], [6, 0, 0, 0, 6]]
```

Fig. 9. the screenshot of confusion matrix in exp.#3

Table 6. Confusion matrix table

correct class (that should have been assigned)	classes assigned by the learner				
	story	ask_hn	show_hn	poll	total
story	125750	578	524	0	126852
ask_hn	16	5438	0	0	5454
show_hn	44	29	4830	0	4903
poll	6	0	0	0	6

According to the above results, after removing tokens with length less than or equal to 2 or greater than or equal to 9, the overall accuracy of the classifier is slightly reduced. For each class, the number of correct judgments for the "story" class has decreased, while the correct numbers of "ask_hn" and "show_hn" have increased. The reason for this phenomenon may be that the majority of the removed tokens belong to the "story" class, which reduces the feature of the this class. Then the proportion of the tokens of "ask_hn" and "show_hn" increases, thereby increasing their recalls. Also, deleting tokens that are too long will likely delete many proper noun phrases, which will also delete a lot of meaningful features, so it will reduce the accuracy to a certain extent. Overall, the length of the tokens is not directly related to the frequency of the tokens, so we think this factor has little effect on the performance of the classifier.

5. Results and analysis of the infrequent word filtering experiment (exp.#4)

5.1 Brief description

In this task, as the same as the last time. we filtered the original cleaned data to remove certain words but using the frequency (the times it appears in the training set) . We did it from only removing the words only exist once, then words appears less than 5 times, 15 times , 20 times, then remove the top 5%... until 25%. It is also important to mention that all the movement were based on the original dataset (finished at the task 2),

5.2 Detailed implementation

Remove word with required frequency based on the cleaned data: Based on the Experiment#1 vocabulary, the program recreates a vocabulary and stores the total number of frequencies in each class for each token. The descending order of the total number of frequencies of the new vocabulary is then

performed. After that, the tokens that meet the frequency number requirement or meet the sorted percentage requirement are retained and stored in a list. Then the cleaned titles data of Experiment#1 is processed, leaving only the tokens that exist in that list.

Redo Exp. #1 from build model phase

Redo the above two steps: Redo ten times with different frequency thresholds and record the accuracy and the recall of each class after each test phase according to the score.

Plot the performance of classifier: The accuracies and the recalls of ten times are displayed in a line chart and a random dotted line pattern is generated for each class.

