Journal of Applied Science, Engineering, Technology, and Education *ISSN: 2685-0591 (online)*
Vol. 6 No. 2 (2024)
https://doi.org/10.35877/454RI.asci3519

Comparasion of Naïve Bayes Algorithms and Decision Tree for Classifying Hero Fighter Items in the Mobile Legends

Hasbanur Hafidz* & M. Fakhriza

Department of Computer Science, Faculty of Science and Technology, Universitas Islam Negeri, Sumatera Utara, 20353, Indonesia

Abstract

The classification of hero fighter items in the Mobile Legends Game is a significant challenge due to the complexity of features and the variety of strategies employed by players. This study aims to develop an effective classification model using the Naïve Bayes and Decision Tree algorithms and compare the performance of these two algorithms. The dataset used in this study was obtained from item recommendations during live Gameplay, community sources such as forums, and Game guides. This dataset contains relevant information to support the classification of items for hero fighters, such as hero attributes, roles, and enemy types. The model training process was conducted using the scikit-learn library, with data split into 80% for training and 20% for testing. The study results show that the Decision Tree algorithm consistently delivers better performance than Naïve Bayes. In the evaluation using accuracy metrics, Decision Tree achieved an accuracy rate of 84.78%, significantly higher than Naïve Bayes, which only reached 45.65%. Furthermore, the precision, recall, and f1-score metrics for Decision Tree also showed superior results for almost all classes compared to Naïve Bayes. Based on these findings, the Decision Tree algorithm is recommended as a more suitable choice for classifying hero fighter items in the Mobile Legends Game.

Keywords: Data mining, Classification, Naïve Bayes, Decision Tree, Mobile Legends, Hero Fighter

Received: 29 October 2024 Revised: 15 December 2024 Accepted: 27 December 2024

1. Introduction

Mobile Legends has become one of the most popular games in the world, especially among mobile players. In this game, players choose heroes that are divided into various categories such as tank, marksman, mage, support, assassin, and fighter (Bayulianto et al., 2023). Each hero category has a specific role and strategy in the game. In particular, fighter heroes have an important role in dealing damage and defending in close combat (Habibah et al., 2023).

Choosing the right items for each hero fighter is crucial in determining the player's performance and success in the game (Ananda et al., 2022). These items provide additional attributes and abilities that can increase the hero's durability, attack, or special abilities (Ditendra et al., 2022). Therefore, the right classification of items for a hero fighter can give players a competitive advantage (Fauzan & Hikmah, 2022).

In the realm of technology and data analysis, machine learning has shown great potential in processing complex data and providing accurate recommendations (Rachmat et al., 2024). Naive Bayes and Decision Tree are two algorithms that are often used in classification due to their simplicity and effectiveness (Hariyanti et al., 2024). Naive Bayes, with its probability-based approach, and Decision Tree, with its decision-based problem solving method, offer different but equally effective approaches in data classification (Alfudola et al., 2023).

This research aims to compare the performance between classification models using Naive Bayes and Decision Tree algorithms in determining the most suitable items for fighter heroes in the Mobile Legends game (Ramadhani & Ramadhanu, 2024). Thus, this research is expected to contribute in the form of more accurate and optimal item recommendations, which in turn can improve player performance in the game (Farid et al., 2022).

* Corresponding author.
E-mail address: bannhfdz@gmail.com

Through this research, it is hoped that a more effective and efficient algorithm can be found in the classification of hero fighter items, so that it can be implemented in the in-game item recommendation system (Amani et al., 2024). This not only provides benefits for players in improving the playing experience (Mukhsinin et al., 2024), but also provides insight for game developers in optimizing the item recommendation feature in Mobile Legends (Arrayyan et al., 2024).

2. Methods

2.1. Data Collection

At this stage, it is determined what variables or features will be used to classify hero fighter items in mobile legend games using the Naïve Bayes and Decision Tree algorithms (Ramadhon et al., 2024). Furthermore, data from the variables that will be used in carrying out the research is collected (Hajaroh et al., 2024). The data used in this study were obtained from:

a. Data Source: Mobile Legends game data which includes hero fighter information and items used in various matches.

b. Type of Data: this data includes hero fighter attributes (for example: hero name, basic attributes), types of items used (for example: attack, defense items), and match results.

c. Data Collection Method: the data is collected through scraping from official websites, community forums, and data provided by game developers(Iqbal et al., 2023).

2.2. Data Processing

After the data collected is sufficient, the next stage will be data processing (Macfud et al., 2023). Data processing will be selected, among others, by eliminating incomplete data and duplicate data (Nurnawati et al., 2023). Furthermore, the data will be separated based on its function, 80% of the overall data will be used as training data, while the other 20% is used as test data.

2.3. Building the Model

Based on the data that has been processed, the research model will be built (Permana et al., 2021). This stage involves selecting the right algorithm to achieve the research objectives (Irfon & Soen, 2022). This research will build two classification models using the Naive Bayes and Decision Tree algorithms for the classification of hero fighter items in the Mobile Legends game (Punkastyo et al., 2024).

a. Naive Bayes Model: Naive Bayes algorithm will be applied to build a classification model based on the probability of each attribute.

b. Decision Tree Model: The Decision Tree algorithm will be applied to build a classification model based on the decision tree formed from the data attributes (Purnomo & Abdul, 2022).

2.4. Data Analysis Techniques

Data analysis techniques in this study, as follows:

a. Collect data from relevant sources, such as in-game item recommendations, community forums, and game guides.

b. Perform a labeling process on the data.

c. Performing data preprocessing which consists of 4 steps:

   1) Data Cleaning: Removes irrelevant or duplicate data.
   2) Label Encoding: Converts categories into a numeric format for use in the model.
   3) Split Dataset: The data is divided into two parts, 80% for training and 20% for testing.
   4) Data Transformation: Perform normalization or other transformations to prepare the data

d. Divide training data and test data.

e. Model Evaluation.

   After training, the model is evaluated using several metrics:

1) Accuracy: Measured to determine how well the model performs classification.
2) Precision, Recall, and F1-score: These metrics are used to evaluate the model's performance in identifying different classes.

f. Deploy model, if the model has achieved the desired performance, then the model will be deployed for use in a real environment or applied to scenarios that are in accordance with the research. The model will be deployed into a web-based application.

3. Results and Discussion

3.1. Results

In this section, we will explain the implementation and evaluation results of the hero fighter item classification model in the Mobile Legends game using the Naive Bayes and Decision Tree algorithms. The research stages involve preprocessing the dataset, training the model, and evaluating the model performance based on predetermined metrics. Figure 1 illustrates the diagram of the hero fighter item classification model in the Mobile Legends game using the Naive Bayes and Decision Tree algorithms.

