**VIETNAM NATIONAL UNIVERSITY – HCM**

**INTERNATIONAL UNIVERSITY**

**DEPARTMENT OF COMPUTER SICENCE & ENGINEER**

SEMESTER 1 (2024-2025)

**IT160IU**

**DATA MINING PROJECT**

**Course by Dr. Nguyen Thi Thanh Sang & MSc. Nguyen Quang Phu**

**Topic name: Obesity Risk.**

*December 2024*

**Group A:**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Full name - Student name** | **Student ID** | **Contribution (%)** |
| 1 | PHAM VU TUYET ANH | ITDSIU21073 | 25 |
| 2 | DINH THI THANH HANG | ITDSIU21083 | 25 |
| 3 | LE THI TUYET MY | ITDSIU21005 | 25 |
| 4 | DINH VU NGOC LINH | ITDSIU21095 | 25 |

**Table of Contents**

[I. INTRODUCTION 4](#_Toc184034534)

[1 Background 4](#_Toc184034535)

[2 Goals & Objectives 4](#_Toc184034536)

[3 Understanding the story of the Dataset Attributes 4](#_Toc184034537)

[4 Methodologies 4](#_Toc184034538)

[II. THEORETICAL BASIS 6](#_Toc184034539)

[5 Data Mining Concepts 6](#_Toc184034540)

[6 Single model 6](#_Toc184034541)

[6.1 Zero Rule 6](#_Toc184034542)

[6.2 Instance-Based K-Nearest 7](#_Toc184034543)

[6.3 Sequential Minimal Optimization (SMO): 7](#_Toc184034544)

[6.4 Naive Bayes: 7](#_Toc184034545)

[6.5 Decision Tree (C4.5 Algorithm): 7](#_Toc184034546)

[7 Ensemble model 7](#_Toc184034547)

[III. RELATIVE WORKS. 9](#_Toc184034548)

[1 Data Pre-Processing 9](#_Toc184034549)

[1.1 Raw data overview 9](#_Toc184034550)

[1.2 Handling missing value: 10](#_Toc184034551)

[1.3 Checking duplicates 11](#_Toc184034552)

[1.4 Addressing and removing outliers 12](#_Toc184034553)

[1.5 Data Transformation 14](#_Toc184034554)

[1.6 Drop, move and rename columns 16](#_Toc184034555)

[2 Single Classification Algorithm 18](#_Toc184034556)

[2.1 Model Selection 18](#_Toc184034557)

[2.2 Result 23](#_Toc184034558)

[3 Improvement of Results. 24](#_Toc184034559)

[3.1 Methodology 24](#_Toc184034560)

[3.2 Comparison of the results 30](#_Toc184034561)

[4 Model Evaluation 31](#_Toc184034562)

[4.1 Performance Metrics 31](#_Toc184034563)

[4.2 Analysis of Results 31](#_Toc184034564)

[IV. CONCLUSION 32](#_Toc184034565)

[1 Summary 32](#_Toc184034566)

[2 What we learn after project 32](#_Toc184034567)

[3 Completed work & Future work. 32](#_Toc184034568)

[V. REFERENCES 34](#_Toc184034569)

**TABLE OF FIGURES**

[Figure II‑1 KDD presentation 9](#_Toc184034570)

[Figure III‑1 Summary of numerical attributes in datasets 13](#_Toc184034571)

[Figure III‑2 The code and result of checking missing value 14](#_Toc184034572)

[Figure III‑3 The code and result of checking duplicate 14](#_Toc184034573)

[Figure III‑4 The code to select numerical column 15](#_Toc184034574)

[Figure III‑5 The function to remove outliers 15](#_Toc184034575)

[Figure III‑6 The box plot before remove outliers 16](#_Toc184034576)

[Figure III‑7 The box plot after remove outliers 16](#_Toc184034577)

[Figure III‑8 Label encoding for ordinal attributes 17](#_Toc184034578)

[Figure III‑9 One-hot encoding for nominal attributes 17](#_Toc184034579)

[Figure III‑10 The code of conversion 17](#_Toc184034580)

[Figure III‑11 The code of normalization 18](#_Toc184034581)

[Figure III‑12 The histograms before normalization 18](#_Toc184034582)

[Figure III‑13 The histograms after normalization 18](#_Toc184034583)

[Figure III‑14 The code to drop columns 19](#_Toc184034584)

[Figure III‑15 The code to move and rename columns 19](#_Toc184034585)

[Figure III‑16 The code to rename values 19](#_Toc184034586)

[Figure III‑17 The first ten rows of cleaned dataset 20](#_Toc184034587)

[Figure III‑18 The code to split data 20](#_Toc184034588)

[Figure III‑19 Convert datasets to ARFF format 21](#_Toc184034589)

[Figure III‑20 Constructor of J48 algorithm 22](#_Toc184034590)

[Figure III‑21 Building model method of J48 algorithm 22](#_Toc184034591)

[Figure III‑22 Evaluation method of J48 algorithm 22](#_Toc184034592)

[Figure III‑23 Prediction method of J48 algorithm 23](#_Toc184034593)

[Figure III‑24 Display method of J48 algorithm 23](#_Toc184034594)

[Figure III‑25 Constructor of SMO algorithm 23](#_Toc184034595)

[Figure III‑26 Building model method of SMO algorithm 23](#_Toc184034596)

[Figure III‑27 Evaluation method of SMO algorithm 24](#_Toc184034597)

[Figure III‑28 Prediction method of SMO algorithm 24](#_Toc184034598)

[Figure III‑29 Display method of SMO algorithm 24](#_Toc184034599)

[Figure III‑30 Constructors of Naive Bayes algorithm 25](#_Toc184034600)

[Figure III‑31 Building model method of Naive Bayes algorithm 25](#_Toc184034601)

[Figure III‑32 Evaluation method of Naive Bayes algorithm 25](#_Toc184034602)

[Figure III‑33 Prediction method of Naive Bayes algorithm 26](#_Toc184034603)

[Figure III‑34 Display method of Naive Bayes algorithm 26](#_Toc184034604)

[Figure III‑35 Codes to launch the J48 (Decision Tree) algorithm 26](#_Toc184034605)

[Figure III‑36 Codes to launch the SMO (SVM) algorithm 27](#_Toc184034606)

[Figure III‑37 Codes to launch the Naive Bayes algorithm 27](#_Toc184034607)

[Figure III‑38 Flow preprocessing 27](#_Toc184034608)

