TOP

Fake news classifier

Abdelrahman Ibrahim:

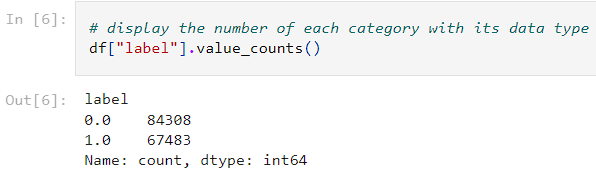
Ali Hisham: 211006277

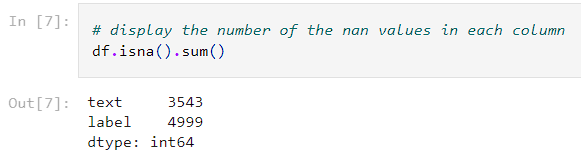
**Problem Statement:**

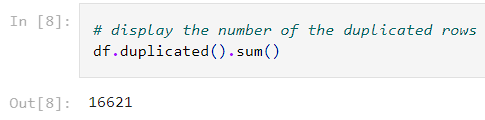
Fake news has become a threat in the digital age. Spreading like wildfire online, it shows discord and mistrust. Elections can be swayed by fabricated scandals, public health measures undermined by fictitious dangers, and social cohesion fractured by the manipulation of emotions through sensational headlines. However, our machine learning projects TOP is emerging as powerful tools in the fight against misinformation. By analyzing vast amounts of data, TOP can identify patterns in language, source credibility, and dissemination methods that are often hallmarks of fake news. This allows TOP to flag suspicious content, empowering users to make informed decisions about the information they consume.

**Data Description:**

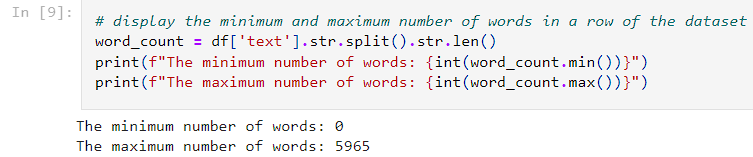
The data that’s used has many sources and we managed to collect it from different data storages.

as you can see, the raw data has a +150k corpus of text column (news article to be classified), a label data that has float classes. The class 1.0 means the article is true, the class 0.0 means it is false.

The label column has a biased class (0.0) that needs to be fixed.

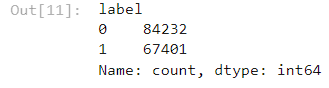
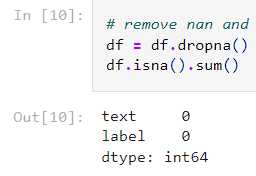
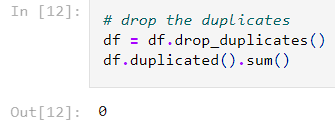
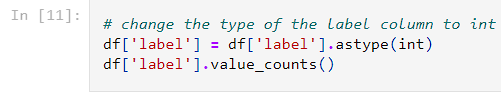
The total number of not a number and null values is high.

The number of duplicates is very high.

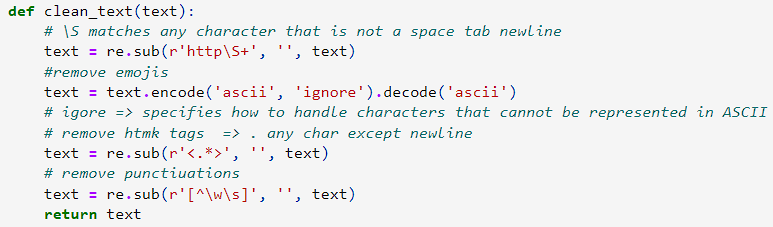
the min and max number of words in each row have a big difference which indicates that there are outliers.

**Data Preprocessing:**

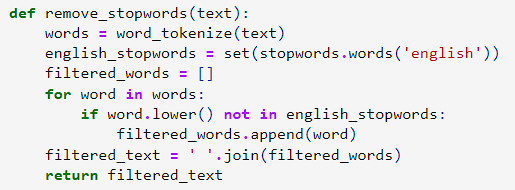
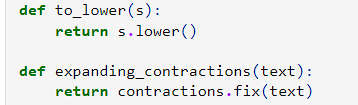
After analyzing the data, we had to eliminate the nan values, drop the duplicated rows, change the type of the label column from float to int.

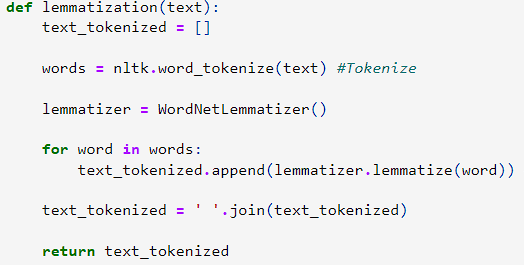
 

Then we had to shuffle the rows concerning the column orders because we saw that the data had many rows with false/true label in one place.

