TOUNSI Aya
HARAUCOURT Jade
UNG Norith

ECE
ÉCOLE D'INGÉNIEURS
ENGINEERING SCHOOL

Machine Learning Project

Summary

I. Data Analysis

1.1 Introduction to the dataset

We are working on a dataset containing features of mobile phones to predict their price range. We are assisting 'Bob', who has started his own mobile company, in estimating the prices of the mobiles his company produces based on features such as RAM, internal memory, and more. The price range is categorized from 0 (low price range) to 3 (very high price range). Before moving to prediction, we need to perform data analysis to prepare the data and extract statistical information.

Link to the dataset : https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification/data

The dataset contains **21 features** with **2,000 records.**

1.2 Extracting Statical Information

Overview of the dataset :

The image below provides a summary of the dataset's features and class proportions.

```
Number of Features: 21

Feature Types:
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object

Class Proportions (price_range):
Class 1: 25.00%
Class 2: 25.00%
Class 3: 25.00%
Class 0: 25.00%
```

Here's the statistical information we can extract:

- The number of features used is **21 features**.
- All features are **numeric** except for **the target variable price_range (dtype: object)**.
- The target variable price_range is divided into **four classes**: 0, 1, 2, and 3, with **equal proportions** (25% each). This indicates a perfectly balanced dataset with equal representation of each price range.
- The features include:
    - **Device specifications:** battery_power, blue (Bluetooth support), clock_speed, dual_sim, four_g, three_g, touch_screen, wifi.

- **Performance-related:** int_memory (internal memory), n_cores (number of CPU cores), ram (random access memory).
- **Display-related:** px_height (pixel height), px_width (pixel width), sc_h (screen height), sc_w (screen width), m_dep (mobile depth).
- **Camera specifications:** fc (front camera), pc (primary camera).
- **Usability features:** mobile_wt (mobile weight), talk_time.

Missing values:

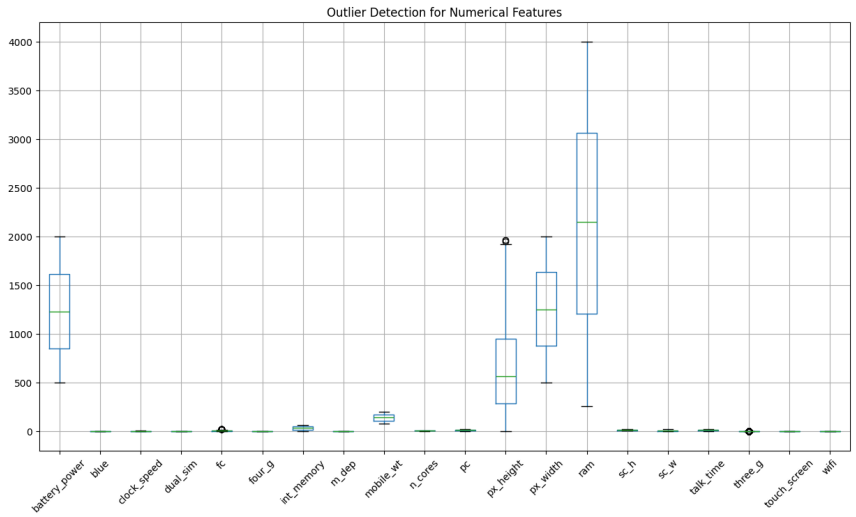
The image below shows a summary of missing values in the dataset.

```
Missing Values:
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

The dataset contains **no missing values** across all 21 features. Each feature has complete data for all 2,000 records.

Outlier Detection

The boxplot provided showcases outlier detection for numerical features in the dataset :



- Binary features like blue, four_g, three_g, wifi, and touch_screen do not exhibit outliers, as expected, since they take values of either 0 or 1.
- Battery_power and Ram features have no extreme outliers, but the distributions show a wide range of values, with some devices having higher values in their respective ranges.

The Z-score: measures how far a data point is from the mean in terms of standard deviations. A common threshold is ±3 (data points outside this range are considered outliers).

