FIT2086 Studio 7
Naïve Bayes and Logistic Regression

Daniel F. Schmidt

August 30, 2018

Contents

1 Introduction

Studio 7 introduces you to the use of Naïve Bayes methods and logistic regression for classification. During your Studio session, your demonstrator will go through the answers with you, both on the board and on the projector as appropriate. Any questions you do not complete during the session should be completed out of class before the next Studio. Complete solutions will be released on the Friday after your Studio.

2 Naïve Bayes and Logistic regression

In this second question we will use R to learn Naïve Bayes and logistic regression models for a problem with a binary target, and 100 binary predictors. To do this question you will need to install two packages, as R does not have good built-in tools for Naïve Bayes classification. One of the advantages of R is the very large number of free packages, for an enormous number of statistical models and algorithms, that are available on the online repository. First, install the `naivebayes` package from the R online repository using the code

```
install.packages("naivebayes")
```

Once a package is installed, it needs to be loaded into memory before we can use it. To do this, use the code

```
library(naivebayes)
```

R will report that it has successfully loaded the package. You can now use the functionality of the package. As a general rule you can get information on how to run it using a command like `??naivebayes`. We also need to install the `pROC` package to get functionality for producing area-under-the-curve statistics. Install the `pROC` package and load it into memory.

1. Load the `gene.train.csv` file into memory. This has $n = 200$ observations on $p = 100$ binary predictors, each marking the presence or absence of a genetic mutation at a certain point on a persons genome (called SNPs, pronounced "snips"). A "mutation" is defined as having a different genetic marker to the general population. Notice that all the predictors, and the target `Disease`, are factor variables. This means they are categorical variables. You can find the valid levels of a factor using

   ```
   levels(gene.train$Disease)
   ```

   This will return "`N`" and "`Y`" as the two levels. The second level is treated as the "success" by R's logistic regression functions.

2. Try and fit a logistic regression model to all the data using the `glm()` function. This works similar to the `lm()` function for linear models.

   ```
   fullmod=glm(Disease ~ ., data=gene.train, family=binomial)
   ```

   Note that the main difference is we specify that the `family` argument is `binomial`, to let R know that we want a logistic regression on a binary target. This uses all 100 predictors. Use `summary()` on `fullmod` to get a summary of the fitted model. Note that only a few of the 100 predictors appear to have p-values suggestive of association. Take note of the following statistics:

   - The "null deviance", which is twice the negative log-likelihood of the model using just the intercept, as a reference; this is available as `fullmod$null.deviance`

- The "residual deviance" which is twice the negative log-likelihood of the model we have fitted; the difference from the null gives an indication of how much better we fit the data over the null model; this is available as `fullmod$deviance`
- The AIC score reported at the bottom which gives us the goodness-of-fit penalized by the model complexity. In R this is two times the minimised negative-log-likelihood of our fitted model, plus two times the number of fitted parameters. This is available as `fullmod$aic`. Calculate this AIC from the residual deviance and the fact that we have fitted 101 parameters – 100 predictors plus the intercept. It should match the reported score.

3. Let us see how well our model predicts onto future data. Load the file `gene.test.csv`, which contains $10,000$ new observations of people, with disease status and the same 100 genetic mutations. By default, if our model is a logistic regression, the `predict()` function produces the log-odds of being in the success class ("`Y`", in our case).

```
pred = predict(fullmod, gene.test)
```

   We can also calculate the conditional probabilites predicted by our logistic model. Do this using:

```
prob = predict(fullmod, gene.test, type = "response")
```

   If we want our predictions to be best guesses at the class, we have to do a little extra work:

```
pred = factor(predict(fullmod, gene.test,type="response")>1/2,c(F,T),c("N","Y"))
```

   will turn the predictions into a categorical variable with level "`N`" assigned if the predicted probability is less than $1/2$, and "`Y`" if it is greater than half (see help on `factor()`). We can then calculate the confusion matrix using:

```
table(pred, genes.test$Disease)
```

   What does this confusion matrix suggest about the ability of this logistic model to predict disease status correctly?

4. Calculate the classification accuracy (the proportion of correctly classified samples in the test data) using

```
mean(pred == gene.test$Disease)
```

   We can also calculate the area-under-the curve, and plot the receiver-operating curve (ROC) using the `pROC` package. To do this once we have the conditional probabilities of success for each individual, we can then run

```
roc.fullmod = roc(response=(as.numeric(gene.test$Disease)-1), prob)
```

   Note that we need to convert the target to a numeric 0 and 1 for `roc()` to work. This is a weakness of the R philosophy of extending functionality by user-written packages, as many packages have input formats. The AUC is stored in `roc.fullmod$auc`; inspect this value. Finally, we can produce the ROC by using

```
plot(roc.fullmod)
```

   Do you think this model is a good predictor of `Disease`?

5. Look at the file `my.prediction.stats.R`. This contains a function written to automatically compute all of these prediction statistics, as well as a few extra (sensitivity, specificity, logarithmic-loss). Load this function. To call it, use

```
my.pred.stats(prob,gene.test$Disease)
```

   The first parameter is the vector of predicted probabilities from our model and the second is the target variable to test against.

