# Machine Learning - Homework 2

Pedro Curvo (ist1102716) | Salvador Torpes (ist1102474)

1st Term - 23/24

---

## Pen and Paper Exercises

## Dataset

The following dataset will be used for this homework:

| $D$ | | Input | | | | | Output |
|---|---|---|---|---|---|---|---|
| | | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ |
| Training Observations | $x_1$ | 0.24 | 0.36 | 1 | 1 | 0 | A |
| | $x_2$ | 0.16 | 0.48 | 1 | 0 | 1 | A |
| | $x_3$ | 0.32 | 0.72 | 0 | 1 | 2 | A |
| | $x_4$ | 0.54 | 0.11 | 0 | 0 | 1 | B |
| | $x_5$ | 0.66 | 0.39 | 0 | 0 | 0 | B |
| | $x_6$ | 0.76 | 0.28 | 1 | 0 | 2 | B |
| | $x_7$ | 0.41 | 0.53 | 0 | 1 | 1 | B |
| Testing Observations | $x_8$ | 0.38 | 0.52 | 0 | 1 | 0 | A |
| | $x_9$ | 0.42 | 0.59 | 0 | 1 | 1 | B |

Table 1: Dataset

# 1st Question

## a)

In order to build the Bayesian classifier for this dataset, we need to compute the class conditional distributions of $\{y_1, y_2\}$, $\{y_3, y_4\}$ and $y_5$, which are the groups of independent input variables of our dataset as well as the priors.

**Priors** First of all, we will compute the priors $P(y_6 = A)$ and $P(y_6 = B)$:

$$P(y_6 = A) = \frac{3}{7}$$
$$P(y_6 = B) = \frac{4}{7}$$

**Distribution of $y_1$ and $y_2$**

We are told that $y_1 \times y_2 \in \mathbb{R}$ follows a normal 2D distribution. A multivariate normal distribution of $m$ variables $\vec{x} = \{x_1, x_2, ..., x_m\}$ is defined by its mean vector $\vec{\mu}$ and its covariance matrix $\Sigma$:

$$P(\vec{x}|\vec{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^m|\Sigma|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \cdot \Sigma^{-1} \cdot (\vec{x} - \vec{\mu})\right)$$

In our case, we have $m = 2$, $\vec{x} = \{y_1, y_2\}$ and we need to compute two class conditional distributions $p(\vec{x}|y_6 = A)$ and $p(\vec{x}|y_6 = B)$.

**Distribution of $\{y_1, y_2\}$ given $y_6 = A$**

Considering the training data in table **??** with class $y_6 = A$, we can compute the mean vector $\vec{\mu}$ and the covariance matrix $\Sigma$ as follows:

$$\vec{\mu} = \begin{bmatrix} \mu_{y_1} \\ \mu_{y_2} \end{bmatrix} = \frac{1}{3} \cdot \begin{bmatrix} 0.24 + 0.16 + 0.32 \\ 0.36 + 0.48 + 0.72 \end{bmatrix} = \begin{bmatrix} 0.24 \\ 0.52 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma^2_{y_1} & \sigma_{y_1,y_2} \\ \sigma_{y_1,y_2} & \sigma^2_{y_2} \end{bmatrix} = \frac{1}{3} \cdot \begin{bmatrix} \sum_{i=1}^{3}(y_{1i} - \mu_{y_1})^2 & \sum_{i=1}^{3}(y_{1i} - \mu_{y_1})(y_{2i} - \mu_{y_2}) \\ \sum_{i=1}^{3}(y_{1i} - \mu_{y_1})(y_{2i} - \mu_{y_2}) & \sum_{i=1}^{3}(y_{2i} - \mu_{y_2})^2 \end{bmatrix} = \begin{bmatrix} 0.0043 & 0.0064 \\ 0.0064 & 0.0224 \end{bmatrix}$$

Now we need to compute both $|\Sigma|$ and $\Sigma^{-1}$:

$$|\Sigma| = \det \Sigma = 0.0043 \cdot 0.0224 - 0.0064^2 = 5.4613 \cdot 10^{-5}$$

$$\Sigma^{-1} = \begin{bmatrix} 410.156 & -117.188 \\ -117.188 & 78.125 \end{bmatrix}$$

Therefore, we have the normal distribution of $\{y_1, y_2\}$ given $y_6 = A$:

$$P((y_1, y_2)|y_6 = A) = \frac{1}{\sqrt{(2\pi)^2|\Sigma|}} \exp\left(-\frac{1}{2}\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} - \begin{bmatrix} 0.24 \\ 0.52 \end{bmatrix}\right)^T \cdot \begin{bmatrix} 410.156 & -117.188 \\ -117.188 & 78.125 \end{bmatrix} \cdot \left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} - \begin{bmatrix} 0.24 \\ 0.52 \end{bmatrix}\right)\right)$$

$$= \frac{1}{\sqrt{(2\pi)^2 \cdot 5.4613 \cdot 10^{-5}}} \exp\left(-\frac{1}{2}\begin{bmatrix} y_1 - 0.24 \\ y_2 - 0.52 \end{bmatrix}^T \cdot \begin{bmatrix} 410.156 & -117.188 \\ -117.188 & 78.125 \end{bmatrix} \cdot \begin{bmatrix} y_1 - 0.24 \\ y_2 - 0.52 \end{bmatrix}\right)$$

