Homework 3: Introduction to Bayesian Methods

Harvard CS 109B, Spring 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.

Load the data:

```
dictionary_words = read.csv("./CS109b-hw3_datasets/words_preprocessed.txt",
    header = FALSE)
author_train = read.csv("./CS109b-hw3_datasets/dataset1_train_processed_subset.txt",
    header = FALSE)
author_test = read.csv("./CS109b-hw3_datasets/dataset1_test_processed_subset.txt",
    header = FALSE)
```

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.

```
# install.packages('e1071') install.packages('Brobdingnag')
library(Brobdingnag)
library(e1071)

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

accuracy = function(actual, predicted) {
    length(which(actual == predicted))/length(predicted) * 100
}
```

```
#----Naive Bayes Model--------#
naive_bayes_test_predictions = predict(naiveBayes(V1 ~ ., data = author_train),
    newdata = author_test)
# test_predictions = ifelse(test_predictions==
# 'AaronPressman', 1, 0)

# confusion matrix
confmat(author_test$V1, naive_bayes_test_predictions)
```

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

```
# overall accuracy
cat("\noverall accuracy ", accuracy(author_test$V1, naive_bayes_test_predictions),
    "%")
```

```
##
## overall accuracy  76 %
```

Using a 0-1 loss the overall accuracy is 76%. There is a higher error rate for 'Alan Crosby' articles than for 'Aaron Pressman', with 46% of Alan Crosby articles being incorrectly classified.

The problem with using a Naive Bayes classifier is the conditional independence assumption - it assumes that the probability of a word occuring in an article by 'Aaron Pressman' is not influcened by the presence of any other words **if** we already know that the article is written by Aaron Pressman. In this example, some words tend to travel together and are indicative of a particular author so the Naive Bayes model may not be suitable.

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 $(y_1^B, y_2^B, \ldots, y_K^B)$ denote the total counts of the K dictionary words across the

articles by author B. We assume the following *multinomial model*:

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

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

The model parameters $(\theta_1^A, \ldots, \theta_K^A)$ and $(\theta_1^B, \ldots, \theta_K^B)$ 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 $(y_1^A, y_2^A, \ldots, y_K^A)$ from all articles by author A in the training set,
- the total word counts $(y_1^B, y_2^B, \ldots, y_K^B)$ from all articles by author B in the training set, and
- the word counts $(\tilde{y}_1, \ldots, \tilde{y}_K)$ for a new test article.

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

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

library(Brobdingnag)
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(as.brob(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(as.brob(B1 + B2 + B3 + B4 + B5 + B6)))
```

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

#------- HELPER FUNCTION -------------#

class_probability_train_test = function(train, test, class, alpha) {
    y <- lapply(split(train, class), function(x) {
        apply(x, 2, sum)
    })
    yA <- y[[1]]
    yB <- y[[2]]
    pA <- apply(test, 1, function(x) {
        posterior_pA(alpha = alpha, yA = yA, yB = yB, y_til = x)
    })
    pA
}
```

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

Evaluate predictive accuracy with α is set to 1:

```
# a function to calculate the posterior predictive
# probability for a data frame of articles

y <- lapply(split(author_train[-1], author_train[1]), function(x) {
    apply(x, 2, sum)
})
yA <- y[[1]]
yB <- y[[2]]

cat("\n Training set")
```

```
##
##  Training set
```

```r
# ----- Predict on the training set ------------ #
pA_train <- class_probability_train_test(author_train[-1], author_train[-1],
    author_train[1], 1)

dirichlet_train_predictions = factor(pA_train < 0.5, labels = c("AaronPressman",
    "AlanCrosby"))

confmat(author_train$V1, dirichlet_train_predictions)
```

```
##                predicted
## actual          AaronPressman AlanCrosby Sum
##   AaronPressman             1          0   1
##   AlanCrosby                0          1   1
```

```r
cat("\n Testing set")
```

```
##
##  Testing set
```

```r
# ----- Predict on the testing set ------------ #
pA_test <- class_probability_train_test(author_train[-1], author_test[-1],
    author_train[1], 1)

dirichlet_test_predictions = factor(pA_test < 0.5, labels = c("AaronPressman",
    "AlanCrosby"))

confmat(author_test$V1, dirichlet_test_predictions)
```

```
##                predicted
## actual          AaronPressman AlanCrosby  Sum
##   AaronPressman          0.92       0.08 1.00
##   AlanCrosby             0.16       0.84 1.00
```

```r
# overall accuracy
cat("\noverall accuracy on testing set ", accuracy(author_test$V1,
    dirichlet_test_predictions), "%")
```

```
##
## overall accuracy on testing set  88 %
```

```r
pred_compare = data.frame(bayes = naive_bayes_test_predictions,
    dirichelt = dirichlet_test_predictions, actual = author_test$V1)
```

