**Introduction**

A very useful machine learning method which, for its simplicity, is incredibly successful in many real world applications is *the Naive Bayes classifier*. I am currently taking a machine learning module as part of my data science college course and this week’s practical work involved a classification problem using the Naive Bayes method. I thought that I’d write a two-part piece about the project and the concepts of probability in order to consolidate my own understanding as well to share my method with others who may find it useful. If the reader wishes to get straight into the Bayes classification problem, please see Part 2 [here](https://wordpress.com/post/sw23993.wordpress.com/1197).

The dataset used in the classification problem is titled “Breast cancer data” and is sourced from Matjaz Zwitter and Milan Soklic from the Institute of Oncology, University Medical Center in Ljubljana, Slovenia (formerly Yugoslavia). It is an interesting dataset which contains categorical data on two classes with nine attributes. The dataset can be found [here](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer). The two outcome classes are “no-recurrence events” and “recurrence events” which refer to whether breast cancer returned to a patient or not having previously being diagnosed  with the disease and treated for it.

Our goal was to use these data to build a Naive Bayes classification model which could predict with good accuracy whether a patient diagnosed with breast cancer is likely to experience a recurrence of the disease based on the attributes. First, I will briefly discuss the basic concepts of probability before moving on to the classification problem in Part 2.

**Fundamentals of Probability**

Probability boils down to working out proportions and performing mathematical operations on them such as addition, multiplication and division. The fact that Naive Bayes classification has been so successful given this simple foundation is truly remarkable. There are a number of rules and concepts involved in probability calculations which one should be aware of in order to understand how the Naive Bayes classifier works. We will discuss these now but first I want to explain independent events, dependent events and mutual exclusiveness.

*Independent events* are those which, upon occurrence, do not affect the probability of another event occurring. An example would be Brazil winning the football World Cup and a hurricane forming over the Atlantic Ocean in September. *Dependent* events are those which, upon occurrence, affect the probability of another event occurring (i.e. they are linked in some way). An example would be getting a cold and going to work the following day. The probability of you going to work can be affected by the occurrence of a cold as you don’t feel very well and would like to not infect your colleagues. When two or more events are *mutually exclusive*, they cannot occur simultaneously. What is the probability of rolling a 6 and a 3 on a single dice roll? Zero. It is impossible. These two outcomes are mutually exclusive.

To calculate the probability of a single event occurring, sometimes called *the observational probability*, you just take the number of times the event occurred and divide it by the total number of processes which occurred that could have lead to that event. For example, let’s say I sprint 100 metres fifty times over the course of a month. I want to know the probability of my time being below 10 seconds based on the previous sprints. If I ran sub-10 second 100 metres on 27 occasions, then the observational probability of me running a sub-10 second 100 metres is simply:

P(<10 second 100 m) = (27 / 50) = 0.54 (54%)

Therefore, I can expect to sprint 100 metres in under 10 seconds about half of the time based on records of the fifty completed sprints. Please note that the P() wrapper is used to denote the probability of some outcome and is used throughout these posts.

**Multiplication Rules for AND Events**

When calculating the probability of two or more events occurring simultaneously, we first consider whether the events are independent or dependent. If they are independent we can use the *simple multiplication rule*:

P(outcome 1 **AND** outcome 2) = P(outcome 1) \* P(outcome 2)

If I were to calculate the probability of Brazil winning the football World Cup and a hurricane forming over the Atlantic Ocean in September, I would use this simple multiplication rule. The two events are independent as the occurrence of one does not affect the other’s chance of occurring. If Brazil have a probability of winning the World Cup of 41% and the probability of a hurricane over the Atlantic Ocean in September is 91%, then we calculate the probability of both occurring:

P(Brazil **AND** Hurricane) = P(Brazil) \* P(Hurricane)

= (0.41) \* (0.91)

= 0.3731 (37%)

If two or more events are dependent, however, we must use the *general multiplication rule.* This formula is always valid, in both cases of independent and dependent events, as we will see.

