Matlab Machine Learning

Ioannis Ioannidis
University of Piraeus
NCSR Demokritos

1. Introduction

Machine learning algorithms are mathematical model mapping methods used to learn or uncover underlying patterns embedded in the data. As in most of mathematics, someone could approach a problem in various ways, ending up at a satisfactory solution. However, the quality of the results, as well as the time needed to go through the problem, differs from one approach to another.

Considering this, in the present project we examine the outcomes for different implementations about the same Machine Learning algorithm. We go over both Supervised and Unsupervised Learning, using Linear - Logistic Regressions and K-means methods respectively.

The report's structure is as following: In section 2, we describe the *Theoretical Background* of the algorithms that will be tested, getting an intuitive taste of how they work. The basic implementation we went through, as well as the results exported, are presented in the *Experiments* chapter. Finally, all the remarks highlighted and the observations we ended up, are imprimted in the *Conclusions* part.

2. Theoretical Background

2.1. Linear Regression

Linear regression is useful for finding relationship between variables, looking for statistical relationships. One is the response or dependent variable and the others are the predictors or independent variable.

When we deal with just one predictor, we call the procedure *Simple Linear Regression*. Therefore, when there exist two or more independent variables, the corresponding name is *Multivariate Linear Regression*.

However, both procedures use the same formula, in order to predict the response variable. The core idea is to obtain a line or hyperplane (depending on the number of input variables) that best fits the data. By the term best fit, we mean the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.

This formula is mathematically expressed as

$$h_\theta(x) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + ... + \theta_n X_n$$

, where $h_\theta(x)$ is the predicted output variable and X terms are the corresponding input variables.
Finding the values of these constants θ is what regression model does, as described above, by minimizing the error and fitting the best line or hyperplane.
This is achieved by minimizing the cost function

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x_i) - y_i)^2$$

2.2. Logistic Regression

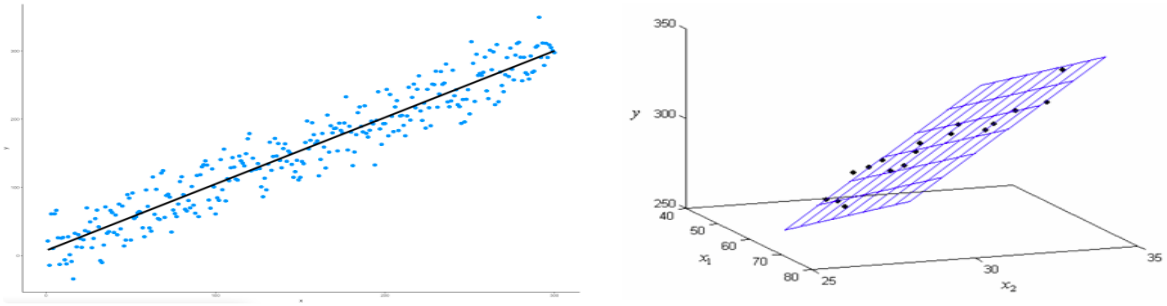
Logistic Regression is a Machine Learning algorithm, which is used for the classification problems. It is a predictive analysis method and based on the concept of probability.

We can call a Logistic Regression model as a Linear Regression one, but here we use a more complex cost function. This cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function. Specifically, in Logistic Regression we have the formula

$$h_\theta(x) = g(\theta^T X) = \frac{1}{1 + e^{-\theta^T x}} = P(y = 1|x; \theta$$

