Classification Model Evaluation

Ruoqing Zhu

Last Updated: October 03, 2024

Contents

Evaluating Classification Models

In previous lectures we always use the classification error as a criterion to evaluate a classification error. That is

$$\frac{1}{n_{\text{test}}} \sum_{i \in \text{test}} \mathbf{1}\{\widehat{y}_i \neq y_i\}$$

For most applications, this might be OK. For example, we can predict $\widehat{y}$ as 1 if the estimated probability form a logistic regression $\widehat{P}(Y|X = x)$ is larger than 0.5. However, if we have an **unbalanced data** situation, this may not provide enough we might be in trouble, since most or even all observations will be predicted as one class. And the prediction accuracy would simply be the proportion of y_i's being one of the class. For example, if we have a dataset that 90% of the observations are label 0, then simply predicting all observations to 0 will give us 90% accuracy.

For logistic regression, or any classification models, this situation may happen when the dataset is extremely unbalanced. Here is an example from the `ROSE` package.

```
  library(ROSE)
## Loaded ROSE 0.0-4

  # this is a simulated data
  data(hacide)

  # class label frequency in testing data
  table(hacide.test$cls)
##
##   0   1
## 245   5
```

```
  # fit a logistic regression
  logistic.fit <- glm(cls ~., data = hacide.train, family = binomial)
  pred = predict(logistic.fit, newdata = hacide.test, type = "response")
```

If we use 0.5 as the cut-off, then everything is predicted into 0. However, the error rate of this mode is $5/(245+5) = 0.02$.

```
  table(pred > 0.5, hacide.test$cls)
##
##           0   1
##   FALSE 245   5
```

However, if we use 0.11 as the cut-off, we can identify a subset of observations with high chance of being 1. Although the error rate is worse $7/250 = 0.028$, but this model is more useful. But how do we evaluate a model?

```
  table(pred > 0.11, hacide.test$cls)
##
##           0   1
##   FALSE 240   2
##   TRUE    5   3
```

It seems that we 1) need to use different cut-off values, which will provide a more comprehensive evaluation of the model performance, and 2) need some new measurements other than the prediction error to evaluate different results generated by these different cut-offs.

Example: Diagnostic Testing

The approaches we are considering are originated from diagnostic testing. For example, the anitgene test of SARS-CoV-2 works by binding antibody with the virus. A user can then visualize whether there are captured virus on the device. Preganency test works in the same way.

add covid.png here

However, these tests are not 100% accurate. For example, one need to (visually) decide if the test line on the device is strong enough to conclude a positive test result. If this line is too weak, then one may make a wrong decision.

```
  table(pred > 0.01, hacide.test$cls)
##
##           0   1
##   FALSE 191   1
##   TRUE   54   4
```

Depending on the situation, there could be a benefit of using higher error rate, but being more sensitive of the test. For example, if we use 0.01 as the cut-off, we can detect 4 out of 5 positive samples. However, to do this, we need to report 58 samples as positive. This is not a very specific test. If we use 0.11 as the cut-off, we report 8 positives, with 3 of them as correct ones. All of these decisions are actually worse than the 5/250 error rate by simply reporting everything as negative. However, that model is not really useful.

Sensitivity and Specificity

	True 0	True 1
Predict as 0	True Negative (TN)	False Negative (FN)
Predict as 1	False Positive (FP)	True Positive (TP)

- **Sensitivity** (also called "Recall") is the defined as the **true positive rate** (among the True 1 population, what proportion is correctly identified by the model)

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **Specificity** is the defined as the **true negative rate** (among the True 0 population, what proportion is correctly identified by the model)

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

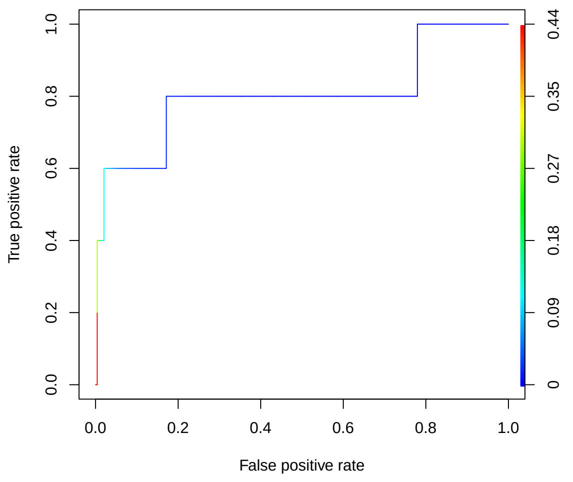
In the previous example, if we use 0.5 as the cut-off, the sensitivity is $0/5 = 0$, meaning that the model is not sensitive in terms of telling which subject may be at risk of an event. But the model is super specific (100%) because it is very conservative in terms of concluding any one at risk.

