CS 331 Assignment #3: The Sarcasm Detector

Out: May 18, 2016
Due: June 1, 2016, 17:00:00
Type: Team Assignment (Groups of 2 max)

Description

Sarcasm is a very cultural sentiment. It is difficult to translate and relies on a lot of linguistic and cultural quirks. In English, it is also a staple of comedy, and especially useful in derisiveness. It seems like something that would be incredibly difficult for a computer to understand.

Surprisingly, we do have models that can, at the very least, make a decent stab at understanding sarcasm. Recent papers by Tsur et al. [1] and Davidov et al. [2] presented models for sarcasm detection applied to Tweets and Amazon customer reviews. You are going to try to do the same thing. We will use a subset of the data from [1] and [2]. We have also made some minor formatting modifications to this data set.

You will be using naive Bayes for this classification problem. Given an input sentence, you are to determine whether or not a sentence is sarcastic*. This may seem difficult but your results on this dataset may surprise you.

* Strictly speaking, you are determining whether a given sentence is **extremely** sarcastic or not. We split the data set so that "sarcastic" is all sentences that scored a 5 with their paper's metric, and "not sarcastic" is all the other scores.

File Format

You will be given two files in the same format. They are *training_text.txt* and *test_text.txt*. These files consist of sentences in the following format:

("This is a review",0)
('This sentence is false',1)
("REF This is a tweet",0)

In general, the format of a line is ("some text",classlabel). Note that the text inside the quotes may contain arbitrary punctuation other than quotation marks and the quotation marks my be " or '.

Pre-processing step

This step converts each sentence into a feature vector plus a class label that is read in by your Naive Bayes algorithm. You will be representing each sentence as a "bag of words". The steps for this conversion as are follows:

1. Strip the punctuation. For this assignment, we will not be using the punctuation (unlike the original paper). You can remove apostrophes from words to make stripping the punctuation simpler to do.

You do not lose much accuracy by confusing edge cases like "wont" and "won't" in our data set. This ambiguity also allows you to get the benefit of recognizing that common bouts of laziness like "youre" and "you're" are meant to be the same word.

2. Form the vocabulary. The vocabulary consists of the set of all the words that are in the training data. The vocabulary will now be the features of your training data. Keep the vocabulary in alphabetical order to help you with debugging your assignment.
3. Now, convert the training AND test data into a set of features. Let M be the size of your vocabulary. For each sentence, you will convert it into a feature vector of size M+1. Each slot in that feature vector takes the value of 0 or 1. For the first M slots, if the ith slot has the value 1, it means that the ith word in the vocabulary is present in the sentence; otherwise, if it is 0, then the ith word is not present in the sentence. Most of the first M feature vector slots will be 0. Since you are keeping the vocabulary in alphabetical order, the first feature will be the first word alphabetically in the vocabulary. The (M+1)th slot corresponds to the class label. A 1 in this slot means the message is (extremely) sarcastic while a 0 in this slot means the message is not. We will refer to the "featurized" training data as the pre-processed training data.
4. Output the pre-processed training and testing data to two files called preprocessed_train.txt and preprocessed_test.txt. The first line should contain the alphabetized words in the vocabulary, separated by commas, followed by a dummy non-word called "classlabel". The lines that follow the vocabulary words should be the featurized versions of the sentences in the training data, with the features separated by commas. Your file should look something like:

   aardvark,almost,anticipate,...,classlabel
   0,1,0,...,0
   1,0,1,...,1

   Note that even though we are asking you to output the training data, the classification step should happen right after the pre-processing step (just pass the preprocessed data directly to your classifier. Don't save it out and reload it back in).

Classification step

Build a naive Bayes classifier as described in class.

1. In the training phase, the naive Bayes classifier reads in the training data along with the training labels and learns the parameters used by the classifier
2. In the testing phase, the trained naive Bayes classifier classifies the data in the test_text.txt data file. Use the preprocessed data you generated above.
3. Output the accuracy of the naive Bayes classifier by comparing the predicted class label of each sentence in the test data to the actual class label. The accuracy is the number of correct predictions divided by the total number of predictions.

You may output extra data (e.g. confusion matrices, incorrectly classified sentences) if you think it makes your program's performance more clear, but we should be able to immediately identify and read the accuracy in your output without effort.

A couple of hints:

- As mentioned, strip any punctuation and whitespace when you count and/or identify the words.
- Make sure that you follow the implementation hints given in the lecture. Specifically, do the

probability calculations in log space to prevent numerical instability. Also, use uniform Dirichlet priors to avoid zero counts.
- The accuracy for the training data should be very good (>85%), we will not tell you the accuracy of the test data.

Files

Here are the training and test data sets:

1. Training: training_sentences.txt
2. Test: test_sentences.txt

Results

Your results must be stored in a file called results.txt.

1. Run your classifier by training on training_text.txt then testing on training_text.txt. Report the accuracy in results.txt (along with a comment saying what files you used for the training and testing data). In this situation, you are training and testing on the same data. This is a sanity check: your accuracy should be high i.e. > 85%
2. Run your classifier by training on training_text.txt then testing on test_text.txt. Report the accuracy in results.txt (along with a comment saying what files you used for the training and testing data). We will not be letting you know beforehand what your performance on the test set should be.

What to hand in

All of your source code and the results.txt file. Zip everything up with a zip program. To hand in your assignment, go to the TEACH electronic handin site: https://secure.engr.oregonstate.edu:8000

1. Login to the TEACH system
2. Click on the "Submit Assignment" link on the left hand side
3. Select ProgAssn3 from the dropdown menu, hit submit query
4. Enter the path of your zip file. Hit Submit Query to hand everything in.

How the assignment will be marked

- Pre-processing step (25 points)
- Naive Bayes classifier (25 points)

We will be running your code, so please make sure it will run on the ENGR systems!

References

[1] Tsur, O., Davidov, D. and Rappoport, A. (2010). ICWSM - A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Product Reviews. In Proceedings of the AAAI Conference on Weblogs and Social Media (ICWSM-10).

[2] Davidov, D., Tsur, O. and Rappoport, A. (2010). Semi-Supervised Recognition of Sarcastic Sentences in Twitter and Amazon. In Proceedings of the Fourteenth Conference on Computational Natural Language Learning, (pp. 107-116).