[Figure III‑39 Flow Post-processing 28](#_Toc184034609)

[Figure III‑40 Feature importance by Random forest 30](#_Toc184034610)

[Figure III‑41 Function to build RandomForest model 30](#_Toc184034611)

[Figure III‑42 Predict function of model RandomForest 30](#_Toc184034612)

[Figure III‑43 Evaluation function of model RandomFores 31](#_Toc184034613)

[Figure III‑44 Implement RandomForest model 31](#_Toc184034614)

[Figure III‑45 All functions in Neural Network 31](#_Toc184034615)

[Figure III‑46 Implementing Neural Network 32](#_Toc184034616)

[Figure III‑47 All functions in Gradient Boosting 32](#_Toc184034617)

[Figure III‑48 Implementing Gradient Boosting model 32](#_Toc184034618)

[Figure III‑49 Function to build Voting classifier 33](#_Toc184034619)

[Figure III‑50 Function to evaluate and predict in Voting Classifier 33](#_Toc184034620)

[Figure III‑51 Implement Voting Classifier 33](#_Toc184034621)

**TABLES**

[Table III.1 Accuracy comparison between algorithms 19](#_Toc184117367)

[Table III.2 Comparison of multiple model results 25](#_Toc184117368)

[Table III.3 Comparison of advance model results 32](#_Toc184117369)

[Table III.4 Comparison of J48 and Voting Classifier On Some Performance Metrics 32](#_Toc184117370)

[Table III.5 Class Details of Voting Classifier 32](#_Toc184117371)

[Table III.6 Class Details Of` J48 33](#_Toc184117372)

[Table III.7 Comparison details 36](#_Toc184117373)

**Task Responsibilities**

|  |  |  |
| --- | --- | --- |
| **No** | **Task** | **Responsibility** |
| 1 | Analyze the issue | All of members |
| 2 | Find the dataset | Tuyet Anh |
| 3 | Preprocessing data | Ngoc Linh |
| 4 | Build classification model | Tuyet My |
| 4 | Improvement result | Tuyet Anh |
| 5 | Evaluation performance | Thanh Hang |
| 6 | Write Report | All of members |
| 7 | Preparing file presentation | All of members |
| 8 | Preview + Demo | All of members |

# INTRODUCTION

## Background

Obesity is a growing global health issue influenced by genetic, behavioral, environmental, and lifestyle factors. Understanding these factors is essential for effective prevention and intervention strategies. This dataset provides detailed insights into obesity by examining demographics, physical traits, lifestyle habits, and behaviors. It facilitates research into the correlations and patterns contributing to obesity. By analyzing this data, researchers can uncover how lifestyle choices and environmental factors affect obesity and develop targeted recommendations for managing its risks.

## Goals & Objectives

* Goal: Analyze the dataset and apply machine learning to predict obesity risk levels. This addresses the challenges of identifying the multifactorial causes of obesity, such as genetic, behavioral, environmental, and lifestyle factors, as outlined in the Background part. The goal is to uncover patterns and develop predictive models to support effective prevention and intervention strategies.
* Objective: To build a data mining framework incorporating a classification/prediction model and a sequence mining algorithm.

## Understanding the story of the Dataset Attributes

* Dataset Used: The dataset utilized in this study is the Obesity Level Dataset from Kaggle. This dataset provides a comprehensive overview of factors influencing obesity, including demographic, physical, behavioral, and lifestyle attributes. It is specifically curated for research in health and lifestyle domains, offering key data points to analyze obesity risk levels.
* This dataset provides comprehensive information on individuals, encompassing key attributes such as gender, age, height, weight, family history with overweight, dietary habits, physical activity, transportation mode, and the corresponding obesity level. The dataset is meticulously curated for research and analysis in the domain of health and lifestyle studies.

## Methodologies

The methodology for this project is divided into four key parts: Data Preprocessing, Model Implementation, Improvement of result, and Model Evaluation.

Part 1: Preprocessing

* In this section, the raw data is cleaned and prepared for analysis, and prediction by removing outliers, transforming the types of data.

Part 2: Implement a Classification/Prediction Algorithm

* In this step, the cleaned data is then splitted into training, and testing sets, and then fed into a classification or prediction algorithms by Weka Library. The model is trained on the training data and evaluated by testing set.

Part 3: Improvement of result

* This section will provide methods to improve efficiency, by implementing another algorithm, or trying to clean the data with a different method.

Part 4: Model Evaluation

* After training, and evaluating models, we will choose the 2 best models and evaluate these two models on a 10-fold cross validation and compare the results.

# **THEORETICAL BASIS**

## Data Mining Concepts

* Data mining is the practice of examining large databases in order to generate new information. The process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cut costs, or both.
* Data mining is defined as the process of discovering patterns in data and the process must be automatic or (more usually) semiautomatic. The patterns discovered must be meaningful in that they lead to some advantage, usually an economic one. The data is invariably presented in substantial quantities.

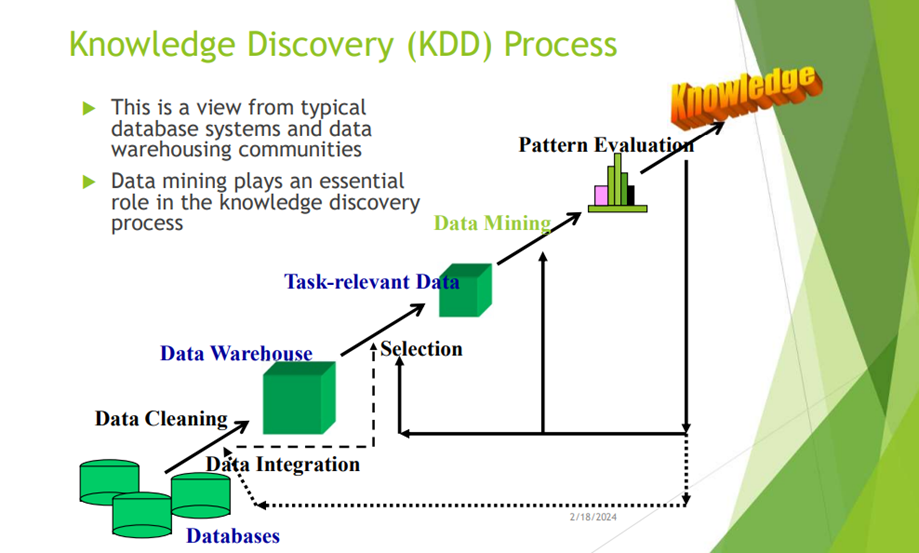


Figure ‑ KDD presentation

## Single model

### Zero Rule:

* Definition: The algorithm predicts the most frequent class label (mode) or the mean (for regression tasks) for all instances, without any regard to the input features.
* Purpose: We use this model as the baseline model for further comparison with more complex models.

