Machine Learning Programming
Exercise Session Part 1

19th September, 2018

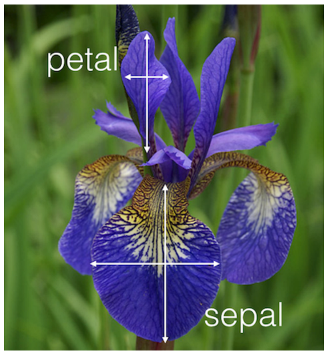
The exercise session is a **non-graded assignment** that is here to help you familiarize with the level of Matlab programming you are expected to provide during this **Machine Learning Programming course**. It is also here to introduce you some basics concept of Machine Learning such as *dataset visualization*, *dataset splitting*, *model training* and *best model selection*. Be aware that although it is not graded, some of the functions you will develop here could be useful for the rest of the course. Let us head to the first part of the exercises that will teach you the basic functions to handle a *dataset*.

1 Loading a dataset, visualizing its content and splitting it for training

In this first part of the exercise session you will see how to load a dataset, create some functions to visualize the data and learn how to properly split it for training Machine Learning (ML) models. First, let us start by loading the `iris` dataset in Matlab. The following commands will import the `iris` dataset and create a `featuresLabel` vector that contains the names of the feature data represented by each column of the `meas` array.

```
In [2]: load fisheriris
        featuresLabel = {'Sepal length','Sepal width','Petal length','Petal width'};
```

The `iris` dataset contains 150 iris flowers of three species: *setosa*, *versicolor* and *virginica*. The features of each flowers are recorded and stored in `meas` while its species is stored in `species` array. You will first define some plotting functions to vizualize the content of the dataset.

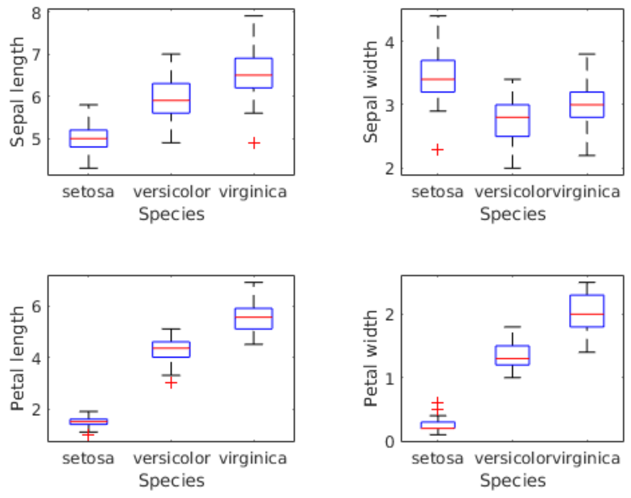


An Iris flower with annotated features

2 Exercise1: Visualizing the content of a dataset

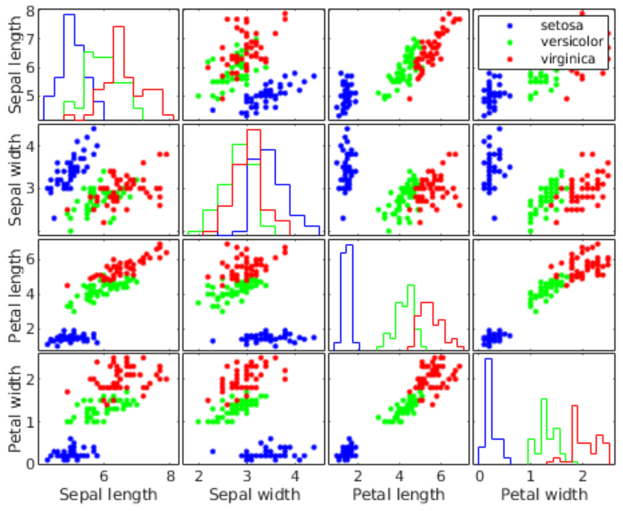
Head first for the `univariateBoxplot.m` file and complete the `univariateBoxplot` function. This function will represent the repartition of each features for each categories using boxplot.

```
In [3]: univariateBoxplot(meas, species, featuresLabel)
```