P(outcome 1 **AND** outcome 2) = P(outcome 1) \* P(outcome 2 | outcome 1)

P(outcome 2 | outcome 1) refers to the *conditional probability* of outcome 2 occurring given outcome 1 has already occurred. One can immediately see how this formula incorporates dependence between the events. If the events were independent, then the conditional probability is irrelevant as one outcome does not influence the chance of the other and P(outcome 2 | outcome 1) is simply P(outcome 2).  The formula just becomes the simple multiplication rule already described.

We can apply the general multiplication rule to a deck of cards example. What would be the probability of drawing a King of any suit and an Ace of any suit from a 52-card deck with just two draws? There are 4 Kings and 4 Aces in a standard deck so we know that drawing an Ace immediately affects our chances of drawing a King as there are fewer cards in the deck after the first draw. We can use the general multiplication formula as follows:

P(Ace **AND** King) = P(Ace) \* P(King | Ace)

= (4 / 52) \* (4 / 51)

= 0.006033183 (0.6%)

Obviously, if two events are mutually exclusive and cannot occur simultaneously, they are disjoint and the multiplication rules cannot be applied. The dice roll example describes such a scenario. The best we can do in such a case is state the probability of either outcome occurring. This brings us to the *the simple addition rule.*

**Addition Rules for OR Events**

When calculating the probability of either one event or the other occurring, we use *the addition rule*. When the outcomes are mutually exclusive, we use the *simple addition* formula:

P(outcome 1 **OR** outcome 2) = P(outcome 1) + P(outcome 2)

Applied to the dice roll example, what is the probability of rolling a 6 or a 3? Both outcomes cannot occur simultaneously. The probability of rolling a 6 is (1 / 6) and the same can be said for rolling a 3. Therefore,

P(6 **OR** 3) = (1 / 6) + (1 / 6) = 0.33 (33%)

However, if the events are not mutually exclusive and can occur simultaneously, we must use the following *general addition* formula which is always valid in both cases of mutual exclusiveness and non-mutual exclusiveness.

P(outcome 1 **OR** outcome 2) = P(outcome 1) + P(outcome 2) – P(outcome 1 **AND** outcome 2)

Applied to the World Cup and hurricane example, this would mean calculating the probabilities of both outcomes as well as the probability of both occurring simultaneously.

P(Brazil) + P(Hurricane) – P(Brazil **AND** Hurricane)

We know that the outcomes are independent, the occurrence of one  does not affect the probability of the other occurring, and we can therefore use the *simple multiplication rule* for two simultaneous events to calculate P(Brazil **AND** Hurricane).

P(Brazil **AND** Hurricane) = (0.41) \* (0.91)

= 0.3731 (37%)

Finally, we can plug this probability into the main formula to get our answer:

P(Brazil **OR** Hurricane) = (0.91) + (0.41) – (0.3731) = 0.9469 (95%)

This makes sense right? Brazil are indeed an awesome football team. However, what drives this probability up is the fact that September is one of the months in the Atlantic hurricane season and you are almost guaranteed to see a hurricane event.

**Summary**

That was a brief introduction to probability and the rules associated with it. I hope it is clear and easy to follow. The important rules to remember are the general forms of the multiplication rule and the addition rule as they are valid in all cases. Using them in the above examples in place of the simple rules still yields the same results. As such, it may be more useful to memorise these general forms. Here they are once more:

P(outcome 1 **AND** outcome 2) = P(outcome 1) \* P(outcome 2 | outcome 1)

P(outcome 1 **OR** outcome 2) = P(outcome 1) + P(outcome 2) – P(outcome 1 **AND** outcome 2)

I have chosen to use “outcome 1” and “outcome 2” instead of the usual “A” and “B” format because I believe it easier to understand Naive Bayes when replacing the letters with meaningful words. Here is how the general rules are typically written:

P(A and B) = P(A) \* P(B | A)

P(A or B) = P(A) + P(B) – P(A and B)

In Part 2 I describe how to implement the Naive Bayes classifier in R and explain how it works based on the fundamentals of probability outlined here.

