CSC 483/583: Programming Project (200 pts) The Fake News Challenge: Stance Detection

By

Mithun Paul

Due before 11:59 P.M., May 7

For this project you must submit:

• An archive containing all the source code.

• Binaries/jars (if a language that requires compilation is used) and/or instructions how to run the code.

• A PDF document that must contain at least:

– Description of the command line to run your code, with an example.

– Description of the code. You don't have to describe every function implemented. But you should describe the main part of the code and indicate where each question is addressed.

– Results, i.e., the output of your code, for all the questions below that required

programming.

– Answers for all questions that do not require programming.

Answers are always graded by inspecting both code and documentation. Missing code yields no credit. Missing documentation yields partial credit (if code exists and produces correct results).

Because the credit for graduate students adds to more than 200, graduate students' grades will be normalized at the end to be out of 200. For example, if a graduate student obtains 220 do not points have on to this solve project, the problems her final marked grade "grad will be students $220\times \frac{200}{250}$

only." = 176. If they Undergraduate do, their grades students will not be normalized but will be capped at 200. For example, if an undergraduate student obtains 210 points on this project (by getting some credit on the "grad students only" problem), her final grade will be 200.

Important notes:

This project follows the Fake News Challenge (FNC), available at: http://www.fakenewschallenge.org. Please read the information on this website in addition of the text provided here!

This project requires machine learning rather than (or in addition of) information retrieval! If you prefer to work solely with information retrieval models, please choose the other project offered ("Building a part of Watson").

Implement a submission for the FNC using machine learning.

From the FNC website:

"The goal of the Fake News Challenge is to explore how artificial intelligence technologies, particularly machine learning and natural language processing, might be leveraged to combat the fake news problem. We believe that these AI technologies hold promise for significantly automating parts of the procedure human fact checkers use today to determine if a story is real or a hoax.

Assessing the veracity of a news story is a complex and cumbersome task, even for trained experts 3. Fortunately, the process can be broken down into steps or stages. A helpful first step towards identifying fake news is to understand what other news organizations are saying about the topic. We believe automating this process, called Stance Detection, could serve as a useful building block in an AI-assisted fact-checking pipeline. So stage #1 of the Fake News Challenge (FNC-1) focuses on the task of Stance Detection."

Task Definition:

Input: A headline and a body text – either from the same news article or from two different articles.

Output: Classify the stance of the body text relative to the claim made in the headline into one of four categories:

• Agrees: The body text agrees with the headline.

• Disagrees: The body text disagrees with the headline.

• Discusses: The body text discuss the same topic as the headline, but does not take a position

• Unrelated: The body text discusses a different topic than the headline

See the FNC for examples for these four classes, as well as for examples and descriptions of the actual data.

Please note that the FNC provides the official testing dataset on June 1st, which, unfor- tunately, is after our class ends. For this reason, we will not be using the official datasets for this project. Instead, the instructor will partition the official training dataset into two dataset: our own, local training dataset, and our own (smaller) testing dataset. Please see D2L for these data.

Qn) Instructions to run the code?

Ans:

Prerequisites:

You will need the following python libraries installed in python 2.7

1. Scikit-learn
2. Nltk
3. numpy
4. smtplib
5. **from** csv **import** DictReader
6. **from** __future__ **import** division

Steps to run:

1. Unzip
2. Cd to the folder which has initializer.py
3. Type: **python initializer.py mithunpaul08@gmail.com**

Note1: It takes around 20 mins for the entire code to run. If you specify your email id at the end of the command line input, it will send you an input when it finishes training and testing.

Note2: If you don't want an email to be sent, just run it without the email id

Eg: **python initializer.py**

Note3: I have coded this up on python 2.7 . Might not work on python 3+ since it is not backward compatible.

Qn) Description of the code. You don't have to describe every function implemented. But you should describe the main part of the code and indicate where each question is addressed.

Ans: details mentioned as part of Ans 1 below.

Qn)Results, i.e., the output of your code, for all the questions below that required Programming.

Ans: details mentioned as part of Ans 1 below.

Qn) Answers for all questions that do not require programming.

