Department of Electrical Engineering
School of Science and Engineering

EE514/CS535 Machine Learning

HOMEWORK 1 – SOLUTIONS

Due Date: 23:55, Tuesday, March 02, 2021 (Submit online on LMS)
Format: 7 problems, for a total of 100 marks
Contribution to Final Percentage: 2.5%
Instructions:

- Each student must submit his/her own hand-written assignment, scanned in a single PDF document.
- You are allowed to collaborate with your peers but copying your colleague's solution is strictly prohibited. Anybody found guilty would be subjected to disciplinary action in accordance with the university rules and regulations.
- Note: Vectors are written in lowercase and bold in the homework, for your written submission kindly use an underline instead. In addition, use capital letters for matrices and lowercase for scalars.

Problem 1 (20 marks)

(**Note:** Compile and submit screenshots of your plots for this question.)

Polynomial Regression - Polynomial regression is a form of regression analysis in which the relationship between the independent variable and the dependent variable is modeled as an n-th degree polynomial. The model equation that relates the input to the output is of the form:

$$y_i(x_i) = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \cdots + \theta_M x_i^M.$$

(a) [**4 marks**] Assuming we use an M^{th} degree polynomial for our regression analysis and we have N outputs $y_1, y_2, \ldots, y_N$, express the regression model in the matrix form $\boldsymbol{y} = A\boldsymbol{\theta}$.

(b) [**8 marks**] We will now implement the regression model on one feature variable using programming tools. Use the following code to generate some toy data:

```
import pandas as pd
xdic={'x': {11: 300, 12: 170, 13: 288, 14: 360, 15: 319, 16: 330,
17: 520, 18: 345, 19: 399, 20: 479}}
ydic={'y': {11: 305, 12: 270, 13: 360, 14: 370, 15: 379, 16: 400,
```

```
17: 407, 18: 450, 19: 450, 20: 485}}
x=pd.DataFrame.from_dict(xdic)
y=pd.DataFrame.from_dict(ydic)
import numpy as np
x_seq = np.linspace(x.min(),x.max(),300).reshape(-1,1)
```

Next, use the scikit-learn library to implement polynomial regression of degree 2 on the data. Finally, use the following code to display the data points and the regression curve:

```
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(x,y)
plt.plot(x_seq,polyreg.predict(x_seq),color="black")
plt.title("Polynomial regression with degree "+str(degree))
plt.show()
```

(c) [**8 marks**] Repeat the regression process for degrees 1 (i.e., linear regression), 3, 4, and 5. What problem do you observe as the degree of the polynomial increases? (Hint: think about the trained model's accuracy when implemented on a test dataset.)

Solution:

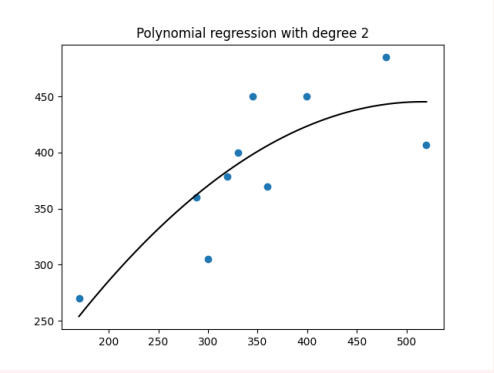
(a)

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^M \\ 1 & x_2 & x_2^2 & \dots & x_2^M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^M \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_M \end{bmatrix}$$

