



EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF SOFTWARE TECHNOLOGY

Mastodon social media threat alert extension

Supervisor:

Zoltán Gera

tanársegéd, programozó matematikus MSc

Author:

Kajdomcaj Dion

Computer Science BSc

Budapest, 2022

Thesis Registration Form

Student's Data:

Student's Name: Kajdomcaj Dion

Student's Neptun code: E9TZ67

Course Data:

Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: Gera Zoltán

Supervisor's Home Institution:

ELTE Faculty of Informatics

Address of Supervisor's Home Institution:

1117 Pázmány Péter stny 1/C

Supervisor's Position and Degree:

egyetemi tanársegéd, programozó matematikus MSc

Thesis Title: Mastodon Social Media Threat Alert Extension

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis of your planned thesis.)

Social media thread alert extension is an extension which is planned to improve the security of Mastodon social media platform by letting the users know when there is a possibility for a threat towards them in this social media. This extension will provide social media threat alert for the users, for example when an account texts you it will detect and let you know whether that account has a possibility of being a fake account and afterwards will allow you to decide whether you want to be still connected with that account or not. Besides this it will prevent these kind of users getting the attention they seek in this social media and by that it will start to get them off the network since they will not get the attention they seek for in it, other than this it will encourage students and professors to use their university social media even more than they actually do.

Budapest, 2021. 11. 30.

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Thesis structure	3
2	User documentation	4
2.1	Project Description	4
2.2	Installation guide	4
2.2.1	Running the application	5
2.3	Logging in	6
2.3.1	Valid log in data	7
2.3.2	Invalid log in data	8
2.4	Actions for the possible threat account	9
2.4.1	Account action	12
2.4.2	Domain action	13
2.5	Exiting the application	14
3	Developer documentation	15
3.1	Mastodon API	15
3.1.1	API rate limits	16
3.1.2	API initialization	16
3.1.3	API instance	17
3.1.4	Notifications	18
3.1.5	Getting account's data	19
3.1.6	Getting user's following accounts	20
3.1.7	Blocking an account	21
3.1.8	Muting an account	21
3.1.9	Blocking a domain	21
3.2	Predictive Model	22

3.2.1	Data preprocessing	22
3.2.2	Model performance	23
3.3	Database	26
3.3.1	Handled accounts table	27
3.3.2	Handled domains table	27
3.4	GUI	27
3.5	Testing	27
4	Conclusion	28
	Bibliography	29
	List of Figures	30
	List of Tables	31
	List of Codes	32

Chapter 1

Introduction

1.1 Motivation

Nowadays we all know someone who has been affected negatively by the bullying happening on social media, which is becoming a very common problem in our society. Sometimes those cases escalated to the point where it turned into a serious mental health problem for the ones who have been affected. I was always wondering for a way to help people prevent similar cases and feel safe while surfing the social media feed. Everyone has it's rights to feel safe while doing a certain activity, but bringing safety to the social media in these kind of cases has been neglected throughout these last years.

The main reasons mentioned before have pushed me to brainstorm an idea to help people feel safe on Mastodon, which is open source, and enjoy the beauty of connecting with real profiles and use the social media to the point where it changes their life for good.

1.2 Thesis structure

This thesis consists of 4 chapters, which help users understand the installation steps of the software, the correct way of using the software and the possible behavior that can lead to errors, as well as help developers understand the functionality behind the software, the software architecture and the testing made on the software. Besides those main chapters it also contains a bibliography, a list of figures, a list of codes and a list of tables.

Chapter 2

User documentation

In this chapter we will discuss the installation steps, the correct way of using the software and a brief description about the software.

2.1 Project Description

My project is a desktop application that is meant to run on the background while using Mastodon social media. The whole project was built using the latest *Python* 3 version. The main goal of the project is to detect possible threats coming from other accounts in form of direct messages and tags, after warning the user about possible threats it let's the user decide the kind of action he wants to take against the account that may be a threat and the domain where the account came from.

As a prerequisite for using this desktop application is a stable internet connection and a Mastodon account.

2.2 Installation guide

As earlier mentioned, in order to use Mastodon social media threat alert application we need to have a stable internet connection and a Mastodon account in any server and *Python* 3.10.2.

The application is currently supporting Windows and Linux but the goal is to extend it as a mobile application which supports IOS and Android. Hence, the installation steps are the same for both, Windows and Linux, but we will go through the steps in Windows specifically.

To download the application we need to clone the following repository:
<https://github.com/DionKajdomcaj/Mastodon-Social-Threat-Alert.git>.

