# 2198-CSE-6363-001 Machine Learning

# Project 1 Report

The given dataset is (http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data).

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Predicted attribute: class of iris plant.

**Attribute Information:**

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

## 1. Training the dataset:

In order to understand the dependencies between variables, I have implemented the covariance operation. This will define the correlation function between the arrays. It will receive two arrays as parameters and will return the covariance(x,y) value.

Then, I tried to compare the implemented function with the NumPy function.

Values used in code:

print (covariance ([2,4,8], [3,6,9]))

print (np.cov([2,4,8], [3,6,9]))

Please Note that I calculated cov (a,b), and NumPy generated a matrix of all the combinations cov(a,a), cov(a,b), so the result should be equal to the values (1,0) and (0,1) of that matrix

This was just a minimal test of the correlation function as defined earlier, receive two arrays, such as covariance, and use them to get the final value.

## 2. Classification:

I tested this function again with two sample arrays, and compare this with the (0,1) and (1,0) values of the correlation matrix from NumPy:

Values used in code:

print (correlation ([8,6,7,5], [4,3,2,1]))

print (np.corrcoef ([8,6,7,5], [4,3,2,1]))

I atleast got results in unidecimals, therefore correlation is present in the arrays

Now seaborn is used to get a visual representation of the trained set,

Seaborn's pairplot function provides a complete graphical summary of all the variable pairs, represented as scatterplots, and a representation of the univariate distribution for the matrix diagonal.

Lets' select two variables that, from our initial analysis, have the property of being linearly dependent. They are petal_width and petal_length:

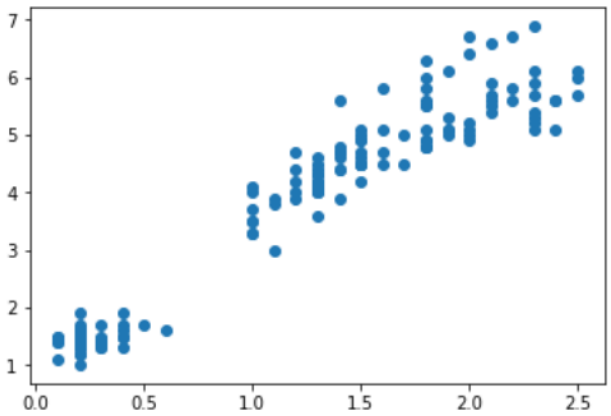
X=iris2['petal_width'] Y=iris2['petal_length']

Let's now take a look at this variable combination, which shows a clear linear tendency

```
In [89]: X=iris2['petal_width']
         Y=iris2['petal_length']

In [36]: plt.scatter(X,Y)

Out[36]: <matplotlib.collections.PathCollection at 0x19bc70a9438>
```




This is the current distribution of data that we will try to model with our linear prediction function.

## 3. Cross validation:

### *Forming the prediction function*

Firstly, define the function that will abstractedly represent the modeled data, in the form of a linear function, with the form *y=beta*x+alpha*:

```
def predict(alpha, beta, x_i): return beta * x_i + alpha
```

### *Defining the Error function*

Now to define the function that will show us the difference between predictions and the expected output during training. There are two main alternatives: measuring the absolute difference between the values (or L1), or measuring a variant of the square of the difference (or L2). I have defined both versions, including the first formulation inside the second.

Code:

```
def error(alpha, beta, x_i, y_i): #L1 return y_i - predict(alpha, beta, x_i)

def sum_sq_e(alpha, beta, x, y): #L2

return sum(error(alpha, beta, x_i, y_i) ** 2 for x_i, y_i in zip(x, y))
```

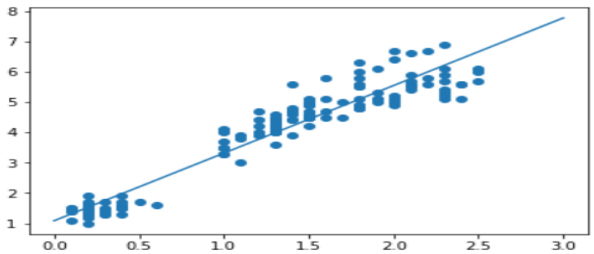
### *Correlation fit*

Now, I define a function implementing the correlation method to find the parameters for our regression and run the fitting function and print the guessed parameters:

Now to graph the regressed line with the data in order to intuitively show the appropriateness of the solution.

**Result:** This is the final plot I got with the recently calculated slope and intercept

```
In [116]: plt.scatter(X,Y)
          xr=np.arange(0,3.75)
          plt.plot(xr,(xr*beta)+alpha)
Out[116]: [<matplotlib.lines.Line2D at 0x19bce47a358>]
```




```
In [ ]:
```