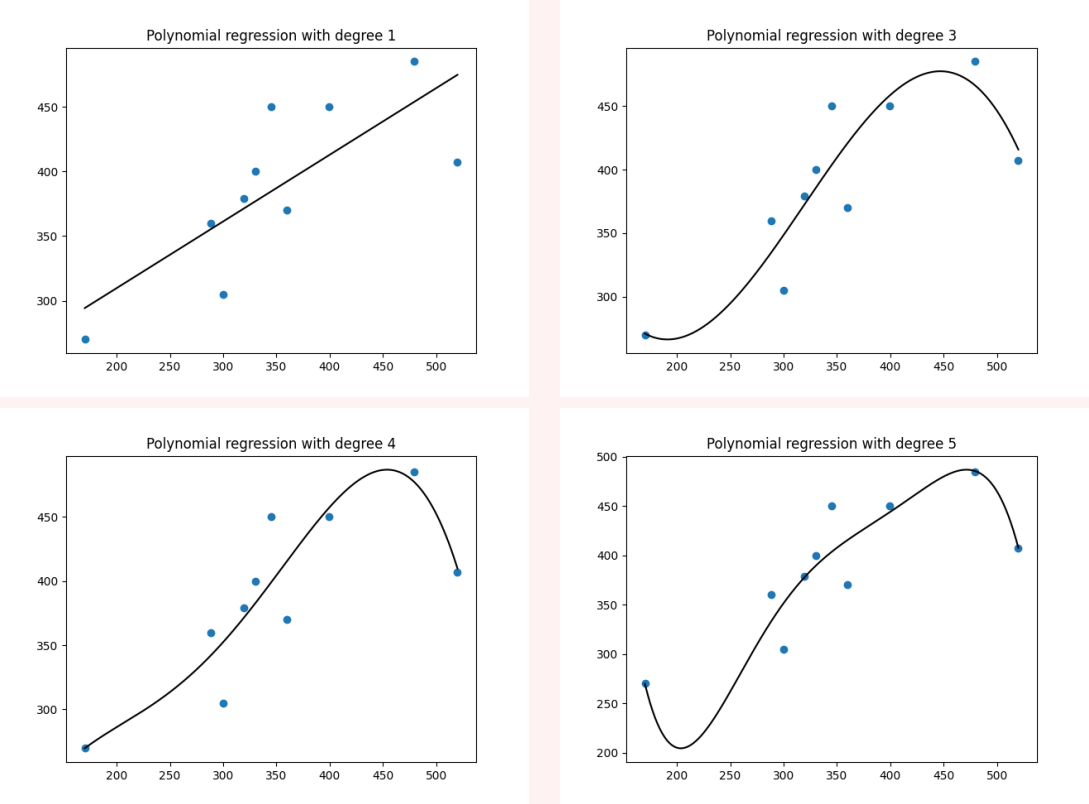
(b) Code to implement polynomial regression:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
degree = 2
polyreg=make_pipeline(PolynomialFeatures(degree),LinearRegression())
polyreg.fit(x,y)
```

Plot:



(c) Plots:



As the degree of the polynomial increases, the regression curve overfits on our training data, which is likely to give a bad accuracy on a test dataset from the same distribution.

Problem 2 (10 marks)

Least Squares Formulation - In this question we will derive the least squares solution from the perspective of an optimization problem.

(a) **[2 marks]** First let us formulate the problem in terms of an optimization problem. Consider the following system: The correct output is given by $\boldsymbol{y} \in \mathbb{R}^n$. The measured output $\hat{\boldsymbol{y}} \in \mathbb{R}^n$ for a total of m instances is given by $A = [\hat{\boldsymbol{y}}_1, \hat{\boldsymbol{y}}_2, \ldots, \hat{\boldsymbol{y}}_m] \in \mathbb{R}^{n \times m}$ (where $n > m$). Using these m instances, we want to determine the best approximation of $\boldsymbol{y}$ by taking a weighted sum of these m measured outputs, where the weights are given by the vector $\boldsymbol{\theta} \in \mathbb{R}^m$.

The weight vector $\boldsymbol{\theta} \in \mathbb{R}^m$ minimizes the mean-square error between the correct and measured output. We can define this as an optimization problem as shown:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{minimize}} \quad f(\boldsymbol{\theta}),$$

where $f(\boldsymbol{\theta})$ is the cost function which we aim to minimize. For this problem, write down the expression for $f(\boldsymbol{\theta})$.

(b) **[8 marks]** Find the expression for $\boldsymbol{\theta}^*$ that minimises the cost function $f(\boldsymbol{\theta})$ by computing its gradient.
Note: You might find the following relationships useful for this proof:

$$\|\boldsymbol{a}\|_2^2 = \boldsymbol{a}^T \boldsymbol{a}$$
$$(AB)^T = B^T A^T$$

You will also need to review derivatives of matrices and vectors.

This final expression is also known as the **Left Pseudo-Inverse**, verify your answer by looking up the formula.

Solution:

(a) $f(\boldsymbol{\theta}) = \|\boldsymbol{y} - A\boldsymbol{\theta}\|_2^2$

(b)

$$\begin{aligned} f(\boldsymbol{\theta}) &= \|\boldsymbol{y} - A\boldsymbol{\theta}\|_2^2 \\ &= (\boldsymbol{y} - A\boldsymbol{\theta})^T(\boldsymbol{y} - A\boldsymbol{\theta}) \\ &= (\boldsymbol{y}^T - \boldsymbol{\theta}^T A^T)(\boldsymbol{y} - A\boldsymbol{\theta}) \\ &= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T A\boldsymbol{\theta} - \boldsymbol{\theta}^T A^T\boldsymbol{y} + \boldsymbol{\theta}^T A^T A\boldsymbol{\theta} \end{aligned}$$

