Term: M1-RAIA

Semester: 2

AY: 2022-2023

Abdelbacet Mhamdi

Dr.-Ing. in Electrical Engineering

Senior Lecturer at ISET Bizerte

abdelbacet.mhamdi@bizerte.r-iset.tn

Artificial Intelligence - Part 2

LAB MANUAL

INSTITUT SUPÉRIEUR DES ÉTUDES TECHNOLOGIQUES

Higher Institute of Technological Studies of Bizerte

Available @ https://github.com/a-mhamdi/jlai/

Honor code

The University of North Carolina at Chapel Hill
Department of Physics and Astronomy
`http://physics.unc.edu/undergraduate-program/labs/general-info/`

"During this course, you will be working with one or more partners with whom you may discuss any points concerning laboratory work. However, you must write your lab report, in your own words.

Lab reports that contain identical language are not acceptable, so do not copy your lab partner's writing.

If there is a problem with your data, include an explanation in your report. Recognition of a mistake and a well-reasoned explanation is more important than having high-quality data, and will be rewarded accordingly by your instructor. A lab report containing data that is inconsistent with the original data sheet will be considered a violation of the Honor Code.

Falsification of data or plagiarism of a report will result in prosecution of the offender(s) under the University Honor Code.

On your first lab report you must write out the entire honor pledge:

The work presented in this report is my own, and the data was obtained by my lab partner and me during the lab period.

On future reports, you may simply write *"Laboratory Honor Pledge"* and sign your name."

Contents

In order to activate the virtual environment and launch **Jupyter Notebook**, we recommend you to proceed as follow

① Press simultaneously the keys ⊞ & R on the keyboard. This will open the dialog box `Run`;

② Then enter `cmd` in the command line and confirm with ↵ key on the keyboard;

③ Type the instruction `jlai.bat` in the console prompt line;

```
Command Prompt

C:\Users\admin> jlai.bat
```

④ Finally press the ↵ key.

LEAVE THE SYSTEM CONSOLE ACTIVE.

1 Linear Regression

Student's name	. . . .	. . . .	. . . .
Score ***/20***	. . . .	. . . .	. . . .

Detailed Credits

Anticipation ***(4 points)***	. . . .	. . . .	. . . .
Management ***(2 points)***	. . . .	. . . .	. . . .
Testing ***(7 points)***	. . . .	. . . .	. . . .
Data Logging ***(3 points)***	. . . .	. . . .	. . . .
Interpretation ***(4 points)***	. . . .	. . . .	. . . .

In order to conduct the labs effectively, it is highly recommended to check the codes available at `https://github.com/a-mhamdi/jlai/` → *Codes* → *Julia* → *Part-2* → *.*regression.*jl*

Linear regression is a type of **Machine Learning** (ML) algorithm that is used to predict a continuous outcome variable based on one or more predictor variables. It is a type of regression analysis that models the relationship between the dependent variable *y*, aka target, and the independent variable *x*, aka feature, by fitting a straight line to the data. This line can then be used to make predictions about the value *y* based on the values of *x*. Linear regression is a simple and popular method for modeling relationships in data and is often used as a starting point for more complex ML algorithms.

Hereafter is an example of how we might implement linear regression in **Julia**:

JULIA CODE

```julia
# Define the input data
X = [0 2; 1 1; -1 0.5; 1 5] # matrix of input data
y = [2; 3; 4; 5] # vector of output values

```

```julia
          #= NORMAL EQUATION =#

# Compute the coefficients using the normal equation
coefficients = (X' * X) \ X' * y
# Print the coefficients
println("Coefficients are $coefficients")

# Define some test input
x_test = [1 6]
# Compute the predicted output
y_pred = x_test * coefficients
# Print the predicted output
println("Predicted output is $y_pred")

          #= LOADING `LinearRegressor` FROM `MLJLinearModels` =#

# Import the required library
using MLJ

LR = @load LinearRegressor pkg=MLJLinearModels
lr_ = LR(fit_intercept=false)
# Bind an instance of `lr_` to training data
lr = machine(lr_, table(X), y) |> fit!
# Display the fitted parameters
println("Fitted parameters are $(fitted_params(lr))")

# Recall the same previously defined test input
x_test = [1 6]
# Compute the predicted output
y_hat = predict(lr, x_test)
# Print the predicted output
println("Predicted output is $y_hat")
```

