CSE 454 Project Part 1

Text Classification

Due October 8, 2015 at Noon

Abstract

In this assignment students will use machine learning to train a text classifier to predict whether the body of an email message is spam or "ham" (i.e. the opposite of spam). Specifically, we provide two folders of 1000 documents, one containing spam and the other ham. You'll convert them into a feature-vector form, train a naive Bayes classifier, and then run it on a test set of 500 documents. Using our answer key, you'll then compute and report precision, recall and F1.

1 Collaboration

This assignment and assignment 2 should be done individually; please do not share code. After assignment 2, you will form teams and work as a group.

2 Programming Language

You may use any language you like, but we recommend that you use Python. Our instructions assume Python and will point you to important NLP and machine-learning tools. Alternative choices exist for Java and other languages, but you are on your own in this regard.

3 Recommended Python Libraries

Make sure you have the latest version of Python 2.X (probably 2.7.10), but not Python 3.X, which has incompatible language extensions. If necessary, use pip to install nltk, numpy, scipy, scikit-learn, and matplotlib.

4 Training & Test Data

The training data is stored in a zip file which expands into two directories, one for positive examples of spam and the other containing ham. Each email body is represented as a text file. There should be 1000 examples of each class. You should partition these files into a training set and a development set, keeping (say) 150 files of each class in the development set. It's up to you, but this split will allow you to tune the feature set and regularization.

The test set is in another folder — it contains an unlabeled mix of positive and negative examples (i.e., both spam and ham). Don't look at the contents of these files. And don't run your learned classifier on them until you have finished tuning it (using the development set) and won't change your code any more. This is standard protocol for how to do experiments in machine learning and eliminates or minimizes the chance of over-fitting to the test set. Since you won't be graded on how well your classifier does, there's no incentive to 'cheat' here. But see http://bit.ly/1NNwJls for the recent story of how Baidu got banned from the ImageNet competition for this tactic.

An answer key is provided in .csv format, with each line containing the filename of one message and either a zero, indicating a "ham" email, or a one, indicating spam. A Python script, *compare_results.csv*, is also provided that will perform a comparison between this answer key and the results of your classifier, if provided in the same csv format. After you have run your classifier on the test set, save the results to a .csv file and run: `python compare_results.py my_results.csv`, where my_results.csv is the filename containing the classifications proposed by your classifier.

5 Assignment

You should write functions to perform the following:

5.1 Feature Selection

The simplest feature representation is a bag of words. Since there are so many possible words, it's useful to reduce the set of words considered to a smaller, informative set. The easiest thing to do is to throw out stop words, perhaps using http://www.ranks.nl/stopwords. A more sophisticated approach scans the corpus of positive and negative training data, tokenizes it, e.g. using the nltk.tokenize package, computes a frequency histogram, and discards the most common and most rare words. A simple approach simply discards all words that don't occur in at least $k = 2, 3$ or 4 documents. (Remember, don't look at the words in the test set when doing this analysis). A variation on this approach is to stem words before computing their frequency, e.g. using the Snowball stemmer in nltk.stem. When you are done, you'll have a list of words (or stems) that form your features. Sort these so they are in a canonical order.

Note that you should only use the actual text of the messages when selecting features, not file metadata.

5.2 Convert to Features

This code should convert a document into a feature vector. Each element of this vector can be Boolean (does this word appear in the document, yes or no?) or an integer (the number of times it occurs). Boolean is slightly simpler, but may not work as well. Obviously, if you used stemming when selecting features, you need to stem during the process of featurizing a document.

5.3 Training

Naive Bayes classifier For an introduction to the Naive Bayes classifier, see section 5 of the NLTK introduction to text classification: http://www.nltk.org/book/ch06.html The NLTK library itself contains a basic version of the Naive Bayes classifier: http://www.nltk.org/api/nltk.classify.html

However, if you're interested in experimenting with a version of the Naive Bayes with more options, such as the multinomial Naive Bayes with Laplace smoothing, you can look into the version available in scikit-learn: http://scikit-learn.org/stable/modules/naive_bayes.html A wrapper for some scikit-learn functions is available through NLTK as described in the NLTK link above.

6 Evaluation

Train your system on your chosen subset of training data and evaluate on your development set. Typical F1 scores are 90%. If you wish, you can try changing the feature-set or other aspects of your learner, or try SVMs or decision forests. Once you are satisfied, resolve not to change your code any further and retrain your system on all provided training data (including the development set) to generate your final model.

Use the test set to compute precision, recall and balanced F1.

7 Turn-in

Please hand in the following using the following Catalyst Dropbox:

- Your code
- Answers to the following questions
    1. How did you compute features?
    2. Did you try different settings or learners?
    3. What was your final precision, recall and F1?
    4. Is this good enough for use in a practical spam detector? Improving precision usually hurts recall and vice versa - which is more important for this application?