Sample Portfolio
Saugat Gyawali/Bishal Neupane
09/29/2022

C++ Code and Output:

```
#include<iostream>
#include<fstream>
#include<String>
#include<vector>
#include<sstream>
#include<algorithm>
#include<Eigen/Dense>
#include<chrono>


using namespace Eigen;
using namespace std;
using namespace chrono;

const int MAX_LEN = 1050;
vector<string> sex;
vector<string> survived;
void setValue(string);
double alpha = 0.001;
double e = 2.71828;
int epoch = 10;
MatrixXd weight(2, 1);

int read_data(string);
void setvalue(string);
void display(vector<string>);
MatrixXd set_datamatrix(vector<string>);
MatrixXd set_weight();
MatrixXd multiply_matrix(MatrixXd, MatrixXd);
MatrixXd train_data();
vector<int> test_data();
MatrixXd set_testData();
double checkAccuracy(vector<int>);
double checkSensitivity(vector<int>);
double checkSpecificity(vector<int>);
int numberofIteration = 52;

int main() {

      read_data("titanic_project.csv");
    auto start = steady_clock::now();
    MatrixXd final_weight = train_data();
    auto end = steady_clock::now();
    duration<double> elapsed_time = end - start;
    cout << "*****************************************" << endl;
    cout << "             Coefficients                " << endl;
    cout << "*****************************************" << endl;
    cout << "w0 = " << final_weight(0) << endl;
    cout << "w1 = " << final_weight(1) << endl;
    cout << "*****************************************" << endl;
    cout << "Required equation: " << endl;
```

```
    cout << "Survived = " << final_weight(0) << " + " << final_weight(1) << " age "
<< endl;
    cout << "*****************************************" << endl;

    vector<int> result = test_data();
    cout << "*****************************************" << endl;
    cout << "                  Metrics                " << endl;
    cout << "*****************************************" << endl;
    cout << "Accuracy = " << checkAccuracy(result) << endl;
    cout << "Sensitivity = " << checkSensitivity(result) << endl;
    cout << "Specificity = " << checkSpecificity(result) << endl;

    cout << "*****************************************" << endl;
    cout << "Number of run times = " << numberofIteration << endl;
    cout << "*****************************************" << endl;

    cout << "*****************************************" << endl;
    cout << "elapsed time for training data: " << elapsed_time.count() << " second "
<< endl;
    cout << "*****************************************" << endl;
    //cout << "elapsed time: " << elapsed_time.count() << "s\n";
    return 0;
}

int read_data(string filename) {
    fstream inFS;
    string line;
   // string sex_temp, survived_temp;
    vector<string> lineAll;

    std::cout << "Opening file " << filename << endl;

    inFS.open(filename);

    if (!inFS.is_open()) {
        std::cout << "Could not open a file" << endl;
        return 1;
    }

    //std::cout << "Reading line 1" << endl;
    std::getline(inFS, line);
    //std::cout << "Heading " << line << endl;

    while (inFS.good()) {
        std::getline(inFS, line);
        setValue(line);
    }
}

void setValue(string line) {
    // survived @ 2nd column, and sex @ 3rd column. Started from index 0.
    stringstream s_stream(line);
    vector<string> result;
    while (s_stream.good()) {
        string substr;
        getline(s_stream, substr, ','); //get first string delimited by comma
        result.push_back(substr);
    }
```