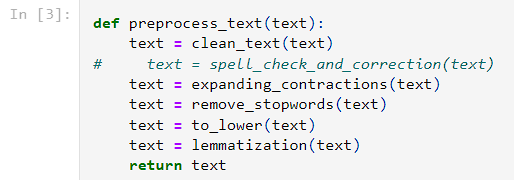
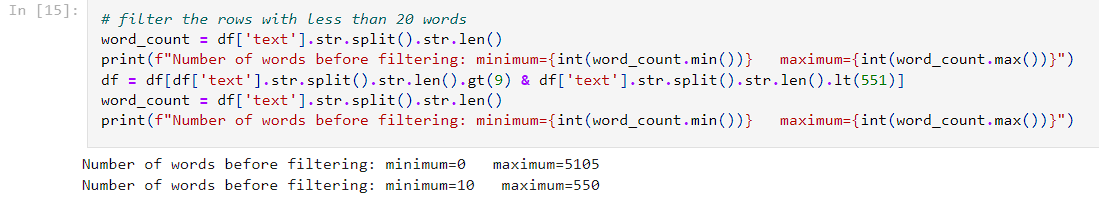
Then the most important step of all. Natural Language Processing (NLP) preprocessing techniques.

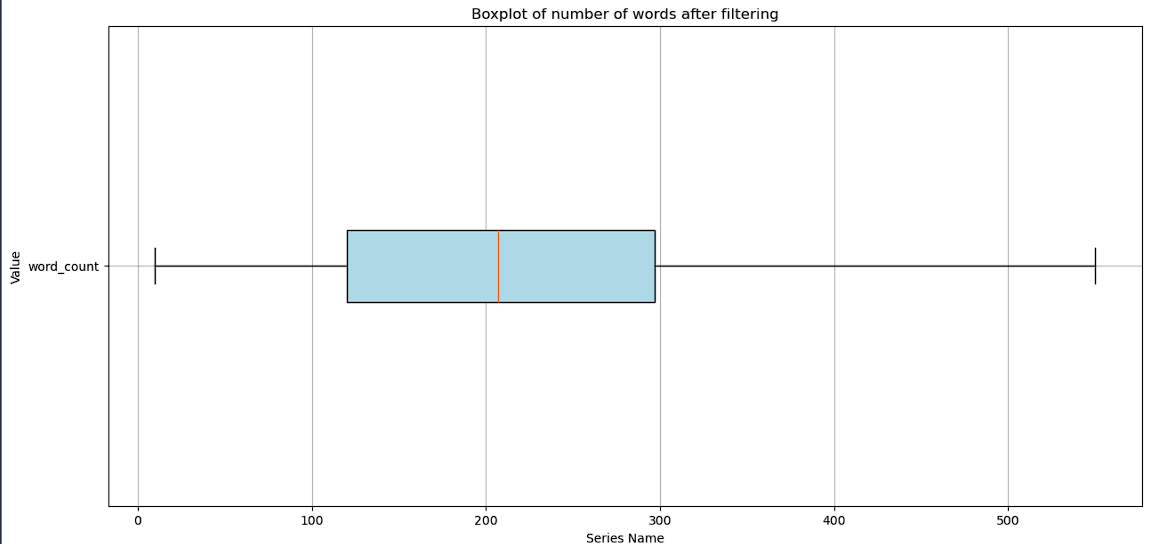
We defined a clean function that removes any unnecessary elements in one corpus, emojis, html tags, and punctuation.

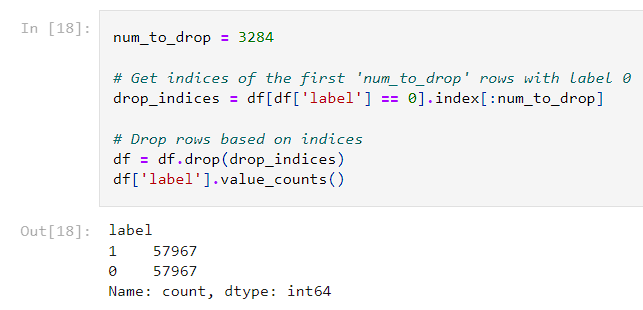
Removing the stop words is like removing a heavy load from ml model because the stop words have no meaning and does not represent any thing in the corpus.

The upper function makes all the letters in small cases. The lower one expands any contraction words in the corpus.

