Gibbs Sampler for Email Classification

Patrick Mellady

Here is an implementation of a Gobbs sampler to email spam/not-spam data to perform classification.

Distributional Assumptions

We start by assuming distributional properties about our data. We make the assumption that emails are drawn from a mixture of two poisson distributions. Let Y_i be the number of "spam" words in the i^{th} email, then we are assuming

$$Y_i \sim qpois(\lambda_1) + (1-q)pois(\lambda_0)$$

In this situation, the λ_1 distribution represents the count of "spam" words in emails that are not spam while the λ_2 distribution represents the count of "spam" words in emails that are spam. When we say "spam" words, we mean words that are typically seen in spam email. We will see more on this later.

We will introduce a latent variable, Z_i, which labels each email as spam or not. Using the notation that $\boldsymbol{\lambda} = (\lambda_0, \lambda_1)$, $\mathbf{Y} = (Y_1, Y_2, \cdots, Y_n)$, and $\mathbf{Z} = (Z_1, \cdots, Z_n)$, we have the following heierarchical structure

$$\begin{aligned} Y_i|q, \boldsymbol{\lambda}, Z_i &\sim pois(\lambda_{Z_i}) \\ Z_i|q &\sim bern(q) \\ \lambda_1, \lambda_2 &\sim exp(\beta) iid \\ q &\sim beta(a, b) \end{aligned}$$

The idea of the method is this:

We start by finding or approximating the value of each Y_i (approximation for implementation purposes). After we have the Y_is, we will sample repeatedly in the following manner

$$\begin{aligned} \lambda_i^{(k+1)} &\sim \lambda_i|q^{(k)}, \mathbf{Y}, \mathbf{Z}^{(k)} \\ q^{(k+1)} &\sim q|\boldsymbol{\lambda}^{(k)}, \mathbf{Y}, \mathbf{Z}^{(k)} \\ Z_i^{(k+1)} &\sim Z_i|q^{(k)}, \mathbf{Y}, \boldsymbol{\lambda}^{(k)} \end{aligned}$$

Upon sampling long enough, we will implement a decision rule to classify emails as spam or not-spam using the sampled Z_is.

Distributional Derivations

Starting with the heierarchical model stated above, we derive each of the conditional distributions required in the model. We start with the simplest

Distribution of $\mathbf{Y}|q, \boldsymbol{\lambda}, \mathbf{Z}$

Since Z_i identifies he distribution from which the email was drawn, we have that

$$\begin{aligned} Y_i &\sim pois(\lambda_1) \text{ if } Z_i = 1 \\ Y_i &\sim pois(\lambda_0) \text{ if } Z_i = 0 \end{aligned}$$

Distribution of $\lambda_i|\mathbf{Y}, q, \mathbf{Z}$

Let us start with λ_0. We can write

$$P(\lambda_0|\lambda_1, \mathbf{Y}, q, \mathbf{Z}) = P(\lambda_0|\lambda_1, \mathbf{Y}, \mathbf{Z}) = c\pi(\lambda_0)\prod_{i=1}^{n} f(y_i|\lambda_0)^{1-z_i} f(y_i|\lambda_1)^{z_i}$$

This is a straightforward application of Bayes' rule where the likelihood of the data is given by $\prod_{i=1}^{n} f(y_i|\lambda_0)^{1-z_i} f(y_i|\lambda_1)^{z_i}$, the posterior distribution is $\pi(\lambda_0)$ and the constant c is for normalization. Since we assume an exponential distribution on λ_0 and since $f(y_i|\lambda_1)^{z_i}$ has no dependence on λ_0, we can simplify this as follows:

$$P(\lambda_0|\lambda_1, \mathbf{Y}, q, \mathbf{Z}) = c_1\frac{1}{\beta}e^{-\lambda_0/\beta}\prod_{i=1}^{n} f(y_i|\lambda_0)^{1-z_i} = c_1\frac{1}{\beta}e^{-\lambda_0/\beta}\prod_{i=1}^{n}(\frac{e^{-\lambda_0}\lambda_0^{y_i}}{y_i!})^{1-z_i}$$