Prior to cloning the repository we need to make sure that we have git. If git is missing, the user can download it at the following url: <https://git-scm.com/download/win>
If the git prerequisite is met, we can clone the repository by entering the following command in the command prompt:

```
1 C:\Users\dionk>git clone https://github.com/DionKajdomcaj/  
Mastodon-Social-Threat-Alert.git
```

Code 2.1: Cloning Repository

After succeeding to clone the repository we need to install the requirements for our environment. In order to install the requirements we need to make sure we have pip command. If the pip command is not installed, the user can install it by clicking the following url: <https://phoenixnap.com/kb/install-pip-windows>.

In the application folder there is a file called requirements, installing it will download all the libraries needed to run the application.

We can install them by using the following command in command prompt:

```
1 C:\Users\dionk\Mastodon-Social-Threat-Alert>pip install -r  
requirements.txt
```

Code 2.2: Installing requirements

Now we are ready to run the application.

2.2.1 Running the application

In order to run the application we need to make sure that we are in the correct directory and then run the following command:

```
1 C:\Users\dionk\Mastodon-Social-Threat-Alert>python ThreatAlert.py
```

Code 2.3: Running the application

After running the command, if all the prerequisites are met, we can see our application log in page.

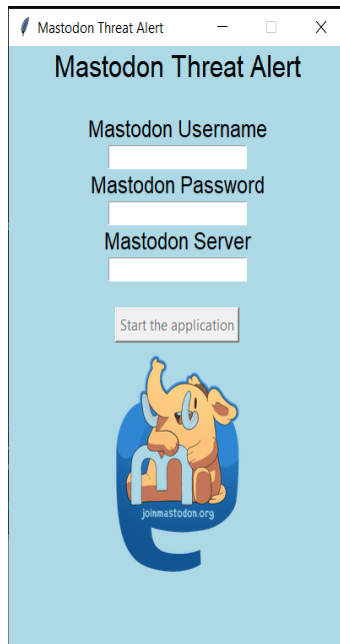


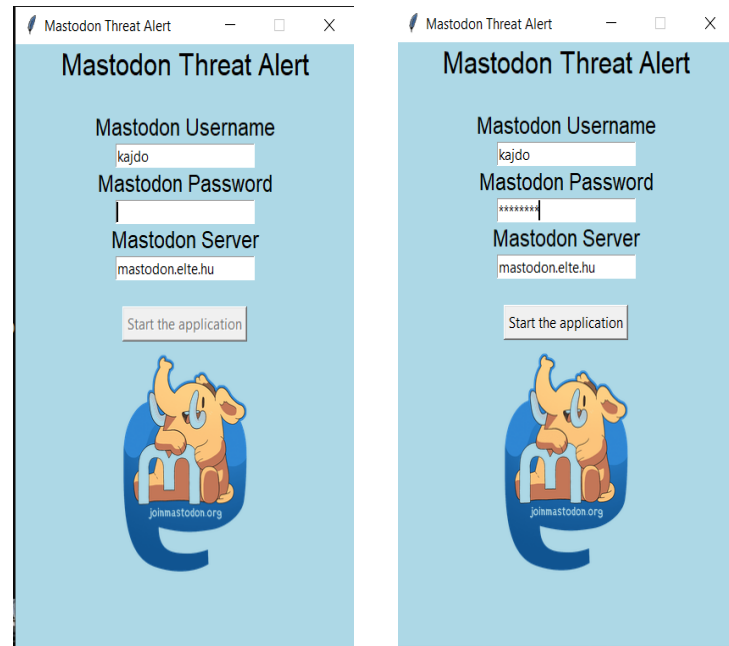
Figure 2.1: Application main page

2.3 Logging in

As we saw in Figure 2.1 the button to actually start the application is disabled. In order to enable it we must fill all the necessary fields which are:

- **Mastodon username**, which is your original Mastodon account username.
- **Mastodon password**, which is your original Mastodon account password, that you use to log in to Mastodon.
- **Mastodon server**, which is the server your account is currently registered.

If only one of them is missing then the user will not be able to start the application nor use it.



The figure consists of two side-by-side screenshots of a web application window titled "Mastodon Threat Alert". Both screenshots show a light blue background with a cartoon elephant logo at the bottom that says "joimastodon.org". The form contains three input fields: "Mastodon Username" with the value "kajdo", "Mastodon Password", and "Mastodon Server" with the value "mastodon.elte.hu". Below the fields is a button labeled "Start the application". In screenshot (a), the button is disabled (greyed out) because the password field is empty. In screenshot (b), the button is enabled (blue) because the password field now contains masked characters (*****).

(a) Missing password field -
Button disabled

(b) No missing field -
Button enabled

Figure 2.2: Button disabled and enabled

The fields must be filled with the Mastodon account information. In the username field you must enter your Mastodon username, and the same goes up to password. About the server you need to know which server are you in and only type the domain, for example in case there is an account like: `example@mastodon.social` then the username is `example` and the server is `mastodon.social`. All of the data are case sensitive, so you must give them exactly as they are originally.

