Laboratory 2

In this laboratory we will introduce the IRIS dataset. We will see how we can load and visualize the dataset, and compute some statistics of the data.

The IRIS Dataset

The dataset was introduced in Fisher, R.A. "The use of multiple measurements in taxonomic problems", Annual Eugenics, 7, Part II, 179-188 (1936). The dataset contains information on 150 samples (instances) of iris flowers belonging to 3 different families (classes): iris setosa, iris versicolor and iris virginica. There are 50 samples for each class. For each sample, the dataset provides 4 attributes (features): sepal length (cm), sepal width (cm), petal length (cm), petal width (cm).

Loading the dataset

We want to load the dataset into a numpy 2-dimensional array of shape (4×150)

$$\mathbf{D} = \begin{bmatrix}\mathbf{x}_1 \dots \mathbf{x}_N\end{bmatrix} \ , \tag{1}$$

where $\mathbf{x}_i$ is the i-th sample of the dataset, and $N = 150$ is the number of samples. Each row of the data matrix corresponds to an attribute (feature), whereas each column represents a sample. We want to also construct a 1-dimensional array of class labels

$$\mathbf{L} = \begin{bmatrix}l_1 \dots l_N\end{bmatrix} \ .$$

Iris setosa will be indicated with value 0, iris versicolor with value 1 and iris virginica with value 2.
The dataset is provided as a comma-separated values (csv) file, where each line represents a data point (sample). Each line contains the four features and the family name, for example, for the j-th sample:

```
5.1,3.5,1.4,0.2,Iris-setosa
```

we will have

$$\mathbf{x}_j = \begin{bmatrix}5.1\\3.5\\1.4\\0.2\end{bmatrix} \ , l_j = 0$$

The dataset can be found in `Solution/iris.csv`. Write a `load` function that, given the dataset filename, returns the (4×150) data matrix, and the corresponding 1-dimensional array of size 150 containing the class labels. Suggestion: for each line, create a 4×1 vector with the feature values. Store all vectors in a list, and then concatenate the vectors.

Visualizing the dataset

We want to visualize the distribution of the different features for the different classes.

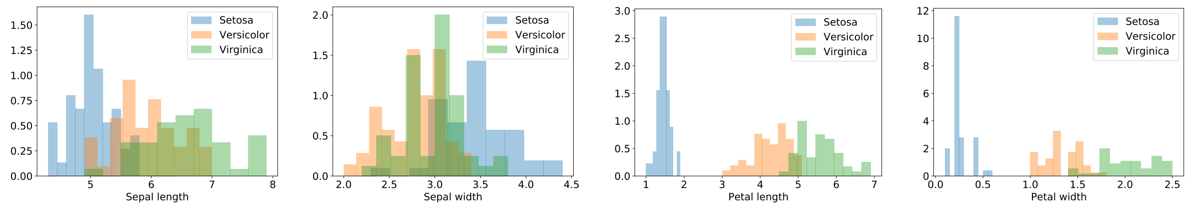
- For each feature, plot the corresponding histogram for each class. You can use `matplotlib.pyplot.hist`, providing a 1-dimensional numpy array with the target data. Try with different number of bins.

  You can normalize the histogram setting the parameter `density = True`. To create a new figure, you can use `matplotlib.pyplot.figure`. To visualize the current figure(s), use `matplotlib.pyplot.show`.

  NOTE: `matplotlib.pyplot.show` shows the current image(s) in different windows, and blocks the execution until the windows are closed.

