ML perceptron learning rule (P2)

v0.1

Generated by Doxygen 1.13.2

Chapter 1

ML-Perceptron-learning-rule

1.1 Student

Name: Stan Merlijn

Student nummer: 1863967

1.2 Introduction

In this repository, we will implement and test a perceptron using the learning rule. This will be demonstrated by creating AND and XOR gates and by evaluating the perceptron's ability to classify the `Iris dataset`. The performance will be measured using the `MSE` metric. You can find the assignment `here`.

1.3 Documentation

For this assignment, the documentation was generated with Doxygen. The LaTeX documentation is available `here` and, to view the HTML documentation locally, open `index.html` in a browser.

1.4 Installing

Enter the test directory and then Generate build files:

```
cmake -S . -B build
```

Build the project:

```
cmake --build build
```

Run the executable:

```
./build/MLPerceptronTest
```

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 irisData Struct Reference

A structure to hold the features and targets read from a CSV file. This is fdor the iris data set.

```
#include <csv_reader.hpp>
```

Public Attributes

- std::vector< std::vector< float > > features
- std::vector< int > targets

4.1.1 Detailed Description

A structure to hold the features and targets read from a CSV file. This is fdor the iris data set.

This structure contains two members:

- features: A 2D vector of floats where each inner vector represents a set of features for a single data point.
- targets: A vector of integers where each element represents the target value corresponding to the features.

Definition at line 18 of file csv_reader.hpp.

4.1.2 Member Data Documentation

4.1.2.1 features

```
std::vector<std::vector<float> > irisData::features
```

Definition at line 27 of file csv_reader.hpp.

4.1.2.2 targets

```
std::vector<int> irisData::targets
```

Definition at line 28 of file csv_reader.hpp.

The documentation for this struct was generated from the following file:

- /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/csv_reader.hpp

4.2 Perceptron Class Reference

A simple perceptron model for binary classification.

```
#include <perceptron.hpp>
```

Public Member Functions

- Perceptron (std::vector< double > weights, double bias, double learningRate)

  Constructs a Perceptron with given weights, bias, and learning rate.
- int predict (const std::vector< float > &x) const

  Predicts the output for a given input vector.
- void update (const std::vector< std::vector< float > > &inputs, const std::vector< int > &targets, int epochs)

  Trains the perceptron using the given dataset. Using the learning rule to update the weights.
- double loss (const std::vector< std::vector< float > > &inputs, const std::vector< int > &targets) const

  Calculates the loss of the perceptron. Based of the Mean Squared Error (MSE).
- void __str__ (int verbose) const

  Prints perceptron details.

4.2.1 Detailed Description

A simple perceptron model for binary classification.

Definition at line 20 of file perceptron.hpp.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Perceptron()

```
Perceptron::Perceptron (
            std::vector< double > weights,
            double bias,
            double learningRate)
```

Constructs a Perceptron with given weights, bias, and learning rate.

Parameters

weights	Initial weights.
bias	Initial bias.
learningRate	Learning rate for training.

Definition at line 13 of file perceptron.cpp.

4.2.3 Member Function Documentation

4.2.3.1 __str__()

```
void Perceptron::__str__ (
             int verbose) const
```

Prints perceptron details.

Parameters

verbose	Verbosity level.

Definition at line 61 of file perceptron.cpp.

4.2.3.2 loss()

```
double Perceptron::loss (
             const std::vector< std::vector< float > > & inputs,
             const std::vector< int > & targets) const
```

Calculates the loss of the perceptron. Based of the Mean Squared Error (MSE).

Parameters

inputs	Input vector for all posible inputs.
targets	Target vector for all posible inputs.

Returns

double

Definition at line 50 of file perceptron.cpp.

4.2.3.3 predict()

```
int Perceptron::predict (
             const std::vector< float > & x) const
```

Predicts the output for a given input vector.

Parameters

inputs	Input vector.

Returns

1 if activated, otherwise 0.

Definition at line 16 of file perceptron.cpp.

4.2.3.4 update()

```
void Perceptron::update (
            const std::vector< std::vector< float > > & inputs,
            const std::vector< int > & targets,
            int epochs)
```

Trains the perceptron using the given dataset. Using the learning rule to update the weights.

Parameters

inputs	Input samples.
targets	Target outputs.
epochs	Number of training iterations.

Definition at line 27 of file perceptron.cpp.

The documentation for this class was generated from the following files:

- /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/perceptron.hpp
- /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/perceptron.cpp

Chapter 5

File Documentation

5.1 /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/importing_dataset.py File Reference

Script to import and save the Iris dataset.

Variables

- importing_dataset.iris = load_iris()

5.1.1 Detailed Description

Script to import and save the Iris dataset.

