

CS 177, Homework 3

Applications of Probability in Computer Science: Winter 2007

Due Date: Tuesday January 30th, 12 noon

Reading

You should review your notes from class on conditional independence and naive Bayes classifiers. As additional background reading you can look at the links to articles about spam email and Bayes classifiers on the class Web page.

Summary of the Naive Bayes Programming Assignment

In this assignment you will implement the algorithm for learning a naive Bayes classifier and you will apply it to a data set from a set of real emails. You are required to use MATLAB for this assignment. The assignment has 3 major parts.

1. Write a function called `nbayes_learn.m` that takes in training data and returns the probabilities for a naive Bayes classifier as discussed in class. A template for this function is available on the class Web page.
2. Write a function called `nbayes_predict.m` that takes a set of test data vectors and returns a set of class label predictions for each vector. Again a template is on the class Web site.
3. Use both functions to conduct experiments as described later in this assignment.

Both the MATLAB functions above and a document called `experiments.txt` or `experiments.doc` (a 1-page summary of your experiments) should be uploaded to the Homework 3 folder on EEE. Nothing else needs to be handed in, i.e., no hardcopies are needed.

Spam Email Data Sets

For your assignment you will be working with a data set that was created a few years at Hewlett Packard Research Labs as a testbed data set to test different spam email classification algorithms. The HP researchers gathered a set of both spam and non-spam emails and computed a set of “feature values” (e.g., based on the presence of certain words) for each email. The idea is for an algorithm to try to predict whether an email is spam or not based on the features.

On the class Web page is a pointer to three files:

1. **Features File:** The first called **spam_features.txt** contains an ascii file of $n = 4601$ rows and $d = 57$ columns. Each column represents a single binary variable $X_j, 1 \leq j \leq d$, i.e., the set of features for your classifier. Each row represents a particular email, and the values of the 57 random variables in that row represent the values of different features that were calculated for that email.
2. **Labels File:** The second file, **spam_labels.txt**, consists of 4601 numbers, which are the class labels, with each number (label) corresponding to a row in the first file). The label 2 corresponds to emails that were labeled as “spam” and the label 1 to emails that were labeled as non-spam.
3. **Additional Documentation (Optional):** As background information, if you wish, you can read the documentation file **spambase.DOCUMENTATION** and the **spambase.names** file that describe the data in more detail: how the data was collected, what the features mean, basic statistics about the data, and so on. Note that the original features are real-valued (many of them are expressed as the percentage of words in an email that correspond to a particular word). So to create discrete-valued variables that we can use in a naive Bayes classifier, we converted each of the original real-valued features to a binary random variable by making all values less than or equal to the median (across all 4601 rows for that feature) be 1 and all values above be 2. There are other ways one could do this (e.g., one could convert to $d > 2$ values for example): if you want to look at the original raw data, take a look at: <ftp://ftp.ics.uci.edu/pub/machine-learning-databases/spambase>

Learning a Naive Bayes Classifier from Data

You will implement a MATLAB function `nbayes_learn.m` for learning a naive Bayes classifier. In summary, your function will:

- Separate the examples into their respective classes using the class labels provided.
- Let n_1 and n_2 be the number of examples that have class value 1 and 2 (respectively), and let n be the total number of examples in the training data (so $n = n_1 + n_2$). Note here that for the spam email data set there are just 2 values, $c_1 = 1$ and $c_2 = 2$.
- Calculate the marginal (unconditional) probability $p(c_i)$ for each class as follows:

$$p(c_i) = \frac{n_i}{n}, \quad i = 1, 2$$

- For each class value c_i , for each attribute X_j , and for each of the possible value k of the attribute X , calculate the “smoothed” conditional probability $p(X_j = k|C = c_i)$ as

$$p(X_j = k|C = c_i) = \frac{n_{jk,c_i} + 0.5}{n_i + 1}, \quad i = 1, 2, \quad j = 1, \dots, d, \quad k = 1, 2$$

as discussed in class, where n_{jk,c_i} is the number of examples (rows) where (a) feature (column) X_j takes value k and (b) the class label is c_i .

