Homework 3

Ihsaan Patel

February 18, 2017

Libraries

```
library(e1071)
library(ggplot2)
library(MCMCpack)
library(MGLM)
library(plyr)
```

Problem 1: Authorship Attribution

```
# Load preprocessed words to label columns
words_preprocessed <- scan("words_preprocessed.txt", what = character(),sep = "")
dataset1_column_names <- c("author", words_preprocessed)

# Load training and testing data, passing the preprocessed words as column names
dataset1_train <- read.table("dataset1_train_processed_subset.txt", header = FALSE, sep = ",", col.names
dataset1_test <- read.table("dataset1_test_processed_subset.txt",header = FALSE, sep = ",", col.names =
```

Part 1a: Naive Bayes Classifier

```
# fit and predict naive bayes models
model.nb <- naiveBayes(author ~ ., data = dataset1_train)
pred.nb <- predict(model.nb, dataset1_test, type = "class")

# Get Naive Bayes Model Accuracy
table.nb <- table(pred.nb, dataset1_test$author)
accuracy.nb <- sum(diag(table.nb))/ sum(table.nb)
print(sprintf("Naive Bayes Model Accuracy: %.2f", accuracy.nb))
```

```
## [1] "Naive Bayes Model Accuracy: 0.76"
```

Naive Bayes assumes that the occurrence of one word does not depend on the occurrence of another word, however this is clearly not the case for written pieces. For example, the use of the word 'currency' in a piece increases the chance that 'deposit' will occur as well. Naive Bayes doesn't account for that which makes it a problem for this task.

Part 1b: Dirichlet-Multinomial Model

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #
# 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
    pA = R_A/(R_A + R_B)
    return(pA)
}
```

```
# Function to log-loss of DMM
log_loss <- function(author_identity, posterior_probability){
  # Input:
  #   Vector of binomial values corresponding to author 'A': 'author_identity',
  #   Vector of posterior probabilities corresponding to author_identity: 'posterior_probability',
  # Output:
  #   Log-loss probability of the model

  n <- length(author_identity)
  log_loss_probability <- 0
  for (i in 1:n) { #loop through each author, adding the log of posterior probability or its inverse
    if (author_identity[i] == 1) { # check if the author's identity corresponds to 'A'
      log_loss_probability <- log_loss_probability + log(posterior_probability[i])
    } else {
      log_loss_probability <- log_loss_probability + log(1 - posterior_probability[i])
    }
  }
  return(-1./n * log_loss_probability)
}
```

```
# Function to get the posterior probability from a DMM
dmm_model <- function(alpha, train_a, train_b, test_set) {
  # Input:
  #   Float of the alpha parameter for a DMM: 'alpha',
```

```
  #   Vector with sum of key word frequencies for author 'A': 'train_a',
  #   Vector with sum of key word frequencies for author 'B': 'train_b',
  #   Data Frame of article key words for unknown author: 'test_set',
  # Output:
  #   Posterior probability for the unknown author: 'posterior_prob'

  n <- nrow(test_set)
  posterior_prob = rep(0., n)
  for (i in 1:n) { # loop through each record in the test_set and predict posterior probaility
    posterior_prob[i] <- posterior_pA(alpha, train_a, train_b, test_set[i, ])
  }
  return(posterior_prob)
}
```

```
# Function to compute k-fold cross-validation accuracy for a given classification model
crossval_dmm <- function(data, param_val, k) {
  # Input:
  #   Training data frame: 'data',
  #   Vector of span parameter values: 'param_val',
  #   Number of CV folds: 'k'
  # Output:
  #   Vector of Log-Loss values for the provided parameters: 'cv_log_loss'

  # Split dataset into author A and B
  train_A <- data[data$author == "AaronPressman", ]
  train_B <- data[data$author != "AaronPressman", ]

  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_A), replace = TRUE)

  cv_log_loss = rep(0., num_param) # Store cross-validated log-loss for different parameter values

  # Iterate over parameter values
  for(i in 1:num_param){
    # Iterate over folds to compute log-loss for parameter
    for(j in 1:k){
      # Choose portions of author sets to train model on
      train_A_total <- colSums(train_A[folds!=j, ][-1])
      train_B_total <- colSums(train_B[folds!=j, ][-1])

      # Combine remaining portions into a test set
      val_set <- rbind(train_A[folds == j, ], train_B[folds == j, ])

      # Calculate the posterior probability for the test set
      posterior_prob <- dmm_model(param_val[i], train_A_total, train_B_total, val_set[-1])
      val_identity <- as.numeric(val_set$author == "AaronPressman")


      # Compute log-loss for predicted values
      cv_log_loss[i] = cv_log_loss[i] + log_loss(val_identity, posterior_prob)
```

