**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY**

**Kholvad Campus, Kamrej, Surat - 394190**



**LABORATORY MANUAL**

**CS 601: INTRODUCTION TO MACHINE LEARNING**

**B.TECH - SEMESTER VI [JAN-23]**

**Course Faculty**

**Dr. Pradeep Kumar Roy**

NAME: **CHARCHIT BHATNAGAR**

ROLL NO:

BRANCH: **COMPUTER SCIENCE AND ENGINEERING**

**UI20CS15**

**CONTENT PAGE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp. No** | **Name of the Experiments** | **Page no** | **Date of Experiment** | **Date of Submission** |
| **1** | Write a Python script (a) To collect tweets that may incorporate owner, date of post, number of re-tweet, no of followers, no of follower, and other associated information from Twitter and store it into a .csv file. (b) To scrap users' reviews from any E-commerce portal (Ex- Amazon, Flipkart) and store it into a csv file that may incorporate date of post, number of likes/dislikes, and other associated fields. |  |  |  |
| **2** | Linear regression: a) Read the .csv file and use that data for programming purposes. (b) Plot the input-output relation for a single attribute also for the multiple attributes. (c)Find the best fitted line that gives the minimum least square error. |  |  |  |
| **3** | Repeat LAB 2 with sample data X={2,4,6,8} and Y={3, 7,5,10}. Plot Error Vs Slop Graph for different value of learning rate (alpha) and find the best suited value of slope (m). |  |  |  |
| **4** | Prepare a report to discuss about any five RL environments available in the OpenAI Gym. |  |  |  |
| **5** | Perform Logistic Regression on SMS Spam Dataset by following the instructions. |  |  |  |
| **6** | Implement Logistic Regression, and Naive Bayes Classifier considering the following data pre-processing steps and check the model's performance. |  |  |  |
| **7** | Embed developed Machine Learning Models to Web page. |  |  |  |
| **8** | Deploy a machine learning model on a web interface where users can test their data. (i) single data item will be tested (ii) If multiple data exist, allow the user to upload the test data in a file (CSV, Zip). | **3-7** | 28-03-2023 | 26-04-2023 |
| **9** | Deploy a machine learning model on the web, which can be used to test real-time data. |  |  |  |

* **LAB 1:**

# Aim: Write a Python script:

# To collect tweets that may incorporate owner, date of post, number of re-tweets, no of followers, no of follower, and other associated information from Twitter and store it into a .csv file.

1. To scrap users' reviews from any E-commerce portal (Ex- Amazon, Flipkart) and store it into a csv file that may incorporate date of post, number of likes/dislikes, and other associated fields.

---------------------------------------------------------------------------------------------------------

**Description:**

1. Twitter is a goldmine of data. Unlike other social platforms, almost every user’s tweets are completely public and pullable.

We can use Tweepy Library by importing, by using tweepy we can authenticate uor api keys to get data. It enables us to collects different types of details such as user\_id, user\_no\_of\_followers, user\_tweets/retweets, etc.

We can store the concern information in a csv file for future reference.

1. We can collect all the required information from a website with just a few lines of Python codes. Yes, it’s possible by using Python libraries such as Beautiful Soup and Requests. This is known as Web Scraping.

Web scraping is a technique to automate the extraction process of a large amount of data from the website. The data present on the websites will be in unstructured format but with the help of Web scraping, we can scrape, access, and store the data in a much more structured and cleaner format for our further analysis.

---------------------------------------------------------------------------------------------------------

Source Code:

# To collect tweets that may incorporate owner, date of post, number of re-tweets, no of followers, no of follower, and other associated information from Twitter and store it into a .csv file.

# Importing the required libraries:

import tweepy             # Python wrapper around Twitter API

import json

import csv

from datetime import date

from datetime import datetime

import time

import pandas as pd

* **Authenticating tweeter keys:**

api\_key= "CXS7iSbOOWfUjBVoHzztS9ZUd"

api\_key\_secret= "jILGeSAfjXWXjB4CqnrMFd3mXaDThqgUGpYYLWt0ESrErLdJed"

access\_token= "733711577334075393-JRUkTcHVmKiAv6Yz0pQt7gEdBw28Kkj"

access\_token\_secret= "hHnpcNZ4RHiVwrkGrks3QkotO8MLi3jEtCAE3oASOpt8a"

auth = tweepy.OAuthHandler(api\_key, api\_key\_secret)

auth.set\_access\_token(access\_token, access\_token\_secret)

api = tweepy.API(auth)

* **Getting User ID using user screen name:**

put\_your\_screen\_name = "cristiano"

user1 = api.get\_user(screen\_name=put\_your\_screen\_name)

user1.id

*Output: 155659213 //User ID of the user with screen name: “cristiano”*

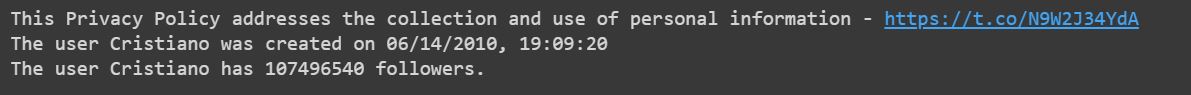
* **More details:**

print(user1.description)

print(f'The user {user1.screen\_name} was created on {(user1.created\_at).strftime("%m/%d/%Y, %H:%M:%S")}')

print(f"The user {user1.screen\_name} has {user1.followers\_count} followers.")

Output:



*Figure 1: Details of the user with user\_name = “Cristiano”:*

* **Code for getting tweets done by a particular user:**

#API object allows us to pull a maximum of 200 tweets. Even if we specify count as 300.

tweets = api.user\_timeline(screen\_name='cristiano', count=200,

                           tweet\_mode='extended')

# create DataFrame

columns = ['TweetId', 'User', 'Tweet','Time']

data = []

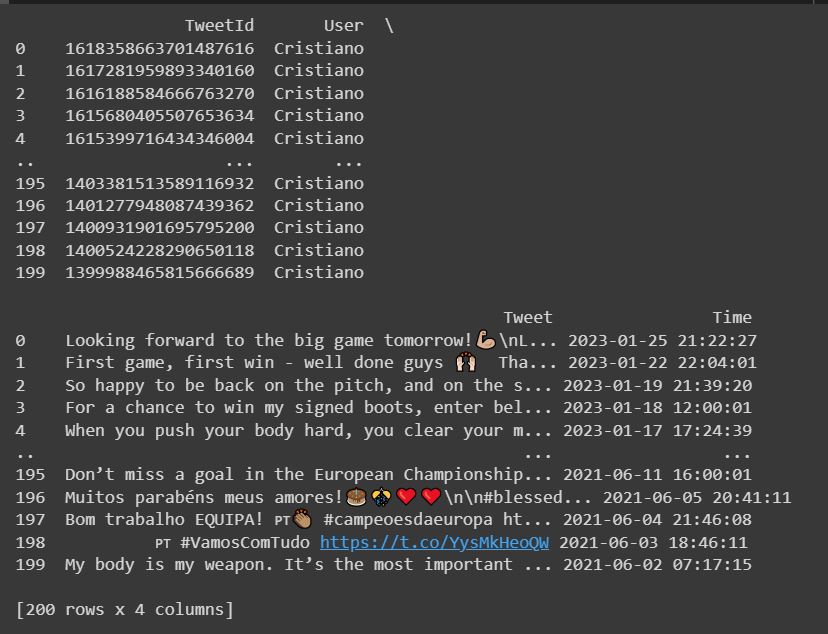
for tweet in tweets:

    data.append([tweet.id, tweet.user.screen\_name, tweet.full\_text,tweet.created\_at])

df = pd.DataFrame(data, columns=columns)

print(df)

Output:



*Figure 2: Details of all the recent 200 tweets done by “Cristiano”, along with date and time:*

* **Code for saving the data in a csv file with name as “tweets.csv”:**

df.to\_csv('tweets.csv')

* **Code for getting replies to a particular tweet by using tweet\_id:**

#Getting replies to a tweet

tweet\_id = '1618358663701487616'

user\_name = 'Cristiano'

columns = ['User Name', 'User Screen Name', 'Reply Text','Time']

data = []

for tweet in tweepy.Cursor(api.search, q='to:'+user\_name, result\_type='recent').items(1000):

    if hasattr(tweet, 'in\_reply\_to\_status\_id\_str'):