On the other hand, if we use 0.11 as the cut-off, the sensitivity is much better $3/5 = 60$, but we do not sacrifice much the specificity, $240/245 = 97.96$. How about we summarize all of these information into one measurement?

The ROC Curve and AUC

If we alter the cut off value, we can get different sensitivity and specificity values for each choice of the cut off value. Then we can plot `1 - specificity` (false positive rate) versus the `sensitivity` (true positive rate). This is called the ROC (Receiver Operating Characteristic) curve, and it can be calculated automatically with existing packages. The closer this curve to the top-left, the better performance this model is. A common measure is the area under the ROC curve (AUC).

```r
library(ROCR)
roc <- prediction(pred, hacide.test$cls)

# calculates the ROC curve
perf <- performance(roc,"tpr","fpr")
plot(perf,colorize=TRUE)
```



```
  # this computes the area under the curve
  performance(roc, measure = "auc")@y.values[[1]]
## [1] 0.8040816
```

There are many other packages that can perform similar calculations. However, be careful that some of them may require a specific orientation of your confusion table. Always read the documentation before using a new package.

Practice Question

Use the following data to calculate the AUC if we fit a logistic regression using the `glm()` function.

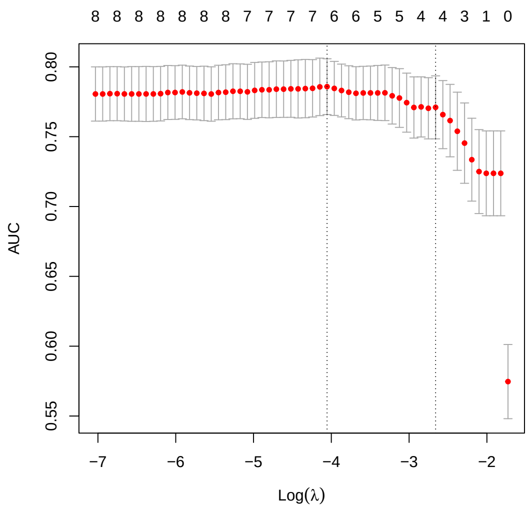
```
    set.seed(1)
    n = 150
    x = rnorm(n)
    eta = exp(-1 + 2*x) / (1 + exp(-1 + 2*x))
    y = rbinom(n, size = 1, prob = eta)
```

```
    logistic.fit <- glm(y ~ x, family = binomial)
    pred = predict(logistic.fit, newdata = data.frame("x" = x), type = "response")

    roc <- prediction(pred, y)
    perf <- performance(roc,"tpr","fpr")
    plot(perf,colorize=TRUE)
    performance(roc, measure = "auc")@y.values[[1]]
```

Cross-validation Using AUC

We can use AUC as the criteria to select the best model instead of using the prediction accuracy. In the `glmnet` package, we can specify `type.measure = "auc"`.

```
    library(glmnet)
## Loading required package: Matrix
## Loaded glmnet 4.1-8
    library(ElemStatLearn)
    data(SAheart)
    lasso.fit = cv.glmnet(x = data.matrix(SAheart[, 1:9]), y = SAheart[,10],
                          nfold = 10, family = "binomial", type.measure = "auc")

    plot(lasso.fit)
```



```
    lasso.fit$lambda.min
## [1] 0.01733797
    coef(lasso.fit, s = "lambda.min")
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept) -6.007447249
## sbp          0.002418113
## tobacco      0.064475340
## ldl          0.126087677
## adiposity    .
## famhist      0.735876295
## typea        0.023547995
## obesity      .
## alcohol      .
## age          0.040806121
```

For the AUC, the larger value the better. Hence we pick the λ value that gives the best cross-validated AUC. This is still called `lambda.min`.

Practice Question

Use the Cleveland heart disease data and use `num > 0` as the event definition. Fit a ridge regression and use AUC to select the best model.

```
    heart = read.csv("processed_cleveland.csv")
    heart$Y = as.factor(heart$num > 0)
    table(heart$Y)
##
## FALSE  TRUE
##   164   139
```

```
    lasso.fit = cv.glmnet(x = data.matrix(heart[, 1:13]),
                          y = heart[,15], nfold = 10,
                          family = "binomial",
                          alpha = 1,
                          type.measure = "auc")

    plot(lasso.fit)
    lasso.fit$lambda.min
    coef(lasso.fit, s = "lambda.min")
```