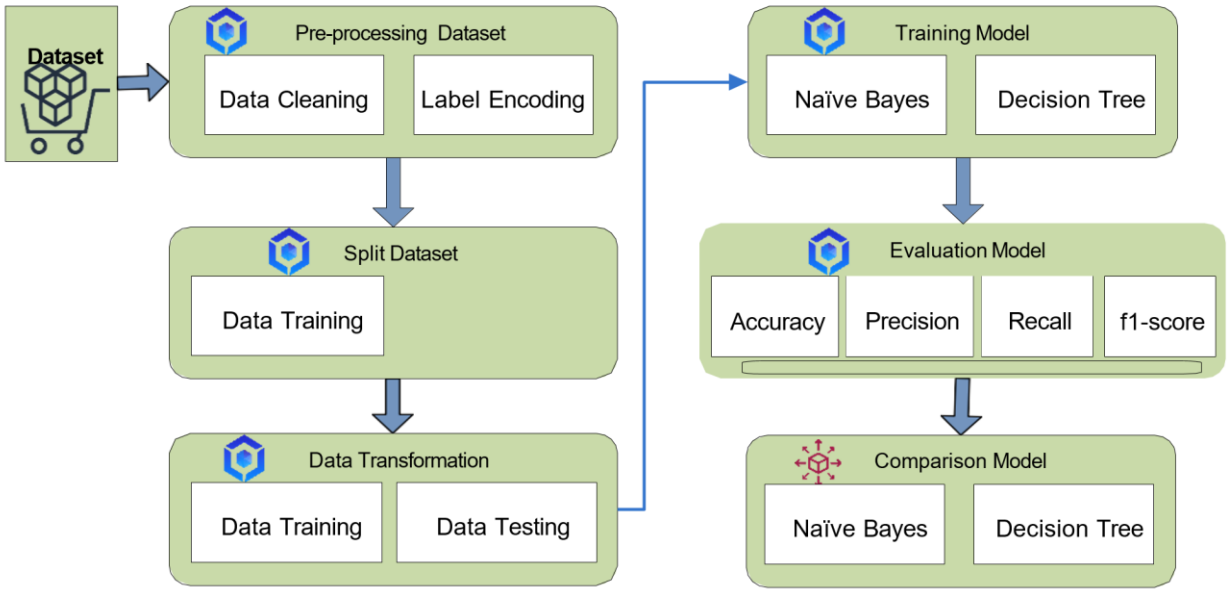

Figure 1. Classification Model Diagram

The Figure 1 shown illustrates the flow of the research process in building a fighter item classification model in the Mobile Legends game using the Naive Bayes and Decision Tree algorithms. This diagram includes several main stages that are arranged systematically, starting from preprocessing the dataset, splitting the dataset, normalizing the dataset (data transformation), training the model to evaluating and comparing models.

3.1.1. Dataset Collection

The dataset contains information about fighter items in the Mobile Legends game. This dataset is the basic material for training and testing classification models. The dataset used in this research is obtained from item recommendation data from live games, community sources such as forums, as well as game guides. This dataset contains relevant information to support the classification of items on fighter heroes in the Mobile Legends game. This dataset contains 229 rows of data covering various combinations of attributes and targets. This sufficient amount of data is expected to provide optimal training results for the classification model to be built. Figure 2 displays a sample of the dataset used in classifying items on fighter heroes in the Mobile Legends game.

	No	Hero	Role	Enemy_Type	Map_Condition	Player_Strategy	Hero_Level	Gold_Collected	Team_Composition	Objective_Priority	Win_Probability	Recommended_Item
0	1	Hilda	Fighter	Magical	Late Game	Balanced	6	5739	1 Marksman, 1 Tank	Farming	13.49	Blade of Despair
1	2	Hilda	Fighter/Tank	Magical	Late Game	Offense	6	19987	1 Tank, 2 Mage	Teamfight	72.01	Endless Battle
2	3	Alucard	Fighter/Tank	Physical	Early Game	Defense	1	13352	3 Fighters	Pushing	23.30	Warrior Boots
3	4	Lapu-Lapu	Fighter	Physical	Early Game	Defense	3	4803	1 Support, 1 Tank, 1 Marksman	Teamfight	87.54	Immortality
4	5	Freya	Fighter/Assassin	Hybrid	Early Game	Offense	3	18581	2 Fighter, 1 Assassin	Pushing	27.85	Immortality
...	...	...	...	...	...	...	...	...	...	...	...	...
224	225	Alucard	Fighter/Assassin	Magical	Early Game	Offense	8	3508	1 Fighter, 2 Mage	Farming	82.08	Endless Battle
225	226	Martis	Fighter/Assassin	Physical	Early Game	Defense	12	11048	3 Fighters	Teamfight	19.86	Queen's Wings
226	227	Chou	Fighter	Physical	Mid Game	Defense	3	16998	3 Fighters	Teamfight	73.72	Bloodlust Axe
227	228	Chou	Fighter	Hybrid	Mid Game	Offense	7	16944	2 Assassin, 1 Tank	Pushing	15.03	Blade of Despair
228	229	Leomord	Fighter/Tank	Physical	Mid Game	Balanced	7	4848	1 Fighter, 2 Mage	Farming	38.28	Athena's Shield

229 rows × 12 columns

Figure 2 Sample Dataset

Based on Figure 2, the dataset consists of several main attributes that can be described as follows:

a. No: Represents the sequence number of the dataset row.
b. Hero: The name of the specific hero that will be used in the classification (example: Chou and Alucard).
c. Role: Some fighters in the Mobile Legends game have sub-roles, such as tank or assassin.
d. Enemy_Type: The type of opponent encountered (physical, magical, and hybrid). The type of opponent will affect the item selection (for example, fighting an opponent with a magic attack requires a magic defense item).
e. Map_Condition: The game condition (early game, mid game, and late game) determines the item type (cheap items for early game, expensive items for late game).
f. Player_Strategy: Player strategy (offense, defense, and balanced). The player's strategy, whether they focus more on offense, defense, or balance.
g. Hero_Level: The hero's level (1-15) affects survival and attack abilities, thus affecting item selection.
h. Gold_Collected: The amount of gold the hero has (0-20,000). Heroes with more gold have access to more expensive and more powerful items.
i. Team_Composition: Team composition (for example, 2 marksmen, 1 tank). The team composition affects the hero's role, for example, a fighter hero is more defensive if there is no tank.
j. Objective_Priority: The player's focus on game objectives (farming, pushing, and teamfight), thus affecting the type of items required.
k. Win_Probability: The team's win probability (estimated from game conditions). Provides context about the game state, whether the team is in a winning or losing position.
l. Recommended_Item: Outputs the most suitable item for a combination of heroes and other conditions.