5.3 Result Analysis

Recalls and Accuracy:

```
Recalls and Accuracy:
[[0.98076, 0.9693, 0.96946, 0.97015, 0.97098, 0.99962, 0.9998, 0.99986, 0.99989, 0.99991], [0.99688, 0.9978, 0.9978, 0.9978, 0.9978, 0.00403, 0.00128, 0.00018, 0.00018,
0.00018], [0.97695, 0.98837, 0.98919, 0.99021, 0.99062, 0.00612, 0.00469, 0.00347, 0.00245, 0.00143], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.98122, 0.97107,
0.97125, 0.97192, 0.97271, 0.92451, 0.92451, 0.92448, 0.92447, 0.92445]]
```

Fig. 10. the screenshot of recalls and accuracy in exp.#4

Table 7. Recalls and accuracy table

	1	5	10	15	20	5%	10%	15%	20%	25%
story recall	98.076 %	96.930 %	97.015 %	97.098 %	97.098 %	99.962 %	99.980 %	99.986 %	99.989 %	99.991 %
ask_hn recall	99.688 %	99.78 %	99.78 %	99.78 %	99.78 %	0.403 %	0.128 %	0.018 %	0.018 %	0.018 %
show_hn recall	97.695 %	98.837 %	98.919 %	99.021 %	99.062 %	0.612 %	0.469 %	0.347 %	0.245 %	0.143 %
poll recall	0	0	0	0	0	0	0	0	0	0
accuracy	98.122 %	97.107 %	97.127 %	97.192 %	97.271 %	92.451 %	92.451 %	92.448 %	92.447 %	92.445 %

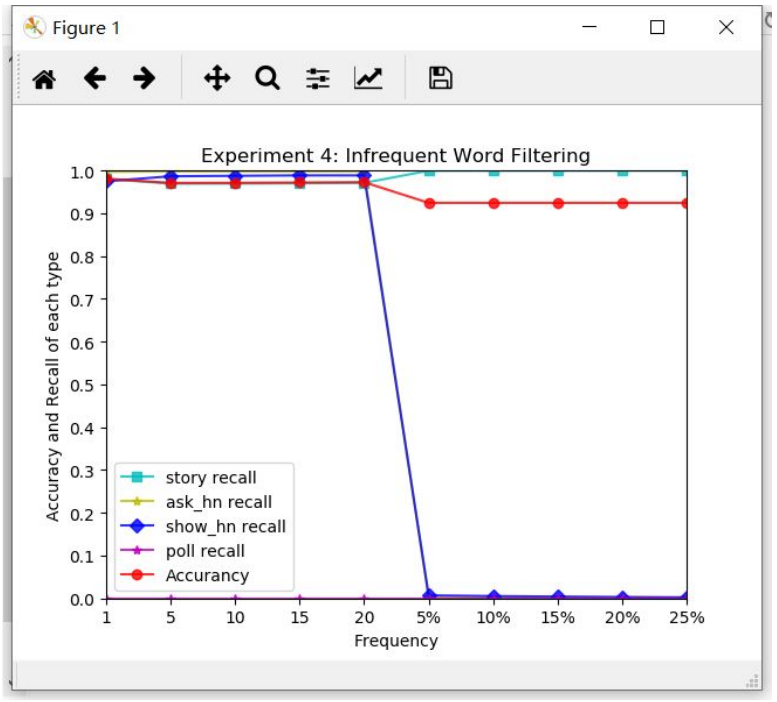

Fig. 11. the line chart of recalls and accuracy in exp.#4

Based on the result showing above, with the number of removing word increasing, the accuracy of the testing result decreases in general, even some of the types got increased. Specifically, In the stage of removing the token with a frequency of 1 to 20, the recall of the "story" class has a slight drop. At the same time, the recalls of "ask_hn" and "show_hn" have a little rise. Overall, the accuracy rate has not changed much at this stage. However, there is a typical decrease after we removed the top 5% of the most frequently appear works. This is especially shown in "ask_hn" and "show_hn" classes and their values stay close to zero. Thus, removing the high-frequency token has a huge impact on these two types. As we understand, the reason for this phenomenon is probably that the amount of test data of these two classes is much smaller than that of the story class, and the training data of these two classes are concentrated on the high-frequency tokens. Therefore, the frequency of the token has a greater influence on the class with a smaller amount of training data.

6. Results and analysis of the smoothing experiment (exp.#5)

6.1 Brief description

In this particular experiment, we rebuilt the model for eleven times with different smoothing number from 0 to 1 with difference as 0.1.

6.2 Detailed implementation

Redo Exp. #1 from build model phase with the specific smooth value

Repeat the above step eleven times: use 0.0 to 1.0 as the value of smooth.

Plot the performance of classifier: The accuracies and the recalls of eleven times are displayed in a line chart and a random dotted line pattern is generated for each class.