and we attempt to minimize the cost function as we described in previously

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x_i) - y_i)^2$$



(a) Simple Linear Regression

(b) Multivariate Linear Regression

Fig. 1: Linear Regression

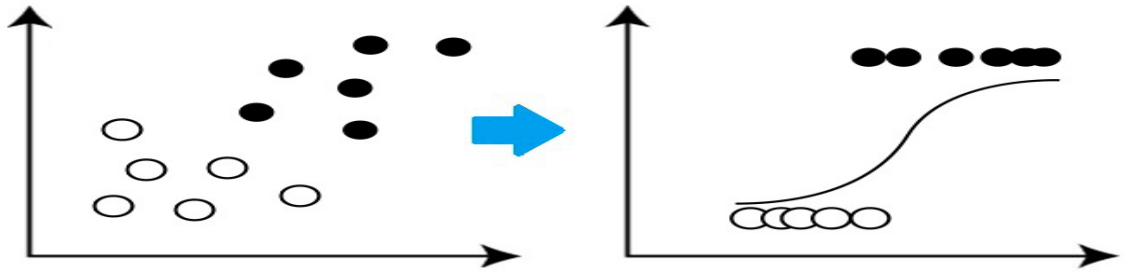


Fig. 2: Sigmoid Function In Logistic Regression

2.3. K-means Clustering

The theory behind the algorithm is actually pretty straight forward. To begin, we choose a value for k (the number of clusters) and randomly pick an initial centroid (centre coordinates) for each cluster. We then apply a two step process:

- Assignment step — Assign each observation to it's nearest centre.
- Update step — Update the centroids as being the centre of their respective observation.

We repeat these two steps over and over until there is no further change in the clusters. At this point the algorithm has converged and we may retrieve our final clusterings.

In mathematical terms, for a cluster of observations $x_i \in \{x_1, x_2, ..., x_n\}$ we wish to find the centroid C that minimizes

$$J(x) = \sum_{i=1}^{n} ||x_i - C||^2$$

where, since we have defined the centroid as simply being the centre of its respective observations, we may calculate

$$C = \frac{\sum_{i=1}^{n} x_i}{n}$$

This equation gives us the sum of squared error for a single centroid C. However, what we really wish to minimize the sum of squared error across all centroids $C_j \in \{C_1, C_2, ..., C_k\}$ for all observations $x_i \in$

$\{x_1, x_2, ..., x_n\}$. Thus the goal of K-means is to minimize the total sum of squared error (SST) objective function

$$J(x) = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^{(j)} - C_j||^2$$

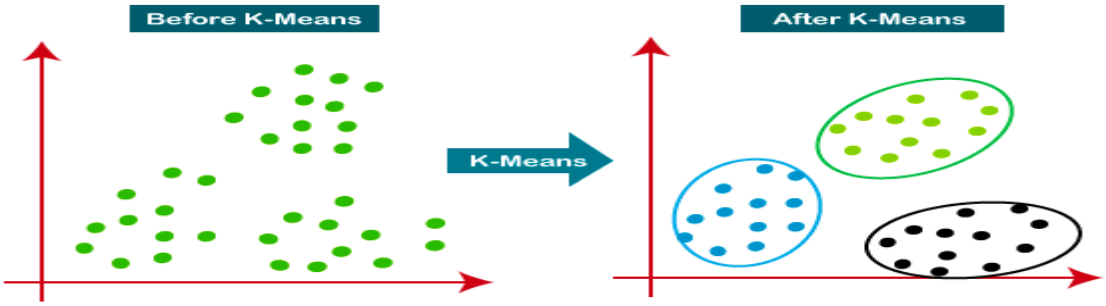
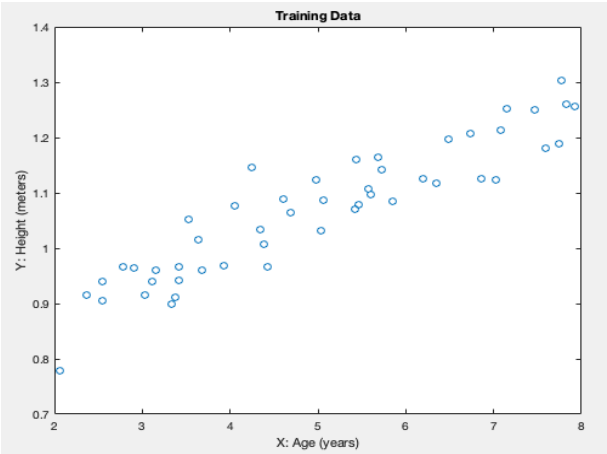


Fig. 3: K-means Clustering

3. Experiments

3.1. Simple Linear Regression

As we mentioned before, in Simple Linear Regression we deal with one predictor and one response variable. In simple words we need a dataset, which contains one variable X and one Y. In that way, we will use the following observations for the training procedure.



