# Naïve Baye's Algorithm from Scratch.

Name: Bishal Neupane

Partner's Name : Saugat Gyawali

The code written by me can be mentioned as below:

```
#include <iostream>

using namespace std;

#include <iostream>
#include <conio.h>
#include <string>
#include <vector>
#include <fstream>
#include <algorithm>
#include <cmath>
#include <chrono>

using namespace chrono;

vector<int> pclass;
vector<int> survived;
vector<int> sex;
vector<double> age;

// Function for reading file content and printing details of file
void OpeningFileForReading(vector<int> &pclass, vector<int> &survived,
vector<int> &sex, vector<double> &age)
{
    fstream inFs;
    string pclass_in, survived_in, line;
    string sex_in, age_in;
    string row_no;

    cout << "Opening file titanic_project.csv" << endl;
    inFs.open("C:/Users/bisha/OneDrive/Desktop/Fall 2022/Machine
Learning/Programming/titanic_project.csv", ios::in);
    if (inFs.is_open())
    {
        cout << "titanic_project.csv opened" << endl;
    }
    if (!inFs.is_open())
```

```
    {

        cout << "Could not open file titanic_project.csv" << endl;
    }

    cout << "Reading line 1 " << endl;

    getline(inFs, line);

    cout << "heading: " << line << endl;
    int numObservations = 0;
    while (inFs.good())
    {
        getline(inFs, row_no, ',');
        getline(inFs, pclass_in, ',');
        getline(inFs, survived_in, ',');
        getline(inFs, sex_in, ',');
        getline(inFs, age_in, '\n');
        pclass.push_back(stod(pclass_in));
        survived.push_back(stod(survived_in));
        sex.push_back(stod(sex_in));
        age.push_back(stod(age_in));
        numObservations++;
    }

    cout << "new length: " << pclass.size() << endl;
    cout << "Closing file titanic_project.csv" << endl;
    inFs.close();
    cout << "Number of records: " << numObservations << endl;
}

// Function to calculate zero or 1  in particular column
int zeroOrOneCount(int toCount, vector<int> column)
{
    int nRows = 800;
    int Count = 0;
    for (int i = 0; i < nRows; i++)
    {
        if (column[i] == toCount)
        {
            Count++;
        }
    }
    return Count;
}
```

```cpp
// Function to calculate Aprior
vector<double> calculateAprior(vector<int> survived)
{
    vector<double> aprior;
    int nRows = 800;
    int zeroCount = zeroOrOneCount(0, survived);
    int oneCount = zeroOrOneCount(1, survived);

    // cout<< zeroCount << "," << oneCount<< endl;

    aprior.push_back((double)zeroCount / nRows);
    aprior.push_back((double)oneCount / nRows);
    return aprior;
}

//  Function to calculate likelihood for pclass
vector<vector<double>> likHoodPclass(vector<int> survived, vector<int> pclass)
{
    vector<vector<double>> lhPclass(2, vector<double>(3, 0));
    int zeroCount = zeroOrOneCount(0, survived);
    int oneCount = zeroOrOneCount(1, survived);

    for (int i = 0; i < 800; i++)
    {

        if (survived[i] == 0 && pclass[i] == 1)
        {
            lhPclass[0][0]++;

            // cout << sum << endl;
        }
        if (survived[i] == 0 && pclass[i] == 2)
        {
            lhPclass[0][1]++;
        }
        if (survived[i] == 0 && pclass[i] == 3)
        {
            lhPclass[0][2] = lhPclass[0][2] + 1;
        }
        if (survived[i] == 1 && pclass[i] == 1)
        {
            lhPclass[1][0]++;
        }
        if (survived[i] == 1 && pclass[i] == 2)
```

```
        {
            lhPclass[1][1]++;
        }
        if (survived[i] == 1 && pclass[i] == 3)
        {
            lhPclass[1][2]++;
        }
    }

    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (i == 0)
            {

                lhPclass[i][j] = lhPclass[i][j] / zeroCount;
