**COMP 6721 Project 1 Report**

Authored by Junwei Zhang

**ID** - 40050122 [zhangjw248@gmail.com](mailto:zhangjw248@gmail.com)

1. Introduction
   1. Purpose of the project

This project is a natural language processing based project. We using our processed text from the training set to build our probabilistic model and use this model to implement and test a Naïve Bayes Classifier to classify ‘Post Type’ into their likely class in test dataset.

For the training dataset and test dataset, we use dataset of Hacker News fetched from Kaggle.

We separate training and testing dataset according to years, use 2018 as training and 2019 as testing.

Meanwhile, using four types in ‘Post Type’ column as the type we classified – ‘story’, ‘ask\_hn’, ‘show\_hn’ and ‘poll’.

After we build our training model and implement our classified on test dataset, we output test file with classification results. With the results, we need to calculate accuracy, precision, recall and F1-measure for each class and analyze them.

Finally, allowing change some parameters to see the performance change and analyze them in different situation, like delete stopwords or words length filter etc.

* 1. Preprocessing technical details

In order to build training model we use, we need to preprocess the text from the raw dataset.

For me, because in training phase testing classification phase we only about to use ‘Title’ and ‘Post Type’ columns, I fetch these two columns as numpy arraylist to analyze to reduce time complexity.

### **Lowercase and lemmatize.** The first thing is lower() text in list for each line of row, then use nltk word\_tokenize() split each line of text.

Second, design my grammar to add label to each two words noun or proper noun or nouns and also add label to possessive case of nouns like following:

grammar = r"""

NP: {<NNS|NN><NN|NNS>}

NP\_P: {<NN|NNS><POS>}

"""

Then, use pos\_tag() identified part of speech. With above method we defined, now we can use this method to lemmatize each line of text in the ‘Title’ column and add the words after lemmatization into a list which replace the previous line in numpy array. Now I just encapsulate the method for later use.

### **Separate the text base on their class.** I separate the text into four, based on their ‘Post Type’, for each I build a dict to count what and how many words in each type and its frequency. I can also count each Type probability P(story), P(ask\_hn), P(show\_hn), P(poll) using separation. Finally, I using lemmatize() method designed to lemmatize each part of class and count the words into a dict, which label as total\_voca and print its words as our baseline-vocabulary.

1. Results analysis
   1. Results and Analyze Baseline-model (exp. #1)

After preprocessing the text, we create a probability model based on that, with smoothed = 0.5 and use log10 count each words probability in case of underflow of number, we create a file with format on the requirement to output our model.

Using this model implement naïve bayes classifier to classfy the tesing dataset, and output a file with the given format. After several time testing and trying I checked from my vocabulary with some words are not helping with classification and I delete them from the vocabulary and here are the words that not helping and fit:

Any words startswith symbol, any words constitute with symbol not single number or character,

Any words is digit, any words startswith character plus space, any words startswith character plus full stop, any words . I delete it as not seeing them as word and not helping with classification because basically they all appears in story class, and its not helping me to classify the rest class.

Apart from that, I delete some words like any words startswith f-, or b-, or c-, or car plus space etc.

Simply because it is meaningless and affect the accuracy because appear to much in story and test case.

For the result, the following is the confusion matrix:

**Table 1.** confusion matrix for exp. #1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Correct Class** | **Class assigned by the learner** | | | | **Total** |
|  | story | ask\_hn | show\_hn | poll |  |
| story | 126841 | 3 | 0 | 0 | 126844 |
| ask\_hn | 4765 | 689 | 0 | 0 | 5454 |
| show\_hn | 4047 | 0 | 856 | 0 | 4903 |
| poll | 6 | 0 | 0 | 0 | 6 |

The following are the four indication-accuracy, precision, recall and F1-measure for each class type:

*Accuracy(for total system)* = 126841 + 689 + 856/126844 = 93.57%

*Recall(for story) = 126841/126844 = 99.99%*

*Recall(for ask\_hn) = 689/5454 = 12.63%*

*Recall(for show\_hn) = 856/4903 = 17.46%*

*Recall(for poll) = 0/6 = 0%*

*Precision(for story) = 126841/126841 + 4765 + 4047 + 6 = 93.50%*

*Precision(for ask\_hn) = 3/689 + 3 = 99.57%*

*Precision(for show\_hn) = 0/856 = 100%*

*Precision(for poll) = NaN*

*F1-measure(for story) = 96.64% (function as follow)*

*F1-measure(for ask\_hn) = 22.42% (function as follow)*

*F1-measure(for show\_hn) = 29.73% (function as follow)*

*F1-measure(for poll) = NaN (function as follow)*

Because there are lot of training dataset coming from story, the number of words in my dictionary story are very high. That causes for each word wi, the probability P(wi / story) is possibly been higher than other three class.

Also, since most document are coming from story, the probability P(story) could also been very high compare to the rest of the class it helps when classify type story, as you can see the recall(story) is very high, but because the recall(ask\_hn), recall(show\_hn), recall(poll) is not very high, so the precision of story is some level neutralized.

The recall for class type ask\_hn, show\_hn is really low, but their precision is really high, which means when system classify the type to ask\_hn or show\_hn the percent of the correctness is great.

For class type poll, because of the document belong to poll is too small, only 25 doc belong to poll and only 106 words belong to it. For me, it nearly imporssible to classify correct.

Since in the F1-measure function:

*F1=*

In F1-measure function I define the β = 1, cause it take precision and recall as same importance.

But since we don’t have precision in poll class, we don’t have F1-measure in poll class either.

* 1. Results and Analyze stopwords-model (exp. #2)

For this task, since we remove stopwords from the doc, I get improvement for my system.

Accuracy for total = 95.42%

Recall(story) = 99.99%

Recall(ask\_hn) = 41.80%

Recall(show\_hn) = 36.71%

Recall(poll) = 0%

Precision(story) = 95.28%

Precision(ask\_hn) = 99.82%

Precision(show\_hn) = 99.94%

F1-measure(story) = 97.58%

F1-measure(ask\_hn) = 58.93%

F1-measure(show\_hn) = 53.70%

F1-measure(poll) = NaN

This may be caused by these stopwords are most frequently words. Since we have story class doc the most, the frequency of these words are also very high in story dictionary. That cause the score(wi/story) of these word are very high and easily disturbed other type classified.

* 1. Results and Analyze wordlength-filter-model (exp. #3)

For this task, we somehow slightly improve the accuracy but not for whole performance. The results is as follow:

Accuracy for total = 93.75%

Recall(story) = 99.99%

Recall(ask\_hn) = 2.3%

Recall(show\_hn) = 33.94%

Recall(poll) = 0%

Precision(story) = 93.67%

Precision(ask\_hn) = 99.21%

Precision(show\_hn) = 100%

F1-measure(story) = 96.73%

F1-measure(ask\_hn) = 4.5%

F1-measure(show\_hn) = 50.68%

F1-measure(poll) = NaN

As we can see the recall for type ask\_hn is largely decrease, that because of the most feature take apart ask\_hn from other type is the word ‘ask’ and ‘hn’.

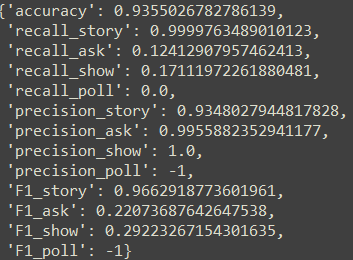
Since ‘ask’ pos\_tag in part of speech is ‘JJ’-adjective and ‘hn’ is ‘NN’-noun. I didn’t combine them together and that makes the word length become 3 and 2, which the word ‘hn’ is deleted. Then we lost one of the most feature to classified the type ask\_hn.

Because in type show\_hn, the most feature word can be put together (pos\_tag both ‘NN’-noun) so the recall(show\_hn) is improved largely.

