**Program Requirements**

Pre Requirements:

* Pyhon3.9
* Pip (“python3 -m pip install --upgrade pip”)

Python pip Requirements: (All pip requirements are in requirements.txt. Just run the command "pip install -r requirements.txt".)

* flask
* pandas
* snscrape
* scikit-learn

**To run the program**

1. First of all, the terminal screen should be opened in the file location where the codes are located.
2. Run python3 app.py
3. And visit the address http://127.0.0.1:8000/

# Selecting Dataset

There are many datasets for bot account detection on the Internet. However, many of these datasets are not suitable for use with machine learning. The main reason for this is the absence of data labels. Many datasets give an account information but do not present the label that the account is a bot or a human.

In addition, the data set is important in choosing a machine learning algorithm. Our problem is figuring out how similar a user is to a bot, by percentage. So we need to make a classification as a percentage. Classifier algorithms are trained with categorical and continuous features in the data set. To detect how bot-like a new user is, we need to have all the attributes used in the training. For this reason, the data set should not contain data that we cannot retrieve through APIs.

Academic articles were reviewed. An article named "Twitter Bot Account Detection Using Supervised Machine Learning" was found published by Pramitha et al in 2021. The dataset used in this article is available on Kaggle. <https://www.kaggle.com/datasets/joopedrolimo/bots-in-twitter>

This data set contains 37000 user data. Attributes of the dataset:

1. **id**, is an integer representation of the unique identifier for the user. id of type int.

2. **u\_name**, or username is a name that identifies an account where its function is very important to distinguish one user from another.

3. **screen\_name**, is the display name or alias on an account, screen\_names are unique but can change, screen\_name is an object type.

4. **verified**, to find out whether an account is verified or not, verified is a boolean type.

5. **geo\_enabled**, to attach user geographic data, geo\_enabled is a boolean type.

6. **default\_profile**, if “True”, indicates that the user has not changed the theme or background of their user profile. default\_profile of type boolean.

7. **default\_profile\_image,** indicates that the user has not uploaded their own profile picture and the default image is used instead. default\_profile\_image of type boolean.

8. **favorites\_count,** is the number of tweets that the user liked during the active period of the account. favorites\_count of type int.

9. **followers\_count,** the number of followers currently on an account. followers\_count of type int.

10. **friends\_count**, is the number of users following an account (followers or “following”). friends\_count of type int.

11. **statuses\_count,** is the number of tweets (including

retweets) issued by the user. statuses\_count of type int.

12. **network**, a feature that is processed from the friends\_count and followers\_count features. network type float.

13. **average\_tweets\_per\_day**, is the average tweet issued per day, average\_tweets\_per\_day type int.

14. **created\_at**, is the time of creation of an account, created\_at type object.

15. **account\_type**, is the type of an account, whether bot or non-bot (human), account\_type type object.

# Training Machine Learning Algorithm

After the data set is selected, the training of the machine learning algorithm can be performed. Machine learning algorithms work with continuous and discrete numerical variables. For this reason, first of all, the step called preprocessing must be performed. The file named **train\_model.py** contains our machine learning steps and preprocessing codes.

import pandas as pd  
df = pd.read\_csv("datasets/bots\_in\_twitter.csv", lineterminator='\n')

Reading data set in csv format with Pandas library. The read data is stored in a data frame called df.

df = df[["default\_profile",  
 "default\_profile\_image",  
 "favourites\_count",  
 "followers\_count",  
 "friends\_count",  
 "geo\_enabled",  
 "verified",  
 "average\_tweets\_per\_day",  
 "account\_age\_days",  
 "account\_type"]]

Machine learning algorithms work with categorical or continuous numerical variables. We can use 10 features in our dataset for machine learning. We extract the relevant columns from the data set we read above.

from sklearn import preprocessing  
label\_encoder = preprocessing.LabelEncoder()  
  
df['default\_profile'] = label\_encoder.fit\_transform(df['default\_profile'])  
df['default\_profile\_image'] = label\_encoder.fit\_transform(df['default\_profile\_image'])  
df['geo\_enabled'] = label\_encoder.fit\_transform(df['geo\_enabled'])  
df['verified'] = label\_encoder.fit\_transform(df['verified'])  
df['account\_type'] = label\_encoder.fit\_transform(df['account\_type'])

