Text Analytics - Homework 2 - Luis Steven Lin

Question 1: Tokenization

You will report the tokenization differences between both approaches. For each line where the tokenization differs between the two approaches, show the original string and the two alternative tokenizations. What patterns do you see in the differences?

Code:

- Functions.java
- FunctionsCorenlp.java
- FunctionsLucene.java
- hw2_1_tokenization.java

Inputs:

- wsj_0063

Outputs:

- hw2_1_tokenization_0_Raw.txt
- hw2_1_tokenization_1_Corenlp.txt
- hw2_1_tokenization_2_Lucene.txt
- hw2_1_tokenization_all.txt

hw2_1_tokenization_all shows all lines groups by original, corenlp tokenization, and lucence tokenization. Tokenization differs for every line. See Appendix 1 with the selected differences highlighted. It seems that Corenlp is much better at tokenizing. Some patterns in the difference:

- Punctuation: Lucene removes them, while Corenlp tokenizes them (e.g. “,”). The exception is numbers, where both keep as token (e.g. [62.256])
- Abbreviations (of companies, salutation): Lucene removes the punctuation, Corenlp keeps it (e.g. [Ltd] vs [Ltd.], [Mr], [Mr.])
- Symbols: Lucene removes all symbols, Corenlp keeps them (e.g. $, “”, --). This exception is for possessive and negation (see below).
- Hyphenated words: Lucene tokenizes as separate tokens, Corenlp keeps the hyphenated work as one token (e.g. 70-a-share)
- Possessive: Lucene removes, Corenlp tokenizes (“ ‘ “)
- Abbreviation possessive: Lucene tokenizes as one, Corenlp as separate tokens (e.g [they’re] vs. [they],[‘re])
- Abbreviation negation: : Lucene tokenizes as one, Corenlp as separate tokens [didn’t] vs. [did] [n’t].

Question 2: Normalization

You will report the differences between these four approaches, for each line where the normalization differs between any of them. Describe the main ways the lemmatizer differs from the stemmers. Also describe the patterns you see in the differences between stemmers.

Code:

- Functions.java
- FunctionsCorenlp.java
- FunctionsLucene.java
- hw2_2_normalization.java

Inputs:

- wsj_0063

Outputs:

- hw_2_normalization_EnglishAnalyzer.txt
- hw2_2_normalization_PorterStem.txt
- hw2_2_normalization_KStem.txt
- hw2_2_normalization_corenlp.txt
- hw2_2_normalization_raw.txt
- hw2_2_normalization_all.txt

hw2_1_normalization_all shows all lines groups by different normalization methods. Tokenization differs for every line. See Appendix 2 with the selected differences highlighted.

First of all, the tokenization discussed above applies here (e.g. punctuation removed by lucene, but tokenized by corenlp, etc). One difference between the lemmatizer and the stemmers, is that the stemmers lower case everything, while the lemmatizer identifies companies names, abbreviations and day of weeks and keeps the upper case (e.g. Sea Containers Ltd., Tuesday).

The EnglishAnalyzer and the PorterStem seem to do a classic stem, that is just cutting the words into their root form (e.g. “increas” vs “increase”), while KStem and Corenlp lemmatizer do not stem. One difference between Corenlp and Kstem is that the lemmatizer reduces words to the simplest form (e.g. “pressed” to “press”), while Kstem seems that is not changing it. There are certain words that Kstem processes differently than the other stemmers and lemmatizer. For example “European”, is processed as “europe” by Kstem, but as “european” by the rest. Adjectives and adverbs are also processed differently: “jointly” (lemmatizer), “joint” (Kstem), “jointi” (other stemmers).

Some verb forms and contractions are treated differently by all of the stemmers and lemmatizer: “they’re” as [they, be] (lemmatizer), [they’r] (englishanalyzer), [they, re] (kstem) and [thei, re] (porter).

Question 3: Tokenization and Normalization

You will decide on a tokenization and normalization algorithm to apply to classbios.txt from the previous assignment. Make an argument as to why you selected that particular tokenization and normalization algorithm. Include your normalized classbios.txt file in the zip.

