CS 678 Machine Learning
Programming Assignment #2 Part II (Optional)

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. The assumption helps make the math tractable and the computation feasible. Although it is obviously not true for email or text messages (words in an email/text are clearly associated with each other), it is surprisingly effective.

Resources

- A tutorial describing the operation of the Naïve Bayes classification algorithm has been posted to BlackBoard.

Data Set

A sample dataset (`textMsgs.data`) has been posted to Blackboard. It consists of a collection of 5574 labeled SMS text messages. About 13% of the messages are spam.

Dataset format:

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

Dataset details:

- The data has not been pre-processed.
- No information is available regarding the order of the text messages.
- They're texts, so contain lots of abbreviations.
- The messages were largely acquired in the U.K. and British-speaking countries. You may notice alternative spellings, acronyms, conventions, etc.

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 (applied to the Training Set)

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
   - Estimate the probability of each class:
     $Msgs_j$ ← training messages for which the classification is c_j
     $P(c_j)$ = #$Msgs_j$ / #training messages |
   - Probability of k^{th} word in vocabulary, given a message of type *j*
     $Text_j$ ← 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:
     $P(w_k \mid c_j) = (n_k + 1) / (n + |Vocabulary|)$

Classify (applied to the Test set)

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 i^{th} position)

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

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: because of the multiplied probabilities 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. We "pretend" each word is seen at least once.

- Note that we are 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 utilize an arbitrary-accuracy library, or recall these relationships:

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

  and apply to your computations to avoid underflow.

Requirements

- You will need to create your own Training/Test sets out of the collected data.
- You may use alternative aggregation methods to those described in the Assignment.
  - The Python dictionary should be particularly useful.
- 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 (single PDF); be prepared to present your solution in 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 various partitioning of the Training/Test sets.
- Experiment with data pre-processing. Does it help to:
  - Convert all words to lowercase
  - Keep/remove all numbers and/or punctuation
  - Remove all stopwords such as articles, adjectives and pronouns (that would be expected to appear in every document).
- Characterize misclassification
  - This would be a good application of Precision / Recall analysis.