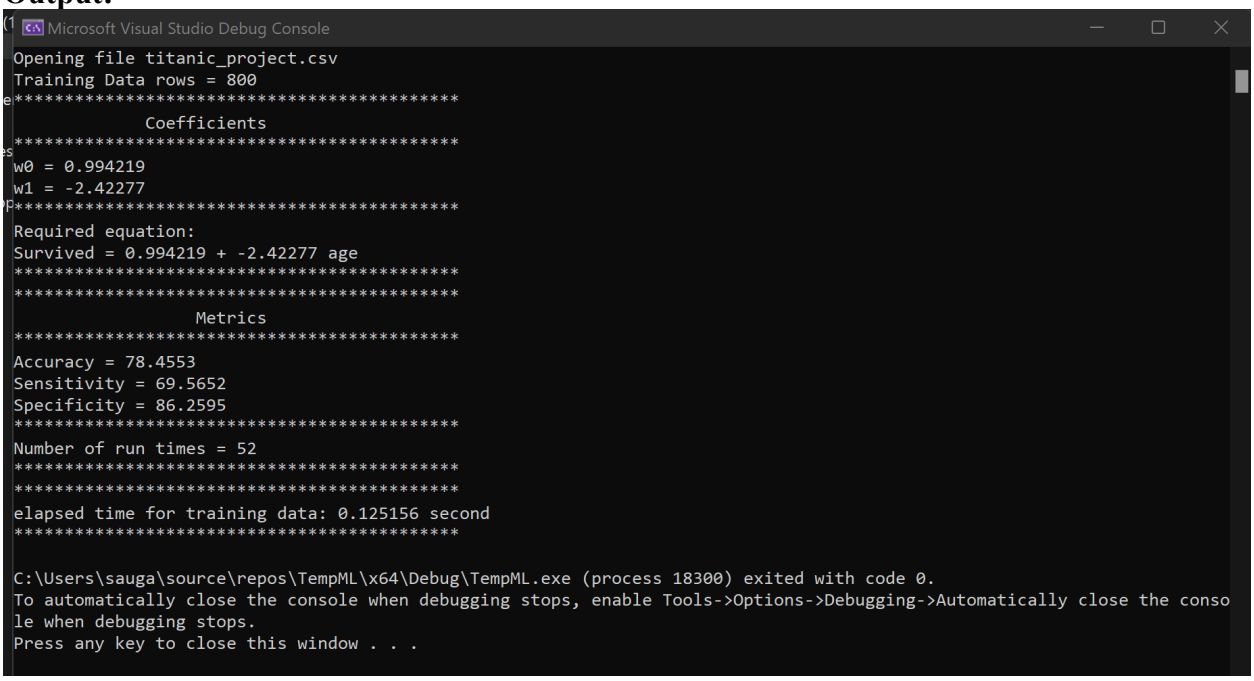
```
    sex.push_back(result.at(3));
    survived.push_back(result.at(2));

}

double signoid(int z) {
    return double(1/(1 + pow(e, -z)));
}

MatrixXd set_weight() {
    weight.setOnes();
    return weight;
}

MatrixXd set_datamatrix(vector<string> sex1) {
    MatrixXd data_matrix(800,2);
    data_matrix.col(0).setOnes();

    for (int i = 0; i < 800; i++) {
        double data = stod(sex1.at(i));
        data_matrix(i, 1) = data;
    }

   // cout << data_matrix << endl;
    return data_matrix;
}


MatrixXd multiply_matrix_for_train(MatrixXd data_matrix, MatrixXd weight) {
    //MatrixXd data_matrix = set_datamatrix(sex);
    //weight = set_weight();
    MatrixXd prod = data_matrix * weight;
    MatrixXd temp = prod.unaryExpr(&signoid);
    return temp;
}

MatrixXd multiply_matrix_for_test(MatrixXd test, MatrixXd wt) {
    MatrixXd prod = test * weight;
    MatrixXd temp = prod.unaryExpr(&signoid);
    return temp;
}

MatrixXd convert_label_train_data(vector<string>survived1) {
    MatrixXd label(800,1);
    for (int i = 0; i < 800; i++) {
        double data = stod(survived1.at(i));
        label(i) = data;
    }
    cout << "Training Data rows = " << label.rows() << endl;
    return label;
}

MatrixXd convert_label_test_data(vector<string>survived1) {
    MatrixXd label(246, 1);
    int testData = 246;
    int count = 0;
    int start = 800;
    for (int i = 0; i < testData; i++) {
```

```
        double data = stod(survived1.at(start));
        label(i) = data;
        count++;
        start++;
    }
    //cout << "Test label count = " << endl;
    //cout << "Test Label rows = " << label.rows() << endl;

    return label;
}

MatrixXd set_testData() {
    int totalSize = sex.size();
    int trainingData = 800;
    int testData = totalSize - trainingData;
    int count = 0;
    MatrixXd testMatrix(246,2);
    //cout << testMatrix.rows() << endl;

    testMatrix.col(0).setOnes();
    for (int i = 0; i < testData; i++) {
        double data = stod(sex.at(trainingData));
        count++;
        testMatrix(i, 1) = data;
        trainingData++;
    }
    return testMatrix;
}

vector<int> test_data() {
    //MatrixXd wt = train_data();
    int truePositive = 0;
    int trueNegative = 0;
    int falsePositive = 0;
    int falseNegative = 0;
    vector<double> storing;
    vector<int> finalRes;
    vector<pair<double, double>> tempRes;