### Instance-Based K-Nearest (IBk):

* Definition: IBk is an implementation of the k-Nearest Neighbors (k-NN) algorithm. It classifies data based on the majority class of its nearest neighbors, where "k" is a parameter representing the number of neighbors considered.
* Purpose: IBk is significant in cases where the relationship between features and labels is complex and non-linear. Hence, we use this model to consider the relationship between attributes based on the accuracy it provides.

### Sequential Minimal Optimization (SMO):

* Definition: SMO is an algorithm for training Support Vector Machines (SVM). It breaks the complex optimization problem of training an SVM into smaller subproblems that are easier to solve iteratively.
* Purpose: SMO is useful for classification problems, especially when dealing with high-dimensional data. In the performance of IBk, we know that our data is high-dimensional, so we choose this model for consideration.

### Naive Bayes:

* Definition: Naive Bayes is a probabilistic classifier based on Bayes' theorem, assuming that features are independent. It calculates the likelihood of a class given the observed feature values and assigns the class with the highest probability.
* Purpose: Since this model is simple, fast, and achieves great performance even on a small dataset, we also take this model into consideration.

### Decision Tree (C4.5 Algorithm):

* Definition: C4.5 is a classification algorithm used to build decision trees. It works by splitting the dataset into subsets based on features that provide the most information about the target variable. At each step, the algorithm selects the feature that best separates the data into distinct classes, forming a tree-like structure of decisions.
* Purpose: C4.5 is powerful for numeric datasets as it efficiently handles continuous data, produces interpretable models, and is robust to missing values.

## Ensemble model

An **ensemble model** combines predictions from multiple individual models to improve overall performance. The idea is that by aggregating diverse models, an ensemble can reduce errors caused by bias, variance, or noise in individual models, often leading to better generalization. The models should make uncorrelated errors for the ensemble to work effectively. This model combines output using techniques in this project like:

* **Bagging techniques (Random Forest)**. It builds multiple decision trees during training, each trained on a different bootstrap sample (random subset with replacement) of the dataset, so, it called “Bootstrap Aggregating”. When making classification, it uses majority voting across the trees. This is a better choice than a single model when it reduces overfitting, handles high dimensional data, and produces feature importances. Random Forest's ability to combine the strengths of multiple trees while mitigating overfitting makes it a highly effective and versatile model.
* **Boosting techniques (Gradient Boosting model)**: Gradient Boosting is a popular boosting technique that uses decision trees as weak learners. It minimizes errors by optimizing a loss function through gradient descent. Each new tree is trained to predict the residuals (errors) of the previous trees, gradually improving the model. This is a better choice than a single model when it implements sequential training, loss function optimization, and high accuracy. However, it can be computationally intensive and prone to overfitting without proper tuning.
* **Voting Classifier** is an ensemble technique that combines predictions from multiple models (called base models) to make a final prediction. It is primarily used for classification tasks and works by aggregating the outputs of different models. There are two types of voting, hard voting and soft voting. Voting classifiers are effective when individual models perform reasonably well, and their errors are uncorrelated. It improves prediction accuracy by aggregating diverse perspectives.This type of ensemble produces diversity when models used in the ensemble can be of different types (e.g., decision trees, SVMs, logistic regression), enhancing performance by leveraging their strengths. It also is simplicity and flexibility.

# RELATIVE WORKS.

## Data Pre-Processing

This part aims to cleanand prepare the raw dataset for analysis and modeling. This part includes the following steps:

### Raw data overview

* The dataset was collected from Kaggle, dataset source: [Link](https://www.kaggle.com/datasets/jpkochar/obesity-risk-dataset/data)
* The dataset contains 18 columns in total. The number of instances is20757.
* Dataset’s columns meaning:
* **id:** Numbers of instances.
* **Gender**: Biological sex of the individual.
* **Age**: Age of the individual in years.
* **Height**: Height of the individual in meters.
* **Weight**: Weight of the individual in kilograms.
* **family\_history\_with\_overweight**: Indicates if there is a family history of being overweight.
* **FAVC**: Indicates frequent consumption of high-caloric food.
* **FCVC**: Frequency of vegetable consumption.
* **NCP**: Number of main meals consumed daily.
* **CAEC**: Frequency of consuming food between meals.
* **SMOKE**: Indicates smoking status.
* **CH2O**: Daily water consumption in liters.
* **SCC**: Indicates consumption of caloric beverages.
* **FAF**: Frequency of physical activity.
* **TUE**: Time spent daily using technological devices.
* **CALC**: Frequency of alcohol consumption.
* **MTRANS**: Usual mode of transportation.
* **0be1dad**: Obesity level classification.
* Numerical attributes: Age, Height, Weight, family\_history\_with\_overweight, FAVC (Frequent consumption of high-caloric food), FCVC (Frequency of consumption of vegetables), NCP (Number of main meals), CH2O (Daily water consumption), FAF (Physical activity frequency), TUE (Time spent using technological devices), SMOKE, SCC (Caloric beverages consumption).
* Categorical attributes: Gender, CAEC (Consumption of food between meals), CALC (Consumption of alcohol), MTRANS (Mode of transportation), 0be1dad (Target variable representing obesity level).
* 16 input features: Gender, Age, Height, Weigh, family\_history\_with\_overweight, FAVC, FCVC, NCP, CAEC, SMOKE, CH2O, SCC, FAF, TUE, CALC, MTRANS.
* 1 target variable:0be1dad (Obesity level). Since the target variable 0be1dad (Obesity Level) is categorical, this is a classification problem, serving as the basis for selecting appropriate machine learning models in subsequent steps.
* Key characteristics:Includes a mix of categorical and numerical data. Captures both personal attributes and lifestyle habits. The target variable categorizes individuals based on obesity levels.
* Summary table:

A screenshot of a login

Description automatically generated

Figure ‑ Summary of numerical attributes in datasets

### Handling missing value:

We check that there are any rows containing missing value. Use df.isnull().sum() for checking missing value. As the result shown in *Figure 2.2.1,* there is no missing value in the dataset.

