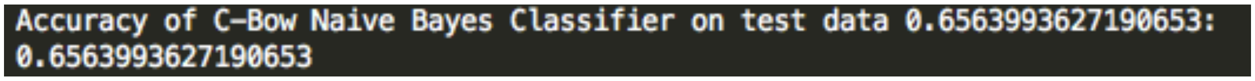
Project Report

Experiment 1:

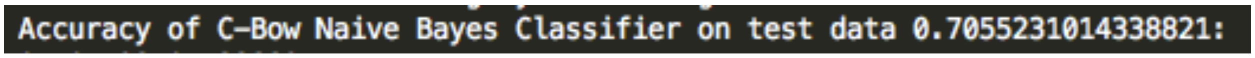
As the common preprocessing step across all three models, first alphanumeric strings were selected. Then the tokens that are not alphabetic were eliminated and stop words were removed. These steps provided the basic preprocessing. As additions, I experimented with a lemmatizer and an "improved" lemmatizer. The removal of stop words decreased each model performance by ~%1.5. The addition of lemmatizer did not give any statistically significant improvement on any model. This is because the WordNetLemmatizer can make a lot of mistakes in lemmatizing given the position tag of the word is not provided. What I did was to first get position tag of each word, and then use this position tag as an input when lemmatizing. Although this improved C-Bow model by a lot and decreased vocabulary size by 7500, the extra preprocessing increased the overall run time from under a minute to 30-40 minutes. Therefore I commented out the lemmatizing part in final version of the code.
Calculation of conditional probabilities was very similar to the homework description, but I created a new token <UNK>, which represented all the unknown (not seen) words in a category. This decreased space complexity by a lot since the model was able to represent all unseen words in a category as a single token instead of size (vocab_size - words in category). All the conditional probability calculation across three models is done under the function log_probabilities_for_NB (), albeit differently to model specifications. The training and testing of all models are nearly identical.

All models performed better on random data, due to a smaller number of vocabulary size.
C-Bow Model:
By Date:



```
Accuracy of C-Bow Naive Bayes Classifier on test data 0.6563993627190653:
0.6563993627190653
```

Random Data:



```
Accuracy of C-Bow Naive Bayes Classifier on test data 0.7055231014338821:
```

The underlying idea behind this multinomial Naïve Bayes Model is that "if a single occurrence of a word is a good clue that a document belongs to a class, then 5 occurrences should be even more predictive." Thus the important question here is whether local document frequency is actually the right way to use word frequency information to enhance classification. In some cases were document is long (found especially in religious texts) and C-Bow Model performs very well as there is much more opportunity for words to be emphasized by being used many time. But medium to short sized documents, especially in emails that are spread around different categories, this model struggles. Also a serious problem with this model is the fact that repeated occurrences of the same word within a document might not be independent, which may violate our independence assumption.[1] It should be said that on this dataset

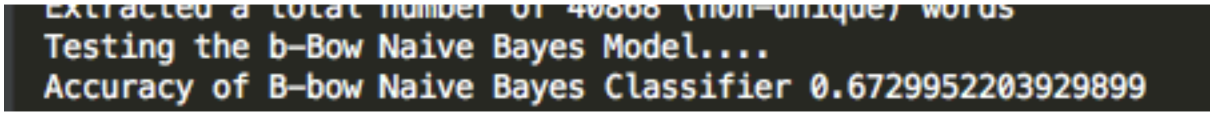
- [1] \Schneider, K.-M. (2003). A comparison of event models for Naive Bayes anti-spam e-mail filtering. EACL'03.
- Pavlov, D., & Balasubramanyan, R., et al. (2004). Document preprocessing for naive Bayes classification and clustering with mixture of multinomials. KDD'04, Seattle, Washington.

achieves statistically similar performance with B-BoW model with a test set that includes mostly short to medium range documents, because Multinomial Model is much more expressive than its binary version. To improve on this model, a new approach to use of local frequency in classification might be needed. An example would be to give less weight or remove words whose frequency is less than some predetermined threshold.
The lemmatizer improved the model performance by only %1.

B-Bow Model:

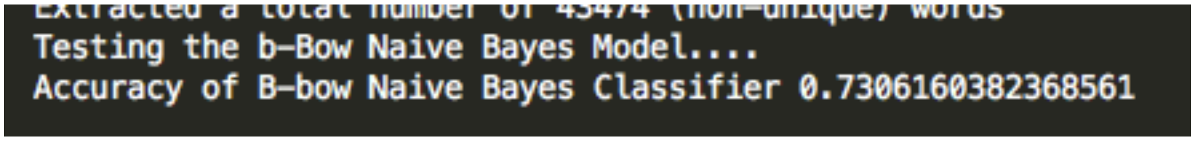
This model performed slightly well than the rest, given its great performance on a lot of emails that were short documents. It made mistakes on long religious texts that C-Bow model successfully classified. Just like the other models, it performed ~%7 better on randomly selected data. Also since I made the feature selection choice of removing numeric tokens, it made a lot of mistakes regarding all the comp and hockey categories. The overall better performance of B-bow shows the absence and presence of a word turned a to be a more important feature than local word frequency for our real world data set.

