CS6200 Information Retrieval

Homework7 : Unigram/Bigram Classifier, Spam

Objective

Build a Spam Classifier using Machine Learning and ElasticSearch.

Data

Consider the trec07_spam set of documents annotated for spam, available “data resources”.
First read and accept agreement at http://plg.uwaterloo.ca/~gvcormac/treccorpus07/. Then download the 255 MB Corpus (trec07p.tgz). The html data is in data/; the labels ("spam" or "ham") are in full/.

Index the documents with ElasticSearch, but use library to clean the html into plain test first. You dont have to do stemming or skipping stopwords (up to you); eliminating some punctuation might be useful.
Cleaning Data is Required: By "unigram" we mean an English word, so as part of reading/processing data there will be a filter step to remove anything that doesnt look like an English word or small number. Some mistake unigrams passing the filter are acceptable, if they look like words (e.x. "artist_", "newyork", "grande") as long as they are not overwhelming the set of valid unigrams. 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 TRAIN 80% and TEST 20%. One easy way to do so is to add to each document in ES a field "split" with values either "train" or "test" randomly, following the 80%-20% rule. Thus there will be 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 TRAIN and TEST sets.

matrix 1

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, for each feature. There are ways to ask ES to give all matches (aka feature values) for a given ngram, so you dont have to query (ngram, doc) for all docs separately.

For part 1, you can use a full matrix since the size wont 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.

matrix 2

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 TRAIN 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 TEST 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 to query for your features, use the scores are feature values.

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

} 3 separate runs for each matrix (libraries)

matrix 3

Part 2: All unigrams as features

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 many 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.

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 part1

5 points	A reasonable list of spam features
15 points	ES queries for feature values (separately training set and testing set)
10 points	Training the model
10 points	Run model on testing data and display top spam docs

Rubric part 2

15 points	A proper SPARSE static matrix setup for feature values and labels including unigrams
15 points	ES calls to extract all unigrams feature values
15 points	Training successfully a learning algorithm that uses sparse data format
15 points	Evaluation on test set