CS 109B, Spring 2017, Homework 3: Introduction to Bayesian Methods

Jan 2016

Solutions by: **CS109B Staff**

Problem 1: Authorship Attribution

In this problem, the task is to build a model that can predict if a given news article was written by one of two authors. We will explore two different probabilistic models for this binary classification task. Your model will be evaluated based on its classification accuracy.

Pre-Processing Details

The data is a subset of the Reuter's 50×50 data set in the UCI repository. We provide you with pre-processed versions of these data sets `dataset1_train_processed_subset.txt` and `dataset1_test_processed_subset.txt`.

The text articles have been converted into numerical feature representations. We use the *bag-of-words* feature encoding, where each article is represented by a vector of word counts. More specifically, we construct a dictionary of K frequent words in the corpus, and represent each article using a K-length vector: the i-th entry in this vector contains the number of times the dictionary word i occurs in the article.

We then further preprocessed the data to include the **100 words** that are distinctive to each author to improve our predictions. The dictionary of words used for this representation have been provided in the file `words_preprocessed.txt`.

Hint: Make use of the `header` argument in either `read.csv` or `read.table`.

We begin with a simple Naive Bayes classifier for this task, and will then explore a fully Bayesian modeling approach.

Part 1a: Naive Bayes Classifier

Fit a Naive Bayes classification model to the training set and report its classification accuracy on the test set. The input to this model is the word count encoding for an article, and the output is a prediction of the author identity.

Questions:

- Using 0-1 loss what is the overall accuracy?
- Explain to a layperson in clear language the problem in using a Naive Bayes probabilistic model for this task.

Hint: You may use the `naiveBayes` function in the `e1071` library for fitting a Naive Bayes classifier.

Solution

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Load libraries and data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

suppressMessages(library(e1071))
suppressMessages(library(caret))
suppressMessages(library(mclust))
suppressMessages(library(MLmetrics))

train = read.csv('datasets_preprocessed_code/dataset1_train_processed_subset.txt', header = FALSE)
test = read.csv('datasets_preprocessed_code/dataset1_test_processed_subset.txt', header = FALSE)

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Fit Naive Bayes Classifier
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

nb.label = test[, 1]
nb.features = test[, -1]

nb = naiveBayes(V1~., data = train)
nb.pred = predict(nb, nb.features)

table(nb.pred, nb.label)
```

```
##                 nb.label
## nb.pred          AaronPressman AlanCrosby
##   AaronPressman             49         23
##   AlanCrosby                 1         27
```