Code:

- Functions.java
- FunctionsCorenlp.java
- hw2_3_normalize.java
- hw_2_grams.py

Output:

- classbios_normalized.txt (java)
- classbios_normalized_line.txt (python)
- classbios_normalized_sentence.txt (python)
- classbios_normalized_tokens.txt (python)

Note: after talking to the professor, it was ok to remove special characters (i.e. non-ascii) since Python cannot handle correctly. The results that follow are based on pre-processing the classbios file by removing non-ascii characters. Therefore, the results are slightly different from using Corenlp java or including ascii characters. The differences are pointed out since the code was also run using the tokenized output from Java. The classbios normalized python are the result of doing pre-processing in python to remove lines that are not interested and non-ascii characters. The classbios normalized java includes all lines tokenized.

The lemmatizer seems to be more accurate since it involves using context and part of speech in a sentence. The lemmatizer is able to extract names, dates and abbreviations and processed them accordingly. On the other hand, stemming is fast since no word analysis is performed. But this means that results from stemming might lose context and precision. Therefore, since the classbios is not a big file and lemmatizing provides advantages to consider topics that might occur in class bios (e.g. location, school name, time points, etc), then the lemmatizer was chosen to normalize the classbios.

Question 4: Unigram

Write a program to calculate the frequency of each (normalized) word in the normalized classbios.txt corpus. Report the top 20 most frequent words, excluding stop words and punctuation, and their frequencies. Describe whether you think this list gives much information about this corpus. Did you learn anything about the backgrounds of the class?

Code:

- hw_2_grams.py

Output:

- classbios_top20Unigrams.json
- output.txt

```
Number of unigrams (after removing unigrams with stop words and punctuation): 1830

Top 20 Unigrams:

          datum : 127
       analytic : 104
           text : 82
           work : 69
        program : 44
     University : 42
        project : 41
            use : 39
       research : 36
          field : 35
       analysis : 35
           year : 32
     interested : 29
       business : 29
       interest : 27
   Northwestern : 26
         degree : 25
        science : 24
        develop : 24
       graduate : 24

*Note: if there are ties for the values for k, it includes all those values as well.
(So the function might return more than k elements)
```

Note: if the tokenized file of Java is used, some frequencies slightly change:

- program: 43
- Northwestern: 27
- degree: 24

The unigrams gives some idea of the background in the class. The top words by far data and analytics, which might represent interests in the subject or current involvement professionally or academically. The fact that “text” is frequent is not clear without having a knowledge of the class, but a reasonable assumption is that is linked to the top

words (data and analytics), which means the interest or experience is in the this particular field. The words “Northwestern”, “University”, “degree”, “graduate” and “program” suggest the class is in an academic setting. The action words “interest”, “research”, “analysis”, “develop” and “use” are more likely linked to the top words and suggest that people have interest or have been involved in these areas.

In conclusion, the unigram list gives some information about the corpus, but is not enough since it does not offer a coherent and specific picture about the corpus. Using bigrams might provide better context and give more specific information that can be inferred from the corpus and background of the class.

Question 5: Bigram

Write a program to calculate the frequency of each bigram in the corpus (i.e., sequence of two words). Report the top 20 most frequent bigrams, excluding bigrams that include a stop word or punctuation, and their frequencies. Describe whether you think this list gives much information about this corpus. Did you learn anything about the backgrounds of the class?

Code:

- hw_2_grams.py

Output:

- classbios_top20Bigrams.json
- output.txt

```
Number of Bigrams (after removing bigrams with stop words and punctuation): 1853

Top 20 Bigrams:

           text, analytic : 46
 Northwestern, University : 17
          datum, analysis : 14
           msium, program : 13
       become, interested : 12
           datum, science : 12
        machine, learning : 11
              text, datum : 8
         datum, scientist : 7
           master, degree : 6
        computer, science : 6
            look, forward : 6
               big, datum : 6
      unstructured, datum : 6
            social, media : 6
             text, mining : 6
              would, like : 5
    undergraduate, degree : 5
            datum, mining : 5
        predictive, model : 5
          datum, analytic : 5
           datum, quality : 5
               also, work : 5

*Note: if there are ties for the values for k, it includes all those values as well.
(So the function might return more than k elements)
```

Note: if the tokenized file of Java is used, some frequencies slightly change.