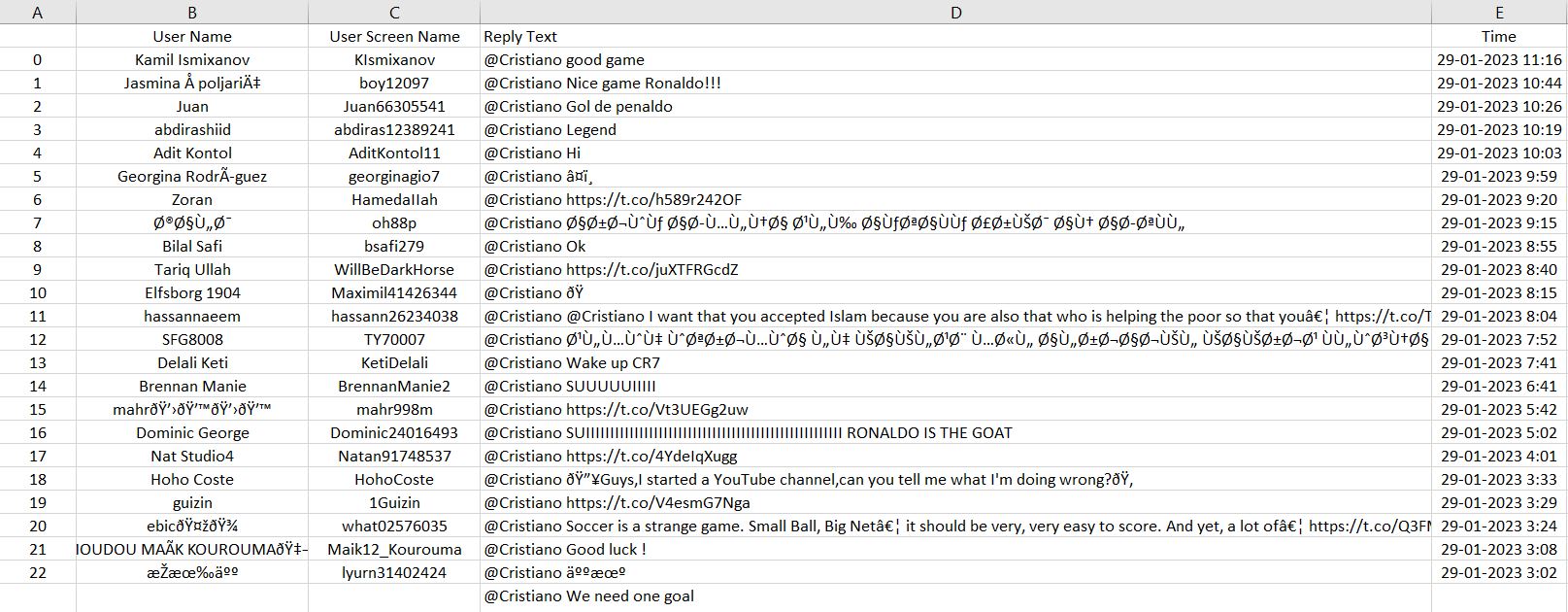
        if (tweet.in\_reply\_to\_status\_id\_str == tweet\_id):

            data.append([tweet.user.name, tweet.user.screen\_name, tweet.text,tweet.created\_at])

            print(f"{tweet.user.name} ({tweet.user.screen\_name}) replied with- {tweet.text}")

df = pd.DataFrame(data, columns=columns)

df.to\_csv('public\_replies.csv')

Output:

*Figure 3: Replies of the tweet*:

----------------------------------------------------------------------------------------------------------------------------------

1. To scrap users' reviews from any E-commerce portal (Ex- Amazon, Flipkart) and store it into a csv file that may incorporate date of post, number of likes/dislikes, and other associated fields.

Source Code:

import pandas as pd

import requests

from bs4 import BeautifulSoup

HEADERS = ({'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) \AppleWebKit/537.36 (KHTML, like Gecko) \Chrome/90.0.4430.212 Safari/537.36','Accept-Language': 'en-US, en;q=0.5'})

# Scrape the data

def getdata(url):

    r = requests.get(url, headers=HEADERS)

    return r.text

def html\_code(url):

    # pass the url

    # into getdata function

    htmldata = getdata(url)

    soup = BeautifulSoup(htmldata, 'html.parser')

    # display html code

    return (soup)

url = "https://www.amazon.in/Samsung-Storage-sAmoled-Purchased-Separately/product-reviews/B09XJ5LD6L/ref=cm\_cr\_dp\_d\_show\_all\_btm?ie=UTF8&reviewerType=all\_reviews"

soup = html\_code(url)

def all\_rev(soup,name,rev):

    # find the Html tag

    # with find()

    # and convert into string

    data\_str = ""

    cus\_list = []

    div\_all = soup.find\_all("div", class\_="a-section review aok-relative")

    data\_str = ""

    for item in div\_all:

      div\_name = item.find("span", class\_="a-profile-name")

      data\_str = data\_str + div\_name.get\_text()

      name.append(data\_str)

      data\_str = ""

      div\_rev = item.find("div",class\_="a-row a-spacing-small review-data")

      data\_str = data\_str + div\_rev.get\_text()

      rev.append(data\_str)

      data\_str = ""

name = []

rev = []

for i in range (100):

    tempurl = url + "&pageNumber="+ str(i+1)

    soup = html\_code(tempurl)

    cus\_name = []

    rev\_result = []

    all\_rev(soup,cus\_name,rev\_result)

    name.extend(cus\_name)

    rev.extend(rev\_result)

    cus\_name = []

    rev\_result = []

print(len(name))

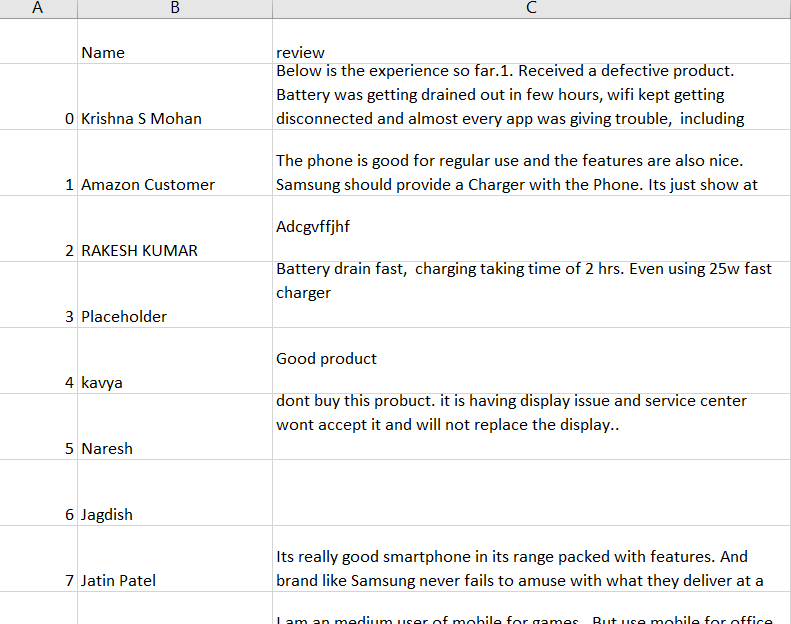
print(len(rev))

data = {'Name': name, 'review': rev}

df = pd.DataFrame(data)

df.to\_csv('reviews.csv')

Output:



----------------------------------------------------------X----------------------------------------------------------------------

* **LAB 2:**

# Aim: Linear Regression (Use python as a programming platform) :

# Read the .csv file and use that data for programming purposes.

# Plot the input-output relation for a single attribute also for the multiple attributes.

1. Find the best fitted line that gives the minimum least square error.

---------------------------------------------------------------------------------------------------------

**Description:**

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

### **Cost function-**

* The different values for weights or coefficient of lines (a0, a1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
* Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
* We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)**cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

Mean Square Error can be calculated as:

Linear Regression in Machine Learning

---------------------------------------------------------------------------------------------------------

**Source Code:**

1. Read the .csv file and use that data for programming purposes.

Source Code:

import pandas as pd

import numpy as np

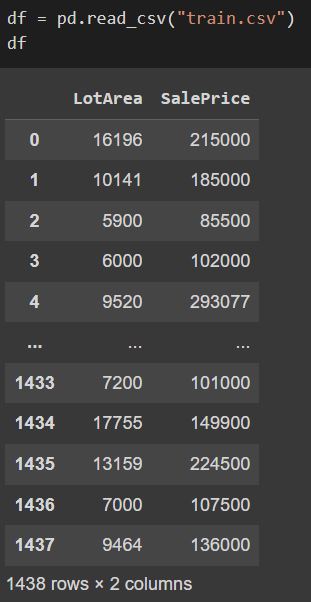
import matplotlib.pyplot as plt

from sklearn import linear\_model

df = pd.read\_csv("train.csv") //reading train file

te=pd.read\_csv("test.csv") //reading test file

Output:



*Fig 1: Printing data of the file:*

1. Plot the input-output relation for a single attribute also for the multiple attributes.

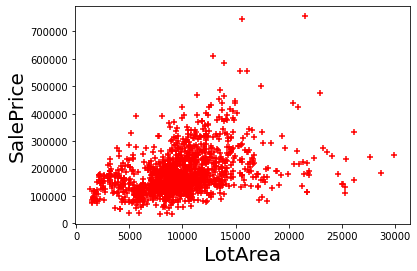
%matplotlib inline

plt.xlabel('LotArea',fontsize=20)

plt.ylabel('SalePrice',fontsize=20)

plt.scatter(df.LotArea,df.SalePrice,color='red',marker = '+')

