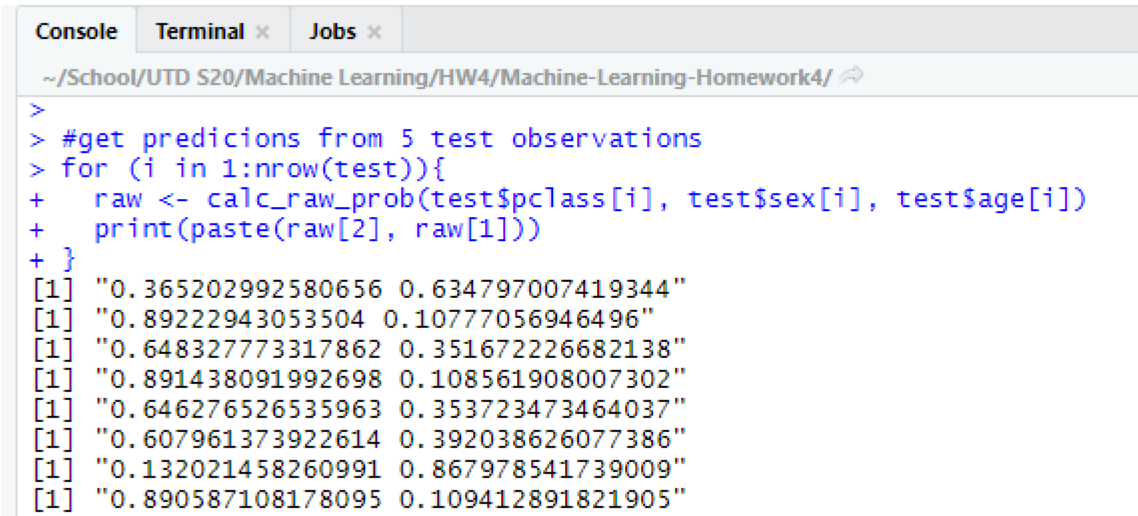
Mariana Castaneda Roiz, Emanuel Rios
Mazidi
CS 4375
02/16/2020

Naïve Bayes

In R:

We began by loading the titanic data and splitting into test and train sets. The train set being the first 900 instances and test being the remaining. We ran Naïve Bayes on the data using the e1071 package first to get base results. Then, following the book on Naïve Bayes and the code on git-hub we adapted the Naïve Bayes from scratch to work on our data. The way the algorithm works is explained in the C++ section of Naïve Bayes.



```
Console   Terminal ×   Jobs ×
~/School/UTD S20/Machine Learning/HW4/Machine-Learning-Homework4/
>
> #get predicions from 5 test observations
> for (i in 1:nrow(test)){
+   raw <- calc_raw_prob(test$pclass[i], test$sex[i], test$age[i])
+   print(paste(raw[2], raw[1]))
+ }
[1] "0.365202992580656 0.634797007419344"
[1] "0.89222943053504 0.10777056946496"
[1] "0.648327773317862 0.351672226682138"
[1] "0.891438091992698 0.108561908007302"
[1] "0.646276526535963 0.353723473464037"
[1] "0.607961373922614 0.392038626077386"
[1] "0.132021458260991 0.867978541739009"
[1] "0.890587108178095 0.109412891821905"
```

Results from the Naïve Bayes from scratch in R.

In C++:

The C++ code for Naïve Bayes uses the Armadillo library as well in order to facilitate the programming. All the metrics such as likelihood matrices and raw probabilities are identical in the R script as the C++ code. The program begins by reading the data from the CSV and splitting it into test and train.

```
int main(){

    vector<Passenger> passengers = readCSVData();
    vector<Passenger> test;
    vector<Passenger> train;
    //get the first 900 instances as train subjects
    for(int i=0; i<900; i++){
        train.push_back(passengers[i]);
    }
    //train
    for(int i=900; i<passengers.size(); i++){
        test.push_back(passengers[i]);
    }
```

The program then gets the counts and priors for the people who survived and perished. After that we’re able to compute the likelihoods of both class and sex. This is done by counting survived and perished for each class and dividing by the total count in the train data.

```cpp
mat get_likelihood_pclass(vector<Passenger>& pass, mat count){
    mat likelihood(2,3); likelihood.zeros();
    for(int sv =0; sv<2; sv++){
        for(int pc=0; pc<3; pc++){
            int numSurvived = 0;
            for(auto p: pass){
                if(p.survived==sv and p.pclass==(pc+1)){
                    numSurvived++;
                }
            }
            likelihood(sv, pc)  = numSurvived/count(0, sv);
        }
    }
    return likelihood;
}

mat get_likelihood_sex(vector<Passenger>& pass, mat count){
    mat likelihood(2,2); likelihood.zeros();
    for(int sv =0; sv<2; sv++){
        for(int sx=0; sx<2; sx++){
            int numSurvived = 0;
            for(auto p: pass){
                if(p.survived==sv and p.sex==sx){
                    numSurvived++;
                }
            }
            likelihood(sv, sx)  = numSurvived/count(0, sv);
        }
    }
    return likelihood;
}
```