    MatrixXd testMatrix = set_testData();
    MatrixXd label = convert_label_test_data(survived);
    MatrixXd pred1 = multiply_matrix_for_test(testMatrix, weight);

    int start = 800;
    for (int i = 0; i < pred1.rows(); i++) {
        if (pred1.coeff(i) < 0.5) {
            storing.push_back(0);
        }
        else {
            storing.push_back(1);
        }
    }

    for (int i = 0; i < storing.size(); i++) {
        tempRes.push_back(make_pair(storing.at(i), stod(survived.at(start))));
        start++;
    }

    //cout << tempRes.size() << endl;
```

```
    for (int i = 0; i < tempRes.size(); i++) {
        //cout << tempRes.at(i).first << "-------" << tempRes.at(i).second << endl;
        if (tempRes.at(i).first == 1 && tempRes.at(i).second == 1) {
            truePositive++;
        }
        else if (tempRes.at(i).first == 0 && tempRes.at(i).second == 0) {
            trueNegative++;
        }
        else if (tempRes.at(i).first == 1 && tempRes.at(i).second == 0) {
            falsePositive++;
        }
        else if (tempRes.at(i).first == 0 && tempRes.at(i).second == 1) {
            falseNegative++;
        }
    }
    /*
    cout << "TP = " << truePositive << endl;
    cout << "TN = " << trueNegative << endl;
    cout << "FP = " << falsePositive << endl;
    cout << "FN = " << falseNegative << endl;

    */

    finalRes.push_back(truePositive);
    finalRes.push_back(trueNegative);
    finalRes.push_back(falsePositive);
    finalRes.push_back(falseNegative);
    return finalRes;
}

MatrixXd train_data() {
    weight = set_weight();
    MatrixXd label = convert_label_train_data(survived);
    for (int i = 0; i < 52; i++) {
        MatrixXd setTemp = set_datamatrix(sex);
        MatrixXd prob = multiply_matrix_for_train(setTemp,weight);
        MatrixXd error = label - prob;
        weight = weight + alpha * setTemp.transpose() * error;
    }
    return weight;
}

/*
res[0] = truePositive;
res[1] = trueNegative;
res[2] = falsePositive;
res[3] = falseNegative;
*/

double checkAccuracy(vector<int> res) {
    return double(double(res.at(0) + res.at(1)) / double(res.at(0) + res.at(1) +
res.at(2) + res.at(3)) * 100);
}

double checkSensitivity(vector<int> res) {
    return double(double(res.at(0)) / double(res.at(0) + res.at(3)) * 100);
}
```

```
double checkSpecificity(vector<int> res) {
    return double(double(res.at(1)) / double(res.at(1) + res.at(2)) * 100);
}
```

