Homework 3: Introduction to Bayesian Methods

Harvard CS 109B, Spring 2017

Danqing Wang

2/20/2017

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.

```r
# import data
train <- read.csv('./CS109b-hw3_datasets/dataset1_train_processed_subset.txt',
                  header = FALSE)
test <- read.csv('./CS109b-hw3_datasets/dataset1_test_processed_subset.txt',
                 header = FALSE)
words_given <- read.csv('./CS109b-hw3_datasets/words_preprocessed.txt',
                        header = FALSE)

# str(train)
# str(test)
# str(words_given)
```

```r
library(e1071)

# Using naive bayes to train, and predict on the test set
pred <- predict(naiveBayes(V1 ~ ., data = train), newdata = test)

# overall accuracy
accuracy <- mean(test$V1 == pred)
print(paste0('The overall accuracy is ', accuracy, '.'))
```

```
## [1] "The overall accuracy is 0.76."
```

```r
# confusion matrix, as proportions.
confmat <- function(actual, predicted) {
    addmargins(
        prop.table(
            table(actual, predicted),
            margin = 1),
        margin = 2)
}

confmat(test[['V1']], pred)
```

```
##                  predicted
## actual            AaronPressman AlanCrosby  Sum
##   AaronPressman            0.98       0.02 1.00
##   AlanCrosby               0.46       0.54 1.00
```

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.

ANSWER

- The overall accuracy is 0.76. The accuracy in predicting 'AaronPressman' right is 0.98, and in predicting 'AlanCrosby' right is 0.54.
- By using a naive bayesian approach, the method assumes the appearance of words in the list are independent of each other. This is to say, there is no connecting between word i (for some i between 1 and 101) appearing and word j (for some j between 1 and 101, j != i) appearing at the same time. However, we should note that this should not be the case in real life since every author has a set of words and phrases that he tends to use more often than others, and therefore the words in real life are not independent of one another.

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 $(y_1^A, y_2^A, \dots, y_K^A)$ 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.

```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))
    R_A = exp(A1 + A2 + A3 + A4 + A5 + A6)
    # 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))
    R_B = exp(B1 + B2 + B3 + B4 + B5 + B6)
    # probability of being class A
    ratio_BA = exp(B1 + B2 + B3 + B4 + B5 + B6 - A1 - A2 - A3 - A4 - A5 - A6)
    pA = as.numeric(1/(1 + ratio_BA))
    return(pA)
}
```

Alpha = 1

```
yA = as.numeric(apply(train[train$V1 == 'AaronPressman', -1], 2, sum))
yB = as.numeric(apply(train[train$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
```

```
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'

## The following object is masked from 'package:base':
##
##     Recall
```

```
# logloss
label.bin <- ifelse(labels == 'AaronPressman', 1, 0)
acc.logloss <- LogLoss(dirichlet.probs, label.bin)

# accuracy based on 0-1 loss
acc.overall <- mean(round(dirichlet.probs) == label.bin)
```

```r
print(paste0('For alpha = 1, the log loss is ', round(acc.logloss, 4),
             '. The overall accuracy is ', acc.overall, '.'))
```