This bigram list gives a more specific and enough information about the corpus and the background of the class. “Text, analytic” is by far the most frequent, suggesting this is a text analytics class and the topic of the corpus. The action bigram “become, interested” is also high frequency, suggesting that the topic of discussion in the corpus is related to describing the interest. “Northwestern, University” suggests were the class is located.

The bigram “msium, program” suggests that most people are from this program or the class is offered under this program. Bigrams such as “datum, science”, “computer, science”, “datum, scientist” and “master, degree” give information about the academic or professional background of the class. Bigrams such as “machine, learning”, “text, mining”, “datum, mining”, “datum, quality”, “predictive, model” and “datum, analysis” provides insight of the techniques and usual tasks that the class has experience with.

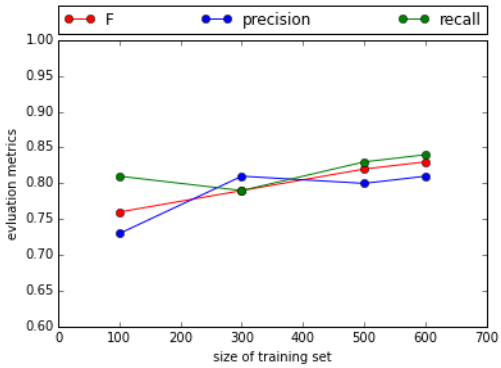
In conclusion, bigrams provide much more specific information about the class and a clear topic and context for the corpus.

Question 6: Sentiment Analysis

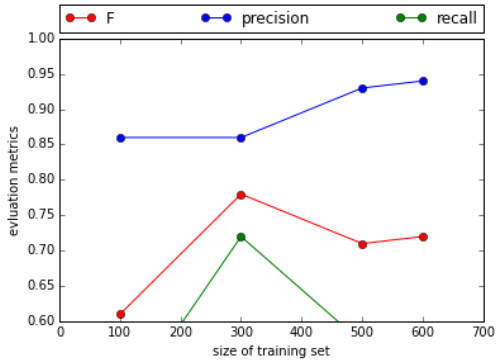
Results from training in the first "size" documents, and testing on documents 600 to 799

	NB			SVM		
size	F	precision	recall	F	precision	recall
100	0.76	0.73	0.81	0.61	0.86	0.47
300	0.79	0.81	0.79	0.78	0.86	0.72
500	0.82	0.80	0.83	0.71	0.93	0.57
600	0.83	0.81	0.84	0.72	0.94	0.59

Naïve Bayes



SVM



Do you think performance would improve with even more examples?

It seems that the curve is increasing and flattening out. Thus, it is expected that increasing the number of examples would steadily increase the performance measures, but this improvement might be marginal and not substantial, eventually reaching a limit.

Report how much you've improved performance on this set (precision, recall, F score)

Results from SVM show that that precision improves compared to NB. However, F values and recall get worse.

Results from training in the first "size" documents, and testing on documents 800 to 999

	NB			SVM		
size	F	precision	recall	F	precision	recall
800	0.79	0.8	0.77	0.63	0.93	0.47

If your original classifier performance was substantially different evaluated on cv8 and cv9 than it was previously on cv6 and cv7, why do you think that was?

The original classifier performance was similar on the hold-out data (if the performance was substantially different, then overfitting might be an issue or the fact that the hold-out data set is very different from the training data set, such as containing a very different vocabulary and new words that did not exist in the training).

If you did the bonus, and your new system didn't outperform your original classifier on this new data, but did outperform it previously on the cv6 and cv7 files, what do you think happened?

In this case the SVM compared to NB behaved similar in the training and new data, that is the precision for SVM was higher but the F-value and recall were lower than NB (if this wasn't the case it might be overfitting)