Mehran Sahami
CS109
Handout #16
March 1, 2010

Problem Set #6

Due: 2:15pm on Wednesday, March 10th

Note: The last day this assignment will be accepted is Friday, March 12th

As noted above, the last day this assignment will be accepted (using a late day) is Friday, March 12th, as solutions will be made available then (in the handout bins in Gates). Since class does not meet on Friday, March 12th, if you plan on turning in the problem set on that day, you should make arrangements with your primary course contact to submit the problem set *directly to them*.

For each of the written problems, **explain/justify how you obtained your answer** in order to obtain full credit. In fact, most of the credit for each problem will be given for the derivation/model used as opposed to the final answer. Make sure to describe the distribution and parameter values you used, where appropriate. It is fine for your answers to include factorials, exponentials, or combinations, unless you are specifically asked for a computed numeric answer.

Written problems

1. Say we want to determine a confidence interval for μ, the true mean running time of a particular algorithm. We run the algorithm 300 times, gathering the following statistics for the sample mean and sample variance: $\bar{X} = 447$ seconds and $S^2 = 376.36$ seconds^2.

   a. Estimate the true mean μ with a 90% confidence interval.

   b. Estimate the true mean μ with a 95% confidence interval.

   c. Estimate the true mean μ with a 99% confidence interval.

2. Consider the Exponential distribution. It is your friend… really. Specifically, consider a sample of I.I.D. random variables $X_1, X_2, \ldots, X_n$, where each $X_i \sim \text{Exp}(\lambda)$.

   a. Derive the maximum likelihood estimate for the parameter λ in the Exponential distribution.

   b. Is the estimator you derived in part (a) biased? You only need to give a "yes" or "no" answer, and a short informal justification for your answer. A formal derivation/proof is not needed. (Hint: Johan Jensen might be interested in your answer).

   c. Is the estimator you derived in part (a) consistent? (Again, you only need to give a "yes" or "no" answer, and a short informal justification for your answer. A formal derivation/proof is not needed).

3. Say you have a set of binary input features/variables X_1, X_2, …, X_m that can be used to make a prediction about a discrete binary output variable Y (i.e., each of the X_i as well as Y can only take on the values 0 or 1). In using the input features/variables X_1, X_2, …, X_m to make a prediction about Y, recall that the Naïve Bayesian Classifier makes the simplifying assumption that $P(X_1, X_2, ..., X_n \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y)$ in order to make it tractable to compute $\arg\max_y P(\boldsymbol{X}, Y) = \arg\max_y P(X_1, X_2, ..., X_n \mid Y) P(Y)$.

    a. Show that the prediction for Y made by the Naïve Bayesian Classifier is actually based on selecting a value of Y that maximizing a ***linear*** function of the X_i. (Hint: show how to transform $P(\boldsymbol{X}, Y)$ into a linear function of the form: $C + \sum_{i=1}^{n} f(X_i)$, noting that each variable X_i appears in separate terms in the sum. It is fine for Y to appear in any terms in the sum (i.e., it is fine for *C* or the function *f* to depend on Y). Note: this question is not as complicated as it may first appear to be, so you shouldn't worry if you find the answer to be straightforward.)

    b. Now say that the first *k* input variables X_1, X_2, …, X_k are actually all *identical* copies of each other, so that when one has the value 0 or 1, they all do. Explain informally, but precisely, why this may be problematic for the model learned by the Naïve Bayesian Classifier.

Programming Problems

For the following problems, you will be implementing two learning algorithms: the Naïve Bayesian Classifier and Logistic Regression. You may implement your algorithms in your choice of C, C++, Java, or R. You can feel free (but are under no obligation) to use the CS106A ACM Java libraries, the CS106B/X C++ libraries, or the C++ Standard Template Libraries (STL). If you use R, however, you must develop your own code for parameter estimation from data in the model – you may **not** make use of any external libraries or high-level built-in functions (e.g. `glm`) that build the prediction models for you. For each algorithm, **you should turn in your source code as well as answers to the questions listed below**. It's fine if your implementation of both algorithms is in a single file or if you use multiple files. In either case, please provide a printout of any code you write.