```
    }

    # Average log-loss across k folds
    cv_log_loss[i] = cv_log_loss[i] / k
  }

  # Return cross-validated log-loss values
  return(cv_log_loss)
}
```

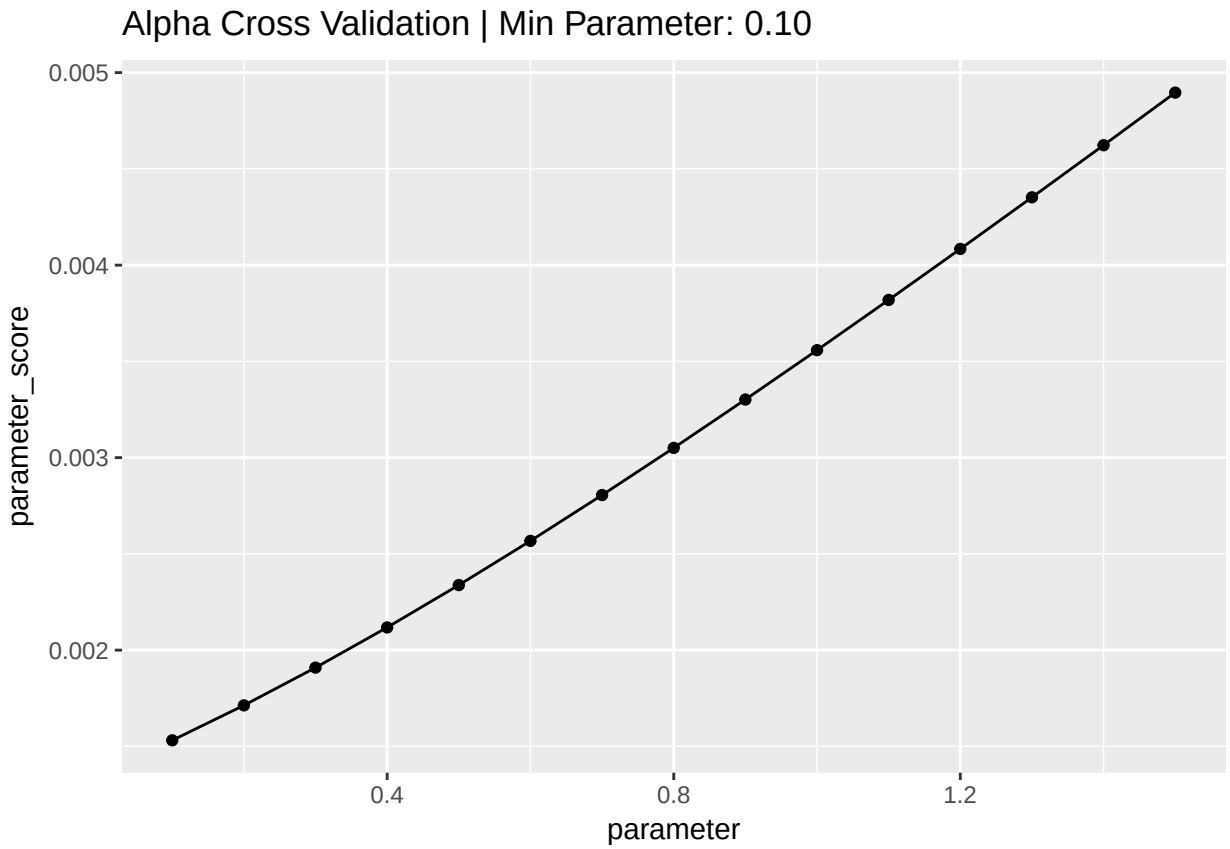
```
# Format training and testing data for DMM
author_A <- colSums(dataset1_train[dataset1_train$author == "AaronPressman", ][-1])
author_B <- colSums(dataset1_train[dataset1_train$author != "AaronPressman", ][-1])
author_til <- dataset1_test[-1]
```

```
# Calculate posterior probability and log loss for DMM w/ alpha = 1
posterior_prob_a1 <- dmm_model(1, author_A, author_B, author_til)
author_identity_test_a1 <- as.numeric(dataset1_test$author == "AaronPressman")
print(sprintf("Alpha = 1; Log-Loss Prediction Error: %.4f", log_loss(author_identity_test_a1, posterior
```

```
## [1] "Alpha = 1; Log-Loss Prediction Error: 0.4079"
```

An Alpha of 1 assumes a uniform distribution across all the keywords, implying that the probability of finding a particular keyword in an article is equal to the probability of finding any other keyword.

```
# Tune the alpha parameter through cross validation for the dmm
alphas <- c(1.:15.)/10.
dmm_cv_scores <- crossval_dmm(dataset1_train, alphas, 5)
alpha_min_parameter <- alphas[which.min(dmm_cv_scores)]
ggplot(data = data.frame(parameter = alphas, parameter_score = dmm_cv_scores), aes(x = parameter, y = pa
```




```
# Calculate and print the log loss of the optimal alpha
print(sprintf("Optimal Value of Alpha: %.2f", alpha_min_parameter))
```

```
## [1] "Optimal Value of Alpha: 0.10"
```

```
posterior_prob_amin <- dmm_model(alpha_min_parameter, author_A, author_B, author_til)
author_identity_test_amin <- as.numeric(dataset1_test$author == "AaronPressman")
print(sprintf("Log-Loss Prediction Error: %.4f", log_loss(author_identity_test_amin, posterior_prob_amin
```

```
## [1] "Log-Loss Prediction Error: 0.4652"
```

```
# Calculate accuracy of the tuned dmm
pred.dmm <- (posterior_prob_amin>.5)
table.dmm <- table(pred.dmm, author_identity_test_amin)
accuracy.dmm <- sum(diag(table.dmm))/ sum(table.dmm)
print(sprintf("Tuned DMM Accuracy: %.2f", accuracy.dmm))
```