Output:



*Fig 2: the input-output relation for a single attribute.*

**Training the model:**

reg = linear\_model.LinearRegression()

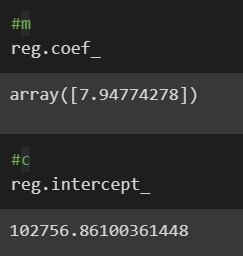
 ## Training the Model

reg.fit(df[['LotArea']],df.SalePrice)

## Predicting the sale price for lot area = 7917

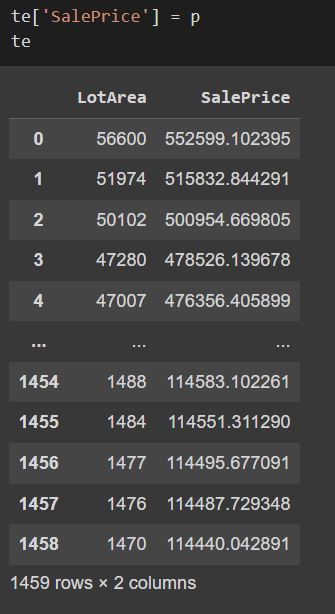
reg.predict([[7917]])

array([165679.14059889])



1. Find the best fitted line that gives the minimum least square error.

p = reg.predict(te) //predicting the values



*Fig 3: printing predicted data*

te.to\_csv("predicted.csv") //Saving the file..

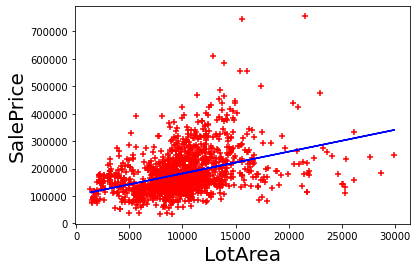
Plotting the best fit line:

%matplotlib inline

plt.xlabel('LotArea',fontsize=20)

plt.ylabel('SalePrice',fontsize=20)

plt.scatter(df.LotArea,df.SalePrice,color='red',marker = '+')

plt.plot(df.LotArea,reg.predict(df[['LotArea']]),color='blue')

*Fig 4: Potting the best fitted line that gives the minimum least square error.*

* **LAB 3:**

# Aim: Repeat LAB 2 with sample data X={2,4,6,8} and Y={3, 7,5,10}. Plot Error Vs Slop Graph for different value of learning rate (alpha) and find the best suited value of slope (m)

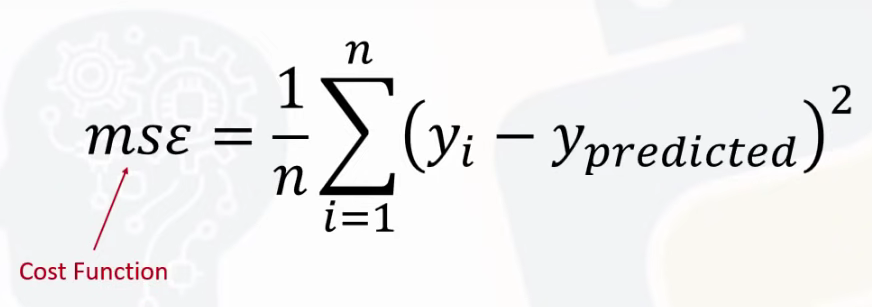
---------------------------------------------------------------------------------------------------------

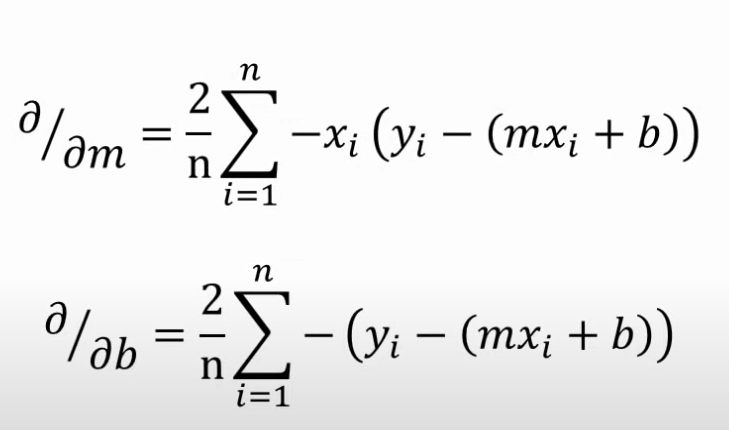
**Description:**

1. Gradient Descent: Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates.
2. Learning Rate: This is the hyperparameter that determines the steps the gradient descent algorithm takes. Gradient Descent is too sensitive to the learning rate.

* If it is too big, the algorithm may bypass the local minimum and overshoot.
* If it too small, it might increase the total computation time to a very large extent.

1. Cost Function: The cost function is the technique of evaluating “the performance of our algorithm/model”. It takes both predicted outputs by the model and actual outputs and calculates how much wrong the model was in its prediction. It outputs a higher number if our predictions differ a lot from the actual values.

 Fig1. Formula for MSE = Mean Square Error:

Fig 2. Slope(m) and intercept(b)

---------------------------------------------------------------------------------------------------------

Source Code:

import numpy as np

def gradient\_descent(x, y):

    error = []

    slope = []

    m\_curr = c\_curr = 0

    iteration = 100000

    n = len(x)

    learning = 0.01

    for i in range(iteration):

        slope.append([m\_curr])

        y\_pre = m\_curr\*x+c\_curr

        cost = (1/n) \* sum ([val\*\*2 for val in (y-y\_pre)])        ##cost is Mean Square Error (MSE) :

        error.append([cost])

        md = -(2/n)\*sum(x\*(y-y\_pre))

        cd = -(2/n)\*sum(y-y\_pre)

        m\_curr = m\_curr-learning\*md

        c\_curr = c\_curr-learning\*cd

        print("m = {} , c = {} , cost = {} , iteration = {}".format(m\_curr, c\_curr, cost, i))

    plt.plot(slope,error,color='blue')

x = np.array ([2,4,6,8])

y = np.array ([3,7,5,10])

gradient\_descent(x,y)

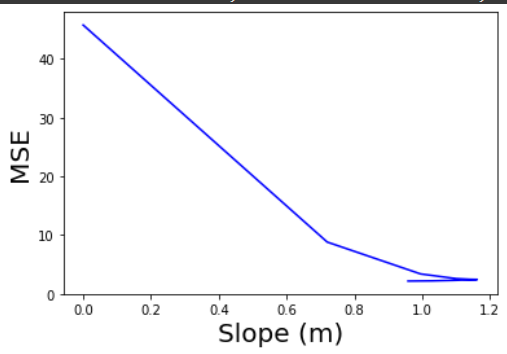
Output:

Fig 3: Graph of Slope vs MSE

* **LAB 4:**

# Aim: Prepare a report to discuss about any five RL environments available in the OpenAI Gym.

# ------------------------------------------------------------------------------------------------

# Description:

# What is Reinforcement Learning? :

# *Reinforcement learning* is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation.

# *Reinforcement learning* differs from *supervised learning* in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

# What is OpenAI Gym? :

OpenAI gym is an environment for developing and testing learning agents. It is focused and best suited for reinforcement learning agent but does not restrict one to try other methods such as hard coded game solver / other deep learning approaches.

# RL environments available in the OpenAI Gym:

# Taxi

# Reacher

# Pusher

# Inverted Pendulum

1. Inverted Double Pendulum
2. Half Cheetah
3. Hopper
4. Swimmer
5. Walker2d
6. Ant
7. Humanoid
8. Humanoid Standup

# Cart Pole

# Mountain Car

# Pendulum

# Acrobat

# Lunar Lander

# Lunar Lander Continuous

# Bipedal Walker

# Bipedal Walker Hardcore

# Car Racing

# Blackjack

# Frozen Lake

# Cliff Walking

# ------------------------------------------------------------------------------------------------

# Report:

1. **The Taxi Problem:**

* Description:

There are four designated locations in the grid world indicated by R(ed), G(reen), Y(ellow), and B(lue). When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger’s location, picks up the passenger, drives to the passenger’s destination (another one of the four specified locations), and then drops off the passenger. Once the passenger is dropped off, the episode ends.



* Actions:

There are 6 discrete deterministic actions:

0: move south

1: move north

2: move east

3: move west

4: pickup passenger

5: drop off passenger

* Observations:

There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations.

Note that there are 400 states that can actually be reached during an episode. The missing states correspond to situations in which the passenger is at the same location as their destination, as this typically signals the end of an episode. Four additional states can be observed right after a successful episode, when both the passenger and the taxi are at the destination. This gives a total of 404 reachable discrete states.