3.1.2. Dataset Preprocessing

The dataset that has been collected will then be subjected to a preprocessing stage. The preprocessing stage is a process to ensure the dataset is ready to be used in model training. This stage aims to improve the quality of the dataset so that it can be used effectively by the model. Raw datasets often contain anomalies, empty values, or irrelevant attributes. Therefore, preprocessing includes several steps, namely data cleaning, label encoding, split dataset, and data transformation.

a. Data Cleaning

Data cleaning is performed to ensure that the dataset is clean and free from any interference that could affect the model training results. Data cleaning includes removing irrelevant data, duplicates, or missing values. This process ensures that the data used in training is free from inconsistencies. The steps taken include:

1) Identification of Irrelevant Columns

Attributes such as 'No' are removed as they do not contribute directly to the classification results. The 'No' attribute is simply a numeric label that is irrelevant to the classification process based on other numeric attributes.

2) Handling Missing Values

The dataset is checked to ensure that there are no missing values. If empty values are found, the following steps are performed:

a) For numeric attributes such as 'Hero_Level', the blank value is filled with the average (mean) of the column.
b) If the number of blank values is significant, the corresponding data rows are deleted so as not to affect the quality of the dataset.

Referring to the dataset collection results, there are no missing values.

3) Outlier Check

Data with values that are too extreme or outside the normal range are identified and considered for deletion or modification according to context. Referring to the results of the dataset collection, for the attribute 'Gold_Collected' and the attribute 'Win_Probability' have values that are too extreme and are therefore considered for deletion.

Figure 3 is a sample dataset resulting from the data cleaning process carried out at the dataset preprocessing stage.

	Hero	Role	Enemy_Type	Map_Condition	Player_Strategy	Hero_Level	Team_Composition	Objective_ Priority	Recommended_Item
0	Hilda	Fighter	Magical	Late Game	Balanced	6	1 Marksman, 1 Tank	Farming	Blade of Despair
1	Hilda	Fighter/Tank	Magical	Late Game	Offense	6	1 Tank, 2 Mage	Teamfight	Endless Battle
2	Alucard	Fighter/Tank	Physical	Early Game	Defense	1	3 Fighters	Pushing	Warrior Boots
3	Lapu-Lapu	Fighter	Physical	Early Game	Defense	3	1 Support, 1 Tank, 1 Marksman	Teamfight	Immortality
4	Freya	Fighter/Assassin	Hybrid	Early Game	Offense	3	2 Fighter, 1 Assassin	Pushing	Immortality
...	...	...	...	...	...	...	...	...	...
224	Alucard	Fighter/Assassin	Magical	Early Game	Offense	8	1 Fighter, 2 Mage	Farming	Endless Battle
225	Martis	Fighter/Assassin	Physical	Early Game	Defense	12	3 Fighters	Teamfight	Queen's Wings
226	Chou	Fighter	Physical	Mid Game	Defense	3	3 Fighters	Teamfight	Bloodlust Axe
227	Chou	Fighter	Hybrid	Mid Game	Offense	7	2 Assassin, 1 Tank	Pushing	Blade of Despair
228	Leomord	Fighter/Tank	Physical	Mid Game	Balanced	7	1 Fighter, 2 Mage	Farming	Athena's Shield

229 rows × 9 columns

Figure 3. Sample Dataset Results of Data Cleaning

Figure 3 is a sample dataset after the data cleaning stage, which eliminates the 'No', 'Gold_Collected', and 'Win_Probability' attributes. This is done because these attributes are considered irrelevant for the classification process based on other attributes.

b. Label Encoding

The use of Machine Learning in Python cannot be done in multi-class if the data type on the feature (X) and label (Y) have different data types, so label encoding must be done. Label encoding aims to convert categorical values in the dataset into a numerical format that can be understood by the algorithm. The dataset contains categorical attributes such as 'Hero'. Since classification algorithms require numeric inputs, these categorical attributes are converted into numeric representations using the label encoding method which serves to simplify the computational process. The application of this label encoding method ensures that the dataset has a format that is suitable for the machine learning algorithm used.

In the record dataset, there are 8 columns that contain string/object data, namely the 'Hero', 'Role', 'Enemy_Type', 'Map_Condition', 'Player_Strategy', 'Hero_Level', 'Team_Composition', 'Objective_Priority', and 'Recommended_Item' columns, as presented in Figure 4.

```
RangeIndex: 229 entries, 0 to 228
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Hero                 229 non-null    object
 1   Role                 229 non-null    object
 2   Enemy_Type           229 non-null    object
 3   Map_Condition        229 non-null    object
 4   Player_Strategy      229 non-null    object
 5   Hero_Level           229 non-null    int64
 6   Team_Composition     229 non-null    object
 7   Objective_ Priority  229 non-null    object
 8   Recommended_Item     229 non-null    object
dtypes: int64(1), object(8)
```

Figure 4. Dataset Type Format

In Figure 4, there are 9 data that have string/object types so that when the label encoder is done, the data type is converted to an integer. Table 1 below is an example of the label encoding result on the column 'Role'.

Table 1. Encoder Label on Role Column

No.	Role	Label Encoding
1.	Fighter	0
2.	Fighter/Assassin	1
3.	Fighter/Tank	2

The sample dataset after the label encoding process for all columns in the dataset can be presented in Figure 5.

	Hero	Role	Enemy_Type	Map_Condition	Player_Strategy	Hero_Level	Team_Composition	Objective_ Priority	Recommended_Item
0	3	0	1	1	0	6	1	0	1
1	3	2	1	1	2	6	3	2	3
2	0	2	2	0	1	1	6	1	6
3	4	0	2	0	1	3	2	2	4
4	2	1	0	0	2	3	5	1	4
...	...	...	...	...	...	...	...	...	...
224	0	1	1	0	2	8	0	0	3
225	6	1	2	0	1	12	6	2	5
226	1	0	2	2	1	3	6	2	2
227	1	0	0	2	2	7	4	1	1
228	5	2	2	2	0	7	0	0	0

229 rows × 9 columns

Figure 5. Sample Dataset of Label Encoding Results

c. Split Dataset

After the data cleaning and label encoding stages are completed, the dataset is divided into two parts, namely training data and testing data. The main purpose of splitting the dataset is to ensure that the model can learn the data patterns effectively and can be tested for accuracy on new data. The process of splitting the datasets is done using the train_test_split function from the scikit-learn library, with randomization parameters to ensure an even distribution of data in both groups. The dataset is split by composition:

1) Training Data

A total of 80% of the dataset is allocated as training data. This data is used to train the model and learn about the patterns in the dataset. Figure 3.6 is a sample of the training data.