```
Number of outliers detected using Z-score: 12

Columns with Outliers: ['fc']
```

Outliers are present in the feature fc (Front Camera). This indicates that some devices have unusually high or low front camera resolutions compared to the rest of the dataset. After investigating these outliers, we observed that they are valid data points, and we chose to keep them. Since they represent a small number of instances, they are unlikely to significantly impact the overall performance of the analysis or model.

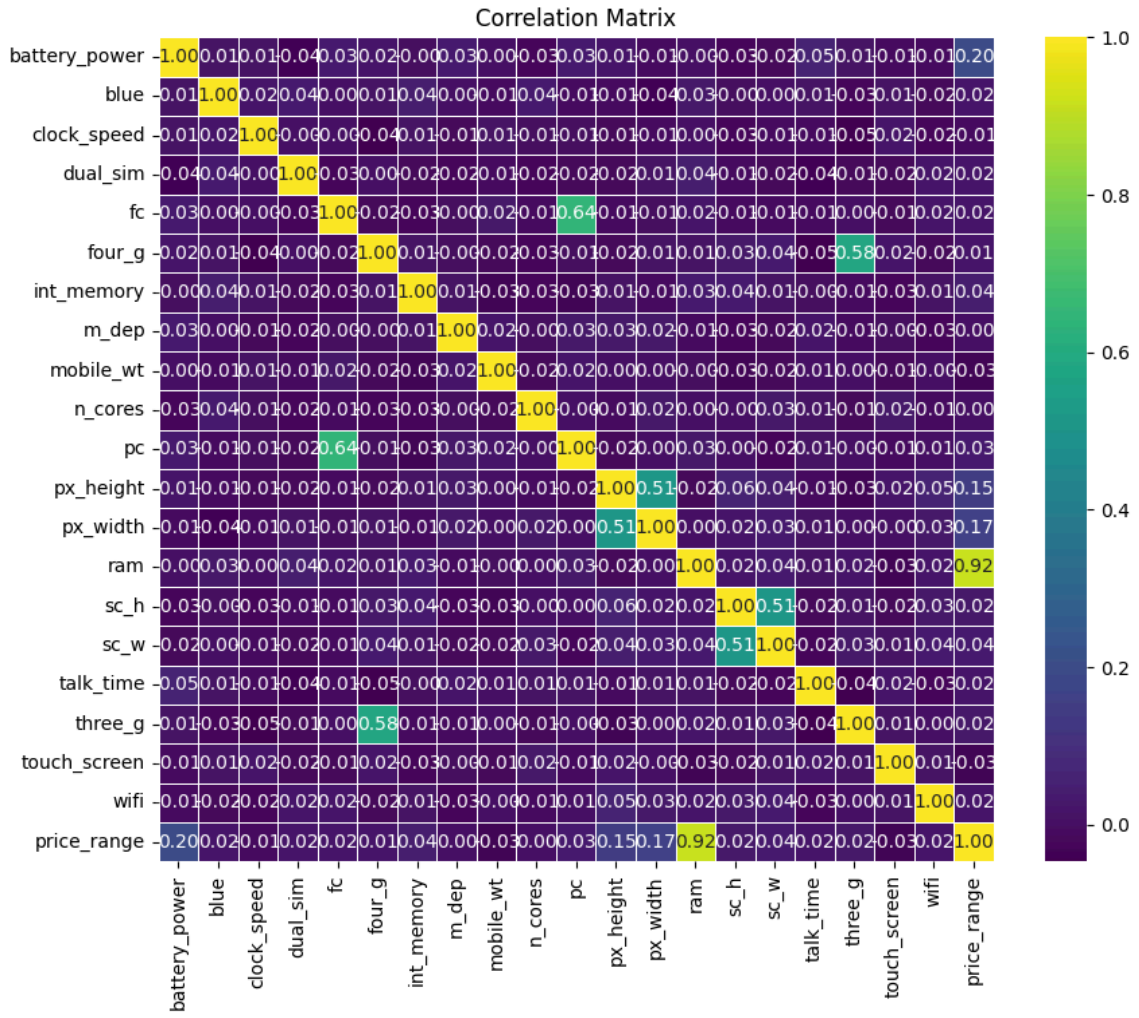
Standardization :

The dataset was standardized using z-score normalization, which transforms the data to have **a mean of 0** and **a standard deviation of 1** for each feature. All features are scaled to the same range, ensuring fair contribution to distance-based models (e.g., KNN, SVM) and gradient-based models (e.g., logistic regression, neural networks).
Each feature's values are now relative to their mean and standard deviation:

- **Negative Values:** Indicate data points below the feature's mean.
- **Positive Values:** Indicate data points above the feature's mean.
- **Values Near 0:** Indicate data points close to the feature's mean.

Correlation between the variables :

The correlation matrix shows the strength and direction of the linear relationship between the features and the target variable (price_range). Here’s a detailed analysis:



- **Ram** is the **most important feature** for determining the price_range, followed by battery_power, and display resolution (px_width and px_height).
- Features like **dual_sim, blue, three_g, four_g**, and **wifi** are not significant differentiators in price range because these are now standard in most phones.
- Features with very low or no correlation (e.g., m_dep, n_cores, clock_speed) may not significantly impact the model and can potentially be dropped to simplify the model.

Feature Reduction: Removing Specific Columns

To improve model efficiency and focus on the most relevant features, the following columns were removed from the dataset: **m_dep (Mobile Depth)**, **n_cores (Number of CPU Cores)**, and **clock_speed (Processor Clock Speed)**. Removing these low-impact features helps **reduce noise in the data**, allowing the model to focus on the features that have a stronger correlation with the target variable.

II. Classification

2.1.Introduction

In this section, the aim is to predict the class of a new set of objects. Four methods will be employed: Support Vector Machine (SVM),Decision tree classifier, k-nearest neighbors (KNN) and Naive Bayes Classifier.
We will study for each model the performance (accuracy), the classification report (precision,recall,f1-score,support) and the confusion matrix.