By Date:



```
Extracted a total number of 40868 (non-unique) words
Testing the b-Bow Naive Bayes Model....
Accuracy of B-bow Naive Bayes Classifier 0.6729952203929899
```

By Random:

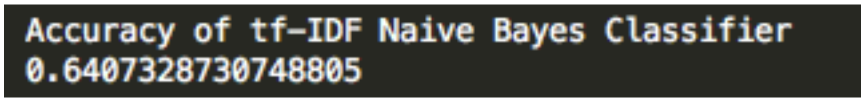


```
Extracted a total number of 43474 (non-unique) words
Testing the b-Bow Naive Bayes Model....
Accuracy of B-bow Naive Bayes Classifier 0.7306160382368561
```

Tf-IDF Model:

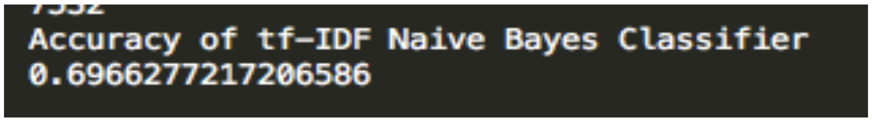
I expected this model to perform better from the rest, but I believe it requires some additional feature selection methods (information gain, document frequency threshold) to show the effectiveness of this model compared to binary and multinomial models. Another reason might be that the conditional independence of the model is violated by real world data that we are using.

By Date:



```
Accuracy of tf-IDF Naive Bayes Classifier
0.6407328730748805
```

Random Data:



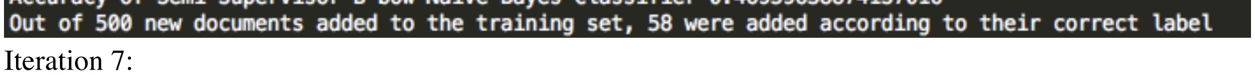
```
7532
Accuracy of tf-IDF Naive Bayes Classifier
0.6966277217206586
```

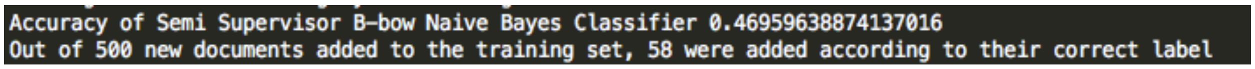
Experiment 2:

-

For experiment 2, inside the B-Bow Naïve Bayes Class, I wrote a function called build_semi_classifier_classifier, which got %5, %10 or %50 of training data given the input and created a new training set. The remaining documents were put under the testing set. Then the model was trained with this new training set (using a different function) and test accuracy was measured. After this, the model was updated by using top-k or threshold filter. The same xset of steps was repeated under the function iterate_semi_supervisor_function until no elements remained in the test set for top-k filter, and until 5 iterations were reached in threshold filter. Below I analyze results of these two filters separately.

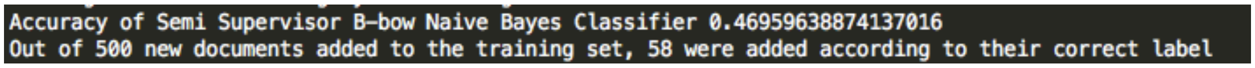
Top-K Filter:

Initially I extracted 500 test documents with maximum conditional probability. The problem with this approach, as cited by other students as well, is that the maximum probabilities corresponded to documents with least number of words, regardless of whether the maximum probabilities of "shorter" documents were a more confident measure of predicted label compared to longer documents. For example, with only using %50 of training data most of long religious documents were labeled correctly in the first iteration. Not only were they labeled correctly, but also the difference between their probability and the probability of the "second best label" was much larger than some of the shorter documents that were selected by the top-k filter. To prove this, I tracked the number documents added to the training set according to their correct label. In first iteration, only ~130-150 documents were added with their correct label. However, I must also say that number of documents added to training set according to their correct label showed a steady increase. During the last two iterations (out of 12), the documents added with correct labels corresponded to %80.
This might show that Bag of Words representation does not suffer from confidences on different scales, but it should be noted that training with %50 of training data builds a good "base" for our model (~%63 test accuracy), and initial incorrect additions to our training set does not affect performance that much (or the damage might be limited to a single category, which in my case was hockey). I though the problem of scaling would show itself when we start with %10 of training data, which builds a model that is much more vulnerable to additions of documents with incorrect labels. That did not exactly turn out to be correct. In first iteration our model starts with %52 test accuracy and only 64 documents with correct labels are added to the training set. What happens is that these incorrectly labeled documents decrease the accuracy by a little margin in first couple of iterations, but as soon as the percentage of correctly labeled added documents surpasses our test accuracy, our model starts learning. No matter what type (short-long) of documents we add initially to our training set, the model will soon show that some top-k documents will be ~%54 correctly labeled *at some point*. At this tipping point the model will start learning from the semi-supervised procedure and regain initial losses in test accuracy. An example is shared below: On iteration 7, 271 out 500 documents were added with correct labels (%54) which surpasses the test accuracy of our model. From this point on, the test accuracy of the model keeps increasing until termination. The number of k also plays a role. Selecting k lower than 200 means we will learn slower, and not learn much since it takes the model longer to reach the tipping point.
Iteration 1:

```
Accuracy of Semi Supervisor B-bow Naive Bayes Classifier 0.46959638874137016
Out of 500 new documents added to the training set, 58 were added according to their correct label
```
Iteration 7:



```
Accuracy of Semi Supervisor B-bow Naive Bayes Classifier 0.46959638874137016
Out of 500 new documents added to the training set, 58 were added according to their correct label
```