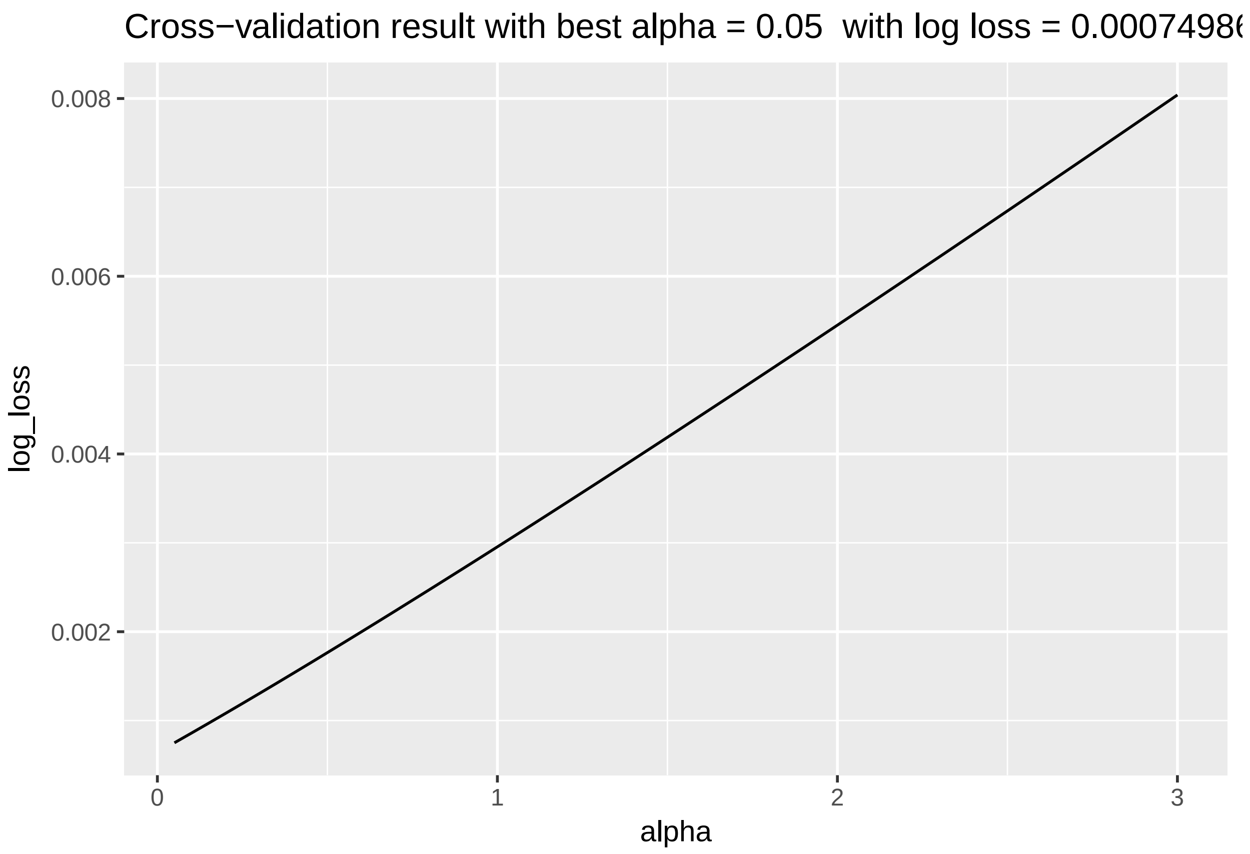
```
## [1] "For alpha = 1, the log loss is 0.4079. The overall accuracy is 0.88."
```

Cross-validation

```r
# Function for k-fold cross-validation
crossval = function(train, param_val, k) {
  set.seed(109)
  num_param = length(param_val)
  folds = sample(1:k, nrow(train), replace = TRUE)
  loss_score = rep(0., num_param)

  for(i in 1:num_param){
    for(j in 1:k){
      train_ = train[folds!=j, ]
      test_ = train[folds==j, ]

      yA = as.numeric(apply(train_[train_$V1 == 'AaronPressman', -1], 2, sum))
      yB = as.numeric(apply(train_[train_$V1 == 'AlanCrosby', -1], 2, sum))

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

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

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

      label.bin <- ifelse(labels == 'AaronPressman', 1, 0)
      loss_score[i] = loss_score[i] + LogLoss(dirichlet.probs, label.bin)
    }

    # Average across k folds
    loss_score[i] = loss_score[i] / k
  }

  # Return cross-validated R^2 values
  return(loss_score)
}
```

```r
alpha <- seq(0.05, 3, by = 0.05)

log_loss_score <- crossval(train, alpha, k = 5)
cross_val_logloss <- data.frame(alpha = alpha, log_loss = log_loss_score)

alpha.best <- alpha[which.min(log_loss_score)]
```

```r
logloss.best <- min(log_loss_score)

library(ggplot2)
ggplot(cross_val_logloss, aes(x = alpha, y = log_loss))+
  geom_line()+
  ggtitle(paste('Cross-validation result with best alpha =',
                alpha.best, ' with log loss =', logloss.best))
```



The best alpha is 0.05 with logloss of 0.00075.

Predicting on the test set:

```r
# Predicting on the test set
yA = as.numeric(apply(train[train$V1 == 'AaronPressman', -1], 2, sum))
yB = as.numeric(apply(train[train$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 = alpha.best, yA = yA, yB=yB, y_til = y_til)
}
```

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

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

```r
# calculate the logloss
label.bin <- ifelse(labels == 'AaronPressman', 1, 0)
acc.cv.logloss <- LogLoss(dirichlet.probs, label.bin)

# accuracy based on 0-1 loss
acc.cv.overall <- mean(round(dirichlet.probs) == label.bin)

print(paste0('For alpha = 0.05, the log loss is ',
             round(acc.cv.logloss, 4),
             '. The overall accuracy is ', acc.cv.overall, '.'))
```

```
## [1] "For alpha = 0.05, the log loss is 0.4771. The overall accuracy is 0.92."
```

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?