$$\begin{aligned} \frac{d}{d\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{d}{d\boldsymbol{\theta}} \left(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T A\boldsymbol{\theta} - \boldsymbol{\theta}^T A^T\boldsymbol{y} + \boldsymbol{\theta}^T A^T A\boldsymbol{\theta}\right) &= 0 \\ -A^T\boldsymbol{y} - A^T\boldsymbol{y} + 2A^T A\boldsymbol{\theta} &= 0 \\ A^T A\boldsymbol{\theta} &= A^T\boldsymbol{y} \\ \boldsymbol{\theta}^* &= (A^T A)^{-1}A^T\boldsymbol{y} \end{aligned}$$

Problem 3 (15 marks)

Multivariate Gradient Descent - You are presented with the following feature vector X, the parameter vector θ, and the gold label y. The initial values of the parameter vector are also given, and assume x_0 to be the bias absorbed into the feature vector.

$$\boldsymbol{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 12 \\ 7 \\ 5 \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 2.0 \\ -0.2 \\ 0.3 \\ 0.7 \end{bmatrix}, \quad \boldsymbol{y} = 7$$

Perform 3 iterations of batch gradient descent on the dataset. Assume learning rate $\alpha = 0.001$ and round your answers to 4 decimal places. Show all your steps and the mathematical equations that you use.

Solution: We use the following loss function for our calculations:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}||(\boldsymbol{y} - \boldsymbol{X\theta})||_2^2$$

Then we simultaneously update our parameters using the following equation:

$$\theta_i = \theta_i - \alpha\frac{\partial\mathcal{L}}{\partial\theta_i}$$

The parameters vector in the 3 iterations is then as follows:

$$\boldsymbol{\theta_1} = \begin{bmatrix} 0.6992 \\ 1.9984 \\ -0.2097 \\ 0.2943 \\ 0.6960 \end{bmatrix}, \quad \boldsymbol{\theta_2} = \begin{bmatrix} 0.6987 \\ 1.9974 \\ -0.2159 \\ 0.2907 \\ 0.6934 \end{bmatrix}, \quad \boldsymbol{\theta_3} = \begin{bmatrix} 0.6983 \\ 1.9967 \\ -0.2203 \\ 0.2881 \\ 0.6916 \end{bmatrix}$$

and the cost $\mathcal{L}(\boldsymbol{\theta})$ decreases from 0.8100 to 0.5180 to 0.3655.

Problem 4 (10 marks)

Hamming Distance - For categorical data in k-NN classification, we often use the Hamming distance as our distance metric. In the table provided below, you are provided the binary vectors for a set of colours, and their frequency in the training data:

Colour	Binary Vector				No. in Train Data
	1	2	3	4	
White	0	0	0	1	2
Pink	1	0	0	1	1
Red	1	0	0	0	2
Orange	1	1	0	0	3
Yellow	0	1	0	0	3
Green	0	1	1	0	1
Blue	0	0	1	0	1
Purple	1	0	1	0	2

Table 1: Binary coded vectors for different colours

(a) [**4 marks**] Suppose you are provided an unknown colour, defined by the vector: '1011'. Run the k-NN algorithm with $k = 3$ using Hamming distance to predict the colour. Show all of your working.