This script uses scikit-learn to load the Iris dataset and writes the data and corresponding target values into CSV files. The features and targets are saved in 'data/iris.csv' and 'data/iris_target.csv' respectively.

Definition in file importing_dataset.py.

5.1.2 Variable Documentation

5.1.2.1 iris

```
importing_dataset.iris = load_iris()
```

Definition at line 14 of file importing_dataset.py.

5.2 /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/importing_dataset.py

Go to the documentation of this file.

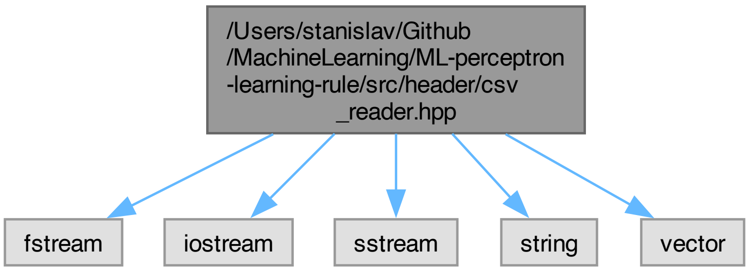
```
from sklearn.datasets import load_iris


if __name__ == '__main__':
    iris = load_iris()
    # Save the dataset (features and target) into a file.
    with open('data/iris.csv', 'w') as f:
        for i in range(len(iris.data)):
            f.write(','.join([str(x) for x in iris.data[i]]) + ',' + str(iris.target[i]) + '\n')

    # Save the target values into a separate file.
    with open('data/iris_target.csv', 'w') as f:
        for i in range(len(iris.target)):
            f.write(str(iris.target[i]) + '\n')
```

5.3 /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/csv_reader.hpp File Reference

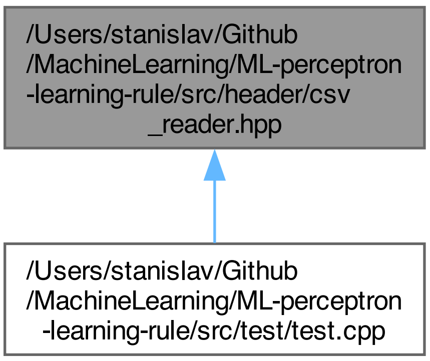
In this class the CSV reader is defined. This is for reading the iris data set.

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
```

Include dependency graph for csv_reader.hpp:


/Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/csv_reader.hpp
fstream
iostream
sstream
string
vector

This graph shows which files directly or indirectly include this file:


/Users/stanislav/Github /MachineLearning/ML-perceptron -learning-rule/src/header/csv _reader.hpp
/Users/stanislav/Github /MachineLearning/ML-perceptron -learning-rule/src/test/test.cpp

Classes

- struct irisData

  A structure to hold the features and targets read from a CSV file. This is fdor the iris data set.

Functions

- std::vector< std::vector< std::string > > read_csv (const std::string &filename, char delimiter=',')

  Reads a CSV file and returns a vector of vectors.
- std::vector< int > get_targets (const std::vector< std::vector< std::string > > &data)

  Extracts the features from the data (column).
- std::vector< std::vector< float > > get_features (const std::vector< std::vector< std::string > > &data)

  Extracts the features from the data.
- irisData filter_data (const std::vector< std::vector< float > > &features, const std::vector< int > &targets, int target)

  Filters out data points with a specific target value.

5.3.1 Detailed Description

In this class the CSV reader is defined. This is for reading the iris data set.

Author

Stan Merlijn

Version

0.1

Date

2025-02-14

Copyright

Copyright (c) 2025

Definition in file csv_reader.hpp.

5.3.2 Function Documentation

5.3.2.1 filter_data()

```
irisData filter_data (
            const std::vector< std::vector< float > > & features,
            const std::vector< int > & targets,
            int target)
```

Filters out data points with a specific target value.

This function takes a set of features and corresponding target values, and filters out the data points where the target value matches the specified target. The remaining data points are returned in a new irisData structure.

Parameters

features	A vector of vectors containing the feature data.
targets	A vector containing the target values corresponding to the feature data.
target	The target value to filter out from the data.

Returns

irisData A structure containing the filtered feature data and target values.

Definition at line 128 of file csv_reader.hpp.

5.3.2.2 get_features()

```
std::vector< std::vector< float > > get_features (
            const std::vector< std::vector< std::string > > & data)
```

Extracts the features from the data.

This function extracts the features from the data and returns a vector of vectors containing the features.

Parameters

data	A vector of vectors representing the rows in the CSV file.

Returns

A vector containing the features.

Definition at line 102 of file csv_reader.hpp.

5.3.2.3 get_targets()

