CS464
Introduction to Machine Learning

Homework 1

Yiğit Ali Karadoğan

21902218 – IE

Section 1

Question 1: Programmers and Energy Drink

Question 1.1

D → Programmer whether consumed energy drink or not (1=Yes, 0=No)

I → Programmer whether showed an improvement or not (1=Yes, 0=No)

Let E and I be our binary random variable. We know that:

P(D=1)=0.6   P(I=1|D=1)=0.7

P(D=0)=0.4   P(I=1|D=0)=0.4

The question is asking **P(I=1)=?**

P(I=1) = P(D=1) * P(I=1|D=1) + P(D=0) * P(I=1|D=0)

= 0.6*0.7+0.4*0.4

= **0.58**

Question 1.2

Let's introduce another binary random variable

E → Programmer whether is an experienced or not (1=Yes, 0=No)

P(E=1|D=1) = 0.5, P(I=1|D=1, E=1) = 0.8

The question is asking P(E=1|D=1, I=1)=?

By the Product Rule, we know that P(E=1|D=1, I=1) * P(D=1, I=1) = P(E=1, D=1, I=1)

P(E=1, D=1) = P(E=1|D=1) * P(D=1) = 0.6 * 0.5 = 0.3

P(I=1, E=1, D=1) = P(I=1|D=1, E=1) * P(D=1, E=1) = 0.80 * 0.3=0.24

P(E=1|D=1, I=1) = P(I=1, E=1, D=1) / (P(I=1|D=1) * P(D=1))

= 0.24 / (0.7 * 0.6) = 0.24 / 0.42

= **0.5714**

Question 1.3

P(E=0|D=0) = 0.6, P(I=0|D=0, E=0) = 0.41, P(I=1|E=0) = 0.56, P(E=0|I=1) = ?

P(E=0|I=1) = P(E=0,I=1)/P(I=1) = P(I=1|E=0)*P(E=0)/P(I=1) = 0.56* P(E=0)/0.58

By Bayes' Rule: P(E=0) = P(E=0|D=0)P(D=0)+ P(E=0|D=1)P(D=1)

= 0.6*0.4+0.5*0.6

=0.54

So, P(E=0|I=1) = 0.56* 0.54/0.58 = **0.5214**

Question 1.4

Let's introduce another binary random variable

C → Programmer whether consumes caffeine daily or not (1=Yes, 0=No)

P(C=1|E=1, I=1)=0.7, P(C=1|E=0, I=1)=0.3, P(C=1|I=0)=0.2, P(C=1|I=1)=?

P(C=1|I=1) = P(C=1|I=1,E=1)*P(E=1)+ P(C=1|I=1,E=0)*P(E=0)

= 0.7 * 0.46 + 0.3 * 0.54

= **0.484**

Question 2: Height of giants!?

Q 2-1)

Methodology and Derivation

$$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Probability of samples D={x_1,….,x_n}

$$P(\mathrm{D}|\mu,\sigma^2) = \prod_{i=1}^{n} f(x_i;\mu,\sigma^2)$$

$$P(\mathrm{D}|\mu,\sigma^2) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n \prod_{i=1}^{n} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Taking the logarithm of the likelihood function (log-likelihood):

$$\ln P(\mathrm{D}|\mu,\sigma^2) = \ln\left[\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n \prod_{i=1}^{n} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right]$$

$$= -N\ln\sigma\sqrt{2\pi} - \sum_{i=1}^{n}\left[\frac{(x_i-\mu)^2}{2\sigma^2}\right]$$

The derivative of the log-likelihood with respect to (μ) is:

$$\frac{d}{d\mu}\ln P(\mathrm{D}|\mu,\sigma^2) = \frac{d}{d\mu}(-N\ln\sigma\sqrt{2\pi}) - \frac{d}{d\mu}(\sum_{i=1}^{n}\left[\frac{(x_i-\mu)^2}{2\sigma^2}\right])$$

Setting the derivative equal to zero:

$$= \sum_{i=1}^{n}\frac{(x_i-\mu)}{\sigma^2} = 0$$

$$= \sum_{i=1}^{n}(x_i) - n\mu = 0$$

$$\mu_{\mathrm{MLE}} = \frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{x_i}$$

The derivative of the log-likelihood with respect to σ is:

$$\frac{d}{d\sigma}\ln P(\mathcal{D}\mid\mu,\sigma) = \frac{d}{d\sigma}\left[-N\ln\sigma\sqrt{2\pi} - \sum_{i=1}^{N}\frac{(x_i-\mu)^2}{2\sigma^2}\right]$$

Setting the derivative equal to zero:

$$= -\frac{N}{\sigma} + \sum_{i=1}^{N}\frac{(x_i-\mu)^2}{\sigma^3} = 0$$

$$\hat{\sigma}^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N}(x_i-\hat{\mu})^2$$

However, it is biased estimator.