**Distribution of $\{y_1, y_2\}$ given $y_6 = B$**

Considering the training data in table **??** with class $y_6 = B$, we can compute the mean vector $\vec{\mu}$ and the covariance matrix $\Sigma$ as follows:

$$\vec{\mu} = \begin{bmatrix} \mu_{y_1} \\ \mu_{y_2} \end{bmatrix} = \frac{1}{4} \cdot \begin{bmatrix} 0.54 + 0.66 + 0.76 + 0.41 \\ 0.11 + 0.39 + 0.28 + 0.53 \end{bmatrix} = \begin{bmatrix} 0.5925 \\ 0.3274 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma^2_{y_1} & \sigma_{y_1,y_2} \\ \sigma_{y_1,y_2} & \sigma^2_{y_2} \end{bmatrix} = \frac{1}{4} \cdot \begin{bmatrix} \sum_{i=1}^{4} (y_{1i} - \mu_{y_1})^2 & \sum_{i=1}^{4} (y_{1i} - \mu_{y_1})(y_{2i} - \mu_{y_2}) \\ \sum_{i=1}^{4} (y_{1i} - \mu_{y_1})(y_{2i} - \mu_{y_2}) & \sum_{i=1}^{4} (y_{2i} - \mu_{y_2})^2 \end{bmatrix} = \begin{bmatrix} 0.0171 & -0.0073 \\ -0.0073 & 0.0236 \end{bmatrix}$$

Now we need to compute both $|\Sigma|$ and $\Sigma^{-1}$:

$$|\Sigma| = \det \Sigma = 0.0075 \cdot 0.0075 - (-0.0025)^2 = 3.519 \cdot 10^{-4}$$

$$\Sigma^{-1} = \begin{bmatrix} 67.1101 & 20.7954 \\ 20.7954 & 48.7831 \end{bmatrix}$$

Therefore, we have the normal distribution of $\{y_1, y_2\}$ given $y_6 = B$:

$$P((y_1, y_2)|y_6 = B) = \frac{1}{\sqrt{(2\pi)^2 |\Sigma|}} \exp\left(-\frac{1}{2}\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} - \begin{bmatrix} 0.5925 \\ 0.3274 \end{bmatrix}\right)^T \cdot \begin{bmatrix} 67.1101 & 20.7954 \\ 20.7954 & 48.7831 \end{bmatrix} \cdot \left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} - \begin{bmatrix} 0.5925 \\ 0.3274 \end{bmatrix}\right)\right)$$

$$= \frac{1}{\sqrt{(2\pi)^2 \cdot 3.519 \cdot 10^{-4}}} \exp\left(-\frac{1}{2}\begin{bmatrix} y_1 - 0.5925 \\ y_2 - 0.3274 \end{bmatrix}^T \cdot \begin{bmatrix} 67.1101 & 20.7954 \\ 20.7954 & 48.7831 \end{bmatrix} \cdot \begin{bmatrix} y_1 - 0.5925 \\ y_2 - 0.3274 \end{bmatrix}\right)$$

**Distribution of $y_3$ and $y_4$**

The class conditional distributions of $y_3$ and $y_4$ come directly from the information in table **??** and they are given by:

| $P(y_3 \cap y_4 \| y_6 = A)$ | | $y_3 = 0$ | $y_3 = 1$ |
|---|---|---|---|
| $y_4$ | 0 | $P(y_3 = 0 \cap y_4 = 0 \| y_6 = A) = 0$ | $P(y_3 = 1 \cap y_4 = 0 \| y_6 = A) = \frac{1}{3}$ |
| | 1 | $P(y_3 = 0 \cap y_4 = 1 \| y_6 = A) = \frac{1}{3}$ | $P(y_3 = 1 \cap y_4 = 1 \| y_6 = A) = \frac{1}{3}$ |

Table 2: Distribution of $y_3$ and $y_4$ given $y_6 = A$

| $P(y_3 \cap y_4 \mid y_6 = B)$ | | $y_3 = 0$ | $y_3 = 1$ |
|---|---|---|---|
| $y_4$ | 0 | $P(y_3 = 0 \cap y_4 = 0 \mid y_6 = B) = \frac{1}{2}$ | $P(y_3 = 1 \cap y_4 = 0 \mid y_6 = B) = \frac{1}{4}$ |
| | 1 | $P(y_3 = 0 \cap y_4 = 1 \mid y_6 = B) = \frac{1}{4}$ | $P(y_3 = 1 \cap y_4 = 1 \mid y_6 = B) = 0$ |

Table 3: Distribution of $y_3$ and $y_4$ given $y_6 = B$

**Distribution of $y_5$**

The class conditional distribution of $y_5$ is given by:

| $P(y_5 \mid y_6)$ | | $y_5 = 0$ | $y_5 = 1$ | $y_5 = 2$ |
|---|---|---|---|---|
| $y_6$ | A | $P(y_5 = 0 \mid y_6 = A) = \frac{1}{3}$ | $P(y_5 = 1 \mid y_6 = A) = \frac{1}{3}$ | $P(y_5 = 2 \mid y_6 = A) = \frac{1}{3}$ |
| | B | $P(y_5 = 0 \mid y_6 = B) = \frac{1}{4}$ | $P(y_5 = 1 \mid y_6 = B) = \frac{1}{2}$ | $P(y_5 = 2 \mid y_6 = B) = \frac{1}{4}$ |

Table 4: Distribution of $y_5$ given $y_6$

## b)

In order to classify the testing observations, we will need to compute the posterior probabilities. Under a MAP assumption, the predicted class for each testing observation is the one that maximizes the posterior probability. Since we are only interested in the maximum value over all classes, we can ignore the denominator of the posterior probability formula. We have two testing observations, $x_8$ and $x_9$, and we will compute the posterior probabilities for each of them:

**Posterior probabilities for $x_8$**

This training observation has the following values for the input variables: $y_1 = 0.38$, $y_2 = 0.52$, $y_3 = 0$, $y_4 = 1$ and $y_5 = 0$.

$$P(y_6 = A \mid x_8) = \frac{P(x_8 \mid y_6 = A) \cdot P(y_6 = A)}{P(x_8)} \propto P(x_8 \mid y_6 = A) \cdot P(y_6 = A) =$$

$$= P(y_1 = 0.38, y_2 = 0.52 \mid y_6 = A) \cdot P(y_3 = 0, y_4 = 1 \mid y_6 = A) \cdot P(y_5 = 0 \mid y_6 = A) \cdot P(y_6 = A) =$$

$$= \frac{1}{\sqrt{(2\pi)^2 \cdot 5.4613 \cdot 10^{-5}}} \exp\left(-\frac{1}{2}\begin{bmatrix} 0.38 - 0.24 \\ 0.52 - 0.52 \end{bmatrix}^T \cdot \begin{bmatrix} 410.156 & -117.188 \\ -117.188 & 78.125 \end{bmatrix} \cdot \begin{bmatrix} 0.38 - 0.24 \\ 0.52 - 0.52 \end{bmatrix}\right) \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{3}{7} =$$

$$= 0.3868 \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{3}{7} = 0.01842$$

$$P(y_6 = B|x_8) = \frac{P(x_8|y_6 = B) \cdot P(y_6 = B)}{P(x_8)} \propto P(x_8|y_6 = B) \cdot P(y_6 = B) =$$

$$= P(y_1 = 0.38, y_2 = 0.52|y_6 = B) \cdot P(y_3 = 0, y_4 = 1|y_6 = B) \cdot P(y_5 = 0|y_6 = B) \cdot P(y_6 = B) =$$

$$= \frac{1}{\sqrt{(2\pi)^2 \cdot 3.519 \cdot 10^{-4}}} \exp\left(-\frac{1}{2}\begin{bmatrix}0.38 - 0.5925 \\ 0.52 - 0.3274\end{bmatrix}^T \cdot \begin{bmatrix}67.1101 & 20.7954 \\ 20.7954 & 48.7831\end{bmatrix} \cdot \begin{bmatrix}0.38 - 0.5925 \\ 0.52 - 0.3274\end{bmatrix}\right) \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{4}{7} =$$

$$= 1.7677 \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{4}{7} = 0.06313$$

**Posterior probabilities for $x_9$**

This training observation has the following values for the input variables: $y_1 = 0.42$, $y_2 = 0.59$, $y_3 = 0$, $y_4 = 1$ and $y_5 = 1$.

$$P(y_6 = A|x_9) = \frac{P(x_9|y_6 = A) \cdot P(y_6 = A)}{P(x_9)} \propto P(x_9|y_6 = A) \cdot P(y_6 = A) =$$

$$= P(y_1 = 0.42, y_2 = 0.59|y_6 = A) \cdot P(y_3 = 0, y_4 = 1|y_6 = A) \cdot P(y_5 = 1|y_6 = A) \cdot P(y_6 = A) =$$

$$= \frac{1}{\sqrt{(2\pi)^2 \cdot 5.4613 \cdot 10^{-5}}} \exp\left(-\frac{1}{2}\begin{bmatrix}0.42 - 0.24 \\ 0.59 - 0.52\end{bmatrix}^T \cdot \begin{bmatrix}410.156 & -117.188 \\ -117.188 & 78.125\end{bmatrix} \cdot \begin{bmatrix}0.42 - 0.24 \\ 0.59 - 0.52\end{bmatrix}\right) \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{3}{7} =$$

$$= 0.1013 \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{3}{7} = 0.00482$$

$$P(y_6 = B|x_9) = \frac{P(x_9|y_6 = B) \cdot P(y_6 = B)}{P(x_9)} \propto P(x_9|y_6 = B) \cdot P(y_6 = B) =$$

$$= P(y_1 = 0.42, y_2 = 0.59|y_6 = B) \cdot P(y_3 = 0, y_4 = 1|y_6 = B) \cdot P(y_5 = 1|y_6 = B) \cdot P(y_6 = B) =$$

$$= \frac{1}{\sqrt{(2\pi)^2 \cdot 3.519 \cdot 10^{-4}}} \exp\left(-\frac{1}{2}\begin{bmatrix}0.42 - 0.5925 \\ 0.59 - 0.3274\end{bmatrix}^T \cdot \begin{bmatrix}67.1101 & 20.7954 \\ 20.7954 & 48.7831\end{bmatrix} \cdot \begin{bmatrix}0.42 - 0.5925 \\ 0.59 - 0.3274\end{bmatrix}\right) \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{4}{7} =$$

$$= 1.4927 \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{4}{7} = 0.05331$$

**Predicted classes**

Organizing the posterior probabilities in a table, we have:

| Observation | $P(y_6 = A\|x_i)$ | $P(y_6 = B\|x_i)$ |
|---|---|---|
| $x_8$ | 0.01842 | **0.06313** |
| $x_9$ | 0.00482 | **0.05331** |

Table 5: Posterior probabilities for the testing observations

Therefore, the predicted class for both $x_8$ and $x_9$ is $y_6 = B$.