Each state space is represented by the tuple: (taxi\_row, taxi\_col, passenger\_location, destination)

An observation is an integer that encodes the corresponding state. The state tuple can then be decoded with the “decode” method.

Passenger locations:

0: R(ed)

1: G(reen)

2: Y(ellow)

3: B(lue)

4: in taxi

Destinations:

0: R(ed)

1: G(reen)

2: Y(ellow)

3: B(lue)

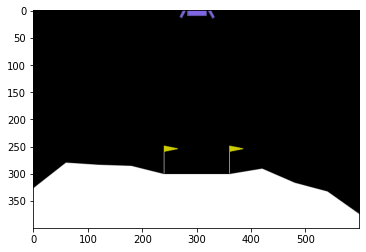
* Rewards:
* -1 per step unless other reward is triggered.
* +20 delivering passenger.
* -10 executing “pickup” and “drop-off” actions illegally.

# ------------------------------------------------------------------------------------------------

1. **The Lunar Lander Problem:**

* Description:

This environment is a classic rocket trajectory optimization problem. According to Pontryagin’s maximum principle, it is optimal to fire the engine at full throttle or turn it off. This is the reason why this environment has discrete actions: engine on or off.

There are two environment versions: discrete or continuous. The landing pad is always at coordinates (0,0). The coordinates are the first two numbers in the state vector. Landing outside of the landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt.

* Actions space:

There are four discrete actions available: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

* Observations space:

The state is an 8-dimensional vector: the coordinates of the lander in x & y, its linear velocities in x & y, its angle, its angular velocity, and two Booleans that represent whether each leg is in contact with the ground or not.

* Rewards:

Reward for moving from the top of the screen to the landing pad and coming to rest is about 100-140 points. If the lander moves away from the landing pad, it loses reward. If the lander crashes, it receives an additional -100 points. If it comes to rest, it receives an additional +100 points. Each leg with ground contact is +10 points. Firing the main engine is -0.3 points each frame. Firing the side engine is -0.03 points each frame. Solved is 200 points.

* Starting State:

The lander starts at the top centre of the viewport with a random initial force applied to its centre of mass.

* Episode Termination:

The episode finishes if:

1. the lander crashes (the lander body gets in contact with the moon);
2. the lander gets outside of the viewport (x coordinate is greater than 1);
3. the lander is not awake. From the Box2D docs, a body which is not awake is a body which doesn’t move and doesn’t collide with any other body:

**------------------------------------------------------------------------------------------------**

1. **The Frozen Lake Problem:**

* Description:

Frozen lake involves crossing a frozen lake from Start(S) to Goal(G) without falling into any Holes(H) by walking over the Frozen(F) lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake.

* Action Space:

The agent takes a 1-element vector for actions. The action space is (dir), where dir decides direction to move in which can be:

0: LEFT

1: DOWN

2: RIGHT

3: UP

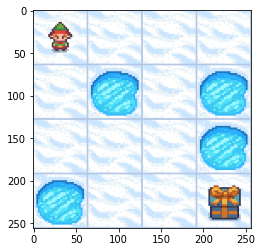
* Observation Space

The observation is a value representing the agent’s current position as current\_row \* nrows + current\_col (where both the row and col start at 0). For example, the goal position in the 4x4 map can be calculated as follows: 3 \* 4 + 3 = 15. The number of possible observations is dependent on the size of the map. For example, the 4x4 map has 16 possible observations.

* Rewards:

Reward schedule:

1. Reach goal(G): +1
2. Reach hole(H): 0
3. Reach frozen(F): 0

****

**------------------------------------------------------------------------------------------------**

1. **The Cliff Walking Problem:**

* Description:

The board is a 4x12 matrix, with (using NumPy matrix indexing):

* [3, 0] as the start at bottom-left
* [3, 11] as the goal at bottom-right
* [3, 1..10] as the cliff at bottom-centre

If the agent steps on the cliff, it returns to the start. **An episode terminates when the agent reaches the goal.**

* Actions:

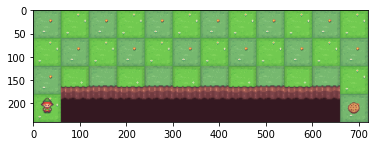
There are 4 discrete deterministic actions:

* 0: move up
* 1: move right
* 2: move down
* 3: move left
* Observations:

There are 3x12 + 1 possible states. In fact, the agent cannot be at the cliff, nor at the goal (as this results in the end of the episode). It remains all the positions of the first 3 rows plus the bottom-left cell. The observation is simply the current position encoded as flattened index.

* Rewards:

Each time step incurs -1 reward, and stepping into the cliff incurs -100 reward.

****

**------------------------------------------------------------------------------------------------**

1. **The Bipedal Walker Problem:**

* Description:

This is a simple 4-joint walker robot environment. There are two versions:

* Normal, with slightly uneven terrain.
* Hardcore, with ladders, stumps, pitfalls.

To solve the normal version, you need to get 300 points in 1600-time steps. To solve the hardcore version, you need 300 points in 2000-time steps.

* Action Space:

Actions are motor speed values in the [-1, 1] range for each of the 4 joints at both hips and knees.

* Observation Space:

State consists of hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar rangefinder measurements. There are no coordinates in the state vector.

* Rewards:

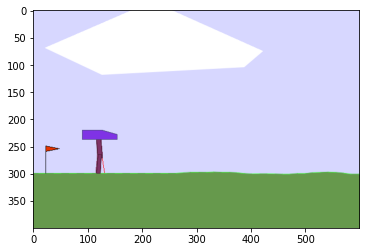
Reward is given for moving forward, totaling 300+ points up to the far end. If the robot falls, it gets -100. Applying motor torque costs a small amount of points. A more optimal agent will get a better score.

* Starting State:

The walker starts standing at the left end of the terrain with the hull horizontal, and both legs in the same position with a slight knee angle.

* Episode Termination:

The episode will terminate if the hull gets in contact with the ground or if the walker exceeds the right end of the terrain length.



**---------------------------------------------------------------------------------------------------------**

* **LAB 5:**

# Aim: Perform Logistic Regression on SMS Spam Dataset by following the instructions.

---------------------------------------------------------------------------------------------------------

**Description:**

Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set.

A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. For example, a logistic regression could be used to predict whether a political candidate will win or lose an election or whether a high school student will be admitted or not to a particular college. These binary outcomes allow straightforward decisions between two alternatives.

A logistic regression model can take into consideration multiple input criteria. In the case of college acceptance, the logistic function could consider factors such as the student's grade point average, SAT score and number of extracurricular activities. Based on historical data about earlier outcomes involving the same input criteria, it then scores new cases on their probability of falling into one of two outcome categories.

Logistic regression has become an important tool in the discipline of machine learning. It allows algorithms used in machine learning applications to classify incoming data based on historical data. As additional relevant data comes in, the algorithms get better at predicting classifications within data sets.

Logistic regression can also play a role in data preparation activities by allowing data sets to be put into specifically predefined buckets during the extract, transform, load (ETL) process in order to stage the information for analysis.

Below is the function for logistic regression:



* E is log base
* X is the numerical value that needs to be transformed.

---------------------------------------------------------------------------------------------------------

**Source Code:**

1. Importing required libraries:

import pandas as pd

import zipfile

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from nltk.corpus import stopwords

import nltk

nltk.download('stopwords')

1. Reading .csv file

file\_encoding = 'cp1252'

df= pd.read\_csv('1spam.csv',usecols = ['label','message'], encoding=file\_encoding)

*file\_encoding = 'cp1252'* is setting the encoding of the file to cp1252. cp1252 is a character encoding used in Microsoft Windows to represent text in Western European languages such as English, French, German, Spanish, Portuguese, and Italian. It is used when reading text data from a file to ensure that the correct characters are decoded and represented in the program. If the encoding is not set correctly, special characters or non-English characters may not be read or displayed correctly in the program.

