MACHINE LEARNING COURSEWORK 2020

Provide answers to all 4 problems. Maximum number of possible mark points are given below each problem. Each question adds up to **[100 points]**, which will be averaged over all the questions, multiplied by 0.2 and added to the marks obtained from the exam [80 points].

Much of reporting in the domain of machine learning is done using Latex, which is a document preparation system for high-quality typesetting available for Linux, Windows and Mac. Present your solution within this Latex file i.e. edit the source, type your solution in `"\solution { type your solution here }` and compile the file. The solutions must be clearly presented with equations or figures if appropriate but also succinctly. Extensive copying of lecture material indicates weak understanding and poor effort, so points will be deducted for including irrelevant information. Use mathematical formulations, rather than verbal descriptions, as much as possible.

Do not: remove the questions, or change the document style.

Submission deadline: See the date on Blackboard.

Submission format: Generate a pdf file, rename it to your `username.pdf`, e.g. `ab01346.pdf`, and submit to Blackboard as the coursework assignment.

Declaration:
I, Baptiste Provendier, pledge that this assignment is completely my own work, and that I did not make use of the coursework from any other person, and that I did not allow any other person to make use of portions of my coursework. I understand that if I violate this honesty pledge, I am subject to disciplinary action pursuant to the appropriate sections of Imperial College London policies.

1. Practical machine learning (ML) scenario.

Consider a fitness tracker for an accurate training program. Design an ML predictor for energy estimation within 10s intervals. Available measurements are: the number of steps, the altitude difference, the average heart rate, and the body weight. The processing hardware constrains the predictor to have 13 parameters only.

The predicted energy would be compared to a safety threshold set by the athlete to reduce the training effort if the threshold is exceeded. Note that exceeding the threshold due to an underestimation of the energy by the predictor will require 3 times longer recovery time compared to overestimating.

Formulate this as an ML problem and design a loss function. The ML solution will be useful if we can guarantee that the test error does not differ from the training error by more than 10% with 90% confidence.

a) Identify relevant ML components and formulate it as an ML problem. [40 marks]

Solution:

- The input $\mathbf{x}_i \in \mathbb{R}^4$ belong to the input set $\mathscr{X}$, itself part of the dataset $\mathscr{D}$. x $\in \mathscr{X} \subset \mathscr{D}$. The four input features are the number of steps, the altitude difference, the average heart rate and the body weight, each given per interval of 10s.

- The target function f gives an output $\mathbf{y}_i$ representing the enrgy estimation within the 10s interval from an input vector $\mathbf{x}_i$ such that f: $\mathscr{X} \rightarrow \mathscr{Y}$ with $\mathbf{x}_i \in \mathscr{X}$ and $\mathbf{y}_i \in \mathbb{R}$. Hence, we are dealing with a regression problem.

- The hypothesis class $\mathscr{H} \subset [h : \mathscr{X} \rightarrow \mathscr{Y}]$, we take a hypothesis g such that g$\simeq$f. As we will explain in 1)c), we take d_{vc}=13.

- We use n $\geq$ 125,007 given that we have training data $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2) ... (\mathbf{x}_n, \mathbf{y}_n)$.

- For the error function, we know that exceeding the threshold due to an underestimation (false negative) is 3 times more costly than an overestimation (false positive). We also define T the threshold defined by the user.

$$\hat{R}_n(h) = \frac{1}{n} \Sigma_n \alpha (h(\mathbf{x}_i) - y_i)^2$$

$$\text{with } \alpha \begin{cases} 3, & \text{if } h(\mathbf{x}_i) < y_i \text{ and } h(\mathbf{x}_i) > T \\ 1, & \text{otherwise} \end{cases}$$

b) Your colleague advised to use features resulting from $\mathbf{x}_i \mathbf{x}_i^\top$, where $\mathbf{x}_i$ is a data sample in the original space. Discuss the implications of using this approach.

[15 marks]

Solution:

First let's compute the matrix $\mathbf{x}_i\mathbf{x}_i^\top$:

$$\mathbf{x}_i\mathbf{x}_i^\top = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} x_1^2 & x_1x_2 & x_1x_3 & x_1x_4 \\ x_2x_1 & x_2^2 & x_2x_3 & x_2x_4 \\ x_3x_1 & x_3x_2 & x_3^2 & x_3x_4 \\ x_4x_1 & x_4x_2 & x_4x_3 & x_4^2 \end{bmatrix}$$