▼ Remark 1

If you want to avoid the warning that pops up at the REPL, it is more convenient to coerce to continuous the scitypes of data being fed to model in machine when using ***MLJ*** *package. You can further check the documentation by typing:*

```
julia> @doc MLJLinearModels.LinearRegression
```

Exercise № 1:

x_1 and x_2 both are vectors of 10000 random values. x_1 is an array of float values sampled from a normal distribution with mean equal to $\mu = -3$ and a standard deviation fixed to $\sigma = 2.7$. The step change could be set at 0.01. The elements of x_2 are however integer values depicted from a uniform distribution ranging from -8 to 4.

a) Write **Julia** code to generate and plot histograms of x_1 and x_2

b) Standardize x_1 using the package **Distributions** at first, and then **MLJ**.

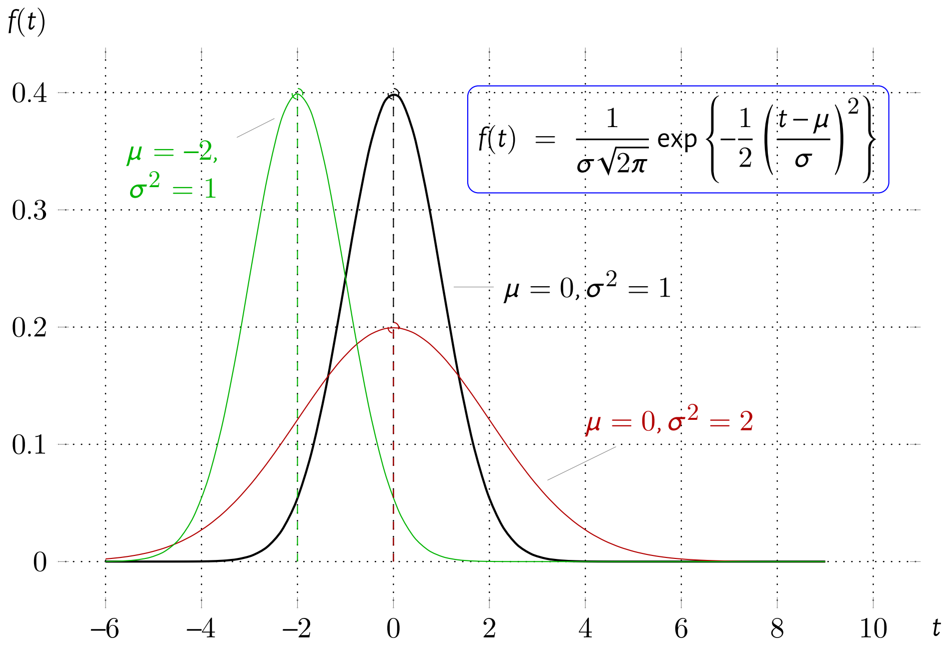
c) Normalize x_2

d) Say that we have a target $y = -x_1 + 3.5x_2$

- Generate the vector y;
- By applying normal equation, do you get $\hat{\theta} = \begin{bmatrix} -1 & 3.5 \end{bmatrix}^T$
- Using **MLJ** package, load a linear regression model, bind an instance of it to the data in $X = \text{hcat}(x_1, x_2)$. Compute $\hat{\theta}$ again.

▼ Remark 2

The graphs of some continuous univariate normal distributions are shown below:



What needs to be retained

Model refers to the mathematical formula or equation that is used to make predictions based on the data.

Coefficient represent the strength and direction of the relationship between a particular predictor variable and the predicted variable.

Residual is the difference between the actual and predicted outputs. It is used to measure the goodness of fit of the model.

2 Logistic Regression

Student's name	. . . .	. . . .	. . . .
Score **/20**	. . . .	. . . .	. . . .

Detailed Credits

Anticipation ***(4 points)***	. . . .	. . . .	. . . .
Management ***(2 points)***	. . . .	. . . .	. . . .
Testing ***(7 points)***	. . . .	. . . .	. . . .
Data Logging ***(3 points)***	. . . .	. . . .	. . . .
Interpretation ***(4 points)***	. . . .	. . . .	. . . .

The notebook is available at `https://github.com/a-mhamdi/jlai/` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-2* ⟶ *logistic-regression.jl*

Logistic regression is a type of statistical model that is used to predict the likelihood of an event occurring. It is a type of regression analysis that is used when the dependent variable is binary, meaning it can only take on one of two values, such as 0 or 1. Logistic regression is used to model the relationship between a dependent variable and one or more independent variables by fitting a logistic curve to the data. This curve can then be used to make predictions about the likelihood of an event occurring.

