

# Binary Classification on dataset Accelerometer Gyro Mobile Phone using Methods KNN, Naive Baiyes and Decision Tree

Romario Viegas F. Marcal - 1301225492

Ari Ramadhan - 1301224458

Edric Veda Adiyatma - 1301224245

## DATA DESCRIPTION

---

The dataset is titled "**Accelerometer+Gyro+Mobile Phone Dataset**" and contains data collected from an accelerometer and gyroscope on a mobile phone.

The dataset can be used for a variety of tasks, such as:

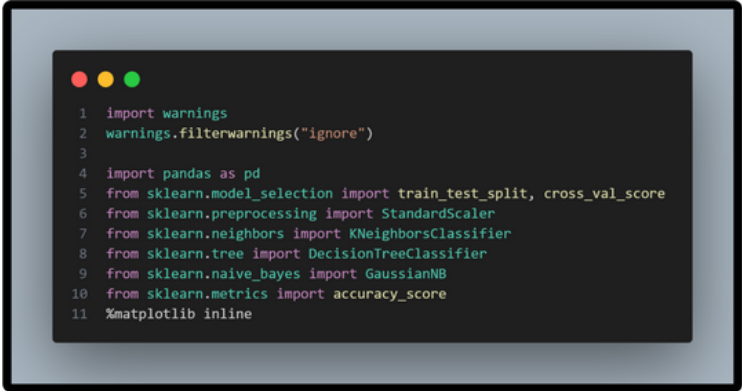
- **Activity recognition**: The data can be used to train models to recognize different types of physical activity, such as walking, running, and sitting.
- **Fall detection**: The data can be used to develop algorithms to detect falls, which could be helpful for elderly people or those with health conditions that make them more prone to falling.
- **Gesture recognition**: The data can be used to train models to recognize different gestures made with the phone, which could be used for a variety of applications, such as controlling games or interacting with other devices.

This dataset was donated to the UCI Machine Learning Repository on November 24, 2022. It contains data collected in 2022 at King Saud University in Riyadh. And this dataset are focusing on the activities recognized are standing and walking. The dataset contains 8 features: accelerometer X, accelerometer Y, accelerometer Z, gyroscope X, gyroscope Y, gyroscope Z, timestamp, and activity. There are no missing values in the dataset.

## LIBRARIES IMPORTED

---


**Importing Libraries:** The code imports necessary libraries for data manipulation (pandas), machine learning algorithms (scikit-learn), data preprocessing, model selection, and evaluation.



```
1 import warnings
2 warnings.filterwarnings("ignore")
3
4 import pandas as pd
5 from sklearn.model_selection import train_test_split, cross_val_score
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.neighbors import KNeighborsClassifier
8 from sklearn.tree import DecisionTreeClassifier
9 from sklearn.naive_bayes import GaussianNB
10 from sklearn.metrics import accuracy_score
11 %matplotlib inline
```

## LOADING DATA

---



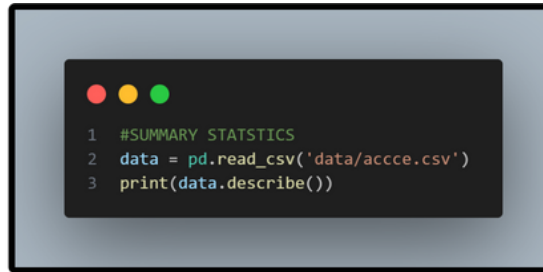
```
1 #LOAD THE DATA
2
3 data = pd.read_csv('data/accelerometer.csv')
```

**Loading Data:** Our dataset, which contains accelerometer and gyroscope data collected from mobile phones for human activity recognition, is loaded into a pandas Data Frame named data.