Following on from [Part 1](https://wordpress.com/post/sw23993.wordpress.com/693) of this two-part post, I would now like to explain how the Naive Bayes classifier works before applying it to a classification problem involving breast cancer data. The dataset is sourced from Matjaz Zwitter and Milan Soklic from the Institute of Oncology, University Medical Center in Ljubljana, Slovenia (formerly Yugoslavia) and the attributes are as follows:

**age:** a series of ranges from 20-29 to 70-79

**menopause:** whether a patient was pre- or post-menopausal upon diagnosis

**tumor.size:** the largest diameter (mm) of excised tumor

**inv.nodes:** the number of axillary lymph nodes which contained metastatic breast cancer

**node.caps:** whether metastatic cancer was contained by the lymph node capsule

**deg.malign:** the histological grade of the tumor (1-3 with 3 = highly abnormal cells)

**breast:** which breast the cancer occurred in

**breast.quad:** region of the breast cancer occurred in (four quadrants with nipple = central)

**irradiat:** whether the patient underwent radiation therapy

Some preprocessing of these data was required as there were some NAs (9 in total). I imputed predicted values using separate Naive Bayes classifiers. The objective here is to attempt to predict, using these attributes, with relatively high accuracy whether a recurrence of breast cancer is likely to occur in patients who were previously diagnosed and treated for the disease. We can pursue this objective by using the Naive Bayes classification method.

**Naive Bayes’ Classification**

Below is the Naive Bayes’ Theorem:

P(A | B) = P(A) \* P(B | A) / P(B)

Which can be derived from the general multiplication formula for AND events:

P(A and B) = P(A) \* P(B | A)

P(B | A) = P(A and B) / P(A)

P(B | A) = P(B) \* P(A | B) / P(A)

If I replace the letters with meaningful words as I have been adopting throughout, the Naive Bayes formula becomes:

P(outcome | evidence) = P(outcome) \* P(evidence | outcome) / P(evidence)

It is with this formula that the Naive Bayes classifier calculates conditional probabilities for a class outcome given prior information or evidence (our attributes in this case). The reason it is termed “naive” is because we assume independence between attributes when in reality they may be dependent in some way. For the breast cancer dataset we will be working with, some attributes are clearly dependent such as age and menopause status while some may or may not be dependent such as histological grade and tumor size.

This assumption allows us to calculate the probability of the evidence by multiplying the individual probabilities of each piece of evidence occurring together using the simple multiplication rule for independent AND events. Another point to note is that this naivety results in probabilities that are not entirely mathematically correct but they are a good approximation and adequate for the purposes of classification. Indeed, the Naive Bayes classifier has proven to be highly effective and is commonly deployed in email spam filters.

**Calculating Conditional Probabilities**

Conditional probabilities are fundamental to the working of the Naive Bayes formula. Tables of conditional probabilities must be created in order to obtain values to use in the Naive Bayes algorithm. The R package e1071 contains a very nice function for creating a Naive Bayes model:

library(e1071)

model <- naiveBayes(class ~ ., data = breast\_cancer)

class(model)

summary(model)

print(model)

The model has class “naiveBayes” and the summary tells us that the model provides a-priori probabilities of no-recurrence and recurrence events as well as conditional probability tables across all attributes. To examine the conditional probability tables just print the model.

One of our tasks for this assignment was to create code which would give us the same conditional probabilities as those output by the naiveBayes() function. I went about this in the following way:

tbl\_list <- sapply(breast\_cancer[-10], table, breast\_cancer[ , 10])

tbl\_list <- lapply(tbl\_list, t)

cond\_probs <- sapply(tbl\_list, function(x) {

  apply(x, 1, function(x) {

    x / sum(x) }) })

cond\_probs <- lapply(cond\_probs, t)

print(cond\_probs)

The first line of code uses the sapply function to loop over all attribute variables in the dataset and create tables against the class attribute. I then used the lapply function to transpose all tables in the list so the rows represented the class attribute.

