

Machine Learning with C++

Bridgette Bryant & Isabelle Kirby

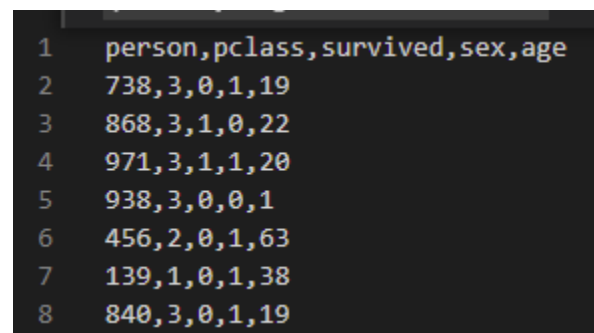
About LogisticRegression.cpp:

Purpose:

The purpose of the program was to build logistic regression from scratch in C++ by creating and evaluating a model. It processes a file named 'titanic_project.csv'. It has 1047 entries by default and five columns, person, pclass, survived, sex, and age. It was derived from the data from the titanic catastrophe and gives details of a person and whether they died or not. Our model's goal is to predict if somebody survived or not based on their sex. It was created on Windows using VS Code and MinGW-w64 g++ compiler. It displays the number of lines of the file, confusion matrix, accuracy, sensitivity, specificity, and duration of our model. You run the program using the terminal './LinearRegression' after building it.

Input:

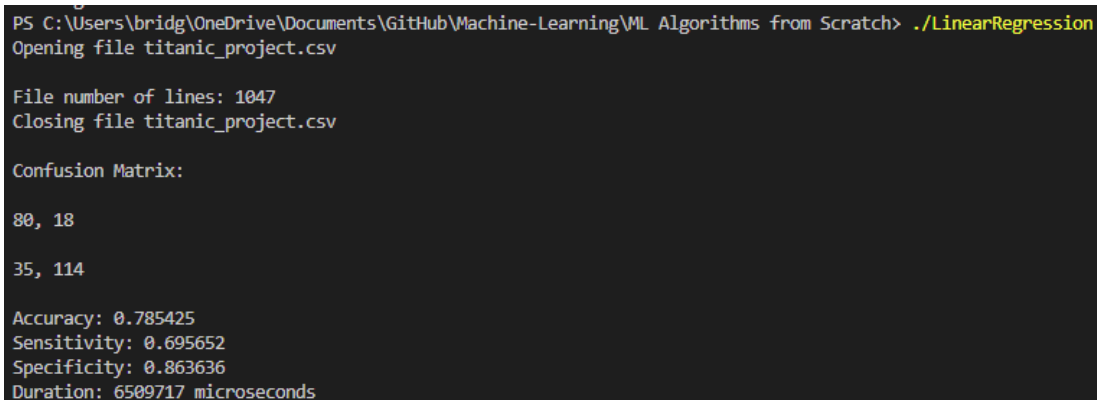
The titanic_project.csv, a data file with 5 columns the first is person which contains an ID to identify them, then has pclass which ranges from 1 to 3 and stands for the class they stayed in, next is survived which is 0 for died and 1 for survived, then it shows sex which is 0 for female and 1 for male, and finally their age which has a large range of different ages.



	person	pclass	survived	sex	age
1	738	3	0	1	19
2	868	3	1	0	22
3	971	3	1	1	20
4	938	3	0	0	1
5	456	2	0	1	63
6	139	1	0	1	38
7	840	3	0	1	19
8					

Output:

When you run the program in the terminal the output will look like this:



```
PS C:\Users\bridg\OneDrive\Documents\GitHub\Machine-Learning\ML Algorithms from Scratch> ./LinearRegression
Opening file titanic_project.csv

File number of lines: 1047
Closing file titanic_project.csv

Confusion Matrix:

80, 18

35, 114

Accuracy: 0.785425
Sensitivity: 0.695652
Specificity: 0.863636
Duration: 6509717 microseconds
```

Functions:

main: This function is the main driver functions. It reads in the input file, if it cannot be found/read it will then exit displaying a message. Otherwise, it will continue and read in the titanic_project.csv, display the number of lines, and setup necessary vectors for each column of the data. It will split the data into train and test. It will skip the first line (because it is the header) and then read all other lines organizing the columns into their given vectors, dividing them into train (lines 1 – 800) and test (lines 801-1047). Then it will close the file and display the appropriate message to the user. After that, it will create the training and testing matrix for the sex column and set their values. Finally, we will begin creating our Logistic Regression model. We start by creating necessary vectors, setting a learning rate, and a necessary array to hold the transpose of our training matrix. Next, we take the dot product of our training matrix and our weights vector, then we take the transpose of our training matrix. Here we start the clock for our model as it will iterate 300 times for the following:

- We create a probability vector from the dot product of our train data and current weights vector, we also calculate the error vector by subtracting the difference between survived and predicted probability
- We will get the dot product of the transposed training sex matrix and error vector
- Then we will use our learning rate to scale our dot product from the transposed training sex matrix and error vector
- Finally, we will adjust the weight vector using the current weight, learning rate, and the dot product of the transposed sex training matrix and error vector.
- We continue this loop 300 times

After, main will then stop the clock for our Logistic Regression model and create a few necessary vectors to hold log odds, the probability vector, and the sex testing predictions. We calculate the log odds by taking the dot product of testing sex matrix and weights vector. Then we create the probability vector by unlogging the log odds and showing them as percentages. Next, we will predict using our probabilities from our model on the test data. Then we calculate and display the confusion matrix, accuracy, sensitivity, and specificity of our model. It will then print the metrics to the user and exit.

getNumLines: This function a string filename and counts the number of lines in the given file, returning it as an integer.

dotProd: This function takes an integer matrix of nx2 and a double type vector it calculates the dot product between them and returns it as a double type vector.

dotProdTwo: This function takes an integer matrix of 2xn and a double type vector it calculates the dot product between them and returns it as a double type vector.

calcAccuracy: This function takes the true-positive, true-negative, false-positive, and false-negative values from a confusion matrix as integers. It returns the calculated accuracy from these values as a double.

calcSensitivity: This function takes the true-positive and false-negative values from a confusion matrix as integers. It returns the calculated sensitivity from these values as a double.

calcSpecificity: This function takes the true-negative and false-negative values from a confusion matrix as integers. It returns the calculated specificity from these values as a double.

doPredicts: This function takes the probability vector created by our logistic regression model and its size. It will return an integer vector of predictions on the test data.

printEverything: This function takes the confusion matrix, accuracy, sensitivity, specificity, and time. It prints all of them to the terminal.

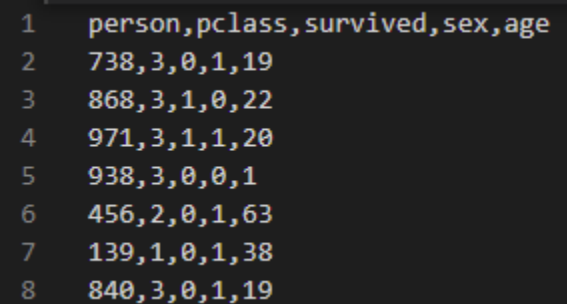
About NaiveBayes.cpp:

Purpose:

The purpose of the program was to build naïve bayes from scratch in C++ by creating and evaluating a model. It processes a file named 'titanic_project.csv'. It has 1047 entries by default and five columns, person, pclass, survived, sex, and age. It was derived from the data from the titanic catastrophe and gives details of a person and whether they died or not. Our model's goal is to predict if somebody survived or not based on their sex. It was created on Windows using VS Code and MinGW-w64 g++ compiler. It displays the number of lines of the file, confusion matrix, accuracy, sensitivity, specificity, and duration of our model. You run the program using the terminal './NaiveBayes' after building it.

Input:

The titanic_project.csv, a data file with 5 columns the first is person which contains an ID to identify them, then has pclass which ranges from 1 to 3 and stands for the class they stayed in, next is survived which is 0 for died and 1 for survived, then it shows sex which is 0 for female and 1 for male, and finally their age which has a large range of different ages.



```
1 person,pclass,survived,sex,age
2 738,3,0,1,19
3 868,3,1,0,22
4 971,3,1,1,20
5 938,3,0,0,1
6 456,2,0,1,63
7 139,1,0,1,38
8 840,3,0,1,19
```

Output:

When you run the program in the terminal the output will look like this:

```
PS C:\Users\bridg\OneDrive\Documents\GitHub\Machine-Learning\ML Algorithms from Scratch> ./NaiveBayes
Opening file titanic_project.csv

File number of lines: 1047
Closing file titanic_project.csv

Probabilities for survival based on sex:

female died    female lived
male died      male lived

0.268966, 0.731034
0.803922, 0.196078

Probabilities for survival based on passenger class:

class1 died    class1 lived
class2 died    class2 lived
class3 died    class3 lived

0.392523, 0.607477
0.572917, 0.427083
0.746193, 0.253807

Predictions
Test:
Survived:      Died:
0.724241       0.275759
0.934594       0.0654065
0.972233       0.027767
0.589097       0.410903
0.381815       0.618185
0.516686       0.483314
0.966961       0.0330392
```

