Programming assignment 1: k-Nearest Neighbors classification

In [1]:

```
import numpy as np
from sklearn import datasets, model_selection
import matplotlib.pyplot as plt
%matplotlib inline
```

Introduction

For those of you new to Python, there are lots of tutorials online, just pick whichever you like best :)

If you never worked with Numpy or Jupyter before, you can check out these guides

- https://docs.scipy.org/doc/numpy-dev/user/quickstart.html (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)
- http://jupyter.readthedocs.io/en/latest/ (http://jupyter.readthedocs.io/en/latest/)

Your task

In this notebook code to perform k-NN classification is provided. However, some functions are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring, which specifies the format of input and output. Write your code in a way that adheres to it. You may only use plain python and `numpy` functions (i.e. no scikit-learn classifiers).

Once you complete the assignments, export the entire notebook as PDF using nbconvert (https://nbconvert.readthedocs.io/en/latest/) and attach it to your homework solutions. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Load dataset

The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set (https://en.wikipedia.org/wiki/Iris_flower_data_set)) is loaded and split into train and test parts by the function `load_dataset`.

In [2]:

```
def load_dataset(split):
    """Load and split the dataset into training and test parts.

    Parameters
    ----------
    split : float in range (0, 1)
        Fraction of the data used for training.

    Returns
    -------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    y_test : array, shape (N_test)
        Test labels.
    """
    dataset = datasets.load_iris()
    X, y = dataset['data'], dataset['target']
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, random_st
ate=123)
    return X_train, X_test, y_train, y_test
```

In [3]:

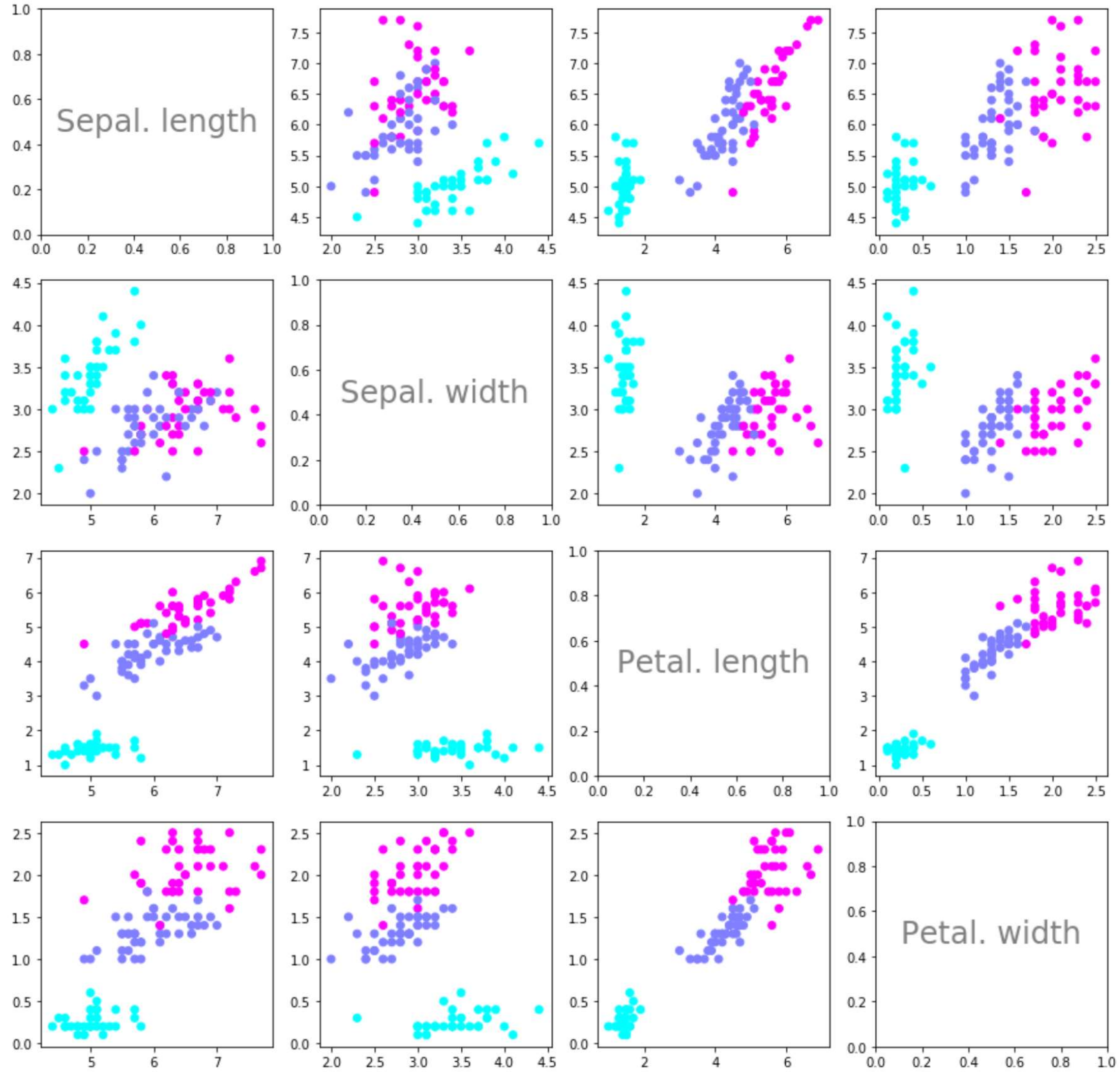
```
# prepare data
split = 0.67
X_train, X_test, y_train, y_test = load_dataset(split)
```

