CS 678 Machine Learning
Programming Assignment #2

Naïve Bayes for Spam Classification

Specification

The idea is to write a program that, when given a collection of training data consisting of labeled (Spam | Ham) text messages, "learns" how to classify (or tag) new messages correctly using a Naïve Bayes classifier. Basically, write a spam filter.

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 a 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 Github page.

Data Set

A sample dataset has been posted to the Course Github page. It consists of a collection of 5574 labeled SMS text messages. Dataset format:

classification TextMsg   // one text message (i.e. example) per line

The data has not been pre-processed.

Assignment

Implement the Naïve Bayes algorithm to create a spam classifier (filter)

Problem: Determine what class (C) a new message (M) belongs to.
There are two classes, 'ham' and 'spam'.

Approach:

- Each word position in a message 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 messages
   - Vocabulary ← set of all distinct words

2. For each class c_j (message type) in C
   - $Msgs_j$ ← training messages for which the classification is c_j
   - Probability estimate of a particular class:
     $P(c_j) = |\,Msgs_j\,| \;/\; |\text{ training messages }|$
   - $Text_j$ ← create a single file per class (concatenate all $Msgs_j$)
   - 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 message type:
     $P(w_k \mid c_j) = (n_k + 1) / (n + |Vocabulary|)$

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

Classify

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

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

Implementation Notes

- What would happen if a word *w* never occurred in a particular message 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 message? Follow through the computation; any message 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 (i.e. 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 will need to create your own training/test sets out of the collected data.
- You may use alternative data structures to those described here. 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, 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 data pre-processing:
  - □ Convert all words to lowercase.
  - □ Remove all punctuation. Remove all short words (one- and two-characters). Remove all stopwords such as articles, adjectives, and pronouns that would be expected to appear in every document.
  - □ Alternatively, count punctuation separately as words (may be useful if some punctuation occurs far more frequently in spam).
  - □ Try applying a stemmer (e.g. Porter's Stemmer). This process removes common English word endings; so, for example, both *biking* and *biked* would become *bik*.
- Characterize misclassification
  - □ This would be a good application of Precision / Recall analysis.
- Experiment with various partitioning of the Training/Test sets.