A screenshot of a computer code

Description automatically generated

Figure ‑ The code and result of checking missing value

### Checking duplicates

We use df.duplicated().sum() to check any duplicates in dataset. As the result shown in *Figure 2.2.2*, there are no duplicates in datasets.

A close-up of a white background

Description automatically generated

Figure ‑ The code and result of checking duplicate

### Addressing and removing outliers

Outliers are detected by using the IQR, which is calculated as the difference between the third quartile Q3 (upper bound) and the first quartile Q1 (lower bound) of a dataset. Any data points below the lower bound or above the upper bound are considered outliers.

* Select numerical columns: numerical features are identified using the select\_dtypes function from pandas.

A close-up of a logo

Description automatically generated

Figure ‑ The code to select numerical column

* Remove outliers:applies the IQR method to filter out rows containing outliers

A screen shot of a computer code

Description automatically generated

Figure ‑ The function to remove outliers

* **Results:** we use box plot to compare dataset’s numerical columns before remove outliers and after remove outliers.

A collage of a diagram

Description automatically generated

Figure ‑ The box plot before remove outliers

A collage of blue squares

Description automatically generated

Figure ‑ The box plot after remove outliers

### Data Transformation

1. **Encoding categorical variables**

* CAEC and CALC is ordinal attributes, then we use label encoding.

A computer code with text

Description automatically generated with medium confidence

Figure ‑ Label encoding for ordinal attributes

* MTRANS and Gender is nominal attributes, then we use one-hot encoding.

A screenshot of a computer code

Description automatically generated

Figure ‑ One-hot encoding for nominal attributes

* After use one-hot encoding, we convert all True/False values into numeric values to build model.

A screen shot of a computer code

Description automatically generated

Figure ‑ The code of conversion

1. **Normalization:**

The dataset has features with different normalizations like "Age" (ranging from 20 to 60), the model may give more importance to features with larger numerical values. Normalization transforms the features so that they are on a similar normalize, making sure no single feature dominates the others due to its larger normalize.

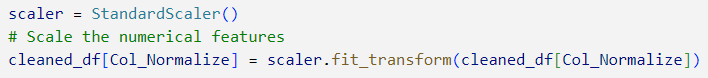


Figure ‑ The code of normalization

* We use histograms to compare dataset’s numerical features before normalization and after normalization.

A graph of a city

Description automatically generated with medium confidence

Figure ‑ The histograms before normalization

A graph of a tall building

Description automatically generated with medium confidence

Figure ‑ The histograms after normalization

### Drop, move and rename columns

Dropping columns:

* Original categorical columns after encoding: CAEC, CALC.
* Unuseful columns: id.
* Constant colum: family\_history\_with\_overweight, FAVC, NCP, SMOKE, SCC.

à Reasons for dropping constant column: A constant column will have the same value for every observation. This means the model cannot use it to distinguish between different instances in the dataset. Moreover, constant columns are effectively redundant. Their presence might lead to overfitting or unnecessarily increase the dimensionality of the model. Constant columns generally improve accuracy by making your model more efficient, reducing overfitting, and focusing on more relevant features.



Figure ‑ The code to drop columns

Move columns and rename columns:

* Move the '0be1dad' (target) column to the last position and rename the column to ‘class’.

A screen shot of a computer code

Description automatically generated

Figure ‑ The code to move and rename columns

* Rename values of column ‘class’ for shorter name:

A screenshot of a computer code

Description automatically generated

Figure ‑ The code to rename values

**Output:** The final cleaned dataset:

A screenshot of a computer

Description automatically generated

Figure ‑ The first ten rows of cleaned dataset

1. **Implementation Process**

Detail the steps to convert data to ARFF format and integrate Weka into the program. Mention any challenges faced during implementation.

* Step 1: Splitting data into training data and testing data using the train\_test\_split function from scikit-learn.
* **validation\_data**: 20% of the cleaned dataset, used to validate model performance and avoid overfitting.
* **Train\_test\_data**: 80% of the cleaned dataset, used to train & test machine learning models.

A screenshot of a computer

Description automatically generated

Figure ‑ The code to split data

* Step 2: Create a function to convert to ARFF format with 3 main features:
* Relation Name: Each ARFF file begins with a @relation tag to define the dataset name.
* Attributes: The @attribute tag defines the dataset's attributes (columns)
* Data: The @data section contains the actual data values, formatted as comma-separated strings.
* Stores the training dataset and validation dataset to train\_data.arff and validation\_data.arff in ARFF format, respectively.

A screen shot of a computer code

Description automatically generated

Figure ‑ Convert datasets to ARFF format

## Single Classification Algorithm

This part aims to implement a classification or prediction model using the Weka library.

### Model Selection

* From the preprocessed data we got from the previous task, we put it in the Weka and ran through multiple algorithms to find the best one. Thus, we got these results:

|  |  |
| --- | --- |
| Algorithm | Accuracy |
| ZeroR | 39.29 % |
| J48 | 91.809 % |
| SMO (SVM) | 85.7052 % |
| NaiveBayes | 86.7805 % |
| IBk (kNN) | 80.32 % |

Table . Accuracy comparison between algorithms

* From the table, we choose the J48 (Decision Tree) as the best algorithm to work with. However, to have more options to evaluate the performance, we still decided to build two more algorithms that are SMO (Support Vector Machine) and Naive Bayes.
* We build the models in Java on the NetBeans platform:
* **J48 algorithm:** The model consists of one constructor, one display method, some methods including building the model, evaluating model’s performance, and predicting the class label.

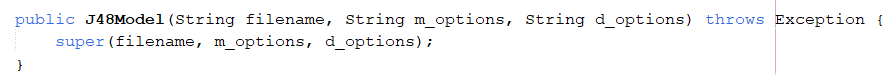


Figure ‑ Constructor of J48 algorithm

A screenshot of a computer code

Description automatically generated

Figure ‑ Building model method of J48 algorithm

A screen shot of a computer code

Description automatically generated

Figure ‑ Evaluation method of J48 algorithm

A screen shot of a computer code

Description automatically generated

Figure ‑ Prediction method of J48 algorithm

A black and pink text

Description automatically generated

Figure ‑ Display method of J48 algorithm

* **SMO algorithm**: The model consists of one constructor, one display method, some methods including building the model, evaluating model’s performance, and predicting the class label.

