Spring 2023
CS 464 INTRODUCTION TO DATA SCIENCE

Yagiz Yaman

2023-04-03

Homework 1

Library Used

The library **data.table** is used to read the files and to perform vectorized operations.

QUESTION 1

A student taking CS 464 course at Bilkent University is hired by an online shopping company to infer some probabilities from historical statistics.

The company aims to find the relationship between the number of sales and customer feedback. For this purpose, they categorized the products according to these features.

Each product's feedback type is set as - **positive (FP)** or - **negative (FN)**

based on the customer comments.

Besides, they are labeled as,

- **popular (P)**,
- **moderately sold (M)**, and
- **unpopular (U)** regarding their number of sales.

According to the historical data,

- 95% of the **popular products**,
- 60% of the **moderately sold products**, and
- 10% of the **unpopular products** have got positive feedback.

Also, it is observed that 45%, 30%, and 25% of the products have been **popular**, **moderately sold**, and **unpopular** respectively.

The company asks the student the following questions:

Question 1.1 [10 pts]

What is the probability that a product gets positive feedback, $P(FP)$?

Answer 1.1

The events are;

- positive feedback,
- negative feedback,

- being popular,
- being moderately sold and
- being unpopular.

The given probabilities are;

$P(\text{popular}) = 0.45$

$P(\text{moderately sold}) = 0.3$

$P(\text{unpopular}) = 0.25$

$P(\text{positive feedback} \mid \text{popular}) = 0.95$

$P(\text{positive feedback} \mid \text{moderately sold}) = 0.6$

$P(\text{positive feedback} \mid \text{unpopular sold}) = 0.1$

In light of these facts, what we are seeking to find is P(positive feedback).

The probability of positive feedback can be expressed as,

$$\begin{aligned}
&P(\text{positive feedback}) = \\
&P(\text{positive feedback} \cap \text{popular}) + \\
&P(\text{positive feedback} \cap \text{moderately sold}) + \\
&P(\text{positive feedback} \cap \text{unpopular}) = \\
&P(\text{positive feedback} \mid \text{popular}) P(\text{popular}) + \\
&P(\text{positive feedback} \mid \text{moderately sold}) P(\text{moderately sold}) + \\
&P(\text{positive feedback} \mid \text{unpopular}) P(\text{unpopular}) = \\
&(0.95)(0.45) + (0.6)(0.3) + (0.1)(0.25) = 0.6325
\end{aligned}$$

Question 1.2 [10 pts]

If a new product gets positive feedback, what is the probability that it will be a popular product, P(P | FP)?

Answer 1.2

It is asked to find P(popular | positive feedback).

Note: We are going to benefit from the previous question as follows,

$P(\text{positive feedback}) = 0.6325$

$P(\text{popular} \cap \text{positive feedback}) = 0.4275$

Hence,

$$\begin{aligned}
&P(\text{popular} \mid \text{positive feedback}) = \\
&\frac{P(\text{popular} \cap \text{positive feedback})}{P(\text{positive feedback})} = \\
&\frac{0.4275}{0.6325} = 0.6759
\end{aligned}$$

Question 1.3 [10 pts]

If a product does not get positive feedback, what is the probability that it will be a popular product [P(P | FN)]?

Answer 1.3

It is asked to find P(popular | negative feedback).

Note: We are going to benefit from the previous question as follows,

P(positive feedback) = 0.6325

P(positive feedback | popular) = 0.95

Hence,

$$\begin{aligned}
&P(\text{popular} \mid \text{negative feedback}) = \\
&\frac{P(\text{popular} \cap \text{negative feedback})}{P(\text{negative feedback})} = \\
&\frac{P(\text{negative feedback} \mid \text{Popular})P(\text{Popular})}{\text{1-P(positive feedback)}} = \\
&\frac{(1-0.95)(0.45)}{1-0.6325} = 0.612
\end{aligned}$$

QUESTION 2

Question 2.1 [8 points]

If the ratio of the classes in a dataset is close to each other, it is called 'balanced' class distribution; i.e it is not skewed. Regarding the class imbalance problem, answer the following questions:

1

What is the percentage of spam e-mails in the y train.csv?