Output:

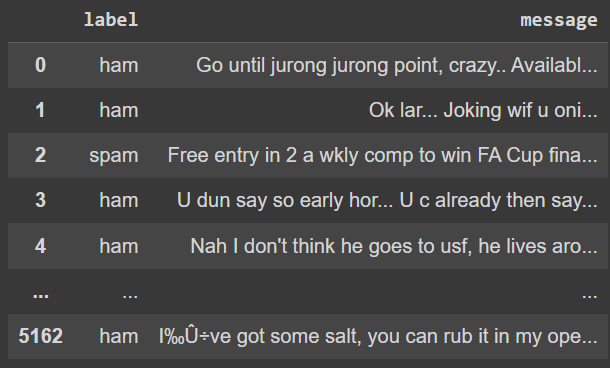


Fig. 1 SMS 1spam .csv file

1. Removing stop words:

stop\_words = set(stopwords.words('english'))

df['message'] = df['message'].apply(lambda x: ' '.join([word for word in x.split() if word.lower() not in stop\_words]))

df

Output:

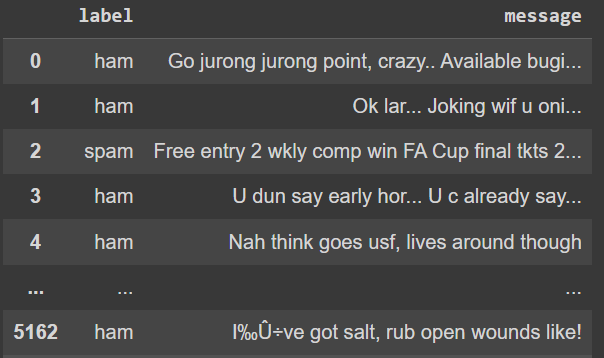


Fig. 2 SMS 1spam .csv file after removing stopwords

Performing text encoding:

1. Performing One-Hot text encoding:

# one-hot encoding

one\_hot = CountVectorizer(binary=True)

one\_hot\_encoded = one\_hot.fit\_transform(df['message'])

one\_hot.vocabulary\_

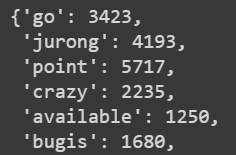
Output:

Fig. 3 output of vocabulary

1. Performing Countvectorizer text encoding:

# count vectorizer

count\_vectorizer = CountVectorizer()

count\_vectorized = count\_vectorizer.fit\_transform(df['message'])

1. Performing Tf-Idf text encoding:

# tf-idf

tf\_idf = TfidfVectorizer()

tf\_idf\_vectorized = tf\_idf.fit\_transform(df['message'])

1. Performing logistic regression with one-hot encoding after splitting dataset into train and test sets.

Test size = 0.2 means 20 % data used for testing, random state means data is chosen randomly.

# split dataset into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(one\_hot\_encoded, df['label'], test\_size=0.2, random\_state=42)

# logistic regression with one-hot encoding

lr\_one\_hot = LogisticRegression()

lr\_one\_hot.fit(X\_train, y\_train)

y\_pred\_one\_hot = lr\_one\_hot.predict(X\_test)

acc\_one\_hot = accuracy\_score(y\_test, y\_pred\_one\_hot)

print('Accuracy with one-hot encoding:', acc\_one\_hot)

Output:



Fig 4. Accuracy with one hot encoding:

1. Performing logistic regression with count vectorizer after splitting dataset into train and test sets

# split dataset into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(count\_vectorized, df['label'], test\_size=0.2, random\_state=42)

# logistic regression with count vectorizer

lr\_count = LogisticRegression()

lr\_count.fit(X\_train, y\_train)

y\_pred\_count = lr\_count.predict(X\_test)

acc\_count = accuracy\_score(y\_test, y\_pred\_count)

print('Accuracy with count vectorizer:', acc\_count)

Output:



Fig 5. Accuracy with count vectorizer encoding:

1. Performing logistic regression with tf-idf after splitting dataset into train and test sets.

# split dataset into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tf\_idf\_vectorized, df['label'], test\_size=0.2, random\_state=42)

# logistic regression with tf-idf

lr\_tf\_idf = LogisticRegression()

lr\_tf\_idf.fit(X\_train, y\_train)

y\_pred\_tf\_idf = lr\_tf\_idf.predict(X\_test)

acc\_tf\_idf = accuracy\_score(y\_test, y\_pred\_tf\_idf)

print('Accuracy with tf-idf:', acc\_tf\_idf)

Output:



Fig 6. Accuracy with tf-idf

-------------------------------------------------------X-------------------------------------------------------

* **LAB 6:**

# Aim: Dataset: SMS Spam or similar ones.

# Pre-processing Steps:  1. Remove Stop Words 2. Convert to lower case 3. Lemmatization 4. Stemming 5. Spell Checker (Correct Misspelled Words)

# Check the performance of the mentioned classifier with each and collective preprocessing steps. Conclude which Preprocessing steps are better to achieve the best accuracy with

# (i) LR and (ii) NB classifier.

---------------------------------------------------------------------------------------------------------

**Description:**

LR stands for Logistic Regression, which is a statistical method used to analyze a dataset with one or more independent variables that determine an outcome. It is commonly used for binary classification problems where the goal is to predict whether an instance belongs to one class or another.

NB stands for Naive Bayes, which is a probabilistic machine learning algorithm based on Bayes' theorem. It is commonly used for text classification problems such as spam filtering, sentiment analysis, and topic classification.

Both LR and NB are popular classifiers used in various machine learning applications, including natural language processing, computer vision, and predictive analytics.

---------------------------------------------------------------------------------------------------------

**Source Code:**

1. Importing required libraries:

import pandas as pd

import numpy as np

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer, PorterStemmer

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from autocorrect import Speller

import nltk

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

1. Loading .csv file

df = pd.read\_csv('1spam.csv', encoding='cp1252')

df = df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'])

df = df.rename(columns={'v1': 'label', 'v2': 'message'})

df['label'] = df['label'].map({'ham': 0, 'spam': 1})

1. Defining and implementing pre-processing functions:

# Define preprocessing functions

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

stemmer = PorterStemmer()

spell = Speller(lang='en')

def preprocess(text):

    # Convert to lower case

    text = text.lower()

    # Remove stop words

    words = word\_tokenize(text)

    words = [word for word in words if not word in stop\_words]

    # Lemmatize words

    words = [lemmatizer.lemmatize(word) for word in words]

    # Stem words

    words = [stemmer.stem(word) for word in words]

    # Correct spelling mistakes

    words = [spell(word) for word in words]

    return ' '.join(words)

1. Apply pre-processing to messages

# Apply preprocessing to messages

df['message'] = df['message'].apply(preprocess)

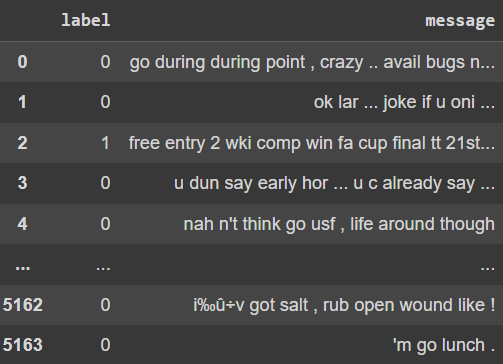
Output:

Fig. 1 data after preprocessing.

1. Splitting dataset into training and testing sets:

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['message'], df['label'], test\_size=0.2, random\_state=42)

1. Vectorize text using tf-idf:

tfidf = TfidfVectorizer()

X\_train\_tfidf = tfidf.fit\_transform(X\_train)

X\_test\_tfidf = tfidf.transform(X\_test)

1. Training logistic regression model and evaluating performance.

lr\_model = LogisticRegression()

lr\_model.fit(X\_train\_tfidf, y\_train)

lr\_score = lr\_model.score(X\_test\_tfidf, y\_test)

print("Accuracy of Logistic Regression Model: {:.2f}%".format(lr\_score\*100))

Output:



Fig 2. Accuracy of LR model

1. Train Naive Bayes model and evaluating performance.

nb\_model = MultinomialNB()

nb\_model.fit(X\_train\_tfidf, y\_train)

nb\_score = nb\_model.score(X\_test\_tfidf, y\_test)

print("Accuracy of Naive Bayes Model: {:.2f}%".format(nb\_score\*100))

Output:



Fig 3. Accuracy of NB model

-------------------------------------------------------X-------------------------------------------------------

* **LAB 7:**

# Aim: Embed developed Machine Learning Models to Web page.

---------------------------------------------------------------------------------------------------------

**Description:**

LR stands for Logistic Regression, which is a statistical method used to analyze a dataset with one or more independent variables that determine an outcome. It is commonly used for binary classification problems where the goal is to predict whether an instance belongs to one class or another.

NB stands for Naive Bayes, which is a probabilistic machine learning algorithm based on Bayes' theorem. It is commonly used for text classification problems such as spam filtering, sentiment analysis, and topic classification.

Both LR and NB are popular classifiers used in various machine learning applications, including natural language processing, computer vision, and predictive analytics.