            }

            if (i == 1)
            {
                lhPclass[i][j] = lhPclass[i][j] / oneCount;
            }
        }
    }

    return lhPclass;
}

// Likelihood for sex
vector<vector<double>> likHoodSex(vector<int> survived, vector<int> sex)
{
    vector<vector<double>> lhSex(2, vector<double>(2, 0));
    int zeroCount = zeroOrOneCount(0, survived);
    int oneCount = zeroOrOneCount(1, survived);

    for (int i = 0; i < 800; i++)
    {

        if (survived[i] == 0 && sex[i] == 0)
        {
            lhSex[0][0]++;

            // cout << sum << endl;
        }
```

```
        if (survived[i] == 0 && sex[i] == 1)
        {
            lhSex[0][1]++;
        }

        if (survived[i] == 1 && sex[i] == 0)
        {
            lhSex[1][0]++;
        }
        if (survived[i] == 1 && sex[i] == 1)
        {
            lhSex[1][1]++;
        }
    }

    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            if (i == 0)
            {

                lhSex[i][j] = lhSex[i][j] / zeroCount;
            }

            if (i == 1)
            {
                lhSex[i][j] = lhSex[i][j] / oneCount;
            }
        }
    }

    return lhSex;
}

// Mean for age
vector<double> calculateMeanAge(vector<double> age, vector<int> survived)
{
    vector<double> mean;
    int zeroCount = zeroOrOneCount(0, survived);
    int oneCount = zeroOrOneCount(1, survived);
    double age0 = 0;
    double age1 = 0;
    for (int i = 0; i < 800; i++)
    {
```

```
        if (survived[i] == 0)
        {
            age0 = age[i] + age0;
        }
        if (survived[i] == 1)
        {
            age1 = age[i] + age1;
        }
    }
    mean.push_back(age0 / zeroCount);
    mean.push_back(age1 / oneCount);
    return mean;
}

// calculating variance for age
vector<double> calculateVarianceAge(vector<double> age, vector<int> survived)
{
    vector<double> var;
    int zeroCount = zeroOrOneCount(0, survived);
    int oneCount = zeroOrOneCount(1, survived);
    double var0 = 0;
    double var1 = 0;
    double mean0 = calculateMeanAge(age, survived)[0];
    double mean1 = calculateMeanAge(age, survived)[1];
    for (int i = 0; i < 800; i++)
    {
        if (survived[i] == 0)
        {
            var0 = var0 + (mean0 - age[i]) * (mean0 - age[i]);
        }
        if (survived[i] == 1)
        {
            var1 = var1 + (mean1 - age[i]) * (mean1 - age[i]);
        }
    }
    var.push_back(var0 / zeroCount);
    var.push_back(var1 / oneCount);
    return var;
}

// calculating age likelihood
double calculateAgeLh(double age, double mean, double var)
{
    double pow = ((age - mean) * (age - mean)) / (2 * var);
    return 1 / sqrt(2 * 22 / 7 * var) * exp(-pow);
```

```
}

// calculating raw probability
vector<double> calculateRawProb(int pclass1, int sex1, double age1)
{
    vector<vector<double>> lhPclass = likHoodPclass(survived, pclass);
    vector<vector<double>> lhsex = likHoodSex(survived, sex);
    vector<double> apriori = calculateAprior(survived);
    vector<double> result(2, 0);
    double num_s = 0.0;
    double num_p = 0.0;
    double denominator = 0.0;
    vector<double> mean = calculateMeanAge(age, survived);
    vector<double> var = calculateVarianceAge(age, survived);

    num_s = lhPclass[1][pclass1] * lhsex[1][sex1] * apriori[1] *
calculateAgeLh(age1, mean[1], var[1]);
    num_p = lhPclass[0][pclass1] * lhsex[0][sex1] * apriori[0] *
calculateAgeLh(age1, mean[0], var[0]);
    denominator = num_s + num_p;
    result[0] = (num_p / denominator);
    result[1] = (num_s / denominator);

    return result;
}

// Calculating accuracy
int calculateAccuracy(vector<int> prediction)
{
    int trueResult = 0;
