Project – Fake News Challenge

CS583
Savan Kiran
Master's student

I discuss about the tasks needed to be performed for the project at first and then give an overview of how to run the project.

1. Classification using Machine Learning

Preprocessing

As part of preprocessing, I tried various combinations of stemming & lemmatization, along with stop words removal from Lucene support & custom stop word removal that keeps the context preserving words like 'not', 'ain't', etc which would help in deciding the spin of a sentence.

For the final configuration (for my best result), I did the following:

- Porter Stemmer using Lucene's tokenizer (proved to be better than lemmatization for first phase of identifying if doc is unrelated or related)
- Stop word elimination using English Analyzer's default stop set
- Tokenize
- Lemmatization (disabled, as I observed a better performance without it in the first phase of identifying if the doc is unrelated or related)

```
/**
 * Tokenizeer does porter stemmer & stop word elimination
 * @param text
 * @return
 */
public List<String> tokenize(String text) {
    PorterStemmer stemmer = new PorterStemmer();
    StandardTokenizer tokenizer = new StandardTokenizer(new StringReader(text));
    List<String> tokens = new ArrayList<String>();
    try {
        TokenStream tokenStream = new StopFilter(
                new ASCIIFoldingFilter(new ClassicFilter(new
LowerCaseFilter(tokenizer))),
                EnglishAnalyzer.getDefaultStopSet());
        tokenStream.reset();
        while (tokenStream.incrementToken()) {
            String token =
tokenStream.getAttribute(CharTermAttribute.class).toString();
            if(STEMMING) {
```

```
                stemmer.setCurrent(token);
                stemmer.stem();
                token = stemmer.getCurrent();
            }
            tokens.add(token);
        }
        tokenStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return tokens;
}
```

Classifiers

I have setup three levels of classifiers, namely -

- Classifier 1 – classifies if the body is related to the headline or not.
- Classifier 2 – classifies if the body merely discusses the headline or opinionated it (either agree or disagree).
- Classifier 3 – classifies if the body agrees or disagress to the stand taken in headline.

Each level classifiers works on a set of documents and after processing, passes a subset of these documents to next level classifiers for further analysis. These classify calls return back result of classification of the documents they worked on.

Features

I created an array of features and experimented with them. Finally, I chose a set of features that work well for each classifier.

- Classifier 1 – To find the relevance between headline and body, I used the following features -
    - Bag of words (Jaccard's coefficient)
    - Tf-Idf (I used the DF values that was provided by you on a larger corpus of documents)
    - Binary co-occurrence (How many words of headline appear in body)
    - N-grams (Bi- and Tri-)

```
    Feature bagOfWords = new BagOfWords(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()));
    document.addFeature(bagOfWords);
    Feature tfIdf = new TfIdf(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()));
    document.addFeature(tfIdf);
    Feature binaryCoOccurrence = new
BinaryCoOccurrence(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()));
```

```
    document.addFeature(binaryCoOccurrence);
    Feature nGram_2 = new NGram(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()), 2);
    document.addFeature(nGram_2);
    Feature nGram_3 = new NGram(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()), 3);
    document.addFeature(nGram_3);
```

- Classifier 2 – To check if the body forms some opinion of the headline or not. For these, I used a list of hedge words (that indicate mere discussion without stand), supporting words (that indicate agreement to headline) and refuting words (that indicate non-agreement to headline). Checking merely for the occurrence of these words in the body without looking for the context poisoned my classifier making it make relationships to words/features that carry no real meaning. I finally fixed this (though partially), by looking at context.
  In general, I look for words from headline in the body and look at their prefix (could be 1-word or 1-sentence), and look for spin words that either support/refute this word, thereby removing cases of false classification of document merely because it contained a particular word.
  - Hedge features – (plain checking for headge words in the body)
  - Agree_3 & Disagree_3 – (Looking for headline words in body and analysing their spin by considering upto 3 prefix words)
  - (Not implemented) I planned to extend by adding more features with 4-word, 5-word prefixes to make the system more robust. Right now, the occurrence is very sparse.

```
Feature hedgeFeature = new HedgeWords(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()));
document.addFeature(hedgeFeature);
Feature agree_3 = new Agree(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()), 3);
agree_3.setDictionary(dictionary);
document.addFeature(agree_3);
Feature disagree_3 = new Disagree(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()), 3);
disagree_3.setDictionary(dictionary);
document.addFeature(disagree_3);
```

- Classifier 3 – To verify the opinion of the body, I use the above mentioned features minus hedge feature.
  - Agree_3 & Disagree_3 – (Looking for headline words in body and analysing their spin by considering upto 3 prefix words)

```
    Feature agree_3 = new Agree(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()), 3);
    agree_3.setDictionary(dictionary);
```

```
    document.addFeature(agree_3);
    Feature disagree_3 = new Disagree(document.getHeadline(),
dataRepo.getBodies().get(document.getBodyId()), 3);
    disagree_3.setDictionary(dictionary);
    document.addFeature(disagree_3);
```

Note: I've configured multiple threads to work on feature extraction for performance. So, the list of features to be extracted for each classifier can be found in FeatureExtractionCallable in each classifier.

ML component

I used the liblinear SVM Java library for a Linear classifier. In particular I used L2R_L2LOSS_SVC

with constraint set to 1.0, epsilon to 0.1 and bias to 1.0. I felt it a bit hard to classify 3 labels using a single linear classifier and hence created a hierarchy of it, mostly to have a finer control over what features to use at what level.

2. Measuring Performance

I used the FNC scorer logic to compute the final score. Please find the score of the best version of my system (so far :)).