```
std::vector< int > get_targets (
            const std::vector< std::vector< std::string > > & data)
```

Extracts the features from the data (column).

This function extracts the features from the data and returns a vector of vectors containing the features.

Parameters

data	A vector of vectors representing the rows in the CSV file.

Returns

A vector containing the features.

Definition at line 84 of file csv_reader.hpp.

5.3.2.4 read_csv()

```
std::vector< std::vector< std::string > > read_csv (
              const std::string & filename,
              char delimiter = ',')
```

Reads a CSV file and returns a vector of vectors.

This function reads a CSV file and returns a vector of vectors. Each inner vector represents a row in the CSV file. The function assumes that the CSV file is well-formed and does not contain any missing values.

Parameters

filename	The name of the CSV file to read.
delimiter	The delimiter used in the CSV file.

Returns

A vector of vectors representing the rows in the CSV file.

Definition at line 42 of file csv_reader.hpp.

5.4 csv_reader.hpp

Go to the documentation of this file.

```
00001
00011 #pragma once
00012 #include <fstream>
00013 #include <iostream>
00014 #include <sstream>
00015 #include <string>
00016 #include <vector>
00017
00018 struct irisData
00026 {
00027     std::vector<std::vector<float» features;
00028     std::vector<int> targets;
00029 };
00030
00042 std::vector<std::vector<std::string» read_csv(const std::string& filename, char delimiter=',')
00043 {
00044     // Create a vector to store the rows
00045     std::vector<std::vector<std::string» rows;
00046     std::ifstream file(filename);
00047
00048     // Check if the file is open
00049     if (!file.is_open()) {
00050         std::cerr « "Error: Could not open file " « filename « std::endl;
```

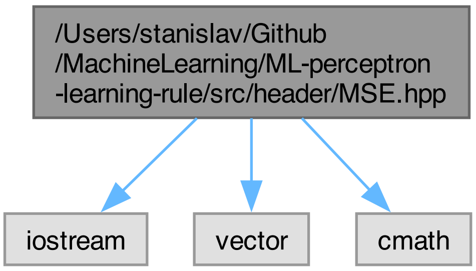
```
        return rows;
    }

    // Read the file line by line
    std::string line;
    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::vector<std::string> cols;
        std::string col;

        while (std::getline(ss, col, delimiter)) {
            cols.push_back(col);
        }

        // Add the columns to the rows
        rows.push_back(cols);
    }

    // Close the file
    file.close();

    return rows;
}

std::vector<int> get_targets(const std::vector<std::vector<std::string»& data)
{
    std::vector<int> targets;
    for (const auto& row : data) {
        targets.push_back(std::stoi(row.back()));
    }
    return targets;
}

std::vector<std::vector<float» get_features(const std::vector<std::vector<std::string»& data)
{
    std::vector<std::vector<float» features;
    for (const auto& row : data) {
        std::vector<float> feature_row;
        // Skip the last column which contains the target
        for (int i = 0; i < row.size() - 1; i++) {
            feature_row.push_back(std::stof(row[i]));
        }
        features.push_back(feature_row);
    }
    return features;
}

irisData filter_data(const std::vector<std::vector<float»& features, const std::vector<int>& targets,
int target)
{
    std::vector<std::vector<float» filtered_features;
    std::vector<int> filtered_targets;
    for (int i = 0; i < features.size(); i++) {
        if (targets[i] != target) {
            filtered_features.push_back(features[i]);
            filtered_targets.push_back(targets[i]);
        }
    }
    return irisData{filtered_features, filtered_targets};
}
```

5.5 /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/MSE.hpp File Reference

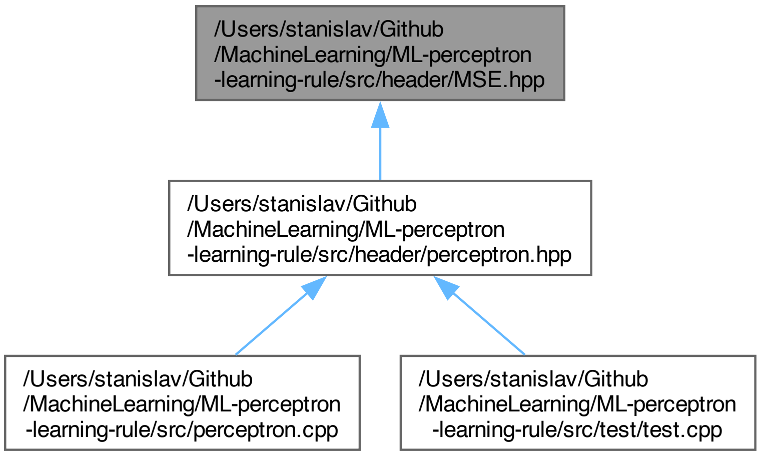
In this file the Mean Squared Error (MSE) function is defined.