True Positives: Examples that belong to the class and that the model correctly identified as belonging to the class.
True Negatives: Examples that do not belong to the class and that the model correctly identified as not belonging to the class.
False Positives: Examples that do not belong to the class but the model incorrectly identified as belonging to the class.
False Negatives: Examples that belong to the class but the model incorrectly identified as not belonging to the class.

The accuracy measures the total of corrected predictions on the total of predictions made. It gives an idea of the performance of the model.

$$\text{Accuracy} = \frac{\text{Total Correct Predictions}}{\text{Total Predictions Made}} = \frac{TP + TN}{TP + TN + FP + FN}$$

The classification report analyzes the metrics for each class (4 in this dataset). It includes four metrics.

- The precision measures the proportion of correct predictions on all the positive predictions made (even if it is true or false)

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

- The recall measures the proportions of real positive predictions on all the actual positive instances

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

The F1-score is the harmonic mean between the precision and the recall.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The support is the total number of real examples in each class.

$$Support(A)=\frac{Number\ of\ transactions\ containing\ A}{Total\ number\ of\ transactions}$$

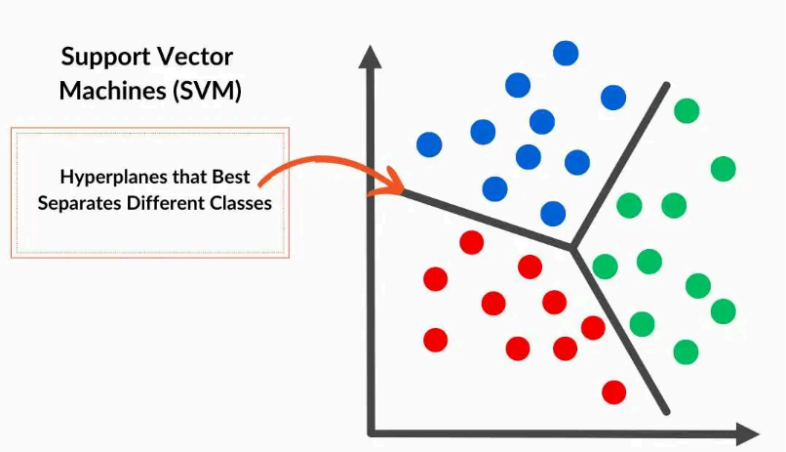
A confusion matrix is a table that evaluates the performances of a classification model comparing the predictions of the model with the actual values. It shows the number of times that a class has been correctly predicted.

For each model, the data is split into training and testing sets.The model is then initialized,trained on the training data, and used to make predictions on the testing data.

2.2. Support Vector Machine (SVM)

2.2.1. Functioning

The Support Vector Machine (SVM) is a supervised learning method used to solve classification and regression issues. The goal of SVM is to find an optimal hyperplane that separates classes of data while maximizing the margin between the hyperplane and the closest points of each class, known as support vectors.



Schema of the support vector machine (SVM)

The formula for SVM is:

$$f(x)=w.x\ +b=0$$

- w is the weight vector,
- b is the bias term that adjusts the position of the hyperplane
- x is the input vector

The aim is to find the optimal hyperplane. SVM solves a quadratic optimization issue with constraints. Thus, the norm of the weight vector w is minimized with this formula:

$$\min \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i$$