1- Answer

```
percentage_of_spam = 100*(nrow(y_train[Prediction==1])/nrow(y_train))
percentage_of_spam
```

```
## [1] 28.5956
```

28.5926% of the e-mails are spam.

2

Is the training set balanced or skewed towards one of the classes? Do you think having an imbalanced training set affects your model? If yes, please explain how it can affect the model briefly.

2- Answer

Based on the definition of a balanced class distribution, where the ratio of the classes in a dataset is close to each other, it can be concluded that our dataset is imbalanced because the percentage of spam emails is significantly lower than the percentage of non-spam emails. Typically, a dataset is considered balanced when the class distribution is between 40% and 60%. In our case it is roughly 29% to 71%. So, the percentage of spam emails is much lower than this range, indicating a severe class imbalance.

I think having an imbalanced training set would affect the model. The model would be likely to predict the majority class most of the time because it is seen more often during training.

In Naive Bayes classification, we calculate the prior probabilities for each class based on the proportion of examples in the training set that belong to each class.

$$C_{MAP} = \underset{c \in C}{argmax}\, P(c|d) \propto \underset{c \in C}{argmax}\, P(d|c)P(c)$$

In an imbalanced dataset, the prior probabilities will be affected, and the model may overestimate the prior probability of the majority class and underestimate the prior probability of the minority class. As a result, the model will be biased towards the majority class, leading to poor performance on the minority class.

For example, in an email spam classification problem, if the majority of the emails in the training set are non-spam, the prior probability of the non-spam class will be higher than the prior probability of the spam class. As a result, the model will be biased towards the non-spam class and may incorrectly classify some spam emails as non-spam.

Also this situation can lead to overfitting, where the model becomes too specialized to the training data and fails to generalize well on new, unseen data because the model simply learns that the majority class is more common and make predictions based on that, without taking into account the actual features that differentiate the classes.

3

If your dataset is skewed towards one of the classes, how does this affect your reported accuracy? If yes, to what extent the reported accuracy is misleading in such an unbalanced dataset?

3- Answer

If the dataset is skewed towards one of the classes, the reported accuracy may be misleading. We can explain this phenomena by giving an extreme example. Assuming 99% of the mails in the train dataset is spam, a simple model that always predicts spam is going to achieve an accuracy of 99%, which is quite successful. However, it is not logical to use this model in practice because it is not able to detect the normal mails.

In unbalanced datasets, accuracy is not that much of a reliable measure of model performance because it neglects the distribution of the classes. Other metrics such as precision and F-measure provide better understanding about the performance of the model.

Question 2.2 (Coding*) [20 points]

Question

Train a Multinomial Naive Bayes model on the training set and evaluate your model on the test set given.

Find and report the *accuracy* and the *confusion matrix* for the test set as well as *how many wrong predictions* were made.

Assumptions

To simulate the behavior of the number *-inf*, -10^{12} is assigned.

Solution

Firstly *priors* are going to be calculated.

```
P_spam = nrow(y_train[Prediction==1])/nrow(y_train)
P_normal = nrow(y_train[Prediction==0])/nrow(y_train)
```

Now, `y_train` dataset is going to be merged to `x_train` dataset to create two datasets, namely `all_spam_mail` (all spam mails combined) and `all_normal_mail` (all normal mails combined).

```
x_train[, Prediction := y_train[, Prediction]]

all_spam_mail <- x_train[Prediction == 1][, Prediction := NULL]
all_normal_mail <- x_train[Prediction == 0][, Prediction := NULL]
```