```
#include <iostream>
#include <vector>
#include <cmath>
```

Include dependency graph for MSE.hpp:


/Users/stanislav/Github
/MachineLearning/ML-perceptron
-learning-rule/src/header/MSE.hpp
iostream
vector
cmath

This graph shows which files directly or indirectly include this file:


/Users/stanislav/Github
/MachineLearning/ML-perceptron
-learning-rule/src/header/MSE.hpp
/Users/stanislav/Github
/MachineLearning/ML-perceptron
-learning-rule/src/header/perceptron.hpp
/Users/stanislav/Github
/MachineLearning/ML-perceptron
-learning-rule/src/perceptron.cpp
/Users/stanislav/Github
/MachineLearning/ML-perceptron
-learning-rule/src/test/test.cpp

Functions

- double MSE (const std::vector< int > &targets, const std::vector< int > &predictions)

  Calculates the mean squared error between two vectors. MSE = $|\, d - y\, |2 / n$.

5.5.1 Detailed Description

In this file the Mean Squared Error (MSE) function is defined.

Author

Stan Merlijn

Version

0.1

Date

2025-02-14

Copyright

Copyright (c) 2025

Definition in file MSE.hpp.

5.5.2 Function Documentation

5.5.2.1 MSE()

```
double MSE (
              const std::vector< int > & targets,
              const std::vector< int > & predictions)  [inline]
```

Calculates the mean squared error between two vectors. MSE = $| d - y |2 / n$.

This function computes the mean squared error (MSE) between the target values and the predicted values. The MSE is a measure of the average squared difference between the estimated values and the actual value.

Parameters

targets	A vector of target values.
predictions	A vector of predicted values.

Returns

The mean squared error between the targets and predictions. Returns -1 if the sizes of the input vectors do not match.

Definition at line 29 of file MSE.hpp.

5.6 MSE.hpp

Go to the documentation of this file.

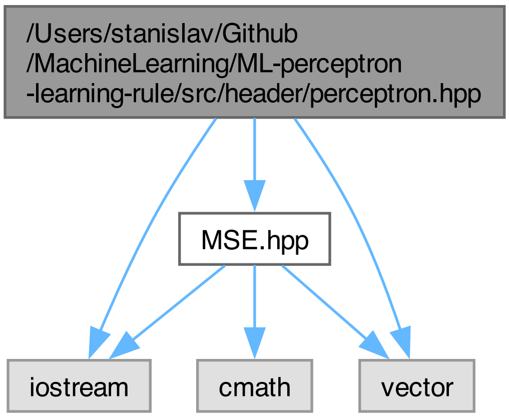
```
00001
00011 #pragma once
00012 #include <iostream>
00013 #include <vector>
00014 #include <cmath>
00015
00029 inline double MSE(const std::vector<int>& targets, const std::vector<int>& predictions)
00030 {
00031     // Ensure both arrays are the same size
00032     if (targets.size() != predictions.size()) return -1;
00033
00034     // Calculate the mean squared error
00035     double sum = 0;
00036     for (int i = 0; i < targets.size(); i++) {
00037         sum += pow(std::abs(targets[i] - predictions[i]), 2);
00038     }
00039     return sum / targets.size();
00040 }
```

5.7 /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/perceptron.hpp File Reference

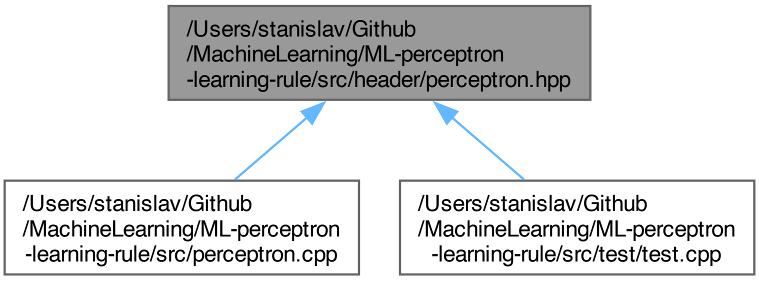
In this file the Perceptron class is declared.