2.3.1 Valid log in data

If every data is correct then the application will be connected to the Mastodon API after we click on the button. The program will let us now that it is running, and it will not change it's state until it recognizes a possible threat for the logged in user.

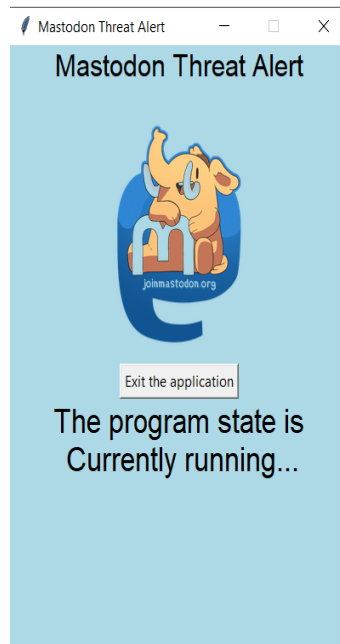


Figure 2.3: Application running

2.3.2 Invalid log in data

Even if the button is enabled it does not mean that the log in data entered by the user are correct. So, if the user does not give the valid log in data then the application will not be connected to Mastodon API. Hence, it will give the user an error message. The following figure is going to show you the message you will receive for giving invalid log in data. After receiving the message you can just press the message button and try again as many times as you need.

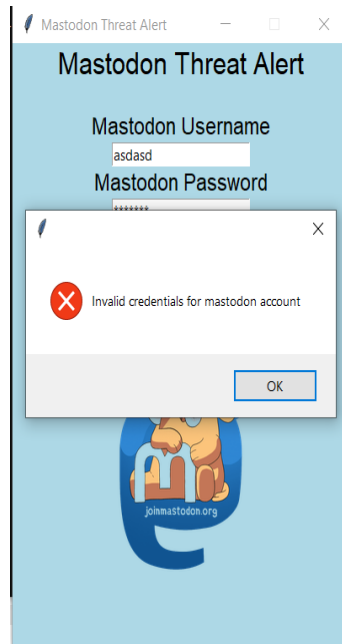


Figure 2.4: Incorrect log in data

2.4 Actions for the possible threat account

Now that we logged in successfully, we can start using Mastodon as usually, but this time we have the Mastodon threat alert application running on the background and looking for possible threats.

Every time that we are going to receive a direct message or a tag notification, that account's data is going to be checked whether it has a possibility to be a threat or not. After checking if the account, that was trying to reach you, is considered to be a possible threat, the application will show you a warning message containing the possible threat account's username and domain, and will ask you to take a certain action against the account.

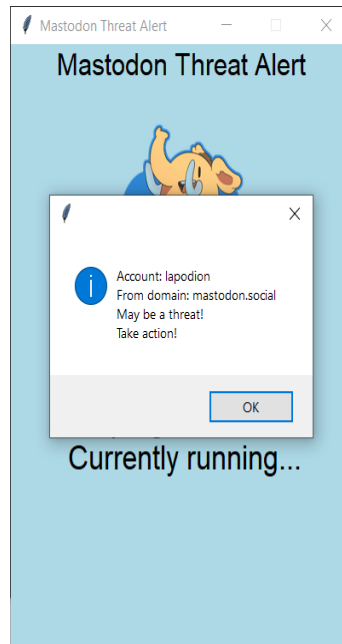


Figure 2.5: Possible threat notification

We have three kind of actions supported in our application depending whether we want to take them against an account or a domain. These actions are

- **Trust**
- **Block**
- **Mute**

The default value for both of them is Trust, which can be changed.

We have to simply choose the action from a combo box for both, possible threat account and it's domain, and click the button in order to perform the actions.

