Cross-Validation: The Right and Wrong Way

Todd DeLuca

10/26/2017

Introduction

In “Cross-Validation: right and wrong” (https://www.youtube.com/watch?v=S06JpVoNaA0), Tibshirani and Hastie propose an experiment to see the effect of feature selection:

- Generate 50 samples for 5000 random predictor variables
- Generate 50 samples of a random binary outcome variable
- Compare the estimated error rate from two approaches using cross-validation:

    1. The Wrong Way:
        - Select the 100 predictor variables most correlated with the outcome
        - Use cross-validation to estimate the error rate by:
            - Training a model
            - Evaluating the model on held-out data
    2. The Right Way:
        - Use cross-validation to estimate the error rate by:
        - Select the 100 predictor variables most correlated with the outcome
        - Train a model
        - Evaluate the model on held-out data

It is easy to see that the true error rate of any model *should be* close to 50%, since the class labels have no association with the predictor variables. Using cross-validation the wrong way, the estimated error rate is less than 50%. This is an example of Leakage (https://www.kaggle.com/wiki/Leakage) of information about test data into training data.

Create the Dataset

```
n = 50 # number of observations
p = 5000 # number of predictor variables
m = 100 # number of predictors to select

set.seed(1)
X = data.frame(replicate(p, rnorm(n)))
# Randomly assigned binary class labels
Y = as.factor(sample(c(rep(1, n/2), rep(0, n/2))))
df = cbind(Y, X)

# train and test indices for sanity checking
permuted.indices = sample(1:n)
test = permuted.indices[1:floor(n/2)]
train = permuted.indices[(floor(n/2)+1):n]
```

Write a function to train a model on a subset of the data.

```
train.model = function(data, index) {
  return(glm(Y~., data=data, family="binomial", subset=index))
  # return(lm(Y~., data=data, subset=index))
}
```

Write a function to compute the error rate of a model on a subset of the data.

```
error.rate = function(model, data, index) {
  df = data[index,]
  probs = predict(model, df, type="response")
  preds = rep("0", length(probs))
  preds[probs>0.5]="1"
  return(mean(preds!=df$Y))
}
```

Write a function to select the n features most correlated with the class labels.

```
association = function(x, y) {
  # a measure of association between numerical X variable and categorical Y variable
  # return(glm(y~x, family="binomial")$aic)
  # return(-summary(aov(x~y))[[1]]$F[[1]])
  # force y to be numeric in the case that it is a binary categorical variable
  return(cor(x,as.numeric(y)))
}

# select the first n columns most associated with the y.index column
# return only the y column and selected columns from data
select.predictors = function(data, index, y.index, n) {
  X = data[index, -y.index] # remove y column
  y = data[index,y.index]
  # select the first n predictors (the columns of dataframe X) most associated with binary vector y
  selected.X.names = names(sort(sapply(X, function(x) association(x, y)))[1:n])
  selected.X.indices = sapply(selected.X.names, function(name) which(colnames(data)==name))
  return(data[,c(y.index, selected.X.indices)])
  # selected.X = X[,selected.X.names]
  # return(cbind(data.frame(Y=y), selected.X))
}
```

Write a function to do cross-validation and return an estimated test error rate.

```
library(caret)
cv.error = function(data, k, train.fun, test.fun, r=1){
  # create list of k folds.  each fold contains a vec of row indices.
  # set.seed(1)
  test.err = 0
  for (i in 1:r) {
    folds = createFolds(1:nrow(data), k)
    # for each fold, train then test.
    for(i in 1:length(folds)) {
      test.idx = folds[[i]]
      train.idx = unlist(folds[(1:length(folds))[-i]]) # combine non-test-fold indices into one vec.
      model = train.fun(data, train.idx)
      test.err = test.err + test.fun(model, data, test.idx)
    }
  }
  return(test.err/(r*k)) # mean test error across folds
}
```

The Right Way: Use k-fold cross-validation for feature selection, training and testing.

```
# Right: for cross-validation, make feature selection a part of model training,
# so it will only be done on training data
train.fun = function(data, index) {
  filtered.df = select.predictors(data, index, grep("^Y$", colnames(data)), m)
  fit = train.model(filtered.df, index)
}
# Right: selectd features only using training data
df1 = select.predictors(df, train, grep("^Y$", colnames(df)), m)
fit1 = train.model(df1, train)
# Training error should be close to 0%, because there are so many predictors and so few observations
error.rate(fit1, df1, train)
```

```
## [1] 0
```

```
# Test error should be close to 50%, because there is no relationship between predictors and outcome.
cv.error(df, k=5, train.fun=train.fun, test.fun=error.rate, r=10)
```

```
## [1] 0.5201566
```

The Wrong Way: Select predictors using all the data and then use cross validation for training and testing

```
# Wrong: select predictors using all the data
df2 = select.predictors(df, 1:nrow(df), grep("^Y$", colnames(df)), m)
fit2 = train.model(df2, train)
# Training error should be close to 0%, because there are so many predictors and so few observations
error.rate(fit2, df2, train)
```

```
## [1] 0
```

```
# Test error should be less than 50%, because the predictors have been selected for
# their association with the outcome
cv.error(df2, k=5, train.fun=train.model, test.fun=error.rate, r=10)
```

```
## [1] 0.3852626
```