Katie McCorkell

Kmccork 0822555

Machine Learning 446

Winter 2014

Homework 3

**2.2 Naïve Bayes Classifier**

To run my code: run the NiaveBayesClassifier.java file with a parameter given.

The code works as follows: first it learns on the training set. To do this, it calculates the percentage of spam emails out of the whole training set. This is **totalCount.pSpam**. Also, out of all the words in the emails, it makes a wordMap<String, WordCount>. WordCount calculates the probability of a given word in spam and non-spam email. This is used for probability (word | spam) and p(word | not spam). For example:

P(word | spam) = 

For the testing set: a new example email is spam if the probability that it is spam is greater than the probability that it is ham. The probability that it is spam is calculated by the prior probability of spam (totalCount.pSpam) multiplied by the product of (for each of the email’s words) p(word|spam). If a word occurs more than once, each occurrence is factored in. In my code I added the logs of the probabilities instead of multiplying the probabilities, in order to avoid underflow. Since we are taking the max, this is acceptable.

The accuracy is the number of emails that the classifier correctly predicts if they are spam or not.

I obtained an accuracy of 90.2%

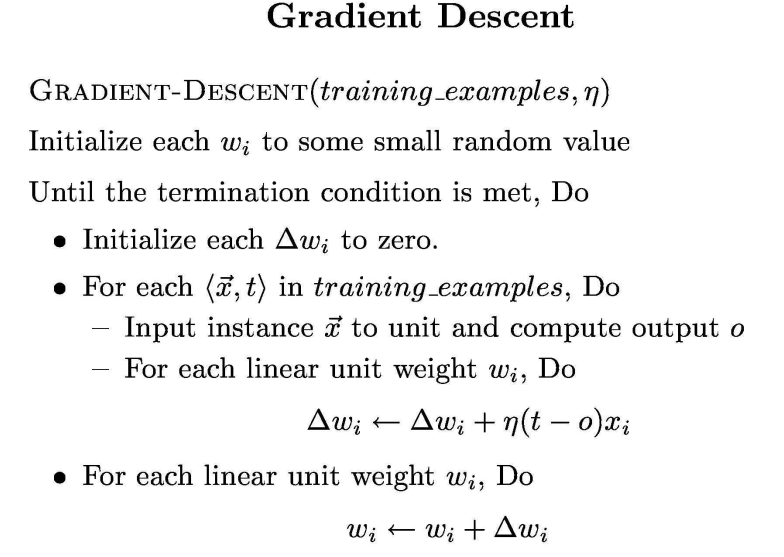
2.3 I implemented the smoothing parameter, as described above it is the alpha value. The most effective was when alpha equal to 10 or 100, however this was based on the test data, so this may well be a case of over fitting to the test data. It would be better if I split the training data up to also include a verification set, and then tested on it to find the optimal parameter.

|  |  |
| --- | --- |
| Parameter | Accuracy on Given Test Data |
| .1 | .902 |
| 1.0 | .902 |
| 10.0 | .904 |
| 100.0 | .904 |
| 1000.0 | .871 |

**Problem 2: Textbook problems re: Neural Networks**

**MITCHELL, 4.5**

The gradient descent algorithm should be implemented as in the course lecture slides on Nueral Networks, slide 15 (below), however a different formula for ∆wi should be used.



∆wi should be equal to –ŋ () as before, but a different value should be used for.

 should be derived as follows

 = 

 =  where  is the o given in problem 4.5, see next page

= 

 = 

 = 

 refers to the equation given in the problem:

This can be rewritten as 

Thus, the derivative of this with respect to weight i, for any  is 

Since only the i terms remain after taking the derivative with repsect to weight i.

This is used in my work on the previous page.

**MITCHELL, 4.10**

This can be implemented by multiplying each weight by the constant (1-2γŋ) upon each iteration.

**PROBLEM 4: BAGGING**

Run the BaggingEnsemble.java file with a integer argument (N).

My code works as follows:

N classfiers are trained on N data sets which are created by sampling with replacement from the training data file. The classifier in this case is the Weka J48 tree. Next each test example is classified based by each of the N decision trees. For each test example, my learner votes based on the majority of the N decision trees’ votes. The vote of my learner is compared to the true vote in order to measure accuracy.

With the parameter of 35 for the bagging size, the BaggingEnsemble obtained accuracies around 84.375%. Sometimes it had accuracy of 87.2%. Changing the bagging parameter to 10 or 100 didn’t have a signficant effect, perhaps because the data set is rather small. Changing the parameter to 1 had an overall negative effect, with the learner often getting accuracies of about 71%. One tree on the original data set has an accuracy of 84.375%. So a bagging parameter of 1 is not as good as a simple tree because it learns on only one bootstrapped sample, which is less data than if you don’t bootstrap the example, and less data leads to a worse learner.

Note, I used the Weka library for the J48 tree. I used the Weka library Classifier interface, which provides the “black box generic interface” requested in the HW writeup. You could easily exchange the J48 tree for a different Weka tree or classifier. The Classifier interface accepts an Instances object to train on. I used the Weka Instances class to store the arff data, this class also has a method resample which uses random sampling with replacement. More info about Weka can be found here: <http://www.cs.waikato.ac.nz/ml/weka/>.