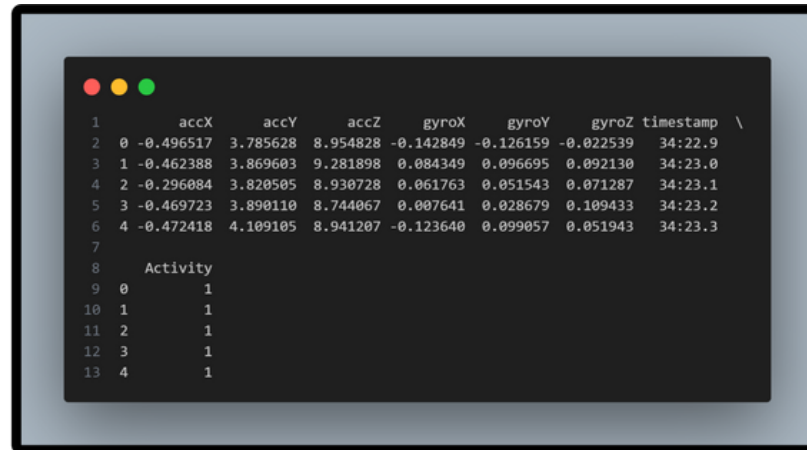
## SUMMARY OF THE DATA

---

The dataset contains 8 features as it mention in the documentation, which is contains variables : accelerometer X, accelerometer Y, accelerometer Z, gyroscope X, gyroscope Y, gyroscope Z, timestamp, and activity.



```
1 #SUMMARY STATISTICS
2 data = pd.read_csv('data/accce.csv')
3 print(data.describe())
```



```
1      accX      accY      accZ      gyroX      gyroY      gyroZ timestamp \
2 0 -0.496517  3.785628  8.954828 -0.142849 -0.126159 -0.022539  34:22.9
3 1 -0.462388  3.869603  9.281898  0.084349  0.096695  0.092130  34:23.0
4 2 -0.296084  3.820505  8.930728  0.061763  0.051543  0.071287  34:23.1
5 3 -0.469723  3.890110  8.744067  0.007641  0.028679  0.109433  34:23.2
6 4 -0.472418  4.109105  8.941207 -0.123640  0.099057  0.051943  34:23.3
7
8      Activity
9 0          1
10 1          1
11 2          1
12 3          1
13 4          1
```

## FEATURES AND TARGET DATA

The **timestamp** variable represents the time at which each reading was taken. This can be useful in some cases, especially if the sequence or timing of the readings is important for your problem. However, the models you're currently using (KNN, Decision Tree, and Gaussian Naive Bayes) don't take the sequence of the data into account, so the timestamp variable wouldn't be directly useful for them.



```
1
2 # Drop the timestamp column as it's not needed for modeling
3 data = data.drop(columns=['timestamp'])
```

**Features** (accelerometer and gyroscope readings) are separated from the target variable (**Activity**: standing or walking).

Information about the features and target variable, such as data types, head (top rows), and summary statistics, is printed to understand the structure and distribution of the data.

# PERFORM EDA

Perform **Explanatory data Analysis**, we are trying to see the sense of the data, which is in this dataset we are trying to see and analyze some several information :

- **accX, accY, accZ**: These columns represent the readings from the accelerometer sensor in the X, Y, and Z directions respectively. Accelerometers measure the acceleration that the smartphone is experiencing relative to freefall.
- **gyroX, gyroY, gyroZ**: These columns represent the readings from the gyroscope sensor in the X, Y, and Z directions respectively. Gyroscopes measure the rate or velocity at which the device is rotating around a specific axis.
- **Activity**: This is likely the target variable, representing the activity that was being performed when the sensor readings were taken. The activities are encoded as numbers. Here in the dataset '1' represent 'standing (stop)', '0' represent 'walking(moving)'. The exact mapping of numbers to activities would be defined elsewhere in your project or in the dataset documentation.