Output:



Analyzing the results of algorithms on the Titanic data:

Using R-studio:
Using confusion matrix, I got:

```
          1  18  80

               Accuracy : 0.7846
                 95% CI : (0.7279, 0.8343)
    No Information Rate : 0.5325
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.5633

 Mcnemar's Test P-Value : 0.02797

            Sensitivity : 0.8626
            Specificity : 0.6957
         Pos Pred Value : 0.7635
         Neg Pred Value : 0.8163
             Prevalence : 0.5325
         Detection Rate : 0.4593
   Detection Prevalence : 0.6016
      Balanced Accuracy : 0.7791

       'Positive' Class : 0
```

Its like what I got from C++. The accuracy of what I did from the C++ code was 0.001 less than what I found using R studio. This is because weights were not exactly equal, that might be the case.

Accuracy of the algorithm was found to be 78.45%. Meaning, the ratio between the sum of true positive and true negative to all the data was found to be 0.7845. True positive are items which are true and were classified as true, False positive are items which are false but were classified as true. True negative are items which are false and classified as false. False negative are those items which are true but were classified as false.

The sensitivity measures the true positive rate. It was found to be 0.8626. It means the ratio of true positive to sum of true positive and false negative was found to be 0.8626.

The specificity measures the true negative rate. For algorithm, it was found to be 0.6957. It means the ratio of true negative to sum of true negative and false positive was found to be 0.6957. It was supposed to be closer to 1, which shows the better result.

For weights, it was w0 = 0.994219 and w1 = -2.4227, using the learning rate 0.001. This means that the required equation would be y = 0.0994219 – 2.4227x. The learning rate here is 0.001 to

optimize the weights of the x and y, so that it would cover all the variables with running rate of 52 iterations.

Comparing and contrasting Discriminative classifiers and Generative Classifiers

Discriminative classifiers are mainly used for supervised learning. Logistic regression uses discriminative classifiers because it directly estimates the parameters of P(Y|X), whereas those who uses generative classifiers, it estimates the parameters for P(Y) and P(X|Y). For example, naïve bayes uses generative classifier. With observation, it can be found that when naïve bayes independence assumptions hold, and the training data gets larger to infinity, they both act as similar classifiers.

For small data sets, naïve bayes is a good one. But as the training data gets increased logistic regression acts as a better classifier. Naïve bayes which follows generative classifiers have higher bias but lower variance than logistic regression which follows discriminative classifier. If there are some outliers in the datasets, then discriminative classifier is a good one. But this classifiers drawback is for the misclassification problem. One of the drawbacks of generative classifiers is that when there is outliers, it affects the model. Regarding, computational cost discriminative models are cheap as compared to generative.

Reference:

Goyal, Chirag. "Deep Understanding of Discriminative and Generative Models in Machine Learning. https://www.analyticsvidhya.com/blog/2021/07/deep-understanding-of-discriminative-and-generative-models-in-machine-learning/. Accessed 09/29/2022

Mazidi, Karen, "Machine Learning Handbook using R and Python." Accessed 09/29.2022

Reproducible research in machine learning

Reproducible research in machine learning is to use other's function and tools by other so that they can repeat again. According to the research paper of Carnegie Mellon University, it states "Methods reproducibility is defined as the ability to implement, as exactly as possible, the experimental and computational procedures, with the same data and tools, to obtain the same

results as in an original work [10]. Methods reproducibility involves providing enough detail about the procedures and data in the study so the same procedures could, in theory or, be exactly repeated. There is several importance of reproducible in machine learning. Some of them are in correctness. According to Section, it states that, if many models are built using the same data and the same processes but giving different results, it challenges the whole process's correctness. Also, it helps in credibility. Thirdly, it helps in easy deployment so that it can be effective for company and save a time to use same and same function.

There are some of the ways how we could attain reproducible machine learning. First it is done by data gathering. Secondly, it is done after creation of feature. According to Section, "To aid in reproducibility, features that are implemented should not change after being created. They need to be immutable. A new feature that is dependent on an already existing one should be represented separately. For example, it should have a new column." Thirdly, there is a model building. It also states, "The potential solution to this is a simple one, keenness to order. Engineers should be keen on the order in which they pass features, hyperparameters, and when to introduce seed values. Non-deterministic algorithms fall under this category as well. Unseeded randomness plays a huge part in making machine learning workflows non-reproducible. An underrated aspect during model building is the use of random seeds." And lastly, there is deployment.

Reference

Ayuya, Collins. "Reproducibility to Improve Machine Learning." Section. https://www.section.io/engineering-education/reproducibility-machine-learning/. Accessed 09/29/2022.

Ding, Zihao et.al. "Reproducibility", https://blog.ml.cmu.edu/2020/08/31/5-reproducibility/. Accessed 09/29/2022.

"The Importance of Reproducibility in Machine Learning Applications", DecisivEdge, https://www.decisivedge.com/blog/the-importance-of-reproducibility-in-machine-learning-applications/. Accessed: 09/29/2022