We will apply three methods, in order to find the line that fits best in these training data. We begin with the built-in Matlab function *fitlm*.

```
% 1. Using 'fitlm' function: "fitlm(x,y)" uses intercept by default 2
a1 = fitlm(x,y);
W=a1.Coefficients{:,1};
```

Another built-in is the *regress* function. The difference with the fitlm is that here there is no standard intercept and we have to insert it on our own.

```
% 2. Using 'regress' function: "regress(y,x)" uses no intercept by default.

% Add intercept term to x
X = [ones(m, 1), x];

a2 = regress(y,X);
```

Finally, we will use the Stohastic Gradient Decent (SGD) method, which iteratively updates the coefficients θ_i using the rule

$$\theta_i = \theta_i - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

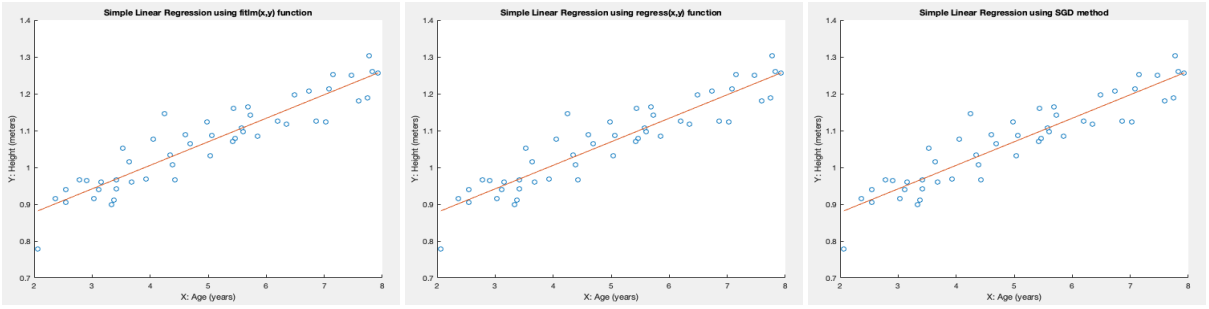
Here, α is pointing the learning rate and m the number of samples. We will apply 1500 iterations and use learning rate $= 0.07$.

```
% 3. Using Stochastic Gradient Descent

% initialize fitting parameters
theta = zeros(2, 1);
grad = zeros(2,1);
iterations = 1500;
alpha = 0.07;

% SGD iterative updates
for num_iterations = 1:iterations
   h = X*theta;    %calculate hypothesis function

   grad = (1/m)*(X.'* (h-y));    %calculate gradient
   theta = theta -alpha*grad;    %update theta

end
```

All methods, finally, converge to the same coefficients vector

$$W = a_2 = \theta = \begin{bmatrix} 0.7502 \\ 0.0639 \end{bmatrix}$$

This can be clarified throught the following plots.



(a) *fitlm* (b) *regress* (c) SGD

Fig. 4: Simple Linear Regression Methods

3.2. Multivariate Linear Regression

In the Multivariate Linear Regression part, we need a dataset containing two or more predictors for the response variable. Here, we will use just two. Hence, it is possible to visualise our datapoints in a 3-D space.