Final Iteration:

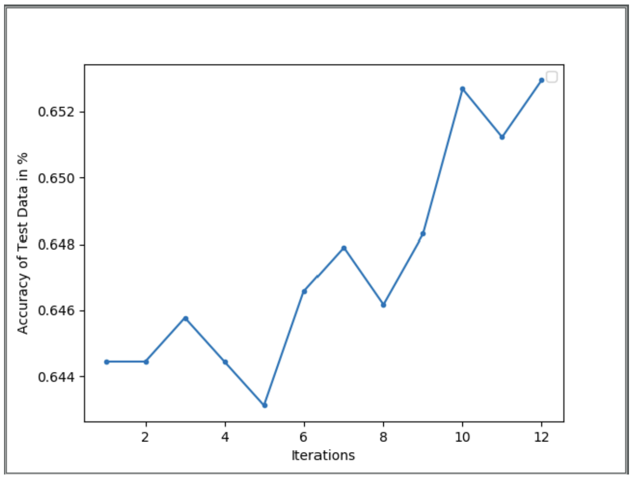


```
Accuracy of Semi Supervisor B-bow Naive Bayes Classifier 0.46959638874137016
Out of 500 new documents added to the training set, 58 were added according to their correct label
```

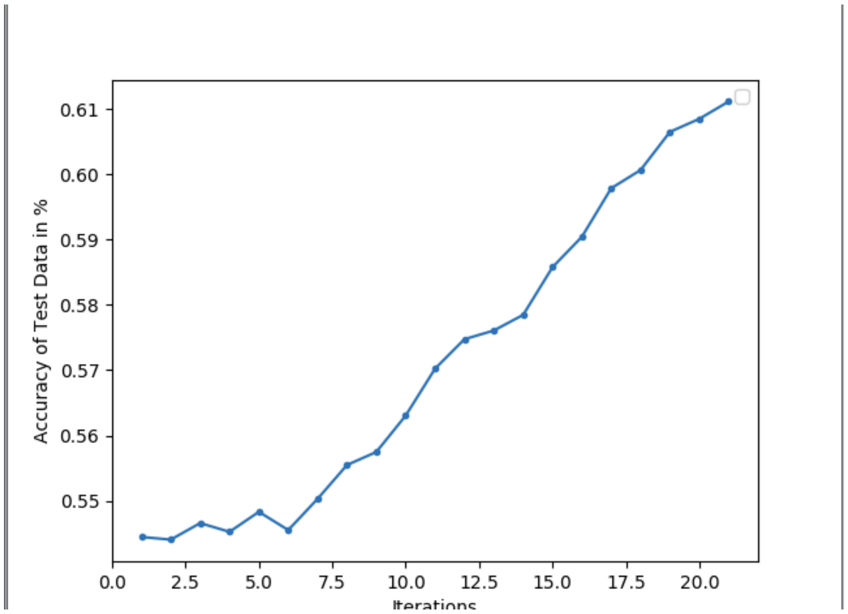
I tied k values of 200 and 500, and both converge to similar final results, although with k=500 learning starts sooner, thus the model learns more and improves test accuracy around ~%2 more.

Results:

P = 50, k = 500. We do not see much learning, since the model is already close to its optimal performance.