```
print(classError(nb.pred, nb.label)$errorRate)
```

```
## [1] 0.24
```

The theory behind the Naive Bayes model is to use Bayes Rule to calculate $P(\text{author } A|W_1, W_2, ...)$, which is the probability we want for classification, using the likelihood $P(W_1, W_2, ...|\text{author } A)$. Here, the W_i are indicators for the number of times that word i is in the document. $P(W_1, W_2, ...|\text{author } A)$ is not a precise likelihood, however, since each exact combination of words occurs very rarely, giving us a very small sample size. The Naive Bayes model makes the assumption that $P(W_1, W_2, ...|\text{author } A)$ can be estimated by $P(W_1|\text{author } A)P(W_2|\text{author } A)...$, that is, that the word occurances are conditionally independent given knowledge of the author. This is a very strong and potentially problematic assumption, since in most writing it is not true that the words are conditionally independent.

Part 1b: Dirichlet-Multinomial Model

Let us consider an alternate Bayesian approach for authorship inference. We recommend a Dirichlet-Multinomial probabilistic model for articles by each author. The author identity for an article can be predicted by computing the posterior predictive probability under this model. This is similar to the approach described in class for the Beatles musical authorship inference example, except that we shall use word features in place of transition couplets. **Probability model:** Let $(y_1^A, y_2^A, \ldots, y_K^A)$ denote the total counts of the

K dictionary words across the articles by author A, and $(y_1^B, y_2^B, \dots, y_K^B)$ denote the total counts of the K dictionary words across the articles by author B. We assume the following *multinomial model*:

$$p(y_1^A, y_2^A, \dots, y_K^A) \propto (\theta_1^A)^{y_1^A} (\theta_2^A)^{y_2^A} \dots (\theta_K^A)^{y_K^A}$$

$$p(y_1^B, y_2^B, \dots, y_K^B) \propto (\theta_1^B)^{y_1^B} (\theta_2^B)^{y_2^B} \dots (\theta_K^B)^{y_K^B}.$$

The model parameters $(\theta_1^A, \dots, \theta_K^A)$ and $(\theta_1^B, \dots, \theta_K^B)$ are assumed to follow a *Dirichlet* prior with parameter α.

We provide you with an `R` function (`posterior_pA`) to calculate the posterior predictive probability under the above model, i.e. the posterior probability that a given test article was written by author A, based on the training data. The input to this function is

- the Dirichlet parameter α,
- the total word counts $(y_1^A, y_2^A, \dots, y_K^A)$ from all articles by author A in the training set,
- the total word counts $(y_1^B, y_2^B, \dots, y_K^B)$ from all articles by author B in the training set, and
- the word counts $(\tilde{y}_1, \dots, \tilde{y}_K)$ for a new test article.

The output is the posterior probability $P(A \mid data)$ that the test article was written by author A.

One can use the above function to infer authorship for a given test article by predicting author A if $p_i = p(A \mid y_i, data) > 0.5$ and author B otherwise.

Loss function

Evaluate the classification accuracy that you get on the test set using the log-loss function

$$\sum_i y_i \log(p_i) + (1 - y_i) \log(1 - p_i),$$

where y_i is the binary author identity in the test set for article i and p_i is the posterior mean probability that article i is written by author A. Along with the following choices of the Dirichlet parameter:

- α is set to 1
- α is tuned by cross-validation on the training set under log-loss – you can use the cross-validation function provided in HW1 as a skeleton.

Questions:

- What does setting $\alpha = 1$ imply?
- Do the above optimization. What is the optimal value of α? What is your final log-loss prediction error?
- For the optimal value of α, how do the accuracies (based on $0 - 1$ loss) obtained compare with the previous Naive Bayes classifier?

Solution

```r
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# function: calculates the probability author is Aaron Pressman
#   See lecture notes for formula
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

posterior_pA = function(alpha, yA = NULL, yB = NULL, y_til = NULL){
    # number of features
    K = length(yA)

    # total word counts
    n = sum(y_til)
    nA = sum(yA)
    nB = sum(yB)

    # posterior predictive distribution of being class A
    A1 = lfactorial(n) + lfactorial(nA) - lfactorial(n + nA)
    A2 = sum(lfactorial(y_til + yA)) - sum(lfactorial(y_til)) - sum(lfactorial(yA))
    A3 = lfactorial(n + nA) + lgamma(K*alpha) - lgamma(n + nA + K*alpha)
    A4 = sum(lgamma(y_til + yA + alpha) - lfactorial(y_til + yA) - lgamma(alpha))
    A5 = lfactorial(nB) + lgamma(K*alpha) - lgamma(nB + K*alpha)
    A6 = sum(lgamma(yB + alpha) - lfactorial(yB) - lgamma(alpha))

    # posterior predictive distribution of being class B
    B1 = lfactorial(n) + lfactorial(nB) - lfactorial(n + nB)
    B2 = sum(lfactorial(y_til + yB)) - sum(lfactorial(y_til)) - sum(lfactorial(yB))
    B3 = lfactorial(n + nB) + lgamma(K*alpha) - lgamma(n + nB + K*alpha)
    B4 = sum(lgamma(y_til + yB + alpha) - lfactorial(y_til + yB) - lgamma(alpha))
    B5 = lfactorial(nA) + lgamma(K*alpha) - lgamma(nA + K*alpha)
    B6 = sum(lgamma(yA + alpha) - lfactorial(yA) - lgamma(alpha))

    ratio_BA = exp(B1 + B2 + B3 + B4 + B5 + B6 - A1 - A2 - A3 - A4 - A5 - A6)

    # probability of being class A
    pA = 1/(1 + ratio_BA)

    return(pA)
}

log_loss= function(posterior, labels){
  eps = 1e-10
  # move predictions to slightly above 0 or slightly below 1
  posterior = pmin(pmax(eps, posterior), 1-eps)
  log_loss = labels * log(posterior) + (1 - labels) * (log(1 - posterior))
  total_log_loss = (-1/length(posterior)) * sum(log_loss)
  return(total_log_loss)
}