Logistic regression is a popular machine learning algorithm that is used for classification tasks. In Julia, you can use the *GLM* package to fit a logistic regression model. Here is an example of how you might do this:

Julia Style Pseudocode

```julia
using GLM

```

```
# Load the data
X = # Matrix of predictors
y = # Vector of target labels

# Fit the logistic regression model
model = glm(@formula(y ~ X), Binomial(), IdentityLink())

# Make predictions using the model
predictions = predict(model, X)
```

In this example, *X* is a matrix of predictors (also known as features) and *y* is a vector of target labels (also known as class labels). The `glm` function is used to fit the logistic regression model, and the predict function is used to make predictions using the trained model.

WHAT NEEDS TO BE RETAINED

Model This refers to the mathematical formula or equation that is used to make predictions based on the data.

Class In classification, a class refers to a group or category to which a data point belongs. For example, in a classification task to predict whether an email is spam or not, the classes would be ”spam” and ”not spam.”

Probability In classification, the probability of a data point belonging to a particular class is often used to make predictions. For example, if a model predicts that a given email has a 90% probability of being spam, it is likely to be classified as spam.

3 | *k*-Nearest Neighbors

Student's name	. . . .	. . . .	. . . .
Score ***/20***	. . . .	. . . .	. . . .

Detailed Credits

Anticipation ***(4 points)***	. . . .	. . . .	. . . .
Management ***(2 points)***	. . . .	. . . .	. . . .
Testing ***(7 points)***	. . . .	. . . .	. . . .
Data Logging ***(3 points)***	. . . .	. . . .	. . . .
Interpretation ***(4 points)***	. . . .	. . . .	. . . .

The notebook is available at `https://github.com/a-mhamdi/jlai/` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-2* ⟶ *knn.jl*

k-nearest neighbors (*k*-NN) is a supervised learning algorithm used for classification and regression. In the classification case, the output is a class membership (*e.g.* "cat" or "dog"). In the regression case, the output is a continuous value (e.g. temperature).

To make a prediction for a new data point, the algorithm finds the closest data points in the training set (*i.e.* the "nearest neighbors") and takes the average (for regression) or the majority vote (for classification) of their outputs as the prediction for the new data point. The number of nearest neighbors (k) is a hyperparameter that must be specified in advance.

k-NN is a simple and effective algorithm, but it can be computationally expensive and is not suitable for large datasets. It is also sensitive to the scale and distribution of the data.

Here are two examples of *k*-NN implemented in *Julia*:

JULIA STYLE PSEUDOCODE

```julia
using Distances
using StatsBase

function knn(X::Array{T, 2}, y::Array{U, 1}, x::Array{T, 1}, k::Int)␣
 ↪where {T <: Real, U}
    # Calculate distances between x and each point in X
    dists = pairwise(Euclidean(), X, x)

    # Sort the distances and indices in ascending order
    sorted_dists = sortperm(dists)

    # Take the top k distances and their corresponding y values
    y_neighbors = y[sorted_dists[1:k]]

    # Return the majority vote of the neighbors
    return mode(y_neighbors)
end
```

```julia

using CSV, DataFrames

## Load Data
df = CSV.read("./datasets/Social_Network_Ads.csv", DataFrame)
x = Float64.(df[!, 2]);
y = df[!, end];

println(typeof(x), size(x))
l = size(x)[1]

# Scatter Plot Of Data
using Plots; # unicodeplots()
g1 = scatter(x, y; c=y, legend=false);

using NearestNeighbors
# KDTree(data, metric; leafsize, reorder)
tree = KDTree(x')
# Initialize k for k-NN
k = 3

tst = rand(1:l, Int(.2*l))
# Find Nearest Neighbors Using k-NN & k-d Tree
idxs, dists = knn(tree, x[tst], k, true)
```

This implementation uses the *Distances* and *StatsBase* packages to calculate distances and perform a majority vote. It takes as input the training data X and labels y, the test point x, and the number of nearest neighbors k, and returns the predicted label for x.

4 Support Vector Machine

Student's name	. . . .	. . . .	. . . .
Score **/20**	. . . .	. . . .	. . . .

Detailed Credits

Anticipation ***(4 points)***	. . . .	. . . .	. . . .
Management ***(2 points)***	. . . .	. . . .	. . . .
Testing ***(7 points)***	. . . .	. . . .	. . . .
Data Logging ***(3 points)***	. . . .	. . . .	. . . .
Interpretation ***(4 points)***	. . . .	. . . .	. . . .