Then we need to convert categorical expressions such as true or false to numeric expressions. We complete the conversion process with the labelEncoder class of the sklearn machine learning library.

from sklearn.model\_selection import train\_test\_split  
labels = df["account\_type"]  
features = df.drop(columns=['account\_type'])  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.20, random\_state=42)

We made our data available to machine learning algorithms. Now we need to divide our data into certain parts for the learning process to take place correctly. First of all, without breaking the data order, we take the account\_type field, which holds whether the accounts are bot or human, into a separate variable. We will use a certain part of the data we have in the testing phase after the learning process is completed. For this stage, 25% to 20% of the data is considered sufficient. For us, we have a tool that splits the data into both training and test data. This tool splits our data into 4 parts.

1. X\_train : Data to be used in training

2. y\_train: Tag information to be used in training (bot/human)

3. X\_test: Data to be used in the testing phase

4. y\_test: Tag information (bot/human) to be used in the test

from sklearn.ensemble import RandomForestClassifier  
clf = RandomForestClassifier(n\_estimators=10)  
clf.fit(X\_train, y\_train)

We acquire machine learning algorithms through the sklearn library. There are many libraries for Python, but sklearn is the most successful library in terms of both performance and adaptability. Many machine learning algorithms have been tried in this step. Artificial neural networks, decision trees, support vector machines and random forest algorithm. The best result was obtained with random forest with an F1 score of 89%.

predicted = clf.predict(X\_test)  
from sklearn.metrics import f1\_score  
print(f1\_score(y\_test, predicted))

The most basic performance metric of a classifier is its accuracy rate. but the accuracy rate can mislead us. There are a article about [F1 score](https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score-dbd973bf1a3#:~:text=F1%2Dscore%20ranges%20between%200,1%2C%20the%20better%20the%20model.). Let's guess the people with glasses in a school classroom. And 8 people in the class do not wear glasses. If we say that everyone does not wear glasses, our accuracy rate would be 80%. Machine learning algorithms can make such generalization errors. For this reason, we need to know the performances of our model in saying right to right and wrong to wrong. F1 Score gives us this value.

import pickle  
file = open('pickle\_variables/machine\_learning\_model.pickle', 'wb')  
pickle.dump(clf, file)  
file.close()

It is time-consuming to train a model when a request for classification comes from each user. For this reason, we save our classifier to disk with the pickle library.

# Receive query from user

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Python has a library called flask that provides a light weight API and web interface. An interface was created using Bootstrap 5, HTML and flask. Therefore, the query is accepted with the username and tweet url. The codes in this section are in the app.py file.

from flask import Flask, render\_template, request  
import pickle  
import pandas as pd  
from scraper import \*  
from fuctions import \*  
app = Flask(\_\_name\_\_)

The required libraries are included and an instance of the flask class is taken. Here are the scraper and function files. The scraper file contains the helper functions of the snscraper scraper library. The functions file contains a few simple tools.

# Loads the machine learning classifier into memory, which is saved in the folder named "pickle\_variables".  
file = open("pickle\_variables/machine\_learning\_model.pickle", "rb")  
clf = pickle.load(file)  
  
# Returns the average statistics of bot users in the dataset.  
bot\_avg = get\_bot\_avg\_statistic()

We load our classifier, which was saved to disk with pickle, back into memory. then we get the average of the bot users' data from our dataset to show it to the users. The function used here is obtained from the functions.py file.

@app.route('/', methods=['GET', 'POST'])  
def home\_page(): # Home Page

The function that will run when there is a get and post request to the root directory of our application. There is no other directory in the application. **The commands after this section are located under this function.**

if request.method == "POST":  
 if request.form.get("action") == "check":  
 user = None  
 # Works when querying with tweet url  
 if request.form.get("url") != "":  
 user = get\_user\_from\_tweet(request.form.get("url"))  
 data["url"] = request.form.get("url")  
  
 # Works when querying with username  
 elif request.form.get("username") != "":  
 user\_name = request.form.get("username")  
 data["user\_name"] = user\_name  
 user = get\_user\_data(user\_name)  
  
