```julia
using Plots
```

```julia
using LinearAlgebra
```

```julia
using Statistics
```

```
PlotlyBackend()
```

```julia
plotly()
```

```
For saving to png with the Plotly backend PlotlyBase has to be installed.
```

Perceptron!

A perceptron in Julia.

Perceptron

Basic Perceptron type, contains the `weights::Vector{Float64}` and the `bias::Float64` .

```julia
"""
Basic Perceptron type, contains the `weights::Vector{Float64}` and the
`bias::Float64`.
"""
struct Perceptron
    weights::Vector{Float64}
    bias::Float64

    function Perceptron(weights::Vector{Float64}, bias::Float64)
        new(weights, bias)
    end
end
```

generate_examples

generate_examples(count::Int64, ratioPositive::Float64, nDims::Int64)

Generates a dataset composed of points of two classes.

```julia
"""
generate_examples(count::Int64, ratioPositive::Float64, nDims::Int64)

Generates a dataset composed of points of two classes.
"""
function generate_examples(count::Int64, ratioPositive::Float64, nDims::Int64)
    positiveCount = ceil(Int64, count * ratioPositive)
    negativeCount = count - positiveCount

    positive = randn(positiveCount, nDims) .+ 14
    negative = randn(negativeCount, nDims) .+ 6

    X = vcat(positive, negative)
    Y = [trues(positiveCount); falses(negativeCount)]

    return X, Y
end
```

transform_examples

transform_examples(X::Matrix{Float64}, Y::BitVector)

Transforms the examples in `X` for easier training of the perceptron.

```julia
"""transform_examples(X::Matrix{Float64}, Y::BitVector)

Transforms the examples in `X` for easier training of the perceptron.
"""
function transform_examples(X::Matrix{Float64}, Y::BitVector)
    newX = X .* [ y ? 1 : -1 for y in Y ]
    return newX
end
```

perceptron_predict

perceptron_predict(weights::Vector{Float64}, bias::Float64, X::Matrix{Float64})::BitVector

Gives predictions for individual data points given a `weights` vector and a `bias`.

```julia
"""perceptron_predict(weights::Vector{Float64}, bias::Float64,
X::Matrix{Float64})::BitVector

Gives predictions for individual data points given a `weights` vector and a
`bias`.
"""
function perceptron_predict(weights::Vector{Float64}, bias::Float64,
X::Matrix{Float64})::BitVector
    scores = X * weights .+ bias
    return [ score > 0 for score in scores ]
end
```

perceptron_predict

perceptron_predict(weights::Vector{Float64}, bias::Float64, X::Matrix{Float64})::BitVector

Gives predictions for individual data points given a `weights` vector and a `bias`.

perceptron_predict(perceptron::Perceptron, X::Matrix{Float64})::BitVector

Returns the predictions given by the `perceptron`.

```julia
"""perceptron_predict(perceptron::Perceptron, X::Matrix{Float64})::BitVector

Returns the predictions given by the `perceptron`.
"""
function perceptron_predict(perceptron::Perceptron,
X::Matrix{Float64})::BitVector
    scores = X * perceptron.weights .+ perceptron.bias
    return [ score > 0 for score in scores ]
end
```

perceptron_train_b

perceptron_train(X::Matrix{Float64}, Y::BitVector)

Train a new perceptron given the dataset `X` and the target classes `Y`. The maximal number of iteration can be set using `maxIters::Int64`.

```julia
"""perceptron_train(X::Matrix{Float64}, Y::BitVector)

Train a new perceptron given the dataset `X` and the target classes `Y`. The maximal number of iteration can be set using `maxIters::Int64`.
"""
function perceptron_train_b(X::Matrix{Float64}, Y::BitVector, maxIters::Int64=Int64(1e6))
    nDims = size(X, 2)
    nExamples = size(X, 1)
    X = hcat(X, ones(nExamples))
    transformedX = transform_examples(X, Y)
    weights = zeros(nDims + 1)
    bias = 0.0
    itersCount = 0

    scores = transformedX * weights
    while itersCount <= maxIters && any([ score <= 0 for score in scores])
        wrongIdx = findfirst(x -> x <= 0, scores)
        weights = weights + transformedX[wrongIdx, :]
        scores = transformedX * weights
        itersCount += 1
    end

    bias = weights[end]
    weights = weights[1:end-1]

    return Perceptron(weights, bias)
end
```

classification_error

classification_error(predicted::BitVector, expected::BitVector)

Returns the 0/1 classification error given the `predicted` and the `expected` labels.

```julia
"""classification_error(predicted::BitVector, expected::BitVector)

Returns the 0/1 classification error given the `predicted` and the `expected`
labels.
"""
function classification_error(predicted::BitVector, expected::BitVector)
    return mean(predicted .!= expected)
end
```

plot_examples

plot_examples(X::Matrix{Float64}, Y::BitVector)

Plots the dataset `X` , `Y` . Meant to be used with 2-dimensional datasets.

```julia
"""plot_examples(X::Matrix{Float64}, Y::BitVector)

Plots the dataset `X`, `Y`. Meant to be used with 2-dimensional datasets.
"""
function plot_examples(X::Matrix{Float64}, Y::BitVector)
    positiveIndices = [ i for i in eachindex(Y) if Y[i] ]
    negativeIndices = [ i for i in eachindex(Y) if !Y[i] ]

    scatter(X[positiveIndices, 1], X[positiveIndices, 2], color = :blue)
    scatter!(X[negativeIndices, 1], X[negativeIndices, 2], color = :red)
end
```

plot_perceptron_and_examples

function plot*perceptron*and_examples(perceptron::Perceptron, X::Matrix{Float64}, Y::BitVector)