ANSWER

- By using $\alpha = 1$, we are placing equal weight on all data. When alpha is large, it is our beief that the data is more dense, this means we have a greater number of non-zero values of wordcount in our data set, this means for each article, out of the list of words given, the author uses many of these words; whereas when alpha is small, it is our belief that the data is sparse, and we have very few non-zero values in the wordcount data matrix, this means for each article, out of the list of words given, the author only uses a few of them. Here, by setting alpha to 1, we are not giving any assumption about how sparse we think the wordcount data is. Alpha = 1 is the uninformative prior.

- The optimal value for α is 0.05. The final log-loss predicting error for the alpha = 0.05 is 0.00075. However, we see a monotonic increase of log-loss here in the cross validation, and it seems we are picking out the best alpha to use (minimum log loss value) arbituarily purely based on what is the smallest value of alpha we used during the cross-validation. This is because the training set is extremely small and sparse, by doing a 5-fold cross validation, we do not have enough data points. Normally if we have a larger data set, we would expect a U-shaped log loss curve and we find the best alpha to use corresponding to the minimum of the U-shaped curve.

- The overall accuracy based on 0-1 loss on the test data set is 0.92, which is higher than the 0.76 overall accuracy on from Naive Bayes model, and also higher than the 0.88 overall accuracy of the model when alpha = 1. One thing to note here is that, if we compare the model of alpha = 0.05 vs alpha = 1, we notice that the overall accuracy is higher at 0.92 for the former and lower at 0.88 for the latter. However, the log loss of 0.4771 is higher for the former compared to 0.406 for the latter. Intuitively, we would think that a higher overall accuracy corresponds with a lower log loss, however this is not necessarily the case as we can see from here. This is because the log loss value is calculated from the probability, which would give a higher value if the two classes are not at large separation distances (i.e. the probabilities for the two classes are near 0.5, rather than polarized near 0 and 1). In this case, we might get a higher accuracy rate which is purely judged by 1s and 0s, but also a higher log loss since the probabilities are not very separable.

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.

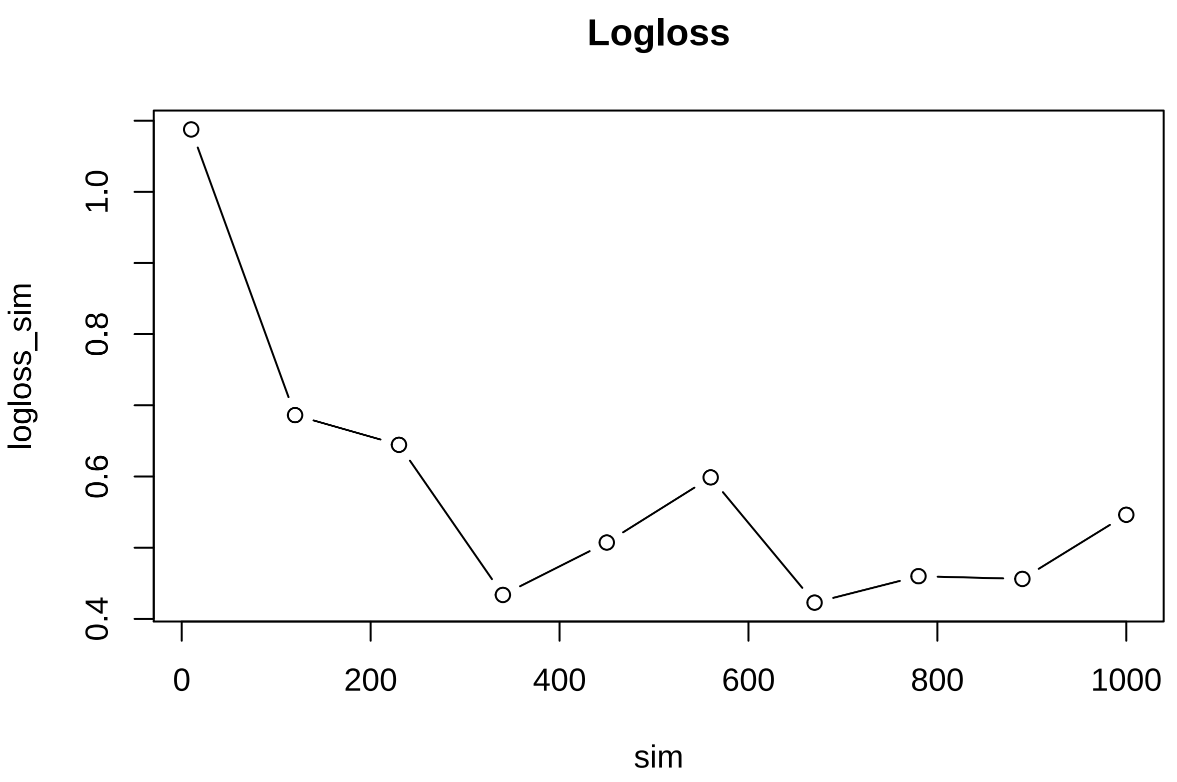
```r
# This function is an approximation of the above exact calculation of p(A|Data):
#
# 1. Make sure to install the MCMCpack and MGLM packages to use this funciton
#
# 2. It isn't written very efficiently, notice that a new simulation from posterior
#    is drawn each time. A more efficient implementation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


library('MCMCpack')
```

```
## Loading required package: coda

## Loading required package: MASS

## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)

## ## Copyright (C) 2003-2017 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park

## ##
## ## Support provided by the U.S. National Science Foundation

## ## (Grants SES-0350646 and SES-0350613)
## ##
```