For the testing, we will try two methods. However, before starting for both of them it is necessary to add the intercept term in x variable.
Our first approach is by using the Matlab's built-in function *mvregress*.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1. Using 'mvregress' function: "mvregress(x,y)" returns the estimated
% coefficients for a multivariate normal regression of the d-dimensional
% responses in Y on the design matrices in X.
beta1=mvregress(X,y)
```

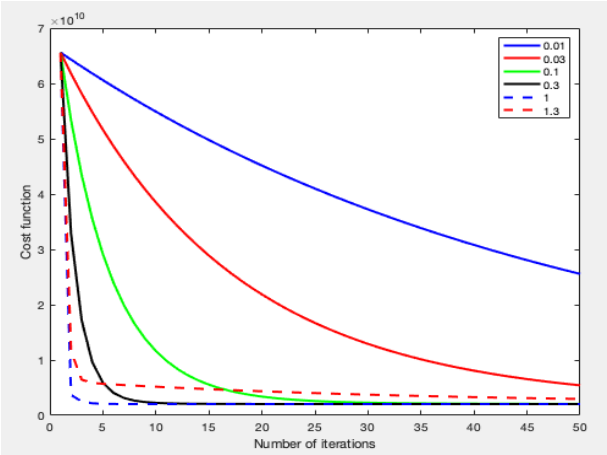
Also, we will apply the SGD as was introduced to the Simple Linear Regression part. However, here we will try various learning rates, in order to test which converges faster within 100 iterations. We need to note that considering we have two variables for X, we need to apply feature scaling in order to control the weights, before moving to the updates.

```
%initialize parameters
figure
iters= 100;
alpha = [0.01, 0.03, 0.1, 0.3, 1, 1.3];
plotstyle = {'b', 'r', 'g', 'k', 'b--', 'r--'};

theta_grad_descent = zeros(size(X_scaled(1,:)));   %to store the values of theta of the best learning rate

for alpha_i = 1:length(alpha)
    theta = zeros(size(X_scaled,2),1);
    Jtheta = zeros(iters, 1);
    for i = 1:iters
        h = X_scaled * theta;
        Jtheta(i) = (1/(2*m)).*(h-y)'* (h -y);      %cost compute
        grad = (1/m)*(X_scaled.'* (h-y));           %grad compute
        theta = theta - alpha(alpha_i).*grad;       %theta update
    end

    plot(1:50, Jtheta(1:50),char(plotstyle(alpha_i)),'LineWidth', 2)   %plot cost vs iter_num over alpha values
    hold on
```

As expected from Simple Linear Regression tests, both *mvregress* and SGD methods return the same coefficients

$$beta1 = theta\_normal = 10^4 \cdot \begin{bmatrix} 8.9598 \\ 0.0139 \\ -0.8738 \end{bmatrix}$$

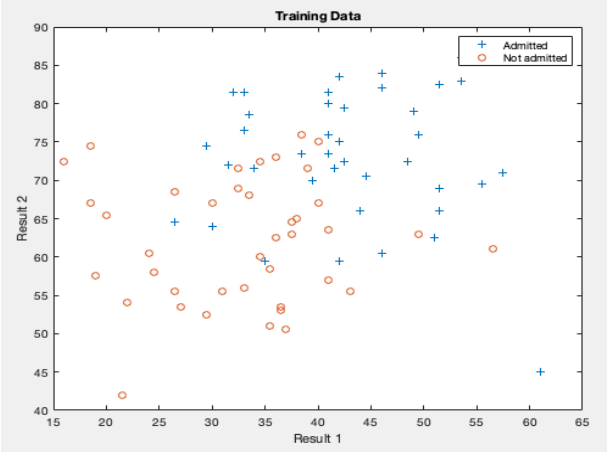
Here as theta_normal we refer to theta found on the SGD method, adapted in the unscaled data.
The intersting part in this experiment is to visualise the convergence comparison, over different learning rates.



As we can see, the best performance is achieved for $\alpha = 1$. However, all tested values finally converge.

3.3. Logistic Regression

For the classification part, using Logistic Regression algorithm we will apply three implementations. The dataset used contains two predictors, while the response variable is a class. In the plot we use blue coloured $'+'$ and orange coloured $'o'$ to seperate the two categories.



We will try to find the decision boundary, which separates best these two classes. We begin with the built in *fitglm* function.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1. Using 'fitglm' function: "fitglm(x,y)" returns a generalized linear model of
% the responses y, fit to the data matrix X.
B= fitglm(X,y,'linear', 'distr', 'binomial' )
coeffs=B.Coefficients.Estimate;
```