(b) [**6 marks**] We can also use Hamming distance to compute the error in the final prediction. For the test data tabulated below: Run the k-NN algorithm for $k = 5$ and

Test Data Point	Binary Vector				No. in Test Data
	1	2	3	4	
X	0	0	1	1	2
Y	0	1	0	1	1
Z	0	1	1	0	2

Table 2: Binary coded vectors for test data

calculate the prediction error using Hamming distance.
Hint: Make sure to average out the total error across all the test data points!

Solution:

(a) Finding the three nearest neighbours, $k = 3$:
Pink: 1, Purple: 2
Predicted colour: **Purple**

Colour	Binary Vector 1	2	3	4	No. in Train Data	Distance
White	0	0	0	1	2	2
Pink	1	0	0	1	1	1
Red	1	0	0	0	2	2
Orange	1	1	0	0	3	3
Yellow	0	1	0	0	3	4
Green	0	1	1	0	1	3
Blue	0	0	1	0	1	2
Purple	1	0	1	0	2	1

Table 3: Hamming distance between test data point and all colours

(b) (Using the same working as last part)
Finding the five nearest neighbours, $k = 5$:

$$\begin{aligned} &\text{X} : \textbf{White} : 2, \text{Blue} : 1, \text{Pink} : 1, \text{Green} : 1, && \Rightarrow \text{Error} = 1 \times 2 = 2 \\ &\text{Y} : \textbf{Yellow} : 3, \text{White} : 2, && \Rightarrow \text{Error} = 1 \times 1 = 1 \\ &\text{Z} : \textbf{Yellow} : 3, \text{Green} : 1, \text{Blue} : 1 && \Rightarrow \text{Error} = 1 \times 2 = 2 \end{aligned}$$

(*Also award marks if X predicted as purple, or any valid answer for any other test point*)
Note: Error is given by Hamming distance between actual binary vector and predicted binary vector, multiplied by the number of the same test points.
We can find the prediction error by summing these errors and dividing by the total number of test points.
Prediction Error $= \frac{5}{5} = 1$

Problem 5 (15 marks)

(**Note:** Compile and submit screenshots of your plots for this question.)

Curse of Dimensionality - In this question we will look closely at the problem associated with higher dimensions in hypercubes when using k-NN. During your lectures, you learnt that as the dimensions increase, the neighbours in a constant vicinity decrease. Let us now visualise this phenomenon through simulations.
The expression for average distance in a unit cube $[0, 1]^3$ is given by:

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} dx_1 dx_2 dy_1 dy_2 dz_1 dz_2$$

Solving this expression is really tedious, and as dimensions increase, it becomes harder to solve and visualise as well. Instead, we will numerically calculate average distance by plotting a frequency histogram of distances.

(a) [**10 marks**] You are required to build a function that will randomly take two points from a $n-$dimensional hypercube and calculate the Euclidean distance for $i = 10000$ iterations. You are given the following starter code:

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