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Run on Test Dataset with alpha parameter = 1
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

# the total word counts from all articles by author A in the training set
yA = as.numeric(apply(train[train$V1 == 'AaronPressman',-1], 2, sum))
```

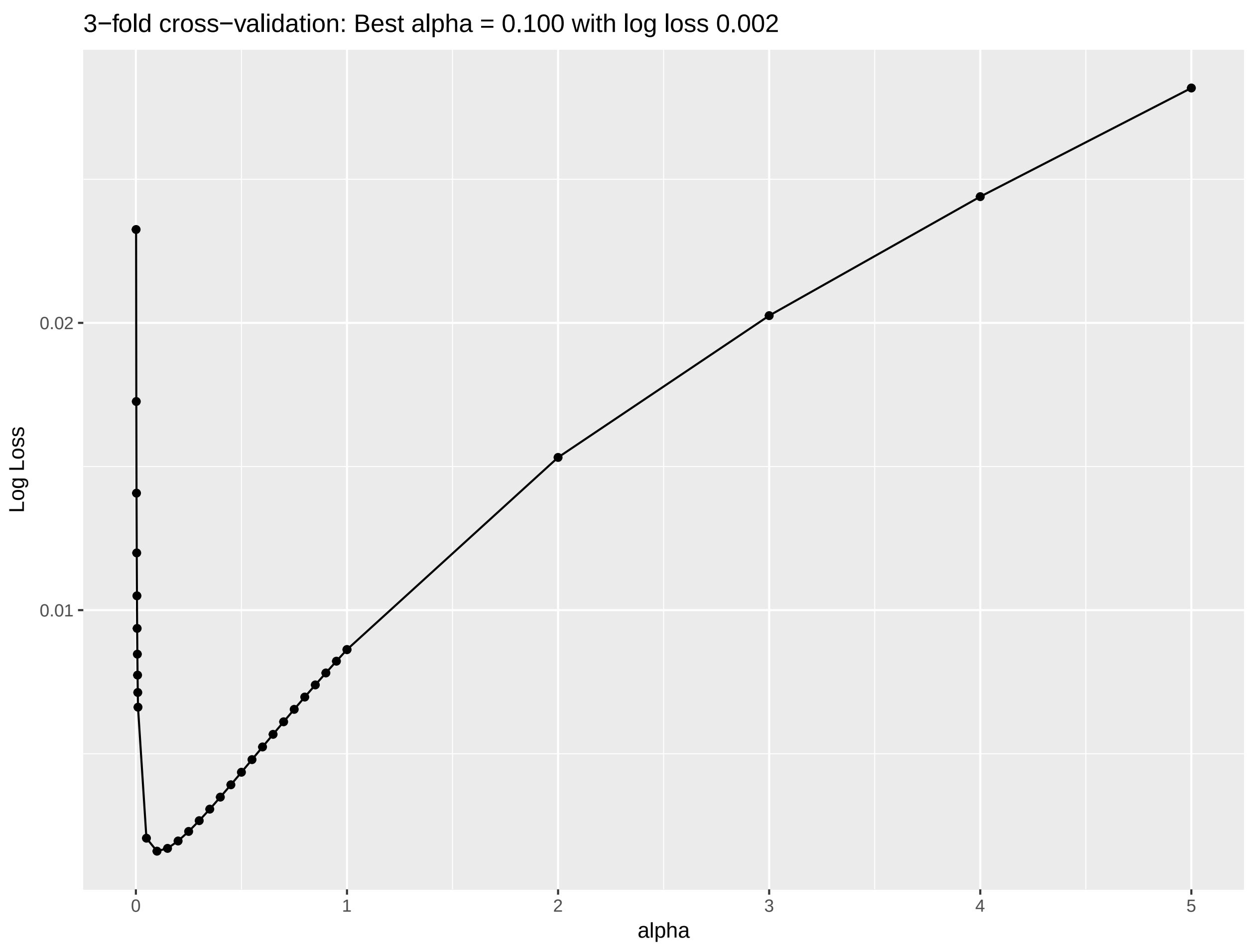
```r
# the total word counts from all articles by author B in the training set
yB = as.numeric(apply(train[train$V1 == 'AlanCrosby',-1], 2, sum))