At this point the training is complete. You should store the various probabilities in the MATLAB data structure `params`, the format of which is described in the template functions. You might want to check that your calculations are correct by ensuring that the various conditional probabilities you have calculated sum to 1, e.g., $\sum_{k=1,2} p(X_j = k|C = c_i) = 1$ where the sum is over the values of attribute X_j for a particular class value c_i .

Using a Naive Bayes Classifier for Classification

Next you will use the classifier learned by the previous function to classify a set of test data vectors. You will write a function `nbayes_predict.m` that finds the most likely class label, i.e., the class c_i for which the probability $p(c_i|x_1, \dots, x_K)$ is highest. Here the x_1, \dots, x_K are known, and the class value c is treated as if it were unknown. The function will repeat this calculation for each row in a table of data vectors.

For each data vector x_1, \dots, x_k , and invoking the conditional independence assumption that the X_j 's are conditionally independent given the class variable C , we can calculate (for each class value):

$$p(c_i|x_1, \dots, x_d) = \frac{p(c_i) \prod_{j=1}^d p(x_j|c_i)}{p(x_1, \dots, x_d)}$$

by the conditional independence assumption. As discussed in class, to find the maximum of this expression (over c_i) we only need to calculate the numerators and not the denominator.

Note that if you calculate this product directly, you may get underflow (since we are multiplying many very small numbers together). We will discuss in class how to do this computation in log-space to get around the underflow problem.

Finally, to predict a class you select the class value c_i for which $p(c_i|x_1, \dots, x_d)$ is largest—this is the predicted class label from the classifier. Your function will take as input a set (a matrix) of examples (each example being a row in the matrix containing 57 values) and return a vector of predicted class labels \hat{c} , one predicted label per example.

Testing the Naive Bayes Classifier

To calculate the accuracy of a set of predicted labels (as returned by your prediction function) you need to compare each label to the true known label (which the prediction function did not get to use in its calculations), and count how many predicted labels match the true labels. Divide the number of correct matches by the total number of predictions made (to get an accuracy number expressed as a number between 0 and 1) and multiple by 100 to express this as a percentage.

Perform the following experiments and put the items requested below in your report:

1. Train the naive Bayes model on the first 1500 examples (the first 1500 rows in the file along with their class labels) and then calculate the accuracy of this model on a test data set, namely the remaining examples in the file (rows 1501 through 4601 in your data). To calculate the accuracy you compare the class label predicted by the model for each example (see previous section on how to generate this) with the true known labels (in this case it would be labels 1501 through 4601). Report the accuracy as a percentage, i.e., 100% if all class labels are predicted correctly.
2. Repeat the first part, now training on just the first 50 examples, and again testing on examples from 1501 through 4601. Again report the accuracy.
3. Repeat again, but now training on just the first 10 examples, and again testing on examples from 1501 through 4601. Again report the accuracy. Comment briefly on any differences in accuracy you see in the 3 experiments so far.
4. How accurate would the classifier be if it randomly guessed a class label (with probability 0.5 spam and probability 0.5 non-spam) for each example in a very large hypothetical test data set (e.g., with millions of examples)? How accurate would it be if it always picked the most common label in the training data (the most common label in examples 1 through 1500) and then used this as the class label prediction for a very large hypothetical test data set? Comment on how these two accuracy numbers (based on random and most common predictions) compare to the accuracies you obtained in parts 1 through 3.
5. For the first 20 examples in the data file, provide a table with 2 probabilities from the naive Bayes model, with for each example, the probability that the email is spam and the probability that it is non-spam, given the model. The probabilities should be generated by a naive Bayes model trained on the first 1500 examples. Note there is no template code provided for doing this calculation, nor are you asked to hand in the code you use to do the calculation. If you wish, you can copy your prediction function to create a new function and modify this new function so that it can generate these probabilities, since it will have the same basic loop structure as your prediction function, but it calculates probabilities of classes instead of finding the most likely class for each example.