For each algorithm you write, you will be testing it with three data sets. A description of the data sets you will be using, the file format for the data files, and instructions on how to obtain the data files are given below. Note: you do **not** need to do any error checking in your file reading code (you can assume the data is always correctly formatted). To simplify your implementation, you can assume that all input features are always binary variables (0 or 1), and the output class is also always a binary variable (0 or 1). For the assignment, our main interest is the results you obtain with the learning algorithms. As a result, you do not need to worry about the generality of your implementation – i.e., you can write your algorithms to only deal with binary input/output features. Your code should, however, be general enough to work for any number of input features or data instances (within reason), as the different datasets you will be dealing with contain different numbers of input features and data instances. We will grading your code only on functionality, not on programming style. With that said, it is still in your interest to write good modular code as there are many opportunities for code reuse in implementing this assignment.

Data Sets

You will be running your learning algorithms on three data sets (each of which has a respective training data file and testing data file). The data sets are described in more detail below.

Simple (`simple-train.txt`, `simple-test.txt`)

This is a simple dataset provided primarily to help you determine that your code is working correctly. There are two input features, and the output class value is determined by the value of the first feature (i.e., $y = x_1$). The training data set and testing data set are identical, each containing four data vectors. Both your Naïve Bayesian Classifier and Logistic Regression implementations should be able to classify all instances in the simple testing data set with 100% accuracy after training on the simple training set.

Congressional voting records (`vote-train.txt`, `vote-test.txt`)

This dataset contains the congressional voting records from the U.S. House of Representatives in 1984 on several key issues. Each input vector represents the voting record for one member of Congress. There are 48 binary input features. The output class value represents the political affiliation of the Congressperson (Democrat or Republican, encoded in binary). The training data set contains 300 data vectors, and the testing data set contains 135 data vectors.

(Thanks to Jeff Schlimmer for providing this data to the UC Irvine Machine Learning Data Repository.)

Heart tomography diagnosis (`heart-train.txt`, `heart-test.txt`)

This dataset contains data related to diagnosing heart abnormalities based on tomography (X-ray) information. Each input vector represents data extracted from the X-ray of one patient's heart. There are 22 binary input features. The output class value represents the diagnosis of the patient's heart (normal or abnormal, encoded in binary). The training data set contains 80 data vectors, and the testing data set contains 187 data vectors.

(Thanks to Lukasz Kurgan and Krzysztof Cios for providing this data to the UC Irvine Machine Learning Data Repository.)

Data file format

All the data files described above adhere to the following file format:

```
<number of input variables per vector>
<number of data vectors in file>
<first data vector>
<second data vector>
...
<n-th data vector>
```

Note that each data vector in the file is comprised of a number of input variable values that are binary (0 or 1). The input variable values are separated by a single space. The last input variable value is immediately followed by a colon character '`:`', then a single space and then the value of the binary output variable for the vector.

For example, here is the annotated **`simple-train.txt`** data file (with annotations in italic font on the right-hand side):

File: **`simple-train.txt`**	Explanation of lines in data file
`2`	← ***There are 2 input variables per vector in the file***
`4`	← ***There are 4 data vectors in the file***
`0 0: 0`	← ***First data vector (has class 0)***
`0 1: 0`	← ***Second data vector (has class 0)***
`1 0: 1`	← ***Third data vector (has class 1)***
`1 1: 1`	← ***Fourth data vector (has class 1)***

How to get data sets
The data files are available on the CS109 website home page: **`http://cs109.stanford.edu`**.

Note that there are different bundles for the PC and Mac, as those platforms treat line endings in text files differently. If you are using UNIX/Linux, most flavors of UNIX/Linux use the same line ending format as the Mac.

Actual programming problems

Training and Testing your algorithms

The "training" data files should be used to train your learning algorithm (i.e., determine the model parameters). The "testing" data file should be used to determine the accuracy of your model *after* the training phase is complete. In other words, when we describe *training* an algorithm below, you should take that to mean that you are working *only* with the "**`-train`**" file for a particular dataset to determine the parameters of your model. When we then describe *testing* a model you should take that to mean that you are using only the "**`-test`**" file for a particular dataset to determine how well your model does at classifying the data.