Now, for applying SGD method we first need to scale our predictors and define the sigmoid function, which will be used for calculating the cost. We will use learning rate $= 0.1$ and run 1500 iterations.

```
% Define the sigmoid function
g = inline ('1.0 ./(1.0 + exp (-z))');

% SGD IMPLEMENTATION HERE

for num_iterations = 1:iterations
   h = g(X_scaled*sgd_theta);

   grad = (1/m)*(X_scaled.'* (h-y));
   sgd_theta = sgd_theta - alpha*grad;
end
```

Our last approach is based on the Newton's method. This algorithm updates the cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}log(h_\theta(x^{(i)})) - (1-y^{(i)})log(1-h_\theta(x^{(i)})))$$

using the rule

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta J$$

where

$$\nabla_\theta J = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

and

$$H = \frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x^{(i)}(x^{(i)})^T]$$

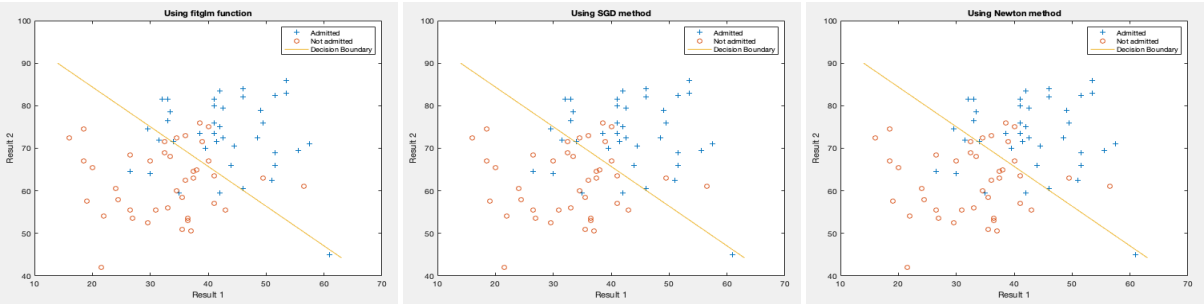
```
for num_iters = 1:iterations
    h = g(X*n_theta);

    % Calculate gradient and hessian.
    grad = (1/m)*(X.'* (h-y));
    H = X.'* diag(h) * diag(1-h) * X;

    % Calculate J (for testing convergence)
    J(num_iters) = (1/m)*sum( (-y).*log(h) - (1-y).*log(1-h) );

    n_theta = n_theta-H\grad;
end
```

Here we need to make clear that Newton's method converges really fast. Someone could confirm that by setting the number of iterations at a low number (e.g 15). However, for the comparison purpose we used 1500 iterations, as many as we did with the SGD.
For this number of updates fitglm and Newton's method give the exact same coefficients

$$coeffs = n\_theta = \begin{bmatrix} -16.3787 \\ 0.1483 \\ 0.1589 \end{bmatrix}$$