```
## [1] "Tuned DMM Accuracy: 0.92"
```

The tuned DMM performs much better than the Naive Bayes model.

Part 1c: Monte-Carlo Posterior Predictive Inference

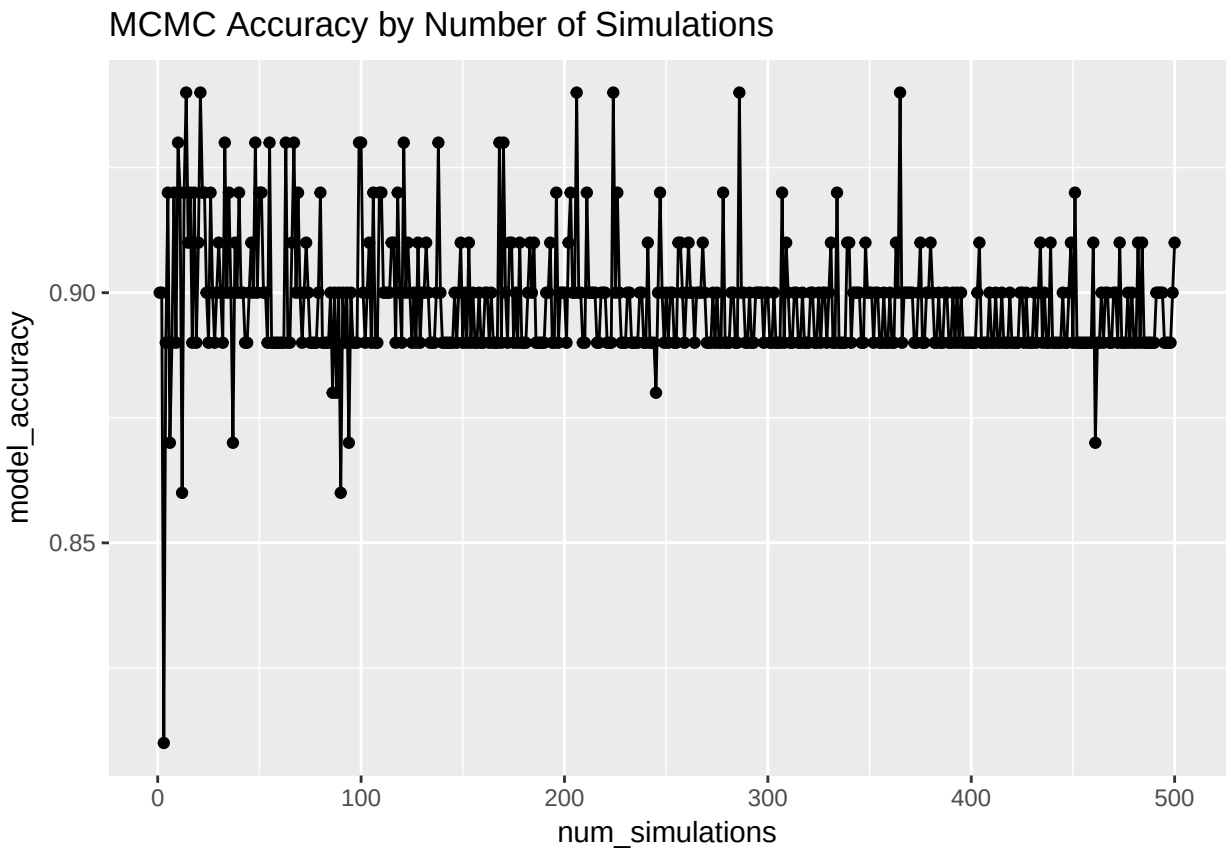
```
# This function is an approximation of the above exact calculation of p(A|Data):
#
# 1. Make sure to install the MCMCpack and MGLM packages to use this funciton
#
```

```
# 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.

approx_posterior_pA = function(post_thetaA = NULL, post_thetaB = NULL, y_til = NULL, n.sim = NULL){
    # 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)
    y_til_mat <- matrix(as.numeric(y_til_mat), n.sim, length(y_til))
    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)
}
```

```
# Function to evaluate the performance of an MCMC model based on the number of trials
eval_mcmc <- function(alpha_parameter, author_A, author_B, author_til, author_identity, param_val) {
  # Input:
  #   Value of alpha for Dirchilet model: 'alpha_parameter',
  #   Vector of keyword counts for author 'A': 'author_A',
  #   Vector of keyword counts for author 'B': 'author_B',
  #   Array of keyword counts of unknown authors: 'author_til',
  #   Vector of binomial values corresponding to author 'A': 'author_identity',
  #   List of number of simulations to run: 'param_val'
  # Output:
  #   Array of number of simulations and corresponding accuracy values: 'acc'

  # number of features
  K = length(author_A)
    alpha0 = rep(alpha_parameter, K)

  n_params <- length(param_val)
  n_records <- nrow(author_til)
  acc = rep(0., n_params)
  for (i in 1:n_params) { # loop through each parameter value and calculate posterior probability of ea
    posterior_prob = rep(0., n_records)

    # simulate parameters from the posterior of the Dirichlet-Multinomial model
      post_thetaA = MCmultinomdirichlet(author_A, alpha0, mc = param_val[i])
      post_thetaB = MCmultinomdirichlet(author_B, alpha0, mc = param_val[i])

    for (j in 1:n_records) { # loop through each record in the unknown author set
      posterior_prob[j] <- approx_posterior_pA(post_thetaA = post_thetaA, post_thetaB = post_thetaB, y_
    }
    table.mcmc <- table((posterior_prob > .5), author_identity)
    acc[i] <- sum(diag(table.mcmc)) / sum(table.mcmc)
  }
  return(acc)
```

```
}
```

```
# Test MCMC model accuracy on differ numbers of simulations
simulations <- c(1:500)
mcmc_acc <- eval_mcmc(alpha_min_parameter, author_A, author_B, author_til, author_identity_test_amin, si
ggplot(data = data.frame(num_simulations = simulations, model_accuracy = mcmc_acc), aes(x = num_simulati
```



The number of simulations does not give more accuracy after 10 simulations, where the model accuracy then fluctuates between a band of .89 and .90. This matches the test accuracy in the DMM. After 10 simulations, increasing the number of simulations does not appear to have a significant effect on model accuracy, presumably because the MCMC model has found the appropriate posterior distribution to sample from, and so the remaining inaccurate predictions are outliers.

Part 1d: Author vocabulary analysis

```
# 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)
}
```

```
# create dataframe with words and variable importance
simulations <- 10 # chosed based on problem 1c
vocab_importance <- data.frame(words = words_preprocessed, importance = posterior_mean_R(alpha = alpha_

# Print important vocab for Aaron Pressman
vocab_importance[with(vocab_importance, order(-importance)), ][1:25, ]
```

```
##                 words importance
## pi.29        consumer  0.9998643
## pi.60           names  0.9998463
## pi.74        proposal  0.9993026
## pi.25  communications  0.9983685
## pi.32        critical  0.9981713
## pi.26       component  0.9976063
## pi.19      businesses  0.9970386
## pi.75 recommendations  0.9969474
## pi.86        software  0.9966781
## pi.31            corp  0.9956383
## pi.77          school  0.9955936
## pi.70        policies  0.9949843
## pi.23         clinton  0.9935766
## pi.3             1933  0.9932945
## pi.54             jim  0.9911502
## pi.94          unions  0.9908013
## pi.48            hill  0.9898483
## pi.42        disputes  0.9880566
## pi.37        delivery  0.9838166
## pi.41        directly  0.9829266
## pi.38        deposits  0.9813186
## pi.91        treasury  0.9685116
## pi.93           union  0.9626859
## pi.69            plus  0.9610373
## pi.12           allow  0.9577900
```

