CS 678 Machine Learning
Programming Assignment #2

Naïve Bayes Document Classifier

Specification

The basic idea is to write a program that, given a collection of training data consisting of category-labeled documents, "learns" how to classify new documents into the correct category using a Naïve Bayes classifier.

Background

The Naïve Bayes algorithm uses probabilities to perform classification. The probabilities are estimated based on training data for which the value of the classification is known (i.e. it is another form of Supervised Learning). The algorithm is called "naïve" because it makes the simplifying assumption that attribute values are completely independent, given the classification.

Resources

- A tutorial describing the operation of the Naïve Bayes classification algorithm has been posted on the course web page.

Data Sets

Sample datasets have been posted on the course web page. They are composed of documents obtained from 20 forums (or newsgroups) focusing on different topics (e.g. religion, baseball, politics). Both training (11293 documents) and test (7527 documents) data sets are provided. Dataset format:

classification documentText   // one document (i.e. example) per line

The data has been pre-processed:

- All punctuation, numbers and special characters have been removed.
- All words have been converted to lowercase.

A filtered (stemmed) version of both datasets has been further pre-processed:

- All one- and two-character words have been removed.
- All SMART stopwords have been removed (i.e. words such as articles, adjectives, pronouns that would be expected to appear in every document). Note: this is the same technique employed by modern search engines.
- Porter's Stemmer has been applied. This process removes common English word endings; so, for example, both *biking* and *biked* would become *bik*.

Implementation

Implement the Naïve Bayes algorithm to create a document classifier

Problem: determine what class (C) a new document (D) belongs to.

Approach:

- Each word position in a document is an attribute
- The value each attribute takes on is the word in that position

Simplifying assumption:

- Word probability is independent of words in other positions

Learn

1. Collect all words occurring in the Sample documents
   - Vocabulary ← set of all distinct words

2. For each class c_j (document type) in C
   - Docs_j ← training documents for which the classification is c_j
   - Probability estimate of a particular class:
     $P(c_j) = |\text{Docs}_j| \;/\; |\text{training documents}|$
   - Text_j ← create a single document per class (concatenate all Docs_j)
   - n = total number of word positions in Text_j
   - For each word w_k in Vocabulary
     n_k = number of times w_k occurs in Text_j
   - Estimate of word occurrence for particular document type:
     $P(w_k \mid c_j) = (n_k + 1) / (n + |\text{Vocabulary}|)$

   Probability of *k*th word in vocabulary, given a document of type *j*

Classify

1. Return classification C_{NB} for new document D
   - Use Naïve Bayes classifier as described in class
   - Positions ← all word positions in D containing tokens in Vocabulary (where a_i denotes word found in ith position)

$$C_{NB} = \max_{c_j \in C}\left(P(c_j) \prod_{i \in Positions} P(a_i \mid c_j)\right)$$

Notes

- What would happen if a word *w* never occurred in a particular document type C_j in the training set? The algorithm would assign a probability of 0 to $P(w|C_j)$. Now what if it *did* appear in a test document? Follow through the computation; any document containing word *w* would never be classified as type C_j. That's the reason for the "smoothing" step in the word occurrence probability computation (adding 1 to the numerator and |Vocabulary| to the denominator.
- What about the fact that we're multiplying a large number of very small probabilities? The computation might result in arithmetic underflow – a number too small to be accurately represented in a computer. You can search for an arbitrary-accuracy library. Or you can recall this relationship:

  $$log\ (a * b) = log(a) + log(b)$$

  and apply it to your computations.

Requirements

You may use either set of training/test data (original or stemmed). Obviously, train your classifier using the training data, and then evaluate its effectiveness using the test data. You may use the language or platform of your choice. You may modify the input files in any way you choose. You may use alternative data structures to those described in the Implementation. However, your program must do all calculations and implementation of the NB algorithm (i.e. do not use an existing library package that implements NB).

Submit a written report and be prepared to present your solution to the class:

- □ Include complete documentation of your code.
- □ Describe your approach, any interesting problems encountered or experiments performed, techniques or data structures used, etc.
- □ Calculate the effectiveness of your classifier (in other words, demonstrate that it can beat random guessing).
- □ Include a discussion/analysis of your results.

Further Investigation

- □ Experiment with alternative/additional data pre-processing.
- □ Characterize misclassification
- □ Locate alternative datasets of interest to you
- □ ?