Text Categorization with the Naive Bayes Classifier

Text categorization project for ECE467

Caleb Zulawski

Classifier information

The classifier is a Naive Bayes classifier implemented in Python 3. A majority of the text processing and tokenization are done with NLTK, and require the NLTK `punkt` and `wordnet` packages. Additionally, it requires NumPy and SciPy for calculations. To store term frequencies and probabilities, dictionaries are used. The classification is performed using log probabilities.

Commands

`classify.py`

The `classify.py` program trains on a corpus of documents, and produces a prediction of classes on another corpus. Its usage is as follows:

```
usage: classify.py [-h] [-s STOPWORDS] training testing predictions
```

The stopwords argument allows you to optionally supply a stopwords list. If none is supplied, the included stopwords list, `stopwords.txt`, is used.

`kfold.py`

The `kfold.py` program splits the supplied training corpus using k-folding. Its usage is as follows:

```
usage: kfold.py [-h] [-k K] [-s STOPWORDS] training predictions
```

The default number of splits is 10. The k argument allows another number to be chosen, and the stopwords option is the same as above.

Scripts (`getcorpora.sh` and `kfold.sh`)

The `getcorpora.sh` script simply downloads the sample corpora. The `kfold.sh` script performs k-folding on the supplied corpus and prints the results with the `analyze.pl` program.

Tokenizer

The tokenizer used several techniques to improve performance.

1. The document is converted to all lowercase letters. This improves performance negligibly, but makes later processing slightly easier.
2. The tokenizer uses the NLTK Punkt sentence tokenizer to split the document into sentences, then the NLTK Punkt word tokenizer to split that sentence into words.
3. The tokens are checked against a stop word list, and removed if present. The stop words list is from http://www.ranks.nl/stopwords (http://www.ranks.nl/stopwords) .
4. Tokens that do not contain any letters (i.e. contains only symbols and/or numbers) are removed. This may not be beneficial for all corpora, but appears to work well in this domain.
5. Stemmers and lemmatizers were both tested, and all increased performance by small amounts. The NLTK WordNet lemmatizer implementation was selected as it appeared to perform slightly better than the others on the second provided corpus.

Smoothing

The smoothing was done with add-one (Laplace) smoothing. Other smoothing techniques would likely further improve performance. If I had implemented n-grams, I would have gone with Kneser-Ney smoothing.

Cross-validation

Cross-validation was done with k-folding, generally with k = 10. The permutation of the document list was randomly generated, and split into 10 parts. The sets were evaluated, trained against the other sets. All of the predicted labels were then stored, and could be compared to the original labels.