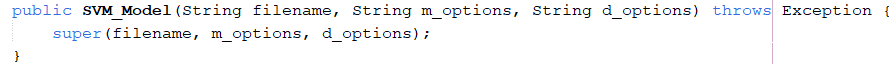


Figure ‑ Constructor of SMO algorithm

A computer code with text

Description automatically generated

Figure ‑ Building model method of SMO algorithm

A computer screen shot of a code

Description automatically generated

Figure ‑ Evaluation method of SMO algorithm

A screenshot of a computer code

Description automatically generated

Figure ‑ Prediction method of SMO algorithm

A close up of a text

Description automatically generated

Figure ‑ Display method of SMO algorithm

* **Naive Bayes algorithm:** The model consists of two constructors, one display method, some methods including building the model, evaluating model’s performance, and predicting the class label.

A computer code on a white background

Description automatically generated

Figure ‑ Constructors of Naive Bayes algorithm

A screen shot of a computer code

Description automatically generated

Figure ‑ Building model method of Naive Bayes algorithm

A computer screen shot of text

Description automatically generated

Figure ‑ Evaluation method of Naive Bayes algorithm

A screen shot of a computer code

Description automatically generated

Figure ‑ Prediction method of Naive Bayes algorithm

A close up of a text

Description automatically generated

Figure ‑ Display method of Naive Bayes algorithm

### Result

Finishing on the implementing algorithms task, we will run these algorithms with the codes below (Please uncomment the code in the corresponding algorithm to run them):

A computer code with text

Description automatically generated

Figure ‑ Codes to launch the J48 (Decision Tree) algorithm

A white background with text

Description automatically generated

Figure ‑ Codes to launch the SMO (SVM) algorithm

A computer code on a white background

Description automatically generated

Figure ‑ Codes to launch the Naive Bayes algorithm

After running the algorithm on NetBeans, we got these results:

Table . Comparison of multiple model results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | Runtime |
| J48 (Decision tree) | 91.7457 % | 0.917 | 0.917 | 1 seconds |
| SMO (SVM) | 80.0127 % | 0.790 | 0.00 | 5 seconds |
| Naive Bayes | 87.413 % | 0.873 | 0.874 | 1 seconds |

## Improvement of Results.

### Methodology

To improve the performance of model in step 2, we analyst data and preprocess data, then we apply another algorithm to improve this new data. Now, we present these steps into two parts.

1. **Preprocessing.**

Some steps are in step 1 which will be changed. Here is the process in step 1:

A diagram of a flowchart

Description automatically generated

Figure ‑ Flow preprocessing

A diagram of power transformer cable

Description automatically generated

Figure ‑ Flow Post-processing

Firstly, we move the remove outlier step to transform data (encoding and power transformer) which is often more effective for the following reasons:

1. **Outlier Identification Accuracy**

* Outliers are identified based on the statistical properties of the data, such as mean, standard deviation, or distribution.
* **Before transformation**, the data may be skewed or have non-normal distributions, which can make it harder to set meaningful thresholds for detecting outliers.
* **After transformation**, the data typically follows a more normalized distribution (e.g., Gaussian), making it easier to accurately identify true outliers.

1. **Handling Skewed Distributions**

* Outliers in skewed data may not truly represent anomalies; instead, they may simply be extreme values due to the skewness of the distribution.
* A power transformer reduces skewness and brings the data closer to a normal distribution, clarifying whether a value is truly an outlier or part of the natural range of the data.

1. **Preservation of Data Integrity**

* If outliers are removed **before transformation**, some data points that appear extreme under a skewed distribution might not be true outliers after normalization.
* Removing these data points prematurely could result in loss of valuable information.

1. **Effectiveness of Transformations**

* Many transformations, such as scaling or power transformations, rely on the presence of the full data range to calculate appropriate adjustments.
* Removing outliers first may cause these transformations to misrepresent the true distribution of the data, leading to poor results.

1. **Encoding Categorical Variables**

* If you encode categorical variables **after** removing outliers, you risk discarding data points without accounting for the relationships between categories and other variables.
* Encoding should be performed first to ensure all features are represented in the same numerical format for subsequent steps like outlier removal or transformation.
* Skewed data may cause you to incorrectly classify valid data points as outliers.
* The thresholds used to identify outliers may not align with the actual distribution after transformation.
* By removing outliers **after** encoding and applying a power transformer, you ensure that your outlier detection process is based on a well-behaved dataset, reducing the risk of losing valuable data or misidentifying outliers.

Secondly, we use power transformers to transform data rather than normalization. Here is the reason:

* + 1. Transformation of Distribution
* Power Transformer aims to make the data distribution closer to a Gaussian distribution. It reduces skewness in the data, stabilizes variance, and handles outliers more effectively. Skewed distributions, which are common in real-world datasets, can harm model performance by biasing learned patterns. The Power Transformer adjusts for this.
* StandardScaler only standardizes the data by removing the mean and scaling to unit variance. It does not change the shape of the data's distribution. If the data is skewed or non-Gaussian, it remains skewed.
  + 1. Handling outliers
* Power Transformer reduces the impact of outliers by transforming the scale and shape of the data. Outliers are "pulled closer" to the center of the distribution, making them less influential.
* StandardScaler does not mitigate the impact of outliers. Outliers can skew the mean and standard deviation, leading to improperly scaled data.
* Outliers can distort feature scaling, making it harder for models to learn patterns. The Power Transformer minimizes this issue.

1. **Consistency Across Features**

Power Transformer transforms both numeric and encoded data, ensuring that all features are on a similar scale and follow similar distributions. While standardscaler only applies to numeric features, leaving encoded features untouched. This can result in inconsistent scaling between feature types, leading to poorer model performance. Applying the Power Transformer to encoded categorical features also ensures they are scaled in a way that aligns with other features, improving downstream model performance.

Finally, we drop the unimportant feature.

* We drop unuseful columns like “id” and “Gender Female”. Besides that, we use Random Forest to find important features. I split the training model into two parts: train and test data. We perform Random Forest model to train dataset and then printing the important feature. The results here:

A graph with blue bars

Description automatically generated

Figure ‑ Feature importance by Random forest

So, I drop all data which do not important:



Now, we have the new data here:

1. **Perform another model.**

We are trying to build some another model for dataset.