```
# Print important vocab for Alan Crosby
vocab_importance[with(vocab_importance, order(importance)), ][1:25, ]
```

```
##               words   importance
## pi.46      ferreira 0.0004872594
## pi.87           spt 0.0009639566
## pi.1             14 0.0012128376
## pi.45      earnings 0.0013676703
## pi.63           ods 0.0014184216
## pi.21  championship 0.0015159101
## pi.16        bourse 0.0018365976
## pi.55         korda 0.0028961082
## pi.14         banka 0.0028962604
## pi.92       turnout 0.0029238393
## pi.35        czechs 0.0032413465
## pi.5           1996 0.0038585479
```

```
## pi.78         seats 0.0042700883
## pi.15        becker 0.0048728328
## pi.100       zagreb 0.0049530230
## pi.50           ing 0.0049918327
## pi.89          team 0.0052020497
## pi.52      investor 0.0053993560
## pi.59      minister 0.0066752602
## pi.99        wouldn 0.0071158906
## pi.79          seed 0.0082623135
## pi.47        henman 0.0084159167
## pi.84     situation 0.0084746562
## pi.95        warsaw 0.0097947612
## pi.67         party 0.0100640154
```

We can interpret $E[R_k]$ as the relative importance of a word for Author A compared to Author B, i.e. a word that is widely used by Author A but not Author B. The higher the value, the more widely used the word is by Author A and unused by Author B and vice-versa. It looks like Aaron Pressman uses words associated with the business world significantly more, including "unions", "businesses", "software", and "corp". Alan Crosby on the other hand uses words more associated with politics, particularly in Eastern Europe, including "turnout", "seats", "constituencies", and "minister".

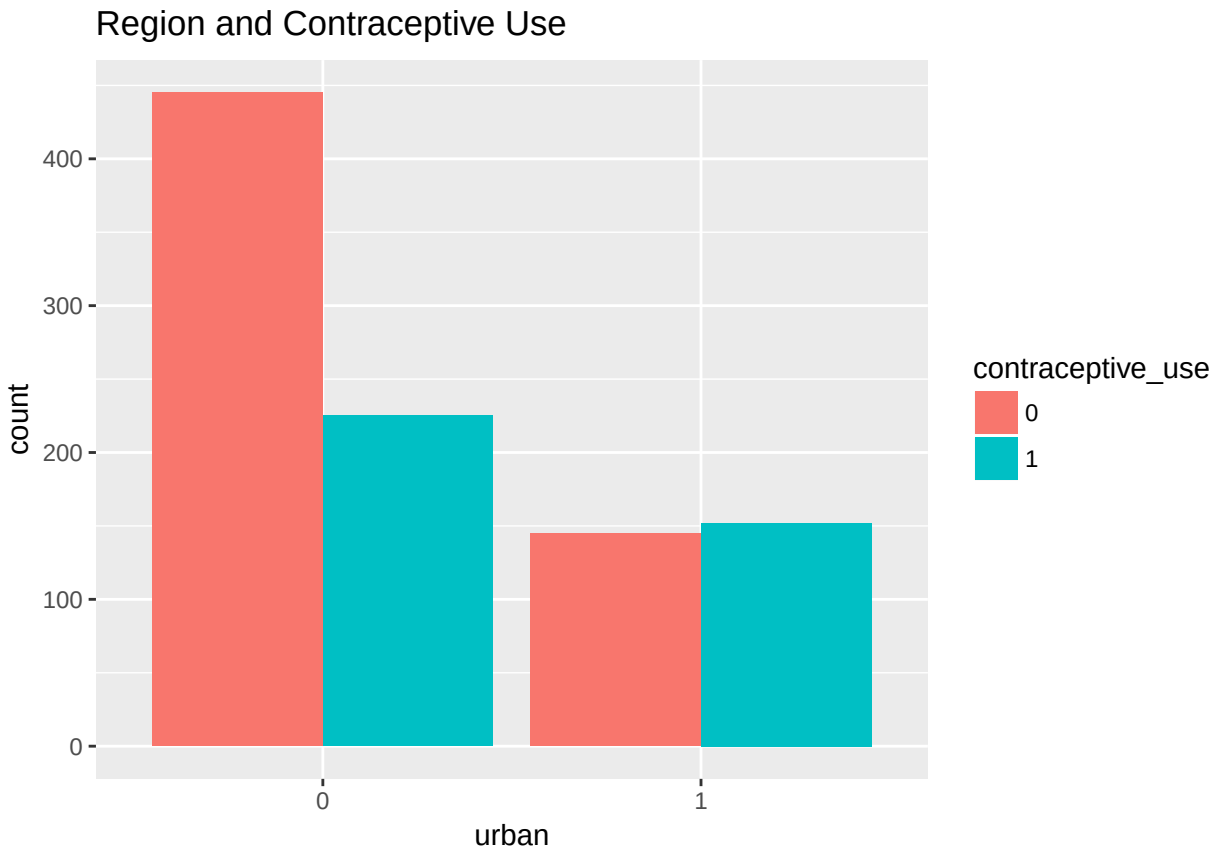
Problem 2: Bayesian Logistic Regression