labels <- test[, 1]
features <- test[, -1]

n.test = nrow(test)
dirichlet.probs = rep(NA, n.test)

for(i in 1:n.test){
    y_til = as.numeric(as.character(features[i, ]))
    dirichlet.probs[i] = posterior_pA(alpha = 1, yA = yA, yB = yB, y_til = y_til)
}

table(labels, dirichlet.probs > 0.5)
```

```
##
## labels          FALSE TRUE
##   AaronPressman     4   46
##   AlanCrosby       42    8
```

```r
label.bin <- ifelse(labels == "AaronPressman", 1, 0)

print(log_loss(dirichlet.probs, label.bin))
```

```
## [1] 0.407914
```

```r
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Above is the confusion matrix, tuning via cross-validation
#  should be quite easy to do from here.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

crossval_dirich = function(alpha_param, k) {
  # set seed here to ensure that we're considering the
  #   same cross-validation groups, so that alpha is
  #   the only variable
  set.seed(2)
  # sample from 1 to k, the number of observations in the data
  train$id <- sample(1:k, nrow(train), replace = TRUE)
  list <- 1:k
  # number of training sets per fold
  n_k = table(train$id)

  # prediction and testset data frames that we
  #   add to with each iteration over the folds
  errorRates = rep(NA, k)  # one for each fold

  for (i in 1:k){
    # remove rows with id i from dataframe to create training set
    # select rows with id i to create test set
    trainingset <- subset(train, id %in% list[-i])
    testset <- subset(train, id %in% c(i))
    # summarize per fold
    yA.CV = as.numeric(as.character(apply(trainingset[trainingset$V1 == 'AaronPressman',-1], 2, sum)))
    yB.CV = as.numeric(as.character(apply(trainingset[trainingset$V1 == 'AlanCrosby',-1], 2, sum)))
    # subset testing set
    labels.CV = testset[, 1]
    features.CV = testset[, -1]
    n.test.CV = nrow(testset)
    dirichletCV.probs = rep(NA, n.test.CV)
    # predict in the testing set
    for(j in 1:n.test.CV){
      y_til.CV = as.numeric(as.character(features.CV[j, ]))
      dirichletCV.probs[j] = posterior_pA(alpha = alpha_param, yA = yA.CV, yB = yB.CV, y_til = y_til.CV
    }

    label.bin.CV <- ifelse(labels.CV == "AaronPressman", 1, 0)
    # calculate fold-level testing log loss
    errorRates[i] = log_loss(dirichletCV.probs, label.bin.CV)
  }
  # calculate overall testing log loss
  mean_error = sum(errorRates*n_k)/sum(n_k)

  return(mean_error)
}

alphas = c(seq(0.001, 0.01, by = 0.001), seq(0.05, 1.0, by = 0.05), seq(1, 5, by = 1))
res = rep(NA, length(alphas))

for (i in 1:length(alphas)) {
  res[i] = crossval_dirich(alphas[i], k = 3)
  # print(paste(i, ':', alphas[i], ':', res[i]))
}

best_alpha = which(res == min(res))
title_str = sprintf("\n3-fold cross-validation: Best alpha = %.3f with log loss %.3f", alphas[best_alpha

# Plot cross-validation log loss
ggplot() +
geom_line(aes(x=alphas,y=res)) + geom_point(aes(x=alphas,y=res)) +
labs(x="alpha" , y = "Log Loss" ,title=title_str )
```



```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Use tuned parameter on test data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

n.test.optimized = nrow(test)
dirichlet.probs.optimized = rep(NA, n.test.optimized)

for(i in 1:n.test.optimized){
  y_til = as.numeric(as.character(features[i, ]))
  dirichlet.probs.optimized[i] = posterior_pA(alpha = alphas[best_alpha][1], yA = yA, yB = yB, y_til = 
}

table(labels, dirichlet.probs.optimized > 0.5)
```

```
##
## labels          FALSE TRUE
##   AaronPressman     4   46
##   AlanCrosby       46    4
```