Plot dataset

Since the data has 4 features, 16 scatterplots (4x4) are plotted showing the dependencies between each pair of features.

In [4]:

```
f, axes = plt.subplots(4, 4,figsize=(15, 15))
for i in range(4):
    for j in range(4):
        if j == 0 and i == 0:
            axes[i,j].text(0.5, 0.5, 'Sepal. length', ha='center', va='center', size=24, alpha=.5)
        elif j == 1 and i == 1:
            axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center', size=24, alpha=.5)
        elif j == 2 and i == 2:
            axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center', size=24, alpha=.5)
        elif j == 3 and i == 3:
            axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center', size=24, alpha=.5)
        else:
            axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.cm.cool)
```



Task 1: Euclidean distance

Compute Euclidean distance between two data points.

In [10]:

```
def euclidean_distance(x1, x2):
    return np.linalg.norm(np.subtract(x1,x2))
```

Task 2: get k nearest neighbors' labels

Get the labels of the *k* nearest neighbors of the datapoint *x_new*.

In [64]:

```
def get_neighbors_labels(X_train, y_train, x_new, k):
    euc_l = np.empty((0, 2), int)
    # getting an array [euclidean distance, label]
    for i in range(len(X_train)):
        euc_l = np.append(euc_l, [[euclidean_distance(X_train[i], x_new), y_train[i]]],
 axis=0)
    # sorting an array through the euclidean distance
    euc_l = euc_l[euc_l[:, 0].argsort()]
    return euc_l[0:k, 1]
```

Task 3: get the majority label

For the previously computed labels of the *k* nearest neighbors, compute the actual response. I.e. give back the class of the majority of nearest neighbors. Think about how a tie is handled by your solution.

In [87]:

```
def get_response(neighbors, num_classes=3):
    m_dict = {'max': [0, 0]}
    """
    generating a dictionary where
    m_dict['max'] = [label, number of values]
    m_dict[label] = number of values
    """
    for i in neighbors:
        if i in m_dict:
            m_dict[i] += 1
            if m_dict[i] > m_dict['max'][1]:
                m_dict['max'][0] = i
                m_dict['max'][1] = m_dict[i]
        else:
            m_dict[i] = 1
            if m_dict['max'][1] == 0:
                m_dict['max'][0] = i
                m_dict['max'][1] = 1
    return m_dict['max'][0]
```

Task 4: compute accuracy

Compute the accuracy of the generated predictions.

In [88]:

```
def compute_accuracy(y_pred, y_test):
    counter = 0
    l = len(y_pred)
    for i in range(l):
        if y_pred[i] == y_test[i]:
            counter +=1
    return counter/l
```

In [89]:

```
# This function is given, nothing to do here.
def predict(X_train, y_train, X_test, k):
    y_pred = []
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        y_pred.append(get_response(neighbors))
    return y_pred
```

Testing

Should output an accuracy of 0.9473684210526315.

In [94]:

```
# prepare data
split = 0.67
X_train, X_test, y_train, y_test = load_dataset(split)
print('Training set: {0} samples'.format(X_train.shape[0]))
print('Test set: {0} samples'.format(X_test.shape[0]))

# generate predictions
k = 3
y_pred = predict(X_train, y_train, X_test, k)
accuracy = compute_accuracy(y_pred, y_test)
print('Accuracy = {0}'.format(accuracy))
```

```
Training set: 112 samples
Test set: 38 samples
Accuracy = 0.9473684210526315
```

In [ ]:

```

```