A dataset called `spam`, representing *spam mails* will be created.

```
col_sums_spam <- colSums(all_spam_mail)
spam <- data.table(words = names(col_sums_spam),
                   sums = as.matrix(col_sums_spam))
setnames(spam, "sums.V1", "number_of_occurrences")
spam[, likelihood := number_of_occurrences/sum(number_of_occurrences)]
```

`spam` dataset

```
##                words number_of_occurrences     likelihood
##    1:            the                  7711 0.004415050220
##    2:             to                  8423 0.004822716639
##    3:            ect                  3003 0.001719413281
##    4:            and                  4890 0.002799843804
##    5:            for                  3637 0.002082419615
##   ---
## 2996: infrastructure                    11 0.000006298217
## 2997:       military                    17 0.000009733608
## 2998:       allowing                     5 0.000002862826
## 2999:             ff                  1864 0.001067261524
## 3000:            dry                     8 0.000004580522
```

Here, the column *likelihood* corresponds to $P(Word|Spam)$.

A dataset called `normal`, representing *normal mails* will be created.

```
col_sums_normal <- colSums(all_normal_mail)
normal <- data.table(words = names(col_sums_normal),
                   sums = as.matrix(col_sums_normal))
setnames(normal, "sums.V1", "number_of_occurrences")
normal[, likelihood := number_of_occurrences/sum(number_of_occurrences)]
```

`normal` dataset

```
##                words number_of_occurrences      likelihood
##    1:            the                 19765 0.0063739682814
##    2:             to                 17335 0.0055903233068
##    3:            ect                 18442 0.0059473171285
##    4:            and                  7927 0.0025563595531
##    5:            for                  9307 0.0030013925016
##   ---
## 2996: infrastructure                     6 0.0000019349259
## 2997:       military                     2 0.0000006449753
## 2998:       allowing                    10 0.0000032248764
## 2999:             ff                  1988 0.0006411054360
## 3000:            dry                    20 0.0000064497529
```

Now we have the **priors** and **likelihood**. It is time to predict *classes/labels (spam or normal)* for the test dataset.

$$C_{NB} = \hat{y}_i = \underset{y_k}{argmax}\, P(Y = y_k|D_i) \propto \underset{y_k}{argmax}\, P(Y = y_k) \prod_{j=1}^{V} P(X_j|Y = y_k)^{t_{w_j,i}}$$

By taking the logarithm,

$$\hat{y_i} = \underset{y_k}{argmax} \left(log\mathbf{P}(Y = y_k) + \prod_{j=1}^{|V|} t_{w_j,i} * log\mathbf{P}(X_j|Y = y_k) \right)$$

Accordingly, we are going to calculate *log_likelihood* in the `normal` dataset.

```
normal[, log_likelihood := log(likelihood)]

# assign -10^12 to -Inf
normal[is.infinite(log_likelihood), log_likelihood := -10^12]
```

Now, the probabilities of being a **normal mail**, i.e. $P(Normal|Mail)$, will be calculated for each mail in the `x_test`.

```
number_of_words <- nrow(x_test)
# Create an empty vector with capacity of nrow(x_test) elements to allocate memory
probs_normal <- rep(log(P_normal), nrow(x_test))


for (i in (1:number_of_words)){
  the_probs <- normal[, log_likelihood]*as.numeric(x_test[i])
  probs_normal[i] <- probs_normal[i] + sum(the_probs)
}
```

log_likelihood will be calculated for the `spam` dataset.

```
spam[, log_likelihood := log(likelihood)]

# assign -10^12 to -Inf
spam[is.infinite(log_likelihood), log_likelihood := -10^12]
```

Now, the probabilities of being a **spam mail**, i.e. $P(Spam|Mail)$, will be calculated for each mail in the `x_test`.

```
number_of_words <- nrow(x_test)
# Create an empty vector with capacity of nrow(x_test) elements to allocate memory
probs_spam <- rep(log(P_spam), nrow(x_test))

for (i in (1:number_of_words)){
  the_probs <- spam[, log_likelihood]*as.numeric(x_test[i])
  probs_spam[i] <- probs_spam[i] + sum(the_probs)
}
```