We can further simplify this by rearranging more constants to get

$$P(\lambda_0|\lambda_1, \mathbf{Y}, q, \mathbf{Z}) = c_2\lambda_0^{\sum x_i(1-z_i)}e^{-\lambda_0(\frac{1}{\beta}+\sum(1-z_i))}$$

This all tells us that $\lambda_0|\lambda_1, \mathbf{Y}, q, \mathbf{Z} \sim gamma(\sum x_i(1-z_i)+1, (\frac{1}{\beta}+\sum(1-z_i))^{-1})$. Similarly, by symmetry, we have $\lambda_1|\lambda_0, \mathbf{Y}, q, \mathbf{Z} \sim gamma(\sum x_i z_i + 1, (\frac{1}{\beta}+\sum z_i)^{-1})$.

The distribution of $q|\mathbf{Y}, \mathbf{Z}, \boldsymbol{\lambda}$

Note that $P(q|\mathbf{Y}, \mathbf{Z}, \boldsymbol{\lambda}) = P(q|\mathbf{Z}) = p(\mathbf{Z}|q)P(q)$. With the conditional distribution of $Z_i|q$ as in the model statement and with the assigned $beta(a, b)$ prior on q, this gives us

$$P(q|\mathbf{Y}, \mathbf{Z}, \boldsymbol{\lambda}) = cq^{a-1}(1-q)^{b-1}\prod_{i=1}^{n} q^{z_i}(1-q)^{1-z_i} = cq^{a+\sum z_i-1}(1-q)^{b+n-\sum z_i-1}$$

and hence $q|\mathbf{Y}, \mathbf{Z}, \boldsymbol{\lambda} \sim beta(a+\sum z_i, b+n-\sum z_i)$.

The distribution of $\mathbf{Z}|q, \mathbf{Y}, \boldsymbol{\lambda}$

Lastly, since we will be sampling the vector of Z_is simultaneously, we need to find the posterior distribution of $\mathbf{Z}|q, \mathbf{Y}, \boldsymbol{\lambda}$. To do this, note that

$$P(\mathbf{Z}|q, \mathbf{Y}, \boldsymbol{\lambda}) = P(\mathbf{Z}|q, \mathbf{Y}) = cP(\mathbf{Z}, q, \mathbf{Y}) = cP(\mathbf{Y}, q, \mathbf{Z})P(q, \mathbf{Z}) = cP(\mathbf{Y}, q, \mathbf{Z})P(\mathbf{Z}|q)P(q)$$

Since the prior distribution of q is seen as constant in the distribution of $\mathbf{Z}$ is can be absorbed into the constant to obtain

$$P(\mathbf{Z}|q, \mathbf{Y}, \boldsymbol{\lambda}) = c_1P(\mathbf{Y}|q, \mathbf{Z})P(\mathbf{Z}|q) = c_1P(\mathbf{Y}|\mathbf{Z})P(\mathbf{Z}|q)$$

Where we simplify $P(\mathbf{Y}|q, \mathbf{Z})$ to $P(\mathbf{Y}|\mathbf{Z})$ since the condition on $\mathbf{Z}$ makes q superfluous. Now, since the Y_is and Z_is are conditionally independent, we can write this as a product of the mass functions as follows

$$P(\mathbf{Z}|q, \mathbf{Y}, \boldsymbol{\lambda}) = c_1\prod_{i=1}^{n} f(y_i|\lambda_0)^{1-z_i} f(y_i|\lambda_1)^{z_i} q^{z_i}(1-q)^{1-z_i}$$

Note that all the bases which are being raised to powers of z_i have no dependence on z_i, so we will let $p_{0i} = f(y_i|\lambda_0)(1-q)$ and $p_{1i} = f(y_i|\lambda_1)q$, which gives

$$P(\mathbf{Z}|q, \mathbf{Y}, \boldsymbol{\lambda}) = c_1\prod_{i=1}^{n} p_{01}^{1-z_i}p_{1i}^{z_i}$$

Defining $p = \frac{p_{1i}}{p_{1i}+]_{0i}}$ and simplifying the above gives

$$P(\mathbf{Z}|q, \mathbf{Y}, \boldsymbol{\lambda}) = c_2 \prod_{i=1}^{n} (1-p)^{1-z_i} p^{z_i} = c_2 \prod_{i=1}^{n} f(z_i|p)$$

Implementation

With the above distributions, we can perform the iterative sampling algorithm as stated from before. We are using the *spam.csv* data set. Here is where we return to the idea of "spam words".