```
# Load training and testing data, passing the preprocessed words as column names
dataset2_train <- read.table("dataset_2_train.txt", header = TRUE, sep = ",")
dataset2_train[, c(2,3,5)] <- sapply(dataset2_train[, c(2,3,5)], as.factor)
dataset2_test <- read.table("dataset_2_test.txt",header = TRUE, sep = ",")
dataset2_test[, c(2,3,5)] <- sapply(dataset2_test[, c(2,3,5)], as.factor)
```

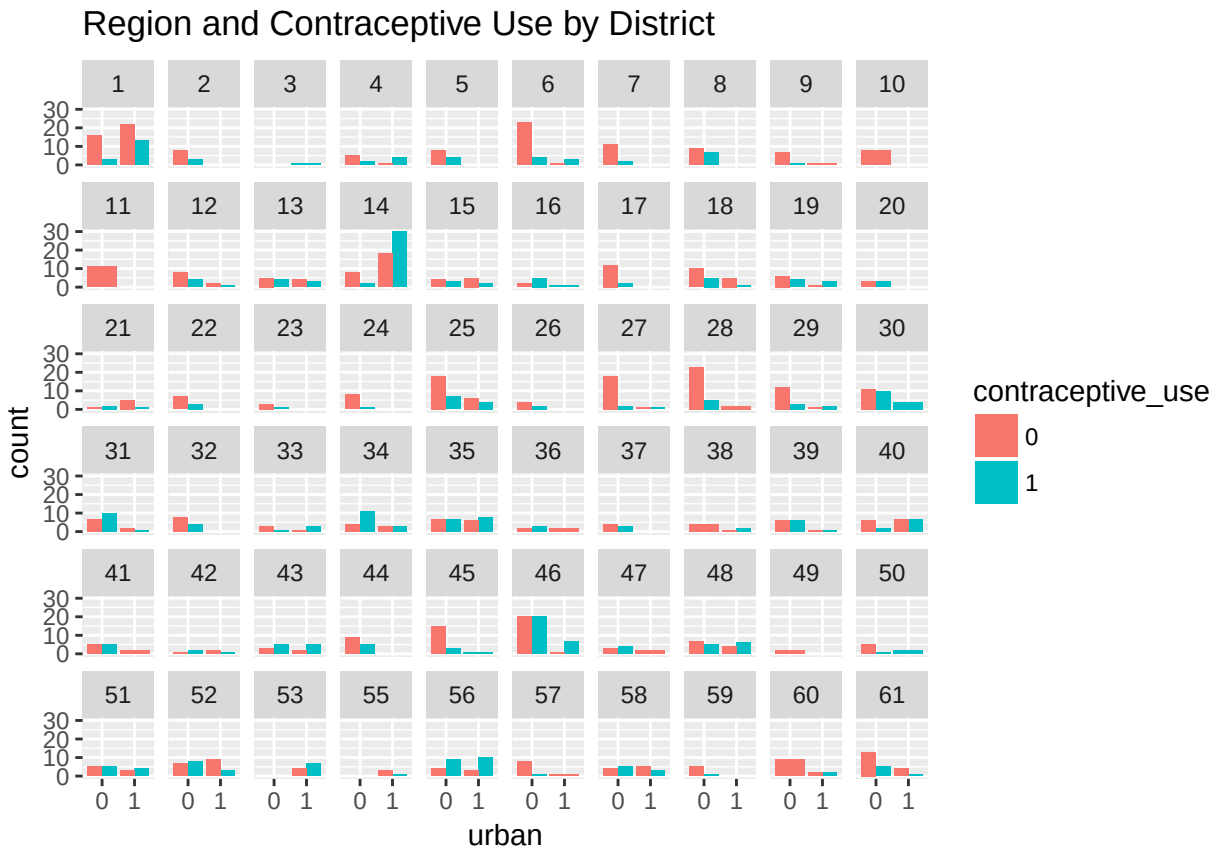
Visualize Data

Region

```
# Plot contraceptive use by region
ggplot(dataset2_train, mapping = aes(x = urban, fill = contraceptive_use)) + geom_bar(position = "dodge
```



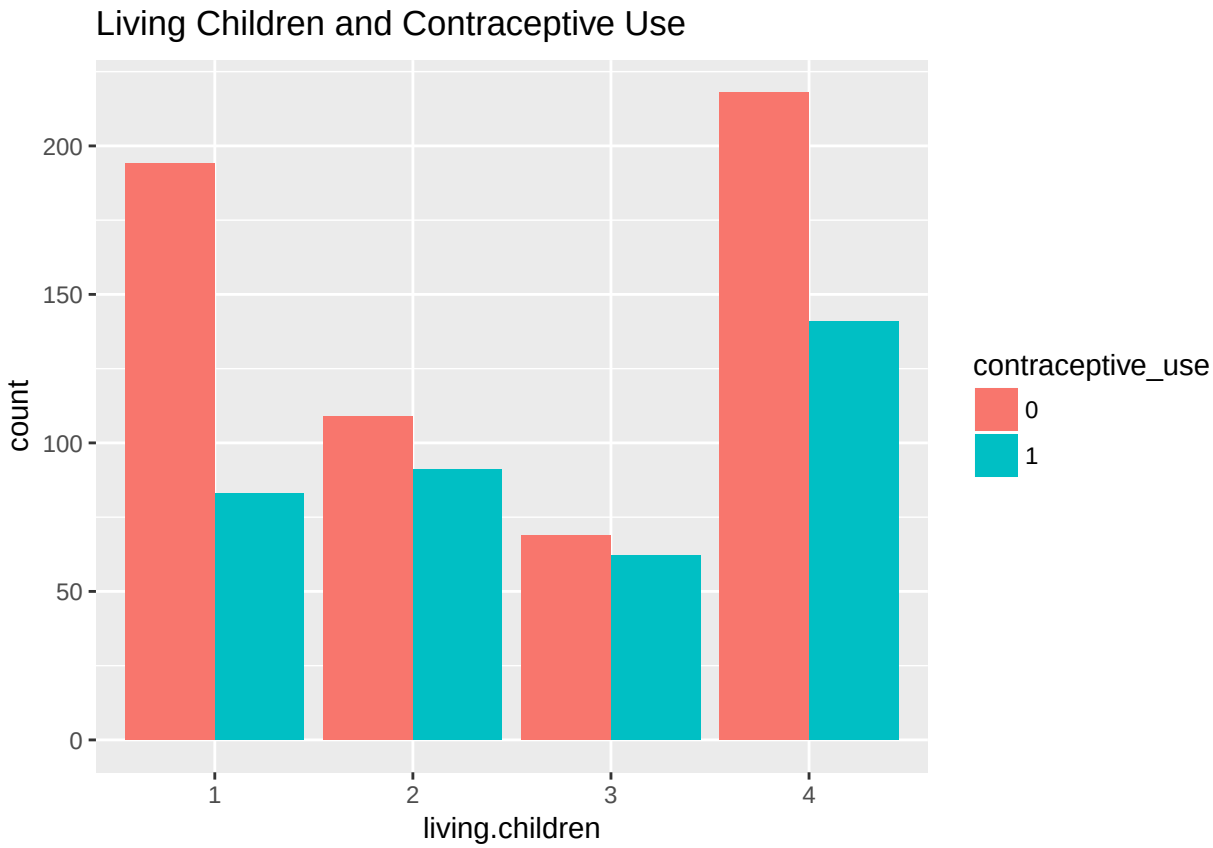