* **Random Forest model**: It is an ensemble learning method that builds multiple decision trees and combines their outputs to improve accuracy and reduce overfitting. It is a type of Bagging, as it trains each tree on a random subset of the data and averages their predictions. Random forest model will be implemented by run this below code:

A computer code with text

Description automatically generated

Figure ‑ Function to build RandomForest model

A computer code with text

Description automatically generated with medium confidence

Figure ‑ Predict function of model RandomForest

A white background with colorful text

Description automatically generated

Figure ‑ Evaluation function of model RandomFores

We run the code below to implement model:

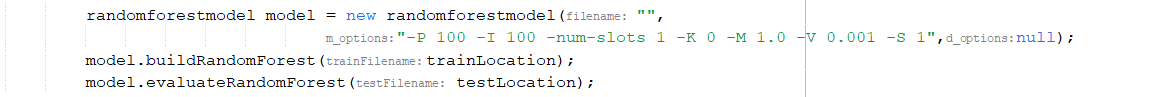


Figure ‑ Implement RandomForest model

* **Neural Network model**: A neural network is a computational model inspired by the human brain, consisting of layers of interconnected nodes (neurons). It learns patterns from data by adjusting weights through training, making it powerful for tasks like classification, prediction, and recognition. I try to use this algorithm because Neural networks outperform decision trees in handling non-linear relationships, high-dimensional data, and large datasets, thanks to their ability to automatically extract features and model complex patterns. However, decision trees are simpler, faster, and more interpretable for smaller datasets. Neural Network model have below functions:

A screenshot of a computer

Description automatically generated

Figure ‑ All functions in Neural Network

We run the below code to implement this model:

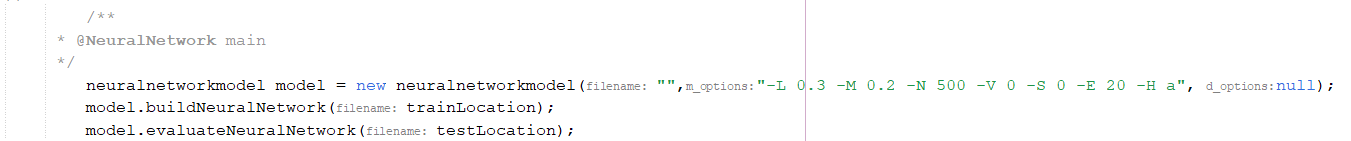


Figure ‑ Implementing Neural Network

* **Gradient Boosting** **model**: Gradient Boosting is an ensemble learning technique that combines multiple weak learners, typically decision trees, to create a strong predictive model. It works iteratively, where each tree corrects the errors of the previous ones by focusing on the misclassified examples, resulting in improved accuracy. In classification tasks, Gradient Boosting often outperforms standalone decision trees like J48 because it reduces bias and variance by leveraging multiple models. While J48 provides a single, interpretable tree, Gradient Boosting is more robust in handling complex patterns, reducing overfitting, and achieving higher predictive performance. Gradient Boosting model has below functions:

A screenshot of a computer

Description automatically generated

Figure ‑ All functions in Gradient Boosting

We run below code to implement this model:



Figure ‑ Implementing Gradient Boosting model

* **Voting Classifier model**: Voting techniques combine predictions from multiple models to improve accuracy and robustness, reducing overfitting and individual model bias. Unlike J48, which may overfit or struggle with complex data, voting leverages diverse models for better generalization in classification tasks. The voting classifier has below functions:

A screenshot of a computer

Description automatically generated

Figure ‑ Function to build Voting classifier

A screen shot of a computer code

Description automatically generated

Figure ‑ Function to evaluate and predict in Voting Classifier

We run the below code to implement this model:

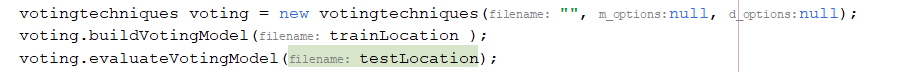


Figure ‑ Implement Voting Classifier

### Comparison of the results

Table . Comparison of advance model results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | Runtime |
| J48 (Decision tree) in past dataset | 91.7457 % | 91.7% | 91.7% | 1 seconds |
| J48 (Decision tree) in new dataset | 93.192 % | 93.1% | 93.2% | 1 seconds |
| Random Forest | 93.6012 % | 93.6 % | 93.6% | 4 seconds |
| Neural Network | 91.1458 % | 91% | 91.1% | 16 seconds |
| Gradient Boosting | 93.0804 % | 93% | 93.1% | 2 seconds |
| Voting Classifier | 94.122% | 94.1% | 94 % | 10 seconds |

## Model Evaluation

### Performance Metrics

In this section, we will use a table to summarize the performance metrics of two final models: J48 and Voting Classifier and compare their performance to see which model performs much efficient.

Table . Comparison of J48 and Voting Classifier on Some Performance Metrics

|  |  |  |
| --- | --- | --- |
|  | Voting Classifier | J48 |
| Correctly Classified Instances | 94.0307 % | 67.8783 % |
| Incorrectly Classified Instances | 5.9693 % | 32.1217 % |
| Kappa statistic | 0.9171 | 0.5978 |
| Mean absolute error | 0.0171 | 0.0943 |
| Root mean squared error | 0.1306 | 0.2859 |
| Relative absolute error | 8.2782 % | 45.9222 % |
| Root relative squared error | 40.7071 % | 89.0945 % |
| Run Time | 10 seconds | 1 second |

Table . Class Details of Voting Classifier

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TP  Rate** | **FP Rate** | **Precision** | **Recall** | **F-Measure** | **MCC** | **ROC** | **PRC** | **Class** |
| 0.821 | 0.020 | 0.759 | 0.821 | 0.789 | 0.773 | 0.900 | 0.637 | OvlII |
| 0.914 | 0.015 | 0.830 | 0.914 | 0.870 | 0.860 | 0.949 | 0.765 | Normal |
| 0.839 | 0.002 | 0.929 | 0.839 | 0.881 | 0.878 | 0.918 | 0.785 | Insuff |
| 1.000 | 0.003 | 0.996 | 1.000 | 0.881 | 0.996 | |  | | --- | | 0.998 | |  | | 0.996 | ObIII |
| 0.974 | 0.002 | 0.989 | 0.974 | 0.981 | 0.978 | 0.986 | 0.967 | ObII |
| 0.736 | 0.011 | 0.821 | 0.736 | 0.776 | 0.763 | 0.863 | 0.621 | OvlI |
| 0.888 | 0.012 | 0.910 | 0.888 | 0.899 | 0.886 | 0.938 | 0.822 | ObI |
| 0.940 | 0.007 | 0.941 | 0.940 | 0.940 | 0.934 | 0.967 | 0.896 | Weighted Avg. |
|  |  |  |  |  |  |  |  |  |