We use he training data to select some words commonly used in spam emails. We will not use every word found in spam emails (because spam emails can contain words that are used in non-spam emails) but we will select the words that are highly unlikely to show up in non-spam emails. This can be seen in the code below.

```
# Reading the data
spam<-read.csv("spam.csv")
n<-length(spam[,1])
real<-rep(1,n)
real[spam[,1]=="spam"]<-0


# Create a list of spam words to test each email against
spam_words<-tolower(c("Free", "entry", "wkly", "comp", "win", "final", "tkts", "Text", "entry",
              "FreeMsg", "darling", "fun", "XxX", "entitled", "Update", "credit", "claim",
              "subscription", "special", "pleased", "valued", "jackpot", "prize"))

# Assigning spam word counts to each email
# Create X vector representing the number of spam words in each email
X<-rep(0,n)

# Search through each email to see update spam word count
vec<-spam_words
for(word in vec){
  for(i in 1:n){
    if(grepl(word, tolower(spam[i,2]))){
      X[i]<-X[i]+1
    }
  }
}
```

As shown above, the commonly seen spam words are stored in a vector. We then use these words to assign a count of all the spam words in any particular email. Now, we will use the distributional model from above to run a Gibbs sampler and classify the emails.

```
Z0<-rbinom(n, 1, .5)        # Initial value of Z for Gibbs Sampler
beta<-1                     # Used in exponential prior
a<-1                        # Used in beta prior
b<-1                        # Used in beta Prior
m<-1000                     # Number of Gibbs Sampler iterations

# Defining functions to make easy sampling
samp<-function(X, l0, l1, p){
  temp<-c()
  p.0<-(1-p)*dpois(X, l0)
  p.1<-p*dpois(X, l1)
  q<-p.1/(p.1+p.0)
```

```
  for(i in 1:n){
    temp<-c(temp, rbinom(1,1,q[i]))
  }
  return(temp)
}

# Running the Gibbs Sampler
Z<-vector(mode='list', length=m+1)                                   # Creating the Z vector
Z[[1]]<-Z0                                                          # Initializing the Z vector
lambda<-vector(mode="list", length=m+1)
mix_param<-c()
for(j in 1:m){
  Y<-Z[[j]]
  l0<-rgamma(1,sum(X*(1-Y))+1, scale=beta/(beta*sum(1-Y)+1))  # Sample conditional lambda0
  l1<-rgamma(1,sum(X*Y)+1, scale=beta/(beta*sum(Y)+1))        # Sample conditional lambda1
  lambda[[j]]<-c(l0, l1)                                      # Saving iterations of the parameters
  p<-rbeta(1, a+sum(Y), b+sum(1-Y))                           # Sample conditional mixing parameter
  mix_param<-c(mix_param, p)                                  # Saving iterations of the mixing paramet
  Z[[j+1]]<-samp(X, l0, l1, p)                                # Sample the labeling variables
}
```

With the sampling done, we can now use the resulting list of **Z** to make a decision rule about whether or not an email is spam.

```
# Finding the posterior mean of Z
Z.bar<-rep(0,n)
for(i in 1:m+1){
  Z.bar<-Z.bar+Z[[i]]
}
Z.bar<-Z.bar/m

# Rounding the estimates to binary spam/not-spam labels
Z.bar<-round(Z.bar)
```

We can now check how our model fits by performing some error analysis with the true classification of the emails.

```
# Error Analysis
# Finding the accuracy of the algorithm
error<-sum((Z.bar-real)^2)
percent_correct<-error/n
percent_correct                               # 91% accurate

## [1] 0.9104451

# Counting the number of emails marked as not spam when they actually are
spam_count<-0
for(i in 1:n){
  if(Z.bar[i]==1 & real[i]==0){
    spam_count<-spam_count+1
  }
}
spam_count

## [1] 266
```

```r
# Counting the number of non-spam emails marked as spam
ham_count<-0
for(i in 1:n){
  if(Z.bar[i]==0 & real[i]==1){
    ham_count<-ham_count+1
  }
}
ham_count
```

```
## [1] 4807
```

We can also see what the model estimates our parameters to be and, using those estimates, we can see which emails the model has difficulty classifying.