```
# Plot contraceptive use by living children by district
ggplot(dataset2_train, mapping = aes(x = urban, fill = contraceptive_use)) + geom_bar(position = "dodge
```



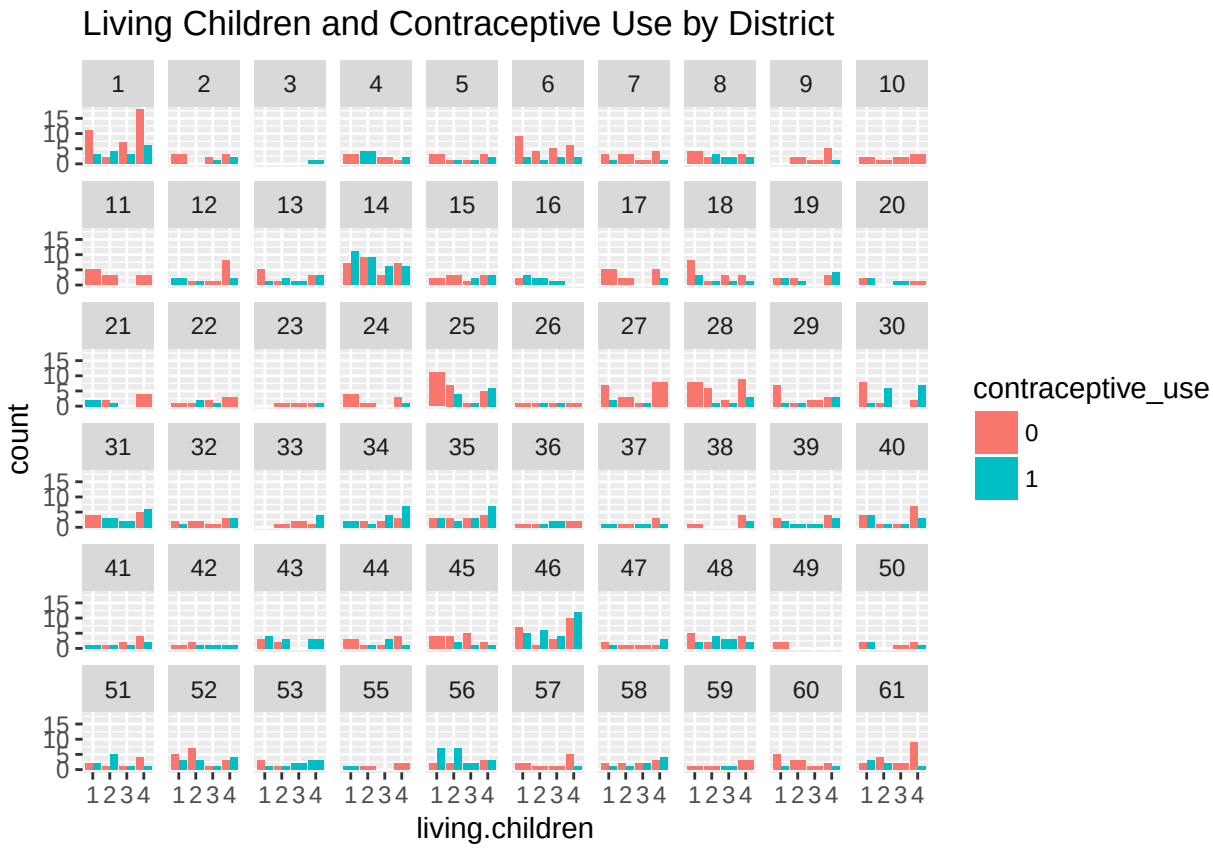
There does seem to be significant variation in regional contraceptive use depending on district, where some non-urban districts still show balanced if not majority contraceptive use (34, 36, and 54).

Children

```
# Plot contraceptive use by living children
ggplot(dataset2_train, mapping = aes(x = living.children, fill = contraceptive_use)) + geom_bar(positio
```