Plots the examples and the perceptrons decision boundary.

```
"""function plot_perceptron_and_examples(perceptron::Perceptron,
X::Matrix{Float64}, Y::BitVector)

Plots the examples and the perceptrons decision boundary.
"""
function plot_perceptron_and_examples(perceptron::Perceptron,
X::Matrix{Float64}, Y::BitVector)
    plot_examples(X, Y)
    slope = -(perceptron.bias / perceptron.weights[2]) / (perceptron.bias /
perceptron.weights[1])
    plot!(x -> x * slope - perceptron.bias / perceptron.weights[2], line = 2)
end
```

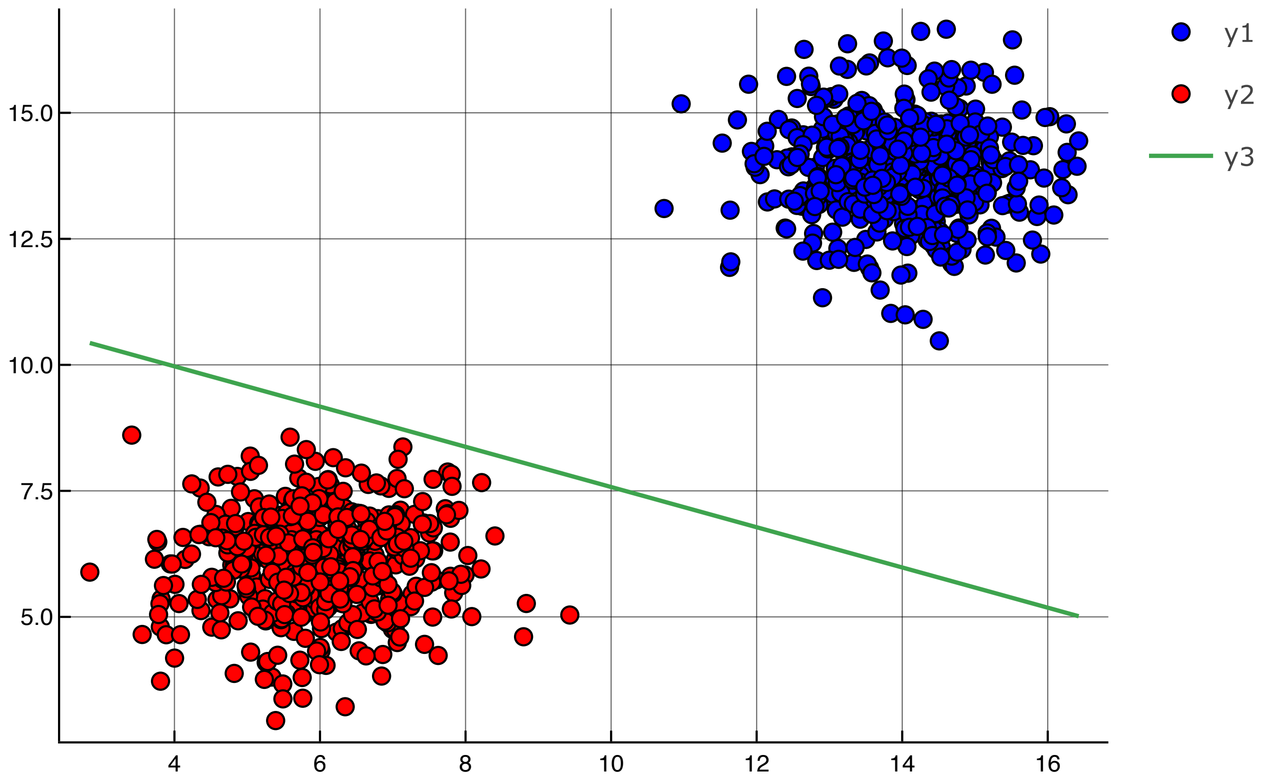
Training a Perceptron!

```
(1000×2 Matrix{Float64}:, BitVector: [true, true, true, true, true, true, true, true
  14.4003   14.8321
  13.6751   14.4088
  12.3921   12.7131
  13.701    13.2155
  13.9056   13.5199
  13.0475   13.3064
  15.5036   14.419
   ⋮
   4.67546   5.76642
   5.49059   3.3733
   5.04708   4.30059
   5.90667   6.28005
   3.80856   3.72415
   6.14805   5.99481
```

```
X, Y = generate_examples(1000, 0.5, 2)
```

```
perceptron =   Perceptron([8.93444, 22.3916], -259.0)
```

```
perceptron = perceptron_train_b(X, Y)
```



```
plot_perceptron_and_examples(perceptron, X, Y)
```

The Perceptron achieves a 1.0 accuracy on the training dataset.

```
begin
    predictions = perceptron_predict(perceptron, X)
    accuracy = 1.0 - classification_error(predictions, Y)
    md"The Perceptron achieves a $accuracy accuracy on the training dataset."
end
```