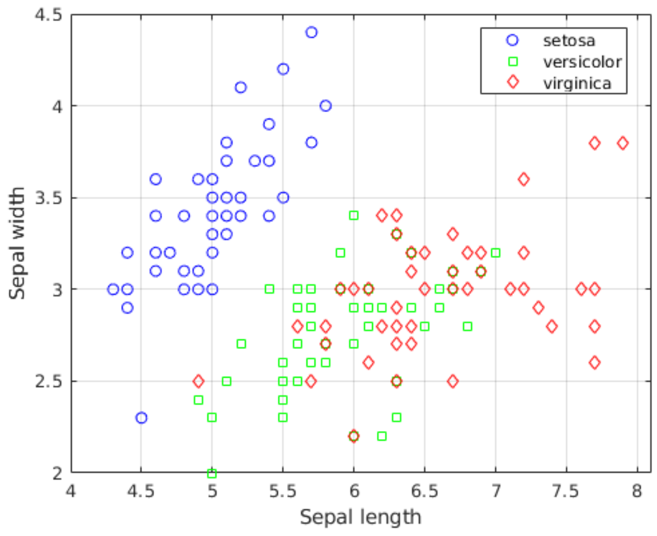
The next plotting function is a *scatter matrix* defined in `scatterMatrix.m`. It represents all the possible 2D plots of your data. The diagonal of the matrix represents an histogram of each feature. To compute and plot this matrix have a look at the `plotmatrix` function in the documentation.

```
In [4]: scatterMatrix(meas, species, featuresLabel)
```



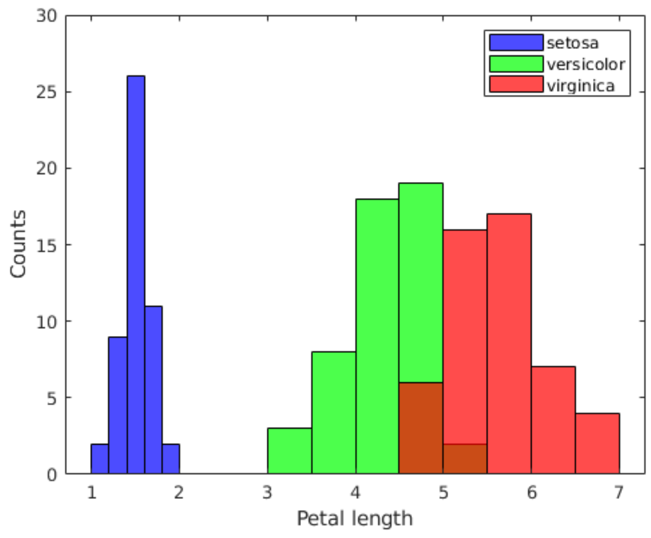
Finally, you will also create the function that render those 2D plots independently in `plot2DSamples.m`. You can change the `axis` variable to select the two features to plot. For example, `axis = [1,2]` should give you the following output.

```
In [5]: axis = [1,2];
        plot2DSamples(meas, species, featuresLabel, axis);
```



If you provide only a scalar number as axis, the function plot return an histogram of the corresponding feature as follow.

```
In [6]: axis = 3;
        plot2DSamples(meas, species, featuresLabel, axis);
```



Complete all the mentionned functions and run `exercise1.m` script. Check that you get the same output as shown in this document. Looking at the data you might already see that the classification of the samples is easier if perform on some specific features.

