Project 1: Text classification

CS 4650/7650: Natural Language Processing

January 15, 2013

Your goal in this project is to build a classifier for subjectivity and sentiment analysis. You will be given a dataset consisting of documents which are marked as either objective, positive, or negative. You will experiment with various classification techniques on this dataset. Later you will be given unlabeled test data. You will have to submit your best system's predictions on this data, without knowing the ground truth.

You will write two basic classifiers, Naive Bayes and Perceptron. Although they are superficially different, they are both linear classifiers and in fact you can share a lot of code between them.

Submission and honor policy On T-square, you should submit a PDF file with the responses to the "deliverables" required here. Please also submit your code in a tarball, and submit your response files to the test data using the appropriate filenames (described in the last section).

If you have questions for me, please use the Wordpress site. You may use an anonymous account or a pseudonym if you do not wish to have your name associated with your comment or question.

Your work should be your own, so please do not discuss the details of the assignment. You may of course help each other with understanding the ideas discussed in lecture and the readings, and with basic questions about programming in Python. There are implementations and source code for many machine learning algorithms on the internet. Please write the code for this assignment on your own, without using these external resources.

1 Data processing

Download the data from `http://www.cc.gatech.edu/~jeisenst/classes/cs7650_sp13/projects/proj1.tgz`. This tarball currently contains the following files and folders:

- TRAIN and DEV, directories which contain the raw text for the training and development sets
- TRAIN.KEY and DEV.KEY, key files that contain the filenames and labels for each document

- SCORER.PY, a scoring script in Python. The script takes two arguments, the **key file** and a **response file** that contains one line for each label, with a line-by-line correspondence with the key file.
- SENTIMENT-VOCAB.TFF, a sentiment vocabulary [1]

Your first step is to write code that can apply the following preprocessing steps. You will have to run this code fairly quickly on the test data when you receive it, so make sure it is modular and well-written.

- Use NLTK's `word_tokenize` to divide the data into tokens. See `http://nltk.org/api/nltk.tokenize.html`
- Downcase all tokens
- Only consider tokens that are completely alphabetic.

Sanity check I get 11,062 distinct words (types) in the training data. How many do you get?

You will need to represent the word counts for each document. There are at least two good ways to do this in python:

- Represent each file as a `defaultdict` (see `http://docs.python.org/2/library/collections.html`). The keys should be words, and the values should be the counts.

  If you store the data this way, you will end up writing a lot of loops or list comprehensions. If you are used to languages like Java, this may be simplest. It may be a little slower, but I wrote the code for this assignment this way and it runs on my laptop.

- Represent the counts as a sparse matrix (see `http://docs.scipy.org/doc/scipy/reference/sparse.html`). Note that there are types of sparse matrices, and their performance characteristics will be different. For most sparse matrix types, it is inefficient to set the counts directly as

  ```
  counts[x,y] += 1.
  ```

  Instead, it may be better to keep three arrays: document, word, and count. Then you can call a constructor that takes these three arrays as arguments.

  If you store the data this way, you will end up writing a lot of matrix operations. So if you are used to Matlab or Octave, this may be best for you.

2 Word lists

The file `sentiment-vocab.tff` contains a list of words with manually annotated sentiment and subjectivity judgments from Wilson *et al.* [1]. Each line corresponds to a word. For this assignment, the fields to pay attention to are `word1` and `priorpolarity`.

Build a rule-based classifier to assigns all documents to Pos if the number of positive words is greater than the number of negative words, and Neg otherwise.

Deliverable 1 Produce a response file for the development data with these classifications. Show the resulting confusion matrix.

Sanity check The accuracy will be low because you are not classifying any documents as Obj, which is the most common class. But hopefully you will get Pos vs Neg correct most of the time.

Now change your algorithm to classify documents as Obj if the absolute value of the difference between the number of positive and negative words is less than some threshold. Tune this threshold on the training data, using the scoring script to find the threshold that gives the best overall accuracy on the training set.

Deliverable 2 Run the resulting classifier on the dev data, produce a response file, run the scorer, and show the resulting confusion matrix.

3 Naive Bayes

Build a Naive bayes classifier, using relative frequency estimation of the parameters,

$$\theta_{ij} = P(x = j | y = i) = \frac{\text{count}(x = j, y = i)}{\text{count}(y = i)} \tag{1}$$

where $y = i$ indicates the class label and $x = j$ indicates word j. For each class, normalize by the sum of counts of words **in that class**. In other words, $\sum_j \theta_{ij} = 1$ for each dictionary and for all i. You will probably need to work with log values of these parameters, but watch out for the log of zero. Don't forget to also estimate the prior $P(y = i)$.

Deliverable 3 Train a classifier from the training data, and apply it to the development data. Report the confusion matrix and the accuracy.