To calculate conditional probabilities for each element in the tables, I used sapply, lapply and anonymous functions. I had to transpose the output in order to get the same structure as the naiveBayes model output. Finally, I printed out my calculated conditional probabilities and compared them with the naiveBayes output to validate the calculations.

**Applying the Naive Bayes’ Classifier**

So I’ve explained (hopefully reasonably well) how the Naive Bayes classifier works based on the fundamental rules of probability. Now it’s time to apply the model to the data. This is easily done in R by using the predict() function.

preds <- predict(model, newdata = breast\_cancer)

You will see that I have trained the model using the entire dataset and then made predictions on the same dataset. In our assignment we were asked to train the model and test it on the dataset, treating the dataset as an unlabeled test set. This is unconventional as the training set and test set are then identical but I believe the assignment was intended to just test our understanding of how the method works. In practice, one would use a training set for the model to learn from and a test set to assess model accuracy.

If one outcome class is more abundant in the dataset, as is the case with the breast cancer data (no-recurrence: 201, recurrence: 85), the data is unbalanced. This is okay for a generative Naive Bayes model as you want your model to learn from real-world events and to capture the truth. Manipulating the data to achieve less skew would be dangerous.

Applying the model to the data gives the following confusion matrix from which a model accuracy of 75% can be calculated:

  conf\_matrix <- table(preds, breast\_cancer$class)

This post has only scraped the surface of classification methods in machine learning but has been a useful revision for myself and perhaps it may help others new to the Naive Bayes classifier. Please feel free to comment and correct any errors that may be present.

## The Best Algorithms are the Simplest

The field of data science has progressed from simple linear regression models to complex ensembling techniques but the most preferred models are still the simplest and most interpretable. Among them are regression, logistic, trees and naive bayes techniques. Naive Bayes algorithm, in particular is a logic based technique which is simple yet so powerful that it is often known to outperform complex algorithms for very large datasets. Naive bayes is a common technique used in the field of medical science and is especially used for cancer detection. This article explains the underlying logic behind naive bayes algorithm and example implementation.

## How Probability defines Everything

We calculate probability as the proportion of cases where an event happens and call it the probability of the event. Just as there is probability for a single event, we have probability of a group of events as the proportion of cases where the group of events occur together. Another concept in probability is calculating the occurrence of events in a particular sequence, that is, if it is known that something has already happened, what will be the probability that another event happens after that. By logic, one can understand that we are narrowing down our scope to only the case when that something has already happened and then calculating the proportion of cases where our second event occurs. To represent it mathematically, If A is the first event and B is the second event, then P(B|A) is our desired probability of calculating probability of event A after occurrence of event B, P(A n B) is probability of the two events occurring together

P(B | A) = P(B) \* P(A | B) / P(A)

This is the foundation pillar for Naive bayes algorithm. Owing to this, Naive Bayes can handle different kind of events which are differentiated by the probabilities of event separately, that is , P(B) and conditional probability P(B|A). If the two probabilities are same, then it means that the occurrence of event A had no effect on event B and the events are known as independent events. If the conditional probability becomes zero, then it means the occurrence of event A implies that event B cannot occur. If the reverse is also true, then the events are known as mutually exclusive events and the occurrence of only one of the events at a time is possible. All other cases are classified as dependent events where the conditional probability can be either lower or higher than the original. In real life, every coin toss is independent of all other coin tosses made previously and thus coin tosses are independent. The outcome of a single coin toss is composed of mutually exclusive events. We cannot have a head and the tails at the same time. When we consider runs of multiple coin tosses, we are talking about dependent events. For a combination of three coin tosses, the final outcome is dependent of the first, second as well as the third coin toss.

**How do we Calculate these Probabilities?**

It is easy to calculate the probability of a single event. It equates to the number of cases when the event occurs divided by the total number of possible cases. For instance, the probability of a 6 in a single six-faced die roll is ⅙ if all the sides have equal chance of coming. However, one needs to be careful when calculating probabilities of two or more events. Simply knowing the probability of each event separately is not enough to calculate the probability of multiple events happening. If we additionally know that the events are independent, then the probability of them occurring together is the multiplication of each event separately.

