AI for Biotechnology
Exercise 2

Prof. Dr. Dominik Grimm
Bioinformatics Research Lab
TUM Campus Straubing for Biotechnology and Sustainability

Exercise E2.1

The euclidean distance between two points $\boldsymbol{x}^{[a]} \in \mathbb{R}^{1\times m}$ and $\boldsymbol{x}^{[b]} \in \mathbb{R}^{1\times m}$ is defined as follows:

$$d(\boldsymbol{x}^{[a]}, \boldsymbol{x}^{[b]}) = \sqrt{\sum_{i=1}^{m} \left(\boldsymbol{x}_i^{[a]} - \boldsymbol{x}_i^{[b]}\right)^2} \tag{1}$$

To implement Equation 1 in Python a `for` loop is needed. This could be computationally expensive when m is large (several thousand to millions of features). The dot-product between two arbitrary vectors $\boldsymbol{a} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{b} \in \mathbb{R}^{n\times 1}$ is defined as follows:

$$\boldsymbol{a}^\top \boldsymbol{b} = (a_1, \ldots, a_n) \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \sum_{i=1}^{n} a_i b_i \tag{2}$$

a) Show that you can reformulate Equation 1 as a dot product.

b) Implement both versions in Python (`euclidean_distance_naive` and `euclidean_distance_dot`) and test your implementations with the following code:

```
import numpy as np
import time

def euclidean_distance_naive(a,b):
    #Write down your code here

def euclidean_distance_dot(a,b):
    #Write down your code here

#Simple Test with row vectors of size 1 x 2
a = np.array([[3,5]])
b = np.array([[6,9]])
print(euclidean_distance_naive(a,b))
print(euclidean_distance_dot(a,b))
print()

#Test with random row vectors of size 1 x 10 Million
#Random numbers with seed
np.random.seed(42)
m = 10**7 #10 Million Features
a = np.random.random((1,m))
b = np.random.random((1,m))

#stop time of computation
start = time.process_time()
#compute distance
```

```python
print(euclidean_distance_naive(a,b))
delta = (time.process_time()-start)
print("Computation Time Naive: %f s" % delta)

start = time.process_time()
print(euclidean_distance_dot(a,b))
delta = (time.process_time()-start)
print("Computation Time Fast: %f s" % delta)
```

Exercise E2.2

We use the `scikit-learn` package to simulate some toy data as illustrated in Figure 1:

```python
%matplotlib inline
import sklearn.datasets as datasets
import pylab as pl
#Simulate Toy Dataset
X, y = datasets.make_circles(n_samples=150, shuffle=True,
                             noise=0.2, random_state=42,
                             factor=0.1)
```

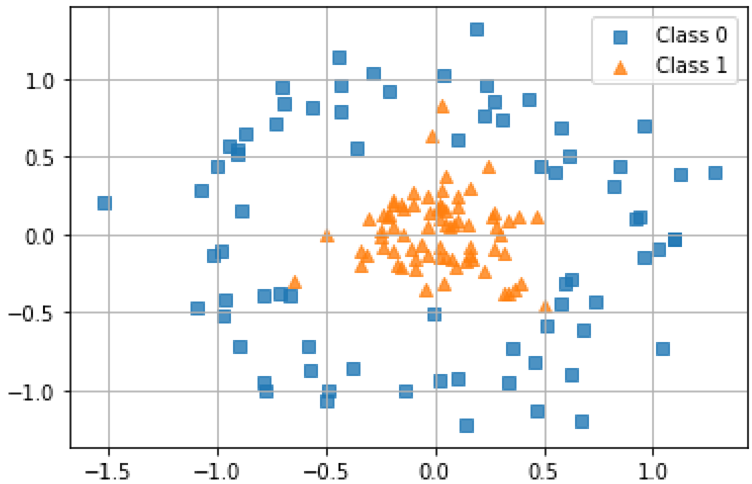

Figure 1: Toy data simulated with `scikit-learn`

a) Write a Python snippet to determine how many samples, features and classes the dataset has. Return also the number of samples for each of the classes (hint: have a look at the NumPy Method `unique`).

b) Reproduce the plot shown in Figure 1 using the `Matplotlib` library (hint: have a look at the `marker` argument of the scatter function using the matplotlib documentation).

c) Add the two query points $q_1 = (-0.8, 0.3)$ and $q_2 = (-0.1, 0.1)$ to the scatter plot.

Exercise E2.3

Implement a function to classify a given query point q using the k-Nearest-Neighbor algorithm from scratch. Do not use the algorithms implemented in `scikit-learn`. The function should have the name `knn_predict` and should accept the arguments, `X`, `y`, `query_point` and `k`. Test your implemented function using the two query points q_1 and q_2 from the previous exercise for $k \in \{1, 10, 50, 100\}$. How are the query points classified using your implementation? (Hint: the `NumPy` functions `unique`, `argsort`, `argmax` and `norm` could be useful. However, there are several ways how to implement this function. You can also compare your implementation with the results from scikit-learn.)