3.1 Smoothing

Next, try smoothing the theta values by adding pseudo-counts. This is equivalent to maximum a posteriori (MAP) estimation of θ under a Dirichlet prior with symmetric parameter α:

$$\theta_{ij} = \frac{\text{count}(x = j, y = i) + \alpha}{\text{count}(y = i) + V\alpha}, \tag{2}$$

where V is the vocabulary size. Try at least seven different values for α.

Sanity check It's usually best to vary α exponentially, e.g. $\alpha \in \{10^{-4}, 10^{-3}, \ldots\}$.

Deliverable 4 Plot the accuracy on both the development and training data for each value of α. Report the confusion matrix on the development data for the best α.

Deliverable 5 Produce a scatter plot of the $\log\theta$ values estimated under the minimum and maximum values of α that you considered ($\alpha > 0$). Also compute the correlation coefficient. Explain what you see.

Deliverable 6 What words are most predictive of positive versus negative text? You can measure this by $\log\theta_{\text{POS},i} - \log\theta_{\text{NEG},i}$.[1] Use the best α from your dev data. List the top 5 words and their counts for each class. Do the same for the top 5 words that predict negative versus positive. Are they in the sentiment vocabulary?

4 Perceptron

Implement a perceptron classifier. Using the feature-function representation, include features for each word-class pair, and also an "offset" feature for each class. Given a set of word counts $\boldsymbol{x}_i$, a true label y_i, and a guessed label $\hat{y}$, your update will be

$$\begin{aligned}\hat{y} &\leftarrow \arg\max_y \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}_i, y)\\ \boldsymbol{w} &\leftarrow \boldsymbol{w} + \boldsymbol{f}(\boldsymbol{x}_i, y_i) - \boldsymbol{f}(\boldsymbol{x}_i, \hat{y}).\end{aligned}$$

Sanity check If you are not careful, this can be slow. You may need to think a little about how to do this update efficiently. On my laptop, I can make 30 passes on the training data in roughly a minute.

Deliverable 7 Using this update, make several training passes through the data (at least 30). Compute the accuracy on both the training and development sets after each pass. Plot them on a line graph, with the number of iterations on the x-axis, accuracy on the y-axis. Print the confusion matrix for the final iteration.

[1] This is similar to the likelihood ratio test.

Deliverable 8 What words are most predictive of positive versus negative text? You can measure this by $w_{\text{Pos},i} - w_{\text{Neg},i}$. List the top 5 words and their counts for each class. Do the same for the top 5 words that predict negative versus positive. Are they in the sentiment vocabulary? Compare with the same results from Naive Bayes.

5 Averaged perceptron

Implement weight averaging for the perceptron. The description in the reading says to store the weights after every update, but this is not practical. A simple alternative is just to take a running sum of the weights, and keep track of how many times you have added to the sum. Even this can be pretty slow if you update after every instance, as the pseudo-code in the reading suggests. Instead, you can update the running total after every several iterations; the interval is up to you, but update at least once per pass through the dataset.

Remember that the weights during learning should be **identical** to the standard perceptron. The weight averaging only happens at the end, before you evaluate on development data.

Deliverable 9 Just as in the previous task, make at least 30 passes through the data, and plot the accuracy for both the training and development sets. Print the confusion matrix for the final iteration.

Deliverable 10 Scatter plot the weights for the Pos class against the log feature probabilities $\log P(x|\text{Pos})$ from the Naive Bayes classifier (with the best α value). Also compute the correlation coefficient. What do you conclude?

6 Making it better

This part is mandatory for 7650, optional for 4650.

Try to improve one of these classifiers. Here are a few ideas:

- Implement the MIRA algorithm described in the reading notes. This controls the step size of the perceptron update, making bigger steps for instances that are "more" wrong.
- Use bigram features. This is simpler for the perceptron, but with a little thought it can be applied to Naive Bayes too.
- Use feature hashing to make the perceptron much faster and more memory efficient. See this paper: `http://alex.smola.org/papers/2009/Weinbergeretal09.pdf`.
- This paper has several ideas for improving Naive Bayes, try one: `http://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf`.

- Add special features for matching elements in the sentiment vocabulary. You could add features for each combination of polarity and subjectivity, e.g. “negative/weaksubj” or “positive/strongsubj.”
- Do more complicated preprocessing, such as stemming.

Deliverable 11 Clearly explain what you did, and why you thought it would work. Do an experiment to test whether it really does work. Creativity and thoughtfulness counts more than raw performance here.

7 Bake-off

48 hours before the assignment is due, I will send you unlabeled test data. Your job is to produce a response file that I can evaluate. I’ll present the results in class and give the best scorers a chance to explain what they did.

Deliverable 12 Run your best system from the main part of the assignment, without any special tricks. Label the response file `lastname-firstname.main.response`.

Deliverable 13 Run your best system from the “making it better” part of the assignment (even if it was not better than your best system from the previous part). Label the response file `lastname-firstname.special.response`. This part is mandatory for 7650, optional for 4650.

References

[1] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity: An exploration of features for Phrase-Level sentiment analysis. *Computational Linguistics*, 35(3):399–433, August 2009.