Hence, we are applying a non-linear transformation to the data in the original space into z such that $\Phi(\mathbf{x_i}) = \mathbf{z_i}$, $\mathbf{z}_i \in \mathbb{R}^{16}$. This non-linear transformation increases the complexity of the hypothesis and now the numbers of components exceeds the number of parameters available. However, looking closer at the matrix we observe that we have a lot of repeated terms which increase the dimensionality but do not help with training. If we remove those repeated terms, we get $\mathbf{z}_i \in \mathbb{R}^{10}$, which means we only need 10 parameters to apply a Linear Regression. The hardware capabilities now allow for this transformation.

c) Propose a hypothesis class for this problem. Calculate the approximate number of data points that are needed for training to make the ML predictions useful. Show your calculations. [25 marks]

Solution:

We define a hypothesis class $\mathscr{H}$ as a polynomial of degree 3 with regularisation. We have 4 dimensions (one for each available measurements) each having 3 parameters and the last one is used for the regularisation parameter λ, which makes it 13 parameters in total. We know that d_{vc}=13 and we now try to find the number of data points starting from VC inequality.

$P(|\nu - \mu| < \varepsilon) \leq\ 4n^{\,d_{vc}}\, e^{\frac{-\varepsilon^2 n}{8}} = \delta$

To guarantee the requirements in the problem, $\varepsilon = 0.1$ and the error probability is $\delta = 10\% = 0.1$.

$4n^{\,d_{vc}}\, e^{\frac{-\varepsilon^2 n}{8}} - 0.1 = 0$

We plug the numbers in and we use an algorithm to find the solution for n:

we get $n \simeq 125,007$.

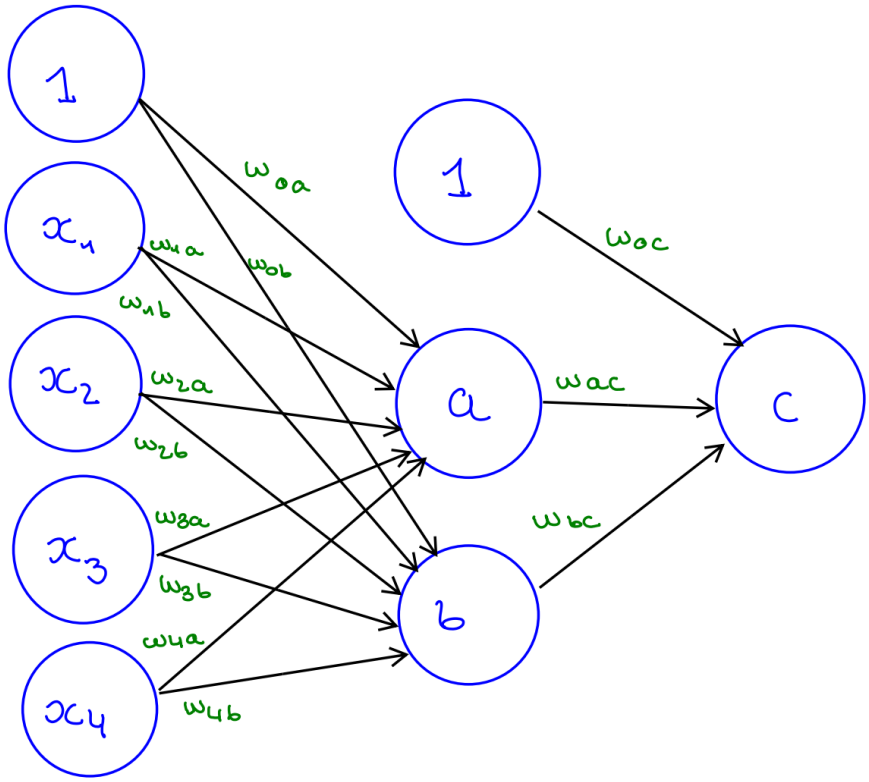
(Note: if we use 4(2n+1) in the VC inequality instead on simply 4n, the estimation becomes $n \simeq 132,849$.)

```cpp
#include <iostream>
#include <cmath>
#include <cstdlib>

int main() {
    double n=1;
    int index= 0;
    double prime = 0;
    for(int i=100000; i < 200000; i++){
        double num = 4*(pow(i,13))*(exp(-0.00125*i))-0.1;
            if(abs(num) < n){
                n = num;
                index = i;
            }
    }
    std::cout << n << std::endl;
    std::cout << index << std::endl;
}
```

d) Design a small neural network architecture with 1 hidden layer to model a predictor suitable for the task defined in this ML problem. Propose the training loss for this network. [20 marks]

Solution:



The goal is to maximise the number of neurons in the hidden layer within the

hardware constraints of 13 parameters. We can fully take advantage of those 13 parameters by including 2 neurons in the hidden layer. The parameters represent the weighting linking the neurons in one layer to the one after that. We use the rectified linear unit activation function so that we don't need to normalise the data and can increase the computational efficiency.