Prints many predictions....

```
Confusion Matrix:

80, 131

35, 1

Accuracy: 0.327935
Sensitivity: 0.695652
Specificity: 0.00757576

Exiting
```

Functions:

main: This function is the main driver functions. It reads in the input file, if it cannot be found/read it will then exit displaying a message. Otherwise, it will continue and read in the titanic_project.csv, display the number of lines, and setup necessary vectors for each column of the data. It will split the data into train and test. It will skip the first line (because it is the header) and then read all other lines organizing the columns into their given vectors, dividing them into train (lines 1 – 800) and test (lines 801-1047). Then it will close the file and display the appropriate message to the user. After that, it will create raw probability matrix for sex and class. It will then calculate the total number of people who died compared to their class and sex based on the training data. Then it calculates the raw probabilities and inputs them into the sex and class matrices. It will then print the matrices to the terminal. Next it will repeat the same steps but for age. However, it divides ages by kids (0-12), teens (13-19), adults (20-39), mid adults (40-59), and seniors (60+). It then calculates the probability of survival for each age category and the variance of the age. It also prints those results to the user. Then it begins prediction on the test data given all the raw predictions from the person's sex, class, and age. It will print a confusion matrix, accuracy, sensitivity, and specificity for the model on the test data. Then it will exit.

ageMean: This function takes the double vector for age and returns the mean of the given ages as a double.

ageVar: This function takes the double vector age and the double average of age. It calculates the variance of the given ages and returns it as a double.

printSex: This function takes the double array for sex and prints the probabilities of the given sex matrix.

printPclass: This function takes the double array for person's class and prints the probabilities of the given person's class matrix.

printAge: This function takes the double array for ages and prints the probabilities of the given ages matrix of the kids, teens, adults, mid-adults, and seniors categories.

getNumLines: This function a string filename and counts the number of lines in the given file, returning it as an integer.

calcSensitivity: This function takes the true-positive and false-negative values from a confusion matrix as integers. It returns the calculated sensitivity from these values as a double.

calcSpecificity: This function takes the true-negative and false-negative values from a confusion matrix as integers. It returns the calculated specificity from these values as a double.

calcProb: This function takes the person's sex, class, and age and uses the raw probabilities matrices for sex, class, and age and predicts the if the person survived or not, it returns a probability vector which holds the probability for died and survived.

Our Logistical Regression vs. Naïve Bayes Results

Confusion Matrix

Our logistic regression model has a good confusion matrix with majority of values in the true-positive and true-negative diagonal in comparison to the false-positive and false-negative diagonal. With 80 in true-positive and 114 in true-negative with only 18 in false-positive and 35 in false-negative.

Our naïve bayes model seems to predict that nearly everyone will survive as shown by the large amount of true and false positives in the matrix. It had 80 true-positives and 131 false positives. This shows the model is very biased towards assuming positive. It also had 35 false-negatives and only 1 true-negative.

However, notice that both models have the same 80 for true-positive and 35 for false-negative, perhaps this points more to the data rather than our models.

Accuracy

Our logistic regression model had a decent accuracy of about 78%, whereas our naïve bayes model had a poor accuracy of about 33%. I think this has to do with the assumption of the naïve bayes model and lack of iteration/correction it has. The logistic regression model iterates to find the gradient decent unlike bayes which utilizes raw probabilities.

Sensitivity & Specificity

Our logistic regression model had a sensitivity of around .69 and a specificity of around .86 which are both pretty close to 1 and relatively good. However, our naïve bayes model had the same sensitivity of about .69 and a specificity of about .01 which is a very poor specificity and shows how much this model predicted false-positives in the confusion matrix.

Generative Classifiers vs Discriminative Classifiers

Generative classifier models will try to model how the data is shaped and focuses on how the data was generated. This causes the model to create shapes/clusters in the data. Whereas discriminative classifiers attempt to draw boundaries within the data to distinguish them and focuses on predicting the labels of the data. For example, if given a data set if you plotted the divisions visually for generative classifiers you would see some drawn shapes around clusters in the data, whereas with discriminative classifiers you would see linear lines as divisions separating (and often cutting through) the different clusters in the data.

Reproducible Research in Machine Learning

Reproducible research is very key to all research fields as if your research/experiment isn't reproducible than it cannot be proven or used by others. The same goes for the reproducible research in machine learning, even if your models and algorithms are perfect, if they cannot be reproduced they are useless.