```
#include "MSE.hpp"
#include <iostream>
#include <vector>
```

Include dependency graph for perceptron.hpp:


/Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/perceptron.hpp
MSE.hpp
iostream
cmath
vector

This graph shows which files directly or indirectly include this file:


/Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/header/perceptron.hpp
/Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/perceptron.cpp
/Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/test/test.cpp

Classes

- class Perceptron

  A simple perceptron model for binary classification.

5.7.1 Detailed Description

In this file the Perceptron class is declared.

Author

Stan Merlijn

Version

0.1

Date

2025-02-14

Copyright

Copyright (c) 2025

Definition in file perceptron.hpp.

5.8 perceptron.hpp

Go to the documentation of this file.

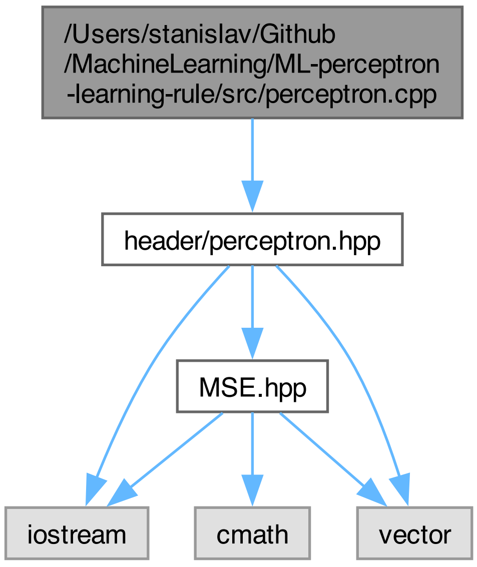
```
#pragma once
#include "MSE.hpp"
#include <iostream>
#include <vector>

class Perceptron
{
public:
    Perceptron(std::vector<double> weights, double bias, double learningRate);

    int predict(const std::vector<float>& x) const;

    void update(const std::vector<std::vector<float»& inputs, const std::vector<int>& targets, int epochs);

    double loss(const std::vector<std::vector<float»& inputs, const std::vector<int>& targets) const;
    void __str__(int verbose) const;

private:
    std::vector<double> weights;
    double bias;
    double learningRate;
};
```

5.9 /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/perceptron.cpp File Reference

In this file the Perceptron class is implemented.

```
#include "header/perceptron.hpp"
```

Include dependency graph for perceptron.cpp:



5.9.1 Detailed Description

In this file the Perceptron class is implemented.

Author

Stan Merlijn

Version

0.1

Date

2025-02-14

Copyright

Copyright (c) 2025

Definition in file perceptron.cpp.

5.10 perceptron.cpp

Go to the documentation of this file.

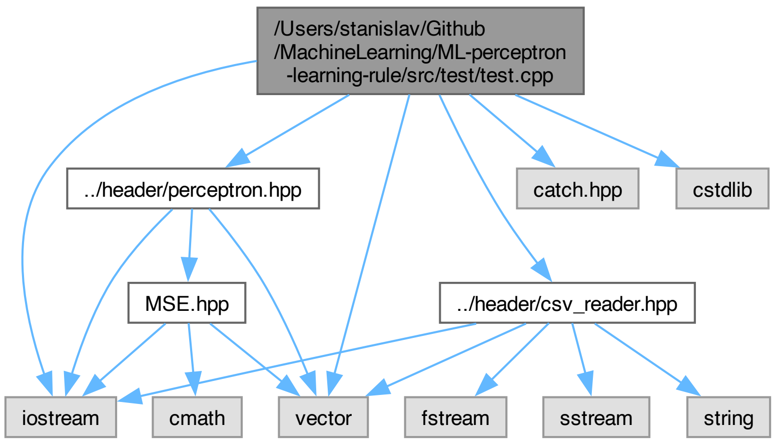
```
#include "header/perceptron.hpp"

Perceptron::Perceptron(std::vector<double> weights, double bias, double learningRate)
    : weights(weights), bias(bias), learningRate(learningRate) {}

int Perceptron::predict(const std::vector<float>& inputs) const
{
    // Dot prodcut for an array of size 2
    double dot_product = bias;
    for (int i = 0; i < weights.size(); i++) {
        dot_product += weights[i] * inputs[i];
    }
    // Threshold function
    return dot_product >= 0 ? 1 : 0;
}

void Perceptron::update(const std::vector<std::vector<float»& inputs, const std::vector<int>& targets,
int epochs)
{
    // ensure both arrays are the same size
    if (inputs.size() != targets.size()) return;

    // Train the perceptron
    for (int epoch = 0; epoch < epochs; epoch++) {
        // Loop through each input
        for (int i = 0; i < inputs.size(); i++) {
            // Get the prediction and error
            double pred = predict(inputs[i]);
            double error = targets[i] - pred;