On constraints:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

- ξ_i slack variables for each data point i
- w weight vector
- b bias
- C regularization parameter
- $\sum_{i=1}^{n} \xi_i$ total sum of errors

Moreover, for non linear data, SVM uses kernels functions to project data into a higher-dimensional space. There are several types of kernels.

Linear kernel:

$$K(X_i, X_j) = X_i \cdot X_j$$

Polynomial kernel:

$$K(X_i, X_j) = (X_i \cdot X_j + c)^d$$

RBF(Radial Basis function) kernel:

$$K(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right)$$

Sigmoid kernel:

$$K(x_i, x_j) = \tanh(\alpha x_i \cdot x_j + c)$$

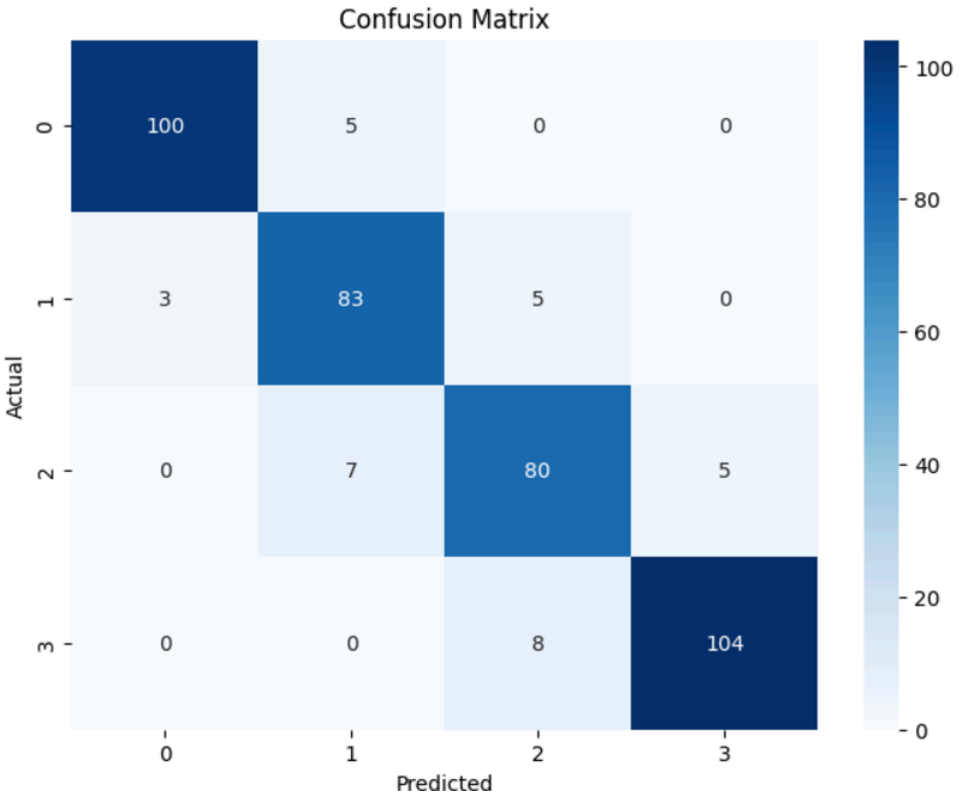
2.2.2. Results

The accuracy is 0.9175.

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.95	0.96	105
1	0.87	0.91	0.89	91
2	0.86	0.87	0.86	92
3	0.95	0.93	0.94	112
accuracy			0.92	400
macro avg	0.91	0.92	0.92	400
weighted avg	0.92	0.92	0.92	400

Screenshot of the classification report for SVM

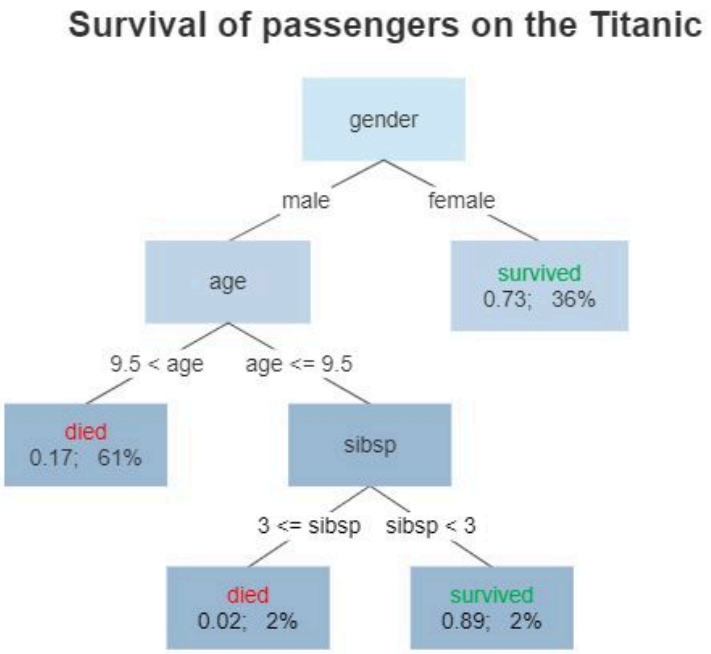


Heatmap of the confusion matrix for SVM

2.3.Decision tree classifier

2.3.1. Functioning

The decision tree classifier divides the dataset into smaller subsets based on the values of each input feature. The division is based on characteristics that maximize a measure of purity or information gain. The goal is to create a model that predicts the value of a target variable based on several input variables.



Example of schema for the Decision tree classifier

A decision tree classifier is composed of nodes and leaves.
The nodes are the characteristics on which decisions are based. The leaves are the final predictions.

How it works:

1. The model selects the characteristic that maximizes a measure of purity or information gain
2. Data are divided into subsets based on the selected characteristic's values
3. The process is repeated for each subset until all data are classified
   or a stopping criterion is reached

The formula for the **entropy** (disorder in a subset of data):

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

Pi is the proportion of class i in the subset S

Information Gain measures the reduction in entropy obtained by dividing the data based on a characteristic

$$\text{Gain}(S, A) = H(S) - \sum_{v \in \text{Valeurs}(A)} \frac{|S_v|}{|S|} H(S_v)$$

Gini impurity is another measure of purity.

$$G(S) = 1 - \sum_{i=1}^{c} p_i^2$$

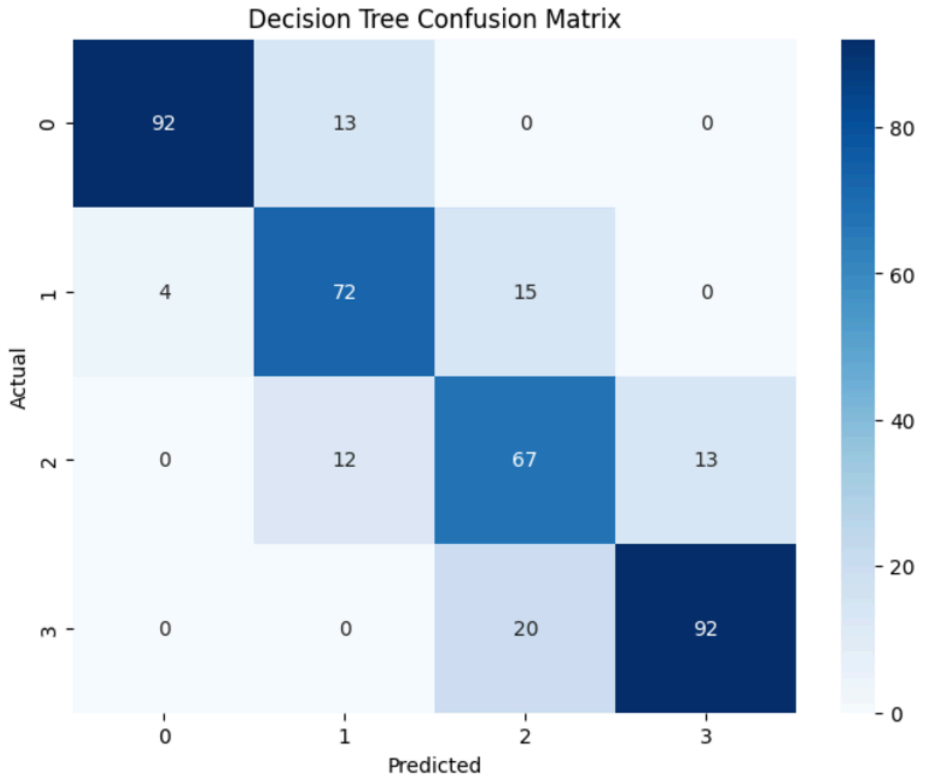
2.3.2. Results

The accuracy is 0.8075.

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.88	0.92	105
1	0.74	0.79	0.77	91
2	0.66	0.73	0.69	92
3	0.88	0.82	0.85	112
accuracy			0.81	400
macro avg	0.81	0.80	0.81	400
weighted avg	0.82	0.81	0.81	400

Schema of the classification report for Decision Tree Classifier