```
dirichlet.results.optimized <- ifelse(dirichlet.probs.optimized > 0.5, "AaronPressman", "AlanCrosby")

print(classError(dirichlet.results.optimized, labels)$errorRate)
```

```
## [1] 0.08
```

Setting α to be 1 implies a uniform distribution over all configurations in the model's support, suggesting that we are making no prior assumptions about our model parameters - if $\alpha > 1$ then the model favors values that are dense and evenly distributed, while a model with $\alpha < 1$ prefer sparse distributions. From the cross validation, the optimal value of α is 0.05. This value of α gives us a 0-1 error rate of 0.09, which suggests better performace than the error rate of 0.24 given by the Naive Bayes classifier.

Part 1c: Monte-Carlo Posterior Predictive Inference

In the above `R` function, the posterior predictive distribution was computed using a closed-form numerical expression. We can compare the analytic results to those resulting from Monte Carlo simulation.

We next provide you with an alternate `R` function (`approx_posterior_pA`) that calculates posterior predictive probabilities of authorship using Monte-Carlo simulation. The code takes the number of simulation trials as an additional input.

```
# This function is an approximation of the above exact calculation of p(A|Data):
#
# 1. Make sure to install the MCMCpack and MGLM packages to use this funciton
#
# 2. It isn't written very efficiently, notice that a new simulation from posterior
#   is drawn each time. A more efficient implementation would be to instead
#   simulate the posteriors (post_thetaA, etc.) once and hand them to
#   approx_posterior_pA to calculate the probability.

suppressMessages(library('MCMCpack'))
```

```
## Warning: package 'MCMCpack' was built under R version 3.3.2
```

```
suppressMessages(library('MGLM'))

approx_posterior_pA = function(alpha = 1, yA = NULL, yB = NULL, y_til = NULL, n.sim = NULL){
  # number of features
  K = length(yA)
  alpha0 = rep(alpha, K)
  # simulate parameters from the posterior of the Dirichlet-Multinomial model
  post_thetaA = MCmultinomdirichlet(yA, alpha0, mc = n.sim)
  post_thetaB = MCmultinomdirichlet(yB, alpha0, mc = n.sim)
  # calculate the likelihood of the observation y_til under simulated posteriors
  # note: ddirm calculates by-row likelihoods for (data, parameter) pairs
  y_til_mat = matrix(rep(y_til, n.sim), nrow = n.sim, byrow = TRUE)
  likeA = exp(ddirm(y_til_mat, post_thetaA))
  likeB = exp(ddirm(y_til_mat, post_thetaB))
  # integrate over simulated parameters
  marginal_pA = sum(likeA)
  marginal_pB = sum(likeB)
  # calculate probability of A
  pA = marginal_pA/(marginal_pA + marginal_pB)

