Statistics for Business Analytics 2: R Laboratory Homework 3

Instructions

This laboratory assignment is designed to look at the Naïve Bayes classifier in a bit more detail. Please submit an R Markdown file for this assignment.

In order to fully grasp how this model works, it is really important to understand how the basic probabilities are calculated and used in the Naïve Bayes classifier. To do this, we are going to build a simple artificial data set and perform some calculations ourselves.

Questions

1. Execute the following series of commands in your R window (0 points):

```
d1 <- c(1, 0, 0, 1, 0, 0, 0, 1)
d2 <- c(0, 1, 0, 1, 1, 0, 0, 0)
d3 <- c(0, 0, 1, 0, 0, 0, 1, 0)
d4 <- c(0, 0, 0, 1, 0, 0, 0, 0)
d5 <- c(0, 0, 0, 0, 0, 0, 1, 0)
d6 <- c(0, 0, 0, 1, 0, 1, 0, 1)
d7 <- c(0, 0, 1, 0, 0, 1, 0, 1)
d8 <- c(1, 0, 0, 0, 0, 0, 0, 1)
d9 <- c(0, 0, 0, 0, 0, 1, 0, 1)
d10 <- c(1, 1, 0, 0, 0, 1, 0, 1)

nb_df <- as.data.frame(rbind(d1,d2,d3,d4,d5,d6,d7,d8,d9,d10))
names(nb_df) <- c("BadCredit", "HasStableJob", "OwnsHouse", "BigLoan",
"HasLargeBankAccount", "HasPriorLoans", "HasDependents", "Decision")
```

We have created a very simple data set consisting of ten observations of seven input features. This is a simplified rendition of the German Credit data set from the UCI repository that we saw in our slides for the Support Vector Machine lab. The output variable is the Decision column. Here, this takes the value 1 when we reject the loan and 0 when we accept. We are going to construct a very simple Naïve Bayes model for this problem but we are going to train this manually.

Before beginning this section, take a moment to refresh your memory on the very simple equation that describes the Naïve Bayes classifier and how this arose from a straightforward application of Bayes rule and the conditional

independence assumptions of the model. The final expression that actually describes how we pick which class to classify is the following:

$$Classify\ C_i : \underset{C}{\operatorname{argmax}}\, P(C) \cdot \prod_{i=1}^{n} P(F_i|C)$$

Make absolutely sure you understand this expression 100% before proceeding. It really is not that complicated. The argmax over C in this equation basically says that we are picking the value of C (i.e. one of the possible output classes) so that the class we pick maximizes the rest of the expression that follows. This expression is a product. The first term is P(C). This is the probability of the class in question. We estimate this as the relative proportion of that class in our training data. The second term of the product is itself a large product. Specifically, it is a product of all the probabilities of the features taking their particular values for the observation in question, given the class C that we are testing. So, in a nutshell, for every observation that we want to make a prediction we are going to compute the above expression for all the possible classes in our problem and then pick the class that gives the largest value of this expression. Simple enough? I hope that you understand this from class and this description.

Training the model is actually going to be a piece of cake, because all we actually have to do is to pre-compute all the probabilities that we are going to need. Now, in our case we have just two output classes so we need to compute two sets of probabilities. In addition our features are binary so each feature produces two possible probabilities for each class, the probability of taking the value 0 and the probability of taking the value 1, under that class. If we have two classes and a binary feature F_1 then that feature will result in a total of 4 stored probabilities. Note that the features can take many values but they cannot be continuous otherwise we could note compute the discrete probability of a feature taking a particular numerical value such as 8.455354. This probability would be zero and the expression above would not work.

Ok, hopefully you have at least a basic idea of how this model works and so we are going to try to put all this together by way of an example.

2. Using the data frame you just constructed, create a vector of class probabilities for the two classes, 0 and 1. That is, compute P(C) for each of these two classes and store this in a vector called priors. Hint: You just need the Decision Column for this, don't you? (2 points)

All right let's try something a little bit harder now. We want to compute the probability that each of our features takes the value 1 for each class. For example, let's take the *BadCredit* feature. We want to determine the probability that this will take the value 1 when we reject a loan (output class = 1) and the same for when we accept the loan (output class = 0).