* 1. Results and Analyze infrequency-filter-model (exp. #4) part one

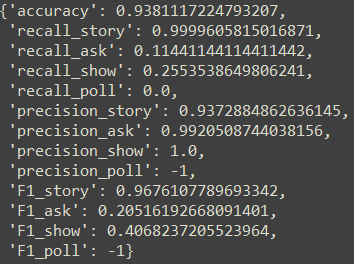
Results of the performance is as follow:

Freq1:



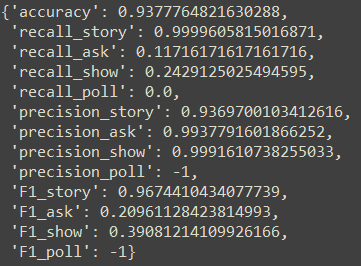
**Fig. 1.** Accuracy, recall, precision, F1-measure for four class type for remove frequency 1 words (‘-1’ means ‘Nan’).

Freq5:



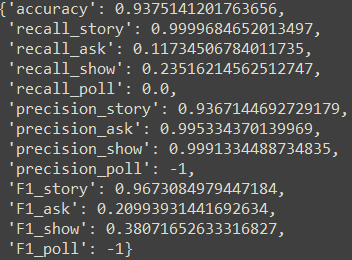
**Fig. 2.** Accuracy, recall, precision, F1-measure for four class type for remove frequency 5 words (‘-1’ means ‘Nan’).

Freq10:



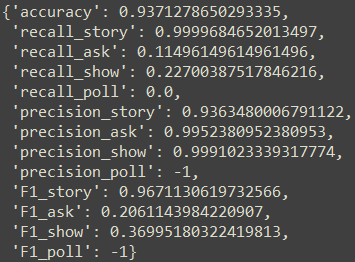
**Fig. 3.** Accuracy, recall, precision, F1-measure for four class type for remove frequency 10 words (‘-1’ means ‘Nan’).

Freq15:

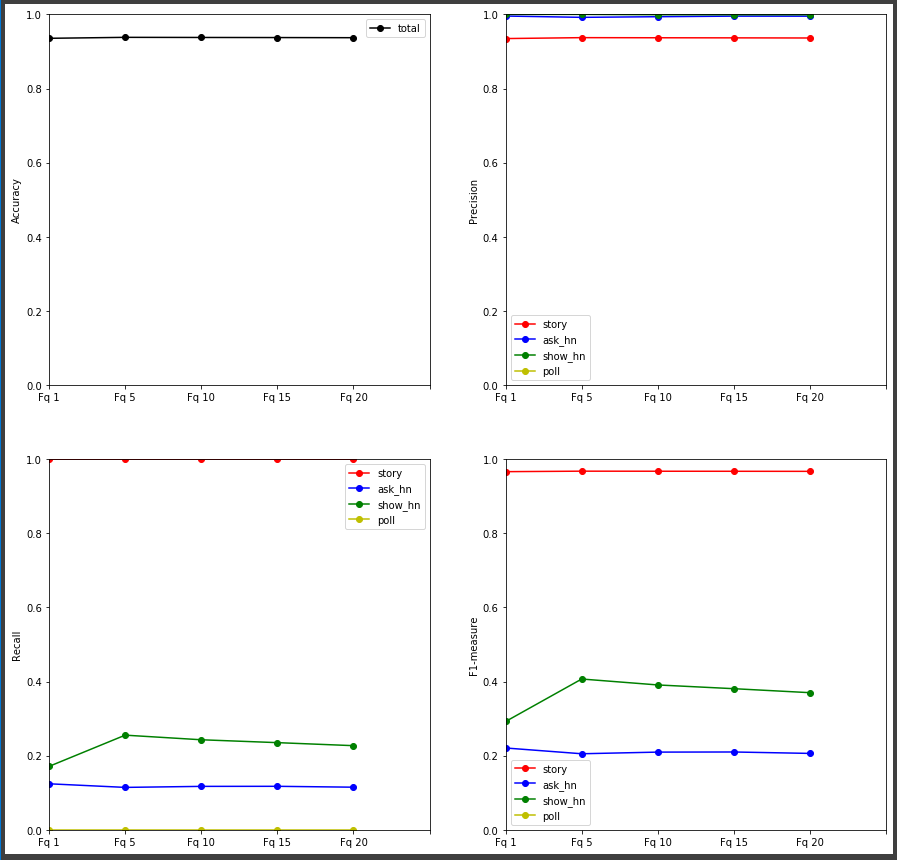


**Fig. 4.** Accuracy, recall, precision, F1-measure for four class type for remove frequency 15 words (‘-1’ means ‘Nan’).