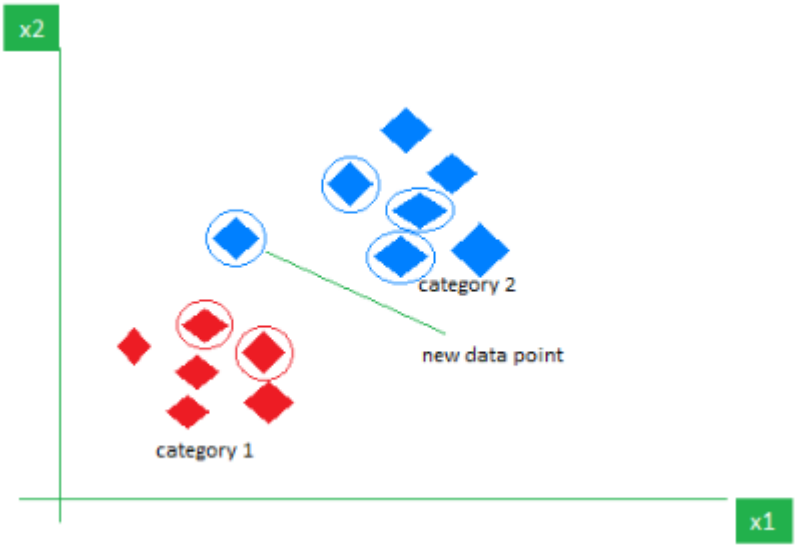


Heatmap of the confusion Matrix for Decision Tree Classifier

2.4. K-nearest neighbors

2.4.1. Functioning

The K-Nearest Neighbors (K-NN) algorithm classifies a data point based on the majority label of its k closest neighbors in the feature space.



Schema of the K-nearest neighbors

1. For each data point, K-NN calculates the distance between this point and all the points in the training set
2. K-NN selects the K closest points based on the calculated distance
3. Makes a prediction based on the most frequent label among these k neighbors

The choice of K is important. If K is too small, it can lead to overfitting whereas if k is too large it can lead to underfitting.
The formula used to calculate the distance of the point is the Euclidienne distance:

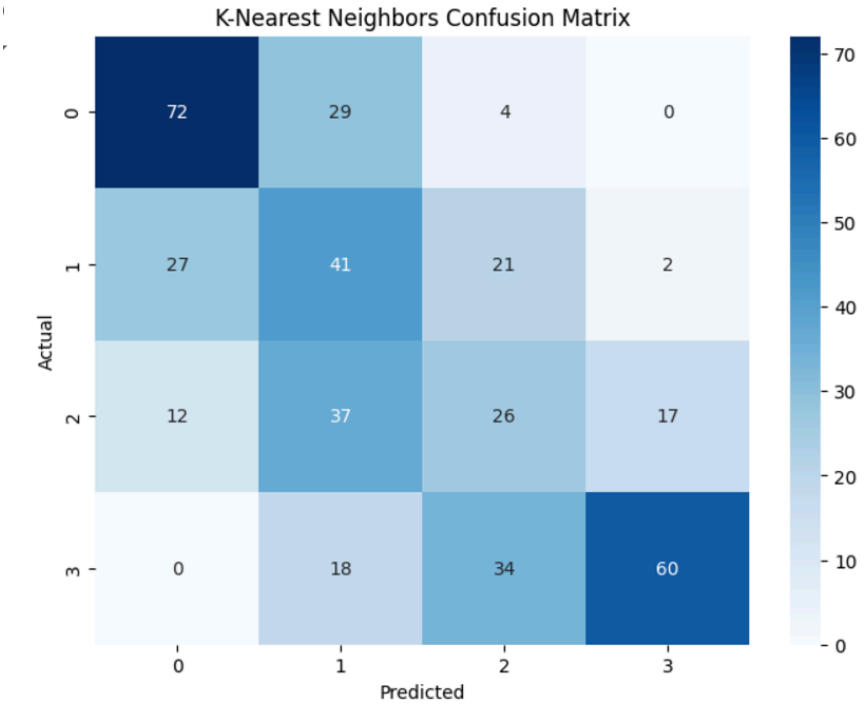
$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

2.4.2. Results
The accuracy is 0.4975.

Classification Report:

	precision	recall	f1-score	support
0	0.65	0.69	0.67	105
1	0.33	0.45	0.38	91
2	0.31	0.28	0.29	92
3	0.76	0.54	0.63	112
accuracy			0.50	400
macro avg	0.51	0.49	0.49	400
weighted avg	0.53	0.50	0.50	400

Screenshot of the classification report for K-NN

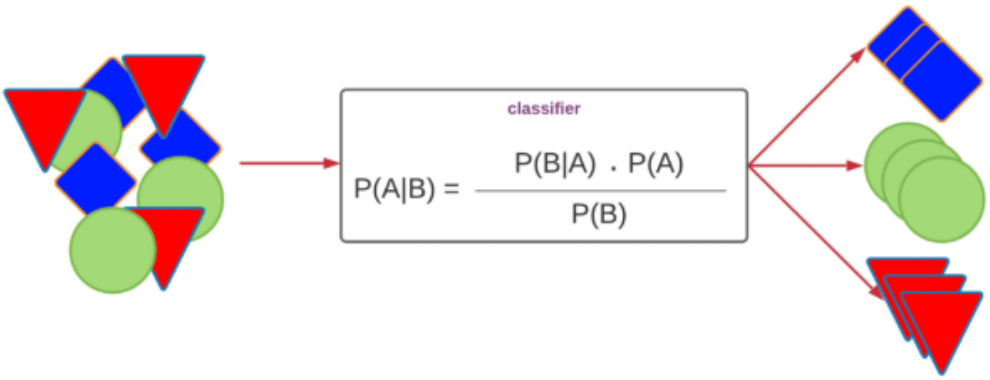


Heatmap of the confusion matrix for K-nearest neighbors

2.5. Naive Bayes Classifier

2.5.1. Functioning

The Naive Bayes Classifier uses Bayes' theorem and the naive assumption that features are independent of each other.



Schema of the Naive Bayes Classifier

Bayes theorem:

It uses to update probability a measure that new information appears with this formula:

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

$p(C_k \mid x)$ is the probability of class C_k given the feature vector X

$p(C_k)$ is the prior probability of class C_k

$p(X)$ is the probability of observing the feature vector X
$p(X \mid C)$ is the probability of observing the feature vector X given class C_k

Naive Assumption:

$$P(X|C) = \prod_{i=1}^{n} P(X_i|C)$$

- X_i represents the i-th feature of the feature vector X

For each class Ck, the probability is calculated using theorem bayes.
The prediction is made for the class with the highest posterior probability,which can be found using the formula:

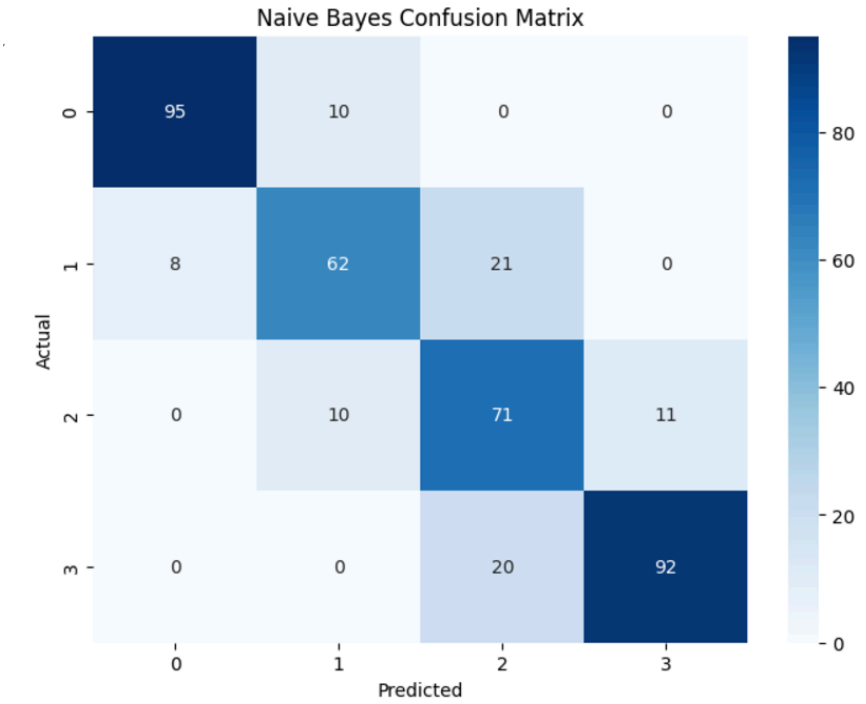
$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

2.5.2. Results
The accuracy is 0.8.

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.90	0.91	105
1	0.76	0.68	0.72	91
2	0.63	0.77	0.70	92
3	0.89	0.82	0.86	112
accuracy			0.80	400
macro avg	0.80	0.79	0.80	400
weighted avg	0.81	0.80	0.80	400

Screenshot of the classification report



Heatmap of the confusion matrix for Naive Bayes

2.6 Conclusion