Label each mail in the `x_test` according to the probabilities of $P(Spam|Mail)$ and $P(Normal|Mail)$.

```
compare <- data.table(for_normal = probs_normal,
                      for_spam = probs_spam,
                      prediction = y_test[, Prediction])

compare[, assigned_class := ifelse(for_spam > for_normal, 1, 0)]
```

Results

```
confusion_matrix <- table(compare$assigned_class, compare$prediction,
                  dnn = c("Assigned Class", "Prediction"))

conf_mat <- data.table(confusion_matrix)
```

```
precision <- conf_mat[4, N]/(conf_mat[4,N]+conf_mat[2,N])
recall <- conf_mat[4, N]/(conf_mat[4,N]+conf_mat[3,N])

specificity <- conf_mat[1, N]/(conf_mat[1, N] + conf_mat[3, N])

F_measure <- 2 * (precision * recall) / (precision + recall)

accuracy_percentage <- 100*(nrow(compare[prediction==assigned_class])/nrow(compare))

correct_predictions <- nrow(compare[prediction==assigned_class])

wrong_predictions <- nrow(compare[prediction!=assigned_class])

test_size <- nrow(compare)
```

```
##
## Confusion Matrix:

##                 Prediction
## Assigned Class   0   1
##               0 703  28
##               1  15 289

##
## Performance Metrics

## Accuracy Percentage: 95.8454%

## Precision: 0.9507%

## Recall: 0.9117%

## F-measure: 0.9308%

## Specificity: 0.9617%

## Correct Predictions: 992

## Wrong Predictions: 43

## Test Size: 1035
```

Question 2.3 (Coding*) [16 points]

Question

Extend your classifier so that it can compute an estimate of θ parameters using a fair Dirichlet prior. *This corresponds to additive smoothing.* The prior is fair in the sense that it "hallucinates" that each word appears additionally α times in the train set.

For this question set $\alpha = 5$. Train your classifier using all of the training set and have it classify all of the test set and report test-set *classification accuracy* and the *confusion matrix*.

Explicitly discuss your results and interpret on the effect of the Dirichlet prior α.

Assumptions

To simulate the behavior of the number *-inf*, -10^{12} is assigned.

Solution

Firstly priors will be calculated as before.

```
P_spam = nrow(y_train[Prediction==1])/nrow(y_train)
P_normal = nrow(y_train[Prediction==0])/nrow(y_train)
```

Calculate V and set α to 5 as given.

```
V <- length(names(x_train))
alpha <- 5
```

Now, `y_train` dataset is going to be merged to `x_train` dataset to create two datasets, namely `all_spam_mail` (all spam mails combined) and `all_normal_mail` (all normal mails combined).

```
x_train[, Prediction := y_train[, Prediction]]

all_spam_mail <- x_train[Prediction == 1][, Prediction := NULL]
all_normal_mail <- x_train[Prediction == 0][, Prediction := NULL]
```

A dataset called `normal`, representing *normal mails* will be created.

```
col_sums_normal <- colSums(all_normal_mail)
normal <- data.table(words = names(col_sums_normal),
                     sums = as.matrix(col_sums_normal))
setnames(normal, "sums.V1", "number_of_occurrences")
normal[, likelihood :=
         (number_of_occurrences + alpha)/(sum(number_of_occurrences)+alpha*V)]
```

Dataset `normal`;

```
##                words number_of_occurrences     likelihood
##    1:            the                 19765 0.006344878316
##    2:             to                 17335 0.005565007081
##    3:            ect                 18442 0.005920281755
##    4:            and                  7927 0.002545653758
##    5:            for                  9307 0.002988543595
##   ---
## 2996: infrastructure                     6 0.000003530281
## 2997:       military                     2 0.000002246543
## 2998:       allowing                    10 0.000004814020
## 2999:             ff                  1988 0.000639622786
## 3000:            dry                    20 0.000008023367
```