```
import math
import random

def freq_plot(n_dim, iterations = 10000):
dist = []
# code required here

plt.figure()
plt.hist(dist, range = [0, math.sqrt(n_dim)], bins=100,
density = True)
title = 'n = ' +  str(n_dim)
plt.gca().set(title=title, xlabel='Distances', ylabel='Frequency')
return
```

Use the **random.random()** function for generating a float between 0 and 1.
Attach a screenshot of your final working function code.

(b) [**5 marks**] Run the function in part (b) for n_dim $= 2, 3, 5, 10, 100, 1000$, and attach screenshots of your plots.

Solution:

(a)
```
import matplotlib.pyplot as plt
%matplotlib inline
import math
import random
import numpy as np

def freq_plot(n_dim, iterations = 10000):
dist = []
for i in range(0, iterations):
    a_vec = []
    b_vec = []
    for n in range(0, n_dim):
        a_vec.append(random.random())
        b_vec.append(random.random())
    dist.append(np.linalg.norm(np.array(a_vec)-np.array(b_vec)))

plt.figure()
plt.hist(dist, range = [0, math.sqrt(n_dim)], bins=100,
density = True)
title = 'n = ' +  str(n_dim)
plt.gca().set(title=title, xlabel='Distances', ylabel='Frequency')
return
```

(b) Plots given in Figure 1

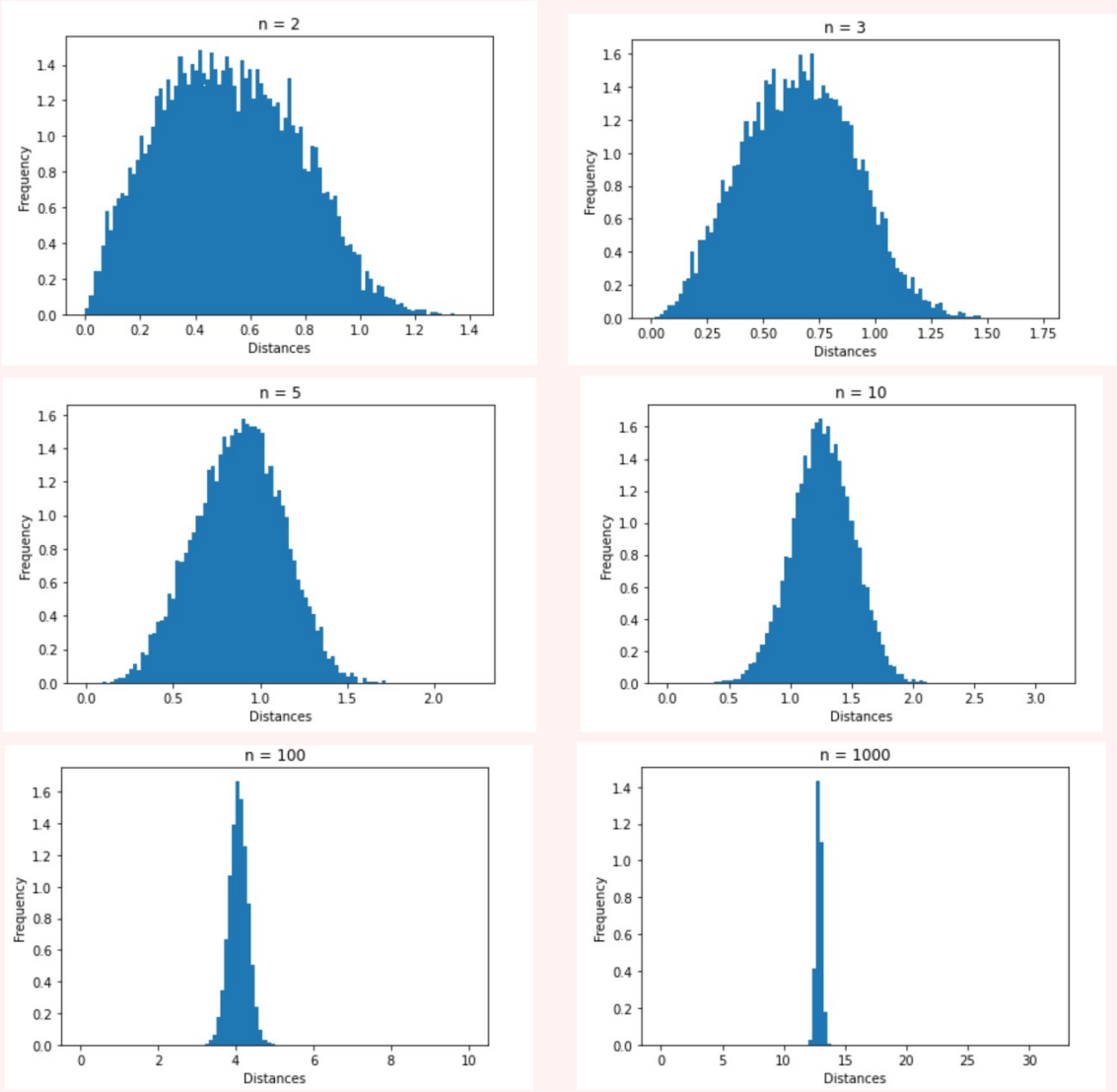


Figure 1: Average distance in a unit hypercube for different dimensions

Problem 6 (15 marks)

KD Trees - One of the most commonly used k-NN algorithms, KD trees are space-partitioning data structures that divide a feature set based on some particular attributes, thereby reducing the time complexity of the k-NN algorithm.

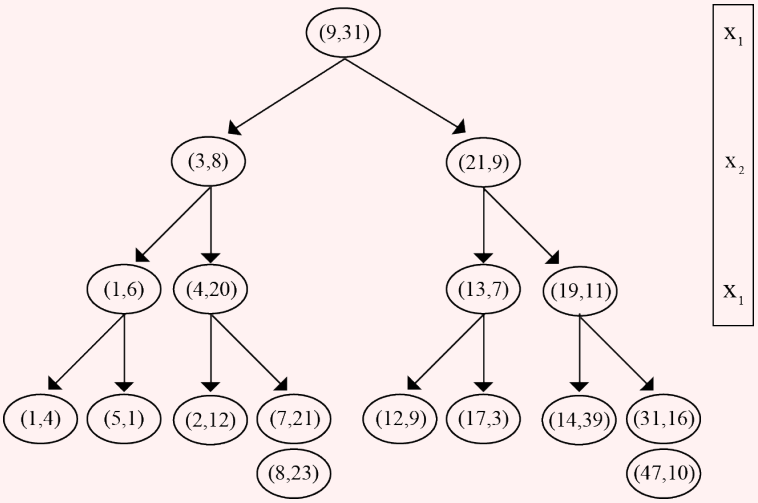
(a) [**10 marks**] The following table contains data points $\mathbf{x} = (x_1, x_2)$ from a distribution $\in \mathbb{R}^2$. Use these data points to create a KD tree. Begin your solution by splitting using the x_1 feature, and then alternate between the 2 features at each level of the tree when determining the next split. Each child node should have 2-3 data points.