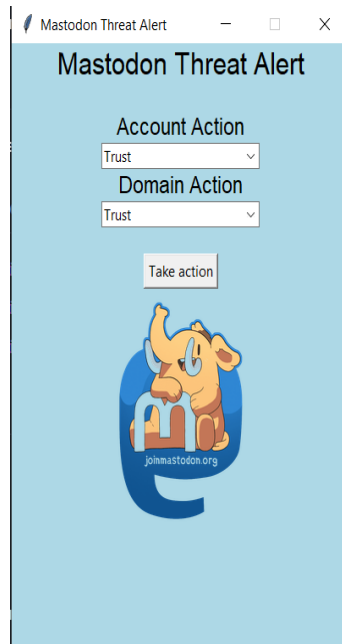
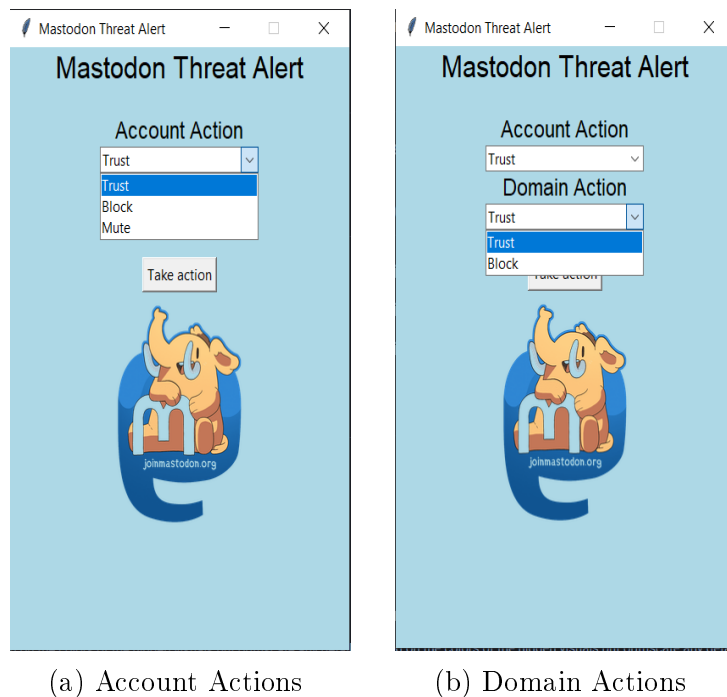


Figure 2.6: Action window

Now we can choose independently the type of action for both, the account and its domain.



(a) Account Actions

(b) Domain Actions

Figure 2.7: Actions available for account and domain

When we click the button, a pop up window will show up, letting you know that the actions were successful.

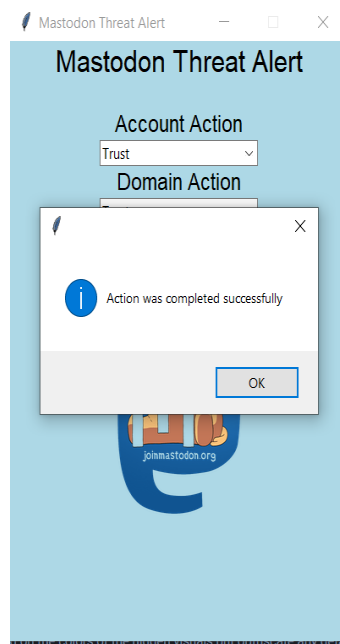


Figure 2.8: Pop up window to confirm the actions

After closing the pop up window, the application will go back to it's running state as in Figure 2.3.

2.4.1 Account action

As it was mentioned before, we can choose the actions separately for the possible threat account and it's domain. In this section we are going to talk about the possible threat account's action.

We have 3 types of actions for the account, which are: Trust, Block, Mute. The following table will describe each action.

Account Actions	
<i>Trust</i>	This action is very simple in context. When the user takes this function it means that he is trusting the possible threat account, and the application will not check the same account again, if it is trying to reach the user again.

<i>Block</i>	This action blocks the possible threat account, meaning it does not let the account reach the user anymore, not only by direct messaging or tagging, but even following, liking or any other social media activity. Besides those things the user will not see any activity from the possible threat account. To sum it up, the possible threat account will be non existent for the user.
<i>Mute</i>	Muting an account is same as ignoring the account, since the user will not receive notifications from the possible threat account anytime it tries to reach the user, but the user will still be able to check it's activity and the possible threat account will be able to try and reach the user, but simply the user will not be notified.

Table 2.1: Description of every action that can be taken against an account

However, every action that has been taken against the possible threat account can be easily reverted in Mastodon.

2.4.2 Domain action

In case of the possible threat account's domain we have less actions that can be taken against it.

The actions are Trust and Block. They might seem the same as the account actions described in table 2.1, but blocking a domain is completely different, since here we are working with the whole domain. However, in case of Trust, the functionality is the same as it was in the possible threat account's action.

In the following table you can understand their functionality.

Domain Actions	
<i>Trust</i>	This action works the same way as it works to trust an account. However, trusting the domain will not take any actual action against the domain. So, simply the accounts that are from that domain can freely reach the user in any form and without any restrictions.
<i>Block</i>	In case of Block the whole domain is blocked, meaning that any account that belongs in that domain can't reach the user. But, the user should always be very careful and is recommended not to block domains unless he/she was disturbed many times by the accounts coming from the same domain.

Table 2.2: Description of every action that can be taken against a domain

Same as in the account actions, the domain actions can be reverted in Mastodon.

2.5 Exiting the application

As we can see in figure 2.3, we can exit the application in two ways, by clicking the button named *Exit the application* or by simply closing the window. It is recommended to exit the application by clicking the button because it is terminated safely, but sometimes when we want to exit it while we are not on the running page, we can exit the application by closing the window as well. However, it is not recommended to close the application while taking actions, like in figure 2.6, because the possible threat account's will not be saved in the database.