We use the same error function we defined before as:

$\hat{R}_n(h) = \frac{1}{n}\Sigma_n \alpha(h(\mathbf{x}_i) - y_i)^2$

with $\alpha \begin{cases} 3, & \text{if } h(\mathbf{x}_i) < y_i \text{ and } h(\mathbf{x}_i) > T \\ 1, & \text{otherwise} \end{cases}$

Since it is easily differentiable, we have efficient backpropagation and it is adapted to this ML problem by taking the different error coefficients.

2. Consider four data points $\mathbf{x}_1 = (0,-4)$, $\mathbf{x}_2 = (2,0)$, $\mathbf{x}_3 = (0,4)$, $\mathbf{x}_4 = (-2,0)$ and their labels $y_1 = -1$, $y_2 = -1$, $y_3 = 1$,$y_4 = 1$.

a) Derive the solution formula for **w** in a linear regression problem using matrix formulation. [25 marks]

Solution:

In linear regression, we use the squared error $[h(x) - f(x)]^2$

Hence we have the in-sample error

$\hat{R}_n(h) = \frac{1}{n}\Sigma_{i=0}^{n} (h(w, x_i) - y_i)^2$

In linear regression, we define our predictor to be $h(x) = w^T x$ and we can hence write our loss function as:

$\hat{R}_n(w) = \frac{1}{n}\Sigma_{i=0}^{n} (w^T x_i - y_i)^2$

Using matrix form now: $\hat{R}_n(w) = \frac{1}{n} \| Xw - y \|^2$

Now we want to minimise E_{in} so we take the gradient of our expression wrt w and equate it to 0. Because $\| Xw - y \|^2 = (Xw - y)(Xw - y)^T$, we get:

$\nabla \hat{R}_n(w) = \frac{2}{N} X^T(Xw - y) = 0$

$X^T X w = X^T y$

$w = (X^T X)^{-1} X^T y = \dot{X} y$ with $\dot{X}$ being the pseudo inverse of X.

b) Input the data points $\mathbf{x}_{1,\ldots,4}$ with their labels $y_{1,\ldots,4}$ in the formula resulting from question (a) and calculate the values of **w** for this regression problem. [20 marks]

Solution:

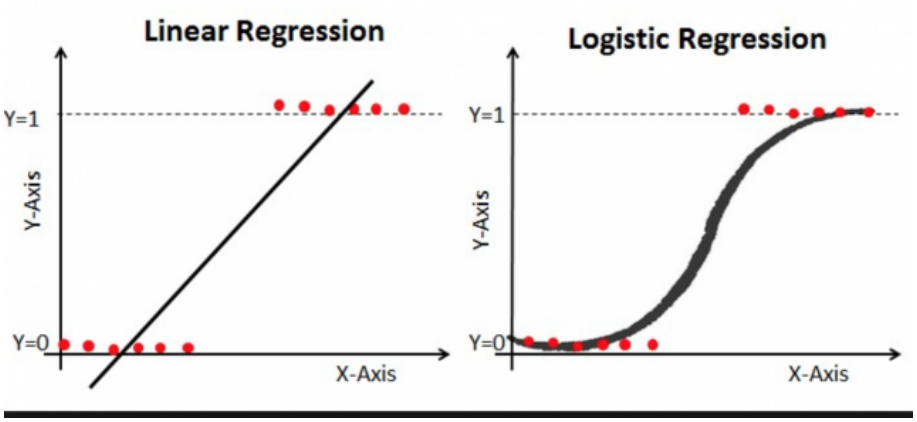
$$\text{X} = \begin{bmatrix} 1 & 0 & -4 \\ 1 & 2 & 0 \\ 1 & 0 & 4 \\ 1 & -2 & 0 \end{bmatrix} \quad X^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 0 & -2 \\ -4 & 0 & 4 & 0 \end{bmatrix} \quad \text{y} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

$$w = (X^T X)^{-1} X^T y = \begin{bmatrix} 0 \\ -1/2 \\ -1/4 \end{bmatrix}$$

c) When can the method for solving linear regression be used to find a solution for a binary classification? Give an example and discuss the benefits and risks. [15 marks]

Solution:

Linear regression is a simple to use model that can be applied in binary classification in some cases but should not. In an example where the data is well balanced, spread out and there is a distinctive separation between data labels, using linear regression would not cause too much trouble. However, using logistic regression is more suitable and desirable



As we can see, with logistic regression we soften the threshold in order to get closer to a binary classifier. In linear regression, points far away from boundary have a high influence on the model and hence increasing the error rate of misclassifying for points closer to the boundary. Linear regression has a closed form solution and is pretty easy to implement whereas logistic regression uses gradient descent. This process being iterative can become computationally expensive depending on the complexity

d) Using the four data points $\mathbf{x}_{1,\ldots,4}$ with their labels $y_{1,\ldots,4}$ write down the solution for ridge regression with $\lambda = 0.1$, and calculate values $\mathbf{w}_r$. [20 marks]

Solution:

The solution for ridge regression is given by: $w_{reg} = (Z^T Z + \lambda I)^{-1} Z^T y$.