Ans: please find below

Your project should address the following points:

1) (125 pts) Classification using machine learning: Implement a machine learning (ML) classification system using Support Vector Machines (SVM) for the four classes above. I recommend one of these packages for the ML part:

• For C/C++: http://svmlight.joachims.org

• For Python: http://scikit-learn.org/stable/

• For Java: https://www.csie.ntu.edu.tw/~cjlin/liblinear/

• For Scala: https://github.com/clulab/processors

Qn) Describe how you pre-processed the terms in the headline and document (stemming, lemmatization, stop words, etc.). Describe what features did you create for your classifier. For example, how did you model the overlap between the headline and the body of the article? How did you model negation? Describe how you implemented the ML component. Note that you are allowed to look at the baseline provided by the FNC organizers, but you are not allowed to use their code. You have to implement your own classifier.

Ans: My code contains the following major Phases/Steps:

1. Training_phase1:
    a. In this phase, the entire training data is read into memory, and tokenized. Then it is given to a TfidfVectorizer from sci-kit which finds the tf-idf scores for each terms. Note that the terms are lemmatized, normalized, stemmed and tokenized at this point. Then a **cosine similarity** is calculated (calculateCosSimilarity()) between the headline and body using the aforementioned-tf_idf score. This is considered a feature at this phase. A feature vector is thus created and given to a classifier, along with the labels (related, unrelated). Note that only "unrelated" is a class given in the training document. The rest (agree,disagree and discuss)is all congregated into one class, called "related". So here the machine learning part is where we decide on the threshold below which the data has to be called "unrelated" . Using the cosine similarity technique I found out that the threshold is 0.1
2. Validation_phase1:
    a. Once the classifier learns the threshold for the aforementioned two classes, it is tested again against the same training dataset. I got a 99% at this point, when I fed the results to the scoring function provided by the FNC group on github. Note that this functionality is turned off in the code I am submitting. If you want to turn it on, you can do it with setting (do_validation_phase1=False;) to True inside the initializer.py.
3. Testing_Phase1:
    a. In this phase the training data is first read into the memory. Then the threshold

learnt from training phase 1 is used to separate out the headline-body pairs into related or unrelated. At this point the results of this phase is separated out into gold labels and predicted labels of the class "unrelated" . This will be later combined with output of phase 2 and given to the final scoring function. Also at this stage the entire testing data is split into two based on this classifier, i.e related and unrelated data. The data in the related class ONLY will be used for step 6 below, i.e next phase, Testing_Phase2.

4. Training_Phase2:
    a. In this phase, i go through the training data set and separate out all the related data. I.e I find all the data which belogs to the gold stance "agree", "disagree" and "discuss" . I then feed this data into the SVM I am using to train the classifier. Here I am using an **SVM** as a classifier. I use the term frequencies as a feature vector at this point. Particularly I feed it the negative words and hedging words provided by the FNC dataset. The thus-trained-SVM will be used in the next two steps viz., Validation_phase2,Testing_Phase2
5. Validation_phase2:
    a. Here the SVM trained in the previous section is fed the same training data. The SVM then separates the data into 3 labels/classes , agree, disagree and discuss. Note that this functionality is turned off in the code I am submitting. If you want to turn it on, you can do it by setting (do_validation_phase2=False;) to True inside the initializer.py.
6. Testing_Phase2
    a. Functionality wise this is the same as the step above, i.e Validation_phase2. Only difference here is that the trained SVM is used now to test on the testing data. I.e the part of the testing data which was split out as "related" in section 3 above. Testing_Phase1. So this data is thus fed into the trained SVM which then classifies it into 3 labels/classes , agree, disagree and discuss.

Data modelling:

1. I used featuer extraction tools provided by scikit (http://scikit-learn.org/stable/modules/feature_extraction.html)
2. Specifically I used the TfidfVectorizer
    a. vectorizer = TfidfVectorizer(tokenizer=normalize, stop_words='english')
3. This internally calls the nltk's tokenize functionality, which calls the potter stemmer internally. Note that it also does a stop word removal.
    a. nltk.word_tokenize(text.lower().translate(remove_punctuation_map)))
    b. stemmer = nltk.stem.porter.PorterStemmer()
    c. remove_punctuation_map = dict((ord(char), None) **for** char **in** string.punctuation)

4. I used tf-idf for overlap between headline and body. Also I trained the classifier first on just the disagree class. Then I manually threw away the words that didn't signify negativity. Between these and the word provided for hedging and negative words, i think my classifier did a good learning.
5. Also I used nltk wordnet for lemmatization
    a. wordnet_lemmatizer = WordNetLemmatizer()
    b. for indivWord in actualBody:
    c. lem_word=wordnet_lemmatizer.lemmatize(indivWord)
    d. actualBody_lemmatized.append(lem_word)

2) (25 pts) Measuring performance: Measure the performance of your FNC system, using the official FNC evaluation measure (see the FNC website for details). Discuss how this measure compares to standard accuracy. Make sure your report includes evaluation scores on our testing dataset.