2.6.1. The accuracy

We observe from the resultats that the accuracy is highest for SVM(0.9175) followed by the Decision tree classifier (0.8075), then Naives Bayes (0.8000) and the least performant is K-NN(0.4975).

2.6.2. The classification report

SVM: its performance is high. The precision, the recall and F1-score are close to 0.9 for all classes.
Decision tree classifier: The precision, the recall and F1-score are around 0.8. Class 0 is well-predicted but class 1 and 2 are more challenging to classify accurately. The model struggles to differentiate between these classes.
K-NN: The performance is low. The precision, the recall and F1-score are close to 0.5. Except for classes 0 and 3, the other classes have low precision, indicating significant classification difficulties.
Naives Bayes: The precision, the recall and F1-score are around 0.8.Class 0 is well-predicted but classes 1 and 2 are more challenging to classify accurately. The model has some difficulties differentiating between these classes.

2.6.3. The confusion matrix

SVM: Errors are minimal. All predictions for class 0 are accurate and predictions are nearly perfect for every class.
Decision tree classifier: The performance is good. For each predicted class, there are few errors. However, it is less accurate than SVM, with more errors occurring in classes 1, 2, and 3.

K-NN: The performance is low. It makes errors primarily between classes 1 and 2. Classes 0 and 3 have good predictions, but there are still significant issues with classes 1 and 2. Overall, it makes a lot of errors.
Naives Bayes: The performance is good. For each predicted class, there are few errors. It is less accurate than SVM but similar to the Decision Tree Classifier. There are more errors in classes 1, 2, and 3.

III. Optimization

3.1 Introduction

Hyperparameter optimisation aims to adjust models to maximise their predictive ability while minimising bias and variance. We have optimised three main models: **SVM**, **Decision Tree** and **KNN**, using techniques such as cross-validation and grid search.

3.2 Support Vector Machine (SVM)

Mathematical Description

The objective of SVM is to find the optimal hyperplane that separates classes in a higher-dimensional space. The optimization problem is defined as:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to: } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0,$$

where:

- **w** and **b** define the hyperplane.
- $y_i \in \{-1, 1\}$ is the class of sample i.
- ξi are slack variables to handle classification errors.

The regularized cost function is:

$$L(\mathbf{w}, b, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i,$$

where **C** controls the trade-off between maximizing the margin and minimizing classification errors.