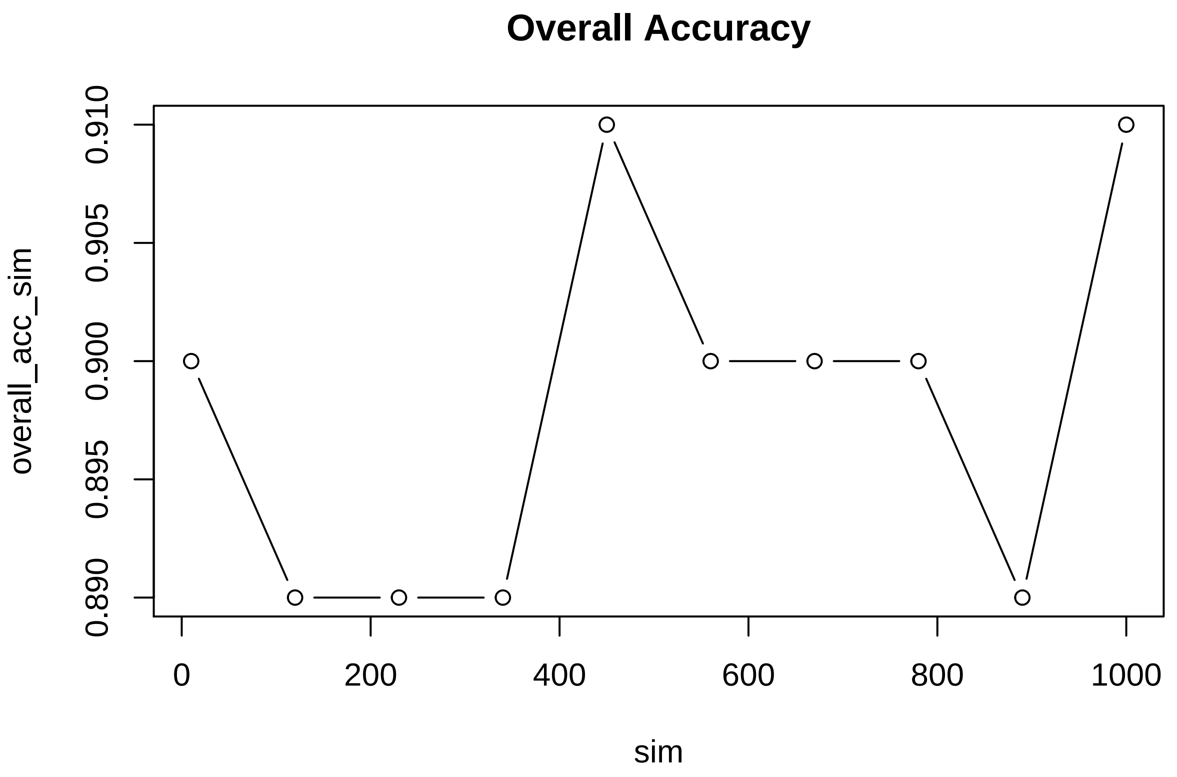
```r
library('MGLM')


approx_posterior_pA = function(alpha = NULL, 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).