So, the unbiased Maximum Likelihood Estimation for σ is the sample variance:

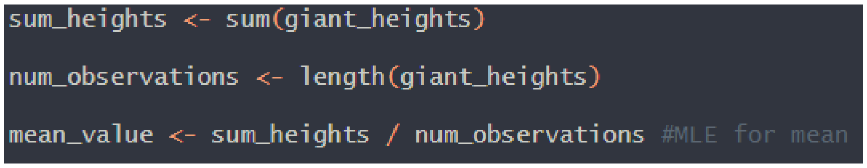
$$\hat{\sigma}^2_{unbiased} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i-\hat{\mu})^2$$

Application in code:

We proved that

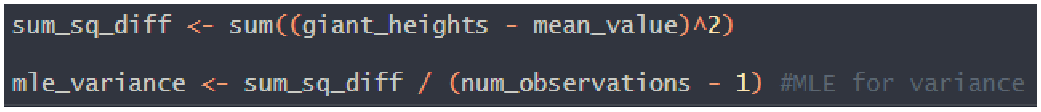
$$\mu_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \hat{\sigma}^2_{unbiased} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \hat{\mu})^2$$

I utilized R for the second question. For MLE of mean, I sum up all values and divide by number of observations as follows:



```
sum_heights <- sum(giant_heights)

num_observations <- length(giant_heights)

mean_value <- sum_heights / num_observations #MLE for mean
```

For MLE of variance, I subtracted each observation from sample mean, and squared it. Then, I sum it up. To be unbiased estimator, I divided by one less of number of observations:

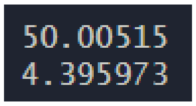


```
sum_sq_diff <- sum((giant_heights - mean_value)^2)

mle_variance <- sum_sq_diff / (num_observations - 1) #MLE for variance
```

After the calculations, MLE for mean and variance is shown at the output:

MLE for Mean:

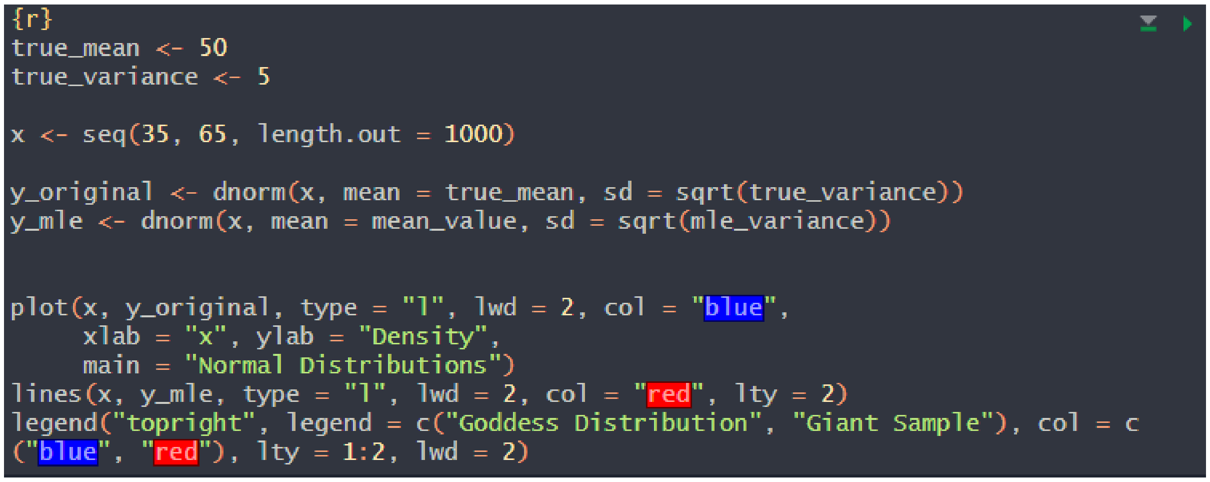
MLE for Variance:



```
50.00515
4.395973
```