Each row in the dataset represents a single observation, with the sensor readings and the corresponding activity. This kind of data is often used in activity recognition tasks, where the goal is to predict the activity based on the sensor readings.

```
1 #READ DATASET
2 print(data.head())
3
```

```
1 #DISPLAY THE CONCISE SUMMARY OF THE DATAFRAME
2 print(data.info())
```

	accX	accY	accZ	gyroX	gyroY	gyroZ	Activity
0	-0.496517	3.785628	8.954828	-0.142849	-0.126159	-0.022539	1
1	-0.462388	3.869603	9.281898	0.084349	0.096695	0.092130	1
2	-0.296084	3.820505	8.930728	0.061763	0.051543	0.071287	1
3	-0.469723	3.890110	8.744067	0.007641	0.028679	0.109433	1
4	-0.472418	4.109105	8.941207	-0.123640	0.099057	0.051943	1

```

1 RangeIndex: 31991 entries, 0 to 31990
2 Data columns (total 6 columns):
3 #   Column   Non-Null Count  Dtype
4 ---  ---
5 0    accX     31991 non-null  float64
6 1    accY     31991 non-null  float64
7 2    accZ     31991 non-null  float64
8 3    gyroX    31991 non-null  float64
9 4    gyroY    31991 non-null  float64
10 5    gyroZ    31991 non-null  float64
11 dtypes: float64(6)
12 memory usage: 1.5 MB
13
14
15 <class 'pandas.core.series.Series'>
16 RangeIndex: 31991 entries, 0 to 31990
17 Series name: Activity
18 Non-Null Count  Dtype
19 -----
20 31991 non-null  int64
21 dtypes: int64(1)
22 memory usage: 250.1 KB

```

The 'timestamp' column is dropped as it's not relevant for the classification task. Features (accelerometer and gyroscope readings) are separated from the target variable (activity: standing or walking). Information about the features and target variable, such as data types, head (top rows), and summary statistics, is printed to understand the structure and distribution of the data.

The code **print(data.info())**: This line is displaying a concise summary of your DataFrame. The info() function provides essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.

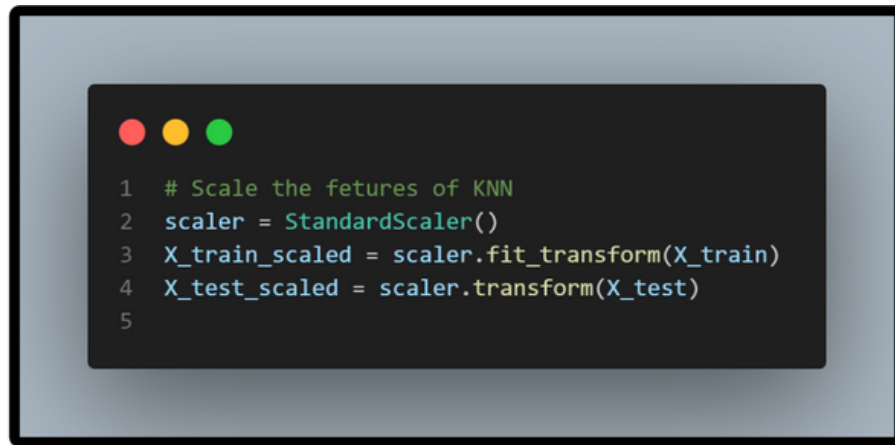
# DATA PRE-PROCESSING

---

**Train-Test Split:** The data is split into training and testing sets with a 70-30 ratio. This allows for training the models on one subset and evaluating their performance on another independent subset.



```
1 # Split the data into training and testing sets
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4)
```



```
1 # Scale the fetures of KNN
2 scaler = StandardScaler()
3 X_train_scaled = scaler.fit_transform(X_train)
4 X_test_scaled = scaler.transform(X_test)
5
```