```r
yA = as.numeric(apply(train[train$V1 == 'AaronPressman', -1], 2, sum))
yB = as.numeric(apply(train[train$V1 == 'AlanCrosby', -1], 2, sum))

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

n.test.optimised = nrow(test)
dirichelt.probs.optimised = rep(NA, n.test.optimised)

sim <- as.integer(seq(10, 1000, length.out = 10))
logloss_sim = rep(NA, length(sim))
overall_acc_sim = rep(NA, length(sim))

for(i in 1:length(sim)){
  for(j in 1:n.test.optimised){
    ytill.tune = as.numeric(as.character(features[j,]))
    dirichelt.probs.optimised[j] = approx_posterior_pA(alpha = alpha.best, yA = yA,
                                                       yB = yB, y_til = ytill.tune,
                                                       n.sim = sim[i])
  }

  label.bin <- ifelse(labels == 'AaronPressman', 1, 0)
  logloss_sim[i] = LogLoss(dirichelt.probs.optimised, label.bin)
  overall_acc_sim[i] = mean(round(dirichelt.probs.optimised) == label.bin)
}


plot(sim, logloss_sim, 'b', main = 'Logloss')
```



```r
plot(sim, overall_acc_sim, 'b', main = 'Overall Accuracy')
```



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?

ANSWER

- The data set is too small and sparse to have number of simulations to have effect. Normally with a larger dataset, we would expect a convergence at some point. From the graph, it looks like above simulation number of 450, we can get a fairly high overall accuracy and a low log loss.
- We need about 450 simulations to match the test accuracy of 0.92 in part 1b. Increasing the numbeer of simulations does not help here as we see the accuracy jumps around. This is because the data set is too small and sparse to have number of simulations to have effect.

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)
}
```

```r
yA = as.numeric(apply(train[train$V1 == 'AaronPressman', -1], 2, sum))
yB = as.numeric(apply(train[train$V1 == 'AlanCrosby', -1], 2, sum))
alpha = 0.05
n.sim = 500

ek <- posterior_mean_R(alpha, yA, yB, n.sim)

combined <- data.frame(phrase = words_given$V1,
           ek = ek)

A <-combined$phrase[order(ek, decreasing = TRUE)][1:25] # AaronPressman
```

```
B <- combined$phrase[order(ek, decreasing = FALSE)][1:25] # AlanCrosby

# print top 25 words used by Aaron Pressman
print(A)
```

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

```
# print top 25 words used by Alan Crosby
print(B)
```

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

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?

ANSWER

- $E[R_k]$ is the expectation value of the ratio $\theta_k^A/(\theta_k^A + \theta_k^B)$. θ_k^A indicates how likely a particular word k is associated with author A (Arron Pressman), the higher the value of θ_k^A, the more likely that the word is associated with author A. Here, the $E[R_k]$ is calculating the expectation value of the word k, the higher the value of $E[R_k]$, the more likely that word k is associated with author A, the lower the value, the more likely that word k is associated with author B.
- It seems like Aaron Pressman writes about American politics while Alan Crosby writes about European finance. Among the words Aaron Pressman uses, 'clinton', 'unions', and 'policies' suggest an American political focus, while Alan Crosby frequently uses words like 'warsaw', 'czechs', 'zagreb' which are places in Europe, and he uses words such as 'investor', 'ods' etc. that are related to finance.

Problem 2: Bayesian Logisitic Regression

You are provided with data sets `dataset_2_train.txt` and `dataset_2_test.txt` containing details of contraceptive usage by 1934 Bangladeshi women. There are 4 attributes for each woman, along with a label indicating if she uses contraceptives. The attributes include:

- `district`: identifying code for the district the woman lives in
- `urban`: type of region of residence

- `living.children`: number of living children
- `age_mean`: age of women (in years, centred around mean)

The women are grouped into 60 districts. The task is to build a classification model that can predict if a given woman uses contraceptives.

Use simple visualizations to check if there are differences in contraceptive usage among women across the districts.