A dataset called `spam`, representing *spam mails* will be created.

```
col_sums_spam <- colSums(all_spam_mail)
spam <- data.table(words = names(col_sums_spam),
                   sums = as.matrix(col_sums_spam))
setnames(spam, "sums.V1", "number_of_occurrences")
spam[, likelihood :=
       (number_of_occurrences + alpha)/(sum(number_of_occurrences)+alpha*V)]
```

Dataset `spam`;

```
##                words number_of_occurrences     likelihood
##    1:            the                  7711 0.004380280563
##    2:             to                  8423 0.004784474415
##    3:            ect                  3003 0.001707605486
##    4:            and                  4890 0.002778832731
```

```
##    5:            for                   3637 0.002067519675
##   ---
## 2996: infrastructure                    11 0.000009083008
## 2997:       military                    17 0.000012489136
## 2998:       allowing                     5 0.000005676880
## 2999:             ff                  1864 0.001061008861
## 3000:            dry                     8 0.000007379944
```

Now we have the **priors** and **likelihood**. It is time to predict *classes/labels (spam or normal)* for the test dataset.

As we added *smoothers* to both *numerator* and *denominator* while calculating *likelihoods*, it is expected to not see any 0 probabilities for any word. In fact, this is the aim of *smoothing*. However, there is a problem to overcome related to the software. The problem is going to be explained with an example.

Let's calculate the P(3rd mail in the test data | Normal).

3rd mail of the `x_test` is as below (Only first 19 columns is represented below for convenience);

```r
x_test[3,1:19]
```

```
##    the to ect and for of  a you hou in on is this enron  i be that will have
## 1:   7  4  17   3   5  3 64   1   7 12 13 10    0     3 53  3    0    0    1
```

And the *likelihood* of each *word* in a `normal` mail is as follows;

```
##                 words     likelihood
##    1:             the 0.006344878316
##    2:              to 0.005565007081
##    3:             ect 0.005920281755
##    4:             and 0.002545653758
##    5:             for 0.002988543595
##   ---
## 2996: infrastructure 0.000003530281
## 2997:       military 0.000002246543
## 2998:       allowing 0.000004814020
## 2999:             ff 0.000639622786
## 3000:            dry 0.000008023367
```

Now, what we want to calculate is $likelihood[i]^{3rdmail[i]}$ for all is ($3rdmail[i]$ represents the ith word of the $3rdmail$).

```r
prob <- normal[, likelihood]^as.numeric(x_test[3])
prob[1:5]
```

```
## [1] 0.0000000000000041396448385953930932008859589
## [2] 0.00000000095909777886581457292947128223659092
## [3] 0.00000000000000000000000000000000000001348405
## [4] 0.00000001649673511607758069619222851542872377
## [5] 0.00000000000023839545830645428260621127014574
```

Now, `prob` represents the *likelihood* of each word in the 3rd row.

```r
prob[prob==0]
```

```
## numeric(0)
```

As seen, there is no 0 probability in `prob`.

Now, we want to calculate product of all the elements of prob and P(Normal). The function *prod()* enables to take product of a vector. For example, if we have a vector c(2, 3, 4, 5), then prod(vector) gives $2*3*4*5 = 120$.

Let's use prod() for `prob`;

```
prod(prob)
```

```
## [1] 0
```

Although there is not any 0 in the vector `prob`, prod(prob) gives the value 0. The reason is, R stores numerical values with a finite number of digits and although there is not any 0, there are values quite close to 0 in `prob`. Due to floating point precision, the product becomes 0.

I suspected that if it is the case with the function prod() and wrote a for loop for multiplication.

```
x <- 1
for (i in 1:length(prob)){
  x <- x*prob[i]
}

x
```

```
## [1] 0
```

However, it again gives 0.