Classifier 1 - Correct detections: 16103 Wrong detections: 852
Classifier 2 - Correct detections: 3698 Wrong detections: 2431
Classifier 3 - Correct detections: 9 Wrong detections: 7
Max score: 7585.25
Null score: 2565.5
Test score: 5879.25
Accuracy: 0.8153936891772339
FNC score: 0.7750898124649814

Confusion matrix:

	Actual				
Predicted		Unrelated	Discuss	Agree	Disagree
	Unrelated	10118	406	213	89
	Discuss	144	3678	1882	398
	Agree	0	26	1	0
	Disagree	0	0	0	0

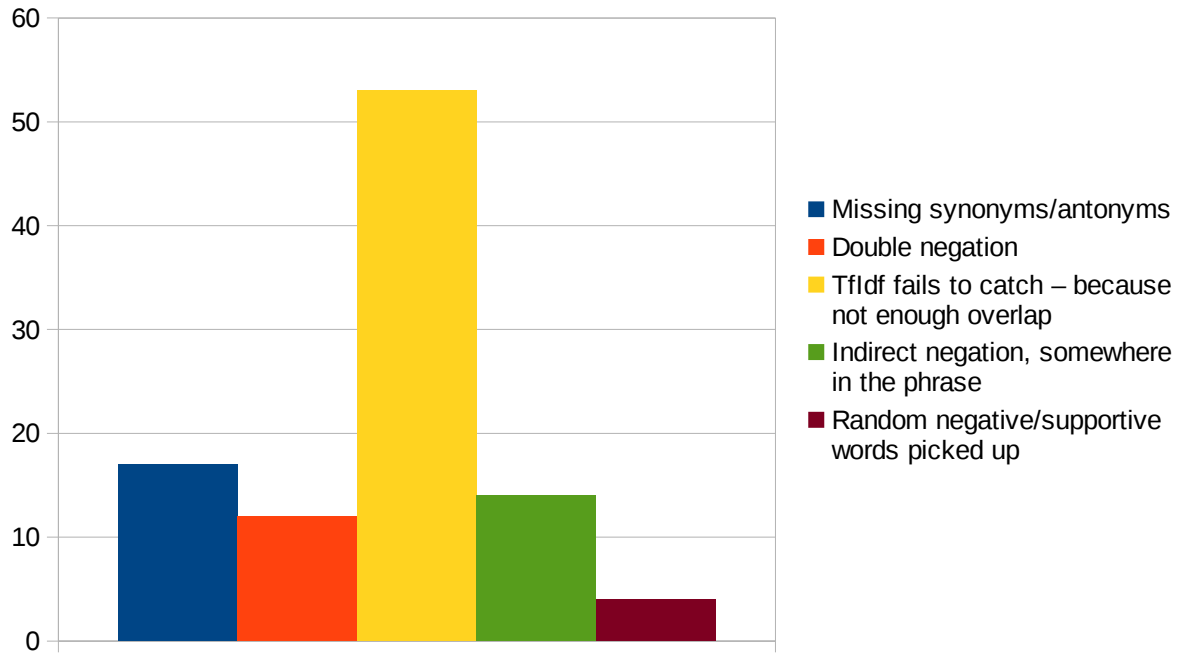
FNC score has weighted scores with different weights for each label. Clearly, identifying some labels are more important the others, unlike a traditional accuracy where each document is viewed equally. As shown, my accuracy according to traditional sense would be **81.5%** where as my FNC score was lower,

i.e., **77.5%**. I was able to correctly identify majority of labels in Classifier 1 correctly, and more than half of labels correctly in Classifier 2. However, I couldn't very successfully find opinion docs in Classifier 2 because of which Classifier 3 had only a few documents to work with.

3. Error Analysis

Analysis

My analysis of error was a little skewed toward errors in Classifier 1 with not being able to identify relevance between headline and body. I also analyzed errors from Classifier 2 & Classifier 3. Following is my observation.



Different configurations

Lemmatization would help me Classifier 2 & 3 as it would still have the part of speech component with it lemmatizing to base word helps match synonyms and antonyms. Whereas, lemmatization at Classifier 1 would be an overkill. However, I observed best results with stemming as against lemmatization and hence kept it in the final submission. Although, you can change the LEMMATIZATION flag in WordUtils.java to enable lemmatization.

- No stemming/lemmatization – Performance was bad as words with same base but different forms did not match.
- Stemming – Performance was best with this
- Lemmatization – Performance was a little lower than with stemming, esp, with Classifier 1.

4. Improving classification

I tried my hands on one optimization using Wordnet and look for synonyms and antonyms.

- Synonym/Antonym – I did not take the traditional path of looking for synonyms and antonyms of words that appear in headline or body. I felt, they could mislead to many matches in many documents and hence would be inconclusive. I designed a model in which I give the system a list of supporting or refuting features as seed at the start. Before the system trains, it parse for all words in these list and looks up for their synonyms. Thus eventually extending lookup space for supporting/refuting words. One can do this recursively on the given list to add more words. Although, I expected a positive outcome, the result was deteriorating. Mostly because the Classifier 2 starts picking up these words in all the documents and makes false connections of these words to discuss label which is predominant.

5. Instructions to run the project

Dependencies

1. Stanford CoreNLP
2. Wordnet extJWNL
3. SVM – Liblinear
4. Apache Commons – CSV Parser

Build and Run

Similar to HW 3 & 4, pom.xml has all the dependencies. Use Maven to compile and run. I checked that extJWNL is pulling wordnet dictionary as part of pom.xml. However, I didn't notice this earlier and have been using a dictionary in my resources path. Pushing all of it to d2l. Sorry for the inconvenience.