Naive Bayes continued and Principal Component Regression

Anthony Garino, Wenduo Wang

August 11, 2016

Continuing on Naive Bayes

To estimate the probability of a word $P(word j is randomly selected)$, the normal ways is

$$\hat{w_j} = \frac{\#word\ j\ in\ corpus}{total\ \#words\ in\ corpus}$$

However there are problems associated with this approach.

```
1. It doesn't work for alien words that appear in test set but not included in training set.
2. The model are biased toward certain words in the sample if the sample size is small and doesn't represent t
```

To overcome this problem, we add a *pseudo-count* so that

$$\tilde{x_{ij}} = x_{ij} + r$$

where r is the *pseudo-count* or *smoothing factor*. Therefore, if a new word shows up, the model will assign a non-zero weight to it, and as the training set grows larger, the effect of a new word becomes smaller. A way to choose r is given by *Laplacian smoothing* $r = \frac{1}{N}$ where N is the total number of words in the corpus.

As an example, if a person only sees 3 white swans, he is not so sure to predict all the swans are white, so he could say there will be a probability $\frac{3}{4}$ that the next swan is also white. But as he becomes with more experience, he is more confident to assert the next swan he will see is white.

Another example of text mining in R

For text mining, `tm` library is the most popular tool in R which provides a ton of classes and functions.

```
library(tm)
```

After reading in the corpus, i.e. a list of several texts in R, use `Corpus(VectorSource())` function to convert the corpus into a `tm` `Corpus` object.

And as a routine, there are a set of procedures to tokenize the corpus, including make all the words lower case, stem numbers, remove punctuations and strip extra spaces.

```
my_documents = tm_map(my_documents, content_transformer(tolower))
my_documents = tm_map(my_documents, content_transformer(removeNumbers))
my_documents = tm_map(my_documents, content_transformer(removePunctuation))
my_documents = tm_map(my_documents, content_transformer(stripWhitespace))
```

Another important procedure is to remove the stopwords, which can be assisted by the `stopwords` methods in `tm` library. Below is an example which will return a list of stopwords in English.

```
stopwords("en")[1:10]
```

```
>  [1] "i"          "me"         "my"         "myself"     "we"
>  [6] "our"        "ours"       "ourselves" "you"        "your"
```

Then remove the stopwords from the corpus.

```
my_documents = tm_map(my_documents, content_transformer(removeWords), stopwords("en"))
```

Next transform the `Corpus` object to a `DocumentTermMatrix` object which is mathematically easier to work with.

With the *DTM* object you can perform different analysis

```
DTM_simon = DocumentTermMatrix(my_documents)
inspect(DTM_simon)
findFreqTerms(DTM_simon, 50)
findAssocs(DTM_simon, "market", .5)
```

As another way to simplify the computation is to remove sparse terms in the *DTM* object and convert it to a normal matrix.

The matrix allows us to calculate the term frequency of each word and apply PCA.

```
DTM_simon = removeSparseTerms(DTM_simon, 0.95)
X = as.matrix(DTM_simon)
X = X/rowSums(X)

pca_simon = prcomp(X, scale=TRUE)
```

Principal Component Regression

In real life, there are high-dimension problems that are better solved by PCR. Such problems have two characteristics:

```
1. Feature # is close to or even larger than sample size
2. Features are highly correlated with each other
```

In such cases, it is generally more efficient to extract the principal components of the samples and translate original values into scores on the PCs, where $score = x \cdot v$, then run regression on the score vectors to fit a model. Thereby comes the name PCR.

Below is an example to apply PCR on gasoline Near-infrared spectroscopy data analysis. In this example, there are more features than observations.

Without PCR, the normal linear regression on the data will return an $R^2 = 1$, which usually indicates a problem. This can be understood in this case that since there are more features, it is possible for the model to create a unique set of features for each observation, but that model will not work for other observations, and therefore is problematic.

```
gasoline = read.csv('data/gasoline.csv', header=TRUE)
X = as.matrix(gasoline[,-1])
y = gasoline[,1]
lm1 = lm(y ~ X)
summary(lm1)$r.squared
```

```
> [1] 1
```

To fix this problem, we use PCA to reduce the dimension of the problem. After extracting the first 5 PCs, we calculate the socres of the observations on the 5 PCs, which are used as the new features to fit a linear model. The new model still fits the observations very well and has $R^2 = 0.979$, which means down to this point, the majority of information among the sample population is preserved.

```
pc_gasoline = prcomp(X, scale=TRUE)

K = 5
V = pc_gasoline$rotation[,1:K]
```

```r
scores = X %*% V
pcr1 = lm(y ~ scores)
summary(pcr1)$r.squared
```

```
> [1] 0.9789781
```