In our case, since we haven't done a non-linear transformation into a higher order z-space, we have $w_r = (X^T X + \lambda I)^{-1} X^T y$

Reusing our matrices we get: $w_r = (X^T X + 0.1I)^{-1} X^T y$

$$w_r = \begin{bmatrix} 0 \\ -0.49382 \\ -0.24922 \end{bmatrix}$$

e) Calculate the predictions of a linear regressor with parameters $\mathbf{w} = (0, -1, 0.25)$

for the four data points $\mathbf{x}_{1,\ldots,4}$ and the error of this predictor. [20 marks]

Solution:

The predictor of a linear progessor is defined as h(w,x)=Xw.

$$\text{h(w,x)=} \begin{bmatrix} 1 & 0 & -4 \\ 1 & 2 & 0 \\ 1 & 0 & 4 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 0.25 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ 1 \\ 2 \end{bmatrix}$$

We know from before that $\hat{R}_n(w) = \frac{1}{n} \parallel Xw - y \parallel^2$

$\hat{R}_n(w) = \frac{1}{n} \parallel h(w,x) - y \parallel^2 = 0.5$

Mean squared error of 0.5.

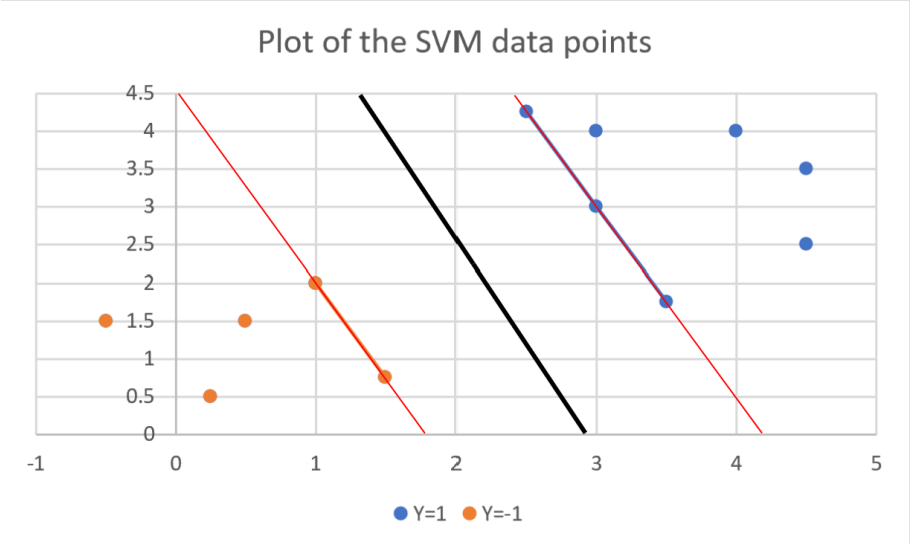
3. a) Support Vector Machines

You are given the following training data examples that belong to two classes:

$\mathbf{x}_i$	x_{i1}	x_{i2}	y_i
$\mathbf{x}_1$	2.5	4.25	+1
$\mathbf{x}_2$	3	3	+1
$\mathbf{x}_3$	3	4	+1
$\mathbf{x}_4$	3.5	1.75	+1
$\mathbf{x}_5$	4	4	+1
$\mathbf{x}_6$	4.5	2.5	+1
$\mathbf{x}_7$	4.5	3.5	+1
$\mathbf{x}_8$	-0.5	1.5	-1
$\mathbf{x}_9$	0.25	0.5	-1
$\mathbf{x}_{10}$	0.5	1.5	-1
$\mathbf{x}_{11}$	1	2	-1
$\mathbf{x}_{12}$	1.5	0.75	-1

i) Plot these points, and draw the maximum margin SVM classifier. Which of the training examples constitute the support vectors? [20 marks]

Solution:



Positive examples are illustrated with blue circles, while the negative examples are represented with orange circles. The maximum margin SVM classifier is the black line, which has five support vectors.

The support vectors are: $\begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}1.5\\0.75\end{bmatrix}, \begin{bmatrix}2.5\\4.25\end{bmatrix}, \begin{bmatrix}3\\3\end{bmatrix}, \begin{bmatrix}3.5\\1.75\end{bmatrix}$.