	Hero	Role	Enemy_Type	Map_Condition	Player_Strategy	Hero_Level	Team_Composition	Objective_ Priority
32	0	2	0	2	1	12	2	1
136	3	0	2	0	2	12	1	1
154	8	2	0	0	1	4	1	0
21	7	0	2	1	0	4	1	2
26	8	2	2	0	0	6	0	0
...	...	...	...	...	...	...	...	...
124	6	2	1	2	2	14	0	2
54	8	2	0	0	0	8	4	0
133	8	2	0	0	0	8	4	0
43	2	2	1	0	2	2	1	2
215	6	1	1	0	0	6	3	0

183 rows × 8 columns

Figure 6. Sample Training Data

Figure 6 is a sample of training data after being split from the original dataset. The percentage of training data is 80% of the entire dataset. Referring to the previous description, the total dataset is 229, so the amount of training data is 183 data (80% of 229).

2) Testing Data

A total of 20% of the dataset is used as testing data. This data is used to test the performance of the model on data that has never been seen by the model before, so as to measure the generalization ability of the model. The testing data provides an overview of how the model will work in real-world situations. Figure 3.7 is a sample of testing data.

	Hero	Role	Enemy_Type	Map_Condition	Player_Strategy	Hero_Level	Team_Composition	Objective_ Priority
198	3	0	1	1	1	8	0	0
92	6	0	1	0	2	15	5	1
91	8	2	2	0	0	6	0	0
17	0	1	1	2	2	3	0	0
191	2	1	2	1	0	4	6	1
0	...	...	...	...	...	...	...	...
46	1	2	0	2	0	10	6	1
8	0	0	0	2	0	11	4	2
214	0	0	0	0	0	13	0	1
28	6	2	1	2	2	14	0	2
127	2	0	0	2	1	1	5	0

Figure 7. Sample Testing Data (Testing Data)

Figure 7 is a sample of testing data after being split from the original dataset. The percentage of testing data is 20% of the entire dataset. Referring to the previous description, the total dataset is 229, so the amount of testing data is 46 data (20% of 229).

d. Data Transformation

At this stage, dataset normalization is performed using Min-Max Scaler. This process aims to align the feature values within a certain range (e.g. 0 to 1). Normalization is performed on training data and testing data to avoid bias in the model due to different feature scales. This is important to ensure machine learning algorithms work optimally. The data

normalization process uses the Min- MaxScaler method. Before normalizing the data, then first know the min-max value for each column in the dataset. For more details, it can be presented in Figure 8.

	Min	Max
Hero	0	9
Role	0	2
Enemy_Type	0	2
Map_Condition	0	2
Player_Strategy	0	2
Hero_Level	1	15
Team_Composition	0	6
Objective_ Priority	0	2
Recommended_Item	0	6

Figure 8. Dataset Min-Max Value

1) Normalization of Training Data

The training data was normalized using the Min-Max Scaler method. The formula used is:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

where:

X' : data that has been
X : data to be
X_{min} : minimum value of the feature in the training data
X_{max} : maximum value of the feature in the training data

This normalization helps the algorithm to learn faster by ensuring that each feature has a balanced contribution to the model. An example of calculating data normalization on training data using the Min-Max Scaler method on the 'Role' column in the first row of training data, it is known that the original value is 2 (Figure 6). While the min value of the 'Role' column based on Figure 8 is 0 and the max value of the 'Role' column is 2. Then after normalization the results are obtained:

$$x^1 = \frac{x - \min(x)}{\max(x) - \min(x)} = \frac{2-0}{2-0} = \frac{2}{2} = 1$$

The results of data normalization on training data can be presented in Figure 9.

	Hero	Role	Enemy_Type	Map_Condition	Player_Strategy	Hero_Level	Team_Composition	Objective_ Priority
0	0.000000	1.0	0.0	1.0	0.5	0.785714	0.333333	0.5
1	0.333333	0.0	1.0	0.0	1.0	0.785714	0.166667	0.5
2	0.888889	1.0	0.0	0.0	0.5	0.214286	0.166667	0.0
3	0.777778	0.0	1.0	0.5	0.0	0.214286	0.166667	1.0
4	0.888889	1.0	1.0	0.0	0.0	0.357143	0.000000	0.0
...	...	...	...	...	...	...	...	...
178	0.666667	1.0	0.5	1.0	1.0	0.928571	0.000000	1.0
179	0.888889	1.0	0.0	0.0	0.0	0.500000	0.666667	0.0
180	0.888889	1.0	0.0	0.0	0.0	0.500000	0.666667	0.0
181	0.222222	1.0	0.5	0.0	1.0	0.071429	0.166667	1.0
182	0.666667	0.5	0.5	0.0	0.0	0.357143	0.500000	0.0

183 rows × 8 columns

Figure 9. Normalization Sample Training Data

Figure 9 is a sample of training data after the data normalization process using the Min-Max Scaler method.

2) Normalization of Testing Data

The test data is normalized using the scale parameters from the training data. This is important to maintain consistency between the training and testing data. An example of calculating data normalization on testing data using the Min-Max Scaler method on the 'Enemy_Type' column in the first row of testing data, it is known that the original value is 1 (Figure 7). While the min value of the 'Enemy_Type' column based on Table 1 is 0 and the max value of the 'Enemy_Type' column is 2. Then after normalization the results are obtained:

$$x^1 = \frac{x - \min(x)}{\max(x) - \min(x)} = \frac{1-0}{2-0} = \frac{1}{2} = 0.5$$

The results of data normalization on testing data can be presented in Figure 10.

	Hero	Role	Enemy_Type	Map_Condition	Player_Strategy	Hero_Level	Team_Composition	Objective_ Priority
0	0.333333	0.0	0.5	0.5	0.5	0.5	0.0	0.0
1	0.666667	0.0	0.5	0.0	1.0	1.0	0.833333	0.5
2	0.888889	1.0	1.0	0.0	0.0	0.357143	0.0	0.0
3	0.0	0.5	0.5	1.0	1.0	0.142857	0.0	0.0
4	0.222222	0.5	1.0	0.5	0.0	0.214286	1.0	0.5
0	...	...	...	...	...	...	...	...
41	0.111111	1.0	0.0	1.0	0.0	0.642857	1.0	0.5
42	0.0	0.0	0.0	1.0	0.0	0.714286	0.666667	1.0
43	0.0	0.0	0.0	0.0	0.0	0.857143	0.0	0.5
44	0.666667	1.0	0.5	1.0	1.0	0.928571	0.0	1.0
45	0.222222	0.0	0.0	1.0	0.5	0.0	0.833333	0.0

Figure 10. Normalization Sample Testing Data (Testing Data)

The normalization process is performed using the Min-MaxScaler library from scikit-learn. This transformation ensures that the test data can be accurately evaluated by the model.