Table . Class Details Of` J48

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TP  Rate** | **FP Rate** | **Precision** | **Recall** | **F-Measure** | **MCC** | **ROC** | **PRC** | **Class** |
| 0.748 | 0.038 | 0.607 | 0.748 | 0.670 | 0.646 | 0.905 | 0.507 | OvlII |
| 0.895 | 0.019 | 0.798 | 0.895 | 0.843 | 0.831 | 0.963 | 0.716 | Normal |
| 0.847 | 0.003 | 0.921 | 0.847 | 0.882 | 0.879 | 0.975 | 0.845 | Insuff |
| 0.500 | 0.002 | 0.996 | 0.500 | 0.666 | 0.585 | 0.767 | 0.759 | ObIII |
| 0.968 | 0.172 | 0.514 | 0.968 | 0.671 | 0.635 | 0.968 | 0.829 | ObII |
| 0.656 | 0.014 | 0.755 | 0.656 | 0.702 | 0.686 | 0.909 | 0.586 | OvlI |
| 0.784 | 0.122 | 0.470 | 0.784 | 0.588 | 0.538 | 0.828 | 0.551 | ObI |
| 0.679 | 0.048 | 0.795 | 0.679 | 0.681 | 0.627 | 0.848 | 0.715 | Weighted Avg |

### Analysis of Results

The performance of the J48 Decision Tree and the Voting Classifier was evaluated using a wide range of metrics, including Correctly Classified Instance, Kappa Statistics, and other metrics. This section will provide a comprehensive analysis of those metrics and discuss any tradeoffs, the use case of each model, and the context of our data.

**Correctly Classified Instances and Incorrectly Classified Instances**

Correctly classified instances, measure the proportion of instances, which have the predicted label matches the actual label of the validation data. Conversely, the percentage of misclassified instances is measured by incorrectly classified instances. A high proportion of correctly classified instances reflects a model’s ability to learn and generalize the underlying patterns of data. In this case, Voting Classifier has the accuracy of 94.0307%, approximately 30% higher than that of J48 (67.8783%), and maintains only 5.9693% of mismatched labels, 25% lower than J48. The high accuracy and low rate of mismatched labels demonstrates that, compared to J48, Voting Classifier is more effective in understanding data distribution, and generalizes well with unseen data, and is more reliable as it produces less mismatched instances. Therefore, Voting Classifier is a strong choice for applications that require high precision, and less errors such as medical diagnosis. These results emphasize the advantage of ensemble learning as Voting Classifier combines predictions to multiple models. However, accuracy does not account for the costs of misclassification and might not be the absolute metrics for evaluating the performance of models, as it could be overfitting by favoring one class over these others. In this case, the data is used to evaluate situations in critical domain like healthcare, even a small percentage of incorrect classifications can have severe consequences. A comprehensive analysis of MCC, F1-score, Precision, Recall, and False Positive Rate should be utilized to better asses the model’s performance.

**Kappa Statistic**

The Kappa Statistic (k), also known as Cohen’s Kappa, is a statistical measure that evaluates the agreement between predicted and actual labels. The higher Kappa Statistic, the closer the predicted labels to the actual labels. The advantage of Kappa Statistic lies on its account of the possibility of correct classifications occur by chance, especially in imbalance dataset. The **Voting Classifier** achieved a Kappa statistic of **0.9171**, indicating *almost perfect agreement* between its predicted and actual labels. This suggests that the Voting Classifier is highly effective in capturing the true underlying patterns in the data. In contrast, **J48** achieved a Kappa statistic of **0.5978**, reflecting a lower level of agreement. While this still indicates moderate to substantial agreement, it highlights the superiority of the Voting Classifier. Kappa provides a single value for agreement, it does not indicate where the model might be failing, such as specific classes or conditions. However, while Kappa offers a more sophisticated evaluation than accuracy alone, it should be complemented with other metrics, such as precision, recall, and F1-score, to ensure a comprehensive assessment of model performance.

**MCC**

To evaluate our model more thoroughly, it is critical to look into the Matthews Correlation Coefficient (MCC) as ittakes into account all four components of the confusion matrix: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

However, MCC does not provide detailed insights into the errors that the model is making, and additional metrics such as Precision Recall is still essential. Voting Classifier achieved a weighted average MCC of 0.934, higher than 0.627 of J48, indicating Voting Classifier’s superior ability to produce accurate predictions, with an excellent balance between correctly predicting positive and negative instances. Besides that, Voting Classifier display consistency among classes, which confirms Voting Classifier’s ability to perform well on every class. Conversely, J48 has inconsistent values of MCC among classes, with some classes with extremely high MCC, and some with extremely low MCC, indicating the inconsistent performance of J48 on different classes. Therefore, Voting Classifier is more robust to variations in data and can perform equally excellently with different types of datasets.

**Precision and Recall:**

In critical domain such as healthcare, which our data, and study question revolves around, it is highly essential to minimize false positives, as they may lead to unnecessary treatments, emotional distress, and false negatives, as they may bring severe consequences for left untreated diseases. As a result, precision- the proportion of correctly predicted positive instances out of all instances predicted as positive and reflects the model's ability to avoid false positives. and recall- the proportion of actual positive instances that were correctly predicted by the model and reflects the model's ability to minimize false negatives will be evaluated. The Voting Classifier achieved an average Precision of 0.941, far higher than the 0.795 achieved by J48. This measure indicates that the Voting Classifier is far more effective in avoiding false positives and making true positive predictions. Besides that, The Voting Classifier outperformed in Recall, with a weighted average of 0.940, compared to 0.679 of J48, which suggests that the Voting Classifier is more advantageous at identifying true positive instances and minimizing false negatives. Furthermore, The Voting Classifier demonstrates a consistency of Precision and Recall values among each class, which affirms that there is no class being favored by the model, and thus, lead to false classifications or overfitting. Conversely, J48 Decision Tree model showcases a significant disparity among each class in Precision and Recall value, with some classes having the Recall as high as 0.968, and close to 1, and some classes having the Recall as low as 0.5, and close to 0.5. This demonstrates that J48 might be biased towards majority class and performing poorly on minority class. Moreover, in some classes, for example. ObIII, J48 achieved an extremely high precision (0.966) and 0.5, which suggests that the model failed to minimize false positive, which can lead to unnecessary interventions. Conversely, in ObI class, J48 achieved a high recall (0.784), and low precision (0.47), which indicates that J48 may produce some false negatives, which can delay diagnoses, and treatments. As a result, in a domain where precise predictions are highly valued, the Voting Classifier will be a stronger choice.