 # Username and tweet may have been entered incorrectly and no users were found.  
 # The is\_valid\_user flag is sent to the frontend to warn the user.  
 data["user"] = user  
 if user is None:  
 data["is\_valid\_user"] = False  
 else:  
 data["is\_valid\_user"] = True

The username and url sent from the form in the interface are checked in this part. If the correct information is not entered, the user information is assigned as "None". Is valid user information is sent to the interface from this section. If the user cannot be validated, a warning message is displayed on the interface.

features = [  
 default\_profile(user),  
 default\_profile\_image(user),  
 user.favouritesCount,  
 user.followersCount,  
 user.friendsCount,  
 geo\_enabled(user),  
 verified(user),  
 average\_tweets\_per\_day(user),  
 account\_age\_days(user)  
]  
  
# feature names for data frame generation  
feature\_names = ["default\_profile",  
 "default\_profile\_image",  
 "favourites\_count",  
 "followers\_count",  
 "friends\_count",  
 "geo\_enabled",  
 "verified",  
 "average\_tweets\_per\_day",  
 "account\_age\_days"]  
feature\_df = pd.DataFrame([features], columns=feature\_names)

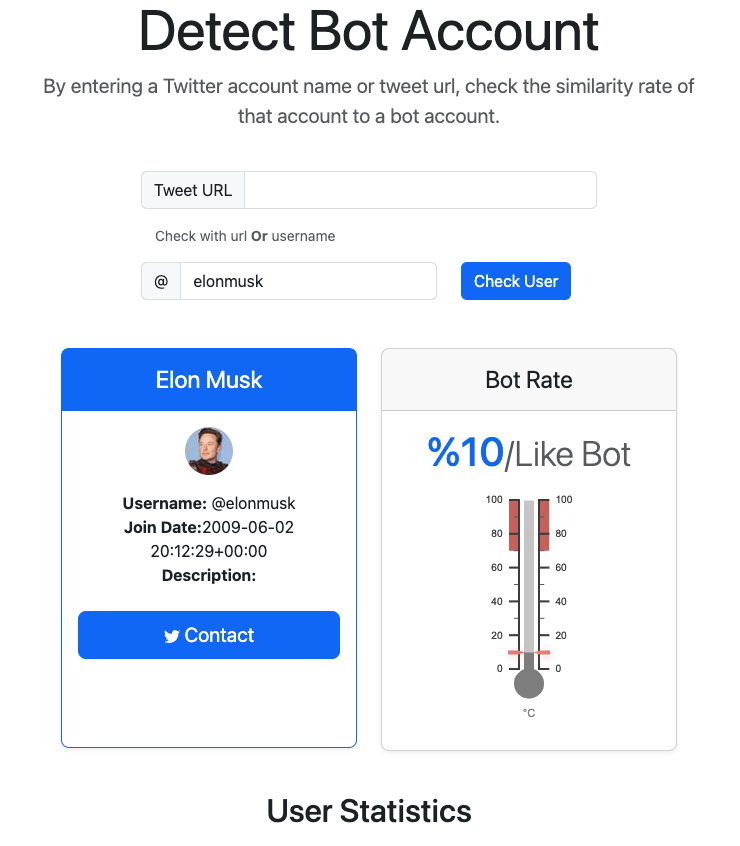
In this part, if the user is valid, the data is obtained through the user class. The library that allows us to obtain the data is snscrape. In order to get the results we want from the snscrape library, the codes in the file named scrape.py are written. **Here, via twitter, the information needed by our classifier is obtained.**

data["features"] = feature\_df.to\_dict('records')[0]

The resulting data is converted to a dict object to be sent to the interface.

roba = clf.predict\_proba(feature\_df)

The information obtained with Snscrape is given to the classifier. And the bot rate is obtained.



The data sent to the frontend is processed by the jinja template. As a result of classification, user information is presented first. Then the Bot rate is given as a percentage. This value is between 0 and 100. It increases and decreases with slices of 10. Coloring changes depending on the possibility of the user being a bot. Between 0 and 30 it is Blue, between 40 and 60 Yellow, between 70 and 100 it is red.

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

Finally, we provide data for the user to understand why they are or not bots. We can compare the user's data used in the classifier with the average data of bot accounts in a single table.