To prevent this situation, the *log transformation* is going to be applied.

```
normal[, log_likelihood := log(likelihood)]

# assign -10^12 to -Inf
normal[is.infinite(log_likelihood), log_likelihood := -10^12]
```

Now, the probabilities of being a **normal mail**, i.e. $P(Normal|Mail)$, will be calculated for each mail in the `x_test`.

```
number_of_words <- nrow(x_test)
# Create an empty vector with capacity of nrow(x_test) elements to allocate memory
probs_normal <- rep(log(P_normal), nrow(x_test))


for (i in (1:number_of_words)){
  the_probs <- normal[, log_likelihood]*as.numeric(x_test[i])
  probs_normal[i] <- probs_normal[i] + sum(the_probs)
}
```

log_likelihood will be calculated for the `spam` dataset.

```
spam[, log_likelihood := log(likelihood)]

# assign -10^12 to -Inf
spam[is.infinite(log_likelihood), log_likelihood := -10^12]
```

Now, the probabilities of being a **spam mail**, i.e. $P(Spam|Mail)$, will be calculated for each mail in the `x_test`.

```
number_of_words <- nrow(x_test)
# Create an empty vector with capacity of nrow(x_test) elements to allocate memory
probs_spam <- rep(log(P_spam), nrow(x_test))

for (i in (1:number_of_words)){
  the_probs <- spam[, log_likelihood]*as.numeric(x_test[i])
  probs_spam[i] <- probs_spam[i] + sum(the_probs)
```

```
}
```

Label each mail in the `x_test` according to the probabilities of $P(Spam|Mail)$ and $P(Normal|Mail)$.

```
compare <- data.table(for_normal = probs_normal,
                      for_spam = probs_spam,
                      prediction = y_test[, Prediction])

compare[, assigned_class := ifelse(for_spam > for_normal, 1, 0)]
```

Results

```
confusion_matrix <- table(compare$assigned_class, compare$prediction,
                  dnn = c("Assigned Class", "Prediction"))

conf_mat <- data.table(confusion_matrix)

precision <- conf_mat[4, N]/(conf_mat[4,N]+conf_mat[2,N])
recall <- conf_mat[4, N]/(conf_mat[4,N]+conf_mat[3,N])

specificity <- conf_mat[1, N]/(conf_mat[1, N] + conf_mat[3, N])

F_measure <- 2 * (precision * recall) / (precision + recall)

accuracy_percentage <- 100*(nrow(compare[prediction==assigned_class])/nrow(compare))

correct_predictions <- nrow(compare[prediction==assigned_class])

wrong_predictions <- nrow(compare[prediction!=assigned_class])

test_size <- nrow(compare)
```

```
##
## Confusion Matrix:

##                 Prediction
## Assigned Class   0   1
##              0 681  17
##              1  37 300

##
## Performance Metrics

## Accuracy Percentage: 94.7826%

## Precision: 0.8902%

## Recall: 0.9464%

## F-measure: 0.9174%

## Specificity: 0.9756%

## Correct Predictions: 981

## Wrong Predictions: 54

## Test Size: 1035
```

Question 2.4 (Coding*) [20 points]

Question

Train a Bernoulli Naive Bayes classifier using all of the data in the training set, and report the testing accuracy and the confusion matrix as well as how many wrong predictions were made.

Solution

Firstly *priors* are going to be calculated.

```
P_spam = nrow(y_train[Prediction==1])/nrow(y_train)
P_normal = nrow(y_train[Prediction==0])/nrow(y_train)
```

Now, `y_train` dataset is going to be merged to `x_train` dataset to create two datasets, namely `all_spam_mail` (all spam mails combined) and `all_normal_mail` (all normal mails combined).