This line is equidistant from three positive and two negative support

vectors. Changing the orientation of the line will reduce the distance to one of these support vectors, which reduces the margin. Therefore, we can claim that this is the maximum margin separating hyperplane.

ii) The hyperplane $H(w,b)$ for this SVM classifier is defined as $w^T x + b = 0$, where $\|w\|^2 = 1$. Characterize the w and b parameters for the above training data. [20 marks]

Solution:

This line is parametrized by -2.5x_1-x_2+7.5=0 (formula from excel sheet).

We can set w= $\begin{bmatrix} \frac{-2.5\times\sqrt{4}}{\sqrt{29}} \\ \frac{-2\times\sqrt{4}}{\sqrt{29}} \end{bmatrix}$, and b = $+\frac{7.5\times\sqrt{4}}{\sqrt{29}}$

Hence, we get w=$\frac{-1}{\sqrt{29}}\begin{bmatrix} 5 \\ 1 \end{bmatrix}$, and b = $+\frac{15}{\sqrt{29}}$

The SVM classifier will be given by:

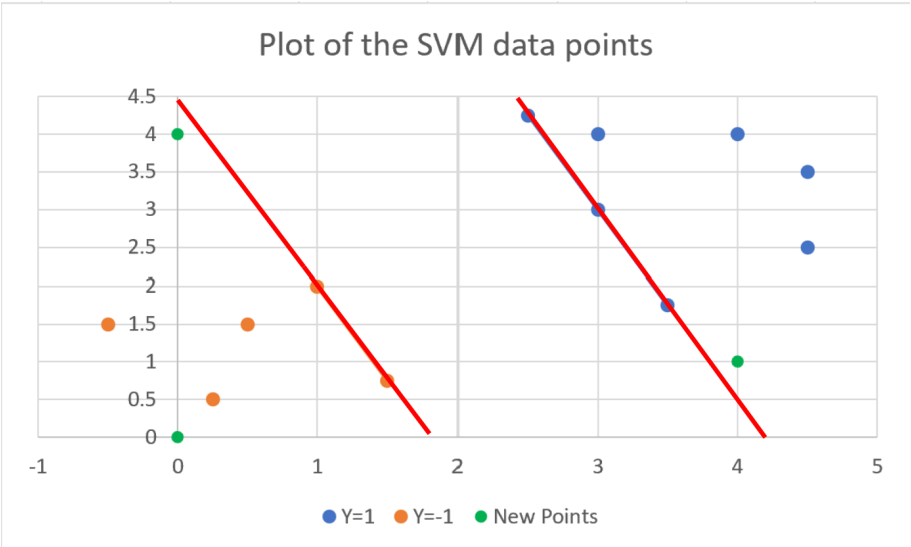
$\hat{y} = sign(\begin{bmatrix} \frac{-5}{\sqrt{29}} & \frac{-1}{\sqrt{29}} \end{bmatrix} \begin{bmatrix} x_1 x_2 \end{bmatrix}^T + \frac{15}{\sqrt{29}})$.

iii) Suppose that you are given the following additional training examples: $\mathbf{x}_{13} = (0,0)$, $y_{13} = -1$, $\mathbf{x}_{14} = (0,4)$, $y_{14} = -1$, and $\mathbf{x}_{15} = (4,1)$, $y_{15} = +1$.

What are the w and b parameters of the SVM classifier now?

[5 marks]

Solution:



The new points are outside the margin and hence they do not change the support vectors. The classifier remains the same.

b) Kernel Method.

i) Consider the mapping $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ that maps a sample $\mathbf{x} = (x_1, x_2)^T$ to a 3-dimensional feature vector as follows:

$$\Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^T.$$

Show that the inner product in the feature space, $\Phi(\mathbf{x})^T\Phi(\mathbf{y})$, can be computed without evaluating the feature vectors explicitly.

[15 marks]

Solution:

Consider the following Kernel:

$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T, \mathbf{y})^2$

$K(\mathbf{x}, \mathbf{y}) = (x_1y_1 + x_2y_2)^2$

$K(\mathbf{x}, \mathbf{y}) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2$

This Kernel corresponds to the inner product between $\Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$ and $\Phi(\mathbf{y}) = (y_1^2, y_2^2, \sqrt{2}y_1y_2)$. As we wanted, we found the inner product without evaluating the feature vectors.

ii) Consider you are given a kernel $K(\cdot, \cdot)$, where the feature map $\Phi : \mathbb{R}^m \to \mathbb{R}^n$ satisfies $\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = K(\mathbf{x}, \mathbf{y})$, $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^m$. Explain how you can calculate the Euclidean distance between $\Phi(\mathbf{x})$ and $\Phi(\mathbf{y})$ in the n-dimensional feature space using the kernel trick:

$$\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| = \sqrt{\sum_{i=1}^{n} (\Phi(\mathbf{x})_i - \Phi(\mathbf{y})_i)^2}$$