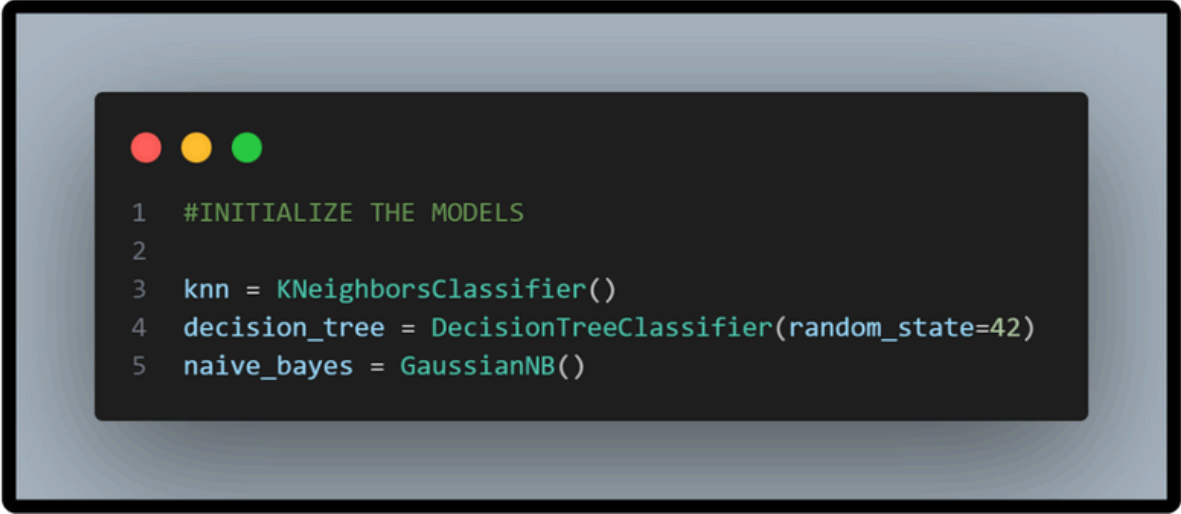
**Feature Scaling:** Since K-Nearest Neighbors (KNN) relies on distance metrics, the features are scaled using StandardScaler to ensure they have the same scale. This step is important for achieving optimal performance from the KNN algorithm.



## MODEL INITIALIZATION

---

Three classifiers are initialized: K-Nearest Neighbors (KNN), Decision Tree, and Naive Bayes. These classifiers will be trained and evaluated using the data.



```
1  #INITIALIZE THE MODELS
2
3  knn = KNeighborsClassifier()
4  decision_tree = DecisionTreeClassifier(random_state=42)
5  naive_bayes = GaussianNB()
```

# TRAINING AND EVALUATION

**KNN:** The KNN classifier is trained on the scaled training data and evaluated on the scaled testing data. Its accuracy score indicates how well it predicts the activity classes (standing or walking) based on accelerometer and gyroscope readings.

**Decision Tree:** The Decision Tree classifier is trained on the original training data (without scaling) and evaluated on the original testing data. Its accuracy score reflects its performance in predicting activity classes.

**Naive Bayes:** The Naive Bayes classifier is trained on the original training data and evaluated on the original testing data. Its accuracy score indicates how well it predicts activity classes based on the provided data.

```
1 #TRAIN THE KNN AND EVALUATE THE KNN
2 knn.fit(X_train_scaled, y_train)
3 knn_predictions = knn.predict(X_test_scaled)
4 knn_accuracy = accuracy_score(y_test, knn_predictions)
```

```
1 #TRAIN AND EVALUATE THE DECISION TREE
2 decision_tree.fit(X_train, y_train)
3 decision_tree_predictions = decision_tree.predict(X_test)
4 decision_tree_accuracy = accuracy_score(y_test, decision_tree_predictions)
```

```
1 #TRAIN AND EVALUATE THE NAIVE BAYES
2 naive_bayes.fit(X_train, y_train)
3 naive_bayes_predictions = naive_bayes.predict(X_test)
4 naive_bayes_accuracy = accuracy_score(y_test, naive_bayes_predictions)
```

