

Project 2: Text Classification

COMS 6998: Search Engine Technology, Fall 2007

In this project, you will experiment with different machine learning methods for text classification.

Here's an overview of what you'll do:

- 1 Extract feature vectors from a corpus of labeled documents.
- 2 Implement feature selection based on Chi-square.
- 3 Use SVM light to train a model using your feature vectors.
- 4 Implement a simple text classifier, a Naïve Bayes, a perceptron or a decision tree classifier. Ask the instructor if you would rather write a different one or implement an SVM from scratch.
- 5 Run experiments using the two different models and different numbers of features and training set size. Report and analyze the results.

Documents as Feature Vectors

Each document can be represented as in the following table:

DOC WORD COUNT123 base 2123 bat 5123 ball 1123 score 2234 puck 2234 goal 3234 score 1...

or as the following vectors:

[base bat ball score puck goal] f(DOC123) = [2 5 1 2 0 0], class = baseball f(DOC234)=[0 0 0 1 2 3],class=hockey

Perceptron Model

You need to learn the weights w_i that correspond to each of the dimensions of the vector. Let w^t be the transposed matrix of weights. Then the decision boundary will be $w^t \cdot f(x) = 0$, or in other words:

if $w^t \cdot f(x) > 0$: classify x as baseball

if $w^t \cdot f(x) < 0$: classify x as hockey

For example, if the weights w were: w

$$= [1 \ 1 \ 1 \ 0 \ -1 \ -1]$$

Then: $w^t \cdot f(\text{DOC123}) = 1*2 + 1*5 + 1*1 + 0*2 + (-1)*0 + (-1)*0 = 8$ Since $8 > 0$, DOC123 is about baseball

$w^t \cdot f(\text{DOC234}) = 1*0 + 1*0 + 1*0 + 0*1 + (-1)*2 + (-1)*3 = -5$ Since $-5 < 0$, DOC234 is about hockey

Note that you can also have an intercept (bias) w_0 which doesn't correspond to a particular word in the input document. In that case, the decision boundary is going to be $w^t \cdot f(x) + w_0 = 0$.

Naïve Bayes

The Naïve Bayes model is covered in detail in the book and lecture notes. Use add one (Laplace) smoothing or a more advanced method to avoid zero probabilities. Make sure to use log probabilities instead of actual probabilities in order to avoid floating point underflow problems.

Feature Selection

By default, you should use all words (after proper tokenization) that appear in the training documents as features for your classifier. Then you will implement feature selection using the Chi-square method and pick the top M most discriminative features (details below).

Resources

- **testing/training data** – The corpus contains a set of 25000 documents from the blogpulse corpus (<http://blogpulse.com>). This is one of the 2004 political blog data set: The Political Blogosphere and the 2004 U.S. Election: Divided They Blog (Lada A. Adamic and Natalie Glance, LinkKDD-2005, Chicago, IL, Aug 21, 2005.) . It has the posts made by the top 20 liberal and top 20 conservative bloggers.

The documents are divided evenly into training and test sets for you, with an even split between the two topics in each set. In the training set you will see two directories ‘0’ and ‘1’. These are the *labels* for the xml files within them.

```
~cs6998/hw2/data/training/0
~cs6998/hw2/data/training/1
~cs6998/hw2/data/testing/0
~cs6998/hw2/data/testing/1
```

- **SVM implementation** – We will be using SVM Light(http://www.cs.cornell.edu/People/tj/svm_light/). It is installed at

```
~cs6998/tools/svm_light
```

It's very straightforward to use. To train a model, type:

```
svm_learn example_file model_file
```

To classify test data based on a trained model, type:

```
svm_classify example_file model_file output_file
```

(Of course, there are additional options you can play with.)

- **Feature extraction** - You can use your code from project 1 or clairlib to extract the word frequencies from each document or write your own extractors. Stemming and lowercasing are reasonable things to do. For simplicity, you should output your feature vectors in the SVM light file format, and write your classifier to accept them in the same format.

Deliverables

You can work in languages like perl or java. It's probably easiest for you if you continue with your code from project 1. (Third-party libraries like clairlib and Lucene would also be okay to use.)