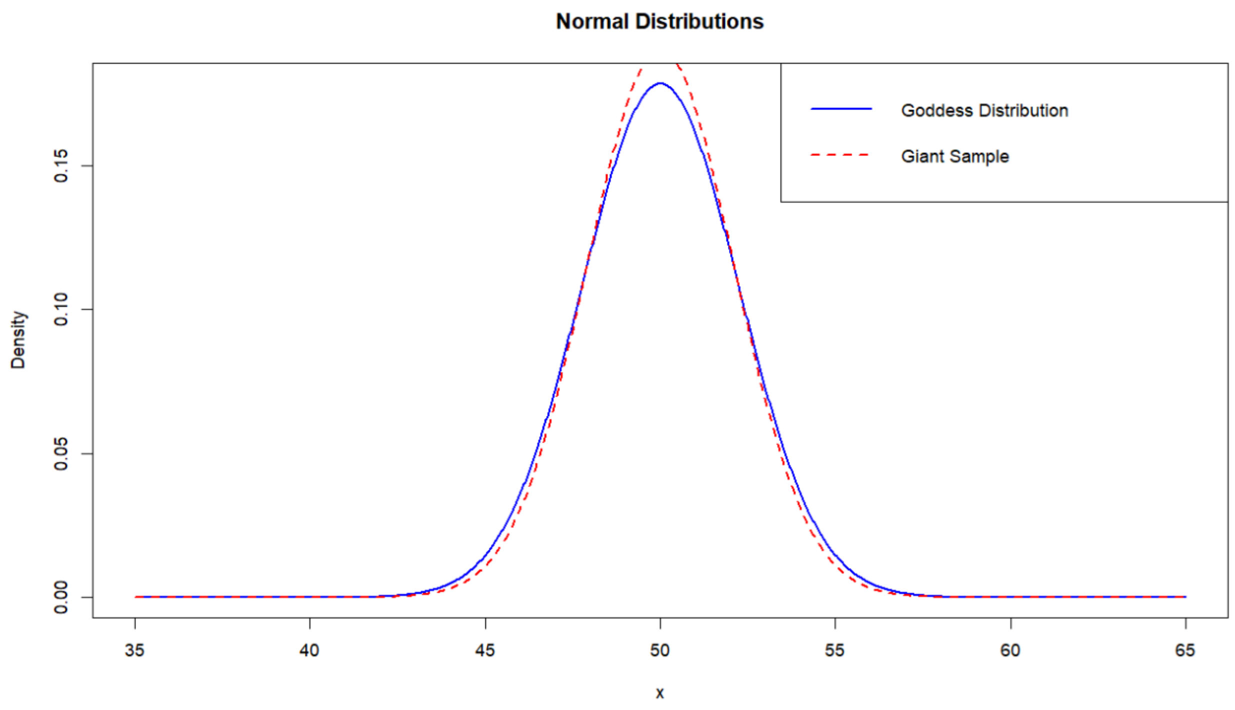
Q 2-2) Plot of Both Distributions

As you can see from the code, there are two y values, one of which represents the distribution of the goddess claim (mean = 50, variance = 5), and the other represents the distribution I've inferred (mean = 50.005, variance = 4,396). I've plotted the goddess claim as solid blue line, my claim as dashed red line.



```
{r}
true_mean <- 50
true_variance <- 5

x <- seq(35, 65, length.out = 1000)

y_original <- dnorm(x, mean = true_mean, sd = sqrt(true_variance))
y_mle <- dnorm(x, mean = mean_value, sd = sqrt(mle_variance))


plot(x, y_original, type = "l", lwd = 2, col = "blue",
     xlab = "x", ylab = "Density",
     main = "Normal Distributions")
lines(x, y_mle, type = "l", lwd = 2, col = "red", lty = 2)
legend("topright", legend = c("Goddess Distribution", "Giant Sample"), col = c
("blue", "red"), lty = 1:2, lwd = 2)
```

From the plots, we can see that distribution of the goddess claim has is more stretched because variance of it is bigger than sample's variance (5 > 4.396). That makes plot of the sample more shrinked towards its mean compared to plot of goddess claim.



Q 2-3) Increasing Sample Size

According to the law of large numbers, as sample size increases, mean of the sample converges to true population mean. This is because as the sample size increases, it becomes more representative of the population. Hence, most probable scenario would be this scenario.