---------------------------------------------------------------------------------------------------------

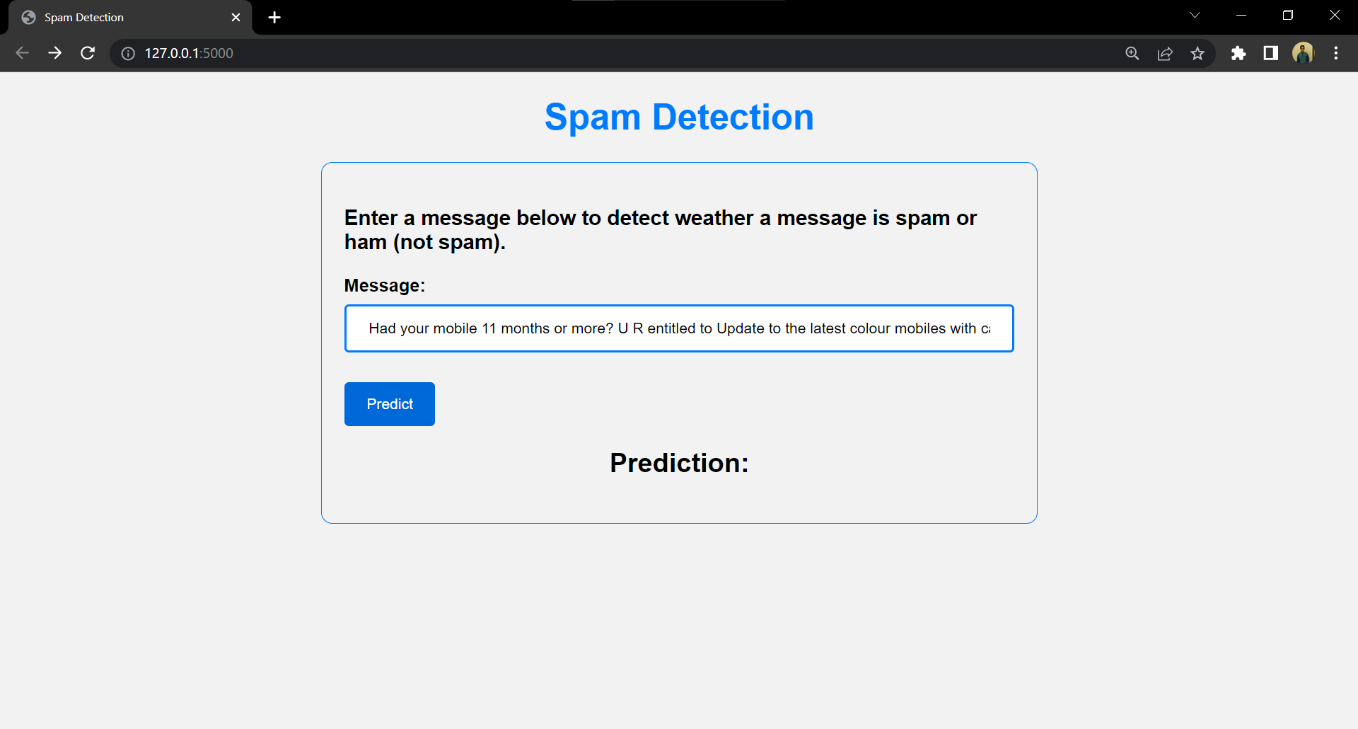
**Source Code:**

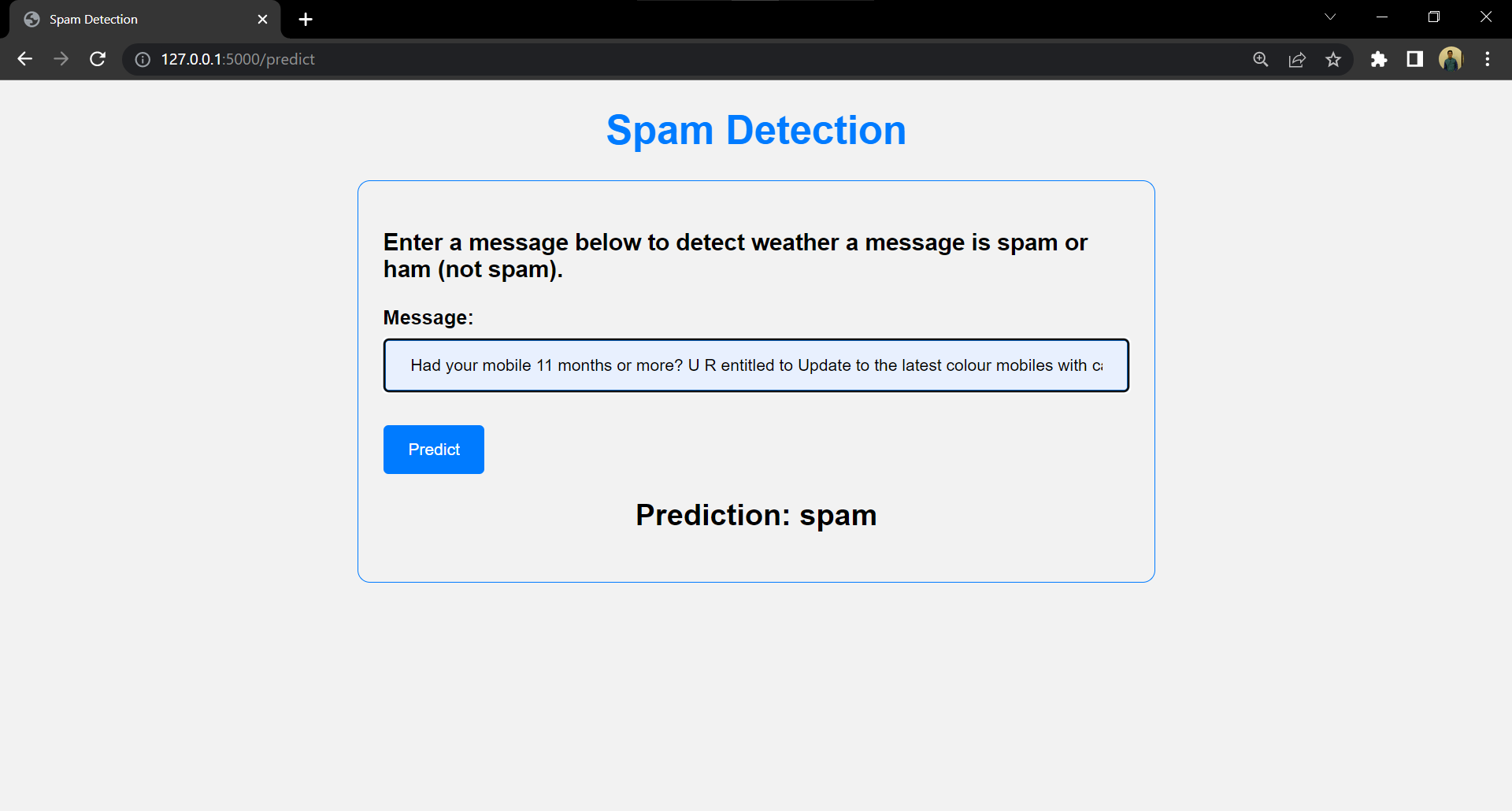
Complete source code link: [Google colab link]

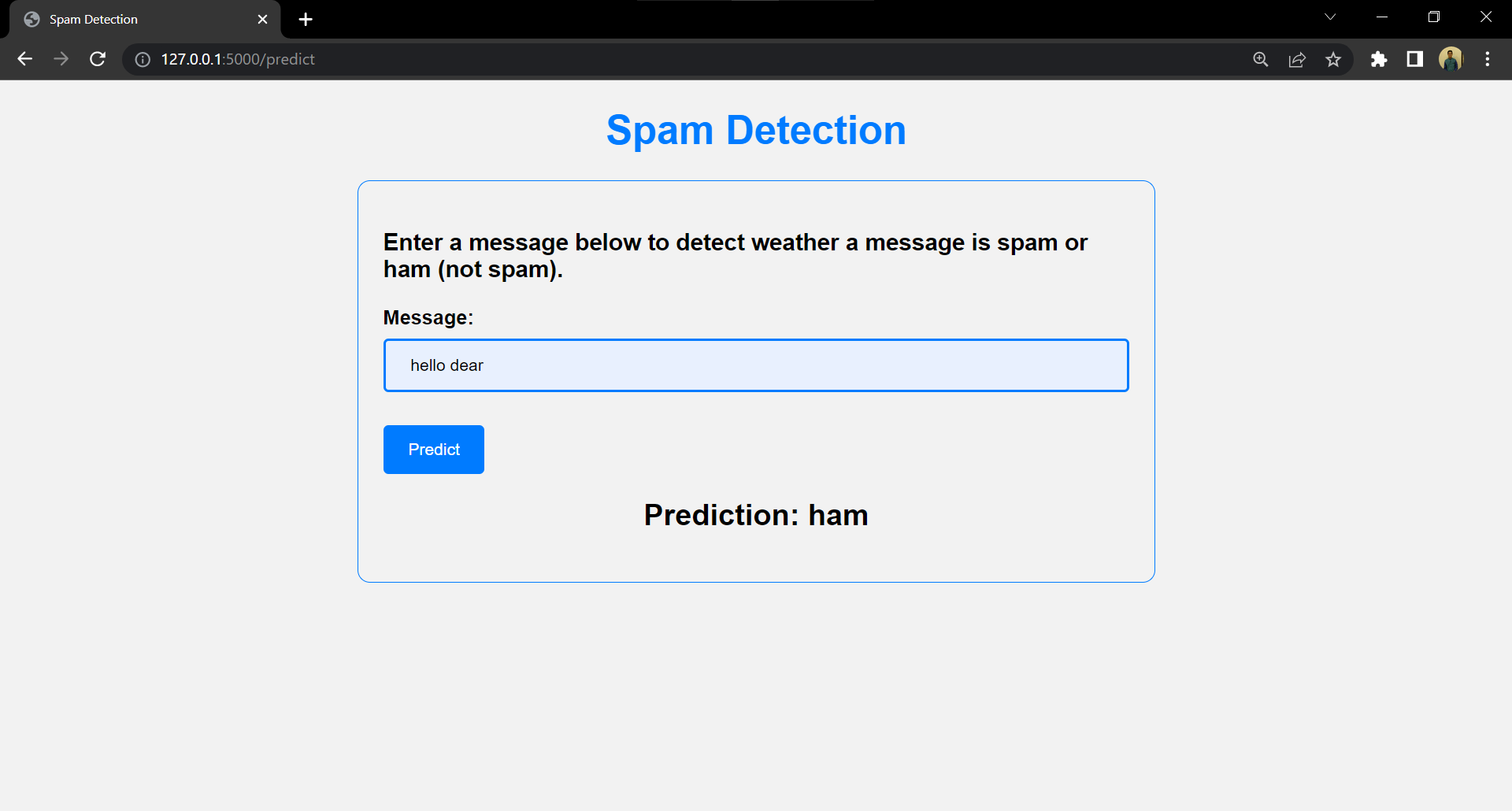
<https://colab.research.google.com/drive/1LhHrKsKjadkYlpiivLTVkVOMu2-f6OZv?usp=sharing>