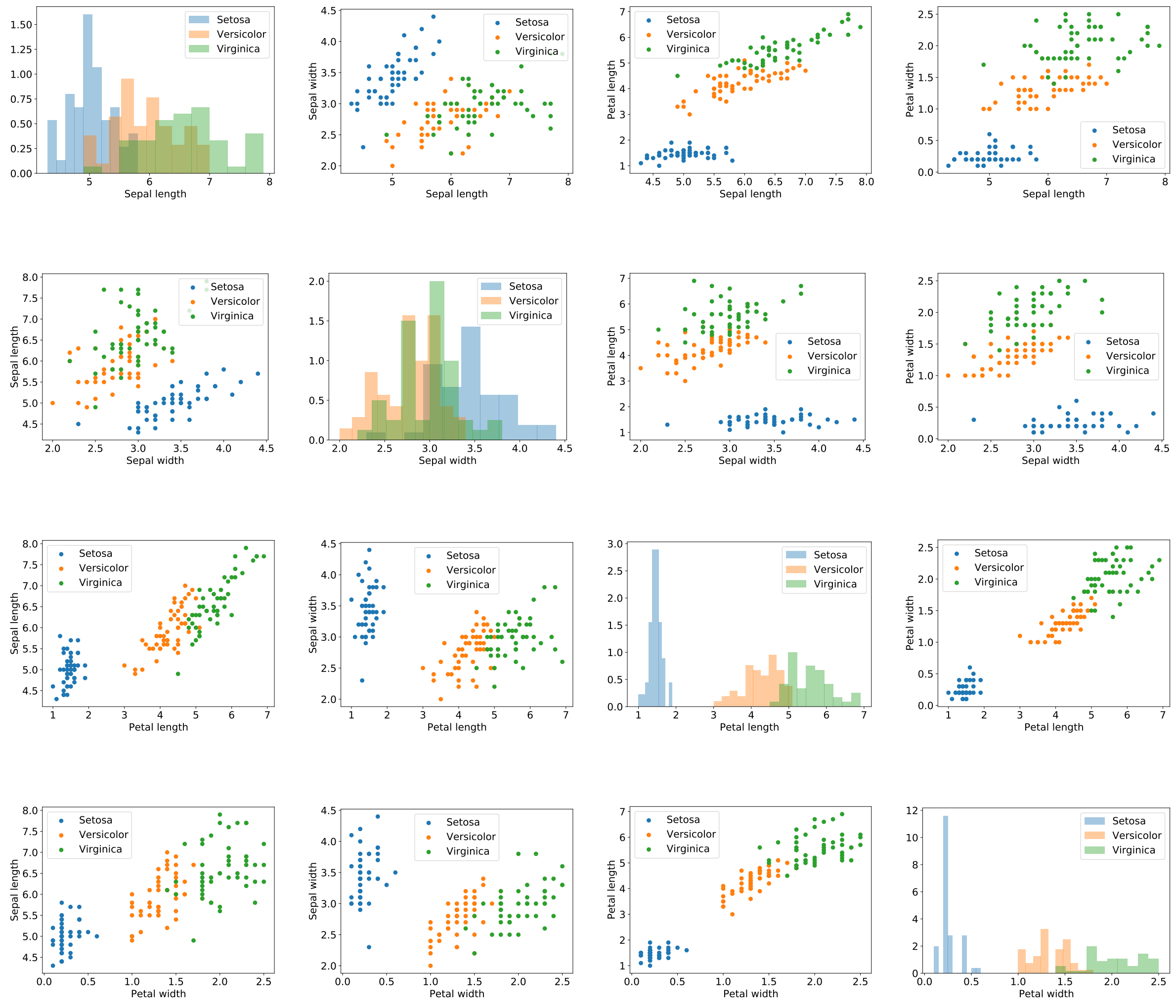
  Suggestion: start extracting from the data matrix the parts corresponding to the different classes, then proceed with the plots. If `D` is the data matrix, and `L` is the label vector (1-dimensional), we can build a mask for each class and use it to filter the columns of `D`:

  ```
  M0 = (L == 0)
  D0 = D[:, M0] # or equivalently D0 = D[:, L==0]
  ```



We can observe that there is large overlap for the first two features, whereas values for the third and fourth features of iris-setosa are well separated from those of the other two flower families

- We now consider pairs of values. Visualize the scatter plots of the different feature pairs for each class.



Statistics

We now look at how to compute simple statistics and transformations of the data.