Also, there are more possible scenarios. For instance, if we increase sample size, we can possibly get the outliers into our sample. Hence, that will skew our sample mean and alter sample variance. The other scenario can be making non-representative sampling. This would cause our sample does not represent true characteristics of the population which increases bias in our estimates.

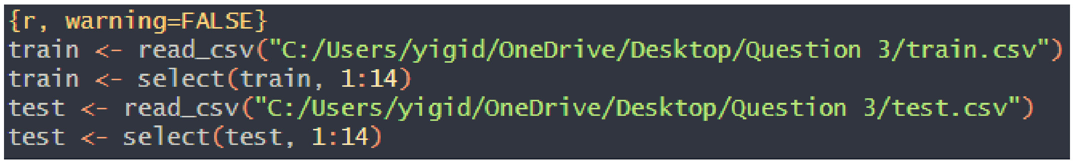
Question 3: UTI Diagnosis

Q 3-1)

In this question, we are asked to train and test a Naïve Bayes Classifier with the UTI Data which has 13 categorical variables and a class variable "Diagnosis". I've utilized R in this question too.

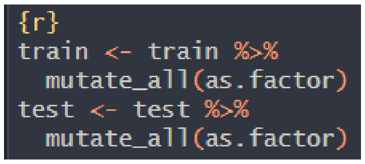
While inspecting the data, I've noticed that excel converted some of the data points into date format. For instance, WBC value of "1-2" is converted to "01-Feb" and RBC value of "12-14" is converted to "Dec-14". To analyze data more meaningfully, I preprocessed these converted data points to their original values in R (it can be seen on .qmd file, not in rendered html).

First I batched the train values, and then test values. I've loaded to R seperately as you can see below:



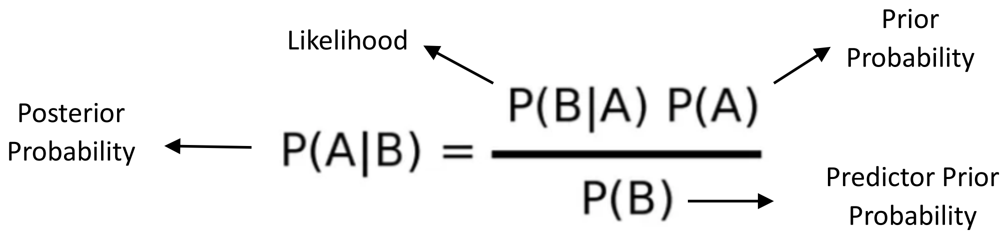
```
{r, warning=FALSE}
train <- read_csv("C:/Users/yigid/OneDrive/Desktop/Question 3/train.csv")
train <- select(train, 1:14)
test <- read_csv("C:/Users/yigid/OneDrive/Desktop/Question 3/test.csv")
test <- select(test, 1:14)
```

After the preprocessing step, I've factorized all variables since all of them are categorical variables. You can see below:

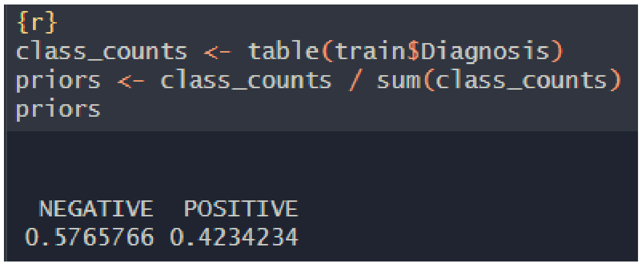


```
{r}
train <- train %>%
  mutate_all(as.factor)
test <- test %>%
  mutate_all(as.factor)
```

We know that posterior probability is proportional to class prior probability times likelihood. Also, we know that this algorithm is called "naïve" since rigid independence assumptions between the input variables are necessary. The formula is shown below:



Hence, in my code, I started calculating with prior class probabilities as below:



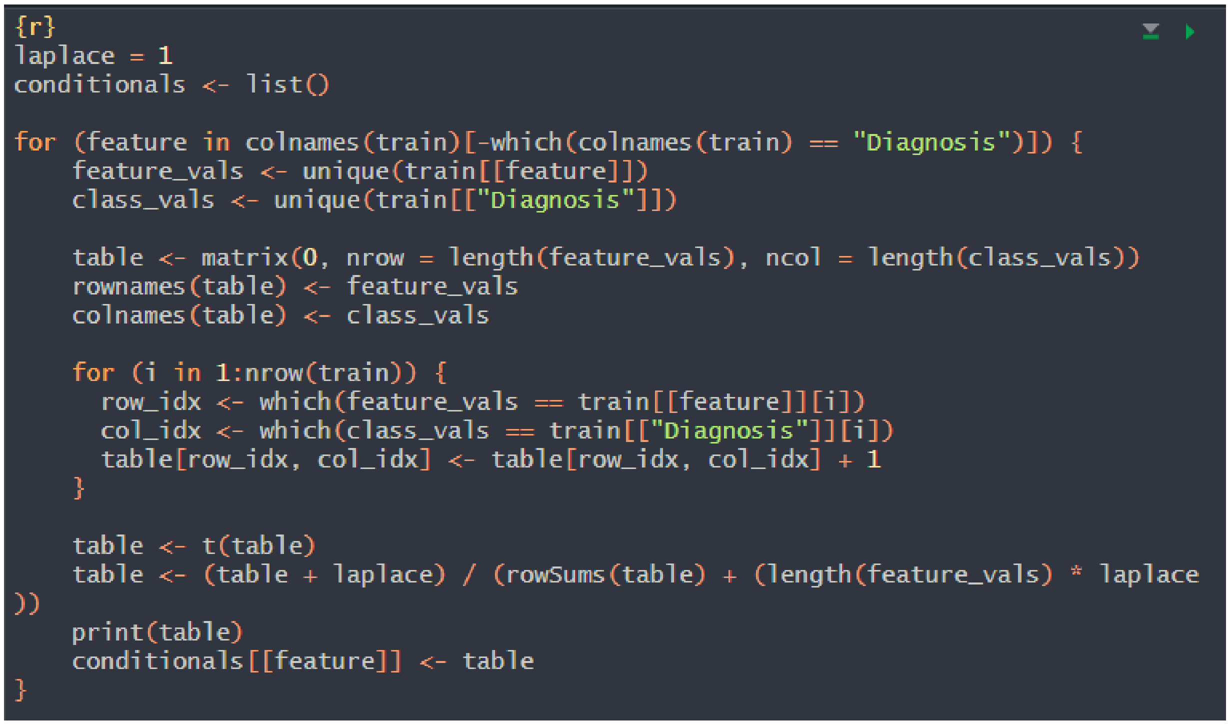
```
{r}
class_counts <- table(train$Diagnosis)
priors <- class_counts / sum(class_counts)
priors


  NEGATIVE  POSITIVE
 0.5765766 0.4234234
```

I’ve calculated prior class probabilities by dividing number of specific class counts to total number of class counts. After that, I’ve calculated conditional probabilities.

I want to emphasize that some of values are seen only once in training set which makes the conditional probability of that value one or zero. In other words, if a specific feature value never appears with a particular class in training data, its conditional probability would be calculated as zero. That's problematic because a single zero probability wipes out the entire likelihood calculation for that class. Hence, I’ve utilized Laplace Smoothing to solve the problem of zero probability. By Laplace Smoothing conditional probability can be written as:

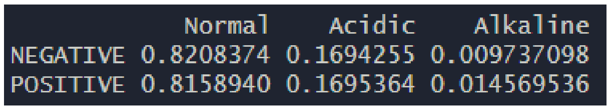
$$p(x_1 = a_j | y = C_k) = \frac{\sum_{i=1}^{N} I(x_{1i} = a_j, y_i = C_k) + \lambda}{\sum_{i=1}^{N} I(y_i = C_k) + A\lambda}$$

where the number of distinct values in a_j is indicated by A. I’ve used lambda = 1 by optimizing the accuracy of the model. Here, you can see:



```
{r}
laplace = 1
conditionals <- list()

for (feature in colnames(train)[-which(colnames(train) == "Diagnosis")]) {
    feature_vals <- unique(train[[feature]])
    class_vals <- unique(train[["Diagnosis"]])

    table <- matrix(0, nrow = length(feature_vals), ncol = length(class_vals))
    rownames(table) <- feature_vals
    colnames(table) <- class_vals

    for (i in 1:nrow(train)) {
      row_idx <- which(feature_vals == train[[feature]][i])
      col_idx <- which(class_vals == train[["Diagnosis"]][i])
      table[row_idx, col_idx] <- table[row_idx, col_idx] + 1
    }

    table <- t(table)
    table <- (table + laplace) / (rowSums(table) + (length(feature_vals) * laplace
))
    print(table)
    conditionals[[feature]] <- table
}
```