## TESTING THE DATA

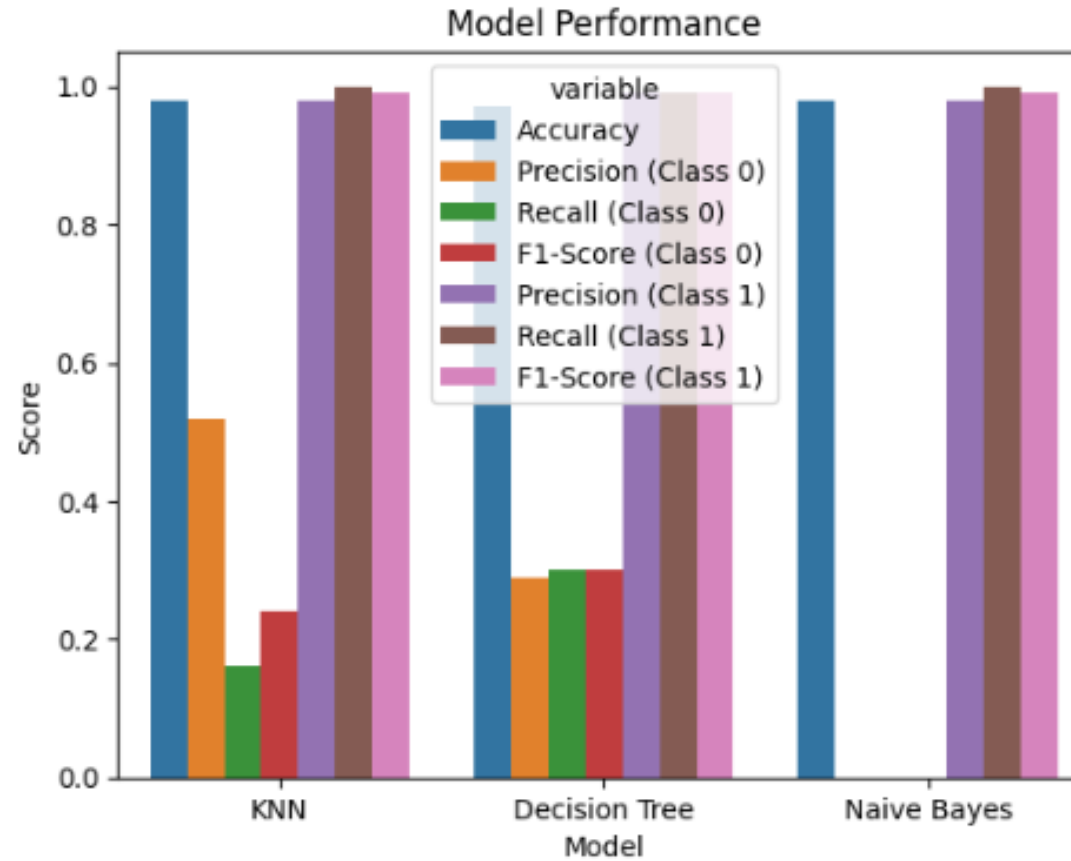
**Testing Metrics:** The overall scores of all three models (KNN, Decision Tree, Naive Bayes) are printed. These scores provide insights into how well each model performs in classifying activities (standing or walking) based on accelerometer and gyroscope data from mobile phones.

```
1 # Model Testing and Evaluation
2 # Show the performance
3 # Write your code here
4
5 from sklearn.metrics import classification_report
6
7 # Print classification report for KNN
8 print("KNN:")
9 print(classification_report(y_test, knn_predictions))
10
11 # Print classification report for Decision Tree
12 print("Decision Tree:")
13 print(classification_report(y_test, decision_tree_predictions))
14
15 # Print classification report for Naive Bayes
16 print("Naive Bayes:")
17 print(classification_report(y_test, naive_bayes_predictions))
```

Model	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-Score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-Score (Class 1)
KNN	0.98	0.52	0.16	0.24	0.98	1.00	0.99
Decision Tree	0.97	0.29	0.30	0.30	0.99	0.99	0.99
Naive Bayes	0.98	0.00	0.00	0.00	0.98	1.00	0.99

## VISUALIZATION

Using **Barplot** to see the details comparison between the accuracy, precision, recall, and F1-score of these 3 methods.



