CS6200 Information Retrieval
Homework7: Unigram/Bigram Classifier, Spam

Objective

Build a Spam Classifier using Machine Learning and ElasticSearch.

We have created a discussion thread in Piazza. It is meant to be a supportive space to help each other succeed in this assignment. Whether you're encountering hurdles, have discovered something interesting, or want to share your progress, this is the place!

Data Set

You will be using the trec07_spam document set that is annotated for spam. It is available in the "data resources" folder.
First read and accept agreement at http://plg.uwaterloo.ca/~gvcormac/treccorpus07/. Then download the 255 MB Corpus (trec07p.tgz). The html data is under data/ and the labels ("spam" or "ham") are under full/.

You may need to think about storage (ElasticSearch is recommended, but not required). You will need to use a library to clean the html into plain text before storage. You don't have to do stemming or skipping stopwords (This is optional). Eliminating some punctuation might be useful.

Cleaning Data is Required: A unigram is a word. As part of reading/processing data you need to filter data such that anything that is not an English word or small number is removed. It is ok to have some invalid unigrams passing the filter as long as they are not overwhelming the set of valid unigrams. Those may look like words (e.x. "artist_", "newyork", "grande"). You can use any library/script/package for cleaning, or share your cleaning code (*but only the cleaning code*) with the other students.
Make sure to have a field "label" with values "yes" or "no" (or "spam"/"ham") for each document.

Partition the spam data set into TRAINING set 80% and TESTING set 20%. One easy way to do so is to add a field "split" to each document in ES with values "train" or "test". You can assign the values randomly by following the 80%-20% rule. You will end up with 2 feature matrices, one for training and one for testing (different documents, same exact columns/features). The spam/ham distribution is roughly a third ham and two thirds spam; you should have a similar distribution in both the TRAINING and TESTING sets.

Part1: Manual Spam Features

Trial A. Manually create a list of ngrams (unigrams, bigrams, trigrams, etc) that you think are related to spam. For example : "free" , "win", "porn", "click here", etc. These will be the features (columns) of the data matrix.
Trial B. Instead of using your unigrams, use the ones from this list; rerun the training and testing.

You will have to use ElasticSearch querying functionality in order to create feature values for each document. There are ways to ask ES to give all matches (aka feature values) for a given ngram, so you don't have to query (ngram, doc) for all docs separately.
If you dont use ES, you will have to explain to the TAs how you match unigrams across documents for values (it should be similar to HW2 indexing basic procedure)

For part 1, you can use a full matrix since the size won't be that big (docs x features). However, for part 2 you will have to use a sparse matrix, since there will be a lot more features.

Train a learning algorithm

The label, or outcome, or target are the spam annotation "yes" / "no" or you can replace that with 1/0.

Using the "train" queries static matrix, train a learner to compute a model relating labels to the features on TRAINING set. You can use a learning library like SciPy/NumPy, C4.5, Weka, LibLinear, SVM Light, etc. The easiest models are linear regression and decision trees.

Test the spam model

Test the model on the TESTING set. You will have to create a testing data matrix with feature values in the same exact way as you created the training matrix: use ElasticSearch (or as approrpiate for your storage) to query for your features, use the scores are feature values. Remember that features have to be consistent across train and test data.

1. Run the model to obtain scores
2. Treat the scores as coming from an IR function, and rank the documents
3. Display first few “spam” documents and visually inspect them. You should have these ready for demo. *IMPORTANT* : Since they are likely to be spam, if you display these in a browser, you should turn off javascript execution to protect your computer.

Train/Test 3 Algorithms

1. decision tree-based
2. regression-based
3. Naive-Bayes

Part 2: All unigrams as features (MS students only)

A feature matrix should contain a column/feature for every unigram extracted from training documents. You will have to use a particular data format described in class (note, toy example), since the full matrix becomes too big. Write the matrix and auxiliary files on disk.

Given the requirements on data cleaning, you should not have too many unigrams, but still enough to have to use a sparse representation.

Extracting all unigrams using Elastic Search calls

This is no diffeernt than part1 in terms of the ES calls, but you'd have to first generate a list with all unigrams.
If you dont use ES, this can be a tricky step, but there are python (poor) or java (better) libraries to extract all unigrams from all docs. Keep in mind that extracting all ngrams (say up to n=5) is a difficult problem at scale.

Training and testing

Once the feature matrices are ready (one for training, the second for testing), run either LibLinear Regression (with sparse input) or a learning algorithm implemented by us to take advantage of the sparse data representations.

Feature analysis

Identify from the training log/model the top (most important) spam unigrams. Do they match your manual spam features from part 1?

Extra Credit

EC1(part1): Test the spam model on your crawl data from HW3.

Check manually if the top 20 predicted-spam documents are actually spam.

EC2(part2): Extract Bigrams (besides unigrams) as features

Add not just the unigrams, but all bigrams from training documents

EC3(part2): Extract skipgrams as features

Replace Unigrams and bigrams with general skipgrams (up to length n=4 and slop=2) from training documents

Rubric

Check Canvas for a detailed rubric