**F-measure**

This section will evaluate the model’s ability to balance between precision and recall by another aggregate metric- the F-measure. Voting Classifier achieved a weighted average 0.94, higher than 0.681 of J48, reaffirms Voting Classifier’s superior ability to balance between precision and recall. Voting Classifier continues to show consistency among every class, while there are inconsistencies between classes for J48, which suggests that while J48 performs well on some certain classes, it performs poorly on other classes.

**Runtime**

Since Voting Classifier is an ensemble learning model, which combines the predictions of all models, and produces final predictions based on that, Voting Classifier is not computationally efficient as J48, as it takes Voting Classifier 10 seconds to run, compared to 1 second of J48. In cases where instant, and quick predictions must be made, and large datasets, Voting Classifier is not an optimal choice.

**Mean Absolute Error, Root Mean Square Error, Relative Absolute Error, Root Relative Square Error**

Apart from accuracy, it is critical to look into the error metrics, to grab a more comprehensive understanding of our model. The definition for each error metric, and their insights, will be demonstrated in the following table.

Table . Comparison details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metric | Definition | Voting Classifier | J48 | Insight |
| Mean Absolute Error (MAE) | Measures the absolute difference between predicted labels and true labels | 0.0171 | 0.0943 | Voting Classifier's MAE is significantly smaller than that of J48, which demonstrates the Voting Classifier’s superior ability to produce more accurate labels |
| Root Mean Squared Error (RMSE) | Measures the square root of the average squared differences between predicted and actual values | 0.1306 | 0.2859 | RMSE of Voting Classifier is almost half of that of J48. This result reaffirms Voting Classifier’s superior ability to produce accurate labels |
| Relative Absolute Error (RAE) | Compares the total absolute error of a model to that of a naive baseline | 8.2782% | 45.9222% | The Voting Classifier achieved 8.2782%, which is smaller than J48 by more than 30%. The low RAE demonstrates that, compared to J48, Voting Classifier performs much more efficient than a naive baseline |
| Root Relative Squared Error (RRSE) | Compare the squared errors of a model to those of a baseline | 40.7071% | 89.0945% | The RRSE of Voting Classifier is nearly half of that of J48, indicating Voting Classifier’s superior ability to outperform a naive baseline model. |

Voting Classifier achieved a low value for all error metrics, which confirms its predictions are not only accurate, but also more excellent than what would be achieved by a naive baseline model. Therefore, Voting Classifier is a more appropriate choice than J48 for tasks requiring accuracy such as medical diagnosis. However, the error metrics alone cannot fully address the efficiency of a model, as it might be bias towards majority classes. As a result, class-specific metrics such as precision, recall, ROC, and PRC area should also be considered to ensure consistent performance among classes

**ROC and PRC Areas**:

To better understand the performance of our models, and to evaluate if there are any tradeoffs between Precision and Recall, and or the ability of model to distinguish between classes, it is critical to take into considerations ROC and PRC. ROC (Area Under the ROC Curve) measures a model’s ability to distinguish between classes, while PRC (Area Under the PRC) measures the ability of the model to maintain high Precision and Recall in every class. Voting Classifier achieved a weighted average ROC and PRC at 0.967 and 0.896, higher than 0.848 and 0.715 of J48. This statistical measurement demonstrates that Voting Classifier is more efficient than J48 at distinguishing between classes, and balance between precision and recall. Moreover, Voting Classifier maintains consistently ROC, and PRC values among every class, which suggests Voting Classifier ensures efficient performance among classes. Conversely, J48 displays inconsistencies, with some classes achieving extremely high ROC, and low PRC, and vice versa. For example, in class ObI, J48 achieved a ROC value of 0.828, and a PRC value of 0.551, indicating that J48 separate classes well, but failed to balance the tradeoffs between precision and recall. This indicates that J48 failed to perform well in some minority classes and is biased towards majority class.

**Conclusion:** In applications such as medical diagnosis, as in our case, precise, and consistent predictions are highly critical, and time is not a priority, Voting Classifier is a stronger choice considering its ability to maintain accurate and consistent predictions, with less false positives, false negatives in imbalanced data.

# CONCLUSION

## Summary

Our team utilized structural-driven analysis techniques to create and execute this program. The outcomes we achieved include:

* Theoretical:
* Establishing the project’s objectives.
* Implementation of Data Mining Framework simply.
* Analyzing the problem and applying learned methodologies.
* Creating some machine learning models.
* Program:
* Using Python to preprocess data which produces clean data.
* Using Java to build Weka API.
* The program can be used to classify the obesity risk.
* However, our model is not yet highly professional. We have not significantly improved the model performance in the improvement step.

## What we learn after project

The members of our group have gained a lot of knowledge after finishing this project, including:

* Preprocessing steps include data cleaning, data integration, data selection, and data transformation.
* Performing many classification algorithms to classify the obesity risk
* Deploying many ways to improve the performance of models.
* Implement evaluation of model.
* Remark and choose the best model.
* Additionally, each group member will benefit from this collaboration by teamwork.
* How to lay out a whole report's structure.

For each person, create a particular implementation strategy with a deadline.

* Analyze to post, ask for a topic, ask for information.

## Completed work & Future work.

We learned and practiced data mining frameworks: Ask a question, collect data, clean data, implement some algorithm, improvement of result and evaluation. We also had an experiment with WEKA API

In the future, I will improve the accuracy of the model by adding some new information or some advance algorithms which do not support by WEKA now.

# REFERENCES

1. <https://www.kaggle.com/datasets/jpkochar/obesity-risk-dataset/data?fbclid=IwY2xjawG3TgZleHRuA2FlbQIxMAABHTRXzp23Y0Ufz-SBJi0IH3exShmPEgJFaLAOwi_G6p0UKVLxyHbY15FPYg_aem_es4gYNyrtL90rWG6zAr1aw>
2. <https://waikato.github.io/weka-wiki/use_weka_in_your_java_code/>
3. <https://www.youtube.com/playlist?list=PLea0WJq13cnBVfsPVNyRAus2NK-KhCuzJ>
4. <https://www.youtube.com/watch?v=Hpa8dnR-_OU&list=PLqbz4rsGemn1YvsMoCd2W0DjoEKSFw7qT>