[20 marks]

Solution:

Let's start with the definition of the squared induced norm:

$\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\|^2 = (\Phi(\mathbf{x}) - \Phi(\mathbf{y}))^T(\Phi(\mathbf{x}) - \Phi(\mathbf{y}))$

$\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\|^2 = \Phi(\mathbf{x})^T\Phi(\mathbf{x}) + \Phi(\mathbf{y})^T\Phi(\mathbf{y}) - \Phi(\mathbf{x})^T\Phi(\mathbf{y}) - \Phi(\mathbf{y})^T\Phi(\mathbf{x})$

Since the values in $\mathbf{x}$ and $\mathbf{y}$ are real, we know that $\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = \Phi(\mathbf{y})^T\Phi(\mathbf{x})$, hence:

$$\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\|^2 = \Phi(\mathbf{x})^T\Phi(\mathbf{x}) + \Phi(\mathbf{y})^T\Phi(\mathbf{y}) - 2(\Phi(\mathbf{x})^T\Phi(\mathbf{y}))$$

We know replace the appropriate terms with the Kernel and get:

$$\|\Phi(\mathbf{x}) - \Phi(\mathbf{y})\| = \sqrt{K(\mathbf{x},\mathbf{x}) + K(\mathbf{y},\mathbf{y}) - 2K(\mathbf{x},\mathbf{y})}$$

which is what we wanted.

iii) You are given the following five points in $\mathbb{R}^2$:

$$\mathbf{x}_1 = \begin{pmatrix}1.2\\2.2\end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix}3.3\\0.2\end{pmatrix}, \mathbf{x}_3 = \begin{pmatrix}4.0\\1.7\end{pmatrix}, \mathbf{x}_4 = \begin{pmatrix}3.9\\4.1\end{pmatrix}, \mathbf{x}_5 = \begin{pmatrix}5.1\\0.2\end{pmatrix}$$

Considering the feature map $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ given in i) above, find the two points with the highest Euclidean distance from each other in the feature space. Are the relative distances between the points preserved with the mapping Φ, i.e., are the furthest points in $\mathbb{R}^2$ the same as the furthest points in $\mathbb{R}^3$?

[20 marks]

Solution:

First let's compute the transformed data into the new space.

$$\Phi(\mathbf{x}_1) = \begin{pmatrix}1.44\\4.82\\3.73\end{pmatrix}, \Phi(\mathbf{x}_2) = \begin{pmatrix}10.89\\0.04\\0.93\end{pmatrix}, \Phi(\mathbf{x}_3) = \begin{pmatrix}16\\2.89\\9.62\end{pmatrix},$$

$$\Phi(\mathbf{x}_4) = \begin{pmatrix}15.21\\16.81\\22.61\end{pmatrix}, \Phi(\mathbf{x}_5) = \begin{pmatrix}26.01\\0.04\\1.44\end{pmatrix}$$

In the original space, we can see that the two points with the highest Euclidean distance from each other are $\mathbf{x_1}$ and $\mathbf{x_5}$. Now, in the mapped space, $\Phi(\mathbf{x_4})$ and $\Phi(\mathbf{x_5})$ are the points furthest away from each other, hence the mapping Φ does not preserve the relative distances between the points.

4. a) KNN Regression.

In this question you will use the country GDP dataset you are provided on Blackboard. Our goal is to estimate the GDPs of the countries using KNN regression. To decide on the K value, we will use the validation dataset.

i) Plot the average training error of KNN regression algorithm using Euclidean distance for the values of $K = 1, \ldots, 9$. [20 marks]

Solution: To build a KNN regression algorithm and plot data, I used Python 3.6 under the Jupyter notebook with the following libraries: Matplotlib, Numpy, Pandas, Seaborn, Scikit Learn. I will go through the main aspects of the code before presenting the graphs.

First, we import the necessary libraries and the data set. We define the first column (country) as an index to separate it from the numeric data.

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
# Use pandas .read_csv() method to read in classified dataset
# index_col -> argument assigns the index to a particular column
df = pd.read_csv('dataset_GDP.csv')
# Use the .head() method to display the first few rows
df = df.set_index("Country")
df = df.apply(pd.to_numeric)
df.head()
```

Country	Area (thousand km2)	Population (millions)	GDP (billion $)
US	9148	331.0	21374
China	9326	1439.3	14343
Australia	7634	25.5	1393
UK	242	67.9	2827
Germany	349	83.8	3846

We then normalise all the numeric data so that the larger values do not influence the estimate over other values. We exclude the last column, the GDP one, from that scaler as this data will be used in ”y” the output.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('GDP (billion $)', axis=1))
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
scaled_features = scaler.transform(df.drop('GDP (billion $)',axis=1))
scaled_features
```

