```fsharp
#r @"..\packages\AForge.Math.2.2.5\lib\AForge.Math.dll"
#r @"..\packages\AForge.Neuro.2.2.5\lib\AForge.Neuro.dll"
#r @"..\packages\FSharp.Data.2.2.5\lib\net40\FSharp.Data.dll"
#I "C:\Users\Matthew\Desktop\LenaDroid\packages"
#r @"..\packages\Accord.MachineLearning.2.15.0\lib\net45\Accord.MachineLearning.dll"
#r @"..\packages\Accord.Math.2.15.0\lib\net45\Accord.Math.dll"
#r @"..\packages\Accord.Neuro.2.15.0\lib\net45\Accord.Neuro.dll"
#r @"..\packages\Accord.Statistics.2.15.0\lib\net45\Accord.Statistics.dll"
#r @"..\packages\AForge.2.2.5\lib\AForge.dll"
#r @"..\packages\AForge.Genetic.2.2.5\lib\AForge.Genetic.dll"
#r @"..\packages\Accord.2.15.0\lib\net45\Accord.dll"
#r @"..\packages\AForge.Math.2.2.5\lib\AForge.Math.dll"
#r @"..\packages\AForge.Neuro.2.2.5\lib\AForge.Neuro.dll"
#r @"..\packages\FSharp.Data.2.2.5\lib\net40\FSharp.Data.dll"
#r @"..\packages\FSharp.Charting.0.90.13\lib\net40\FSharp.Charting.dll"
#r @"XPlot.GoogleCharts.1.2.2\lib\net45\XPlot.GoogleCharts.dll"
#r @"XPlot.GoogleCharts.Deedle.1.2.2\lib\net45\XPlot.GoogleCharts.Deedle.dll"
#r @"Deedle.RPlugin.1.2.4\lib\net40\Deedle.RProvider.Plugin.dll"

open System
open Accord
open Accord.Math
open FSharp.Data
open Accord.Statistics
open Accord.MachineLearning
open Accord.Statistics.Models.Regression
open Accord.Statistics.Models.Regression.Fitting

let trainData = __SOURCE_DIRECTORY__ + "\\WineDataset\\wine.training.data"
let testData = __SOURCE_DIRECTORY__ + "\\WineDataset\\wine.testing.data"

type Wine = CsvProvider<"WineDataset\\wine.training.data">

let wineTrain = Wine.Load(trainData)
let wineTest = Wine.Load(testData)
let classDataOffset = 1

let getInputs (data:Wine) =
    data.Rows
    |> Seq.map
        (fun row -> [|
                        row.Alcohol;
                        row.MalicAcid;
                        row.Ash;
                        row.AlcalinityOfAsh;
                        row.Magnesium |> decimal;
                        row.TotalPhenols;
                        row.Flavanoids;
                        row.NonflavanoidPhenols;
                        row.Proanthocyanins;
                        row.ColorIntensity;
                        row.OD280OD315OfDilutedWines;
                        row.Hue;
                        row.Proline |> decimal;
                    |] |> Seq.map float |> Seq.toArray)
    |> Seq.toArray

let getClasses (data:Wine) =
    data.Rows
    |> Seq.map (fun row -> row.Class - classDataOffset)
    |> Seq.toArray

let classes = getClasses wineTrain

let getColumnsMinsAndMax (data:Wine) =
    let predictors = getInputs data
    [0 .. (data.NumberOfColumns - 2)]
    |> Seq.map (fun i ->
                    predictors.GetColumn(i).Min(), predictors.GetColumn(i).Max())
    |> Seq.toArray

let normalize minmax i (value: float) =
    (value - (fst (Array.get minmax i)))/
    ((snd (Array.get minmax i)) - (fst (Array.get minmax i)))

let getNormalized (data:Wine) =
    (getInputs data) |>
    Array.map(fun row ->
              row
              |> Array.mapi(fun columnNumber value ->
                            normalize (getColumnsMinsAndMax data) columnNumber value))


let normalizedData = getNormalized wineTrain

// Create a new Multinomial Logistic Regression for 3 categories
let mlr = new MultinomialLogisticRegression(13,3)

// Create a estimation algorithm to estimate the regression
let lbnr = new LowerBoundNewtonRaphson(mlr)

// Now, we will iteratively estimate our model. The Run method returns
// the maximum relative

let mutable iter = 0
let rec teach () : unit =
    iter <- iter + 1
    match lbnr.Run(normalizedData, classes) with
    | x when (x > 1e-4 && iter < 1000) -> printfn "%A" x; teach ();
    | _ -> ()

teach()

let getPredictedClassFrom outputLayer offset =
    Array.IndexOf(outputLayer,(Array.max outputLayer)) + offset
```

```
let normalizedTestData = getNormalized wineTest

let testAndCheckAccuracy (mlr: MultinomialLogisticRegression) (data:float[][]) testCl =
    let correctGuesses =
        data
        |> Array.mapi (fun i row ->
                        let outputLayer = mlr.Compute(row)
                        let predictedClass = getPredictedClassFrom outputLayer 0
                        printfn "%A - %A, %A, %A" i (predictedClass = Array.get testCl i) predictedClass (Array.get testCl i)
                        if (predictedClass = Array.get testCl i) then 1 else 0
                       )
        |> Array.sum
    printfn "Correct guesses %A/%A" correctGuesses (Array.length data)
    float(correctGuesses)/float(Array.length data) * 100.0

let accuracy = testAndCheckAccuracy mlr normalizedTestData <| getClasses wineTest
```