In this code, I've iterated each column except "Diagnosis" and got unique values of that feature column and class column. After that I've created a table where rows are unique feature values and columns are class values. Then, I've counted for each feature-class combinations and filled the table. Lastly, I've applied laplace smoothing to overcome zero probabilities (formula is shown above). Here you can see a sample output for "pH_category" feature:

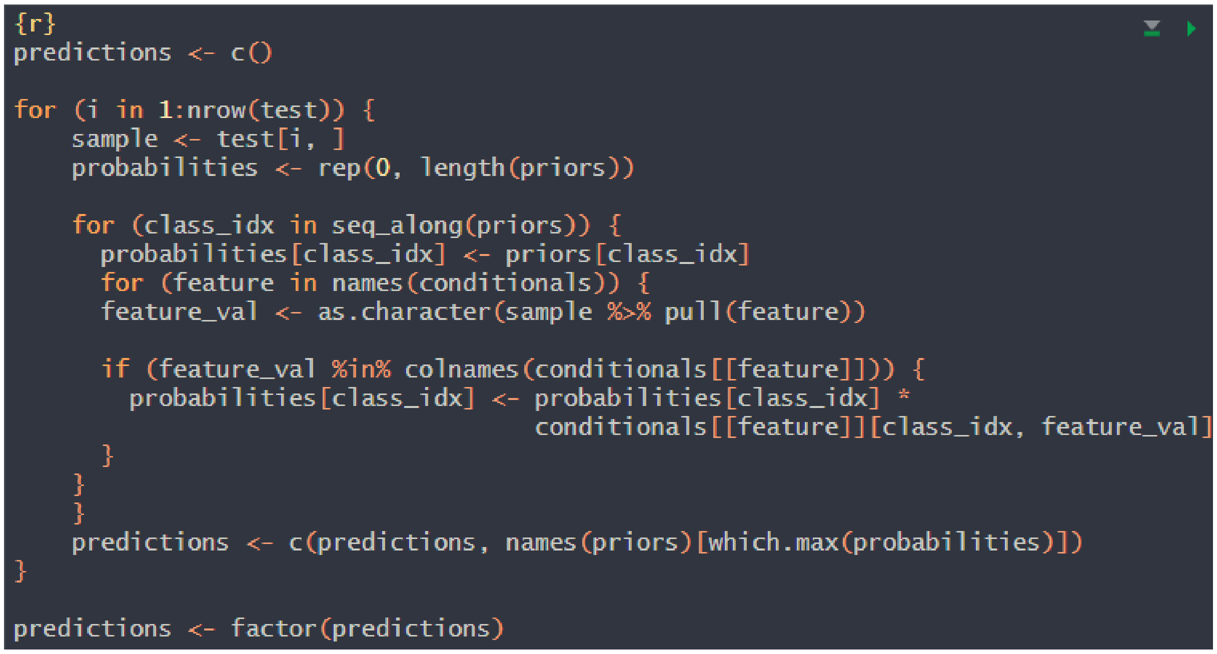


```
            Normal    Acidic     Alkaline
NEGATIVE 0.8208374 0.1694255 0.009737098
POSITIVE 0.8158940 0.1695364 0.014569536
```

P(pH_category=Normal | Diagnosis=Negative)=0.82

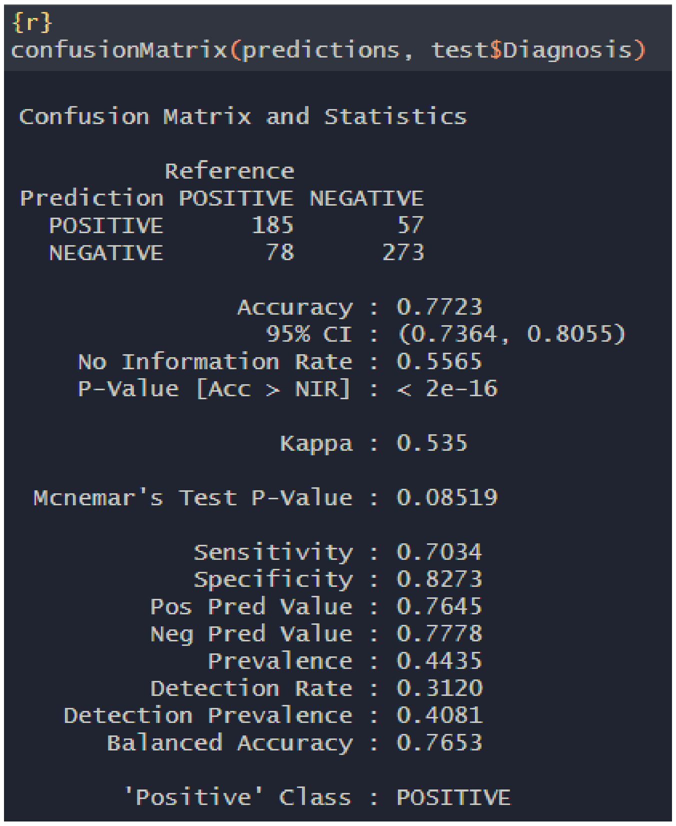
P(pH_category=Acidic | Diagnosis=Positive)=0.16