The notebook is available at `https://github.com/a-mhamdi/jlai/` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-2* ⟶ *svc.jl*

Support vector machines (**SVM**s) are a type of supervised learning algorithm that can be used for classification or regression tasks. SVMs are a powerful and flexible tool for solving a wide range of machine learning problems, and have been widely used in many different fields, including text classification, image classification, and bioinformatics.

Here is an example of how you might implement an **SVM** in *Julia* for classification tasks:

JULIA STYLE PSEUDOCODE

```julia
using LIBSVM

# define the model
model = LIBSVM.SVM(SVC(), LinearKernel())

```

```julia
# train the model on the training data
LIBSVM.fit!(model, train_X, train_y)

# use the trained model to make predictions on the test data
predictions = LIBSVM.predict(model, test_X)

# evaluate the model's performance
accuracy = mean(test_y .== predictions)
```

Note that this is just one way to implement an **SVM** in Julia, and there are many other packages and approaches you can use. This example uses the *LIBSVM* package, which provides a convenient interface for working with **SVM**s in *Julia*.

5 *K*-Means for Clustering

Student's name	....	....	....
Score **/20**	....	....	....

Detailed Credits

Anticipation ***(4 points)***	....	....	....
Management ***(2 points)***	....	....	....
Testing ***(7 points)***	....	....	....
Data Logging ***(3 points)***	....	....	....
Interpretation ***(4 points)***	....	....	....

The notebook is available at `https://github.com/a-mhamdi/jlai/` ⟶ *Codes* ⟶ *Julia* ⟶ *Part-2* ⟶ *kmeans.jl*

K-Means clustering is a method of unsupervised learning in machine learning. It is used to divide a dataset into a specified number (k) of clusters, with each cluster containing data points that are similar to each other. The goal of the algorithm is to minimize the within-cluster sum of squares, which measures the similarity of the data points within each cluster.

To perform *K*-Means clustering, the algorithm first randomly selects k data points from the dataset and assigns them to be the centroids of the k clusters. It then computes the distance between each data point and each centroid, and assigns each data point to the cluster whose centroid is closest to it. The algorithm then updates the centroids of each cluster by taking the mean of all of the data points in the cluster. This process is repeated until the centroids of the clusters no longer change, or until a maximum number of iterations is reached.

K-Means clustering is often used for exploratory data analysis, to identify underlying patterns and structures in a dataset. It is also commonly used in data compression, where it is used to group similar data points together

K-Means is a clustering algorithm that is used to partition a dataset into a specified number of clusters. Here are two examples of K-means implemented in Julia:

JULIA STYLE PSEUDOCODE

```julia
using Clustering

function kmeans(X::Array{T, 2}, k::Int) where {T <: Real}
   # Initialize cluster centers randomly
   centers = zeros(k, size(X, 2))
   for i in 1:k
      centers[i, :] = X[rand(1:size(X, 1)), :]
   end

      # Repeat until convergence
      converged = false
      while !converged
      # Calculate distances between each point and each cluster center
      dists = pairwise(Euclidean(), X, centers)

      # Assign each point to the closest cluster center
      clusters = argmin(dists, dims=1)

      # Calculate the new cluster centers as the mean of all points in␣
 ↪the cluster
      new_centers = zeros(k, size(X, 2))
      for i in 1:k
         if sum(clusters .== i) > 0
            new_centers[i, :] = mean(X[clusters .== i, :], dims=1)
         else
            # If a cluster is empty, randomly initialize a new center
            new_centers[i, :] = X[rand(1:size(X, 1)), :]
         end
      end

      # Check for convergence
      converged = isapprox(centers, new_centers, rtol=1e-6)

      # Update the cluster centers
      centers = new_centers
   end

```

```julia
    return centers, clusters
end
```

```julia
## Import Librairies
using CSV, DataFrames

## Load The Dataset From CSV File
df = CSV.read("./datasets/Mall_Customers.csv", DataFrame);

## Take A Look @ Data
first(df, 5)
income = df[!, 4];
ss = df[!, 5];

## Plots PKG
using Plots
scatter(income, ss, legend=false)

## Clustering PKG
using Clustering

## Features Construction
X = hcat(ss, income);
typeof(X)
hat_clusters = kmeans(X', 5; display=:iter)

## Scatter Plot
scatter(ss, income, marker_z=hat_clusters.assignments,
    color=:winter,
    legend=false)

scatter!(hat_clusters.centers[1,:]', hat_clusters.centers[2,:]',
    color=:black,
    labels=["#1" "#2" "#3" "#4" "#5"],
    legend=true)
```

This implementation uses the *Clustering* package to calculate distances. It takes as input the dataset *X* and the number of clusters *k*, and returns the cluster centers and the cluster assignments of each point.