            // Update each weight based on the input value
            for (int j = 0; j < weights.size(); j++) {
                weights[j] += learningRate * error * inputs[i][j];
            }
            // Update bias
            bias += learningRate * error;
        }
    }
}

double Perceptron::loss(const std::vector<std::vector<float»& inputs, const std::vector<int>& targets)
const
{
    // Get the predictions for the inputs
    std::vector<int> predictions;
    for (int i = 0; i < inputs.size(); i++) {
        predictions.push_back(predict(inputs[i]));
    }
    // Calculate the mean squared error between the targets and predictions
    return MSE(targets, predictions);
}

void Perceptron::__str__(int verbose) const
{
    // Printing the weights
    std::cout « "weights for the perceptron:\n";
    for (int i = 0; i < weights.size(); i++) {
        std::cout « weights[i] « " ";
    }
    // Other info
    if (verbose >= 1) {
        std::cout « "\nbias = " « bias « "\n";
        std::cout « "Learning rate = " « learningRate « std::endl;
    }
}
```

5.11 /Users/stanislav/Github/MachineLearning/ML-perceptron-learning-rule/src/test/test.cpp File Reference

In this file the tests for the Perceptron class are defined.

```
#include "../header/perceptron.hpp"
#include "../header/csv_reader.hpp"
```

```
#include "catch.hpp"
#include <iostream>
#include <vector>
#include <cstdlib>
```
Include dependency graph for test.cpp:



Macros

- #define CATCH_CONFIG_MAIN
- #define WEIGHTS std::vector<double>{0.5, 0.5}
- #define BIAS 0.5
- #define LEARNING_RATE 0.1

Functions

- TEST_CASE ("Perceptron AND gate", "[perceptron]")

  Test case for the AND gate using the Perceptron model.
- TEST_CASE ("Perceptron XOR gate", "[Perceptron XOR]")

  Test case for the XOR gate using the Perceptron model.
- TEST_CASE ("Iris data set - Perceptron Setosa en Versicolor", "[Perceptron Iris]")

  Test case for the Iris data set (Setosa vs Versicolor) using the Perceptron model.
- TEST_CASE ("Iris data set - Perceptron Versicolor en Virginica", "[Perceptron Iris]")

  Test case for the Iris data set (Versicolor vs Virginica) using the Perceptron model.

Variables

- std::vector< std::vector< std::string > > data = read_csv("../../data/iris.csv")
- std::vector< int > targets = get_targets(data)
- std::vector< std::vector< float > > features = get_features(data)

5.11.1 Detailed Description

In this file the tests for the Perceptron class are defined.

Unit tests for the Perceptron, PerceptronLayer and PerceptronNetwork classes.

Author

Stan Merlijn

Version

0.1

Date

2025-02-14

Copyright

Copyright (c) 2025

This file contains a series of test cases to verify the functionality of the Perceptron and PerceptronLayer classes. The tests include training and prediction for various logic gates.

Test Cases:

- Perceptron for AND Gate: Tests the perceptron's ability to learn the AND gate.
- Perceptron for XOR Gate: Tests the perceptron's ability to learn the XOR gate.
- Perceptron for Iris Data Set: Tests the perceptron's ability to learn the Setosa and Versicolor classes.
- Perceptron for Iris Data Set: Tests the perceptron's ability to learn the Versicolor and Virginica classes.

Note

The tests use the Catch2 framework for unit testing.

Definition in file test.cpp.

5.11.2 Macro Definition Documentation

5.11.2.1 BIAS

```
#define BIAS 0.5
```

Default bias for the perceptron.

Definition at line 40 of file test.cpp.

5.11.2.2 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

Definition at line 14 of file test.cpp.

5.11.2.3 LEARNING_RATE

```
#define LEARNING_RATE 0.1
```

Default learning rate for the perceptron.

Definition at line 41 of file test.cpp.

5.11.2.4 WEIGHTS

```
#define WEIGHTS std::vector<double>{0.5, 0.5}
```

Default weights for the perceptron.

Definition at line 39 of file test.cpp.

5.11.3 Function Documentation

5.11.3.1 TEST_CASE() [1/4]

```
TEST_CASE (
            "Iris data set - Perceptron Setosa en Versicolor" ,
            "" [Perceptron Iris])
```

Test case for the Iris data set (Setosa vs Versicolor) using the Perceptron model.

This test trains a perceptron on the subset of the Iris data set that contains only the Setosa and Versicolor classes. The perceptron is expected to correctly separate the two classes with a loss of 0. The final weights and loss are printed.

Definition at line 148 of file test.cpp.

5.11.3.2 TEST_CASE() [2/4]