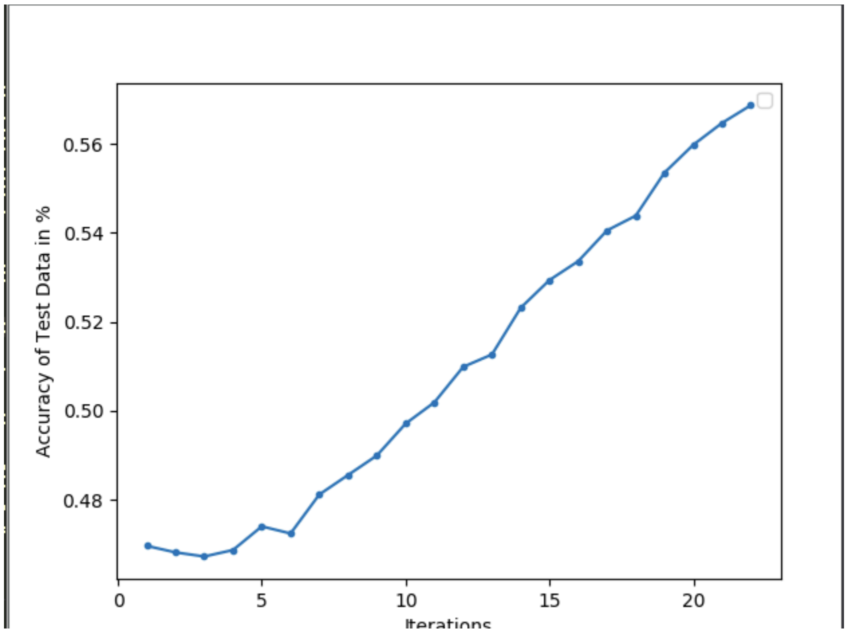


P = 10 k= 500



P = 5 k = 500

We can observe the initial dip until iteration 7 and then the consistent improvement of model until final iteration.

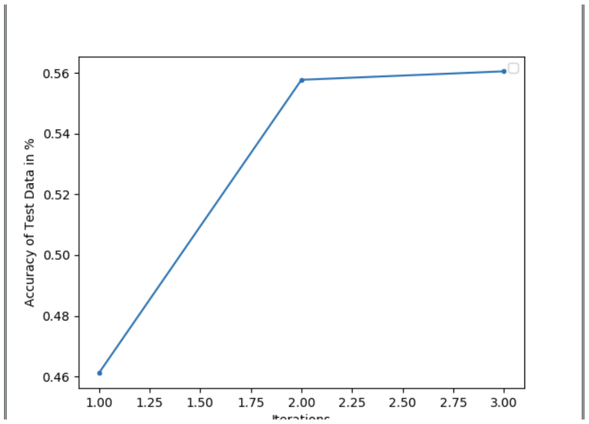


Threshold:

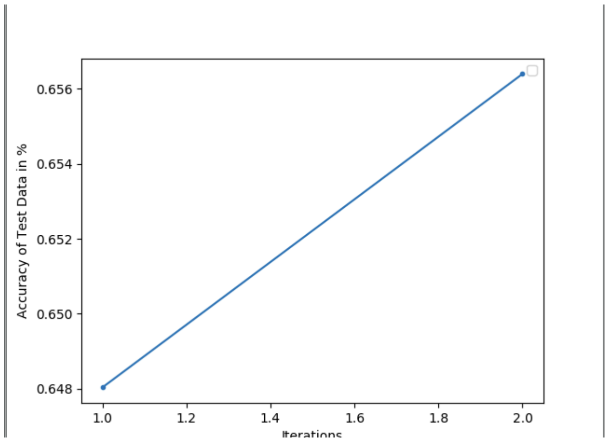
Threshold filter achieves similar performances with top-k, but much faster. Even if I select a very low threshold value that excludes a lot of shorter documents, the recalculated document value is enough to pass the filter.

For p = 5 threshold = -500

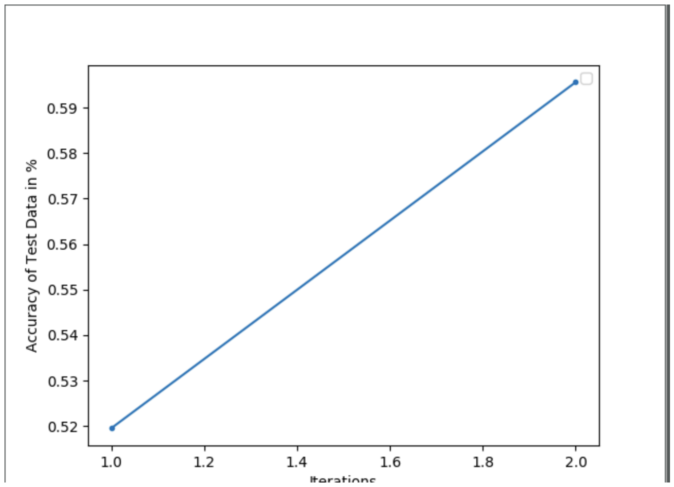
We see that the threshold filter achieves the same test accuracy after only 3 iterations.



For p = 50 threshold = -2000



For p = 10 threshold = -500


Accuracy of Test Data in %
Iterations