Niave Bayes Classification in R

November 27, 2012

Naive Bayes is one of the simplest classification algorithms, but it can also be very accurate. At a high level, Naive Bayes tries to classify instances based on the probabilities of previously seen attributes/instances, assuming complete attribute independence. Oddly enough, this usually works out to give you a good classifier.

Now, I am going to show you how to do Naive Bayes classification in R. First, you need to install a few packages, so time to boot up your R instance!

```
install.packages("klaR")
install.packages("caret")

library("klaR")
library("caret")
```

Caret is a very nice data mining package for R, it has tons of awesome features. The package klaR contains our Naive Bayes classifier.

Everyone does the iris dataset first, so I wont break that trend. Later, I will show you a much more interesting dataset. Loadup the iris dataset and separate the labels from the attributes.

```
x = iris[,-5]
y = iris$Species
```

Now, x has all the attributes and y has all the labels. Now we can train our model.

```
model = train(x,y,'nb',trControl=trainControl(method='cv',number=10))
```

This one line will generate a Naive Bayes model, using 10-fold cross-validation. From above, x is the attributes and y is the labels. The ‘nb’ tells the trainer to use Naive Bayes. The trainController part tells the trainer to use cross-validataion (‘cv’) with 10 folds.

You can then print out the model:

```
model

>
150 samples
  4 predictors
  3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Cross-Validation (10 fold)

Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...

Resampling results across tuning parameters:

  usekernel  Accuracy  Kappa  Accuracy SD  Kappa SD
  FALSE      0.953     0.93   0.0632       0.0949
  TRUE       0.96      0.94   0.0562       0.0843

Tuning parameter 'fL' was held constant at a value of 0
Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were fL = 0 and usekernel = TRUE.
```

Awesome! We have a 94% kappa, life is good! One of the really cool things about caret's train function is that it will fine-tune the parameters to your model (to a certain extent).

Now that we have generated a classification model, how can we use it for prediction? Easy!

```
predict(model$finalModel,x)
```

This will printout a bunch of lines. Near the top you can see the classes it predicted, then you will see the posterior probabilities in the bottom half. As we are only interested in the class predictions, we can grab only those with the following line.

```
predict(model$finalModel,x)$class
```

Lets build a confusion matrix so that we can visualize the classification errors.

```
table(predict(model$finalModel,x)$class,y)

>
            y
```

```
            setosa versicolor virginica
setosa          50          0         0
versicolor       0         47         3
virginica        0          3        47
```

This will generate a confusion matrix of the predictions of your Naive Bayes model versus the actual classification of the data instances.

Now, what I have done here is actually a terrible idea. You never want to use the same data you trained on for testing, but this is only an example. I will provide a better example later on.

That is basically how you do Naive Bayes classification in R with cross-validation. Now, lets try this on a more interesting dataset, spam emails.

```
install.packages('ElemStatLearn')
library('ElemStatLearn')
library("klaR")
library("caret")

sub = sample(nrow(spam), floor(nrow(spam) * 0.9))
train = spam[sub,]
test = spam[-sub,]

xTrain = train[,-58]
yTrain = train$spam

xTest = test[,-58]
yTest = test$spam

model = train(xTrain,yTrain,'nb',trControl=trainControl(method='cv',number=

prop.table(table(predict(model$finalModel,xTest)$class,yTest))

>
       yTest
             email       spam
  email 0.33405640 0.02603037
  spam  0.24945770 0.39045553
```

Here we take 90% of the dataset to train on, and then we test on the remaining 10%.

These results will be different on each run, as sample is a random function. The results aren't great, but for a very simple classifier, they are really good!