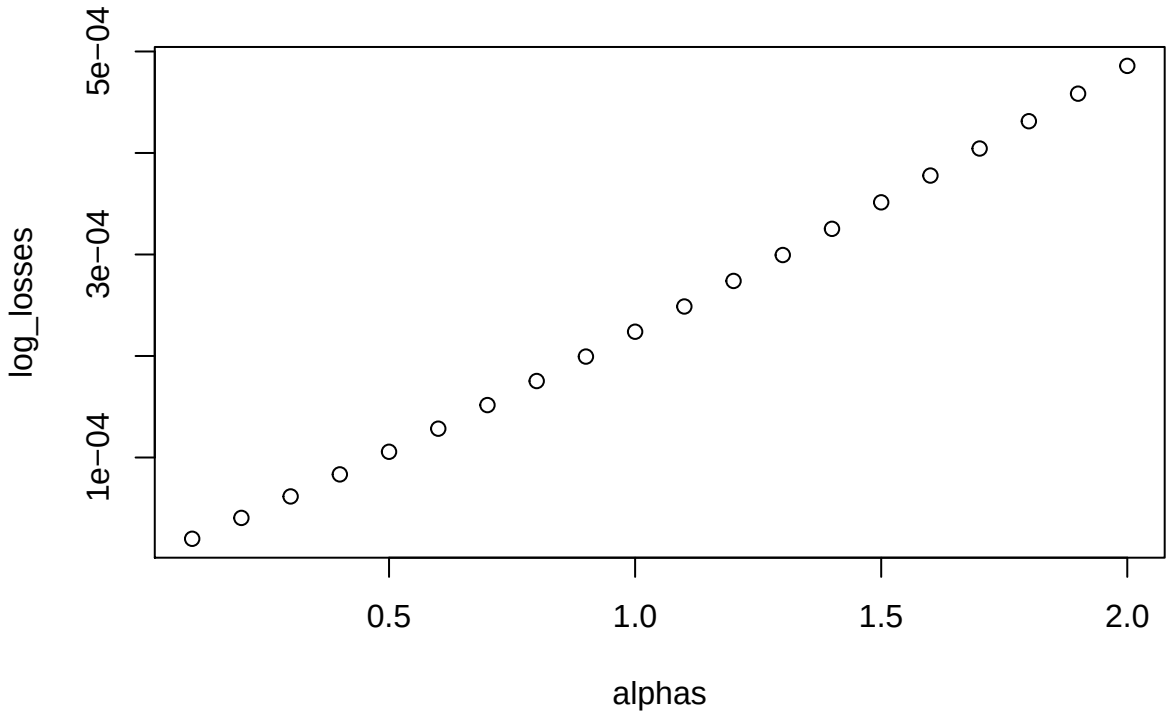
```r
# ---- comaprison of predictions from both models --- #
head(pred_compare, 10)
```

```
##            bayes     dirichelt        actual
## 1     AlanCrosby AaronPressman AaronPressman
## 2  AaronPressman    AlanCrosby AaronPressman
## 3  AaronPressman AaronPressman AaronPressman
## 4  AaronPressman AaronPressman AaronPressman
## 5  AaronPressman    AlanCrosby AaronPressman
## 6  AaronPressman AaronPressman AaronPressman
## 7  AaronPressman    AlanCrosby AaronPressman
## 8  AaronPressman AaronPressman AaronPressman
## 9  AaronPressman AaronPressman AaronPressman
## 10 AaronPressman AaronPressman AaronPressman
```