All values of x_train will be converted to 1 if they are bigger than 0 as we only deal with *occurrences* of a word, not *the number of occurrences* of a word.

```
x_train_for <- copy(x_train)
x_train_for[x_train_for != 0] <- 1

x_train_for[, Prediction := y_train[, Prediction]]

all_spam_mail <- x_train_for[Prediction == 1][, Prediction := NULL]
all_normal_mail <- x_train_for[Prediction == 0][, Prediction := NULL]
```

A dataset called `spam`, representing *spam mails* will be created.

```
col_sums_spam <- colSums(all_spam_mail)
spam <- data.table(words = names(col_sums_spam),
                   sums = as.matrix(col_sums_spam))
setnames(spam, "sums.V1", "number_of_appearance")
spam[, likelihood := number_of_appearance/sum(number_of_appearance)]

spam[, inv_likelihood := 1-likelihood]
```

`spam` dataset

```
##                words number_of_appearance    likelihood inv_likelihood
##    1:            the                  845 0.00378514699      0.9962149
##    2:             to                  977 0.00437643623      0.9956236
##    3:            ect                 1183 0.00529920579      0.9947008
##    4:            and                  786 0.00352085862      0.9964791
##    5:            for                  736 0.00329688543      0.9967031
##   ---
## 2996: infrastructure                    5 0.00002239732      0.9999776
## 2997:       military                    9 0.00004031518      0.9999597
## 2998:       allowing                    5 0.00002239732      0.9999776
## 2999:             ff                  603 0.00270111673      0.9972989
## 3000:            dry                    8 0.00003583571      0.9999642
```

Here, the column *likelihood* corresponds to $P(Word|Spam)$.

A dataset called `normal`, representing *normal mails* will be created.

```
col_sums_normal <- colSums(all_normal_mail)
normal <- data.table(words = names(col_sums_normal),
                     sums = as.matrix(col_sums_normal))
```

```
setnames(normal, "sums.V1", "number_of_appearance")
normal[, likelihood := number_of_appearance/sum(number_of_appearance)]

normal[, inv_likelihood := 1-likelihood]
```

`normal` dataset

```
##               words number_of_appearance     likelihood inv_likelihood
##    1:           the                 2244 0.004704077076      0.9952959
##    2:            to                 2316 0.004855010031      0.9951450
##    3:           ect                 2954 0.006192443709      0.9938076
##    4:           and                 1681 0.003523865225      0.9964761
##    5:           for                 2486 0.005211379506      0.9947886
##   ---
## 2996: infrastructure                   5 0.000010481455      0.9999895
## 2997:       military                   2 0.000004192582      0.9999958
## 2998:       allowing                  10 0.000020962910      0.9999790
## 2999:             ff                 878 0.001840543526      0.9981595
## 3000:            dry                  16 0.000033540657      0.9999665
```

Now we have the **priors**, **likelihoods** and **1 - likelihoods**. It is time to predict *classes/labels (spam or normal)* for the test dataset.

Now, the probabilities of being a **normal mail**, i.e. $P(Normal|Mail)$, will be calculated for each mail in the `x_test`.

```
x_test_for <- copy(x_test)
x_test_for[x_test_for != 0] <- 1
```

log likelihoods will be calculated for the `normal` dataset.

```
normal[, c("log_likelihood", "log_inv_likelihood") :=
         .(log(likelihood), log(inv_likelihood))]
```

Now, the probabilities of being a **Normal mail**, i.e. $P(Normal|Mail)$, will be calculated for each mail in the `x_test`.

```
number_of_words <- nrow(x_test)
# Create an empty vector with capacity of nrow(x_test) elements to allocate memory
probs_normal <- rep(log(P_normal), nrow(x_test))


for (i in (1:number_of_words)){
  take <- as.numeric(x_test_for[i])
  ones_idx <- which(take == 1)
  zeros_idx <- which(take == 0)
  the_probs <- sum(normal[ones_idx, log_likelihood]) +
    sum(normal[zeros_idx, log_inv_likelihood])
  probs_normal[i] <- probs_normal[i] + the_probs
}
```