Measuring model accuracy

After a model is trained, we determine its accuracy by testing it on a new set of data (generally not the same data we used to train the model). We measure the model's accuracy by determining how many of the testing vectors were correctly classified – that is, the number of times the output class value predicted by the model was the same as the actual output class value provided in the data. We report accuracy by indicating the number testing data vectors that were tested of each class, and the number that were correctly classified. For example, say we have a testing data set consisting of 12 vectors total, where the first 5 vectors are of class $y = 0$ and the remaining 7 of class $y = 1$. When we then make predictions for each data vector using our model, say we correctly predict class $\hat{y} = 0$ for 4 out of the first 5 vectors and then correctly predict class $\hat{y} = 1$ for 5 out of the next 7 vectors. Our overall accuracy for the model would be 0.75 since we correctly classified a total of 9 out of 12 vectors. We would report these results as follows:

```
Class 0: tested 5, correctly classified 4
Class 1: tested 7, correctly classified 5
Overall: tested 12, correctly classified 9
Accuracy = 0.75
```

You should use this same accuracy reporting scheme for the algorithms you implement below.

1. Implement the Naïve Bayesian Classifier for binary input/output data. Specifically, your classifier should make predictions for the output variable using the rule: $\hat{Y} = \arg\max_y P(\boldsymbol{X} \mid Y)P(Y)$, by employing the Naïve Bayes assumption, which states that:

$$P(\boldsymbol{X} \mid Y) = P(X_1, X_2, \ldots X_m \mid Y) = \prod_{i=1}^{m} P(X_i \mid Y).$$

Thus, your program will need to estimate the values P(Y) as well as $P(X_i \mid Y)$ for all $1 \leq i \leq m$ from the training data. Note that to estimate the probability mass function $P(X_i \mid Y)$, you will need to estimate both $P(X_i \mid Y = 0)$ and $P(X_i \mid Y = 1)$. For each of parts (a)-(c) below, you should run your algorithm **twice**, the first time computing your probability estimates using Maximum Likelihood Estimators and the second time computing your probability estimates using Laplace Estimators.

   a. Train your algorithm on the data file **`simple-train.txt`**. Test your algorithm on the data file **`simple-test.txt`** and report your classification accuracy. Remember to do this once with Maximum Likelihood Estimators and once with Laplace Estimators. As a sanity check, you should be able to achieve 100% classification accuracy on the testing data using a model trained with Maximum Likelihood Estimators.

   b. Train your algorithm on the data file **`vote-train.txt`**. Test your algorithm on the data file **`vote-test.txt`** and report your classification accuracy. Remember to do this once with Maximum Likelihood Estimators and once with Laplace Estimators. To give you a sanity check of how well you should be doing, you should be able to achieve at least 90% classification accuracy on the testing data using a model trained with Maximum Likelihood Estimators.

   c. Train your algorithm on the data file **`heart-train.txt`**. Test your algorithm on the data file **`heart-test.txt`** and report your classification accuracy.

   d. Make sure to turn in a print out of your code for the Naïve Bayes Classifier.

2. Implement Logistic Regression for binary input/output data. Specifically, you should implement the "batch" gradient algorithm described in class. In each case below, you should train your classifier for 10,000 epochs (passes through the training data).

   Note that you will need to use an exponential function (e^x) to implement Logistic Regression. In C/C++, you will find the function **`double exp(double x)`** from the library **`<math.h>`** helpful. Similarly, for those working in Java, the static method **`double exp(double x)`** from the class **`java.lang.Math`** also computes e^x.

Continued on Back

a. Train your algorithm on the data file `simple-train.txt`. Use learning rate η = 0.0001. Test your algorithm on the data file `simple-test.txt` and report your classification accuracy. You should be able to achieve 100% classification accuracy on the testing data.

b. Train your algorithm on the data file `vote-train.txt`. Use learning rate η = 0.0001. Test your algorithm on the data file `vote-test.txt` and report your classification accuracy. **Were you able to obtain a higher classification accuracy than with the two (parameter estimation) variants of the Naïve Bayes classifier on this data? Briefly explain why or why not this was the case.**

c. Train your algorithm on the data file `heart-train.txt`. Use learning rate η = 0.0001. Test your algorithm on the data file `heart-test.txt` and report your classification accuracy. **Were you able to obtain a higher classification accuracy than with the two (parameter estimation) variants of the Naïve Bayes classifier on this data? Briefly explain why or why not this was the case.**

d. Train your algorithm on the data file `heart-train.txt`. Experiment using different learning rates η, where you are still testing your algorithm on the data file `heart-test.txt`. **Report the highest classification accuracy you could obtain on the testing data, and what was the value of the learning rate η you used to obtain it. Why do you think the learning rate had such an effect on your classification accuracy?**

e. Make sure to turn in a print out of your code for Logistic Regression.