6. We have fitted $p = 100$ predictors to $n = 200$. It is very likely we are overfitting. We can use the `step()` function to perform stepwise selection and prune out variables. `step` has two basic modes: forward selection, and backwards selection. Backwards selection starts from the full model with all predictors included and sequentially removes unimportant variables; foward selection starts from an empty model and sequentially adds important variables. R also lets you do both; to do this, we just need to use the code:

```
step.fit.aic = step(fullmod,direction="both")
```

   The `direction="both"` tells R to use both backwards and forward selection to try and find a good model (i.e., it will add and remove variables until it finds a good one, where good is determined by the information criterion you are using). This may take a little while, as it needs to refit many models with a large number of parameters – this is a potential drawback of backwards selection.

   Calculate the prediction stats for this simplified models. Are they better than the full model?

7. When we have a large number of possible predictors we could include in our model it is sometimes better to be conservative when doing model selection. We can choose to use the Bayesian information criterion when doing our stepwise selection, which penalises extra predictors in the model by a greater amount. To do this run

```
step.fit.bic = step(fullmod, k = log(nrow(gene.train)), direction="both", trace=0)
```

   Note that the `k` argument is set to the logarithm of the number of data points when we use BIC. The `trace=0` is not required for BIC – rather it tells R to be silent while finding the best model, which can be useful if you do not want to clutter up your console with pages of information. Compute the prediction statistics using `my.pred.stats()` for this model. How do they compare to the full model and the one selected using AIC? Specifically, how do the confusion matrices and AUC curves differ between the AIC and BIC models?

8. Run `summary()` on the model selected by BIC. Write down the logistic regression equation for the log-odds. What happens to the log-odds of having the disease a mutation at `SNP12` is present? What happens to the log-odds of having the disease if a mutation at `SNP56` is present?

9. We can build a Naïve Bayes model to predict Disease using the genetic markers. To do this, run

```
nb = naive_bayes(Disease ~ ., data=gene.train)
```

   To produce predictions of the most likely class using this model we can use

```
pred = predict(nb,gene.test)
```

   To produce probabilities of being in the success class, we can use

```
prob = predict(nb,gene.test,type="prob")[,2]
```

By default the Naïve Bayes package returns a column of probabilities for each class, as it can work with multi-class targets, unlike logistic regression. For our example we are interested only in the probability of success. We can then get prediction stats using

```
my.pred.stats(prob, gene.test$Disease)
```

How does the Naïve Bayes model compare to logistic regression models we have tried?

10. Finally, we can try and improve our Naïve Bayes model by fitting it to less variables. In general, most Naïve Bayes packages do not come with built in predictor selection methods like the logistic regression package. However, we can combine information from our logistic fit and restrict our Naïve Bayes model fit to only the two predictors, `SNP12` and `SNP56` that BIC selected:

    ```
    nb = naive_bayes(Disease ~ SNP12 + SNP56, data=gene.train)
    ```

    The "`Disease ~ SNP12 + SNP56`" says to model `Disease` using both `SNP12` and `SNP56`, and nothing else. Compute the prediction statistics for this model. How do they compare to the logistic regression model with BIC?

    In fact, if the predictors are binary and the target is binary, it turns out that Naïve Bayes models and logistic regression are very mathematically similar. This is not the case when the predictors are numerical, as we will see in our next question.

3 Pima Indians Data with Logistic Regression

The last question will examine the use of logistic regression models to analyse the Pima indians data. This exercise will demonstrates three things: (i) the similarities between logistic and linear models when analysing data with non-categorical predictors, (ii) how to tell R to include transformations of your variables, including interactions. Begin by loading the `pima.train.csv` dataset into R; this file contains the following predictors variables:

- Number of Pregnancies (`PREG`)
- Plasma Glucose Concentration (`PLAS`)
- Diastolic Blood Pressure (`BP`)
- Triceps Skin Fold Thickness (`SKIN`)
- 2-hour Serum Insulin Levels (`INS`)
- Body Mass Index (`BMI`)
- Diabetes Pedigree Function (`PED`)
- Age (`AGE`)

as well as the outcome variable Diabetes (`DIABETES`), which is a categorical variable with categories "`Y`" and "`N`".

1. Fit a logistic regression model to the Pima indians data. Which variables appear to be important based on their p-values? Which variables do not appear to be important? Record the residual deviance (goodness-of-fit) of this full model.

2. Load the `pima.test.csv` data. Calculate prediction performance for the logistic regression model we fitted on the training data using this new data, i.e.,

   ```
   my.pred.stats(predict(fullmod,pima.test,type="response"), pima.test$DIABETES)
   ```

   where `fullmod` is the name of the model you fitted in the above question. How well does this model perform?

3. Perform stepwise selection on the full model, using the BIC:

   ```
   back.fit = step(fullmod, trace=0, k = log(nrow(pima.train)), direction="both")
   ```

   The `direction="both"` option says to try a combination of forwards and backwards selection, which is often better than using either one method. The `trace=0` option tells R not to be silent while doing this fit. Look at the fitted model using `summary()`. Calculate prediction statistics for this model. How does it compare to the full model?