Chapter 3

Developer documentation

In this chapter, we are going to get into details about the software architecture, the solutions to certain problems encountered during implementation and how everything came up together in the end.

In order to build and run the application as a developer, you must follow the same steps as in 2.2.

As it was mentioned before the application is completely connected to Mastodon [1]. So, it is useless and not the correct way to use it without using Mastodon.

3.1 Mastodon API

As it was mentioned in 2.1, this application check whether an account that is trying to reach you is a possible threat or not. To do this checking we need the data of the account. So, we have to connect to a API that get the data from the live Mastodon server in order to check the possibility of a threat account. Other than that, the API is used to let us take the actions mentioned in 2.4 directly from our application, without the need of doing it through Mastodon. For this application we used an already created API for Python called *Mastodon API* [2].

Below we can see a scheme of how our this API operates.

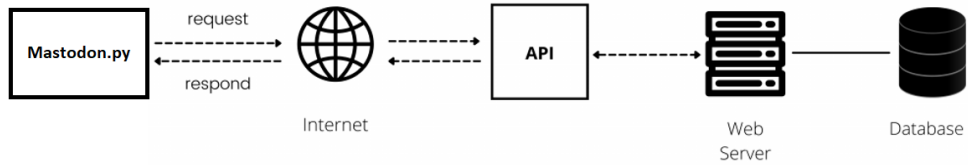


Figure 3.1: Mastodon API scheme

3.1.1 API rate limits

This API has a fixed rate-limit that is 300 requests per 5 minutes, in order not to violate the rate-limit and make the user wait for 5 minutes until the rate-limit resets, we needed to get data from the live server every 2.4 second so the user can get the full experience of Mastodon. This being said, user has enough time to surf the feed without having to wait for 5 minutes.

3.1.2 API initialization

To use this API we need to use the user's credentials to log into Mastodon account through API, and those credentials will be saved in `.secret` files in the directory where we call the API, and that is the reason why the user must give the credentials in the starting of the application. Below is a the snippet for logging in to the Mastodon account through API.

```
1  def __init__(self):
2      self._user = Mastodon()
3      self._userApiInstance = Mastodon()
4      self._mastodonServer = ""
5
6  def createApp(self, mastodon_server):
7      if(mastodon_server.split("://")[0] != "https"):
8          self._mastodonServer = "https://" + mastodon_server
```

```
9     else:
10         self._mastodonServer = mastodon_server
11
12     Mastodon.create_app(
13         "mastodonApiAppUser",
14         api_base_url=self._mastodonServer,
15         to_file='app/secretFolder/mastodonApiAppUser.secret'
16     )
17
18     def setUpAccounts(self):
19         self._user = Mastodon(
20             client_id='app/secretFolder/mastodonApiAppUser.secret',
21             api_base_url=self._mastodonServer
22         )
23
24     def loginAccount(self, username, password, user=True):
25         if user:
26             self._user.log_in(
27                 username,
28                 password,
29                 to_file='app/secretFolder/usercredentials.secret'
30             )
```

Code 3.1: Logging in to Mastodon account through API

3.1.3 API instance

Logging in to the account is not enough in order to use this API, but we can say it is a very important step and a must. To actually start using the API we need to create an API instance for that account. Below you can see the snippet of creating the API instance for both user account or admin account.

```
1     def createApiInstance(self):
2         self._userApiInstance = Mastodon(
3             access_token='app/secretFolder/usercredentials.secret',
4             api_base_url=self._mastodonServer
5         )
```

Code 3.2: Creating API instance

So, we have to create a access token from our the user credentials saved in *.secret* file, and after creating the API instance we can start using the API methods. The main methods that this application uses are:

- **Get the notifications**
- **Clear the notifications**
- **Block an account**
- **Block a domain**
- **Get certain account's data**
- **Get user's following accounts**
- **Mute an account**

We will deep dive into each of the methods in the following sections.

3.1.4 Notifications

To get the accounts that are trying to reach the user, we first need to get the notifications and filter them to only get the direct messages and tags, which is done by setting the parameter named **mentions** to true. To get the notifications through the API we can use the following method:

```
1 def getNotifications(self):  
2     return self._userApiInstance.notifications(mentions_only=True)
```

Code 3.3: Getting the notifications by API

And in order to filter them we need to go through each and filter them by their type and existence in the applications database.

```
1 def startSession(self):  
2     try:  
3         notifications = self.api.getNotifications()  
4         accounts_reaching_user = []  
5         for notification in notifications:
```

```
6         account_id = notification['account']['id']
7
8         inDatabase = self.isAccountInDatabase(int(account_id))
9
10        if (account_id not in accounts_reaching_user and not
11            inDatabase):
12
13            accounts_reaching_user.append(account_id)
14
15        return accounts_reaching_user
16    except Exception:
17        return []
```