  return(pA)
}
```

Consider the situation in Part 1b, using the Monte-Carlo approximation in `approx_posterior_pA` numbers of simulation trials (you may set α to the value chosen by cross-validation in part 1b).

Questions:

- At what point does doing more simulations not give more accuracy in terms of prediction error?
- Report on the number of simulations you need to match the test accuracy in part 1b. Does increasing the number of simulations have a noticeable effect on the model's predictive accuracy? Why or why not?

Solution

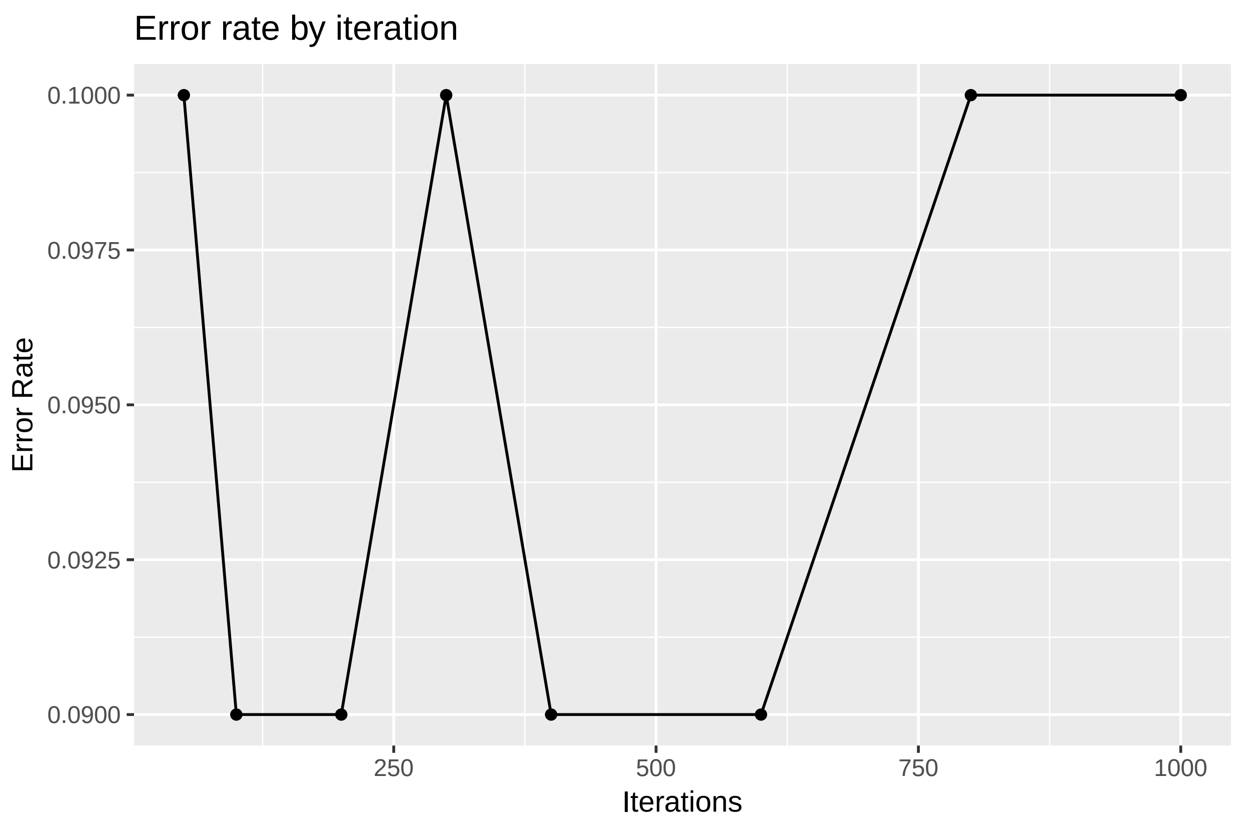
```
alpha = alphas[best_alpha]

n.test = nrow(test)
pMC = rep(NA, n.test)

for(i in 1:n.test){
  y_til = as.numeric(as.character(features[i, ]))
  pMC[i] = approx_posterior_pA(alpha = alpha, yA = yA, yB = yB, y_til = y_til, n.sim = 100)
}

pMC.results <- ifelse(pMC > 0.5, "AaronPressman", "AlanCrosby")
table(labels, pMC > 0.5)
```