Freq20:



**Fig. 5.** Accuracy, recall, precision, F1-measure for four class type for remove frequency 15 words (‘-1’ means ‘Nan’).



**Fig. 6.** Plot performance for accuracy, recall, precision, F1-measure for 4 types removed words frequency from freq 1 to freq 20 (class type represent in different color of the line, see legend)

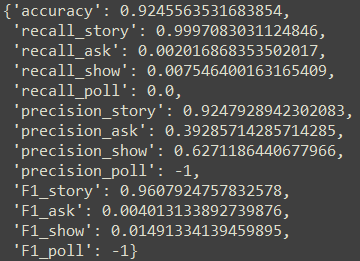
We can see from the performance, when delete frequency less or equal than 5 in the dictionary, we will have the best performance, partly because most of these low frequency words coming from story dict, and affect the identification for other class type, but not too high to delete the feature word in ask\_hn, show\_hn, poll class.

From above we can see the degrading performance coming from fq5, but still better than fq1, maybe lots of feature words in other three class are frequency 1 word. So the best way to improve the system is delete freq 5 words in this case.

* 1. Results and Analyze infrequency-filter-model (exp. #4) part two

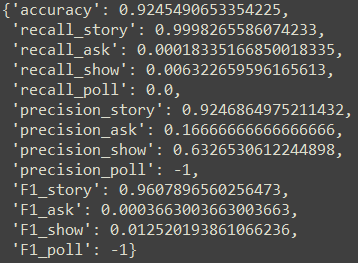
For infrequency filter model part two, I gradually remove the top5%, top10%, top15%, top20%, top25%, frequency word in the original dictionary. The results is like following:

Top5:



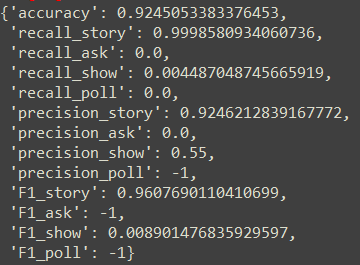
**Fig. 7.** Accuracy, recall, precision, F1-measure for four class type for remove top5 % words (‘-1’ means ‘Nan’).

Top10



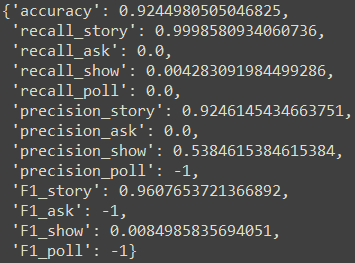
**Fig. 8.** Accuracy, recall, precision, F1-measure for four class type for remove top10 % words (‘-1’ means ‘Nan’).

Top15



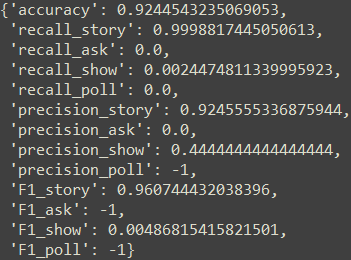
**Fig. 9.** Accuracy, recall, precision, F1-measure for four class type for remove top15 % words (‘-1’ means ‘Nan’).