(Optional) Install script – If you use any third-party libraries that need special installation, please include an `./installscript`. Note that you are NOT allowed to use third party implementations of NB, perceptron or decision tree.

Feature extraction – Your feature extractor reads documents from a corpus and creates feature vectors in the SVM light file format. It optionally performs Chi-square feature selection.

```
./extract_features N train_dir test_dir output [M feature_file]
```

- N– Number of documents to use from each class in training and testing.
- train_dir test_dir – Paths to documents in the training set and test set. You should take the first N documents from each class in each directory. (With N=500, you would use 500 baseball documents and 500 hockey documents from training_dir for training, and the same amount from testing_dir for testing.)
- output– You should produce labeled feature vectors in output.train and output.test for the training and test sets, respectively.
- M– (optional) number of features to use for chi-square feature selection
- feature_file– (optional) When feature selection is used, there is also another parameter, feature_file. You should output the chi-square scores for all words to this file (e.g., each line could be [word] [score]).

Use SVM light to train a classifier based on your extracted features and test it on your training data. (This is a good check to make sure your features are getting extracted properly.)

Classifier -Implement Naïve Bayes, perceptron or decision tree classifier. To train a model, we will call:

```
./learn examples.train model_file
```

The model_file should contain parameters for your model (for instance, for the perceptron this is a list of words and their weights).

To classify examples in the test set, we will call:

```
./classify examples.test model_file output_file
```

The output_file should contain a tab-separated line for each document as follows:

```
docid score computed_class correct_class y/n
```

The last column outputs y if the computed_class matches the correct_class and n otherwise. Your classify program should also report the combined accuracy (over both classes) on the testing set, in other words:

$$\text{Accuracy} = \text{Number of correctly labeled test docs} * 100 / \text{Total number of test docs}$$

- Run experiments with different numbers of features and different numbers of labeled examples. The following table should be used to present a summary of your experiments. You should run two sets of experiments: one with your classifier and one with SVM Light.

		Number of features			
		10	100	1000	All
Size of training set (per class)	20				
	100				
	200				
	300				

Note that if you use a perceptron, it may take a long time to converge, or not converge at all. When this happens, you should restart with different random initial weights.

Deliver a README file with a description of your classifier, experimental results (table above) and a discussion of your results. This can be in txt, ps or pdf. (You can use a pdf printer to print from word to pdf.)

You should submit all your source code with documentation.

Discussion

A significant portion of this assignment is a discussion of your observations. Some points that you could consider are: -

Comparing the accuracy and efficiency of your classifier with that of SVM Light

-Explaining your choice of classifier (Naïve Bayes/perceptron)

-Discussing the effect and implications of varying the number of features and number of training examples on accuracy

-For the perceptron, how does the value of eta affect the convergence rate? What about the number of features/training size?

-For Naïve Bayes, which smoothing method did you use, and why?

These are just suggestions – feel free to add or change topics in your discussion as you see fit.

Grading

- Experimental write-up: 40 pts
 - o Tables
 - o Discussion
- Working code with documentation: 60 pts
 - o Classifier
 - o Chi-square feature selection
 - o Feature extraction

Extra credit

You can also attempt to incorporate one of the following techniques for up to 10 extra points. Extra credit will not be considered unless the technique used is well documented and shown to yield better results.

- Other feature selection algorithms
 - Different methods for smoothing
 - Your own implementation of a kernel to use with SVM
 - Extraction of additional useful features (e.g., phrases)

Note: We encourage you to write code which is clairlib compliant. We will pick the best implementation of Decision Tree for clairlib.

Submission

The project is due by 26th October 2007. Each day you are late, you will lose 10 points. We will not accept projects after 5pm on the due date.

To submit:

 Create a directory with your UNI

 Copy the source code files into the directory and include all the other files that are necessary for your program to run.

 Go back one level and tar the directory. Give your UNI as the tar file name

<XXX123.tar.gz>

- Send the tar file to Akshay and Samreen. The subject of the mail should be “SET HW2 <your uni>” Please write all the necessary information into the Readme and not in the mail. You may send the submission to us again (Don’t change the subject name).

If you have questions, or can’t get something working, email us at

cs6998-set-QA@lists.cs.columbia.edu (you’ll get a faster response from this list than emailing us individually.)