```
##
## labels          FALSE TRUE
##   AaronPressman     3   47
##   AlanCrosby       43    7
```

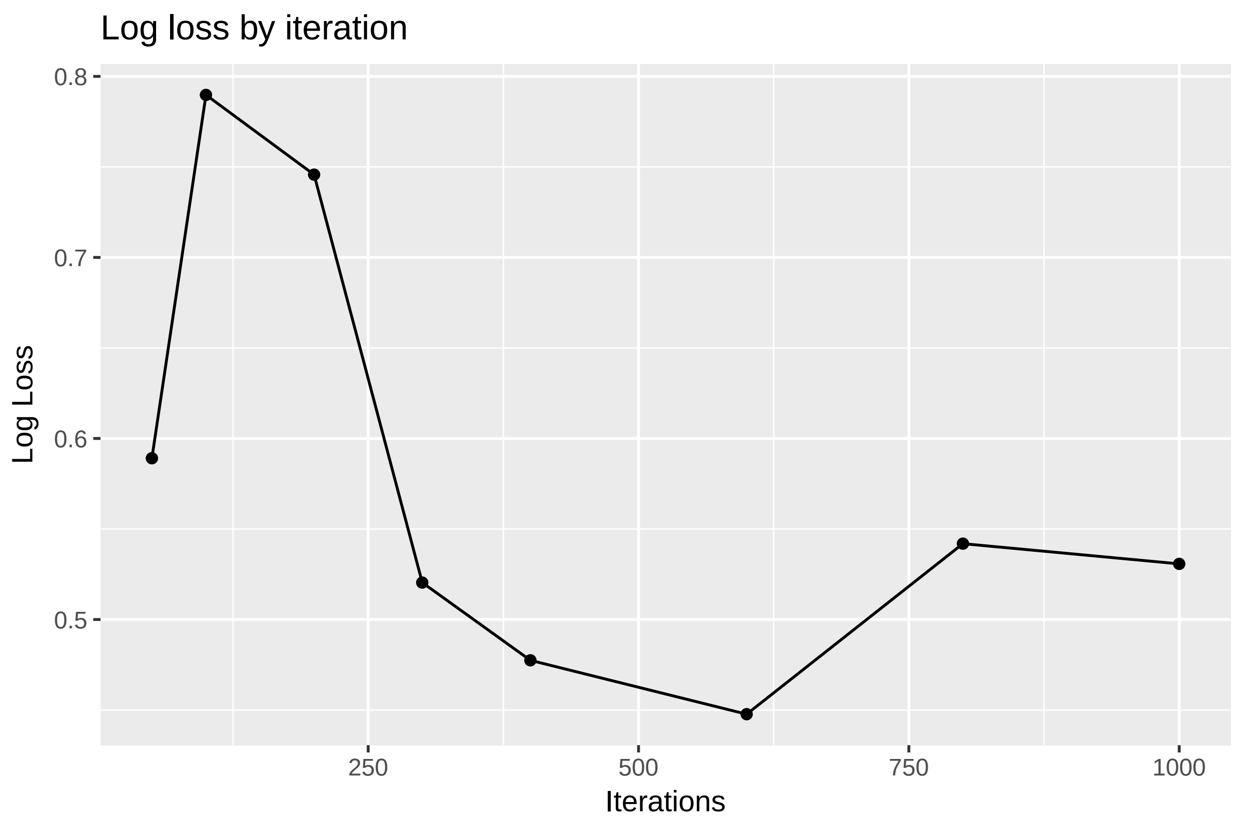
```
print(classError(pMC.results, labels)$errorRate)
```

```
## [1] 0.1
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# Tune iterations
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

set.seed(109)
sims = c(50, 100, 200, 300, 400, 600, 800, 1000)
res.zeroOne = rep(NA, length(sims))
res.logLoss = rep(NA, length(sims))

for (i in 1:length(sims)) {
  pMC.tune = rep(NA, n.test)

  for(j in 1:n.test){
    y_til.tune = as.numeric(as.character(features[j, ]))
    pMC.tune[j] = approx_posterior_pA(alpha = 0.05, yA = yA, yB = yB, y_til = y_til.tune, n.sim = sims[
  }
  pMC.results.tune <- ifelse(pMC.tune > 0.5, "AaronPressman", "AlanCrosby")

  res.zeroOne[i] = classError(pMC.results.tune, labels)$errorRate
  res.logLoss[i] = log_loss(pMC.tune, label.bin)
}

title_str = sprintf("\nError rate by iteration")

# Plot cross-validation error rate
ggplot() +
geom_line(aes(x=sims,y=res.zeroOne)) + geom_point(aes(x=sims,y=res.zeroOne)) +
labs(x="Iterations" , y = "Error Rate" ,title=title_str )
```



```
title_str2 = sprintf("\nLog loss by iteration")