Ans:

As of now, this is the final score of my classification code as produced by the scoring program given by the FNC group. This is evaluated against the testing dataset provided on D2L.

	agree	disagree	discuss	unrelated
agree	1097	113	587	236
disagree	202	77	113	66
discuss	779	146	2670	672
unrelated	60	16	149	25634

Score: 10737.5 out of 13222.75 (81.2047418275%)

The FNC evaluation measure is different from accuracy because accuracy is the ratio of correct predictions over total input. Instead here their score is closer to an F1 score. Here they increase the score by 0.25 if the gold label and predicted labels match. Plus they boost the score by 0.5 if the gold label is one of [agree, disagree, discuss]. This is similar to increasing the true positives. I think they

are trying to make their score closer to precision than recall. Even if the golden label and predicted label are different the score is again increased if either of them is part of [agree, disagree, discuss].

4) (50 pts) Error analysis: Perform an error analysis of your best system. Sample several stances that your system classified incorrectly (say 100), and try to identify error classes in this subset. For example, how many stances were classified poorly due to your incomplete handling of negation? How many were classified incorrectly due to the fact that your system (probably) does not address synonyms and antonyms? Discuss the error classes you found, and include a histogram of these error classes from your analysis (that is, how many questions appear in each class).

Lastly, what is the impact of stemming and lemmatization on your system? That is, what is your best configuration: (a) no stemming or lemmatization; (b) stemming, or (c) lemmatization? Why?

Ans :

I have found the following **error classes**.

1. **Antonyms**
    a. I think some words in the incorrectly classified documents could have been classified correctly if I had added some antonyms also to the feature explicitly.

Eg:

headline:Michelle Obama's face blurred by Saudi state television

actualBody:**However**, observers of the live broadcast — including a Wall Street Journal reporter in the country — said that there was no blurring of Mrs. Obama, and that the broadcast showed her shaking hands with King Salman. A Youtube video **purporting** to show that Saudi television blurred the image of Michelle Obama. Youtube

2. **Mutual information**
    a. I think some words in the incorrectly classified documents could have been classified correctly if I had added some mutual information also to the feature vector explicitly. Also if I had used bigrams or phrases as feature vector, i think the following example could have been captured well.

Eg:

pred_label:agree

gold_int[dataCounter] :discuss

headline:Paul Revere's 1795 time capsule unearthed

actualBody:

The capsule, he explained to CNN, **was documented to have been** first buried in 1795, then was dug up and reburied in 1855 during repairs to the building. "**What we know** the box contains, based on the notes that we have, is a Paul Revere plate, papers, and coins from the 1600s.

3. **Idf from gigaword.**
    a. Mihai had provided standard idf from a gigaword corpus. I didn't use it because I thought it might bias the data. Plus this was a classification task and not information retrieval task. However, I have found examples where this might have been useful.

Eg:

****

errored document count:86

pred_label:0

gold_int[dataCounter] :2

Headline: Paul Revere's 1795 time capsule unearthed

actualBody:History buffs, start your engines: CNN is reporting that a Boston crew has removed a time capsule from the cornerstone of the Massachusetts State House, **believed** to have been stashed there by then-Governor Samuel Adams and patriot Paul Revere. Inside the capsule are contents that will be showcased as soon as the box is removed from the cornerstone, according to Massachusetts Secretary of State William Galvin. The capsule, he explained to CNN, was **documented** to have been first buried in 1795, then was dug up and reburied in 1855 during repairs to the building. The full catalog of its contents are as-yet unknown. "What we know the box contains, based on the notes that we have, is a Paul Revere plate, papers, and coins from the 1600s. It **may** contain other stuff too, we don't know that yet," he said. The Secretary of State's office is looking into whether the box will be **reinstated** and whether new items from the current era will be added to the box and reburied. ABC News notes that the discovery is "the oldest unopened time capsule in the country."

