DISCORD BOT REPORT

Presented by: ELGHALI BENJELLOUN

Supervised by: ALEXANDRE MAURER

1.1 Project Context:

The project emerges within a Discord server undergoing a crucial transition. The advanced management of this virtual space requires a solution adapted to the evolving reality. Our Discord bot is designed as a versatile response to these challenges, introducing advanced member management and an innovative approach to moderation. Beyond mere automation, our bot embodies true artificial intelligence through the integration of machine learning.

1.2 Problem to Solve:

The genesis of this project originated in the drastic reduction in the number of moderators, from 5 or 6 to one person, or even zero. This development poses a major challenge to the stability and quality of moderation, especially in an increasingly dynamic Discord environment. The main challenge is to set up a system capable of detecting, understanding, and reacting to undesirable behavior, particularly through the detection of inappropriate words. All this aims to create a respectful environment, in line with the rules established by the community, despite the small number or absence of human moderators. This ambition requires an innovative approach that combines advanced automation, proactive member management, and intelligent machine learning integration.

1.3 Implemented Solutions:

Proactive member management is a key component of the solution. Warnings and penalties are automatically applied, providing an immediate and consistent response to undesirable behavior. This preventative approach aims to discourage rule violations and build a culture of respect within the community. The most innovative element lies in the integration of a machine learning model. This intelligent addition allows the bot to perform automated detection of unwanted behavior, forming a proactive line of defense against any harmful activity. With this scalable approach, the bot can adjust its responses over time, anticipating and adapting its moderation based on changes in server dynamics.

1.4 Challenges Encountered:

Completing the project was not without its challenges. Integrating the machine learning model required a deep understanding of the concepts and associated libraries. Consistently managing warnings and sanctions, while maintaining the stability of the bot, was a complex challenge. Carefully navigating through interactions on the Discord server to ensure an optimal experience was also an additional challenge. The balance between the bot's varied functionality and its moderation role required careful planning to ensure a seamless and enjoyable user experience.

2. Main.py File:

The "main" page presents the complete code of the Discord bot built with the Discord.py library. The project aims to solve the urgent problem of decreasing the number of moderators on a Discord server from 5 or 6 to one person, or even zero.
The bot offers a variety of features, including generating random quotes, proactively managing members with automated warnings and sanctions, and integrating a machine learning model to automatically detect unwanted behavior.
Commands such as $inspire to display a random quote, $spam to repeat a message, and $stop to stop the bot with data backup, are implemented. Moderation commands such as $unban and $unwarn can be used to lift bans and reduce warnings, respectively.
The $initiate_database command initializes the member database, and $list_members displays a list of members

3.1 Users.py File:

The given Python code defines two classes, `Server` and `User`, to manage user data in a server by storing user information in a JSON file (actually two JSON files: one containing the information of the people currently in the server and the other containing all the people who have ever been in the server, whether they have left or been banned).

Class `Server`

The `Server` class handles the storage and retrieval of data using JSON files. The main methods are as follows:

- `__init__`: Initializes the server and loads existing data.
- `save_data_to_file`: Saves data to JSON files.
- `load_data_from_file`: Loads data from JSON files.
- `load_data`: Attempts to load data and initializes empty dictionaries if the files do not exist.
- `save_data`: Cleans up the data, excluding methods, and saves it to JSON files.

Class `User`

The `User` class represents user data with attributes such as ID, username, game time, warnings, bans, money, and behavior. The main methods are as follows:

- `__init__`: Initializes a user with default values.
- `to_dict`: Converts the user's data into a dictionary.
- `add_game_playtime`: Adds game time for a specific game.
- `remove_game`: Removes a game from the game time dictionary.
- `add_behave`: Manages user behavior, adding warnings and resetting the behavior.
- `add_warning`: Adds warnings and bans the user if the warning threshold is reached.
- `ban_user` and `unban_user`: Define the `is_banned` attribute.
- `get_total_playtime` and `get_game_playtime`: Retrieve game time information.
- `__str__`: Returns a textual representation of the user.

```python
class Server:
    def __init__(self):
        self.player_data = {}
        self.people = {}
        self.load_data()
```

(a) Server Initialization

```python
class User:
    def __init__(self,user_id: int,username,games_played={},warning=0,
                 is_banned=False,money=0,words={},role="member",behave=0):
        self.user_id = user_id
        self.games_played = games_played # Dictionary to store game playtime
        self.warnings =warning
        self.is_banned = is_banned
        self.money=money
        self.behave=behave
        self.username=username
        self.words=words
        self.role =role
```

(b) User Initialization

3.2 Management of Warnings/Bans:

In line with our server rules, we have implemented a system of warnings and bans aimed at maintaining a positive community atmosphere. Users receive warnings for rule violations, and three warnings can result in a temporary or permanent ban. Additionally, our automated system issues a warning after two incidents of inappropriate behavior are detected. We prioritize transparency and fairness in our moderation actions, and users can contact the moderation team with concerns or appeals. This approach helps create a welcoming environment for all community members.

4 Explanation of the TF-IDF Model:

The Term Frequency-Inverse Document Frequency (TF-IDF) model is a numerical statistic that reflects the importance of a term in a document relative to a collection of documents (corpus). It is commonly used in information retrieval and text mining.