```
TEST_CASE (
            "Iris data set - Perceptron Versicolor en Virginica" ,
            "" [Perceptron Iris])
```

Test case for the Iris data set (Versicolor vs Virginica) using the Perceptron model.

This test trains a perceptron on the subset of the Iris data set that contains only the Versicolor and Virginica classes. In this scenario, the perceptron cannot correctly separate the two classes, hence a non-zero loss is expected. The test prints the final weights and computed loss after training.

Definition at line 210 of file test.cpp.

5.11.3.3 TEST_CASE() [3/4]

```
TEST_CASE (
             "Perceptron AND gate" ,
             "" [perceptron])
```

Test case for the AND gate using the Perceptron model.

This test case trains a perceptron on the AND gate truth table. The perceptron is expected to correctly learn the AND behavior:

- For inputs {0, 0}, {0, 1}, and {1, 0} the output should be 0.
- For input {1, 1} the output should be 1. The test prints the final weights and computed loss after training.

Definition at line 59 of file test.cpp.

5.11.3.4 TEST_CASE() [4/4]

```
TEST_CASE (
             "Perceptron XOR gate" ,
             "" [Perceptron XOR])
```

Test case for the XOR gate using the Perceptron model.

This test case trains a perceptron on the XOR gate truth table. Since the XOR function is non-linearly separable, a single perceptron cannot correctly learn the XOR behavior. Therefore, the expected behavior is:

- The perceptron incorrectly predicts the output for {0, 0} (i.e. output is not 0).
- The perceptron correctly predicts the output for {0, 1} (i.e. output is 1).
- The perceptron incorrectly predicts the output for {1, 0} (i.e. output is not 1).
- The perceptron correctly predicts the output for {1, 1} (i.e. output is 0).

The test prints the final weights and computed loss after training.

Definition at line 95 of file test.cpp.

5.11.4 Variable Documentation

5.11.4.1 data

```
std::vector<std::vector<std::string> > data = read_csv("../../data/iris.csv")
```

Definition at line 44 of file test.cpp.

5.11.4.2 features

```
std::vector<std::vector<float> > features = get_features(data)
```

Definition at line 48 of file test.cpp.

5.11.4.3 targets

```
std::vector<int> targets = get_targets(data)
```

Definition at line 47 of file test.cpp.

5.12 test.cpp

Go to the documentation of this file.

```
00001
00011 #include "../header/perceptron.hpp"
00012 #include "../header/csv_reader.hpp"
00013
00014 #define CATCH_CONFIG_MAIN
00015 #include "catch.hpp"
00016
00017 #include <iostream>
00018 #include <vector>
00019 #include <cstdlib>   // For rand()
00020
00021
00037
00038 // Define the default inputs
00039 #define WEIGHTS std::vector<double>{0.5, 0.5}
00040 #define BIAS 0.5
00041 #define LEARNING_RATE 0.1
00042
00043 // Read the iris data set
00044 std::vector<std::vector<std::string» data = read_csv("../../data/iris.csv");
00045
00046 // Extract the features and targets
00047 std::vector<int>                targets  = get_targets(data);
00048 std::vector<std::vector<float» features = get_features(data);
00049
00059 TEST_CASE("Perceptron AND gate", "[perceptron]")
00060 {
00061     Perceptron andGate(WEIGHTS, BIAS, LEARNING_RATE);
00062     std::vector<std::vector<float» inputs = {{0, 0}, {0, 1}, {1, 0}, {1, 1}};
00063     std::vector<int> targets = {0, 0, 0, 1};
00064
00065     // Train the perceptron
00066     andGate.update(inputs, targets, 100);
00067
00068     REQUIRE(andGate.predict({0, 0}) == 0);
00069     REQUIRE(andGate.predict({0, 1}) == 0);
00070     REQUIRE(andGate.predict({1, 0}) == 0);
00071     REQUIRE(andGate.predict({1, 1}) == 1);
00072
00073     // Print the weights
00074     std::cout « "Training for the AND gate:\n";
00075     andGate.__str__(1);
00076
00077     // Calculate the loss
00078     double loss = andGate.loss(inputs, targets);
00079     std::cout « "Loss: " « loss « "\n" « std::endl;
00080 }
00081
00095 TEST_CASE("Perceptron XOR gate", "[Perceptron XOR]")
00096 {
00097     Perceptron xorGate(WEIGHTS, BIAS, LEARNING_RATE);
00098     std::vector<std::vector<float» inputs = {{0, 0}, {0, 1}, {1, 0}, {1, 1}};
00099     std::vector<int> targets = {0, 1, 1, 0};
00100
00101     // Train the perceptron
00102     xorGate.update(inputs, targets, 100);
00103
00104     REQUIRE_FALSE(xorGate.predict({0, 0}) == 0); // This should fail for XOR
00105     REQUIRE(xorGate.predict({0, 1}) == 1); // This should pass
00106     REQUIRE_FALSE(xorGate.predict({1, 0}) == 1); // This should fail for XOR
00107     REQUIRE(xorGate.predict({1, 1}) == 0); // This should pass
00108
00109     // Print the weights
00110     std::cout « "Training for the XOR gate:\n";
00111     xorGate.__str__(1);
00112
00113     // Calculate the loss
00114     double loss = xorGate.loss(inputs, targets);
00115     std::cout « "Loss: " « loss « "\n" « std::endl;
```