Optimized Hyperparameters

1. **CCC**: Controls the importance of classification errors. A high value reduces training errors but risks overfitting.
2. **Kernel**: Transforms data into higher-dimensional space. We used the **RBF (Radial Basis Function)** kernel:

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right),$$

where γ determines the influence of nearby training points.

3. **γ**: Adjusts the complexity of the decision boundary. A low γ results in a smooth boundary, while a high γ increases sensitivity.

Results

- **Cross-validation**: The best accuracy (94%) was achieved with C=1, γ='scale', and the RBF kernel.
- **Impact**:
    - C influences bias-variance trade-off: a high C increases variance.
    - γ affects generalization through boundary flexibility.

3.3 Decision Tree

Mathematical Description

Decision trees recursively split the feature space to minimize an impurity measure I, such as:

$$I = \sum_{i=1}^{k} p_i(1 - p_i),$$

where pi is the proportion of samples in class i. Two criteria were used:

- **Gini Impurity**: Promotes homogeneity within splits.
- **Entropy**:

$$H = -\sum_{i=1}^{k} p_i \log(p_i),$$

which measures information gain.

Optimized Hyperparameters

1. **Maximum Depth** (max_depth): Limits tree depth to prevent overfitting.
2. **Minimum Samples per Leaf** (min_samples_leaf): Controls the granularity of splits.
3. **Criterion**: Determines the splitting measure (Gini or Entropy).

Results

- Best accuracy (88%) was obtained with max_depth=5 and the "entropy" criterion.
- **Impact**:
    - Excessive depth increases variance.
    - Larger leaf sizes reduce sensitivity but may miss complex patterns.

3.4 K-Nearest Neighbors (KNN)

Mathematical Description

KNN classifies samples based on the majority label of the kkk nearest neighbors, using a distance metric d(x,x′). We used the Euclidean distance:

$$d(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x_i')^2}.$$

Optimized Hyperparameters

1. **k**: Number of neighbors considered.
2. **Distance Metric**: Evaluated Euclidean and Manhattan distances.

Results

- Best accuracy (92%9) for k=21
- **Impact**:
    - Small k: High variance and sensitivity to noise.
    - Large k: Stability but risk of oversmoothing.

IV. Improvement

4.1 Analysis of Results

Despite optimization, certain limitations persisted:

- **SVM**: Excellent accuracy but computationally expensive for large datasets.
- **Decision Tree**: Overfitting at higher depths.
- **KNN**: Sensitive to feature dimensionality.

4.2 Improvement Techniques

A. Stacking

Stacking combines multiple models into a meta-model (e.g., logistic regression) to enhance robustness:

$$\hat{y} = g(f_1(x), f_2(x), \ldots, f_n(x)),$$