After I calculate conditionals and prior probabilities, I've calculated the posterior probabilities in the test data. Firstly, I've iterated through all rows in test data and created a vector to hold probabilities for each class. Then, I loop through each class and keep its prior probability. Then, I loop through each feature and extract its conditional probability. Then, I multiplied the prior probability of that class and features of that row and stored in table. Lastly, I've labeled them according to whichever probability is bigger than the other. Here, you can see the code:



```
{r}
predictions <- c()

for (i in 1:nrow(test)) {
    sample <- test[i, ]
    probabilities <- rep(0, length(priors))

    for (class_idx in seq_along(priors)) {
      probabilities[class_idx] <- priors[class_idx]
      for (feature in names(conditionals)) {
      feature_val <- as.character(sample %>% pull(feature))

      if (feature_val %in% colnames(conditionals[[feature]])) {
        probabilities[class_idx] <- probabilities[class_idx] *
                                    conditionals[[feature]][class_idx, feature_val]
      }
    }
    }
    predictions <- c(predictions, names(priors)[which.max(probabilities)])
}

predictions <- factor(predictions)
```

I've noticed that, some of the feature values in test data is not present in the training data (unseen values). I assumed I can ignore them. Lastly, I created a confusion matrix and calculated accuracy of the model.


```
{r}
confusionMatrix(predictions, test$Diagnosis)

 Confusion Matrix and Statistics

           Reference
 Prediction POSITIVE NEGATIVE
   POSITIVE      185       57
   NEGATIVE       78      273

                Accuracy : 0.7723
                  95% CI : (0.7364, 0.8055)
     No Information Rate : 0.5565
     P-Value [Acc > NIR] : < 2e-16

                   Kappa : 0.535

  Mcnemar's Test P-Value : 0.08519

             Sensitivity : 0.7034
             Specificity : 0.8273
          Pos Pred Value : 0.7645
          Neg Pred Value : 0.7778
              Prevalence : 0.4435
          Detection Rate : 0.3120
    Detection Prevalence : 0.4081
       Balanced Accuracy : 0.7653

        'Positive' Class : POSITIVE
```

As you can see below, From the confusion matrix:

- True Positives (TP) = 185
- False Positives (FP) = 57
- False Negatives (FN) = 78
- True Negatives (TN) = 273

According to these values, TPR = TP / (TP + FN) = 185 / (185 + 78) = 0.703 and FPR = FP / (FP + TN) = 57 / (57 + 273) = 0.173. We see that our model does a better job identifying negative values than positive values.

Also, Accuracy = (TP + TN) / (TP + TN + FP + FN) = (185 + 273) / (185 + 273 + 57 + 78) = 0.7723. This means, the model is making correct predictions about 77.23% of the time.

Q 3-2)

Conditional Independence assumption decreases the # of parameters estimated dramatically. We have 13 features and one class variable.

Suppose that a feature in our training data has k unique values. Then, # of parameters we need to estimate for that feature is 2*(k-1). The reason for that is we can find the last one hence k-1 is enough. Also, since we have 2 classes, we need to multiply by 2.