3.1.3. Modeling

The model training process is the core step in this research. At this stage, machine learning algorithms are trained using datasets that have gone through preprocessing, normalization, and division into training data and testing data (Padilah et al., 2024). This research uses two machine learning algorithms for the classification of hero fighter items in the Mobile Legends game, namely Naïve Bayes and Decision Tree (Moerdyanto & Nuryana, 2023). The training dataset (training data) is used to build a classification model (Andriska et al., 2024).

a. Training Model Naïve Bayes

The training process is performed using the *scikit-learn* library with the following steps:

1) Model Initialization

The Naïve Bayes model is initialized using GaussianNB. The following python program code is used to initialize the Naïve Bayes model.

```
from sklearn.naive_bayes import GaussianNB model_nb = GaussianNB(var_smoothing=0.01)
```

The code line from sklearn.naive_bayes import GaussianNB, serves to import the GaussianNB class from the scikit-learn library, which is used to initialize the Naïve Bayes model.

The code line *model_nb=GaussianNB(var_smoothing=0.01)*, serves to initialize the GaussianNB model and provide the value of the var_smoothing parameter. The main function is to prevent division by zero or too small a probability value that could cause errors or poor performance.

2. Model Training

After the model is initialized, the next step is to *train* the model. However, before training the model, the *dataset* will be divided into training data and testing data using *train_test_split()*. The following *Python* program code is used to split the dataset.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=32)
```

The parameter *test_size=0.2* means that 20% of the data is used for testing, while 80% is used for training. The *random_state=32* parameter ensures consistent data distribution each run.

Next, the model is trained using the *fit()* method on the training data. This step allows the model to learn patterns from the training data to produce accurate predictions.

```
model_nb_fit = model_nb.fit(X_train, y_train)
```

The parameter used, *X_train*, is the (independent) feature data for training. It is the *input* used to train the model. The *y_train* parameter is the target (dependent) label corresponding to the *X_train* data. This label is the outcome or class to be predicted. At this stage, the Gaussian *Naïve Bayes* model learns the pattern or probability relationship between the features in the *X_train* data and the label in *y_train*.

After the training is complete, an initial evaluation is performed on the training data to ensure the model understands the basic patterns of the dataset (Lile & Suharjo, 2024). The results of the model evaluation will be discussed in the next section (Tanza & Utari, 2022).

b. Training Model Decision Tree

The training process is performed using the *scikit-learn* library with the following steps:

1. Model Initialization

The Decision Tree model is initialized using DecisionTreeClassifier with customized parameters. The following Python program code is used to initialize the Decision Tree model.

```
from sklearn.tree import DecisionTreeClassifier
model_dt = DecisionTreeClassifier(criterion='entropy', max_depth=10,
min_samples_leaf=1, min_samples_split=2, splitter='best')
```

The line of code from sklearn.tree import DecisionTreeClassifier, serves to import the DecisionTreeClassifier class from the scikit-learn library, which is used to initialize the Decision Tree model. While the next line of code serves to initialize the DecisionTreeClassifie model and provide parameter values *criterion='entropy', max_depth=10, min_samples_leaf=1, min_samples_split=2, splitter='best'*.

2) Model Training

After the model is initialized, the next step is to train the model. However, before training the model, the dataset will be divided into training data and test data using *train_test_split().* The following Python program code is used to split the dataset.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=32)
```

The parameter *test_size=0.2* means that 20% of the data is used for testing, while 80% is used for training. The *random_state=32* parameter ensures consistent data distribution each run.

Next, the model is trained using the *fit()* method on the training data. This step allows the model to learn patterns from the data to produce accurate predictions.

After the model is initialized, the next step is to train the model (Guna et al., 2023). However, before training the model, the dataset will be divided into training data and testing data using *train_test_split().* The following Python program code is used to split the dataset (Dewi et al., 2024).

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2, random_state=32).

The parameter *test_size*=0.2 means that 20% of the data is used for testing, while 80% is used for training. The *random_state=32* parameter ensures consistent data division each time it is run.

Next, the model is trained using the *fit()* method on the training data. This step allows the model to learn patterns from the training data to produce accurate predictions.

3.1.4. Evaluation Mode

Model evaluation is an important step to assess an algorithm's performance in predicting data in classification cases. In this research, the two algorithms compared are Naive Bayes and Decision Tree. Performance assessment is done using several evaluation metrics, such as accuracy, precision, recall, and f1-score. In addition, the evaluation is done using Confusion Matrix and Classification Report.

In the confusion matrix, there are four parameters as a representation of the results of the classification process, namely, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) (Amani et al, 2024) (Supriyadi, 2023).

- True Positive (TP): The amount of data that is truly positive and has been classified as or the number of correctly classified positive classes.
- True Negative (TN): The amount of data that is truly negative and has been classified as or the number of negative classes that are correctly classified.
- False Positive (FP): The amount of data that is truly negative but misclassified as positive or the number of negative classes that are classified as positive classes.

False Negative (FN): The amount of data that is truly positive but misclassified as negative or the number of negative classes that are correctly classified.

a. Naïve Bayes Model Evaluation

The Naive Bayes model that has been trained will be tested using independent testing data (Anwarudin et al., 2022). Referring to the previous description, the testing data used is 46. The confusion matrix for the Naive Bayes model can be presented in Figure 11.

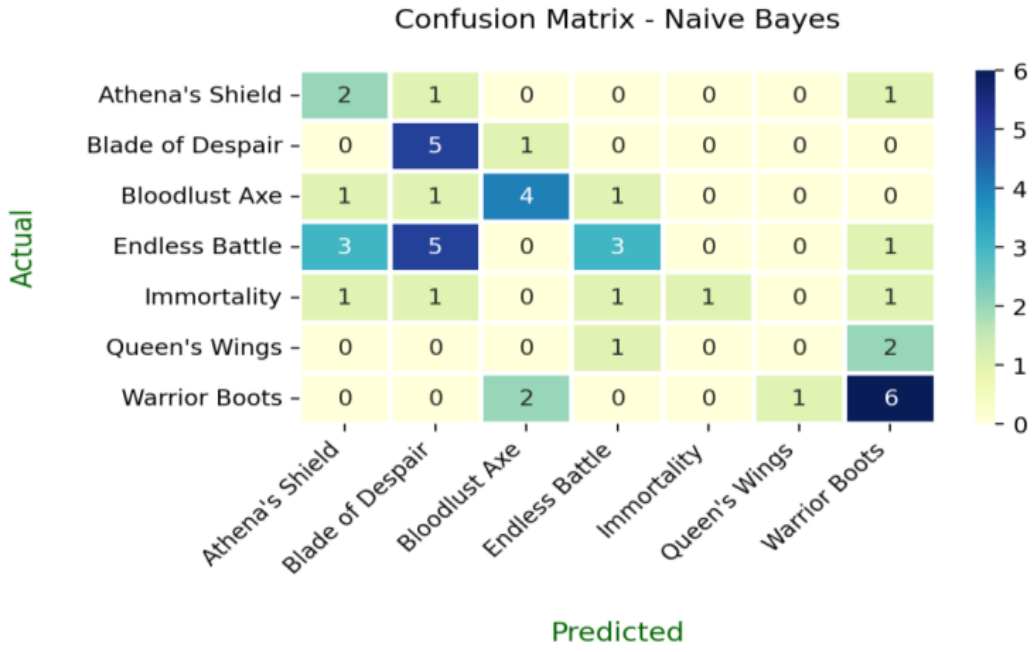

Figure 11. Confusion Matrix Naïve Bayes

The confusion matrix results of the Naive Bayes model shown in Figure 11 show the model's performance in classifying hero fighter items based on test data (Putri & Syarif, 2023). The rows in the matrix represent the actual label, while the columns represent the label predicted by the model. The following is a complete explanation of the matrix results:

- *Athena's Shield:* Out of the total data with the actual label "Athena's Shield," 2 samples were correctly classified. However, there was 1 sample misclassified as "Blade of Despair" and 1 sample misclassified as "Warrior Boots."
- *Blade of Despair:* Most of the data with the label "Blade of Despair" was classified correctly (5 samples), but there was 1 sample that was misclassified as "Bloodlust Axe."
- *Bloodlust Axe:* The model correctly classified 4 samples as "Bloodlust Axe," but 1 sample was incorrectly classified as "Athena's Shield," "Blade of Despair," and "Endless Battle," respectively.
- *Endless Battle:* From the label "Endless Battle," 3 samples were correctly classified. However, 5 samples were misclassified as "Blade of Despair," 3 as "Bloodlust Axe," and 1 as "Warrior Boots."
- *Immortality:* A total of 1 sample of the label "Immortality" was correctly classified, but 1 sample was misclassified to "Athena's Shield," "Blade of Despair," and "Warrior Boots" respectively.
- *Queen's Wings:* The model had difficulty with this label, where only 1 sample was correctly classified as "Queen's Wings," and 2 samples were incorrectly classified as "Warrior Boots."
- *Warrior Boots:* This label had the best results, with 6 samples correctly classified, but 2 samples incorrectly classified as "Bloodlust Axe."

b. Decision Tree Model Evaluation

The Decision Tree model that has been trained will be tested using independent testing data. Referring to the previous description, the testing data used is 46. The confusion matrix for the Decision Tree model can be presented in Figure 12.

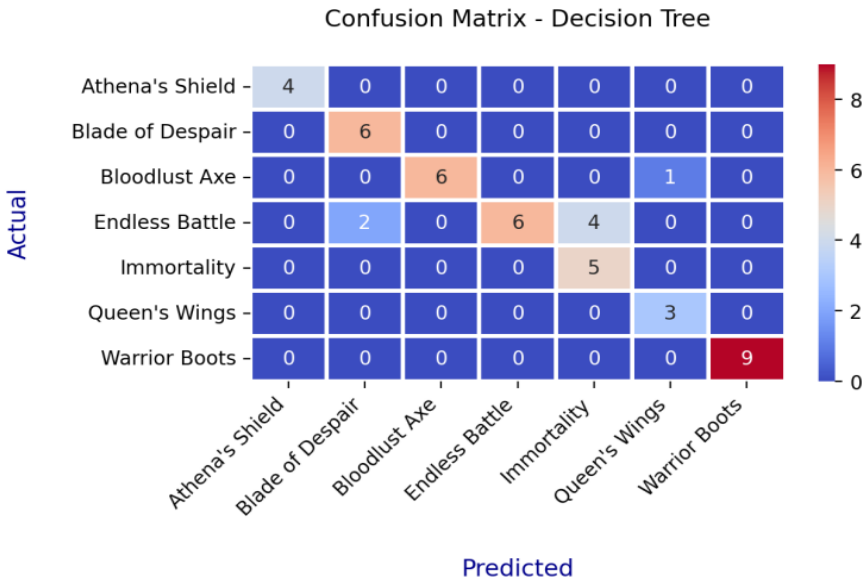

Figure 12. Confusion Matrix Decision Tree

Based on the confusion matrix of the Decision Tree model shown in the figure, the model's performance in classifying hero fighter items in the Mobile Legends game can be explained as follows:

- *Athena's Shield:* All 4 samples from this class were correctly classified by the model, showing perfect accuracy for this class.
- *Blade of Despair:* The model successfully classified 6 out of 6 samples correctly into the Blade of Despair class. This indicates optimal performance in this class without any misclassification.

- *Bloodlust Axe:* Out of a total of 7 samples, 6 were correctly classified into the Bloodlust Axe class, while 1 sample was classified into the Queen's Wings class. Despite one misprediction, the model performed quite well for this class.
- *Endless Battle:* Out of the 12 samples of this class, 6 were correctly classified into the Endless Battle class. However, there was a misclassification where 2 samples were predicted as Blade of Despair and 4 samples were predicted as Immortality. These errors indicate that the model experienced confusion in distinguishing between the Endless Battle class and other classes.
- *Immortality:* All 5 samples from the Immortality class were correctly classified to the appropriate class, showing perfect performance for this class.
- *Queen's Wings:* Out of the 3 Queen's Wings class samples, 3 were classified correctly, indicating that the model successfully identified all samples from this class without error.
- *Warrior Boots:* All 9 samples from this class were correctly classified into the Warrior Boots class, which also shows perfect accuracy for this class.

3.2. Discussion

The Naïve Bayes and Decision Tree models that have been evaluated using testing data will then compare the performance of the two models (Supriyadi, 2023). This is done to determine the best model in classifying hero fighter items in the Mobile Legends game (Atma & Yogyakarta, 2023). Comparisons are made based on evaluation metrics (accuracy, precision, recall, and f1-score). The comparison of performance metrics from models that have been evaluated can be displayed in graphical form in Figure 13.

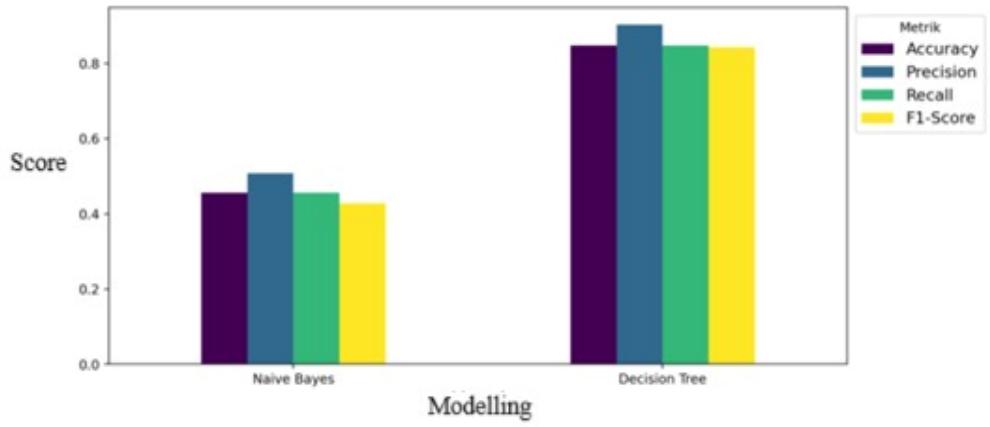

Figure 13 Model Comparison Chart

The Figure 13 shows the results of comparing the two models based on the evaluation metrics. Decision Tree shows better performance than Naive Bayes on almost all metrics. Based on Figure 13, the results of the performance comparison between the Naïve Bayes and Decision Tree models can be presented in Figure 14.