3. Compute a summary data frame in which one row contains the probabilities $P(F_i = 1|Class = 0)$ for all the different features F_i and the other row contains the probabilities $P(F_i = 1|Class = 1)$. For example, the cell at [1,1] could contain the probability that when we accept the loan (class = 0) the loan applicant has bad credit (BadCredit = 1). To compute the probabilities we just need to group our data by class and find the relative proportion of 1's for each feature. This is maximum likelihood estimation. Hint: Try using the aggregate() function, see if that helps. (4 points)

4. Examine this matrix of two rows you just created. One row has the probability of a feature taking the value 1 for class 0 and the other row has the probability of a feature taking the value 1 for class 1. Why do these probabilities not add up to 1 i.e. if we add the two rows together, why do we not get a row of 1's? (2 points)

Now, we said earlier that we need the probabilities of every feature taking every possible value given each class. In the previous step we computed $P(F_i = 1|C)$ for both classes. Now we need to do the same thing for $P(F_i = 0|C)$

5. What trivial expression can be used to convert our matrix of values with $P(F_i = 1|C)$ to one in which the values are $P(F_i = 0|C)$? (2 points)

It should be clear that we now have all the probabilities we need in order to make predictions for a new observation. In effect, we have actually trained a Naïve Bayes classifier because we calculated all the probabilities we need using maximum likelihood estimation. Before we actually go ahead and try to make a prediction however, let's take a look at some of the probabilities we produced.

6. According to our model, what is the probability that the bank will decide to give out a loan (Decision = 0) when the customer has a bad credit score (BadCredit = 0)? Why is this going to be a problem during prediction? (2 points)

A very common occurrence of this same problem is with text data and the bag of words model that we saw in our lab. Some times a particular output variable and feature value combination is not present in our training data, even though this feature value might be frequently encountered for other values of the output variable. For example, suppose we have a bag of words feature for the word "bad". There might be many negative movies reviews with this word and many negative reviews without it. We may however, not have any positive reviews with this word. As a result we will end up with the same problem that we saw in our synthetic data set.

A common approach to this problem is known as **additive smoothing**. The key idea is to always add a fixed constant to the counts of every feature value and output class combination. This value is often 1, so it is also known as **add-one smoothing**. Suppose we have a feature F_1 and class C_1. To estimate $P(F_1=1|C_1)$

we simply would count up all the observations in our training data from class C_1 and take the proportion of these that have $F_1=1$:

$$P(F_1=1|C_1) = N(F_1=1 \ \& \ C=C_1) \ / \ N(C = C_1)$$

where N() means "number of observations". With additive smoothing we will add 1 to every single count of the feature value and class combinations. To account for this in terms of ensuring we get valid probabilities, we need to also adjust our denominators with as many 1's as there are different feature values:

$$P(F_1=1|C_1) = N(F_1=1 \ \& \ C=C_1) + 1 \ / \ N(C = C_1) + d$$

where d is the number of different values that feature F_1 takes. In our simple example we only have binary features so this will always be 2.

6. Recalculate your matrix of probabilities that you computed in step 4 to incorporate additive smoothing. Hint: One easy way to do this is to add extra rows to the original matrix and use the same approach you used before. You don't have to do it this way though; there are many ways to implement this function. When you are done, make sure that your computed probabilities are what you expected to find. (4 points)

Now we are ready to use the probability data structures that we have created in order to implement a simple function for making predictions on new data using our argmax expression. Time to implement this.

7. Implement a simple function that makes predictions for your Naïve Bayes model. The function will take as input a data frame with 7 columns (one for each feature), a probability vector with the class probabilities we computed in step 2, and the conditional feature probabilities that you computed in step 6. Here is the function signature:

```
predict_nb <- function(test_df, priors, prob_matrix) {

  # Your code here
}
```

The output of this function is going to be a vector with the predicted value for each test observation in the input data frame, test_df. Thus, its length will be that of the number of rows (observations) in that data frame. HINT: Try writing a function for computing the right answer for a single observation and then use this function with an apply formula to extend it to work on an entire data frame. (4 points)

We can now use this function to compute estimates of our training accuracy using our simple data frame that we created earlier.

8. Compute the training accuracy of your Naïve Bayes model using the function that you just created. (2 points)

We don't have a separate test data set but if we did, now would be the time to use the predict function you wrote to estimate accuracy on the test set.

Well done! You've built a very simple implementation of a Naïve Bayes model, and added in smoothing to combat the issue of data sparsity. Hopefully, through this process you will have understood how the model works in a bit more detail.