    for (int i = 800; i < 1046; i++)
    {
        if (survived[i] == prediction[i - 800])
        {

            trueResult++;
        }
    }
    return trueResult;
}

// calculating sensitivity
double calculateSensitivity(vector<int> prediction, vector<int> survived)
{
    double truePositive = 0;
```

```
    double falseNegative = 0;
    for (int i = 0; i < prediction.size(); i++)
    {
        if (survived[800 + i] == 1 && prediction[i] == 1)
        {
            truePositive++;
        }
        else if (survived[800 + i] == 1 && prediction[i] == 0)
        {
            falseNegative++;
        }
    }

    double sensitivity = truePositive / (truePositive + falseNegative);

    return sensitivity;
}

// calculating specificity
double calculateSpecificity(vector<int> prediction, vector<int> survived)
{
    double trueNegative = 0;
    double falsePositive = 0;
    for (int i = 0; i < 246; i++)
    {
        if (survived[800 + i] == 0 && prediction[i] == 0)
        {
            trueNegative++;
        }
        else if (survived[800 + i] == 0 && prediction[i] == 1)
        {
            falsePositive++;
        }
    }

    return (trueNegative / (trueNegative + falsePositive));
}

int main()
{

    OpeningFileForReading(pclass, survived, sex, age);
    auto startTime = steady_clock::now();
    vector<double> aprior = calculateAprior(survived);
    cout << "-------------------\n";
```

```
    cout << "\nAPrior \n";
    cout << "-------------------\n";
    cout << aprior[0] << ", " << aprior[1] << endl;

    vector<vector<double>> lhpclass = likHoodPclass(survived, pclass);
    cout << "----------------------\n";
    cout << "Likelihood for pclass \n";
    cout << "----------------------\n";
    cout << "Likelihood values for p(pclass|survived):\n";
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << lhpclass[i][j] << " ";
        }
        cout << endl;
    }

    vector<vector<double>> lhSex = likHoodSex(survived, sex);
    cout << "----------------------\n";
    cout << "Likelihood for sex \n";
    cout << "----------------------\n";
    cout << "Likelihood values for p(psex|survived):\n";
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            cout << lhSex[i][j] << " ";
        }
        cout << endl;
    }
    auto endTime = steady_clock::now();
    // Printing test result
    double average0 = 0;
    double average1 = 0;
    vector<int> prediction;
    cout << "\n\nPrinting test result";
    cout << "\n------------------------\n\n";
    for (int i = 800; i < 1046; i++)
    {

        vector<double> rawResult = calculateRawProb(pclass[i], sex[i], age[i]);
        cout << rawResult[0] << " " << rawResult[1] << endl;
        if (rawResult[0] > rawResult[1])
        {
```

```
            prediction.push_back(0);
        }
        else
        {

            prediction.push_back(1);
        }
    }
    cout << "prediction size = " << prediction.size() << endl;
    // for(int i=0; i < prediction.size(); i++){
    //     cout << "pred = " << prediction[i] << "survival = " << survived[800-i]
<< endl;
    // }

    cout << "\n---------------------------\n";
    cout << "Accuracy = " << (double)calculateAccuracy(prediction) / 246 << endl;
    double sensitivity = calculateSensitivity(prediction, survived);
    cout << "Sensitivity = " << sensitivity << endl;
    double specificity = calculateSpecificity(prediction, survived);
    cout << "Specificity = " << specificity << endl;
    cout << "-------------------------------------\n";
    duration<double> elapsed_time = endTime - startTime;
    cout << "Time taken for training: " << elapsed_time.count() << "s\n\n";
    cout << "-----------------------------------------\n\n";
}
```