For “Gender”: 2 unique values – 2*(2-1) = 2 parameters

For “Color”: 9 unique values – 2*(9-1) = 16 parameters

For “Transparency”: 5 unique values – 2*(5-1) = 8 parameters

For “Glucose”: 6 unique values – 2*(6-1) = 10 parameters

For “Protein”: 5 unique values – 2*(5-1) = 8 parameters

For “WBC”: 69 unique values – 2*(69-1) = 136 parameters

For “RBC”: 49 unique values – 2*(49-1) = 96 parameters

For “Epithelial Cells”: 7 unique values – 2*(7-1) = 12 parameters

For “Mucous Threads”: 6 unique values – 2*(6-1) = 10 parameters

For “Amorphous Urates”: 6 unique values – 2*(6-1) = 10 parameters

For “Bacteria”: 6 unique values – 2*(6-1) = 10 parameters

For “Age_category”: 3 unique values – 2*(3-1) = 4 parameters

For “pH_category”: 3 unique values – 2*(3-1) = 4 parameters

Their sum is 326. Also, we need to estimate one more parameter for prior probability. Hence, our answer is **327**.

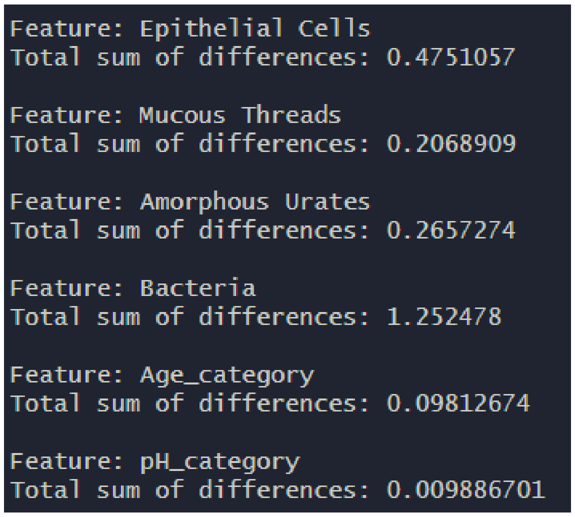
Q 3-3)

We want to calculate P(Disease | Positive Test Result). Equivalently, we need TP / (TP + FP) ratio, which is 185 / (185 + 57) = 0.764. That means If a patient's test result is positive, there's a 76.4% chance they actually have the disease.

Second question asks the True Positive Rate (TPR) that I’ve already calculated. TPR = 0.703. In other words, the model correctly identifies 70.3% of the patients who truly have the disease.

Q 3-4)

For finding most and least effective feature, I subtracted two class probability for each unique feature value. Then, I sum them up for each feature. Example output is shown below:



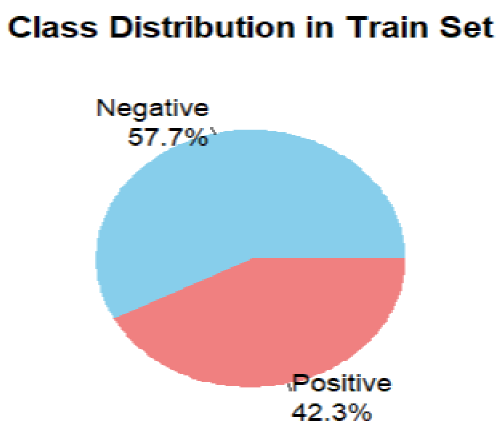
```
Feature: Epithelial Cells
Total sum of differences: 0.4751057

Feature: Mucous Threads
Total sum of differences: 0.2068909

Feature: Amorphous Urates
Total sum of differences: 0.2657274

Feature: Bacteria
Total sum of differences: 1.252478

Feature: Age_category
Total sum of differences: 0.09812674

Feature: pH_category
Total sum of differences: 0.009886701
```

This is the example output and it goes on for each feature. We see from table:

- Most Effective Feature for Positive Diagnosis: **Bacteria**
- Least Effective Feature for Positive Diagnosis: **pH_Category**

Q 3-5)

We see that there is 1024 negative class, 752 positive class points in train set. Meaning that 57.7% of the points belong to negative class, 42.3% of the points belong to positive class. Here you can see the pie chart visualization:



Hence, it is hard to say that we have "balanced" data. It is clear training data is imbalanced. It is skewed towards negative class.

Imbalanced dataset affects naïve bayes model. My model learns the conditional probabilities of features according to their frequency in the training data, as well as the prior probabilities of each class. The majority class prevails in these computations in an unbalanced dataset, which causes the model to become biased in favor of more frequently predicting the majority class. Therefore, our performance metrics doesn't have satisfying values in that case.

To prevent that, we can oversampling (duplicate minority class points to understand its characteristics better) or undersampling (remove points of majority class to make balanced set). In terms of learning style, we can penalize the model more making misclassification for minority class.