The difference with SGD is very small, someone could say negligible as we can see in Figure 5, as it returns

$$theta\_normal = \begin{bmatrix} -16.3772 \\ 0.1483 \\ 0.1589 \end{bmatrix}$$

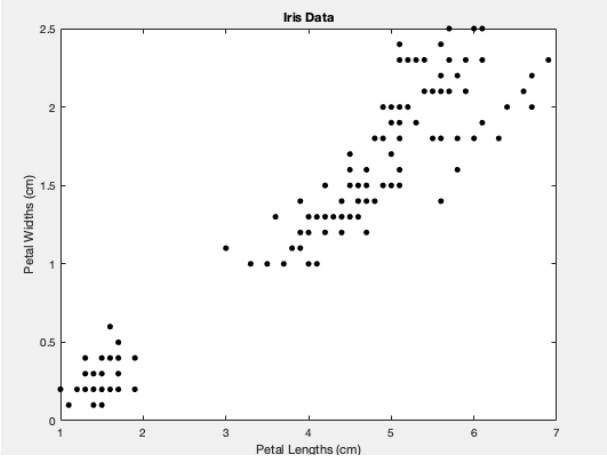


(a) *fitglm* (b) *SGD* (c) Newton's

Fig. 5: Logistic Regression Methods

3.4. K-means Clustering

For the unsupervised learning part, we will apply two different implementations on the K-means algorithm. The dataset used is previewed in the following plot.



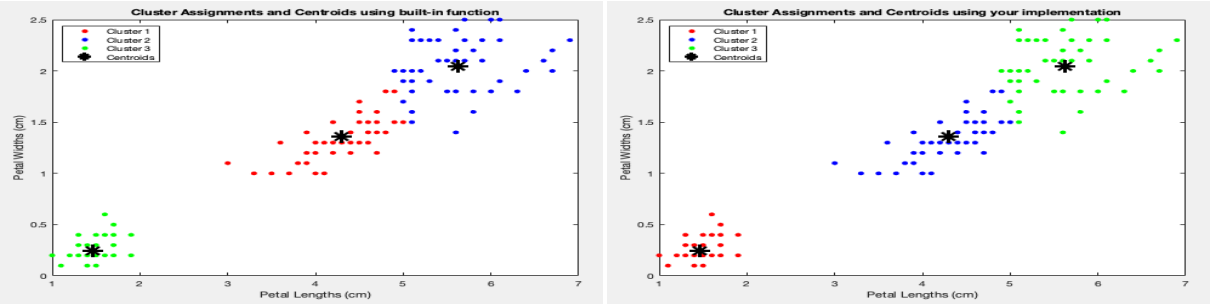
What we will basically try to do is to classify those unlabeled data in K clusters. First way to achieve that is by using the built-in *kmeans* function.

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1. Using 'kmeans' function: "kmeans(x,k)" performs k-means clustering to
% partition the observations of the n-by-p data matrix X into k clusters,
% and returns an n-by-1 vector (idx) containing cluster indices of each observation.
% Rows of X correspond to points and columns correspond to variables.

rng(1); % For reproducibility
[idx,C] = kmeans(X,K);

x1 = min(X(:,1)):0.01:max(X(:,1));
x2 = min(X(:,2)):0.01:max(X(:,2));
[x1G,x2G] = meshgrid(x1,x2);
XGrid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot

idx2Region = kmeans(XGrid,K,'MaxIter',max_iterations,'Start',C);
```

Also, we made an implementation based on the mathematical foundamentals of the algorithm.

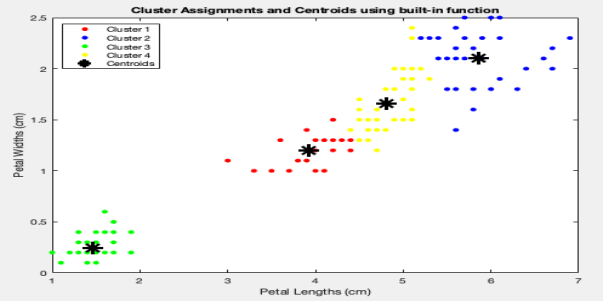
```
for i=1:max_iterations                          %run max_iterations number of iterations
     for num_sample = 1:length(X)               %assign each point to a cluster

        %Find the eulidean distance between a sample and the centroids
        temp = X(num_sample,:) - centroids;
        euc_dist = vecnorm(temp,2,2);