We denote this mathematically as follows:  
P(A and B)=P(A)\*P(B) – For independent events

As I already described, each coin toss is independent of other coin tosses. So the probability of having a Heads and a Heads combination in two coin tosses is  
P(Heads-Heads Combo)=P(Heads in first throw)\*P(Heads in second throw)=½ \* ½ = ¼

If the events are not independent, we can use the probability of any one event multiplied by the probability of second event after the first has happened

P(A and B)=P(A)\*P(B|A) – For dependent events

An example of dependent events can be drawing cards without replacement. If you want to know that the two cards drawn are King and Queen then we know that the probability of the first event is dependent of 52 cards whereas the probability of the second event is dependent on 51 cards.

Thus, P(King and Queen)=P(King)\*P(Queen|King)

Here, P(King) is 4/52. After a King is drawn, there are 4 queens out of 51 cards.  
So, P(Queen|King) is 4/51  
P(King and Queen)=4/52\*4/51=~0.6%

This is known as general multiplication rule. It also applies to the independent events scenario but since the events are independent, P(B|A) becomes equal to P(B)

The third case is for mutually exclusive events. If the events are mutually exclusive, we know that only one of the events can occur at a time. So the probability of the two events occurring together is zero. We are sometimes interested in probability of one of the events occuring and it is the sum of the individual probabilities in this scenario.

P(A OR B)=P(A)+P(B) – for mutually exclusive events

If we’re talking about a single six faced fair die throw, the probability of any two numbers occurring together is zero. In this case the probability of any prime number occuring is the sum of occurrence of each prime number. In this case P(2)+P(3)+P(5)

Had the events not been mutually exclusive, the probability of one of the events would have counted the probability of both events coming together twice. Hence we subtract this probability.  
P(A OR B)=P(A)+P(B)-P(A AND B) – for events which are not mutually exclusive

In a single six faced fair dice throw, the probability of throwing a multiple of 2 or 3 describes a scenario of events which are not mutually exclusive since 6 is both a multiple of 2 and 3 and is counted twice.

Thus,  
P(multiple of 2 or 3)=P(Multiple of 2)+P(Multiple of 3)- P(Multiple of 2 AND 3)  
=P(2,4,6)+P(3,6)-P(6)=3/6 + 2/6 -1/6 = 4/6 =2/3

This is known as general addition rule and similar to the multiplication rule, it also applies to the mutually exclusive events scenario but in that case, P(A AND B) is zero.

This is all we need to understand how Naive Bayes algorithm works. It takes into account all such scenarios and learns accordingly. Let’s get our hands dirty with a sample dataset.

## Naive Bayes – a Not so Naive Algorithm

The reason that Naive Bayes algorithm is called Naive is not because it is simple or stupid. It is because the algorithm makes a very strong assumption about the data having features independent of each other while in reality, they may be dependent in some way. In other words, it assumes that the presence of one feature in a class is completely unrelated to the presence of all other features. If this assumption of independence holds, Naive Bayes performs extremely well and often better than other models. Naive Bayes can also be used with continuous features but is more suited to categorical variables. If all the input features are categorical, Naive Bayes is recommended. However, in case of numeric features, it makes another strong assumption which is that the numerical variable is normally distributed.

R supports a package called ‘e1071’ which provides the naive bayes training function. For this demonstration, we will use the classic titanic dataset and find out the cases which naive bayes can identify as survived.