```
# Plot contraceptive use by living children by district
ggplot(dataset2_train, mapping = aes(x = living.children, fill = contraceptive_use)) + geom_bar(positio
```

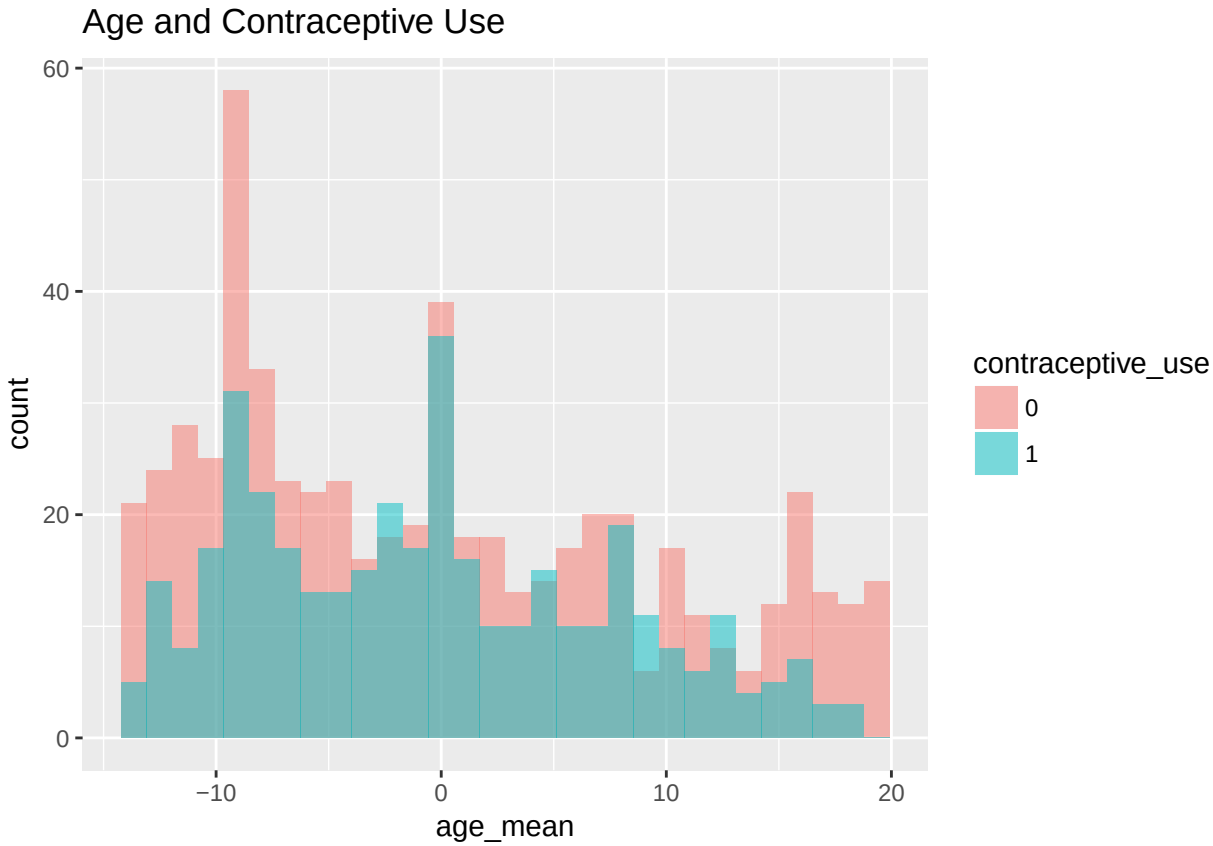


There does seem to be significant variation in contraceptive use by living children depending on district, where some districts shower higher uses of contraception among low-children women (51 and 61) while others show the opposite (28 and 30).

Age

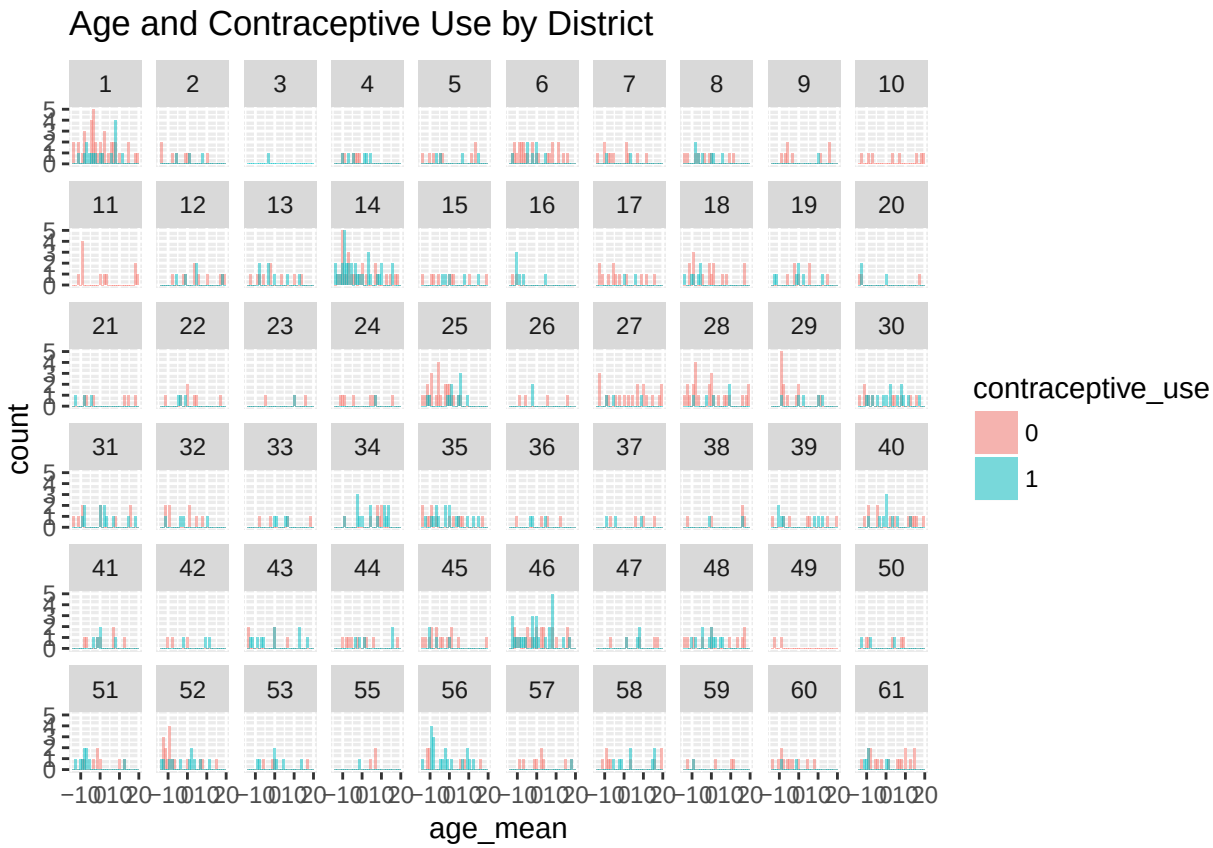
```
# Plot contraceptive use by age
ggplot(dataset2_train, mapping = aes(x = age_mean, fill = contraceptive_use)) + geom_histogram( alpha=.5
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Plot contraceptive use by living children by district
ggplot(dataset2_train, mapping = aes(x = age_mean, fill = contraceptive_use)) + geom_histogram( alpha=.5
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



There does seem to be significant variation in contraceptive use by average age depending on district, where some districts shower higher uses of contraception among younger women (51 and 61) while others show the opposite (6 and 22).