4. **Dependency parsing.**

a. I wanted to use ODIN's dependency parser to model negation completely, didn't get time to. Below is some example where the negativity could have be

Eg:

pred_label:2

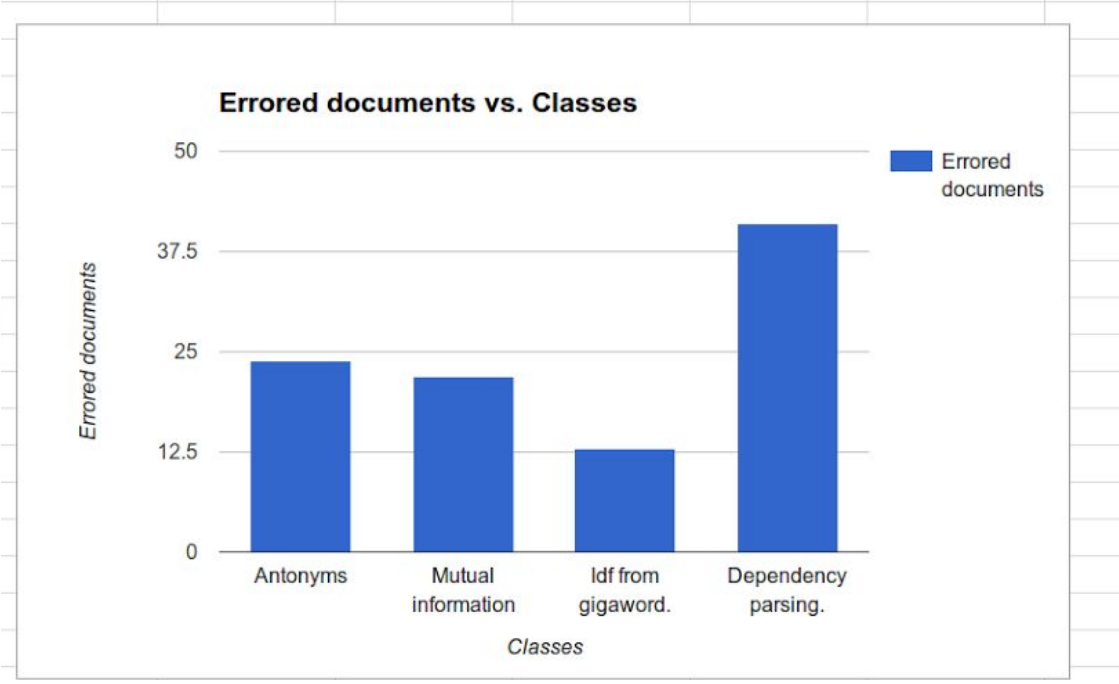
gold_int[dataCounter] :1

headline:A Mass Grave Points to a Student Massacre in Mexico

actualBody:The bodies found in a mass grave were confirmed not to be the missing students.

All these documents are stored in the attached file: **errorAnalysis.txt**

Also below is a **histogram** of the distribution.



Stemming or lemmatization had very less effect in my classification system. The score of 81.20% above was with stemming or lemmatization. Before that it was 79.56%

I don't think stemming and lemmatization has much impact because this is a classification task and not a Information retrieval task. By which what I mean is, in Information Retrieval a word in the

query might appear in the document in its lemmatized form or a stemmed form. For example, if the query has "automotive" and the document has "automobile", the document will be missed unless both the query and document words are stemmed/lemmatized to auto. However, it doesn't matter in case of classification, because they are still going to be in the same class. So the combined weightage/contribution of the lemma towards the class might increase a bit, but not as much as to change the classification from one class to another. After all, like how naive bayes works, in classification, we don't worry too much about the exact probability. We need that for ranking of documents in information retrieval. In classification as long as the weights stay reasonably inside a big margin it shouldn't matter.

5) (50 pts) Improving classification (GRAD STUDENTS ONLY): Improve the above standard stance classification system using natural language processing and/or machine learning. For this task you have more freedom in choosing a solution and I encourage you to use your imagination. For example, you could use WordNet to capture and handle synonyms and antonyms. You could use the simple idea from this paper to handle negation (see the paragraph starting with "One unconventional step..." in: https://arxiv.org/abs/cs/0205070). You can switch to a more complex learning algorithm such as a neural network.

Ans: I used the wordnet synsets to increase the score. For the hedging words and words denoting negativity/disagreement, I found the synonyms from word net and attached it also to the feature vector.

```
from nltk.corpus import wordnet as wn
for indivWord in hedge_words:
 for synset in wn.synsets(indivWord ):
    for lemma in synset.lemmas():
        entire_corpus=entire_corpus+ lemma.name()
```

It increased my score from 81.20% to 83.72%. Please see below.

| | agree | disagree | discuss | unrelated |

agree	1111	27	659	236
disagree	179	39	174	66
discuss	442	15	3138	672
unrelated	61	3	161	25634

Score: 11070.5 out of 13222.75 (83.7231286986%)