## c)

Let's consider the following classifier with a unknown threshold $\theta$ whose value we aim to find:

$$f(x|\theta) = \begin{cases} A & \text{if } P(y_6 = A|x) > \theta \\ B & \text{if } P(y_6 = A|x) \leq \theta \end{cases}$$

**Finding** $P(y_6 = A|x)$

We are now working under an ML assumption, so, in order to classify each test observation we will only need the conditional distribution $P(x|y_6 = A)$ because every class has the same prior probability:

$$P(y_6 = A) = P(y_6 = B) = \frac{1}{2}$$

$$P(y_6 = A|x_8) = \frac{P(x_8|y_6 = A) \cdot P(y_6 = A)}{P(x_8)} = \frac{P(x_8|y_6 = A) \cdot \cancel{P(y_6 = A)}}{P(x_8|y_6 = A) \cdot \cancel{P(y_6 = A)} + P(x_8|y_6 = B) \cdot \cancel{P(y_6 = B)}} =$$

$$= \frac{P(x_8|y_6 = A)}{P(x_8|y_6 = A) + P(x_8|y_6 = B)} =$$

$$P(y_6 = A|x_9) = \frac{P(x_9|y_6 = A) \cdot P(y_6 = A)}{P(x_9)} = \frac{P(x_9|y_6 = A) \cdot \cancel{P(y_6 = A)}}{P(x_9|y_6 = A) \cdot \cancel{P(y_6 = A)} + P(x_9|y_6 = B) \cdot \cancel{P(y_6 = B)}} =$$

$$= \frac{P(x_9|y_6 = A)}{P(x_9|y_6 = A) + P(x_9|y_6 = B)} =$$

**Accuracy** The accuracy of a classifier is given by:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where $TP$ is the number of true positives, $TN$ is the number of true negatives, $FP$ is the number of false positives and $FN$ is the number of false negatives. In our case, accuracy can only have three possible values: 0, 0.5 or 1: 0 when both $x_8$ and $x_9$ are misclassified, 0.5 when only one of them is misclassified and 1 when both are correctly classified. In order to maximize the accuracy, we want it to be 1. We know from table **??** that $x_8$ belongs to class $A$ and $x_9$ belongs to class $B$. Therefore, in order to maximize the accuracy, our classifier needs to classify $x_8$ as $A$ and $x_9$ as $B$. $f(x_8|\theta) = A$ therefore $\theta < P(y_6 = A|x_8) = 0.04298$ and $f(x_9|\theta) = B$ therefore $\theta > P(y_6 = A|x_9) = 0.01126$.

$$\theta \in ]0.01126; 0.04298[$$

Any value of $\theta$ in this interval will maximize the accuracy of our classifier.

# $2^{nd}$ Question

## a)

In order to obtain $y_2$ under an equal-range discretization, we followed the rule:

$$y_{2_{\text{normalized}}} = \begin{cases} 0 & \text{if } y_2 \in [0, 0.5) \\ 1 & \text{if } y_2 \in [0.5, 1] \end{cases}$$

The normalized values are:

| Dataset | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $y_2$ | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |

Table 6: Normalized $y_2$ values

$y_1$ will now be considered an output variable and $y_2$ to $y_6$ will be considered input variables. Considering the normalized values of $y_2$, we can rewrite the dataset as follows:

| D | | Output | Input | | | | |
|---|---|---|---|---|---|---|---|
| Fold | | $y_1$ | $y_{2_{norm}}$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ |
| $F_1$ | $x_1$ | 0.24 | 0 | 1 | 1 | 0 | A |
| | $x_2$ | 0.16 | 0 | 1 | 0 | 1 | A |
| | $x_3$ | 0.32 | 1 | 0 | 1 | 2 | A |
| $F_2$ | $x_4$ | 0.54 | 0 | 0 | 0 | 1 | B |
| | $x_5$ | 0.66 | 0 | 0 | 0 | 0 | B |
| | $x_6$ | 0.76 | 0 | 1 | 0 | 2 | B |
| $F_3$ | $x_7$ | 0.41 | 1 | 0 | 1 | 1 | B |
| | $x_8$ | 0.38 | 1 | 0 | 1 | 0 | A |
| | $x_9$ | 0.42 | 1 | 0 | 1 | 1 | B |

Table 7: Dataset $D$ divided into three folds

Adittionally, we have divided the dataset into three folds, $F_1$, $F_2$ and $F_3$.

## b)

In this exercise we aim to compute a kNN (k nearest neighbors - Lazy Learning) classifier considering the following parameters:

- $k = 3$
- **Hamming Distance** as the distance to be used to compute the nearest neighbors of a given observation.

$$d_H(x_i, x_j) = \sum_{l=1}^{m} \delta(y_{il}, y_{jl})$$

where $m$ is the number of input variables and $y_{il}$ is the value of the $l^{\text{th}}$ input variable of the $i^{\text{th}}$ observation.

We have divided our dataset in folds in order to perform a cross validation. We will only be interested in the first iteration of the cross validation, where $F_3$ is the testing fold and $F_1$ and $F_2$ are the training folds. We will now compute the kNN classifier for each observation in $F_3$ and afterwards we will compute the MAE (Mean Absolute Error) for the testing fold.

