

CS 374 Assignment #3

Estimating Probabilities

Regression

Due the week of February 18, 2008

This week you will follow up on some ideas from last week. In particular, you will explore the theory behind last week's techniques for estimating prior probabilities and likelihoods from a data set. After doing this, you will move on to the topic of regression, where the result to be predicted for each example is a real value rather than a nominal class value. This week one of your exercises will involve programming, and we will take advantage of this opportunity to practice the "standard" tutorial mode of presentation and critique. One member of each group will present his code; the others will be responsible for providing critique.

1 Estimating Probabilities

Last week you implemented Naive Bayes learning, which requires that you compute priors and likelihoods from a training set of data. Witten and Frank do a nice job of describing how to calculate priors and likelihoods, and the methods they outline certainly make a great deal of intuitive sense. What wasn't clear in your readings last week, however, is that there is sound theory to back up these techniques for estimating probabilities. This week you will explore that theory. (Note that Maximum Likelihood Estimation will also be useful in your readings about regression and many other topics this semester. This isn't simply follow-up to your work last week.)

1.1 Reading

In many areas of machine learning, we make assumptions that our data are drawn from a distribution that obeys a particular probability model. The techniques that make this assumption are called *parametric*. Once we assume a model, our task is to estimate the parameters of the model, based on the data that we have.

Please read Sections 4.1, 4.2, and 4.5 of Alpaydin, which trace through the theory behind the estimators for a number of different probability models.

1.2 Exercises

1. (Just do this one for yourself. There's no need to turn it in.) On page 63, Alpaydin sketches the derivation of the rule for estimating the parameter, p , of a Bernoulli distribution. Fill in the details of the derivation. In particular, show dL/dp and solve for 0 to get Equation 4.4.
2. (Please turn in this one.) Now fill in the details for the derivation of Equation 4.8, the maximum likelihood estimators for a Gaussian distribution.

2 Regression

Up to this point, we have devoted our attention to the general task of classification. Given a labeled set of training examples (labeled with correct class names, that is), our goal has been to induce a classifier to predict the labels of previously unseen examples. Now we turn our attention to a related,

but different, task – regression. Here, given an example, our goal will be to predict a real-valued output, rather than a discrete class.

2.1 Reading

Read Sections 4.6 and 5.8 in Alpaydin, which introduce regression.

2.2 Exercise

In the simplest case, as discussed in Section 4.6, we assume that the real-valued output is a function of a single input attribute plus some Gaussian noise. In linear regression, we assume that the function to be learned is linear. We can also assume more generally that the function to be learned is a polynomial of order k . While Alpaydin introduces the more general version, we will concentrate on the linear case.

This exercise will take your group’s designated programmer through the process of implementing linear regression, so that you can gain a better understanding of how it works and what sorts of results it can produce.

Your program should begin by generating a training set.¹ If you’re doing your implementation in Java, you might find it useful to use the `nextGaussian` method in the `Random` class. (The Java API includes the code for the method.) I have also provided for you a method (implemented in Java) to generate Gaussian noise. The method can be found in

```
~andrea/shared/cs374/noise.java
```

The main differences between my implementation and Java’s are that mine allows you to specify a non-zero mean and the variance of your choice. It provides two random numbers rather than just one. Though both of the methods are implemented in Java, you should have no trouble converting either into the language of your choice.

There are several different ways to generate Gaussian noise. The method I have implemented for you uses a technique called the Box-Muller Method. It starts with two uniformly distributed random numbers and returns a pair of numbers from a normal distribution with a given a mean and variance. (The mean and variance are passed in as parameters.)

When you set up your data generator, be sure that you can generate data sets of different sizes. Then, since we are assuming a linear model, you should specify values for w_1 and w_0 . Next you need to generate values for the input attribute x , and, for each x , calculate the corresponding output value r by plugging x into your linear function and then adding some Gaussian noise.

You can use the noise generator to help you generate x values. (Perhaps a better name for the method would be “gaussian” rather than “noise”.) Just specify a mean and variance of your choice. The added noise should be 0-mean Gaussian noise. Again, you select the variance. (I like to use 1.0.)

Now you are ready to implement linear regression as described on pages 73-75 of Alpaydin. Print the resulting w_1 and w_0 values. How do they compare to those you defined earlier? How does the linear function that you output perform on previously unseen data? To determine this, generate a test set. (This time don’t add Gaussian noise, so that you can see how your results compare to the ideal.) Compute the sum of squared errors. How well did the algorithm work?

¹It might seem silly to generate your own data, but it is often very useful. There are times when we want a data set that will serve as a “control” in an experimental evaluation of a technique. Therefore, it is important to learn how to generate data sets that satisfy given sets of constraints.

3 Program and Critique

Only your group's designated programmer needs to implement linear regression. The other group members will be responsible for providing critique of the implementation (to be given in the tutorial session). To allow the critic some thinking time, the programmer should give the code to the critic *before* the tutorial session. The timing is up to each group, but I suggest 24 hours before the tutorial meeting. Each critic should write up their notes before the meeting.

While the critics this week don't need to turn in an implementation of regression, they might find it useful to actually do the implementation. This will give each critic some starting ideas for how the design and implementation might be done.

4 Using Naive Bayes for Regression

Linear regression as described in Alpaydin is not the only technique we have for inducing a regression model (i.e., a model that predicts real values, rather than discrete classes). In fact, we can employ Naive Bayes for this purpose.

4.1 Reading

Please read the paper "Naive Bayes for Regression", by Frank et al. (A link can be found on the Assignments page of the course website.) This paper appeared in the journal *Machine Learning* in 2000. You need not follow all of the details in Section 2, but you should be able to summarize the approach described there at a high level.

4.2 Exercise

Please write a reaction to the algorithm and results described in the paper, and be prepared to discuss your reaction during the tutorial meeting. Does the authors' conclusion match your intuition about the performance of Naive Bayes for regression? Do the authors make a convincing case for their conclusion? Is the regression algorithm sound? Does the evaluation methodology make sense? Are the data sets used in the tests selected well? Your written reaction need not be long – approximately 1-2 pages should be enough to get an interesting discussion started.