Dataset mean

The empirical dataset mean is defined as

$$\boldsymbol{\mu} = \frac{1}{N}\sum_i \mathbf{x}_i$$

where $\mathbf{x}_i$ are the N samples of the data matrix. We can compute the dataset mean using a `for` loop:

```
mu = 0

for i in range(D.shape[1]):
    mu = mu + D[:, i:i+1]

mu = mu / float(D.shape[1])
```

where `D[:, i:i+1]` is the i-th column of `D`, i.e. $\mathbf{x}_i$. The `for` loop is, in general, slow. `Numpy` allows computing the mean of an array through the method `.mean`. The method allows specifying an axis — for 2-D arrays, `axis = 0` allows computing the mean over rows, whereas `axis = 1` allows computing the means over columns. We can thus compute the dataset mean as

```
mu = D.mean(1).reshape(D.shape[0], 1)
```

Pay attention to the shape of `D.mean(1)`: it's a 1-D array, thus we convert it to a column vector through the `.reshape` method.
We now exploit broadcasting to *center the data*, i.e. to remove the mean from all points:

```
DC = D - mu
```

Notice that we want the mean to be a column vector, so we first reshape it.
Suggestion: we will often have to reshape 1-D vectors as column or row vectors. Write a function `mcol` and a function `mrow` that implements the reshaping

You can try plotting again the centered data.

Covariance matrix, variance and standard deviation of features

The empirical covariance matrix is defined as

$$\mathbf{C} = \frac{1}{N}\sum_i (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$$

We can compute the covariance matrix using a for loop:

```
C = 0

for i in range(D.shape[1]):
    C += (D[:, i:i+1] - mu) @ (D[:, i:i+1] - mu).T

C = C / float(D.shape[1])
```

Also in this case, the loop is slow. We can observe that we can arrange computations to express the covariance matrix as

$$\mathbf{C} = \frac{1}{N}\sum_i \mathbf{D}_c\mathbf{D}_c^T$$

where $\mathbf{D}_c$ is the centered data matrix. Using numpy. we can simply compute

```
C = ((D - mu) @ (D - mu).T) / float(D.shape[1])
```

or, using the centered data matrix we computed earlier,

```
C = (DC @ DC.T) / float(D.shape[1])
```

Finally, we can compute the variance of each feature. The variance corresponds to the diagonal of the covariance matrix, and is the square of the standard deviation. Both the variance and standard deviation represent the dispersion of a feature with respect to the class mean, i.e., larger variance implies that, on average, the squared distance of samples from the dataset mean is larger, whereas a small variance indicates that samples are closer to the dataset mean.
The vector of all variances (i.e. the variance of each feature stacked in a column vector) and the vector of all standard deviations can be computed as

```
var = D.var(1)
std = D.std(1)
```

(again, pay attention that the results are 1-D vector, if we need column vectors we need to reshape the results). Repeat the computations for each class, and compare the results. Are there features that show significant differences across classes? Make a similar comparison for the class means. In the future we will see methods that are able to exploit these differences to perform inference.

Project

The project task consists of a binary classification problem. The goal is to perform fingerprint spoofing detection, i.e. to identify genuine vs counterfeit fingerprint images. The dataset consists of labeled samples corresponding to the genuine (True, label 1) class and the fake (False, label 0) class. The samples are computed by a feature extractor that summarizes high-level characteristics of a fingerprint image. The data is 6-dimensional.
The training files for the project are stored in file `Project/trainData.txt`. The format of the file is the same as for the Iris dataset, i.e. a csv file where each row represents a sample. The first 6 values of each row are the features, whereas the last value of each row represents the class (1 or 0). The samples are not ordered.
Load the dataset and plot the histogram and pair-wise scatter plots of the different features. Analyze the plots.

1. Analyze the first two features. What do you observe? Do the class overlap? If so, where? Do the class show similar mean for the first two features? Are the variances similar for the two classes? How many modes are evident from the histograms (i.e., how many “peaks” can be observed)?

2. Analyze the third and fourth features. What do you observe? Do the class overlap? If so, where? Do the class show similar mean for the first two features? Are the variances similar for the two classes? How many modes are evident from the histograms?

3. Analyze the last two features. What do you observe? Do the class overlap? If so, where? How many modes are evident from the histograms? How many clusters can you notice from the scatter plots for each class?