The Titanic dataset in R is a table for about 2200 passengers summarised according to four factors – economic status ranging from 1st class, 2nd class, 3rd class and crew; gender which is either male or female; Age category which is either Child or Adult and whether the type of passenger survived. For each combination of Age, Gender, Class and Survived status, the table gives the number of passengers who fall into the combination. We will use the Naive Bayes Technique to classify such passengers and check how well it performs.  
As we know, Bayes theorem is based on conditional probability and uses the formula

P(A | B) = P(A) \* P(B | A) / P(B)

We now know how this conditional probability comes from multiplication of events so if we use the general multiplication rule, we get another variation of the theorem that is, using P(A AND B) = P(A) \* P(B | A), we can obtain the value for conditional probability: P(B | A) = P(A AND B) / P(A) which is the variation of Bayes theorem.

Since P(A AND B) also equals P(B) \* P(A | B), we can substitute it and get back the original formula  
P(B | A) = P(B) \* P(A | B) / P(A)  
Using this for each of the features among Age, Gender and Economic status, Naive Bayes algorithm will calculate the conditional probability of survival of the combination.

#Getting started with Naive Bayes

#Install the package

#install.packages(“e1071”)

#Loading the library

library(e1071)

?naiveBayes #The documentation also contains an example implementation of Titanic dataset

#Next load the Titanic dataset

data(“Titanic”)

#Save into a data frame and view it

Titanic\_df=as.data.frame(Titanic)

We see that there are 32 observations which represent all possible combinations of Class, Sex, Age and Survived with their frequency. Since it is summarised, this table is not suitable for modelling purposes. We need to expand the table into individual rows. Let’s create a repeating sequence of rows based on the frequencies in the table

#Creating data from table

repeating\_sequence=rep.int(seq\_len(nrow(Titanic\_df)), Titanic\_df$Freq) #This will repeat each combination equal to the frequency of each combination

#Create the dataset by row repetition created

Titanic\_dataset=Titanic\_df[repeating\_sequence,]

#We no longer need the frequency, drop the feature

Titanic\_dataset$Freq=NULL

The data is now ready for Naive Bayes to process. Let’s fit the model

|  |  |
| --- | --- |
| 1  2  3  4 | #Fitting the Naive Bayes model  Naive\_Bayes\_Model=naiveBayes(Survived ~., data=Titanic\_dataset)  #What does the model say? Print the model summary  Naive\_Bayes\_Model |

Naive Bayes Classifier for Discrete Predictors

Call:

naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:

Y

      No      Yes

0.676965 0.323035

Conditional probabilities:

     Class

Y          1st          2nd         3rd         Crew

  No    0.08187919  0.11208054  0.35436242  0.45167785

  Yes   0.28551336  0.16596343  0.25035162  0.29817159

     Sex

Y          Male         Female

  No    0.91543624  0.08456376

  Yes   0.51617440  0.48382560

     Age

Y         Child         Adult

  No    0.03489933  0.96510067

  Yes   0.08016878  0.91983122

The model creates the conditional probability for each feature separately. We also have the a-priori probabilities which indicates the distribution of our data. Let’s calculate how we perform on the data.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | #Prediction on the dataset  NB\_Predictions=predict(Naive\_Bayes\_Model,Titanic\_dataset)  #Confusion matrix to check accuracy  table(NB\_Predictions,Titanic\_dataset$Survived)  NB\_Predictions      No      Yes                  No      1364    362                  Yes     126     349 |

We have the results! We are able to classify 1364 out of 1490 “No” cases correctly and 349 out of 711 “Yes” cases correctly. This means the ability of Naive Bayes algorithm to predict “No” cases is about 91.5% but it falls down to only 49% of the “Yes” cases resulting in an overall accuracy of 77.8%

**Conclusion: Can we Do any Better?**

Naive Bayes is a parametric algorithm which implies that you cannot perform differently in different runs as long as the data remains the same. We will, however, learn another implementation of Naive Bayes algorithm using the ‘mlr’ package. Assuming the same session is going on for the readers, I will install and load the package and start fitting a model

|  |  |
| --- | --- |
| 1  2  3  4  5 | #Getting started with Naive Bayes in mlr  #Install the package  #install.packages(“mlr”)  #Loading the library  library(mlr) |