The model is a **Logistic Regression model** using tf-idf encoding.

GitHub link : <https://github.com/charchit19/Spam-Classifier>

Outputs:





-------------------------------------------------------X-------------------------------------------------------

* **LAB 8:**

# Aim:

Deploy a machine learning model on a web interface where users can test their data. (i) single data item will be tested (ii) If multiple data exist, allow the user to upload the test data in a file (CSV, Zip).

---------------------------------------------------------------------------------------------------------

**Description:**

The aim of this project is to deploy a machine learning model on a web interface, providing users with the ability to test their data for prediction. The web interface will have two options for testing data:

1. Single Data Item: Users will be able to manually input a single data item for testing. This could be a text message, an image, or any other data type depending on the type of machine learning model being deployed.
2. Multiple Data Items: Users will also have the option to upload a file containing multiple data items for testing. The supported file formats for upload will include CSV and ZIP files, allowing users to conveniently test large datasets in bulk.

Upon submitting the data for testing, the web interface will process the data and pass it through the deployed machine learning model for prediction. The predicted results will then be displayed to the user, providing insights or classifications based on the model's predictions. This web interface will be user-friendly, allowing users to easily interact with the machine learning model and obtain predictions for their test data in an efficient and convenient manner.

---------------------------------------------------------------------------------------------------------

**Source Code:**

Complete source code link: [GitHub Link]

<https://github.com/charchit19/Spam-Classifier-2>

import joblib

import pandas as pd

import numpy as np

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer, PorterStemmer

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from autocorrect import Speller

import nltk

from flask import Flask, render\_template, request, send\_file

from nltk.stem import WordNetLemmatizer

# nltk.download('wordnet')

# nltk.download('stopwords')

# nltk.download('punkt')

app = Flask(\_\_name\_\_)

model = joblib.load("spam\_classifier.joblib")

vectorizer = joblib.load("vectorizer.joblib")

@app.route('/')

def index():

    return render\_template('index.html')

# Define preprocessing functions

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

stemmer = PorterStemmer()

spell = Speller(lang='en')

def preprocess(text):

    # Convert to lower case

    text = text.lower()

    # Remove stop words

    words = word\_tokenize(text)

    words = [word for word in words if not word in stop\_words]

    # Lemmatize words

    words = [lemmatizer.lemmatize(word) for word in words]

    # Stem words

    # words = [stemmer.stem(word) for word in words]

    # Correct spelling mistakes

    words = [spell(word) for word in words]

    return ' '.join(words)

def remove\_stop\_words(doc):

    words = word\_tokenize(doc)

    filtered\_words = [

        word for word in words if word.lower() not in stop\_words]

    return ' '.join(filtered\_words)

lemmatizer = WordNetLemmatizer()

def lemmatize\_sentence(sentence):

    # Tokenize the sentence into words

    words = nltk.word\_tokenize(sentence)

    # Lemmatize each word in the sentence

    lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]

    # Join the lemmatized words back into a sentence

    lemmatized\_sentence = ' '.join(lemmatized\_words)

    return lemmatized\_sentence

@app.route('/predict', methods=['POST'])

def predict():

    message = request.form.get("message")

    # file = request.files['file']

    file = request.files['file']

    if message:

        # Preprocess the input message

        message = preprocess(message)

        message\_vector = vectorizer.transform([message])

        # Make a prediction

        prediction = model.predict(message\_vector)[0]

        # Return the prediction as JSON

        return render\_template("index.html", prediction=prediction)

    else:

        df = pd.read\_csv(file, encoding='latin-1')

        df.rename(columns={'v1': 'label', 'v2': 'message'}, inplace=True)

        df['message'] = df['message'].apply(remove\_stop\_words)

        df['message'] = [lemmatize\_sentence(sentence)

                         for sentence in df['message']]

        inp = vectorizer.transform(df['message'])

        prediction = model.predict(inp)

        data = {'message': df['message'],

                'lebel': prediction}

        # Create DataFrame

        cf = pd.DataFrame(data)

        # Save the output.

        cf.to\_csv('pre.csv')

        return send\_file('pre.csv')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

Outputs:

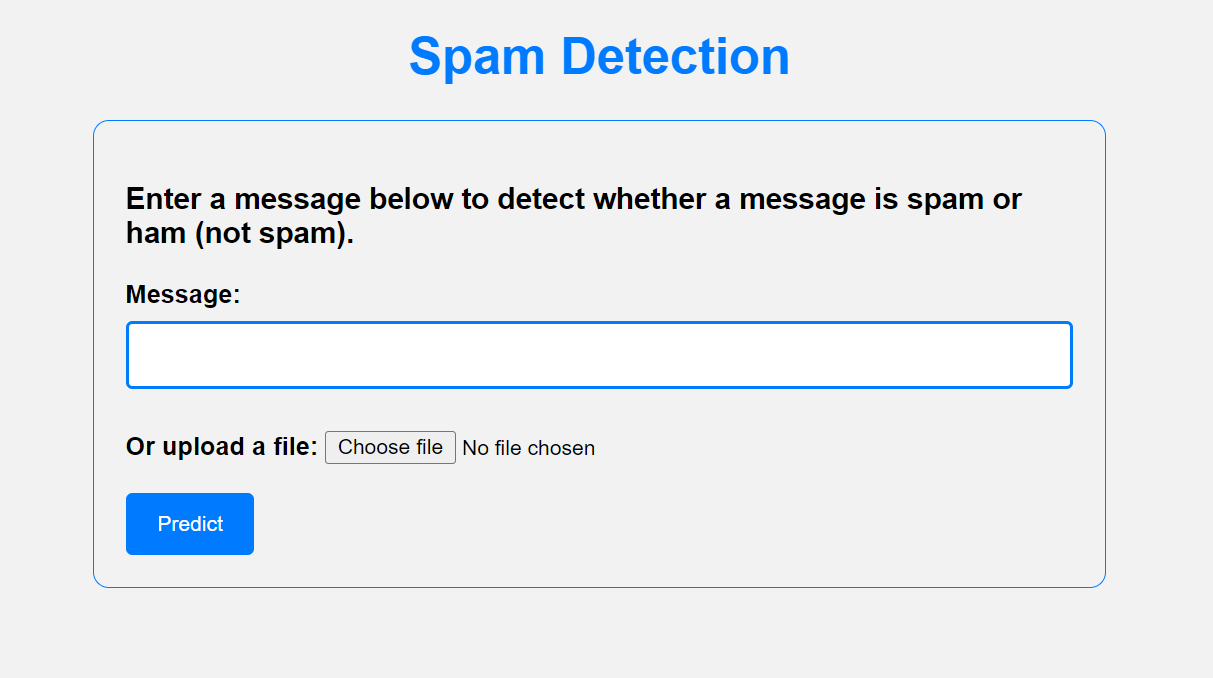


Fig1: Interface

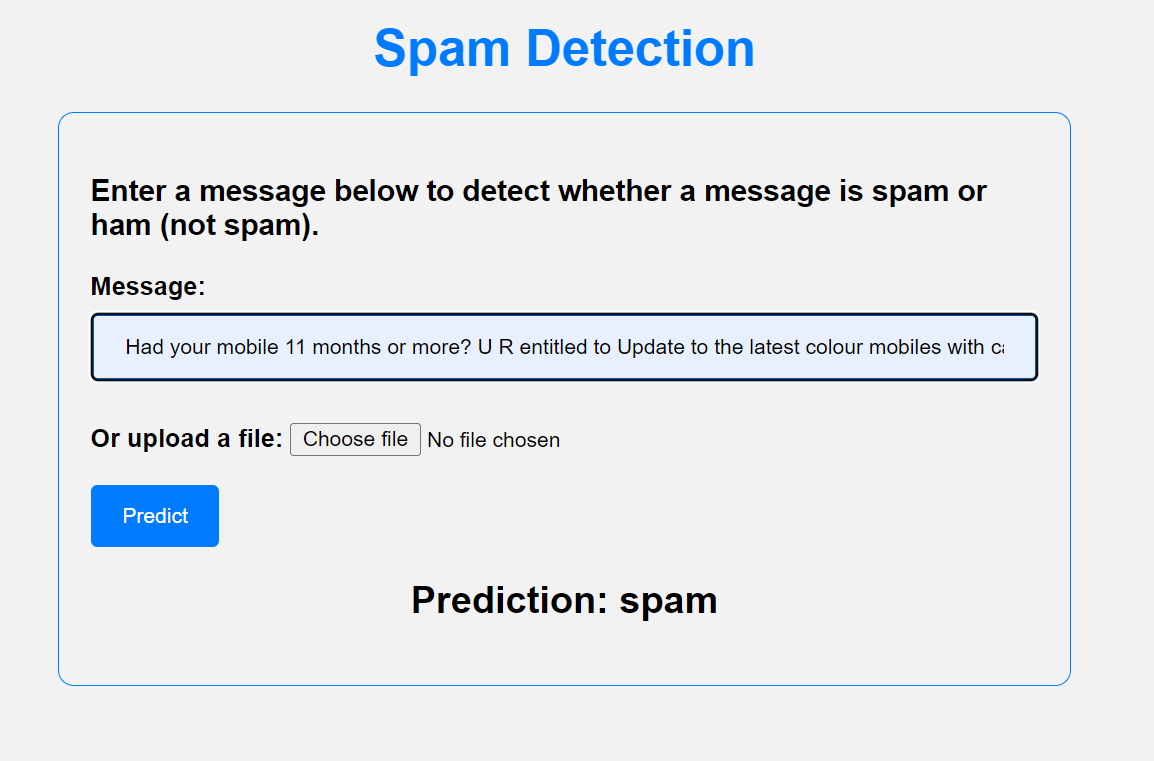


Fig 2: Prediction based on input

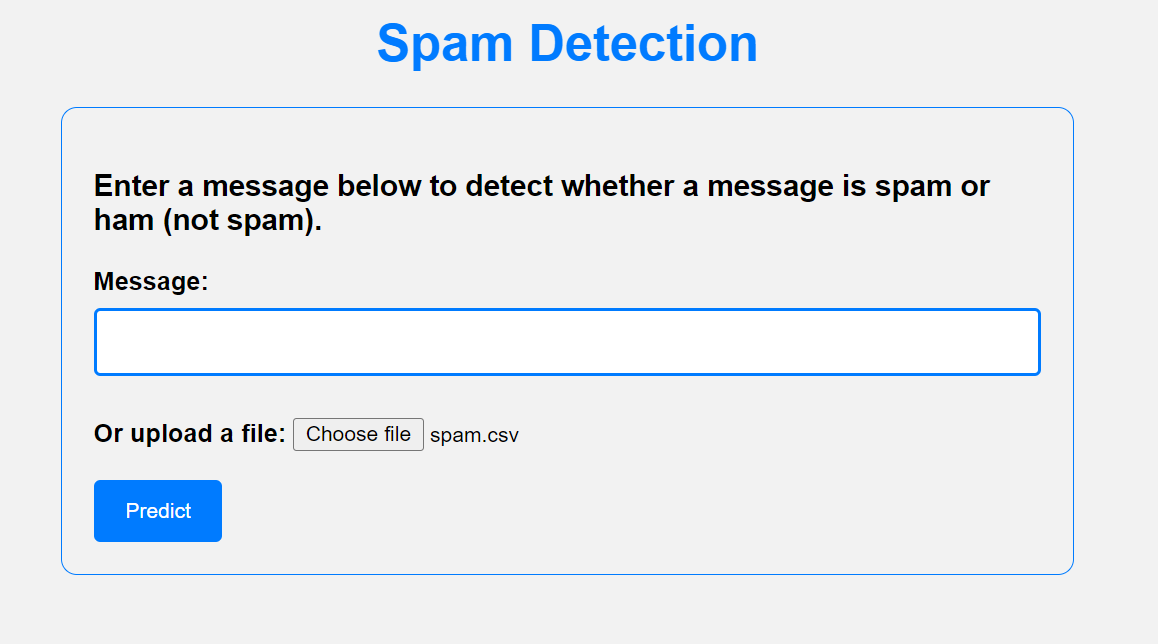


Fig 3: Uploading file to test

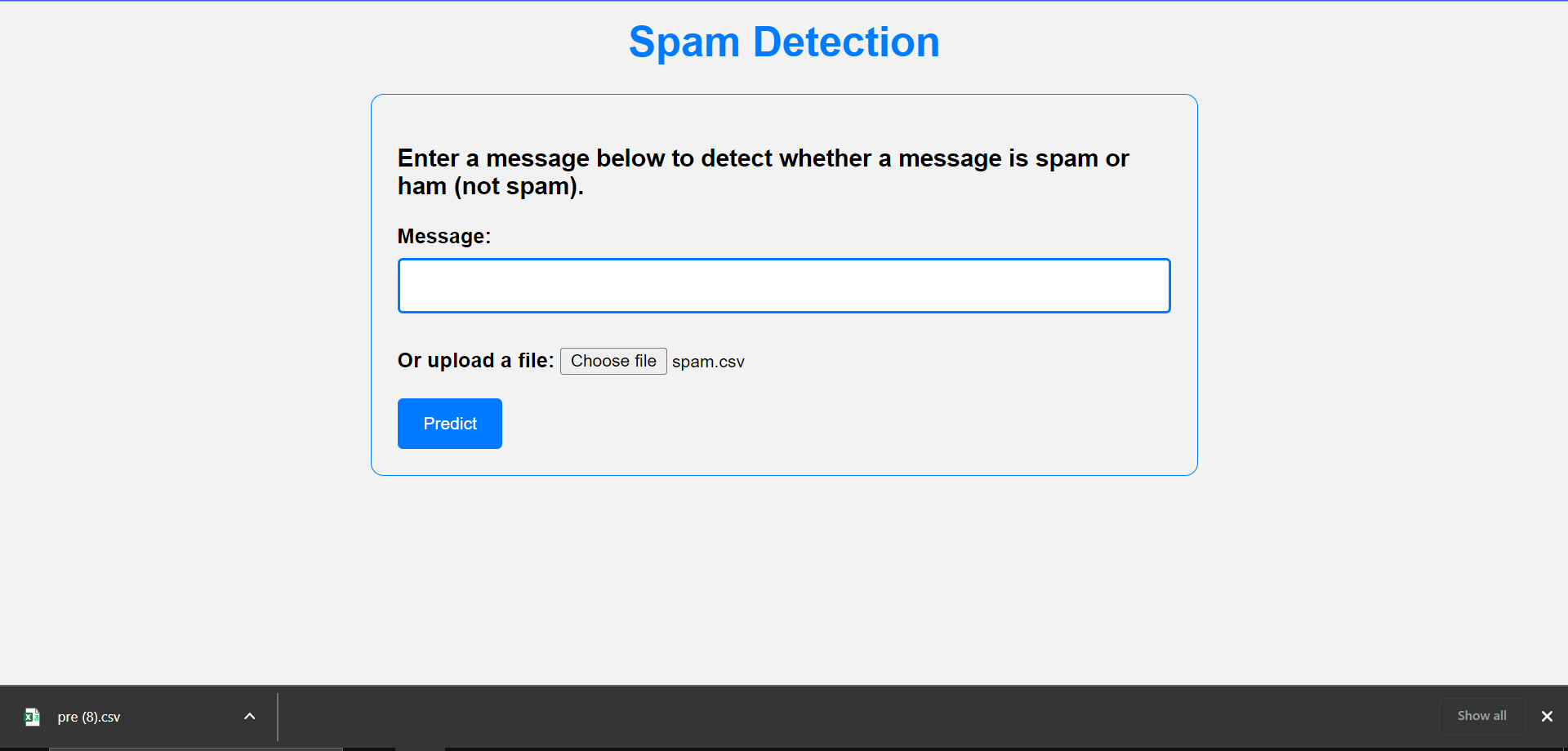


Fig4. File downloaded [predicted file]

-------------------------------------------------------X-------------------------------------------------------