	Model	Accuracy	Precision	Recall	F1-Score
0	Naive Bayes	0.4565	0.5078	0.4565	0.4278
1	Decision Tree	0.8478	0.9028	0.8478	0.8423

Figure 14. Model Performance Comparison

Based on Figure 14, the following analysis of each metric is as follows:

- *Accuracy:* Decision Tree achieves an accuracy of about 84%, while Naive Bayes is only 45%.

- *Precision:* Precision of Decision Tree is much higher than Naive Bayes, especially in certain classes.
- *Recall:* Decision Tree also excels in detecting true positive cases compared to Naive Bayes.
- *F1-Score:* This value shows the balance between precision and recall, where Decision Tree still shows a more consistent performance.

Based on the evaluation results, the Decision Tree model has a better performance than Naive Bayes in classifying fighter items in the Mobile Legends game. Therefore, the Decision Tree model is chosen as the more optimal algorithm for this classification case.

4. Conclusion

The Mobile Legends game by comparing the Naïve Bayes and Decision Tree models, namely a model for the classification of hero fighter items in the Mobile Legends game was built using the Naïve Bayes and Decision Tree algorithms. The model utilizes a dataset that has gone through the preprocessing stage, division of data into training data and test data, and training with both algorithms. The model building process runs according to the designed method, resulting in a model that is able to predict the type of hero fighter item based on the given features. The performance comparison results show that the Decision Tree algorithm has better performance than Naïve Bayes in the classification of hero fighter items in the Mobile Legends game. This can be seen from the higher accuracy, precision, recall, and f1-score values on Decision Tree compared to Naïve Bayes.

Future research can consider using other algorithms such as Random Forest, Gradient Boosting, or deep learning-based algorithms to evaluate performance in hero fighter item classification. This will provide a more comprehensive picture of the most suitable algorithm and can be used as a reference for assessing model performance from another point of view. Further development is expected by applying this model to real applications in the game, so that players can get item recommendations automatically based on the game situation.

References

Alfudola, M., Suarna, N., & Ali, I. (2023). KLASIFIKASI PEMILIHAN TIPE HERO MOBILE LEGENDS TERHADAP MINAT PEMAIN MENGGUNAKAN ALGORIMA NAIVE BAYES STUDI KASUS : KOMUNITAS GAME MOBILE LEGENDS KOTA CIREBON. *JATI (Jurnal Mahasiswa Tektik Infomatika)*, *7*(2), 1269–1273.

Amani, N. N., Martanto, & Hayati, U. (2024). PENGGUNAAN ALGORITMA DECISION TREE UNTUK PREDIKSI PRESTASI. *JATI (Jurnal Mahasiswa Tektik Infomatika)*, *8*(1), 473–479.

Ananda, B. F., Achmad, Z. A., Alamiyah, S. S., Wibowo, A. A., & Fauzan, L. A. (2022). Jurnal Ilmu Komunikasi VARIASI KOMUNIKASI VIRTUAL PADA KELOMPOK PEMAIN GAME MOBILE LEGENDS. *Jurnal Ilmu Komunikasi*, *12*(1).

Andriska, B., Permana, C., Sadali, M., & Ahmad, R. (2024). Penerapan Model Decision Tree Menggunakan Python Untuk Prediksi Faktor Dominan Penyebab Penyakit Stroke. *Infotek : Jurnal Informatika Dan Teknologi*, *7*(1), 23–31.

Anwarudin, Andriyani, W., Purnomosidi DP, B., & Kristomo, D. (2022). The Prediction on the Students ' Graduation Timeliness Using Naive Bayes Classification and K-Nearest Neighbor. *Journal of Intelligent Software System – JISS*, *1*(1), 75–88. https://doi.org/10.26798/jiss.v1l1.597

Arrayyan, A. Z., Setiawan, H., & Putra, K. T. (2024). Naive Bayes for Diabetes Prediction : Developing a Classification Model for Risk Identification in Specific Populations. *Semesta Teknika*, *27*(1), 28–36.

Atma, U., & Yogyakarta, J. (2023). MOBILE LEGEND GAME PREDICTION USING MACHINE LEARNING RE-GRESSION METHOD. *JURTEKSI (Jurnal Teknologi Dan Sistem Informasi)*, *IX*(2).

Bayulianto, S., Purnamasari, I., & Jajuli, M. (2023). Prediksi tingkat kemenangan mobile legends profesional league indonesia season 9 dengan menggunakan algoritma naïve bayes. *JIPI (Jurnal Ilmiah Penelitian Dan Pembelajaran Informatika)*, *8*(2), 538–550.

Dewi, A., Safira, A., & Yamasari, Y. (2024). Penerapan Algoritma Naïve Bayes (NB) untuk Klasifikasi Penyakit

Jantung. *JINACS (Journal of Informatics and Computer Science)*, *05*, 447–455.

Ditendra, E., Romelah, S., Tanjung, M. H. A., & Sarah, M. (2022). Comparison of Classification Algorithms for Sentiment Analysis of Islam Nusantara in Indonesia Perbandingan Algoritma Klasifikasi untuk Analisis Sentimen Islam Nusantara di Indonesia. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, *2*(April), 71–77.

Farid, M., Wibowo, S., Puspitasari, N. F., & Satya, B. (2022). PENERAPAN DATA MINING DAN ALGORITMA NAÏVE BAYES UNTUK METODE KLASIFIKASI. *Journal of Information System Management (JOISM)*, *3*(2).

Fatma, Y. L., & Rochmawati, N. (2024). Prediksi Siswa Putus Sekolah Menggunakan Algoritma. *JINACS (Journal of Informatics and Computer Science)*, *05*, 486–493.

Fauzan, A. C., & Hikmah, K. (2022). IMPLEMENTASI ALGORITMA NAIVE BAYES DALAM ANALISIS POLARISASI OPINI MASYARAKAT TERKAIT VAKSIN COVID-19. *RABIT : Jurnal Teknologi Dan Sistem Informasi Univrab*, *7*(2), 122–128.

Guna, E. A., Ghifary, M. D. D., Sihombing, E. F., & Medan, U. N. (2023). Implementasi Algoritma Decision Tree untuk Klasifikasi Data Evaluation Car Menggunakan Python Ekin. *Jurnal Sistem Informasi Dan Ilmu Komputer*, *1*(4).