What needs to be retained

Cluster It refers to a group of data points that are similar to one another.

Centroid In clustering algorithms such as *K*-Means, the centroid of a cluster is the mean of all the data points in that cluster.

Distance It measures are used to determine how similar or dissimilar two data points are. The distance between two points is often used to determine which points belong in the same cluster.

6 Project Assessment

The final project will offer you the possibility to cover in depth a topic discussed in class which interests you, and you like to know more about it. The overall goal is to provide you with a challenging but achievable assessment that allows you to demonstrate your knowledge and skills in fuzzy logic or neural networks.

Here are some potential machine learning projects that you could consider:

Predicting stock prices: You could try building a model to predict future stock prices using historical data and financial news articles.

Sentiment analysis: You could build a model to classify text data (such as movie reviews or social media posts) as positive, negative, or neutral.

Fraud detection: You could build a model to identify fraudulent transactions in a dataset of credit card or bank transactions.

Image classification: You could build a model to classify images into different categories (such as animals, objects, or scenes).

Spam filtering: You could build a model to classify emails as spam or not spam.

Customer segmentation: You could build a model to cluster customers into different groups based on their characteristics and behavior.

Speech recognition: You could build a model to transcribe spoken words into text.

Recommendation systems: You could build a model to recommend products, movies, or other items to users based on their past behavior and preferences.

You have to provide all necessary resources, such as sample code, relevant datasets, as well as creating a set of slides to present your work. You are expected to demonstrate your understanding of the material covered throughout this course, as well as familiarizing yourselves with relevant programming languages and libraries. The final project is comprised of:

1. proposal;
2. report documenting your work, results and conclusions;
3. presentation;

4. source code *(You should share your project on **GitHub**.)*

Project proposal

It is about two pages long. It includes:

- Title
- Datasets *(If needed!)*
- Idea
- Software *(Not limited to what you have seen in class)*
- Related papers *(Include at least one relevant paper)*
- Teammate *(Teams of three to four students. You should highlight each partner's contribution)*

Project report

It is about ten pages long. It revolves around the following key takeaways:

- Context *(Input(s) and output(s))*
- Motivation *(Why?)*
- Previous work *(Literature review)*
- Flowchart of code, results and analysis
- Contribution parts *(Who did what?)*

Typesetting using LaTeX is a bonus. You can use **LyX** (https://www.lyx.org/) editor. A template is available at https://github.com/a-mhamdi/jlai/tree/main/Codes/Report. Here what your report might contain:

1. Provide a summary which gives a brief overview of the main points and conclusions of the report.
2. Use headings and subheadings to organize the main points and the relationships between the different sections.
3. Provide an outline or a list of topics that the report will cover. Including a table of contents can help to quickly and easily find specific sections of your report.
4. Use visuals: Including visual elements such as graphs, charts, and tables can help to communicate the content of a report more effectively. Visuals can help to convey complex information in a more accessible and intuitive way.

If you used `Julia`, you can generate the documentation using the package **Documenter.jl**. It is a great way to create professional-looking material. It allows to easily write and organize documentation using a variety of markup languages, including **Markdown** and LaTeX, and provides a number of features to help create a polished and user-friendly documentation website.

I will assess your work based on the quality of your code and slides, as well as your ability to effectively explain and demonstrate your understanding of the topic. I will also consider the creativity and originality of your projects, and your ability to apply what you have learned to real-world situations. I also make myself available to answer any questions or provide feedback as you work on your projects.

The overall scope of this manual is to introduce **Artificial Intelligence (AI)** , through either some numerical simulations or hands-on training, to the students enrolled in the master's program **RAIA**.

The topics discussed in this manuscript are as follow:

① Data Preprocessing

cleaning; transformation; normalization

② Regression

model; coefficient; residual

③ Classification

model; class; probability

④ Clustering

cluster; centroid; distance

Julia; REPL; *Pluto*; *Fuzzy*; *MLJ*; *DataFrames*; artificial intelligence; regression; classification; clustering.