Now, the same process will be repeated for spams.

log likelihoods will be calculated for the `spam` dataset.

```
spam[, c("log_likelihood", "log_inv_likelihood") :=
         .(log(likelihood), log(inv_likelihood))]
```

```r
number_of_words <- nrow(x_test)
# Create an empty vector with capacity of nrow(x_test) elements to allocate memory
probs_spam <- rep(log(P_spam), nrow(x_test))

for (i in (1:number_of_words)){
  take <- as.numeric(x_test_for[i])
  ones_idx <- which(take == 1)
  zeros_idx <- which(take == 0)
  the_probs <- sum(spam[ones_idx, log_likelihood]) +
    sum(spam[zeros_idx, log_inv_likelihood])
  probs_spam[i] <- probs_spam[i] + the_probs
}
```

Compare with test results.

```r
compare <- data.table(for_normal = probs_normal,
                      for_spam = probs_spam,
                      prediction = y_test[, Prediction])

compare[, assigned_class := ifelse(for_spam > for_normal, 1, 0)]
```

Results

```r
confusion_matrix <- table(compare$assigned_class, compare$prediction,
                  dnn = c("Assigned Class", "Prediction"))

conf_mat <- data.table(confusion_matrix)

precision <- conf_mat[4, N]/(conf_mat[4,N]+conf_mat[2,N])
recall <- conf_mat[4, N]/(conf_mat[4,N]+conf_mat[3,N])

specificity <- conf_mat[1, N]/(conf_mat[1, N] + conf_mat[3, N])

F_measure <- 2 * (precision * recall) / (precision + recall)

accuracy_percentage <- 100*(nrow(compare[prediction==assigned_class])/nrow(compare))

correct_predictions <- nrow(compare[prediction==assigned_class])

wrong_predictions <- nrow(compare[prediction!=assigned_class])

test_size <- nrow(compare)
```

```
##
## Confusion Matrix:

##                 Prediction
## Assigned Class    0   1
##                0 706  53
##                1  12 264

##
## Performance Metrics

## Accuracy Percentage: 93.7198%
```

```
## Precision: 0.9565%
## Recall: 0.8328%
## F-measure: 0.8904%
## Specificity: 0.9302%
## Correct Predictions: 970
## Wrong Predictions: 65
## Test Size: 1035
```

Question 2.5 [6 points]

Using the confusion matrices that you obtained together with the accuracy values, make a brief comparison regarding your results. How does your algorithm behave for different classes? For a real-world application, which algorithm is better and why? Is accuracy deceiving as a performance metric for this task?

Answer 2.5.

The first model achieved the highest accuracy (95.8454%) and a high specificity (0.9617%). It also had high precision (0.9507%), indicating that it correctly classified a high proportion of spam emails out of all the emails it classified as spam. However, its recall (0.9117%) was slightly lower than the second model, indicating that it missed a few spam emails.

The second model achieved the highest recall (0.9464%), indicating that it correctly identified a high proportion of spam emails out of all the actual spam emails. It also had a high specificity (0.9756%). However, its precision (0.8902%) was lower than the first model, indicating that it incorrectly classified some non-spam emails as spam.

The third model had the highest precision (0.9565%), indicating that it correctly classified a high proportion of spam emails out of all the emails it classified as spam. However, its recall (0.8328%) was the lowest among the three models, indicating that it missed a significant proportion of actual spam emails.

If the aim is to minimize false positives (i.e., incorrectly labeling non-spam emails as spam), then the third model with high precision would be a good fit.

However, if it is more important to correctly identify as many spam emails as possible, then the second model with high recall would be a better choice.

As stated in 2.1.3, In unbalanced datasets, accuracy is not that much of a reliable measure of model performance because it neglects the distribution of the classes. The other performance metrics are important to analyze in this case.