Logistic Regression Models

```
# Load the individual datasets
dataset2_train <- read.table("dataset_2_train.txt", header = TRUE, sep = ",")
dataset2_test <- read.table("dataset_2_test.txt",header = TRUE, sep = ",")

# Loop through each disrict and fit a logistic regression for each one
districts <- sort(unique(dataset2_train$district))
individual_log_acc = rep(0., length(districts))
for (i in 1:length(districts)) {
  train_set <- dataset2_train[dataset2_train$district == districts[i], ]
  test_set <- dataset2_test[dataset2_test$district == districts[i], ]
  model.log <- glm(contraceptive_use ~ urban + living.children + age_mean, data = train_set, family = "
  pred.log <- predict(model.log, newdata = test_set, type = "response")
  table.log <- table((pred.log>.5), test_set$contraceptive_use)
  individual_log_acc[i] <- sum(diag(table.log))/ sum(table.log)
}

# Take weighted average to get total accuracy for individual regression models
model_output_data <- as.data.frame(table(dataset2_test$district))
model_output_data <- rename(model_output_data, c("Var1" = "district", "Freq" = "count"))
```

```
model_output_data$individual_model_acc <- individual_log_acc
acc.individual.log <- sum(model_output_data$count * model_output_data$individual_model_acc)/ sum(model_
```

```
# Fit and measure logistic regression on entire dataset
model.general.log <- glm(contraceptive_use ~ urban + living.children + age_mean, data = dataset2_train,
pred.general.log <- predict(model.general.log, newdata = dataset2_test, type = "response")
table.general.log <- table((pred.general.log>.5), dataset2_test$contraceptive_use)
acc.general.log <- sum(diag(table.general.log))/ sum(table.general.log)
```

```
# Print accuracy of individual and entire dataset models
print(sprintf("Individual District Model Accuracy: %.2f", acc.individual.log))
```

```
## [1] "Individual District Model Accuracy: 0.61"
```

```
print(sprintf("Entire Dataset Model Accuracy: %.2f", acc.general.log))
```

```
## [1] "Entire Dataset Model Accuracy: 0.65"
```

The model fit on the entire dataset performs slightly better than the individual models (weighted average based on number of people on each district), however both show room for potential improvement.

Bayesian Hierarchical Model

```
# Fit an mcmc bayes model
model.hierarchical <- MCMChlogit(fixed = contraceptive_use ~ urban + living.children + age_mean, random
```

```
##
## Running the Gibbs sampler. It may be long, keep cool :)
##
## **********:10.0%, mean accept. rate=0.874
## **********:20.0%, mean accept. rate=0.510
## **********:30.0%, mean accept. rate=0.456
## **********:40.0%, mean accept. rate=0.617
## **********:50.0%, mean accept. rate=0.685
## **********:60.0%, mean accept. rate=0.558
## **********:70.0%, mean accept. rate=0.682
## **********:80.0%, mean accept. rate=0.636
## **********:90.0%, mean accept. rate=0.695
## **********:100.0%, mean accept. rate=0.473
```

```
# helper function to get column names for parsing mcmc coefficient outputs
get_columns <- function(mcmc) {
  columns <- colnames(mcmc)
  for (i in 1:length(columns)) {
    columns[i] <- substr(columns[i], nchar(columns[i])-2+1, nchar(columns[i]))
    columns[i] <- gsub("[.]", "", columns[i])
  }
  return(columns)
}

# helper function to predict test set classification from mcmc bayes model
predict_mcmc_logit <- function(columns, model, newdata) {
  predictions <- rep(0, dim(newdata)[1])
  for (i in 1:dim(newdata)[1]) {
    district <- newdata[i, 1]
```

```
    test_set <- newdata[i, -1]
    individual_model <- model[columns == toString(district)]
    predictions[i] <- 1 / (1 + exp(-individual_model %*% do.call(rbind, c(1,test_set))))
  }
  return(predictions)
}

# Predict test set classification
columns <- get_columns(model.hierarchical$mcmc)
pred.mcmc <- predict_mcmc_logit(columns, summary(model.hierarchical$mcmc)$statistics[, 1], dataset2_tes

# Loop through each individual district and calculate prediction accuracy for mcmc model
mcmc_individual_acc = rep(0., length(districts))
for (i in 1:length(districts)) {
  test_set <- dataset2_test[dataset2_test$district == districts[i], ]
  pred_set <- pred.mcmc[dataset2_test$district == districts[i]]
  table.mcmc <- table(pred_set, test_set$contraceptive_use)
  mcmc_individual_acc[i] <- sum(diag(table.mcmc))/ sum(table.mcmc)
}

# Take weighted average to get total accuracy for mcmc bayes model
model_output_data$mcmc_individual_acc <- mcmc_individual_acc
acc.individual.mcmc <- sum(model_output_data$count * model_output_data$mcmc_individual_acc)/ sum(model_

print(sprintf("MCMC Model Accuracy: %.2f", acc.individual.mcmc))
```

```
## [1] "MCMC Model Accuracy: 0.61"
```

The Bayesian hierarchical model performs almost identically to the locally fitted regression models on the test set. This makes sense given that the single logistic regression acts as a prior constraint on individually fitted models in the Bayesian hierarchical model. As such, the Bayesian hierarchical model should perform similarly to the individually fitted models with the differences caused by this prior constraint.