4.1 Term Frequency Matrix (TF):

The term frequency matrix, denoted TF, represents the frequency of each term in each document. Each element TF_{ij} in the matrix corresponds to the frequency of term j in document i.

$$\text{TF}_{ij} = \frac{\text{Number of occurrences of term } j \text{ in document } i}{\text{Total number of terms in document } i}$$

Document	**Term 1**	**Term 2**	**Term 3**	**Term 4**
Document 1	0.2	0.3	0.1	0.4
Document 2	0.1	0.5	0.2	0.2
Document 3	0.3	0.2	0.4	0.1

Table 1: Term Frequency Matrix (TF)

4.2 Inverse Document Frequency Matrix (IDF):

The inverse document frequency matrix, denoted IDF, measures the importance of each term across the entire corpus. It is calculated as the logarithm of the ratio between the total number of documents N and the number of documents containing term j.

$$\text{IDF}_j = \log\left(\frac{N}{\text{Number of documents containing term } j}\right)$$

Term	IDF
Term 1	0.2
Term 2	0.5
Term 3	0.3
Term 4	0.4

Table 2: Inverse Document Frequency Matrix (IDF)

4.3 TF-IDF Product Matrix:

The TF-IDF product matrix is obtained by multiplying the term frequency matrix (TF) by the inverse document frequency matrix (IDF). Each element TF-IDF_{ij} represents the importance of term j in document i.

$$\text{TF-IDF}_{ij} = \text{TF}_{ij} \times \text{IDF}_j$$

Document	Term 1	Term 2	Term 3	Term 4
Document 1	0.04	0.15	0.03	0.16
Document 2	0.02	0.25	0.06	0.08
Document 3	0.06	0.1	0.12	0.04

Table 3: TF-IDF Product Matrix

Table 3 shows the TF-IDF product matrix, where rows represent documents and columns represent terms. Each cell contains the product of the corresponding values of term frequency and inverse document frequency.

Feature Importance: Terms with higher TF-IDF values are considered more important features for distinguishing documents. These terms contribute more to the overall characterization of a document and are often used as inputs for machine learning models. Document Similarity: TF-IDF can be used to measure similarity between documents. Documents with similar TF-IDF profiles (i.e., similar importance of terms) are considered more similar in content.

5. Random Forest Classification:

The Random Forest classifier is an ensemble learning method that combines predictions from multiple decision trees. Each tree is constructed independently, and randomness is introduced through bootstrap sampling and the random selection of features. The ensemble of trees works together to make predictions, and the final classification decision is often determined by a majority vote.

In the context of our TF-IDF model, the TF-IDF features obtained from the TF-IDF product matrix serve as input for the Random Forest classifier. This model is trained on a labeled dataset where each document is associated with a binary label indicating whether it is disrespectful or not.

5.1 Exemplary Prediction Results

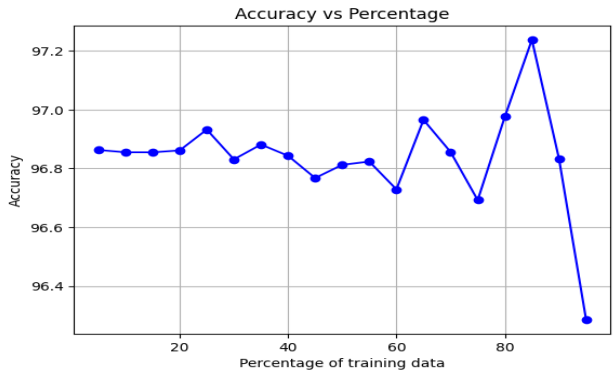
Suppose we have a trained Random Forest model, and we want to present some prediction results on new, unseen documents.

Document	**Predicted Class**	**Actual Class**
Document 4	Disrespectful	Disrespectful
Document 5	Respectful	Respectful
Document 6	Disrespectful	Respectful

Table 4: Random Forest Classification Results

Table 4 presents exemplary prediction results for three documents using the trained Random Forest model. The "Predicted Class" column indicates the model's prediction, and the "Actual Class" column shows the actual class labels.

6. Best Training/Test Ratio:



Through a series of 100 trials to determine the best accuracy between the training and test sets on our dataset, we observed that the average of these trials reached a maximum accuracy of 85

7. Accuracy Test:

Some tests of our dataset, taking 15

```python
from dataset import data
from nlpimport import Prediction
from DATAtoTRAIN import train

def test1(test=[i for i in data if i not in train]):
    count=0
    for i in test :
        if Prediction(i[0])==i[1] :
            count+=1
    if __name__=="__main__":
        print("Accuracy :",count/len(test)*100,"%")
    return count / len(test) * 100

test1()
```

(a) Code of the test

```
Accuracy : 96.82539682539682 %
Accuracy : 95.23809523809523 %
Accuracy : 95.23809523809523 %
Accuracy : 98.4126984126984 %
Accuracy : 93.65079365079364 %
Accuracy : 98.4126984126984 %
Accuracy : 95.23809523809523 %
```

(b) Some outputs of the presented code

8. Conclusion:

The development of the Discord bot within the UM6P-CS department of the university has been a significant undertaking, aimed at solving the critical

challenge of maintaining effective moderation despite a marked reduction in the number of moderators. The bot presents an innovative approach, combining advanced automation, proactive member management, and intelligent integration of machine learning.