The next step is to separate the training data from the testing one and the input from the output. We take the first 24 rows for training and we use the rest (7 rows) for validation purposes. We also keep the

GDP data for our output ”y” to compare it to our prediction later.

```
df_feat = pd.DataFrame(scaled_features, columns = df.columns[:-1])
df_feat.head()
```

	Area (thousand km2)	Population (millions)
0	2.646429	0.561955
1	2.709062	3.817874
2	2.113699	-0.335530
3	-0.487316	-0.210969
4	-0.449666	-0.164259

```
X_train = scaled_features[0:24][:]
X_test = scaled_features[25:][:]
y_train = df.iloc[0:24]["GDP (billion $)"]
y_test = df.iloc[25:]["GDP (billion $)"]
```

Lastly, we create a loop function to train the data for different values of K and append to the training error the euclidean distance between our training prediction and the actual training data that we kept stored in y before. (Note: there is a missing term at the end of the train error on this screenshot and it says 14 instead of 10 in the first for loop)

```
from sklearn.neighbors import KNeighborsRegressor
from scipy.spatial import distance
```

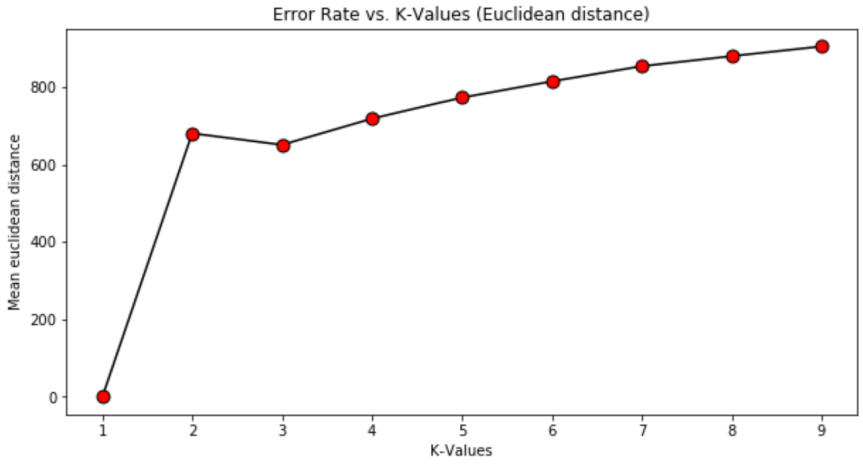
```
error = []
train_error = []

for i in range(1,14):

    knn = KNeighborsRegressor(n_neighbors = i, metric ="euclidean")
    knn.fit(X_train, y_train)
    pred_test = knn.predict(X_test)
    pred_train = knn.predict(X_train)
    error.append(distance.euclidean(pred_test, y_test))
    train_error.append(distance.euclidean(pred_train, y_train))
```

```
plt.figure(figsize=(10,5))
plt.plot(range(1,10), train_error, color='black', marker='o', markerfacecolor='red', markersize=9)
plt.title('Error Rate vs. K-Values in training set (Euclidean distance)')
plt.xlabel('K-Values')
plt.ylabel('Euclidean distance')
```

We plot the results and this is the graph we obtain:



ii) Repeat the same using Manhattan distance. [10 marks]

Solution:

For the Manhattan distance, we simply replace the distance metric from euclidean to Manhattan and repeat the process. We obtain the following graph:

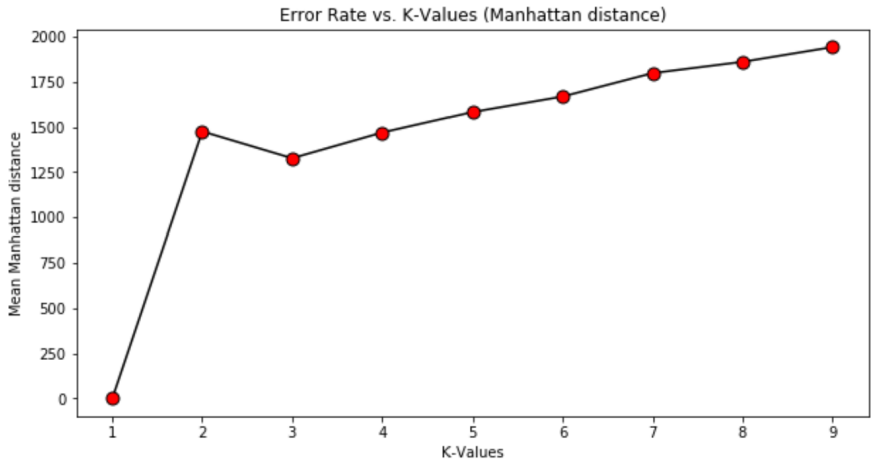
```
from sklearn.neighbors import KNeighborsRegressor
from scipy.spatial import distance
```

```
error = []
train_error = []

for i in range(1,10):

    knn = KNeighborsRegressor(n_neighbors = i, metric ="euclidean")
    knn.fit(X_train, y_train)
    pred_test = knn.predict(X_test)
    pred_train = knn.predict(X_train)
    error.append(distance.cityblock(pred_test, y_test))
    train_error.append(distance.cityblock(pred_train, y_train))
```

```
plt.figure(figsize=(10,5))
plt.plot(range(1,10), train_error, color='black', marker='o', markerfacecolor='red', markersize=9)
plt.title('Error Rate vs. K-Values in training set (Euclidean distance)')
plt.xlabel('K-Values')
plt.ylabel('Euclidean distance')
```