**Computing the Hamming Distances**

| Testing Observation ($x_i$) | $d_H(x_i, x_1)$ | $d_H(x_i, x_2)$ | $d_H(x_i, x_3)$ | $d_H(x_i, x_4)$ | $d_H(x_i, x_5)$ | $d_H(x_i, x_6)$ |
|---|---|---|---|---|---|---|
| $x_7$ | 4 | 4 | 2 | 2 | 3 | 4 |
| $x_8$ | 2 | 4 | 1 | 4 | 3 | 5 |
| $x_9$ | 4 | 4 | 2 | 2 | 3 | 4 |

Table 8: Hamming distances between the testing observation $x_i$ and the training observations $x_j$

We are considering $k = 3$, so we will only need the three nearest neighbors of each testing observation. In the table above we have filled with yellow the three nearest neighbors of each testing observation.

**Predicted value of $y_1$ for each testing observation**

The output value we are working with is numerical, so the predicted value of $y_1$ for each testing observation will be:

$$\hat{y}_{1j} = \frac{\sum_{i=1}^{k} \frac{1}{d_H(x_i, x_j)} \cdot y_{1j}}{\sum_{i=1}^{k} \frac{1}{d_H(x_i, x_j)}}$$

where $k$ is the number of nearest neighbors, $d_H(x_i, x_j)$ is the Hamming distance between the testing observation $x_i$ and the $i^{\text{th}}$ nearest neighbor and $y_{1j}$ is the value of the output variable of the $j^{\text{th}}$ nearest neighbor. $\frac{1}{d_H(x_i, x_j)}$ is the weight of the $i^{\text{th}}$ nearest neighbor.

$$\hat{y}_{17} = \frac{1}{\frac{1}{2} + \frac{1}{2} + \frac{1}{3}} \cdot \left(\frac{1}{2} \cdot 0.32 + \frac{1}{2} \cdot 0.54 + \frac{1}{3} \cdot 0.66\right) = 0.4875$$

$$\hat{y}_{18} = \frac{1}{\frac{1}{2} + \frac{1}{1} + \frac{1}{3}} \cdot \left(\frac{1}{2} \cdot 0.24 + 1 \cdot 0.32 + \frac{1}{3} \cdot 0.66\right) = 0.36$$

$$\hat{y}_{19} = \frac{1}{\frac{1}{2} + \frac{1}{2} + \frac{1}{3}} \cdot \left(\frac{1}{2} \cdot 0.32 + \frac{1}{2} \cdot 0.54 + \frac{1}{3} \cdot 0.66\right) = 0.4875$$

We have the following predicted values for $y_1$:

| Testing Observation ($x_i$) | $\hat{y}_{1i}$ | $y_{1i}$ |
|---|---|---|
| $x_7$ | 0.4875 | 0.41 |
| $x_8$ | 0.36 | 0.38 |
| $x_9$ | 0.4875 | 0.42 |

Table 9: Predicted values of $y_1$ for each testing observation

### MAE

The MAE is given by:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_{1i} - \hat{y_1}_i|$$

where $n$ is the number of testing observations, $y_{1i}$ is the value of the output variable of the $i^{\text{th}}$ testing observation and $\hat{y_1}_i$ is the predicted value of the output variable of the $i^{\text{th}}$ testing observation.

$$\text{MAE} = \frac{1}{3} \cdot (|0.41 - 0.4875| + |0.38 - 0.36| + |0.42 - 0.4875|) = 0.055$$

# Programming and Critical Analysis

## Imports

```python
# Sklearn Imports
import sklearn as sk
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
# Other Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from scipy.io.arff import loadarff
from scipy import stats
```

Listing 1:

## Loading DataSet and Define Directories

```python
IMAGES_DIR = Path('images')
IMAGES_DIR.mkdir(parents=True, exist_ok=True)
DATA_DIR = Path('data')
DATA_DIR.mkdir(parents=True, exist_ok=True)
DATA_FILE = 'column_diagnosis.arff'
DATA_PATH = DATA_DIR / DATA_FILE
data = loadarff(DATA_PATH)
df = pd.DataFrame(data[0])
df['class'] = df['class'].str.decode('utf-8')
# Show the first 5 rows
df.head()
```

Listing 2:

## 1st Question

### 10-fold cross validation with suffling

```python
    # Split into features and labels
    X = df.drop('class', axis=1)
    y = df['class']

```

Listing 3:

```python
    # Stratified 10 fold cross validation
    cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)

```

Listing 4:

### Box Plots for Fold Accuracy

```python
# kNN Classifier with 5 neighbors
knn = KNeighborsClassifier(n_neighbors=5)

# Naive Bayes Classifier
nb = GaussianNB()

# 10-fold startified cross validation with shuffle
knn_score = cross_val_score(knn,
                            X,
                            y,
                            cv=cv,
                            scoring='accuracy')

nb_score = cross_val_score(nb,
                           X,
                           y,
                           cv=cv,
                           scoring='accuracy')
```

Listing 5:

```python
# Create boxplots for the accuracies of both classifiers
plt.boxplot([knn_score, nb_score], labels=['k-NN', 'Naive Bayes'])
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison Between k-NN and Naive Bayes')
plt.savefig(IMAGES_DIR / 'boxplot.png')
plt.show()
```

Listing 6:

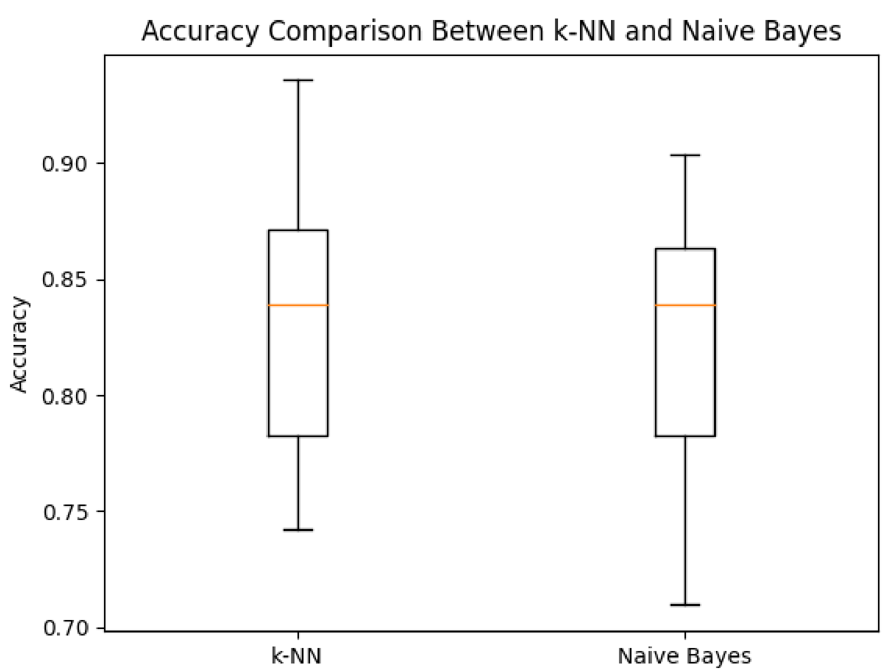


Figure 1: Boxplot for the accuracies of both classifiers

**Comment:** As seen by the box plot above, both models have an equal central tendency, shown by the median. Beyond that, the variance of the models is quite similar, since the overall amplitude of the box plots is similar. However, Naive Bayes has a variance towards the lower end of the box plot, meaning it has a tendency for lower accuracy.

```
  # Perform a two-sample t-test to compare the means
t_stat, p_value = stats.ttest_rel(knn_score, nb_score, alternative='greater')
print(f"t-statistic: {str(t_stat)}")
print(f"p-value: {str(p_value)}")

# Determine if the null hypothesis can be rejected (p < 0.05)
alpha = 0.05
if p_value < alpha:
    print("Null hypothesis rejected: \nkNN is statistically superior to Naive Bayes.")
else:
    print("Null hypothesis not rejected: \nNo significant difference between kNN and
  Naive Bayes.")

```

Listing 7:

```
  t-statistic: 0.9214426752509264
  p-value: 0.19042809062064092
  Null hypothesis not rejected:
  No significant difference between kNN and Naive Bayes.

```

Listing 8:

## 2nd Question

```
# Create k-NN classifiers with k=1 and k=5
knn_k1 = KNeighborsClassifier(n_neighbors=1, weights='uniform', metric='euclidean')
knn_k5 = KNeighborsClassifier(n_neighbors=5, weights='uniform', metric='euclidean')

# Create a confusion matrix for each classifier
confusion_matrix_k1 = np.zeros((3, 3))
confusion_matrix_k5 = np.zeros((3, 3))

for train_index, test_index in cv.split(X, y):
    # Split the data into training and testing sets
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Fit both models to the dataset
    knn_k1.fit(X_train, y_train)
    knn_k5.fit(X_train, y_train)

    # Make predictions using both models
    predictions_k1 = knn_k1.predict(X_test)
    predictions_k5 = knn_k5.predict(X_test)

    # Add the confusion matrices to the cumulative confusion matrices
    confusion_matrix_k1 += confusion_matrix(y_test, predictions_k1)
    confusion_matrix_k5 += confusion_matrix(y_test, predictions_k5)

# Plot the differences between the cumulative confusion matrices
# Get the class names
class_names = np.unique(y)

# Calculate the difference between the cumulative confusion matrices
confusion_matrix_difference = confusion_matrix_k1 - confusion_matrix_k5
```

```python

# Plot the differences between the cumulative confusion matrices
plt.figure(figsize=(8, 6))
plt.imshow(confusion_matrix_difference, cmap='coolwarm', interpolation='nearest')
plt.colorbar(label='Difference')
plt.title('Confusion Matrix Difference (k=1 - k=5)')
plt.xticks(ticks=np.arange(len(class_names)), labels=class_names)
plt.yticks(ticks=np.arange(len(class_names)), labels=class_names)
plt.xlabel('Predicted Class')
plt.ylabel('True Class')

# Loop over the cells and add the values to the plot
for i in range(len(class_names)):
    for j in range(len(class_names)):
        plt.text(j, i, str(confusion_matrix_difference[i, j]), horizontalalignment='center', verticalalignment='center')

plt.savefig(IMAGES_DIR / 'confusion_matrix_difference.png')
plt.show()
```

Listing 9:

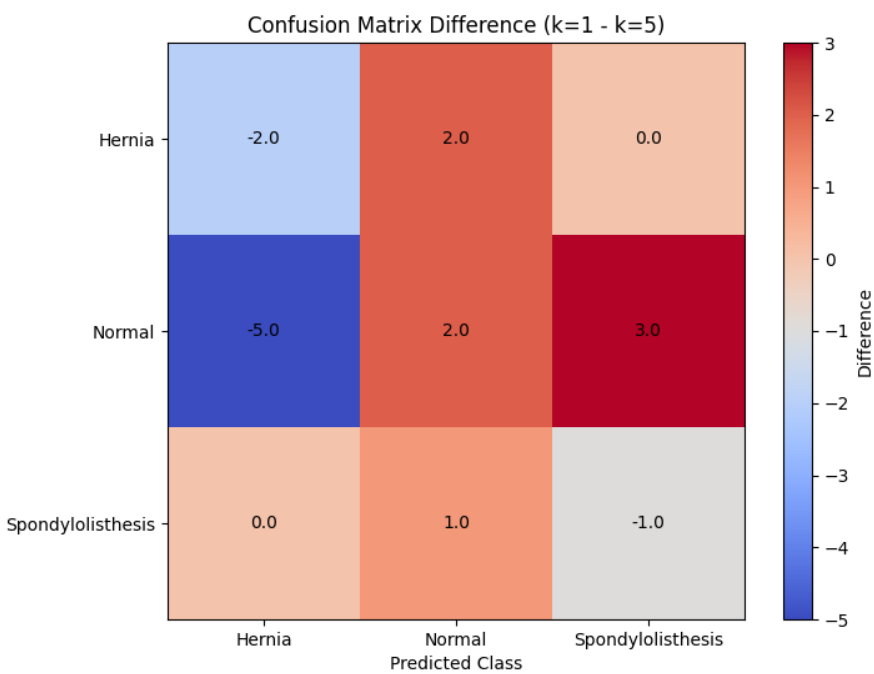


Figure 2: Confusion Matrix Difference (k=1 - k=5)

**Comment:** As seen by the difference confusion cummulative matrix above, the classifiers do not have significant differences, since the highest difference between them is 5. As for some entraces of the matrix, one can say both classifiers have the same performance, since the difference between them is 0. For negative values on the diagonal, it means that the classifier knn5 has a better performance than the classifier knn1, and for positive values, the opposite. As one can see, what stands out prominently in the matrix is the prevalence of positive values, outnumbering the negative ones. This observation implies that k-NN5 demonstrates superior performance in distinguishing between classes and encounters fewer instances of confusion compared to k-NN1.

## 3rd Question

There are some problems that Naive Bayes may encounter. The tree main problems we can think of are:

- Independence Assumption
- Continuous Features
- Imbalanced Classes

**Independence Assumption**

Naive Bayes assumes that features are conditionally independent given the class label. In other words, it assumes that there are no correlations between the features. However, in real-world datasets like medical diagnoses, features may be correlated. For example, certain symptoms or medical test results might be related. If there are strong correlations between features, Naive Bayes may not capture these relationships accurately, leading to suboptimal performance.

**Continuous Feature**

If the dataset contains continuous or numerical features, Naive Bayes relies on assuming that the data follows a specific probability distribution (e.g., Gaussian for Gaussian Naive Bayes). If the data distribution significantly deviates from this assumption, the model's performance can degrade. Medical datasets often contain a mix of continuous and categorical features, and handling continuous data can be a challenge for Naive Bayes.

**Imbalanced Classes**

If the dataset has imbalanced class distributions, where one class significantly outnumbers the others, Naive Bayes can be biased towards the majority class. This is because the class prior probabilities heavily influence the classification decision. In medical datasets, it's common to have imbalanced class distributions, such as a rare disease. Naive Bayes may struggle to correctly classify the minority class due to the bias.

To address if those problems exist in our dataset, we can do the following:

- Check if there are correlations between features
- Check if the data follows the assumed probability distribution
- Check if the class distributions are balanced

We do that in the following code:

### Independence Assumption

```python
# Independence Assumption
# Calculate pairwise feature correlations
correlation_matrix = X.corr()

# Visualize feature correlations using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.savefig(IMAGES_DIR / 'correlation_heatmap.png')
plt.show()
```

Listing 10:

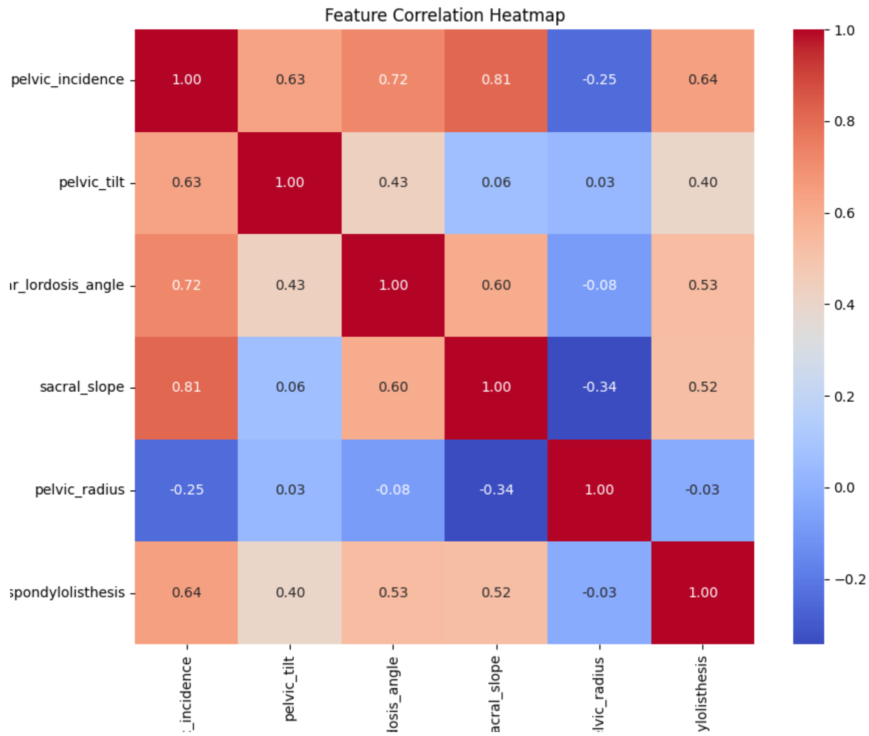

Figure 3: Feature Correlation Heatmap

**Comment:** As one can see in the correlation matrix, there are some features that are correlated. For example, the features "sacral slope" and "pelvic incidence" are correlated with a correlation coefficient of 0.81, which is considered as a strong correlation. This means that Naive Bayes may not be the best choice for this dataset, since it assumes that features are independent.