The mlr package consists of a lot of models and works by creating tasks and learners which are then trained. Let’s create a classification task using the titanic dataset and fit a model with the naive bayes algorithm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | #Create a classification task for learning on Titanic Dataset and specify the target feature  task = makeClassifTask(data = Titanic\_dataset, target = "Survived")  #Initialize the Naive Bayes classifier  selected\_model = makeLearner("classif.naiveBayes")  #Train the model  NB\_mlr = train(selected\_model, task) |

The summary of the model which was printed in e3071 package is stored in learner model. Let’s print it and compare

#Read the model learned

NB\_mlr$learner.model

Naive Bayes Classifier for Discrete Predictors

Call:

naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:

Y

      No      Yes

0.676965 0.323035

Conditional probabilities:

     Class

Y               1st         2nd         3rd         Crew

    No      0.08187919  0.11208054  0.35436242  0.45167785

    Yes     0.28551336  0.16596343  0.25035162  0.29817159

     Sex

Y               Male        Female

     No     0.91543624  0.08456376

    Yes     0.51617440  0.48382560

     Age

Y           Child       Adult

    No      0.03489933  0.96510067

    Yes     0.08016878  0.91983122

The a-priori probabilities and the conditional probabilities for the model are similar to the one calculated by e3071 package as was expected. This means that our predictions will also be the same.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | #Predict on the dataset without passing the target feature  predictions\_mlr = as.data.frame(predict(NB\_mlr, newdata = Titanic\_dataset[,1:3]))    ##Confusion matrix to check accuracy  table(predictions\_mlr[,1],Titanic\_dataset$Survived)          No      Yes    No    1364    362    Yes   126     349 |

As we see, the predictions are exactly same. The only way to improve is to have more features or more data. Perhaps, if we have more features such as the exact age, size of family, number of parents in the ship and siblings then we may arrive at a better model using Naive Bayes.  
In essence, Naive Bayes has an advantage of a strong foundation build and is very robust. I know of the ‘caret’ package which also consists of Naive Bayes function but it will also give us the same predictions and probability.

**Here is the Complete Code (used in this article):**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | #Getting started with Naive Bayes  #Install the package  #install.packages(“e1071”)  #Loading the library  library(e1071)  ?naiveBayes #The documentation also contains an example implementation of Titanic dataset  #Next load the Titanic dataset  data("Titanic")  #Save into a data frame and view it  Titanic\_df=as.data.frame(Titanic)  #Creating data from table  repeating\_sequence=rep.int(seq\_len(nrow(Titanic\_df)), Titanic\_df$Freq) #This will repeat each combination equal to the frequency of each combination    #Create the dataset by row repetition created  Titanic\_dataset=Titanic\_df[repeating\_sequence,]  #We no longer need the frequency, drop the feature  Titanic\_dataset$Freq=NULL    #Fitting the Naive Bayes model  Naive\_Bayes\_Model=naiveBayes(Survived ~., data=Titanic\_dataset)  #What does the model say? Print the model summary  Naive\_Bayes\_Model    #Prediction on the dataset  NB\_Predictions=predict(Naive\_Bayes\_Model,Titanic\_dataset)  #Confusion matrix to check accuracy  table(NB\_Predictions,Titanic\_dataset$Survived)    #Getting started with Naive Bayes in mlr  #Install the package  #install.packages(“mlr”)  #Loading the library  library(mlr)    #Create a classification task for learning on Titanic Dataset and specify the target feature  task = makeClassifTask(data = Titanic\_dataset, target = "Survived")    #Initialize the Naive Bayes classifier  selected\_model = makeLearner("classif.naiveBayes")    #Train the model  NB\_mlr = train(selected\_model, task)    #Read the model learned  NB\_mlr$learner.model    #Predict on the dataset without passing the target feature  predictions\_mlr = as.data.frame(predict(NB\_mlr, newdata = Titanic\_dataset[,1:3]))    ##Confusion matrix to check accuracy  table(predictions\_mlr[,1],Titanic\_dataset$Survived)<span style="font-family: 'Noto Serif', serif"><span>  </span></span> |