# Scrape Code (scraper.py)

Twitter data was obtained using a library called snscrape. <https://github.com/JustAnotherArchivist/snscrape> This library is free and has no limits. Simple and fast, independent of Twitter API.

def get\_user\_data(username):  
 try:  
 profile = sntwitter.TwitterUserScraper(username)  
 results = profile.get\_items()  
 except:  
 return None  
  
 null\_user = True  
 item = None  
 try:  
 for item\_ in results:  
 item = item\_  
 null\_user = False  
 break  
 except:  
 null\_user = False  
  
 if null\_user:  
 return None  
 return item.user

The get\_user\_data function allows us to get the username and tweeter information. This function returns us a user object.

This object contains the properties given in the code below.

return User(  
 username = user['legacy']['screen\_name'],  
 id = int(user['rest\_id']),  
 displayname = user['legacy']['name'],  
 rawDescription = rawDescription,  
 renderedDescription = renderedDescription,  
 descriptionLinks = [TextLink(  
 text = x.get('display\_url'),  
 url = x['expanded\_url'],  
 tcourl = x['url'],  
 indices = tuple(x['indices']),  
 ) for x in user['legacy']['entities']['description']['urls']],  
 verified = user['legacy']['verified'],  
 created = email.utils.parsedate\_to\_datetime(user['legacy']['created\_at']),  
 followersCount = user['legacy']['followers\_count'],  
 friendsCount = user['legacy']['friends\_count'],  
 statusesCount = user['legacy']['statuses\_count'],  
 favouritesCount = user['legacy']['favourites\_count'],  
 listedCount = user['legacy']['listed\_count'],  
 mediaCount = user['legacy']['media\_count'],  
 location = user['legacy']['location'],  
 protected = user['legacy']['protected'],  
 link = link,  
 profileImageUrl = user['legacy']['profile\_image\_url\_https'],  
 profileBannerUrl = user['legacy'].get('profile\_banner\_url'),  
 label = label,  
 )

This class collects all necessary information about a user. If the user is not found, the variable user is set to None.

def get\_user\_from\_tweet(url=""):  
 url = url.split("/")[-1]  
 tweet = sntwitter.TwitterTweetScraper(url)  
 username = ""  
 try:  
 for i in tweet.get\_items():  
 username = i.username  
 except:  
 pass  
 return get\_user\_data(username)

In the program, query can also be made with the tweet url. The get\_user\_from\_tweet function takes a url. Split this url by the "/" character to get the id of the tweet. With this id, information about the tweet is obtained. The user name is reached through the tweet object and the user object is obtained with the get\_user\_from\_tweet function.

All functions after this section return the data obtained from the user object by editing.

def default\_profile(user):  
 # Default Banner Photo Links  
 default\_banner\_photos = ["https://abs.twimg.com/images/themes/theme1/bg.png",  
 "https://abs.twimg.com/images/themes/theme14/bg.gif",  
 None]  
  
 if user.profileBannerUrl in default\_banner\_photos:  
 return 0  
 else:  
 return 1

It checks if there is any customization in the user profile. 0 means there is no editing in the profile, 1 means it has been customized.

def default\_profile\_image(user):  
 # Default Profile Photo Links  
 default\_profile\_photos = ["https://abs.twimg.com/sticky/default\_profile\_images/default\_profile\_normal.png",  
 None]  
  
 if user.profileImageUrl in default\_profile\_photos:  
 return 0  
 else:  
 return 1

Checks if the user has changed their profile photo. 0 means there is no editing in the profile, 1 means it has been customized.

def geo\_enabled(user):  
 if user.location == "":  
 return 0  
 return 1

Checks if the user has shared location information.

def account\_age\_days(user):  
 from datetime import date  
  
 sign\_date = user.created.date()  
 now = date.today()  
 days = (now - sign\_date).days  
 return days

Calculates how many days have passed since the user's account creation date.

def average\_tweets\_per\_day(user):  
 tweets\_count = user.statusesCount  
 account\_age = account\_age\_days(user)  
 return tweets\_count / account\_age

Calculates the daily average of tweets posted by the user.

def verified(user):  
 if user.verified:  
 return 1  
 return 0

Checks if the user has verified.