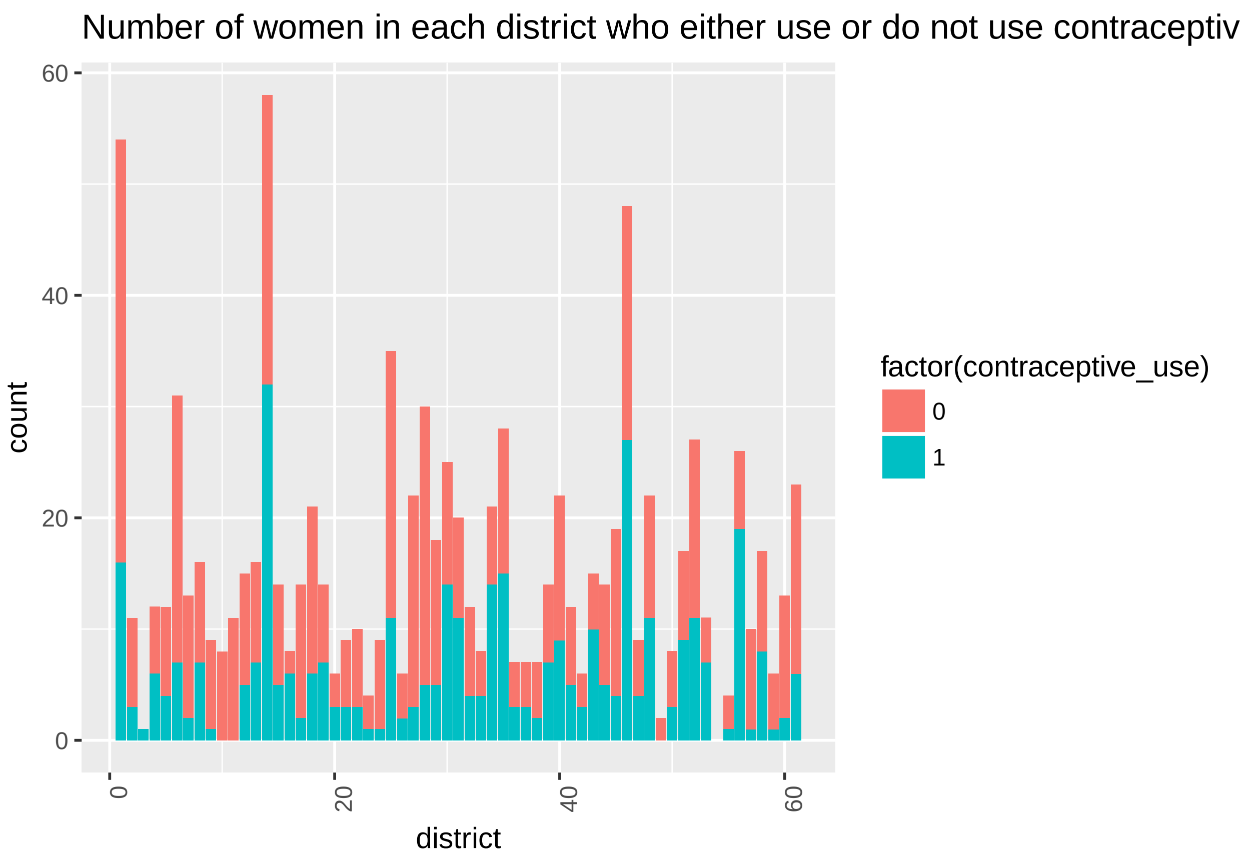
```r
train <- read.csv('./CS109b-hw3_datasets/dataset_2_train.txt')
test <- read.csv('./CS109b-hw3_datasets/dataset_2_test.txt')

str(train)
```

```
## 'data.frame':    967 obs. of  5 variables:
##  $ district         : int  35 22 29 5 34 30 10 5 44 58 ...
##  $ urban            : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ living.children  : int  4 2 2 3 4 4 1 4 4 2 ...
##  $ age_mean         : num  2.44 -1.56 -8.56 -4.56 8.44 ...
##  $ contraceptive_use: int  0 1 1 1 0 1 0 0 0 0 ...
```

```r
summary(train)
```

```
##     district         urban        living.children    age_mean
##  Min.   : 1.00   Min.   :0.0000   Min.   :1.000   Min.   :-13.5600
##  1st Qu.:14.00   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.: -7.5600
##  Median :29.00   Median :0.0000   Median :3.000   Median : -1.5599
##  Mean   :29.68   Mean   :0.3071   Mean   :2.592   Mean   : -0.3106
##  3rd Qu.:46.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:  6.4400
##  Max.   :61.00   Max.   :1.0000   Max.   :4.000   Max.   : 19.4400
##  contraceptive_use
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3899
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

