Applied Textmining

5. Assignment: Text Categorization

Please update the code from the GIT repository at *https://github.com/dimalabs/applied-textmining* by pulling the latest version of the project. to accomplish that simply type the following line into *GIT Bash*:

```
git pull
```

Then have a look at *de.tuberlin.dima.textmining.assignment5.TopicClassifier*. This is the class in which you will implement the naive bayes classifier. It will be evaluated on a data set of postings from 20 newsgroups (check out `http://people.csail.mit.edu/jrennie/20Newsgroups/` for a description of the data). The test class for this assignments takes the classifier and performs 10-fold cross-validation on the data. In cross validation, the data is partioned into k buckets. for each bucket a model is trained on the $k-1$ remaining buckets and then evaluated on the held out bucket. At the end the model has been evaluated on all the data without ever testing on training data. When data is scarce the number of buckets is set equal to the number of data points - leading to what is called leave-one-out cross-validation. We included the mini version of the data set in the project in GitHub which only contains 100 randomly sampled postings per newsgroup. We will evaluate your code on the entire dataset however.

Your Task

You will have to implement the two function `train(List<labeledDocument> trainingDocs)` which estimates the model parameter on the training data and `public String classifyDocument(String document)` which classifies an unseen document and returns the guessed label.

Naive Bayes Classifier: Theory

You can also have a look at `http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html` for an introduction to Naive Bayes text classification. A naive Bayes classifier models a joint distribution over a label Y and a set of observed random variables, or features, $(F_1, F_2 \dots F_n)$, using the assumption that the full joint distribution can be factored as follows (features are assumed to be conditionally independent given the label):

$$P(F_1, F_2 \dots F_n) = P(Y) \prod_i P(F_i|Y)$$

To classify a datum, we can find the most probable label given the feature values for each pixel, using Bayes theorem:

$$P(y|f_1,\ldots,f_m) = \frac{P(f_1,\ldots,f_m|y)P(y)}{P(f_1,\ldots,f_m)} = \frac{P(y)\prod_{i=1}^{m} P(f_i|y)}{P(f_1,\ldots,f_m)}$$

$$argmax_y P(y|f_1 \ldots f_m) = argmax_y \frac{P(y)\prod_{i=1}^{m} P(f_i|y)}{P(f_1,\ldots,f_m)} = argmax_y P(y)\prod_{i=1}^{m} P(f_i|y)$$

Because multiplying many probabilities together often results in underflow, we will instead compute log probabilities which have the same argmax:

$$argmax_y \log P(y|f_1 \ldots f_m) = argmax_y \left\{ \log P(y) + \sum_{i=1}^{m} \log P(f_i|y) \right\}$$

To compute logarithms, use `Math.log()` function.

Put simply, you will have to compute the most likely label given the data (text in the posting to be classified). This means you have to compute the probability for each label and then return the label which recieves the highest probability from your model for the document in question. The class `edu.stanford.nlp.stats.ClassicCounter` might be of help in this assignment again.

Parameter Estimation

Our naive Bayes model has several parameters to estimate. One parameter is the prior distribution over labels (the names of the newsgroups), $P(Y)$. We can estimate $P(Y)$ directly from the training data:

$$\hat{P}(y) = \frac{count(y)}{n}$$

where $count(y)$ is the number of training instances (newsgroup postings) with label y and n is the total number of training instances. The other parameters to estimate are the conditional probabilities of our features given each label y: $P(F_i|Y = y)$, where is the conditional probability of a word w occurring in a posting of class(topic) y. We do this for each possible feature value (each word in the vocabulary).

$$\hat{P}(w|Y = y) = \frac{count(w, y)}{\sum_{w'} count(w', y)}$$

where $count(w, y)$ is the number of times word w occured in the training examples of label y.

Smoothing

Your current parameter estimates are unsmoothed, that is, you are using the empirical estimates for the parameters . These estimates are rarely adequate in real systems. Minimally, we need to make sure that no parameter ever receives an estimate of zero, but good smoothing can boost accuracy quite a bit by reducing overfitting. In this assignment, we use Laplace smoothing, which adds k counts to every possible observation value:

$$P(w|Y=y) = \frac{count(w,y)+k}{\sum_{w'}(count(w',y)+k)}$$

If $k=0$, the probabilities are unsmoothed. As k grows larger, the probabilities are smoothed more and more. You can use part of the training data as a validation set to determine a good value for k if you want - this is optional however! Note: don't smooth the prior label probability P(Y).

Deadline

Because this assignment was posted late, you have until the 23rd of November 2011 to upload your solution as a patch in the ISIS system .