We then wrote a function that returns the mean and variances of the continuous variable age. Those numbers can then be plugged into a probability density function to get the likelihood of a new age value.

```cpp
const double pi = 3.14159265358979323846;
//function to calculate age likelihood
//run like this: calc_age_lh(6, 25.9, 138)
double calc_age_lh(double age, double mean, double var){
    return 1/sqrt(2*pi*var)*exp(-(pow(age-mean, 2)/(2*var)));
}
```

Function used to calculate the age likelihood.

Finally, we write the function that calculates the raw probabilities for survived and perished. The numerator for perished is the product of the likelihoods of perishing for each class and the prior of perishing. The denominator for both is the total probability of perishing and surviving. The numerator for survived is the same as perished save for the likelihoods and prior being for surviving.

```
//function used to calculate raw probabilities
mat calc_raw_prob(int pclass, int sex, double age, mat& lh_pclass, mat&lh_sex, mat&age_mean_var, mat&apriori){
    mat raw_prob(1,2); raw_prob.zeros();

    pclass-=1;//for indexing purposes
    double num_s = lh_pclass(1, pclass)*lh_sex(1, sex)*apriori(0, 1)*calc_age_lh(age, age_mean_var(0,1), age_mean_var(1,1));
    double num_p = lh_pclass(0, pclass)*lh_sex(0, sex)*apriori(0, 0)*calc_age_lh(age, age_mean_var(0,0), age_mean_var(1,0));
    double denominator =    num_s + num_p;
    raw_prob(0,1) = num_s / denominator;
    raw_prob(0,0) = num_p / denominator;

    return raw_prob;
}
```

We then run that function for each instance in the test set to get their raw probabilities. The values we got are identical to the ones from the R script.

```
PS C:\Users\emanu\Documents\School\U
   0.3652   0.6348

   0.8922   0.1078

   0.6483   0.3517

   0.8914   0.1086

   0.6463   0.3537

   0.6080   0.3920

   0.1320   0.8680

   0.8906   0.1094
```

Raw probabilities from C++.

```
Console   Terminal ×   Jobs ×
~/School/UTD S20/Machine Learning/HW4/Machine-Learning-Homework4/
>
> #get predicions from 5 test observations
> for (i in 1:nrow(test)){
+   raw <- calc_raw_prob(test$pclass[i], test$sex[i], test$age[i])
+   print(paste(raw[2], raw[1]))
+ }
[1] "0.365202992580656 0.634797007419344"
[1] "0.89222943053504 0.10777056946496"
[1] "0.648327773317862 0.351672226682138"
[1] "0.891438091992698 0.108561908007302"
[1] "0.646276526535963 0.353723473464037"
[1] "0.607961373922614 0.392038626077386"
[1] "0.132021458260991 0.867978541739009"
[1] "0.890587108178095 0.109412891821905"
```

Raw probabilities from R.

The R program was timed using proc.Time() and the C++ program with a PowerShell script.

```
> endTime <- proc.time()
> #elapsed Time
> endTime-startTime
   user  system elapsed 
   0.05    0.00    0.37 
> 
```

Runtime from Naïve Bayes was 50 milliseconds and…

```
PS C:\Users\emanu\Documents\School\UTD S20\Machi


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 11
Ticks             : 114683
TotalDays         : 1.32734953703704E-07
TotalHours        : 3.18563888888889E-06
TotalMinutes      : 0.000191138333333333
TotalSeconds      : 0.0114683
TotalMilliseconds : 11.4683

Press Enter to continue...:
```

11 milliseconds for C++.

The run time for C++ is a little under 5 times faster. The confusion matrix was also calculated in the C++ code. We got the same accuracy as R.

```
Confusion Matrix:
   69.0000   25.0000
   10.0000   42.0000

Accuracy: 76.0274%
Specifity: 62.6866%
Sensitivity: 87.3418%
```

C++ confusion matrix.

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 69 25
         1 10 42

               Accuracy : 0.7603
                 95% CI : (0.6827, 0.827)
    No Information Rate : 0.5411
    P-Value [Acc > NIR] : 3.612e-08

                  Kappa : 0.5089

 Mcnemar's Test P-Value : 0.01796

            Sensitivity : 0.8734
            Specificity : 0.6269
```

R confusion matrix.

https://www.mathsisfun.com/data/standard-deviation.html was used as a quick reference because I had forgotten how to calculate variance. https://en.cppreference.com/w/ was used to program in C++. http://arma.sourceforge.net/docs.html#example_prog was used to program using armadillo matrices.