Code 3.4: Filtering the notifications

3.1.5 Getting account's data

As it was mentioned in 3.1 we need to get the account's data in order to check for the possibility of the threat. To get the data of a certain account, we need that account's id and we can get public data for that account such as:

- Account ID
- Username
- Domain
- Followers Count
- Followings Count
- Statuses count
- Avatar
- Header
- Date of creation

And in order to fetch those data we can simply call the API method which is as below:

```
1  getAccountData(self, account_id, admin=False):
2      if not admin:
3          return self._userApiInstance.account(account_id)
```

Code 3.5: Fetching certain account's public data

This method will return the data as a dictionary where the keys are the public data names and the values are their values.

3.1.6 Getting user's following accounts

We know that if we follow someone that means we most probably know him. Hence, we need to trust the accounts we follow when we start our application. To trust them we need to get the list of the accounts which is done by the following method:

```
1  def getFollowingAccounts(self):
2      my_id = self._userApiInstance.me()['id']
3      return self._userApiInstance.account_following(my_id)
```

Code 3.6: Method to get the list of the accounts we follow

As we can see, we first need to get our account id and then get the list of the accounts we follow. After having the list of the accounts we follow, we need to trust by inserting each in the application's database so we don't have to check them if they try to reach us.

```
1  def trustFollowings(self):
2      following_accounts = self.api.getFollowingAccounts()
3      for account_data in following_accounts:
4          try:
5              account_id = int(account_data['id'])
6              username = str(account_data['username'])
7              domain = str(account_data['url'].split('/')[2])
8              self.__database.insertAccount(account_id, username, domain,
9                                             False)
9              self.__database.insertDomain(domain, False)
10         except Exception:
```

```
11     print("Error")
```

Code 3.7: Inserting the accounts we follow immediately in the application database

3.1.7 Blocking an account

As it was mentioned in 2.4.1 we have the option to block an account and of course the only way to do it outside the application is by using the API. So, in order to block an account we need to call the API method and pass the account's id as a parameter and the account with that id will be blocked.

```
1  def blockAccount(self, account_id):
2      self._userApiInstance.account_block(account_id)
3      print("done")
4      return True
```

Code 3.8: Blocking an account method

3.1.8 Muting an account

Same as blocking an account, when we want to mute a certain account we need to call the API method and pass the account's id as a parameter.

```
1  def muteAccount(self, account_id):
2      self._userApiInstance.account_mute(account_id)
3      return True
```

Code 3.9: Muting an account method

3.1.9 Blocking a domain

When it comes to blocking a domain the parameter changes, since the domains are not classified with ids. If we want to block a domain we need to pass the domain's name as a parameter, but keep in mind that the domain is case-sensitive.

```
1  def blockDomain(self, domain):
2      self._userApiInstance.domain_block(domain)
3      return True
```

Code 3.10: Blockin a domain method

3.2 Predictive Model

Classifying the possibility of a threat is done by a machine learning predictive model. To create our model we used the *Scikit-learn* [3] python library, to preprocess the data we used *Pandas* [4] python library and in order to save the model so it doesn't need to go through the training process again we used *Pickle* [5]

Since our problem was to classify whether an account has a possibility of being a threat, we had to choose between the classification algorithms. Considering multiple factors and trying multiple algorithms we chose to go with Logistic regression for these reasons:

- **Small dataset**, Logistic regression is known to give the best results in small datasets.
- **Low complexity model**, since we had to classify between two classes and we had 19 features to consider, we needed to use a low complexity model.

3.2.1 Data preprocessing

As it was mentioned above we had to deal with data preprocessing in order to make our dataset suitable to train the model. We needed to read the data from a csv file and create a dataset based on the data that was available on that file. The file contained accounts with account's data and it has already classified all of them as threat and not threat. However, we didn't get all the account's data since some of them were not related to our problem, such as username, account id etc. It is worth mentioning that we considered the following account's data only:

- **Followers count**, the number of the accounts that follow the account.
- **Followings count**, the number of the accounts that the account follows.
- **Statuses count**, the number of the statuses the account has posted.
- **Profile picture**, whether the account has a profile picture.
- **Domain**, account's Mastodon domain.
- **Year of creation**, the year when the account was created.

As we can see we had some of the data in string format, like year of creating and domain, so we had to use the dummy encoder from Pandas and that limited our range of domains and year of creation. So, now the model recognized only the following domains:

- **mastodon.elte.hu**
- **mastodon.social**
- **mastodon.technology**
- **mastodon.xyz**
- **scholar.social**
- **fosstodon.org**
- **hofelho.hu**

About the year range, we thought that it was more than enough to consider accounts created from 2015, since the older the account, the less the possibility of being a threat is. So, after applying the dummy encoder our features increased from 6 to 19. Since some of our data could get big values, like the followers count and following count, we needed to apply normalization on data, so that we could get more accurate predictions in the end. After normalization we split the dataset into training set and testing set, where 70 percent of the data was on the training set and the other 30 percent of the data was on the testing set.