6.3 Result Analysis

Recalls and Accuracy:

```
Recalls and Accuracy:
 [[0.95668, 0.97286, 0.98438, 0.99106, 0.99452, 0.9965, 0.99789, 0.99859, 0.99909, 0.99932, 0.99951], [0.72791, 0.99212, 0.98973, 0.98478, 0.97506, 0.96333, 0.94793, 0.92409,
 0.90191, 0.87642, 0.84892], [0.63451, 0.96818, 0.96084, 0.94758, 0.93229, 0.90883, 0.88905, 0.86274, 0.83663, 0.8142, 0.78646], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
  0.0, 0.0], [0.93603, 0.97341, 0.9837, 0.98921, 0.99148, 0.99201, 0.99197, 0.99073, 0.98937, 0.98778, 0.98587]]
```

Fig. 12. the screenshot of recalls and accuracy in exp.#5

Table 8. Recalls and accuracy table

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
story recall	95.668%	97.286%	98.438%	99.106%	99.452%	99.65%	99.789%	99.859%	99.909%	99.932%	99.951%
ask_hn recall	72.791%	99.212%	98.973%	99.478%	97.506%	96.333%	94.793%	92.409%	90.191%	87.642%	84.892%
show_hn recall	63.451%	96.818%	96.084%	94.758%	93.229%	90.883%	88.905%	86.274%	83.663%	81.42%	78.646%
poll recall	0	0	0	0	0	0	0	0	0	0	0
accuracy	93.603%	97.341%	98.37%	98.921%	99.148%	99.201%	99.197%	99.073%	98.937%	98.778%	98.587%

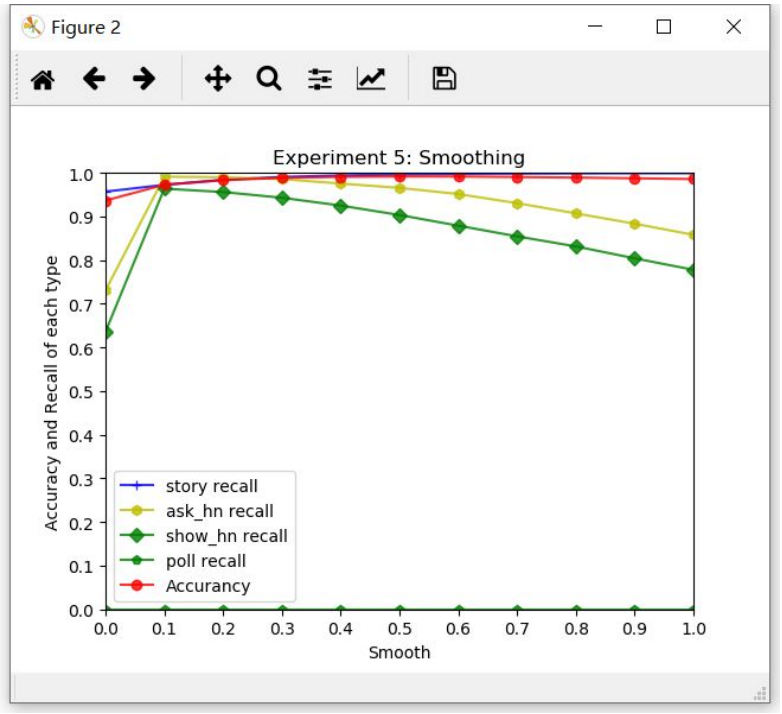

Fig. 13. the line chart of recalls and accuracy in exp.#5

Based on the result showing above, the accuracy got its peak when we picked 0.5 as the smoothing value, and it got the lowest point when smoothing value is 0. Even though some classes such as "ask_hn" and "show_hn" got the highest point as 0.1, but in general 0.5 should be the most suitable number. The meaning of smoothing to address the situation that a word token doesn't exist in this particular class so we add certain times more of each word in the training set to make sure every class contains this word. Consider about this motivation, it is necessary to add a smooth value when calculating the conditional probability for each token and it is important to pick smoothing value not too big or too small, otherwise it will affect result accuracy eventually.

7. Compare and discuss the results of the four experiments

Comparing the former experiments we did in this project, we notice that there are multiple factors will affect the testing result for certain degrees. From our perspectives, among all of them, the most significant one is how we treat the extreme low and high-frequency tokens. On the other hand, this factor is also influenced by the size of the training data.

The frequency of the token is a direct factor affecting the score of the Naïve Bayes Classifier. As shown in the results of Experiment 4, the effect of removing low-frequency tokens on the accuracy of each class is much smaller than the effect of removing high-frequency tokens. Especially the reaction of "ask_hn" and "show_hn" is particularly obvious. The "story" class with a large cardinality in the training set is hardly affected. Therefore, the frequency of tokens is a key factor for classes with smaller training data sets.