Habibah, D., Aini, N., Kurniasari, D., Nuryaman, A., Usman, M., & Lampung, U. (2023). IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORK WITH BACKPROPAGATION ALGORITHM FOR RATING CLASSIFICATION ON SALES OF BLACKMORES IN TOKOPEDIA. *Jurnal Teknik Informatika (JUTIF)*, *4*(2), 365–372.

Hajaroh, Suprapti, T., & Narasati, R. (2024). IMPLEMENTASI ALGORITMA NAIVE BAYES UNTUK ANALISIS SENTIMEN. *JATI (Jurnal Mahasiswa Tektik Infomatika)*, *8*(1), 111–118.

Hariyanti, I., Al-husaini, M., & Raharja, A. R. (2024). Perbandingan Algoritma Decision Tree dan Naive Bayes dalam Klasifikasi Data Pengaruh Media Sosial dan Jam Tidur Terhadap Prestasi Akademik Siswa. *Technologia : Jurnal Ilmiah*, *15*(2), 332–340.

Ikko, I., Rizky, M., Yusuf, S., & Sriyanto, I. (2023). Perbandingan Kinerja Algoritma Naive Bayes , Support Vector Machine dan Random forest untuk Prediksi Penyakit Ginjal Kronis. *Institut Informatika Dan Bisnis Darmajaya*, *18*, 139–151.

Iqbal, M., Wiranata, A. D., Suwito, R., Ananda, R. F., Muhammadiyah, U., & Hamka, P. (2023). Perbandingan Algoritma Naïve Bayes , KNN , dan Decision Tree terhadap Ulasan Aplikasi Threads dan Twitter. *KLIK: Kajian Ilmiah Informatika Dan Komputer*, *4*(3), 1799–1807. https://doi.org/10.30865/klik.v4i3.1402

Irfon, G., & Soen, E. (2022). Implementasi Cloud Computing dengan Google Colaboratory Pada Aplikasi Pengolah Data Zoom Participants. *JITU: Journal Informatic Technology And Communications*, *6*(1), 24–30.

Lile, X. F. K., & Suharjo, I. (2024). OPTIMALISASI PERBANDINGAN ALGORITMA K-NEAREST NEIGHBORS DAN DECISION TREE UNTUK PREDIKSI KEMENANGAN DI MPL SEASON 13 MOBILE LEGEND. *JATI (Jurnal Mahasiswa Tektik Infomatika)*, *8*(4), 7553–7560.

Macfud, A. Z., Kusuma, A. P., & Puspitasari, W. D. (2023). ANALISIS ALGORITMA NAIVE BAYES CLASSIFIER (NBC). *JATI (Jurnal Mahasiswa Tektik Infomatika)*, *7*(1), 87–94.

Moerdyanto, O. P., & Nuryana, I. K. D. (2023). Prediksi Kelulusan Tepat Waktu Menggunakan Pendekatan Pohon Keputusan Algoritma Decision Tree. *JINACS (Journal of Informatics and Computer Science)*, *05*, 90–96.

Mukhsinin, D. A., Rafliansyah, M., & Ibrahim, S. A. (2024). Implementation of Decision Tree Algorithm for Movie Recommendation and Rating Classification on the Netflix Platform Implementasi Algoritma Decision Tree untuk Rekomendasi Film dan Klasifikasi Rating pada Platform Netflix. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, *4*(April), 570–579.

Nurnawati, E. K. N., Sholeh, M., Ariyana, R. Y., & Almuntaha, E. (2023). Jurnal TAM (Technology Acceptance Model) COMPARISON OF DECISION TREE AND NAÏVE BAYES ALGORITHMS IN CLASSIFICATION MODELS TO DETERMINE LECTURER PERFORMANCE USING K FOLD CROSS. *Jurnal TAM (Technology Acceptance Model)*, *14*(2), 152–157.

Padilah, A., Perdana, H., & Aprizkiyandari, S. (2024). IMPLEMENTASI METODE NAÏVE BAYES DALAM PREDIKSI TINGKAT KEMENANGAN PADA GAME MOBILE LEGENDS. *Buletin Ilmiah Math. Stat. Dan Terapannya (Bimaster)*, *13*(4), 437–446.

Permana, A. P., Ainiyah, K., Fahmi, K., & Holle, H. (2021). Analisis Perbandingan Algoritma Decision Tree , kNN , dan Naive Bayes untuk Prediksi Kesuksesan Start-up. *JISKA*, *6*(3), 178–188.

Punkastyo, D. A., Septian, F., & Syaripudin, A. (2024). Implementasi Data Mining Menggunakan Algoritma Naïve Bayes Untuk Prediksi Kelulusan Siswa. *Journal of System and Computer Engineering (JSCE)*, *5*(1), 24–35.

Purnomo, A. J., & Abdul, R. (2022). CLASSIFICATION OF STUNTING STATUS IN TODDLERS USING NAIVE BAYES METHOD IN THE CITY OF MADIUN. *Techno Nusa Mandiri: Journal of Computing and Information Technology*, *19*(2), 69–76.

Putri, A. I., & Syarif, Y. (2023). Implementation of Decision Tree and Support Vector Machine (SVM) Algorithm for Stunting Risk Prediction Implementasi Algoritma Decision Tree dan Support Vector Machine (SVM) untuk Prediksi Risiko Stunting pada Keluarga. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, *3*(October), 349–357.

Rachmat, V., Hidayat, A., Findawati, Y., & Sumarno, S. (2024). Sistem Prediksi Kemenangan Hero Mobile Legends Menggunakan Metode Naive Bayes Mobile Legends Hero Winning Prediction Using Naïve Bayes. *JIKO (Jurnal Informatika Dan Komputer)*, *1*, 100–116. https://doi.org/10.26798/jiko.v8i1.1120

Ramadhani, & Ramadhanu. (2024). Metode Machine Learning untuk Klasifikasi Data Gizi Balita dengan Algoritma Naïve Bayes , KNN dan Decision Tree. *Jurnal SIMETRIS*, *15*(1), 57–68.

Ramadhon, R. N., Ogi, A., & Agung, A. P. (2024). Implementasi Algoritma Decision Tree untuk Klasifikasi Pelanggan Aktif atau Tidak Aktif pada Data Bank. *Karimah Tauhid*, *3*, 1860–1874.

Supriyadi, A. (2023). Perbandingan Algoritma Naive Bayes dan Decision Tree. *Generation Journal*, *7*(1), 39–49.

Tanza, A., & Utari, D. T. (2022). Comparison of the Naïve Bayes Classifier and Decision Tree J48 for Credit Classification of Bank Customers. *EKSAKTA:Journal of Science Data Analysis*, *3*(2), 70–77. https://doi.org/10.20885/EKSAKTA.vol3.iss