3.2.2 Model performance

We mentioned in 3.2 that we needed a low complexity algorithm, and we know that we have three different kinds provided by Scikit-learn which are:

- **Logistic regression**
- **SVC**
- **Naive Bayes**

After trying each of the algorithms, we wanted to go with Logistic regression since the performance, confusion matrix and the ROC curve was showing us that Logistic regression is performing better in comparison to the two others.

```

Training data accuracy: 0.859504132231405
Testing data accuracy: 0.75
Report about performance on testing data
      precision    recall  f1-score   support

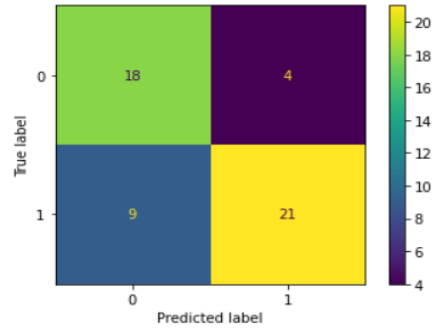
     0       0.67       0.82       0.73        22
     1       0.84       0.70       0.76        30

 accuracy          0.75          0.75          0.75          52
  macro avg       0.75       0.76       0.75          52
 weighted avg     0.77       0.75       0.75          52

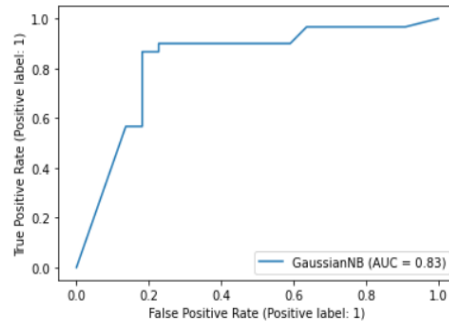
[0]
Mean absolute error on testing data: 0.25

```

(a) Naive Bayes performance scores



(b) Naive Bayes confusion matrix



(c) Naive Bayes ROC curve

Figure 3.2: Naive Bayes overall performance

For the Naive Bayes algorithm we can see that the mean absolute error is quite high, the difference between the training set accuracy and testing test accuracy is more than 10 percent, the f1-score is quite low and the ROC curve isn't the best.

```

Training data accuracy: 0.9173553719008265
Testing data accuracy: 0.9038461538461539
Report about performance on testing data
      precision    recall  f1-score   support

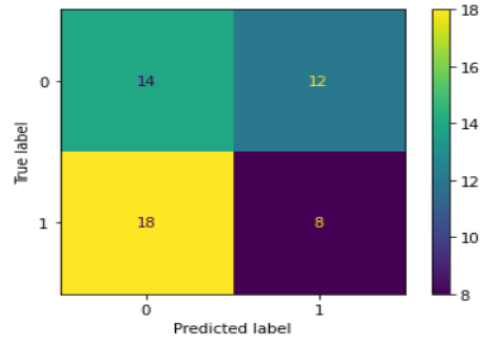
     0       0.81      1.00      0.90        22
     1       1.00      0.83      0.91        30

 accuracy          0.90        52
  macro avg       0.91      0.92      0.90        52
 weighted avg     0.92      0.90      0.90        52

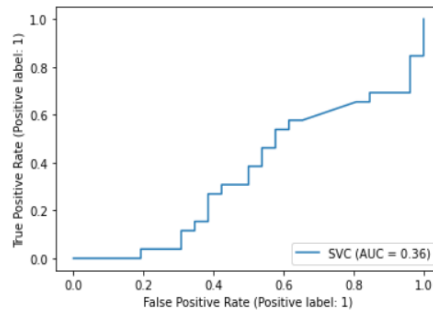
[1]
Mean absolute error on testing data: 0.09615384615384616

```

(a) SVC performance scores



(b) SVC confusion matrix



(c) SVC ROC curve

Figure 3.3: SVC overall performance

Now for the SVC algorithm we can see perfect scores in case of precision, which is impossible to be 1.0 , meaning that we might have the overfitting problem. But, other than precision the other scores show us a promising model until we watch the confusion matrix and the ROC curve. We can see that the confusion matrix is giving us more false predicted values than it should have meaning that most of the predictions the model made were false predictions. Not only confusion matrix is giving us a red light for this model, but even the AUC value which is showing us an awful result, where most of the points are below the middle point.