Furthermore, there is an interesting mistake we made and turned out we got the principle from it. When we were doing Experience 4 which is removing high-frequency words and low-frequency words step by step, we misunderstood the task and used the frequency of each class separately. After communicating with the instructor, we noticed that what we need is the total frequency which makes much more sense because that will implicate the features of each class. To be more clear, the high-frequency words appear everywhere and low-frequency words are just extreme values.

Among the above experimental results, the highest overall accuracy is Experiment 2. It can be seen that removing stop words helps the classifier accuracy most. Except for polls with too little data, the correct number of the other three classes have all improved, which is not found in several other experiments. Because these stopwords have no practical meaning, they sometimes interfere with the judgment of the

classifier. Removing these words can make the features of the original classes more obvious and help the classifier to make judgments.

For calculating the conditional probability of each token, it is necessary to add a smooth value, because this can guarantee that each token in the vocabulary appears in each class. From the result data of Experiment 5, it shows that without smooth operation, the accuracy of each class (especially "ask_hn" and "show_hn" with small training set) is low. For smooth values from 0 to 1, we think 0.5 is the best, and the overall performance of the classifier is the best at this time.

Additionally, we think that the least help for accuracy in the above experiments is the length of the token. Deleting tokens that are too long will delete a lot of phrases, which will also delete a lot of meaningful features, so this will reduce a little accuracy. Deleting only words that are too short, such as 1 and 2, may help the result because this type of word generally has no meaning. However, we have already removed the token of length 1 when cleaning the data, so the accuracy of Experiment 3 is less than that of Experiment 1.

It is worth mentioning that the results of the "poll" class in each of the above experiments are 0, but this is understandable. Because compared to the other three classes, the training data set of this class is too small, which makes it a much smaller proportion in vocabulary than the other three classes. Therefore, in order to improve the classification accuracy of this type, more training data set of this type should be offered.

Consequently, a large number of training sets can ensure the stability of classifier accuracy and make the test results less affected by various aspects. In general, we do find out the basic principle to improve the functionality of the training model, responsively, the accuracy of the testing result, is trying the best to implicate the strongest feature of each class. This shows when we removed all the stopwords, and got the highest accuracy since stop word appears in all the classes. In other words, they are not able to express any feature of any class.

8. Future work

If we have more time and energy, we will work on three aspects in natural language processing (NLP).

Firstly, we would like to study how to clean up data can make it more meaningful and more conducive to building powerful training model. And we have been thinking about this issue in the coding phase. We want to study later whether to restore words with symbols or garbled words to the correct words, or to find and correct common printing errors and spelling errors, or to translate other languages into English. This learning is good for us to understand how various types of data affect the performance of the model.

Additionally, it is meaningful to study how to conduct NLP research under the condition that the amount of training data is small. Just like the "poll" class in this project, its has only 25 training data, which is very small compared to the other three types of tens of thousands of data. Due to the insufficient amount of data in the training phase, the recall of this category has been 0 in the testing phase. Moreover, the small amount of training data will result in test results being susceptible to other factors. For example, after removing high-frequency words in Experiment 4, the recalls of "ask_hn" and "show_hn" which are relatively small compared to the "story" class data decline significantly. However, the recall of the story class with a huge amount of training data has increased slightly. So, we want to study how to perform NLP with low training data.

Furthermore, through the search of online resources, we found that Python has a lot of libraries about NLP. After this project, we want to learn more about the methods to NLP such as the feature extraction, feature selection, distributed vector of the sklearn library to build the model.

References

1. Deleting stop words and text normalization using NLTK and spaCy in Python, https://panchuang.net/2019/08/31/nlp-essentials-removing-stopwords-and-performing-text-normalization-using-nltk-and-spacy-in-python/, last accessed on 2019/08/31.

2. Extracting Information from Text, https://www.nltk.org/book/ch07.html, last accessed on 2019/09/04.