Comparing the two models, the overall accuracy of the dirichlet model is 11% higher than for the naive bayes model.

```
# ----------------- Cross validation ---------------- #

MultiLogLoss <- function(act, pred) {
    eps <- 1e-15
    pred <- pmin(pmax(pred, eps), 1 - eps)
    sum(act * log(pred) + (1 - act) * log(1 - pred)) * -1/NROW(act)
}

alphas <- seq(0.1, 2, by = 0.1)

## vectors to store results
ac <- c()

# Function for k-fold cross-validation to tune span parameter
# in loess
crossval = function(train, param_val, k = 3) {
    # Input: Training data frame: 'train', Vector of span
    # parameter values: 'param_val', Number of CV folds: 'k'
    # Output: Vector of R^2 values for the provided parameters:
    # 'cv_rsq'

    num_param = length(param_val)  # Number of parameters
    set.seed(109)  # Set seed for random number generator

    # Divide training set into k folds by sampling uniformly at
    # random folds[s] has the fold index for train instance 's'
    folds = sample(1:k, nrow(train), replace = TRUE)

    log_loss = rep(0, num_param)  # Store cross-validated R^2 for different parameter values

    # Iterate over parameter values
    for (i in 1:num_param) {
        # Iterate over folds to compute R^2 for parameter
        for (j in 1:k) {

            # 4 folds for training
            subset_train = train[folds != j, ]
            # 1 fold for testing
            subset_test = train[folds == j, ]

            # Fit model on all folds other than 'j' with parameter value
            # param_val[i]
            actual_train <- sapply(subset_train$V1, as.numeric)  # asnumeric changes factors to 1, 2, 3
            actual_train <- replace(actual_train, actual_train ==
                2, 0)  # changes factors from 2 to 0
            pA_train = class_probability_train_test(subset_train[-1],
                subset_train[-1], subset_train$V1, alpha = param_val[i])

            # Compute R^2 for predicted values
            log_loss[i] = MultiLogLoss(actual_train, pA_train)
        }
```

```
        # Average R^2 across k folds
        log_loss[i] = log_loss[i]/k
    }

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

log_losses = crossval(author_train, alphas, 5)
plot(alphas, log_losses)
```




```
best_alpha = alphas[which.min(log_losses)]
min_loss = min(log_losses)

pA_test = class_probability_train_test(author_train[-1], author_test[-1],
    author_test[1], alpha = best_alpha)
optimal_dirichlet_test_predictions = factor(pA_test < 0.5, labels = c("AaronPressman",
    "AlanCrosby"))
confmat(author_test$V1, optimal_dirichlet_test_predictions)
```

```
##                predicted
## actual          AaronPressman AlanCrosby  Sum
##   AaronPressman          0.92       0.08 1.00
##   AlanCrosby             0.08       0.92 1.00
```

```
cat("\noverall accuracy: ", accuracy(author_test$V1, optimal_dirichlet_test_predictions),
    "%")
```

```
##
## overall accuracy:  92 %
```

```
cat("\nmin log loss: ", min_loss)
```

```
##
## min log loss:  1.992334e-05
```

```
cat("\nbest alpha: ", best_alpha)
```

```
##
## best alpha:  0.1
```

Questions:

- What does setting $\alpha = 1$ imply?

$\alpha = 1$ implies a uniform prior distribution for all the words in the dictionary.

- Do the above optimization. What is the optimal value of α? What is your final log-loss prediction error?

The optimal value of alpha is 0.1 resulting in a log loss of 0.000199.

Cross Validation results in a linear plot of log loss vs alpga. There are 100 training datapoints and 100 variables (and hence 99 parameters in the Dirichlet-Multinomial model), therefore with 5-fold cross validation we would have only 80 testing data points per fit and the model is bound to overfit resulting in a very low log loss.

- For the optimal value of α, how do the accuracies (based on $0-1$ loss) obtained compare with the previous Naive Bayes classifier?

The accuracies with the dirichlet multinomial model (with an optimal alpha) are much higher than with Naive Bayes (97% compared to 76%). Instead of treating the absence/presence of each word in the dictionary as a Bernoulli, the dirichlet multinomial model allows us to take the counts of the words into consideration, where each of the words has a fixed probability of being selected.

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.

library("MCMCpack")
library("MGLM")