Top20



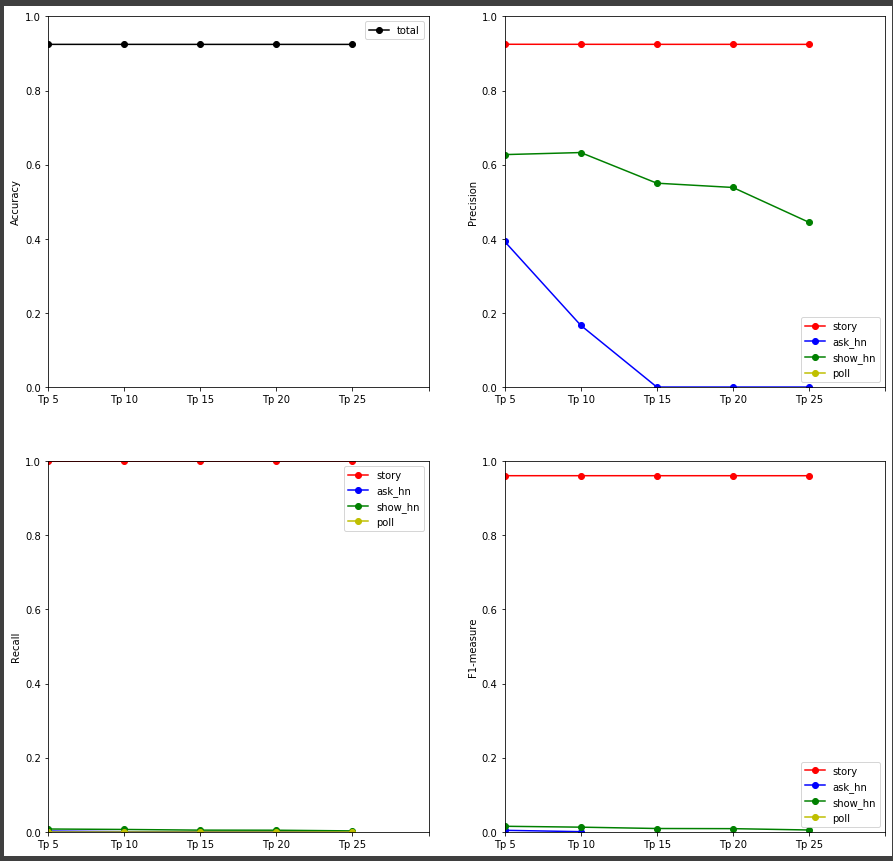
**Fig. 10.** Accuracy, recall, precision, F1-measure for four class type for remove top20 % words (‘-1’ means ‘Nan’).