        % Get the index of centroid with the minimum distance.
        % That will be the cluster we assign the sample to.
        closest_cluster = find(euc_dist == min(euc_dist));

        % Assign the sample to the cluster.
        cluster_assigned(num_sample) = closest_cluster;

        % Add the samples that belong to the same cluster.
        cluster_sum(closest_cluster,:) = cluster_sum(closest_cluster,:) + X(num_sample,:);

        % Track the number of samples in the cluster.
        cluster_samples(closest_cluster) = cluster_samples(closest_cluster)+1;

     end
    % Update the centroids by the mean of all the points that belong to
    % that cluster.
    centroids =  cluster_sum ./ cluster_samples;

    % Reset to zeros for the next iteration
    cluster_sum = zeros(K,2);
    cluster_samples = zeros(K,1);
end
```

In both scipts, as K is defined the number of the clusters given by the user. We tried $K = 3,4,5$.
For $K = 3$ the results were exactly the same for both methods (Figure 6).
However, for $K = 4,5$ the outcome differs from one approach to the other. We can visualise this observation in Figures 7 and 8 respectively.

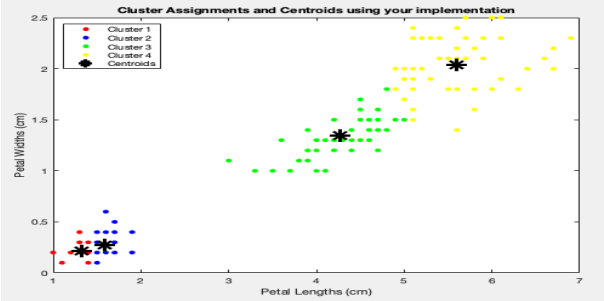


(a) *kmeans* (b) *own implementation*

Fig. 6: K-means with 3 clusters

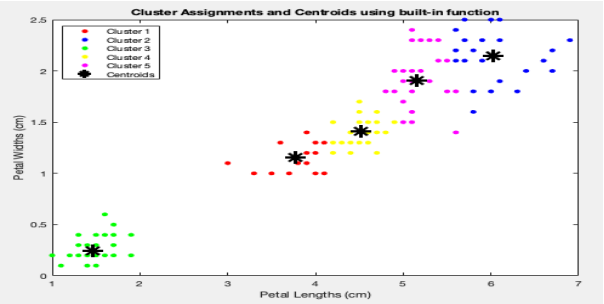


(a) *kmeans*

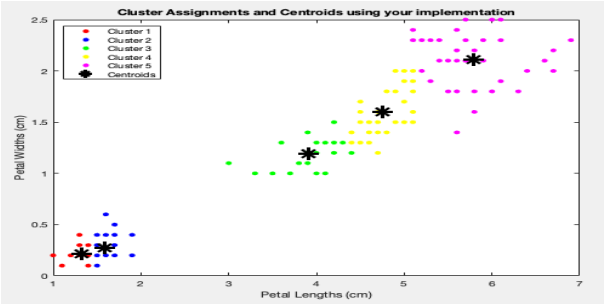


(b) *own implementation*

Fig. 7: K-means with 4 clusters



(a) *kmeans*



(b) *own implementation*

Fig. 8: K-means with 5 clusters

4. Conclusion

There are many ways to approach the operation of a Machine Learning algorithm. Most of the times the results are the same through the different solution manners. However, there exist cases where the final outcome differs. This discrepancy variates, beginning with small contrasts and sometimes may end up to completely inconsistent results.

In the present project we examined possible disagreements between different implementations, imprinting the same algorithm. In both Linear and Logistic Regression there was no effect at all, as all scripts ended up at the exact same results. However for the K-means method, we observed an unexpected upshot. Although for $K = 3$ required clusters the two methods returned the same categores, when we turned $K = 4$ *or* 5 points were classified completely different.

Summing up, we conclude that it would be efficient for any research or tests we go through, to evaluate our final results over unsimilar algorithms and methods, as different approaches may provide different inferences.