approx_posterior_pA = function(alpha = 1, yA = NULL, yB = NULL,
    y_til = NULL, n.sim = NULL) {
    # number of features
```

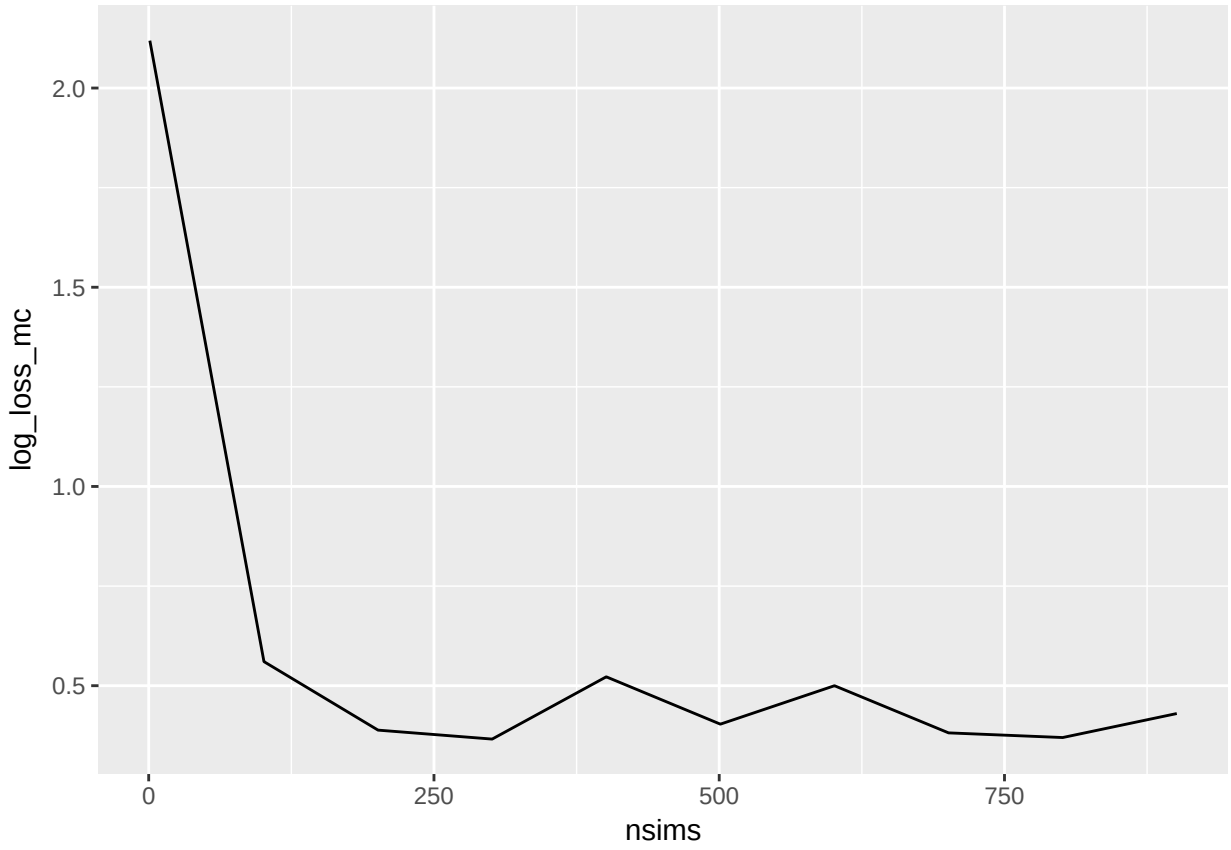
```
    K = length(yA)
    alpha0 = rep(alpha, K)
    # simulate parameters from the posterior of the
    # Dirichlet-Multinomial model
    post_thetaA = MCmultinomdirichlet(yA, alpha0, mc = n.sim)
    post_thetaB = MCmultinomdirichlet(yB, alpha0, mc = n.sim)
    # calculate the likelihood of the observation 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).