where g is the meta-model, and f_i are base models (e.g., SVM, KNN, Decision Tree).

- **Results**: Improved overall accuracy to 97%

B. Boosting with XGBoost

GBoost minimizes an additive cost function:

$$L = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{k=1}^{T} \Omega(f_k),$$

where ℓ measures prediction error, and Ω penalizes model complexity.

- **Results**: Accuracy improved to 96%

C. Feature Selection with RFE

Recursive Feature Elimination (RFE) iteratively removes the least important features to reduce dimensionality.

- **Results**: Reduced training complexity with a +1.5% accuracy gain.

D. Advanced Hyperparameter Tuning with RandomizedSearchCV

RandomizedSearchCV efficiently explores hyperparameter space by random sampling instead of exhaustive grid search.

- **Impact**: Improved tuning speed and model accuracy.

E. Voting System

The VotingClassifier combines predictions from multiple models using:

- **Hard Voting**: Predicts the majority class.
- **Soft Voting**: Averages predicted probabilities.
- **Results**: Achieved an accuracy of 97.5%

4.3 Summary of Model Performance

The results of the model evaluations highlight the following key insights:

1. **SVM**:
    - The SVM model stands out as the best-performing algorithm, achieving an accuracy of **97.25%**.
    - This demonstrates its ability to effectively capture the complex relationships between features and classes, particularly when using the RBF kernel.
2. **Decision Tree**:
    - The Decision Tree achieves a moderate accuracy of **84.75%**.
    - While it provides interpretable results and captures patterns in the data, it struggles with more complex relationships, partly due to its sensitivity to overfitting.
3. **K-Nearest Neighbors (KNN)**:
    - The KNN model delivers the lowest performance, with an accuracy of **57.50%**.
    - This can be attributed to its inability to handle datasets where classes are not well localized or where feature dimensionality poses challenges.
4. **VotingClassifier**:
    - The ensemble approach using VotingClassifier achieves an accuracy of **88.75%**.
    - Although it improves upon individual models like the Decision Tree and KNN, it does not surpass the SVM, suggesting that the latter dominates in terms of predictive power.
5. **Boosting with XGBoost**:
    - XGBoost achieves an accuracy of **96%**, slightly below SVM but with significant robustness against overfitting due to its iterative boosting approach.

So, the SVM model is the most effective for this problem, offering the highest accuracy and robust generalization capabilities. While ensemble methods and boosting enhance performance, they do not outperform SVM, particularly when fine-tuned with hyperparameter optimization.