x_1	1	3	12	7	8	13	2	21	5	17	31	4	47	19	1	9	14
x_2	6	8	9	21	23	7	12	9	1	3	16	20	10	11	4	31	39

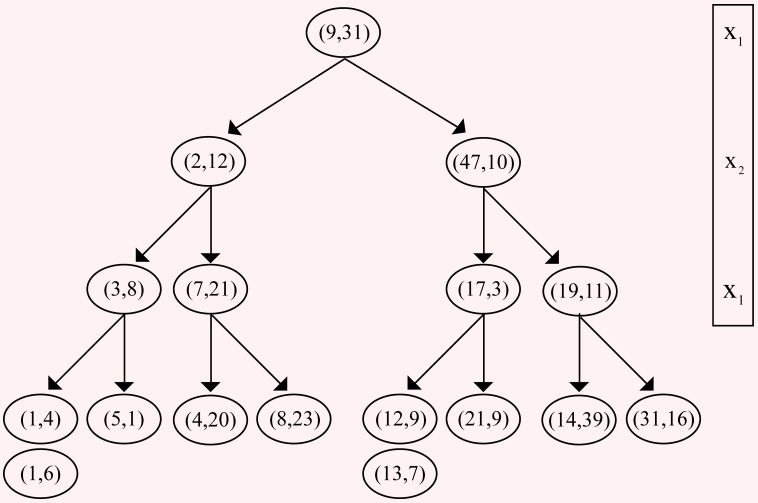
(b) [**5 marks**] What major issue can you observe when using KD trees to classify a test data point?

Solution:

(a) .



The following has also been accepted for this question:



(b) While using KD trees to classify a data point is faster than the conventional k-NN algorithm, it is likely that the closest neighbour is in another branch of the KD tree, hence the likelihood of misclassification increases.

Problem 7 (15 marks)

Principal Component Analysis - In class we studied PCA as a means to reduce dimensionality, by minimizing the squared error of predictions in lower dimension, also known as projections onto a range space, as compared to the actual values. In this question, we will prove that this problem is equivalent to maximizing the squared length of these projections, i.e., maximizing variance of projected data.

Let us define the original problem from the class lecture. Considering n feature vectors of the form $\boldsymbol{x} \in \mathbb{R}^d$. Using only k basis vectors, we want to project $\boldsymbol{x}$ in a new space of lower dimensionality, from $\mathbb{R}^d$ to $\mathbb{R}^k$:

$$\boldsymbol{z} = W^T \boldsymbol{x},$$

where W is the orthogonal mapping matrix of size $d \times k$. To obtain a approximation of $\boldsymbol{x}$ we use the following reconstruction:

$$\hat{\boldsymbol{x}} = W\boldsymbol{z},$$

which is equivalent to:

$$\hat{\boldsymbol{x}} = WW^T\boldsymbol{x}$$
$$\hat{\boldsymbol{x}} = P\boldsymbol{x},$$

where $P = WW^T$ is the projection operator, that is idempotent: has the following properties: $P^2 = P = P^T$.