```r
library(ggplot2)

ggplot(train, aes(x = district, fill = factor(contraceptive_use)))+
  geom_bar()+
  ggtitle("Number of women in each district who either use or do not use contraceptives")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



ANSWER

Yes, from the plot, we can see that there are differences in contraceptive usage among women across the districts. For some districts, all of the women use contraceptives, while for some others, there is a significantly larger proportion of women who use contraceptives than those who do not, and for some districts there is a significantly smaller proportion of women who use contraceptives than those who do not. We also notice that we do not have any data for district #54.

If so, would we benefit by fitting a separate classification model for each district? To answer this question, you may fit the following classification models to the training set and compare their accuracy on the test set:

(1) A separate logistic regression model for each district

```r
# split data according to their districts
y_train <- split(train, train$district)
y_test <- split(test, test$district)

# list of districts
districts <- seq(1, length(y_train))

# initilization
pred <- list() # predicted response according to district
actual <- list() # actual response according to district

for(i in districts){
  # fit glm on each district i
  fit_dist <-  glm(contraceptive_use ~ urban + living.children + age_mean,
                        data = y_train[[i]],
                        family = binomial(link = "logit"))

  # predict on test set of the same district i
  pred[[i]] <- round(predict(fit_dist, newdata = y_test[[i]], type = "response"))

  # record value of actual response
  actual[[i]] <- y_test[[i]]$contraceptive_use
}

# confusion matrix
confmat(unlist(actual), unlist(pred))
```

```
##       predicted
## actual         0         1       Sum
##      0 0.6803419 0.3196581 1.0000000
##      1 0.4947644 0.5052356 1.0000000
```

```
# overall accuracy
mean(unlist(actual) == unlist(pred))
```

```
## [1] 0.6111686
```

ANSWER

The overall accuracy is 0.61. Here, the model is predicting 0 with an accuracy of 0.68 and 1 with an accuracy of 0.51.

(2) A single logistic regression model using the entire training set (ignoring the district information)

Simple `glm` model

```
# simple glm model
fit0 <- glm(contraceptive_use ~ . - district, data = train,
            family = binomial(link = "logit"))

confmat <- function(actual, predicted) {
  addmargins(
    prop.table(
      table(actual, predicted),
      margin = 1),
    margin = 2)
}

confmat(test$contraceptive_use,
        factor(predict(fit0,
                       newdata = test,
                       type = "response") > .5,
              labels = c("0", "1")))
```

```
##       predicted
## actual         0         1       Sum
##      0 0.9025641 0.0974359 1.0000000
##      1 0.7434555 0.2565445 1.0000000
```

```
# overall accuracy
mean(test$contraceptive_use == factor(predict(fit0,
                                              newdata = test,
                                              type = "response") > .5,
                                      labels = c("0", "1")))
```

```
## [1] 0.647363
```

ANSWER

The overall accuracy is 0.65. From the confusion matrix, this model is predicting contraceptive_use = 0 with a high accuracy of 0.90, but contraceptive_use = 1 with a low accuracy of 0.26.

It looks like by separating the data into 60 districts and fitting a glm model on each of them (model 1), we are getting a lower overall accuracy of 0.61 compared to fitting a model without considering the different districts (model 2) which gives us an accuracy of 0.65. Comparing the confusion matrices, we notice that the prediction accuracy for contraceptive_use = 0 is higher for model 2 at 0.90 compared to 0.68 in model 1, but this is at the expense of a very low prediction accuracy of 0.26 for contraceptive_use = 1 in model 2 compared to the higher 0.51 in model 1. Model 2 is favoring predicting a NO (i.e. 0) for contraceptive use compared to model 1. This may mean that model 2 is biased towards predicting NO (i.e. 0). In model 1, where we take districts into consideration, the model is giving a fairly similar accuracy of 0.68 and 0.51 in predicting NO and YES. The model is less biased towards predicting a certain response. Unfortunately, it does not perform significantly better than a 50-50 blind chance guess. The reason for this is that we have a really small dataset, there are only a few data points in each district, therefore separating our data into different districts is not a great idea in this case.