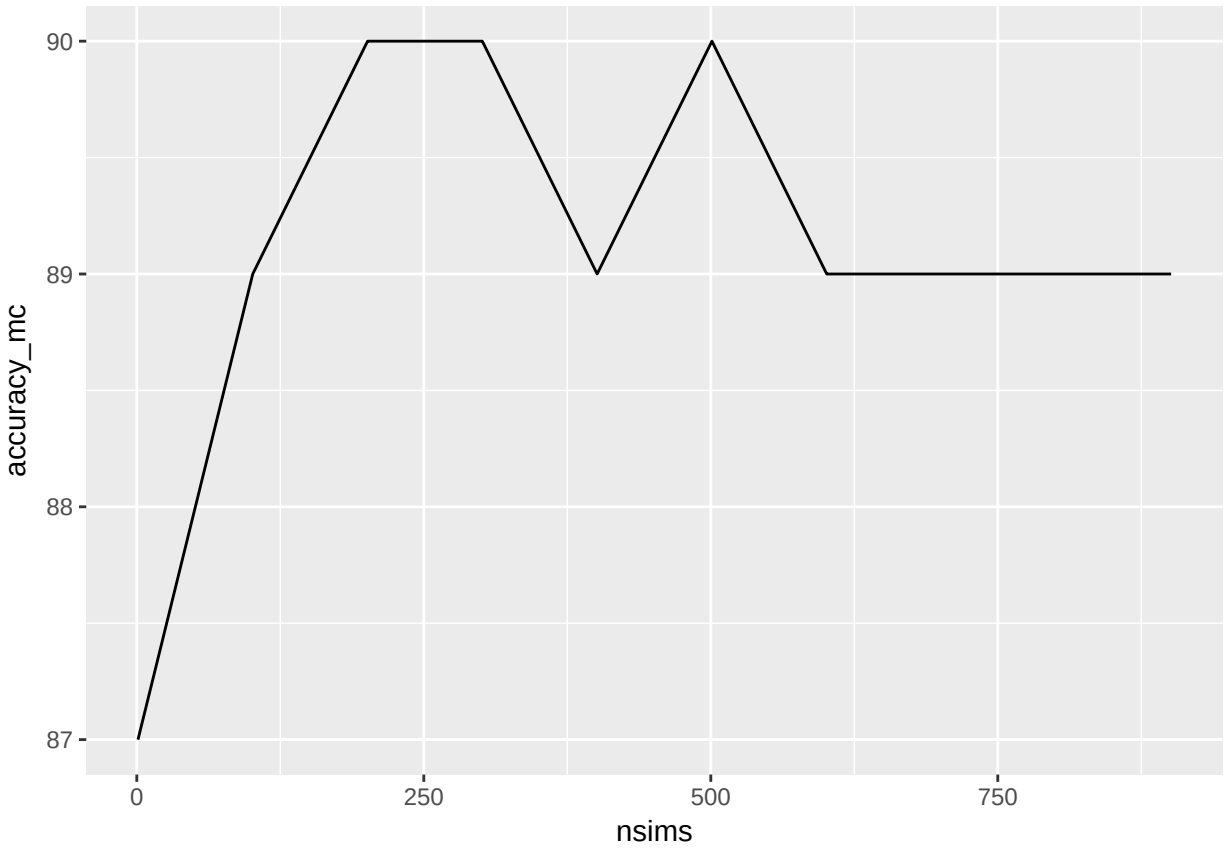
```
# Function to calculate class probabilitues using MC

class_probability_train_test_mc = function(train, test, class,
    alpha, nsims) {
    y <- lapply(split(train, class), function(x) {
        apply(x, 2, sum)
    })
    yA <- y[[1]]
    yB <- y[[2]]
    pA <- apply(test, 1, function(x) {
        approx_posterior_pA(alpha = alpha, yA = yA, yB = yB,
            y_til = x, n.sim = nsims)
    })
    pA
}

# Experiment with different numbers of simulations

nsims <- seq(1, 1000, by = 100)

## vectors to store results
log_loss_mc <- c()
accuracy_mc <- c()

## iterate using a for-loop
for (n in nsims) {

    pa1 <- class_probability_train_test_mc(author_train[-1],
        author_test[-1], author_train[[1]], alpha = best_alpha,
        nsims = n)
```

```
    # from stack overflow
    MultiLogLoss <- function(act, pred) {
        eps <- 1e-15
        pred <- pmin(pmax(pred, eps), 1 - eps)
        sum(act * log(pred) + (1 - act) * log(1 - pred)) * -1/NROW(act)
    }

    actual_test <- as.numeric(author_test$V1)
    actual_test <- replace(actual_test, actual_test == 2, 0)

    predicted_vals = ifelse(factor(pa1 < 0.5, labels = c("AaronPressman",
        "AlanCrosby")) == "AaronPressman", 1, 0)

    log_loss_mc = append(log_loss_mc, MultiLogLoss(actual_test,
        pa1))
    accuracy_mc = append(accuracy_mc, accuracy(actual_test, predicted_vals))
}
```

```
library("ggplot2")
ggplot(NULL, mapping = aes(x = nsims, y = log_loss_mc)) + geom_line()
```




```
ggplot(NULL, mapping = aes(x = nsims, y = accuracy_mc)) + geom_line()
```



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?

After approximately 700 iterations, there is little further reduction in the log loss. The accuracy fluctuates randomly between 89% and 90% for number of simulations greater than approx 300.

Therefore from eyeballing the graphs of accuracy and log loss vs the number of simulations, I would suggest that after approximately 500 simulations, doing further simulations does not give more accuracy in terms of prediction error.

Increasing the number of simulations does not have a noticeable effect on the prediction accuracy. Given the the nature and size of the dataset (100 observations and 100 features) and the small magnitude of the oscillations in the log loss and 0-1 loss, the number of simulations will not cause the accuracy to decrease but can also not cause it to increase further than in part (b).

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 $\mathbb{E}[R_k \,|\, data]$.

```
# 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).