# Plot cross-validation log loss
ggplot() +
geom_line(aes(x=sims,y=res.logLoss)) + geom_point(aes(x=sims,y=res.logLoss)) +
labs(x="Iterations" , y = "Log Loss" ,title=title_str2 )
```



The model first reaches the part 1b value at 450 simulations. After iteration 300, however, the log loss rates oscillate between 0.35 and 0.55, while the 0-1 error rate jumps between 0.08 and 0.11 with an average rate around 0.095, suggesting that increasing the number of simulations past 300 does not have a noticeable effect on the model's predictive accuracy. One explanation for this result is that the training and test sets are both small (100 rows each) and sparse, so there is a ceiling on how accurate the model can get, and running more simulations is not enough to improve the model's predictive power. The range in which the log loss and zero-one loss oscillate are small, and the randomness comes from the random nature of the simulations. The cases where the 0-1 error rate matches the 0-1 error rate obtained in part 1b are thus luckier trials, as the error rates do not seem to converge towards any values.

Part 1d: Author vocabulary analysis

The prescribed Bayesian model can also be used to analyze words that are most useful for inferring authorship. One way to do this is to compute or approximate the posterior distribution of the ratio of multinomial model parameters (relative ratio) for one author relative to the other, and identify the words that receive high values of this ratio. More specifically, we can calculate this ratio of the posterior parameter values

$R_k = \theta_k^A / (\theta_k^A + \theta_k^B), k = 1, ..., K$

and return a Monte-Carlo approximation of $\mathbb{E}[R_k \,|\, data]$. The largest R_k this would indicate high relative usage of a word for author A while the smaller values would indicate the same instead for author B.

We again provide you with the relevant `R` code. The input to the code is the Dirichlet parameter α, the number of MC draws `n.sim` for the approximation and the total word counts $(y_1^A, y_2^A, \ldots, y_K^A)$ and $(y_1^B, y_2^B, \ldots, y_K^B)$ from the training set articles written by author A. The output is a vector containing the approximate values of $\mathbb{E}[R_k \,|\, data]$.

```r
# This function claculates an approximation to E[R_k|data] described above.
posterior_mean_R = function(alpha = 1, yA = NULL, yB = NULL, n.sim = NULL){
  # number of features
  K = length(yA)
  alpha0 = rep(alpha, K)
  # posterior parameter values
  post_thetaA = MCmultinomdirichlet(yA, alpha0, mc = n.sim)
  post_thetaB = MCmultinomdirichlet(yB, alpha0, mc = n.sim)
  # empirical values of R_k
  R = post_thetaA/(post_thetaA + post_thetaB)
  # calculate approximation to E[R_k|data]
  ER = apply(R, 2, mean)
  return(ER)
}
```

Using the `posterior_mean_R` function and the word dictionary `words.txt`, list the top 25 words that are indicative of each author's writing style (you may set α and `n.sim` the values chosen in part 1b and 1c respectively).

Questions:

- Given the above explanation and code, how can we interpret $E[R_k]$?
- Do you find visible differences in the authors' choice of vocabulary?

Solution

```r
vals <- posterior_mean_R(alpha = alphas[best_alpha], yA = yA, yB = yB, n.sim = 500)

A.max = which(vals >= sort(vals, decreasing=T)[25], arr.ind=TRUE)
B.max = which(vals <= sort(vals, decreasing=F)[25], arr.ind=TRUE)

dict = read.csv('datasets_preprocessed_code/words_preprocessed.txt', header = FALSE)
dictVector = dict[,1]

# author A high relative usage words
print(dictVector[A.max])
```

```
##  [1] 1933            allow           business        businesses
##  [5] clinton         codes           communications  consumer
##  [9] corp            delivery        deposits        directly
## [13] disputes        hill            jim             latest
## [17] names           plus            policies        proposal
## [21] recommendations school          software        treasury
## [25] unions
## 100 Levels: 14 15 1933 1994 1996 20 598 600 760 90 advantage ... zagreb
```

```r
# author B high relative usage words
print(dictVector[B.max])
```

```
##  [1] 14             1996           banka          becker
##  [5] bourse         championship   constituencies czechs
##  [9] earnings       ferreira       henman         ing
## [13] investor       korda          minister       ods
## [17] party          seats          seed           situation
## [21] spt            team           turnout        warsaw
## [25] zagreb
## 100 Levels: 14 15 1933 1994 1996 20 598 600 760 90 advantage ... zagreb
```

The words with the 25 largest R_k values are words that author A uses frequently relative to author B, while the words with the 25 smallest R_k values are words that author B uses frequently relative to author A. As shown above, author A's frequent words are largely related to privacy and the United States government, while author B's frequent words represent technical financial analysis and foreign currencies. It is seen that the author's are focused on different subjects, and the nature of these subjects can be inferred from the R_k results.