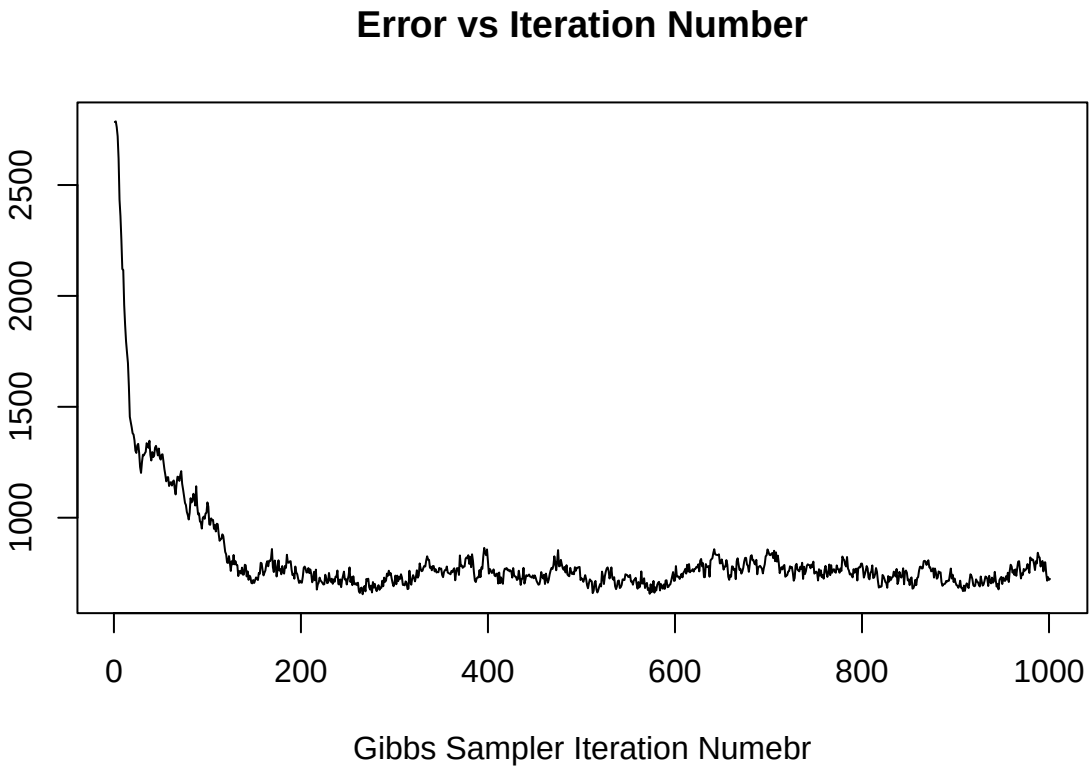
```r
# Determining the real values of the parameters of our model
l0_real<-mean(X[real==0])
l1_real<-mean(X[real==1])
p_real<-sum(real)/n

# Finding the spam word count of the mislabeled emails
missed<-X[real!=Z.bar]
l_missed<-mean(missed)                    # Average spam word count of the missed emails
l_01<-(l0_real-l1_real)/2                 # Average spam word count between the two groups
                                          # the missed emails did not give enough
                                          # information to definitively classify them. Improving the
                                          # spam word bank would better separate these groups
```

Lastly, we plot the misclassification rate with respect to the iteration number to see how the model performs as the iterations increase.

```r
# Finding and plotting the error at each iteration of the Gibbs Sampler
error_vec<-c()
for(i in 1:(m+1)){
  error_vec<-c(error_vec, sum(((1-Z[[i]])-real)^2))
}

plot(error_vec, xlab="Gibbs Sampler Iteration Numebr", ylab=" ",
     main="Error vs Iteration Number", type="l")
```


Error vs Iteration Number
1000
1500
2000
2500
0
200
400
600
800
1000
Gibbs Sampler Iteration Numebr