## ANALYSIS

Model	Accuracy	Precision (Class 0)	Recall (Class 0)	F1-Score (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-Score (Class 1)
KNN	0.98	0.52	0.16	0.24	0.98	1.00	0.99
Decision Tree	0.97	0.29	0.30	0.30	0.99	0.99	0.99
Naive Bayes	0.98	0.00	0.00	0.00	0.98	1.00	0.99

From the results, it's clear that all three models perform well in identifying 'walking' activities (Class 1), with high precision, recall, and F1-scores. However, they struggle with identifying 'standing' activities (Class 0). The **KNN model** performs the best for 'standing' activities, but its performance is still relatively poor.

The **Naive Bayes model** fails to correctly identify any 'standing' activities, as indicated by the precision, recall, and F1-score of 0.00 for Class 0. This suggests that the Naive Bayes model is not suitable for this problem with the current dataset.

The class imbalance in the dataset is likely contributing to the models' poor performance for 'standing' activities. Future work could involve resampling the dataset to balance the classes, generating synthetic samples, or exploring different models or techniques that are better suited to imbalanced datasets.

## **CONCLUSION**

---

In conclusion, all three models perform well for walking activities but struggle with standing activities. This is likely due to the class imbalance in the dataset. To improve the models' performance for standing activities, you could try resampling the dataset to balance the classes, generating synthetic samples, or using different models or techniques that are better suited to imbalanced datasets.



apkh ad yg

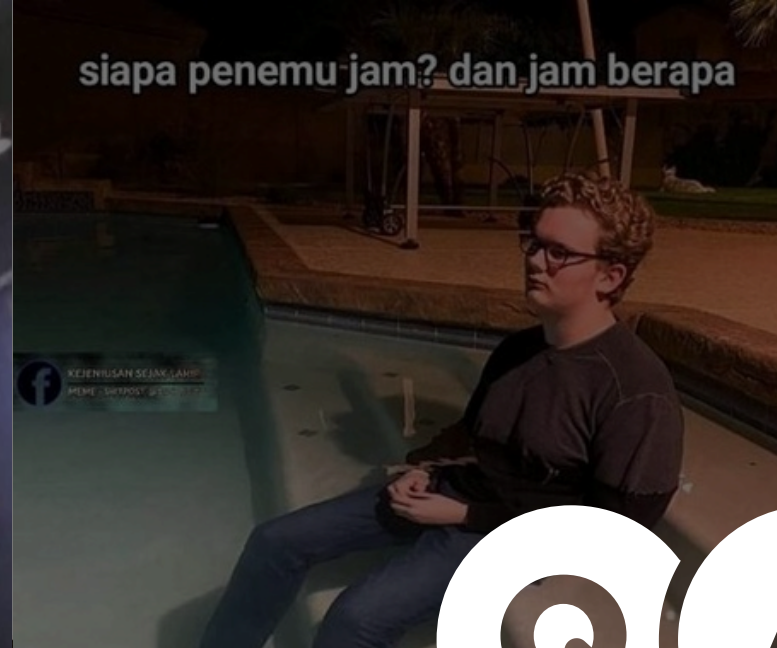
mw brtnya??

siapa penemu jam? dan jam berapa

Saat TV pertama kali dibuat

Presentasi selesai

Ada pertanyaan



**BANYAK NANYA SAAT TEMAN  
PRESENTASI**

**RUMAH-NYA MELEDAK**

makeameme.org