Top25



**Fig. 10.** Accuracy, recall, precision, F1-measure for four class type for remove top25 % words (‘-1’ means ‘Nan’).

Performance

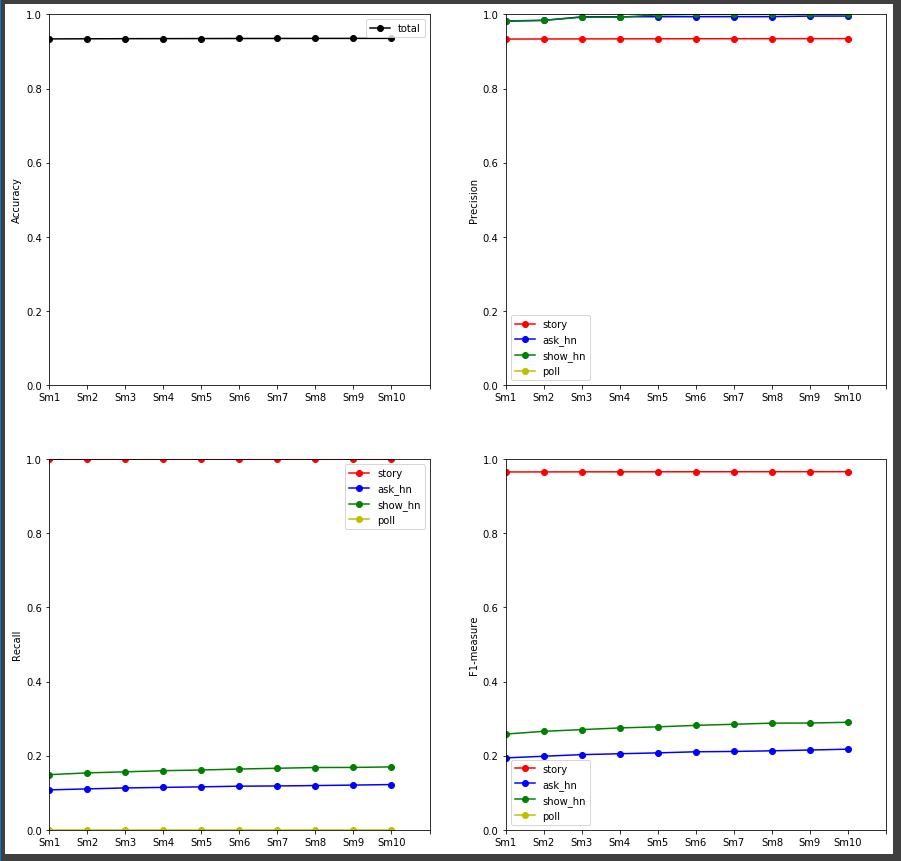


**Fig. 11.** Plot performance for accuracy, recall, precision, F1-measure for 4 types removed top5 to 25 % words (class type represent in different color of the line, see legend)

For delete infrequency word part 2, it affect the most of my system, the recall for ask\_hn, show\_hn, poll are both almost zero from the beginning, the precision of ask\_hn and show\_hn are degrading rapidly since the beginning, most likely reason for that is the feature words in both two classes are deleted in the beginning and cause the recall for story type is nearly 100% for each situation.

* 1. Results and Analyze smoothing-model (exp. #5)

I am only going to show the performance here, because the difference between them are quite small.



**Fig. 12.** Plot performance for accuracy, recall, precision, F1-measure for 4 types changing smoothing gradually from 0.1 to 1.0 in steps of 0.1 (class type represent in different color of the line, see legend)

For gradually changing smoothing value we can see the performance are getting better with the increasing. In this system, the best smoothing value is 1.0 with no doubt.

Although the accuracy is steady, the recall and precision for each class are getting better.

1. Comparing analysis

As I showed in above, the most affected experience of the four experiment are stopwords experiment and top 5-25% filter experiment.

The previous one would enhance the performance of the system, improve not only accuracy but also largely enhance recall for three type class, and slightly enhance for precision.

Unlike the stopwords experiment, the top filter experiment is nothing but negative affect to my system. It was degrading the recall of four classes to nearly zero from the beginning, and largely decreasing the precision for about 40%. But the interesting thing is the accuracy is only decreasing 3% compare to stopwords experiment, which because large training and testing dataset are story and it nearly unaffected its accuracy for classified.

The smoothing experiment is the most stable phase, and the performance is increasing with the smoothing value increase. The reason is since lot of words in total dictionary are not show on show\_hn, ask\_hn and poll type dictionary, the smoothing value will increasing each word probability in those three types and let them more easily to score up to the biggest one to classified as their own type.

The freq 1-20 filter is more aggressive, it enhance performance rapidly and gradually decreasing the performance, but the performance are always better than the first case, so in my system the best performance is near freq5 around but need to be more testing.

For word length experiment, it enhance accuracy but badly decreasing the recall of ask\_hn type, that is because the feature word for ask\_hn type is inside the length we filtered. So, before we do length filtering, we need to investigate the feature word for each type is between which area to reach the best performance.

1. Future Learning

After this project, I was really interested in natural language processing, about how to define the suitable grammar to combine the word, how to use part of speech to generate the best word not meaningless.

That is for the text organization, and for classifier system, how to find the unrelative word that would affect your system performance is my most interested part.

Because for me, the way is to find each classified types feature and keep it when its possible, for the type only take small probability in the whole document, it is crucially important.

But, sometimes there is no notable feature to find or it is taking too much time to looking for these feature, so how to classified automatically correct and improve the performance by its own is the most importance part.

For me that’s the excited part of machine learning!