Our objective is to minimize the sum of squared error during reconstruction, we can write it as follows:

$$\text{minimize} \quad \|\boldsymbol{x} - P\boldsymbol{x}\|_2^2$$

(a) **[10 marks]** Show that this expression can be reduced to:

$$\text{minimize} \quad [\|\boldsymbol{x}\|_2^2 - \|P\boldsymbol{x}\|_2^2]$$

Note: You might find the following relationships useful for this proof:

$$\|\boldsymbol{a}\|_2^2 = \boldsymbol{a}^T\boldsymbol{a}$$
$$(AB)^T = B^T A^T$$

(b) **[5 marks]** Using the final expression in part (a) and the Pythagorean Theorem on $\|\boldsymbol{x}\|_2^2$ to show that this problem is equivalent to maximizing the variance of projected data. You may draw a diagram to augment your point.

Solution:

(a)

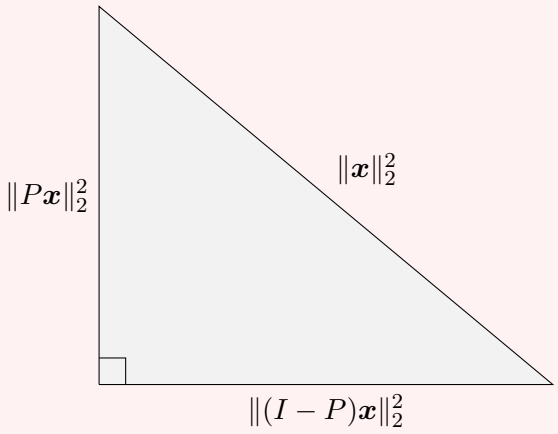
$$\begin{aligned}
\|\boldsymbol{x} - P\boldsymbol{x}\|_2^2 &= (\boldsymbol{x} - P\boldsymbol{x})^T(\boldsymbol{x} - P\boldsymbol{x}) \\
&= \boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T P\boldsymbol{x} + \boldsymbol{x}^T P^T P\boldsymbol{x} \\
&= \boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T P^T P\boldsymbol{x} + \boldsymbol{x}^T P^T P\boldsymbol{x} \text{ (since } P = P^T P) \\
&= \boldsymbol{x}^T\boldsymbol{x} - \boldsymbol{x}^T P^T P\boldsymbol{x} \\
&= \|\boldsymbol{x}\|_2^2 - \|P\boldsymbol{x}\|_2^2
\end{aligned}$$

(b) We can use the Pythagorean Theorem on $\|\boldsymbol{x}\|_2^2$ as shown:

$$\|\boldsymbol{x}\|_2^2 = \|P\boldsymbol{x}\|_2^2 + \|(I-P)\boldsymbol{x}\|_2^2,$$

where the first term is the variance of data and is fixed, the second term is the captured variance, which we want to maximize, the third term is the reconstruction error, which we want to minimize.

We can also draw a right triangle and label it as follows:



It is easy to see that minimizing the reconstruction error is the same as maximizing the captured variance of data.

— End of Homework —