4. R allows us to specify transformations of our variables directly when using `lm()` and `glm()`. We do this by adding terms to the "formula", which is the first parameter. Let us try adding in `log(BMI)` as a possible extra variable:

   ```
   fullmod = glm(DIABETES ~ .  + log(BMI) , data=pima.train, family=binomial)
   ```

   Notice how we include `log(BMI)`. Look at this model. Does `log(BMI)` appear to be associated with diabetes. Decide this by looking at (i) the p-value for `log(BMI)` and (ii) the residual deviance of this new model, compared to the residual deviance of the model without `log(BMI)` (calculated above).

5. We can also include polynomial terms, such as squares, directly. Let us try adding in `PLAS^ 2`:

   ```
   fullmod = glm(DIABETES  ~  .  + I(PLAS^ 2) , data=pima.train, family=binomial)
   ```

   Notice that we have to wrap the squaring in an `I( )`. Do you think `PLAS^ 2` is associated with `DIABETES`?

6. Another type of nonlinear effect is called an interaction. Essentially, we make a new predictor that is the product of two other predictors. The idea is this: in a model without interactions, the change to the log-odds when we increase a predictor is the same irrespective of the values of the other predictors. If we include an interaction of $A \times B$ between predictor A and B, it means that when we increase A in our model, the amount of change on the log-odds will now depend on the value of B, and *vice versa*. That is why we say the variables are "interacting". *(note: interactions can and often also are used in linear models as well as logistic models).*

   From our full model, we saw that `AGE` appears marginally associated with `DIABETES` as it had small p-value, but `SKIN` did not. Sometimes variables that are unimportant by themselves can become important if include them as part of interactions. We can try an interaction between the two using:

   ```
   fullmod = glm(DIABETES  ~  .  + SKIN*AGE , data=pima.train, family=binomial)
   ```

   The "`*`" tells R to build an interaction between the two variables, i.e., make a new predictor which is the product of these two predictors. Looking at the model with the interaction, do you think it is associated with `DIABETES`?

7. As you can see, building a good model is as much an art as a science. It involves exploring transformations or inclusions of variables to see if goodness-of-fit statistics like R^2 or residual deviance change dramatically, etc. We can try a brute-force approach and just throw lots of transformations into a model and hope a method like `step()` can do a good job of sorting out what is important and what isn't. However, it is important to bear in mind that the more predictors you try the greater the chance an unimportant predictor will, by chance, appear important and be included, which can damage our prediction on future data.

   The attached file `pima.example.R` shows how you can do something like this. Open the file and run the code. You can see that it fits a full model that includes all interactions between predictors (using the shorthand "`.*.`"), as well as new predictors that are the logarithms of the original predictors, and all squares of all the predictors – a total of 53 predictors. It then uses BIC to prune out and arrive at a final model. Write down regression equation for this final model. If you were a doctor advising a patient on ways to reduce their risk of developing diabetes, what does this model suggest that they could do?

8. The file also shows that if we set `k = 3` we have another information criteria, called the Kullback information criterion (KIC). This often works better than the AIC as it overfits less, but is less conservative than the BIC. We see from the file that the BIC finds a much more compact model, but the KIC finds a more complex model that includes some interacations, and predicts better. Which one we would prefer depends on the final use of the model – sometimes simpler models are easier for other people, such as doctors, to understand, and can be preferred. In other situations, predictive performance is all that matters.

   Table 1 (overleaf) summarises how to use several standard information criteria in conjunction with `step()`. They are listed in order of how conservative (likely to reject predictors) they are.

Criterion	Command
Akaike information criterion (AIC)	`step(fullmod, k = 2, direction="both")`
Kullback information criterion (KIC)	`step(fullmod, k = 3, direction="both")`
Bayesian information criterion (BIC)	`step(fullmod, k = log(nrow(df)), direction="both")`

Table 1: R commands to use stepwise selection with several different information criteria. In the commands `df` is assumed to be your data frame, and `fullmod` the full model you are pruning that has previously been fitted using `glm()` or `lm()`. Remember you can use the `trace = 0` option to disable output on the console from these functions.

9. Finally, we can use Naïve Bayes to build a classifier, even though the predictors are not categories. The `naive_bayes()` function uses some special techniques (beyond the scope of this subject) to estimate the probability distributions of the numerically predictors. To fit a Naïve Bayes model:

   ```
   nb = naive_bayes(DIABETES ~ ., data=pima.train, usekernel=T)
   ```

   The `usekernel=T` option tells the code to use "smoothing kernels", and should be used when you have numerical predictors. Test the performance of this Naïve Bayes model on the test data. In this case, it performs poorer than our logistic regression model that uses variable transformations, and is very difficult to interpret – it would be hard for a doctor to use this model to make recommendations about lifestyle modifications to reduce risk of diabetes. On the plus side, it is very fast to fit and still performs quite well for such a simple model!