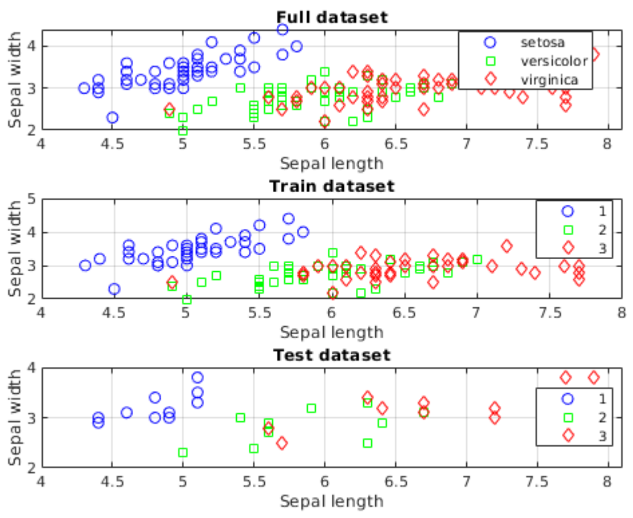
3 Exercise 2: Split a dataset for training

This exercise will cover how to split a dataset for training ML models. First using a splitting ratio, then using a method called *K-Fold cross validation*. Open the `trainTestSplit.m` file and complete the code. The splitted dataset should contain the same repartition of species in both *train* and *test* sets than in the original data. Providing a `seed` will guaranty that you get the exact same splitting each time you run the script.

```
In [7]: seed = rng(42, 'twister');
        validSize = 0.2;
        [xTrain, yTrain, xTest, yTest] = trainTestSplit(meas, species, validSize, seed);
```

Using the plotting functions defined in **Exercice 1** you can represent the data present in both your *train* and *test* sets. The defined `assert` functions will also help you checking that you get the desired results.

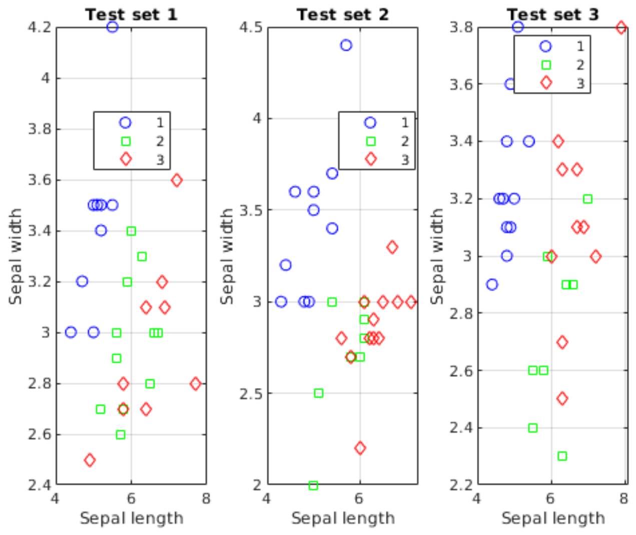
```
In [8]: subplot(3,1,1)
        plot2DSamples(meas, species, featuresLabel, [1,2]);
        title('Full dataset')
        subplot(3,1,2)
        plot2DSamples(xTrain, yTrain, featuresLabel, [1,2]);
        title('Train dataset')
        subplot(3,1,3)
        plot2DSamples(xTest, yTest, featuresLabel, [1,2]);
        title('Test dataset')
```



Now implement the `KFoldSplit` function. *K-Fold cross validation* is a technique where the dataset is splitted into *k folds* of equal (if possible) size. When training a ML model, as you will see in **Part 2** of the exercise session, we select one fold that will be used for testing and the rest is used as training set.

For simplicity, you don't have to consider that the repartition of species in each fold should be similar to the original data and your splitting is not expected to work for values of `k` that are not divisor of the total number of samples in your dataset. Therefore, you can test your splitting with the following values: 1, 2, 3, 5, 6, 10, 15, 25, 30, 50, 75, 150. The following example shows you the plot of the 3 first folds of your dataset for `k=5`.

```
In [9]: k=5;
        for i=1:3
            subplot(1, 3, i)
            % divide the dataset in k fold and set the i-th fold as test set
            [xTrain, yTrain, xTest, yTest] = kFoldSplit(meas, species, k, i, seed);
            plot2DSamples(xTest, yTest, featuresLabel, [1,2]);
            title(['Test set ' num2str(i)])
        end
```


Test set 1
Test set 2
Test set 3
Sepal width
Sepal length