## Output:

```
Opening file titanic_project.csv
titanic_project.csv opened
Reading line 1
heading: "","pclass","survived","sex","age"
new length: 1046
Closing file titanic_project.csv
Number of records: 1046
--------------------

APrior
--------------------
0.61, 0.39
----------------------
Likelihood for pclass
----------------------
Likelihood values for p(pclass|survived):
0.172131 0.22541 0.602459
0.416667 0.262821 0.320513
----------------------
Likelihood for sex
----------------------
Likelihood values for p(psex|survived):
0.159836 0.840164
0.679487 0.320513


Printing test result
------------------------
```

```
Printing test result
------------------------

0.278754 0.721246
0.894082 0.105918
0.782415 0.217585
0.390183 0.609817
0.261872 0.738128
0.292864 0.707136
0.811565 0.188435
0.777628 0.222372
0.80084 0.19916
0.890757 0.109243
0.811537 0.188463
0.812662 0.187338
0.236337 0.763663
0.248999 0.751001
0.422538 0.577462
0.247329 0.752671
0.409861 0.590139
0.419012 0.580988
0.422538 0.577462
0.79709 0.20291
0.882319 0.117681
0.283057 0.716943
0.223149 0.776851
0.806626 0.193374
0.79955 0.20045
0.265496 0.734504
0.794183 0.205817
```

```
0.811537 0.188463
0.812662 0.187338
0.236337 0.763663
0.248999 0.751001
0.422538 0.577462
0.247329 0.752671
0.409861 0.590139
0.419012 0.580988
0.422538 0.577462
0.79709 0.20291
0.882319 0.117681
0.283057 0.716943
0.223149 0.776851
0.806626 0.193374
0.79955 0.20045
0.265496 0.734504
0.794183 0.205817
0.826185 0.173815
0.792789 0.207211
0.26042 0.73958
0.878833 0.121167
0.809316 0.190684
0.883976 0.116024
0.883154 0.116846
0.808328 0.191672
0.795556 0.204444
0.796235 0.203765
0.375386 0.624614
0.39223 0.60777
0.795556 0.204444
```

```
0.878833 0.121167
0.809316 0.190684
0.883976 0.116024
0.883154 0.116846
0.808328 0.191672
0.795556 0.204444
0.796235 0.203765
0.375386 0.624614
0.39223 0.60777
0.795556 0.204444
0.823565 0.176435
0.804589 0.195411
0.228978 0.771022
0.254269 0.745731
0.230193 0.769807
0.230468 0.769532
0.767407 0.232593
0.253952 0.746048
0.282642 0.717358
0.877 0.123
0.887875 0.112125
0.789936 0.210064
0.39223 0.60777
0.786995 0.213005
0.218318 0.781682
0.788157 0.211843
0.788476 0.211524
0.271323 0.728677
0.79955 0.20045
0.813046 0.186954
```

```
0.809316 0.190684
0.381825 0.618175
0.813741 0.186259
0.257207 0.742793
0.791373 0.208627
0.258819 0.741181
0.794969 0.205031
0.228441 0.771559
0.262011 0.737989
0.808163 0.191837
0.857067 0.142933
0.228441 0.771559
0.221403 0.778597
0.223149 0.776851
0.407975 0.592025
0.272834 0.727166
0.357544 0.642456
0.818868 0.181132
0.791373 0.208627
0.806626 0.193374
0.252311 0.747689
0.246226 0.753774
0.803947 0.196053
0.886356 0.113644
0.798239 0.201761
0.27237 0.72763
0.277697 0.722303
0.784489 0.215511
0.442026 0.557974
0.80084 0.19916
```

```
prediction size = 246

---------------------------
Accuracy = 0.784553
Sensitivity = 0.695652
Specificity = 0.862595
-------------------------------------
Time taken for training: 0.004238s

-------------------------------------
```

Output metrics on R:

```
               Accuracy : 0.7805
                 95% CI : (0.7235, 0.8306)
    No Information Rate : 0.5325
    P-Value [Acc > NIR] : 6.353e-16

                  Kappa : 0.5553

 Mcnemar's Test P-Value : 0.04123

            Sensitivity : 0.8550
            Specificity : 0.6957
         Pos Pred Value : 0.7619
         Neg Pred Value : 0.8081
             Prevalence : 0.5325
         Detection Rate : 0.4553
```

## Analyzing the result of my algorithm for the titanic data:

The accuracy of my algorithm for naïve bayes which was written from scratch in C++ is about 78 percentage which seems to be pretty good. It means that 78 of the prediction made by my algorithm was correct out of 100 predictions. Sensitivity means the ratio of true positive and sum of true positive and false negative. Sensitivity is about 0.69. Also, specificity is the ratio of true negative to the sum of true negative and false positive. Specificity for my algorithm is 0.86. In general, sensitivity is the portion of true positive predicted by the model while specificity is the portion of true negative correctly predicted by the model. It measures the performance of the model. These metrics shows that my model is performing well for this dataset. The time taken to train the algorithm is about 0.004238s which is measured by the chrono but the time might vary by little bit depending upon the machine you are running and other factors.