Training data accuracy: 0.8760330578512396

Testing data accuracy: 0.8653846153846154

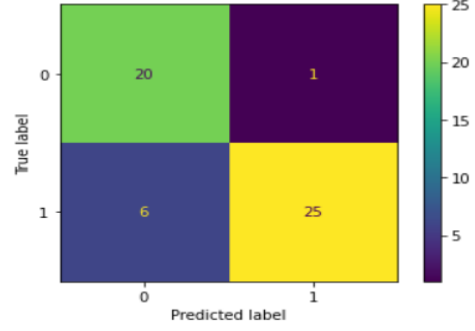
Report about performance on testing data

	precision	recall	f1-score	support
0	0.77	0.95	0.85	21
1	0.96	0.81	0.88	31
accuracy			0.87	52
macro avg	0.87	0.88	0.86	52
weighted avg	0.88	0.87	0.87	52

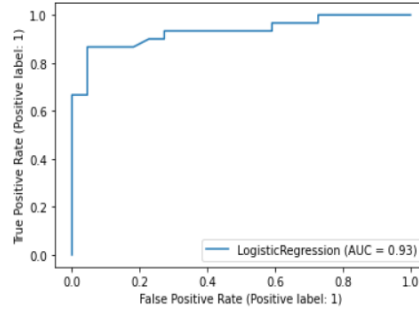
[0]

Mean absolute error on testing data: 0.1346153846153846

(a) Logistic regression performance scores



(b) Logistic regression confusion matrix



(c) Logistic regression ROC curve

Figure 3.4: Logistic regression overall performance

However, if we take a look at the Logistic regression algorithm performance then we can see a good accuracy on both, training set and testing set, with a difference of 1.1 percent. Besides the accuracy we see a good recall and precision score, which makes a good f1 score. The confusion matrix tells us that the most of the values predicted are true, and the ROC curve shows us a very good AUC score of 0.93.

3.3 Database

We used database in our application in order to save the already checked users and domains. For the database we chose to use the *SQLite* [6] with it's API python library called *sqlite3* [7].

We created two tables, one for the handled accounts and one for the handled domains. They are both initialized at the same time and they are not connected with each other, meaning that they behave independently.

As a first step of initializing the database is to connect to it, create the tables and after we are done we need to commit the data.

3.3.1 Handled accounts table

The handled accounts table consists of these data:

- **Account id**, as a primary key.
- **Account username**
- **Account domain**
- **Threat boolean**

So, whenever an account is getting checked for the threat possibility, it will immediately go in the database after the threat possibility result is given, and that account doesn't need to be checked twice since its in the database of the handled accounts already.

3.3.2 Handled domains table

The handled domain table consists of these data:

- **Domain name**, as a primary key.
- **Threat boolean**

Same as in 3.3.1, we will insert the domain into the handled domain table immediately after receiving the threat possibility result.

3.4 GUI

3.5 Testing

Chapter 4

Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus portitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Bibliography

- [1] Mastodon org. *Mastodon open source social media*. URL: <https://www.joinmastodon.org>.
- [2] Lorenz Diener. *Mastodon API for Python*. URL: <https://github.com/halcy/Mastodon.py>.
- [3] F. Pedregosa et al. *Scikit-learn: Machine Learning Library in Python*. URL: <https://scikit-learn.org/stable/>.
- [4] Wes McKinney. *Data structures for statistical computing in python*. URL: <https://pandas.pydata.org/stable>.
- [5] Python Software Foundation. *Python object serialization*. URL: <https://docs.python.org/3/library/pickle.html>.
- [6] Dwayne Richard Hipp. *SQL database engine*. URL: <https://www.sqlite.org/index.html>.
- [7] Gerhard Häring. *DB-API 2.0 interface for SQLite databases*. URL: <https://docs.python.org/3/library/sqlite3.html>.

List of Figures

2.1	Application main page	6
2.2	Button disabled and enabled	7
2.3	Application running	8
2.4	Incorrect log in data	9
2.5	Possible threat notification	10
2.6	Action window	11
2.7	Actions available for account and domain	11
2.8	Pop up window to confirm the actions	12
3.1	Mastodon API scheme	16
3.2	Naive Bayes overall performance	24
3.3	SVC overall performance	25
3.4	Logistic regression overall performance	26

List of Tables

2.1	Description of every action that can be taken against an account . . .	13
2.2	Description of every action that can be taken against a domain . . .	14

List of Codes

2.1	Cloning Repository	5
2.2	Installing requirements	5
2.3	Running the application	5
3.1	Logging in to Mastodon account through API	16
3.2	Creating API instance	17
3.3	Getting the notifications by API	18
3.4	Filtering the notifications	18
3.5	Fetching certain account's public data	20
3.6	Method to get the list of the accounts we follow	20
3.7	Inserting the accounts we follow immediately in the application database	20
3.8	Blocking an account method	21
3.9	Muting an account method	21
3.10	Blockin a domain method	21