iii) Plot the training error as a function of K with Euclidean distance. [10 marks]

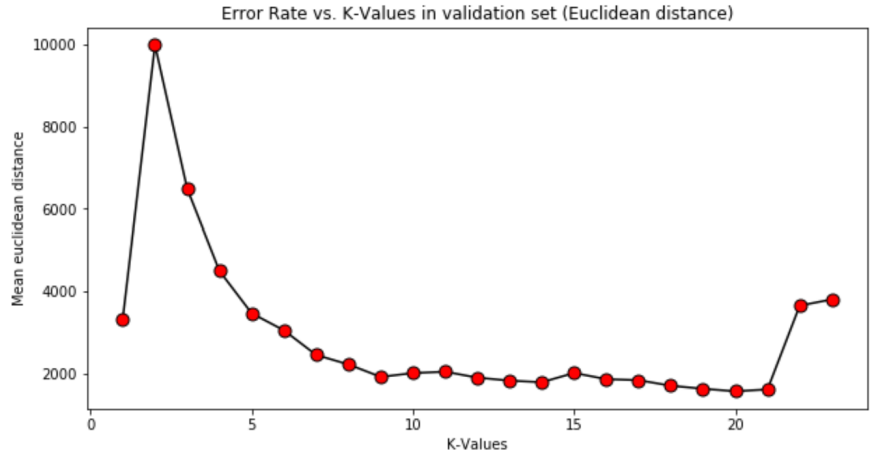
Solution:

For the training error as a function of K, we simply expand the value of K to its maximum value (which is 24 the number of entries in the training set) and do not divide the error by the size of the set. Hence, we obtain the following graph:

iv) Plot the validation error as a function of K with Euclidean distance. [10 marks]

Solution:

Now for the validation error, we simply change the error shown on the graph from the training one to the test or validation one we calculated earlier, which compares our prediction for the testing data to the actual values. We get this graph:



v) Comment on the above results, and decide which K value should be used. [20 marks]

Solution:

Here, we are facing a variance- bias trade off scenario. Looking at the error values in the training set, we would be tempted to choose K=1

since we are doing extremely good (zero error). However, that simply means all the data points have been put in one cluster all together and the training data has been memorized so we can't make a mistake. In that case, the error on the test data is huge. On the other hand, if we look a the error on the test data alone, we would be tempted to use a big K as this gives use the smallest error. Once again, this is the equivalent of using the data individually and the out of sample error will be very big since we only know how to deal with the already had. Instead, we want to choose a compromise between the two and K=9 seems to be a good one. It seems like it is the smallest value we can use before the curves "flatten" and hence represents the ideal value for K.

b) Clustering.

Consider using k-means algorithm to cluster a dataset $\mathscr{S}$. At each iteration of the algorithm we update the cluster centers $\mathscr{C} = \{\mu_1, \ldots, \mu_k\}$ iteratively. In the class, we used random initialization for the first step of the algorithm. Instead, we use the following method:

i) Choose μ_1 uniformly randomly from $\mathscr{S}$. We have $\mathscr{C} = \{\mu_1\}$.

ii) Choose the next center as the point in the dataset that is furthest away from all the cluster centers chosen so far, i.e., $\mu_i = \text{argmax}_{s \in \mathscr{S}} d(s)$, where $d(s)$ for point $s \in \mathscr{S}$ is defined as the minimum distance of s to the cluster heads chosen so far, $d(s) \triangleq \min_{\mu \in \mathscr{C}} \|s - \mu\|^2$. We have $\mathscr{C} = \mathscr{C} \cup \{\mu_i\}$.

iii) Repeat step 2 until all k cluster centers are chosen.

Prove that, with this initialization method the k-means algorithm may converge to a cluster that has an arbitrarily larger cost than the optimal one. (We use $\sum_{s \in \mathscr{S}} d(s)$ as the objective cost function.)

[30 marks]

Solution: type your solution here