```
}


// Perceptron XOR gate
// -------------------------------------------------------------------------------

// FAILED:
//   REQUIRE( xorGate.predict({0, 0}) == 0 )
// with expansion:
//   1 == 0

// FAILED:
//   REQUIRE( xorGate.predict({1, 0}) == 1 )
// with expansion:
//   0 == 1

// weights:
// -0.1 2.77556e-17
// bias = 2.77556e-17
// Learning rate = 0.1
// Loss: 0.5

// A XOR gate is a linearly inseparable function, which means that a single perceptron
// cannot learn the weights to create a XOR gate.

TEST_CASE("Iris data set - Perceptron Setosa en Versicolor", "[Perceptron Iris]")
{
    irisData iris01 = filter_data(features, targets, 2);

    // Student ID for seeding
    std::srand(1863967);

    // Generate random weights, bias, and learning rate
    std::vector<double> weights;
    double bias = double(std::rand()) / RAND_MAX * 0.5;
    double learningRate = double(std::rand()) / RAND_MAX * 0.5;

    // Generate random weights
    for (int i = 0; i < iris01.features[0].size(); i++)
    {
        weights.push_back((double)std::rand() / RAND_MAX - 0.5);
    }

    // Create a perceptron object
    Perceptron irisPerceptron(weights, BIAS, LEARNING_RATE);

    // Train the perceptron
    irisPerceptron.update(iris01.features, iris01.targets, 1000);

    // Print the weights and loss after training
    double loss = irisPerceptron.loss(iris01.features, iris01.targets);
    std::cout « "Training the perceptron on the Setosa and Versicolor" « std::endl;
    irisPerceptron.__str__(1);
    std::cout « "Loss: " « loss « "\n" « std::endl;

    std::vector<int> predictions;
    std::vector<int> targets;

    for (int i = 0; i < iris01.features.size(); i++)
    {
        int prediction = irisPerceptron.predict(iris01.features[i]);
        int target = iris01.targets[i];
        predictions.push_back(prediction);
        targets.push_back(target);
    }

    CHECK(predictions == targets);

    // Training the perceptron on the iris data set
    // ------------------------------------------------
    // Training the perceptron on the Setosa and Versicolor
    // weights:
    // -0.244404 -0.199549 0.569217 0.417662
    // bias = 0.4
    // Learning rate = 0.1
    // Loss: 0
    // The loss is 0 because the perceptron is able to separate the two classes
}

TEST_CASE("Iris data set - Perceptron Versicolor en Virginica", "[Perceptron Iris]")
{
    irisData iris12 = filter_data(features, targets, 0);

    // Student ID for seeding
    std::srand(1863967);

    // Generate random weights, bias, and learning rate
    std::vector<double> weights;
    double bias = double(std::rand()) / RAND_MAX * 0.5;
    double learningRate = double(std::rand()) / RAND_MAX * 0.5;

    // Generate random weights
    for (int i = 0; i < iris12.features[0].size(); i++)
    {
        weights.push_back((double)std::rand() / RAND_MAX - 0.5);
    }

    // Create a perceptron object
    Perceptron irisPerceptron(weights, BIAS, LEARNING_RATE);

    // Train the perceptron
    irisPerceptron.update(iris12.features, iris12.targets, 1000);

    // Print the weights and loss after training
    double loss = irisPerceptron.loss(iris12.features, iris12.targets);
    std::cout « "Training the perceptron on the Versicolor and Virginica" « std::endl;
    irisPerceptron.__str__(1);
    std::cout « "Loss: " « loss « "\n" « std::endl;

    std::vector<int> predictions;
    std::vector<int> targets;

    for (int i = 0; i < iris12.features.size(); i++)
    {
        int prediction = irisPerceptron.predict(iris12.features[i]);
        int target = iris12.targets[i];
        predictions.push_back(prediction);
        targets.push_back(target);
    }

    CHECK(predictions == targets);

    // Training the perceptron on the Versicolor and Virginica
    // weights:
    // 32939.9 14870.2 27760 10130.2
    // bias = 5000.5
    // Learning rate = 0.1
    // Loss: 0.5
    // The loss is 0.5 because the perceptron is not able to separate the two classes
}
```