```
y <- lapply(split(author_train[-1], author_train[1]), function(x) {
    apply(x, 2, sum)
})
yA <- y[[1]]
yB <- y[[2]]

ER <- posterior_mean_R(alpha = 0.1, yA, yB, n.sim = 10)
```

```
words_probs <- data.frame(ER, dictionary_words)
words_probs_author_A = words_probs[order(-ER), ]
words_probs_author_B = words_probs[order(ER), ]
```

Top 25 words for Author A (largest 25 values of E[R])

```
words_probs_author_A$V1[1:25]
```

```
##  [1] unions         software       communications directly
##  [5] businesses     policies       hill           school
##  [9] clinton        names          corp           codes
## [13] consumer       proposal       latest         disputes
## [17] component      division       deposits       treasury
## [21] business       sharp          jim            critical
## [25] 1933
## 100 Levels: \t14 15 1933 1994 1996 20 598 600 760 90 advantage ... zagreb
```

Top 25 words for Author B (smallest 25 values of E[R])

```
words_probs_author_B$V1[1:25]
```

```
##  [1] ods            turnout        team           banka
##  [5] constituencies bourse         seed           1996
##  [9] seats          earnings       warsaw         party
## [13] sidelines      minister       investor       spt
## [17] championship   becker         henman         situation
## [21] \t14           western        ferreira       760
## [25] cheap
## 100 Levels: \t14 15 1933 1994 1996 20 598 600 760 90 advantage ... zagreb
```