### Continuous Features

```
# Continuous Features
# Identify continuous features (assuming all non-integer features are continuous)
continuous_features = [col for col in X.columns if X[col].dtype != 'int64']

# Plot histograms for continuous features
num_cols = 3
num_rows = int(np.ceil(len(continuous_features) / num_cols))
fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 4*num_rows))

for i, feature in enumerate(continuous_features):
    row = i // num_cols
    col = i % num_cols
    sns.histplot(data=X, x=feature, kde=True, color='skyblue', ax=axs[row][col])
    axs[row][col].set_title(f'Distribution of {feature}')

plt.savefig(IMAGES_DIR / 'continuous_features.png')
plt.show()
```

Listing 11:

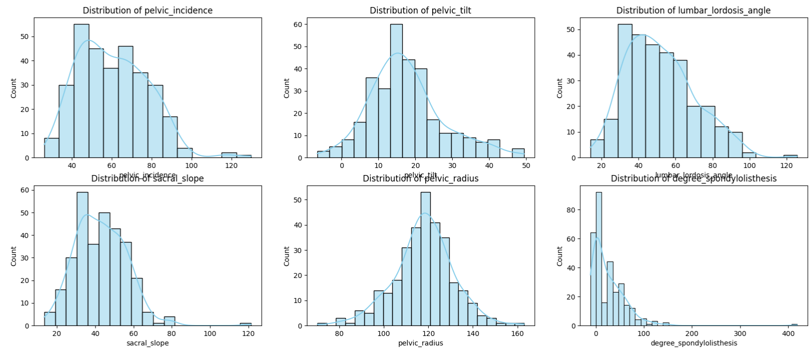


Figure 4: Distribution of continuous features

**Comment:** As one can see in the above plot, almost all features resemble a Gaussian distribution. However, there are some features that do not follow a Gaussian distribution. For example, the Degree Spondylolisthesis is close to the origin, more like a Gamma distribution. This may cause problems for Naive Bayes, since it assumes a Gaussian distribution. To resolve this possible issue, one can try to preprocess the data in order to make it more Gaussian-like.

### Imbalanced Classes

```python
# Imbalanced Classes
# Count class frequencies
class_counts = y.value_counts()

# Visualize class distribution
plt.figure(figsize=(6, 4))
sns.barplot(x=class_counts.index, y=class_counts.values, palette="Set2")
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.savefig(IMAGES_DIR / 'class_distribution.png')
plt.show()

# Output class frequencies
print("Class Frequencies:")
print(class_counts)
```

Listing 12:

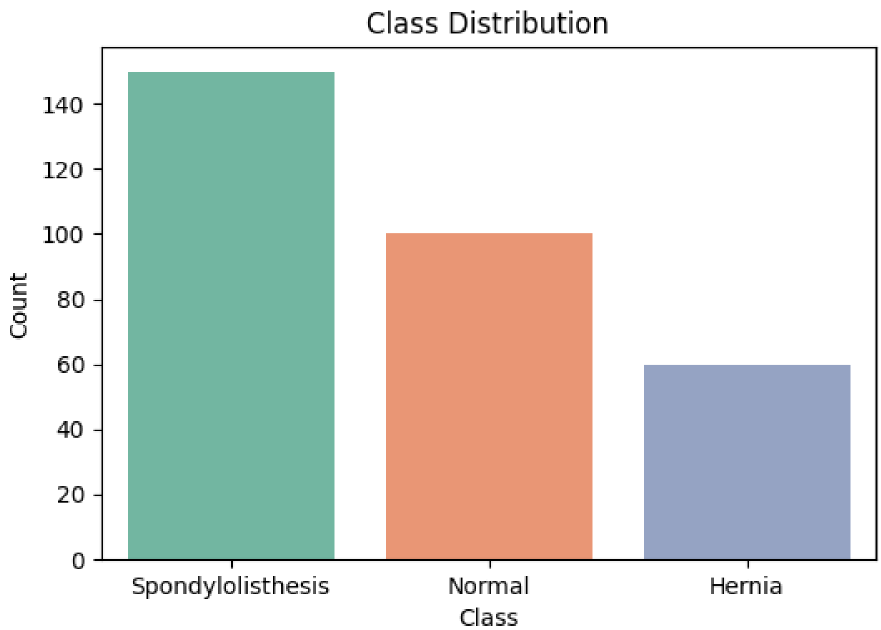


Figure 5: Class Distribution

```
Class Frequencies:
class
Spondylolisthesis    150
Normal               100
Hernia                60
Name: count, dtype: int64
```

Listing 13:

**Comment:** As one can see in the above plot, the class distributions do not seem to be balanced. The Spondylolisthesis class significantly outnumbers the other two classes, making it more than the double of the Hernia class and 50% more than the Normal Class. This may cause problems for Naive Bayes, since it can be biased towards the majority class. To resolve this possible issue, one can try to balance the class distributions by either undersampling the majority class or oversampling the minority classes.