Questions:

- Given the above explanation and code, how can we interpret $E[R_k]$?

$E[R_k]$ can be interpreted as the signature words of each author. The words which have high values of $E[R_k]$ can be thougt of as words which may be representative of Author A's style of writing or the topics they write about.

- Do you find visible differences in the authors' choice of vocabulary?

Comparing the top 25 words for author A and author B, Author A's words are related to politics, featuring words such as 'clinton', 'unions', 'policies'. For Author B, the words appear to be related to europe, with words in portugese, and names of european cities such as warsaw and zagreb.

Problem 2: Bayesian Logisitic Regression

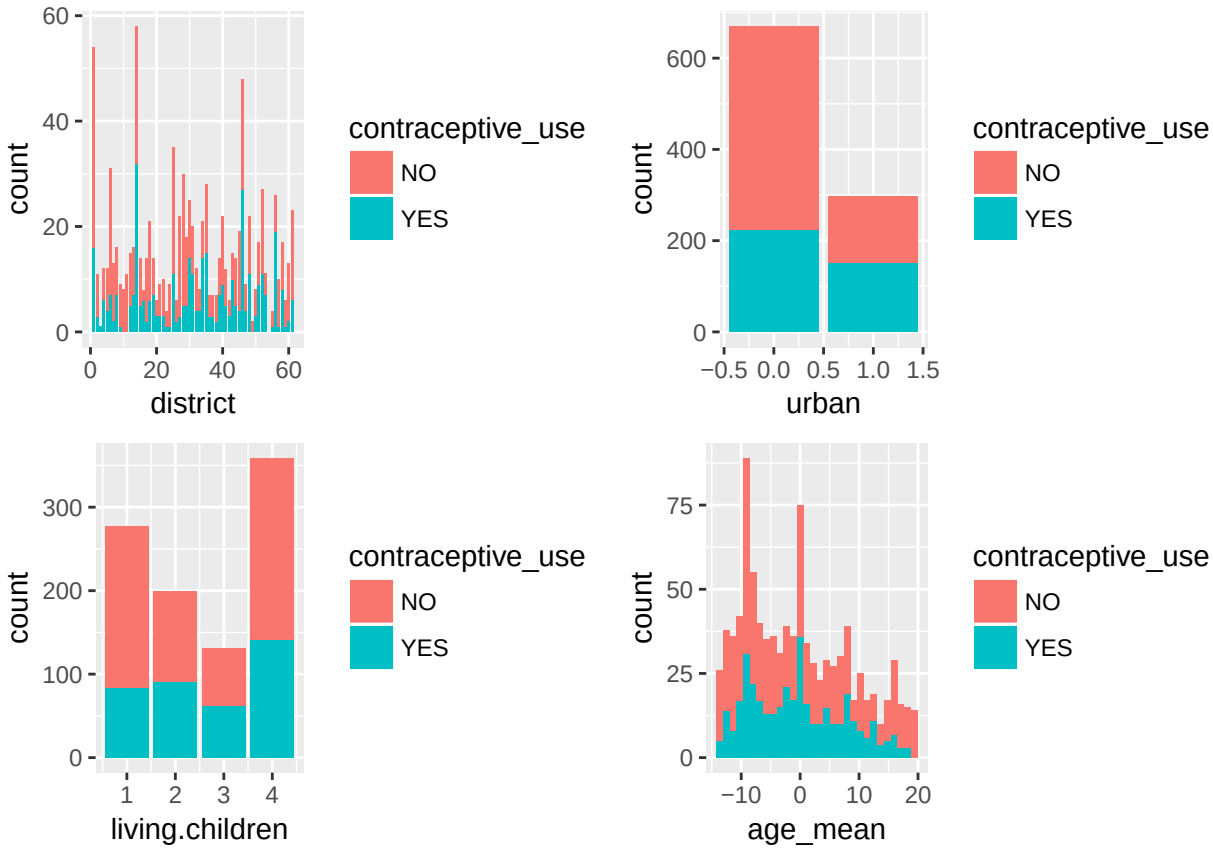
You are provided with data sets `dataset_2_train.txt` and `dataset_2_test.txt` containing details of contraceptive usage by 1934 Bangladeshi women. There are 4 attributes for each woman, along with a label indicating if she uses contraceptives. The attributes include:

- `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.

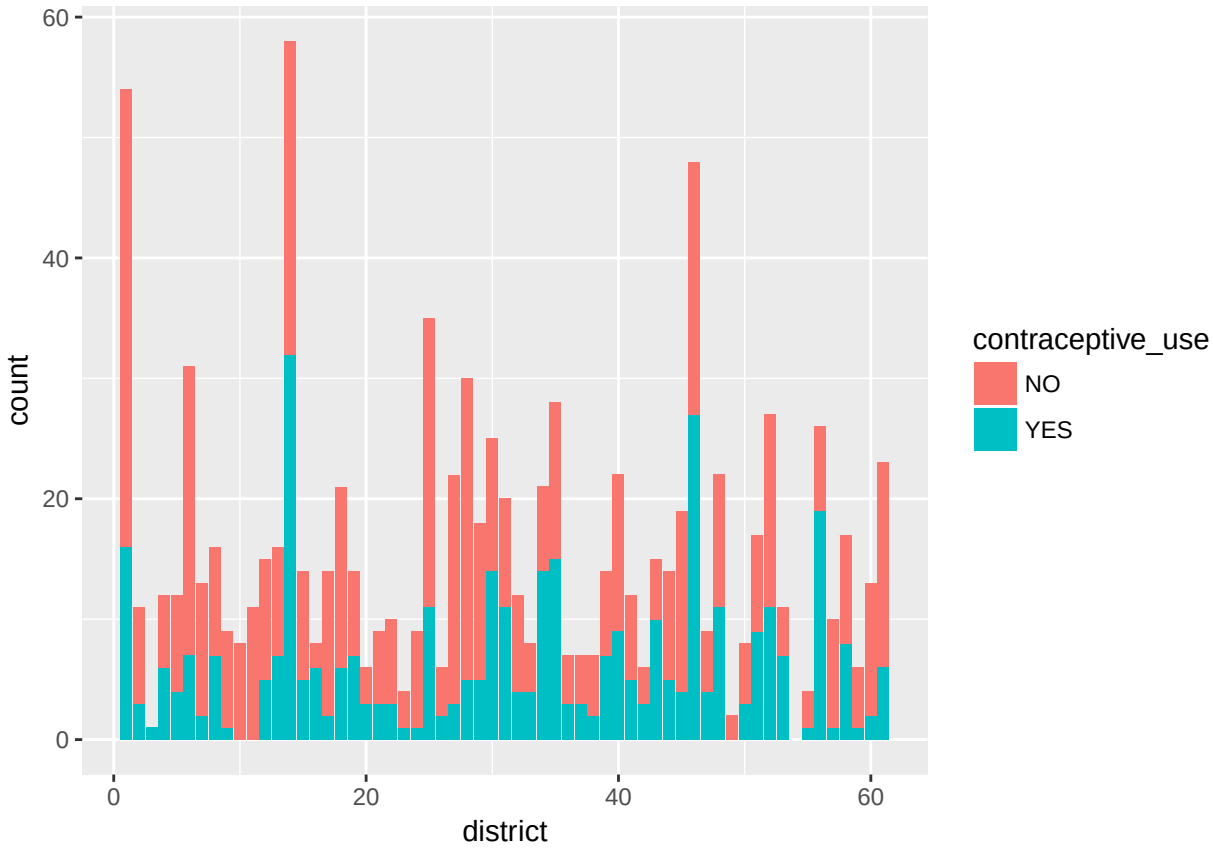
```
data_2_train = read.csv("./CS109b-hw3_datasets/dataset_2_train.txt",
    header = TRUE)
data_2_test = read.csv("./CS109b-hw3_datasets/dataset_2_test.txt",
    header = TRUE)
```

```
library(ggplot2)
require(gridExtra)
data_2_train_plot = data_2_train
data_2_train_plot$contraceptive_use = ifelse(data_2_train$contraceptive_use ==
    1, "YES", "NO")
p1 = ggplot(data_2_train_plot, aes(x = district, fill = contraceptive_use)) +
    geom_bar()
p2 = ggplot(data_2_train_plot, aes(x = urban, fill = contraceptive_use)) +
    geom_bar()
p3 = ggplot(data_2_train_plot, aes(x = living.children, fill = contraceptive_use)) +
    geom_bar()
p4 = ggplot(data_2_train_plot, aes(x = age_mean, fill = contraceptive_use)) +
    geom_histogram()
grid.arrange(p1, p2, p3, p4, nrow = 2, ncol = 2)
```



Closer look at the plot of count vs district

```
ggplot(data_2_train_plot, aes(x = district, fill = contraceptive_use)) +
    geom_bar()
```



There is higher usage of contraceptives amongst women in certain districts, and the proportion of women who do/do not use contraceptives differs due to the differen population sizes. As a percentage of the population, there is greater contraceptive usage in urban rather than rural districts. In rural districts (the majority of the women in the data) do not use contraception.

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

- A separate logistic regression model for each district
- A single logistic regression model using the entire training set (ignoring the district information)

Give an explanation for what you observe.

```
#------- SINGLE LOGISTIC REGRESSION MODEL FOR THE ENTIRE TRAINING SET (ignoring the district predictor)

fit0 <- glm(contraceptive_use ~ age_mean + urban + living.children,
    data = data_2_train, family = binomial(link = "logit"))

confmat(data_2_train$contraceptive_use, factor(predict(fit0,
    newdata = data_2_test, type = "response") > 0.5, labels = c("0",
    "1")))
```

```
##       predicted
## actual         0         1       Sum
##      0 0.8593220 0.1406780 1.0000000
##      1 0.8090186 0.1909814 1.0000000
```

```r
cat("\noverall accuracy on the test set", accuracy(data_2_train$contraceptive_use,
    factor(predict(fit0, newdata = data_2_test, type = "response") >
        0.5, labels = c("0", "1"))), "%")
```

```
##
## overall accuracy on the test set 59.8759 %
```

```r
#------ SEPARATE LOGISITIC REGRESSION MODELS FOR EACH DISTRICT--------

# 1. subset data in training to district `s` 2. fit logistic
# regression to subset in (1) 3. subset data in testing to
# district `s` 4. use model in (2) to predict for data subset
# in (3) -- you can use the `predict.glm` function

districts = unique(data_2_train$district)

district_accuracies = data.frame(district = numeric(), accuracy = numeric())

test_predictions = list()

total_correct_predictions = 0

for (d in districts) {

    subset_districts_train <- data_2_train[data_2_train$district ==
        d, ]
    subset_districts_test <- data_2_test[data_2_test$district ==
        d, ]

    fit_subset <- glm(contraceptive_use ~ ., data = subset_districts_train,
        family = binomial(link = "logit"))

    district_predict = round(predict.glm(fit_subset, newdata = subset_districts_test,
        type = "response"))

    district_accuracy = accuracy(subset_districts_test$contraceptive_use,
        district_predict)

    total_correct_predictions = total_correct_predictions + length(which(subset_districts_test$contracep
        district_predict))

    district_row = data.frame(district = d, accuracy = district_accuracy)
    district_accuracies = rbind(district_accuracies, district_row)
}
# predictions on the scale of the response variable
```

```r
head(district_accuracies)
```

```
##   district accuracy
## 1       35 60.00000
## 2       22 70.00000
## 3       29 71.42857
## 4        5 62.96296
## 5       34 71.42857
## 6       30 61.11111
```

```
cat("accuracy from a district specific model: ", total_correct_predictions/nrow(data_2_test) *
    100, "%")
```

```
## accuracy from a district specific model:  61.11686 %
```

Due to the large variation in contraceptive usage between the different regions, the accuracy of a district specific model is higher than a glm fit to the entire data set.

Fiting a separate glm to each district acknowledges that the relationship between contraceptive usage and the other predictors may differ within each district. However, there is likely to be very little data on which to estimate effects. By pool all the data together and fiting a single glm ignoring the district predictor, the model can make full use of the entire data although failing to recognize differences in effects within groups.

A hierarchical bayesian logistic model can make use of the data from all districts while also recognising district specific effects.