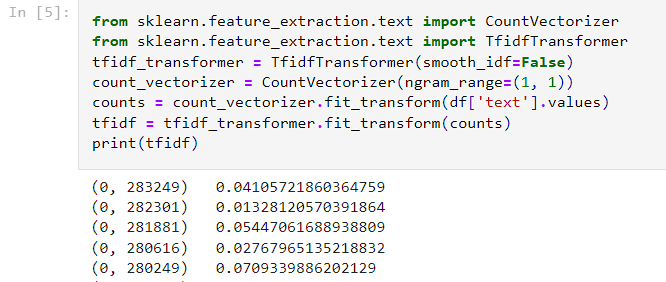
What is lemmatization? Its converting the word to its origin word. The lemmatization step is essential. Its like the stop words removing step, it removes a heavy load from the model because the model don’t care if the word is a noun, verb, etc.

Putting all those functions together to apply it on the whole dataset. The run time for this cell may take from 8 to 10 minutes so be patient.

We removed the outliers by keeping the rows that has at least 10 words and at most 550 words.

We can clearly see that in this boxplot.

Sacrificing a 3284 rows of false label to make the data equal.

initializing count vectorizer to count the frequency of the word in the whole dataset, then tokenizing each word, then applying and calculating TF-IDF with unigram (single word).

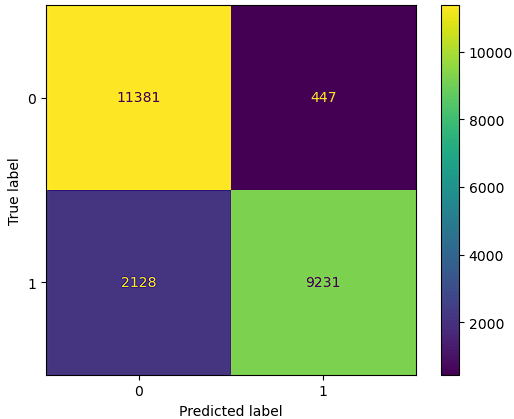
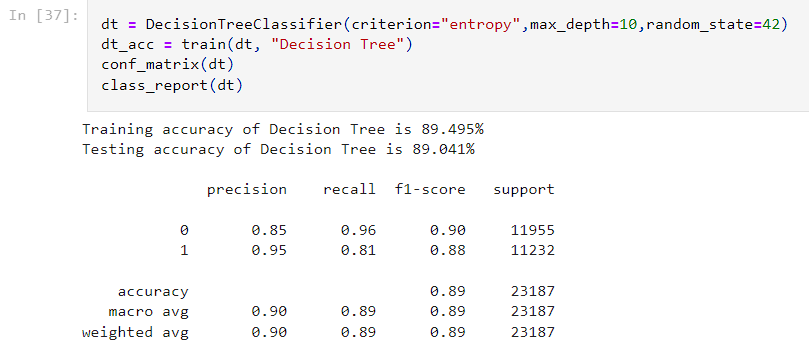
Splitting the data into train and test set with 20% for the test set and don’t shuffle it we already did above.

Now to data is ready for Training.

**Model fitting:**

We decided that we are going to use trees for the implementation.

Lets begin with decision tree.

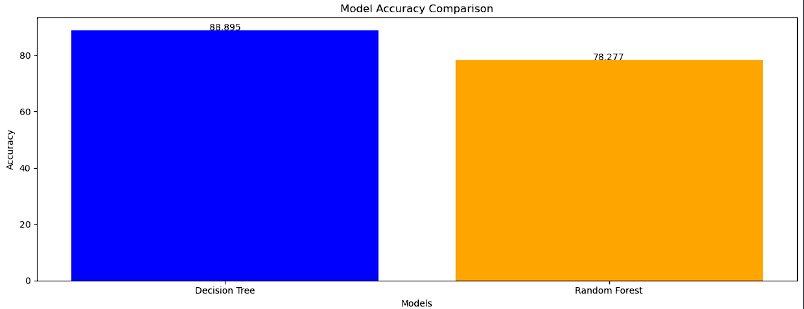
 

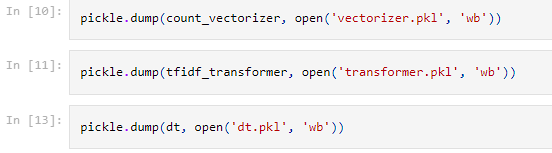
we choose entropy as a criterion and the max depth of the tree is 10 with random state = 42.

We trained the model, the test and train accuracy look very much close to each other. This means no overfitting. Also, the confusion matrix looks acceptable.

The Random Forest is an ensemble learning that contain many other models and they all predict the output and choose the majority. From the name of the model, it’s a bunch of trees together.

10 decision trees each is entropy criterion and have 100 depth, the accuracy is good but there is a big difference between the test and train accuracy. Still the decision tree better.



we will save the decision tree and the TF-IDF vectorizer and transformer as a pickle file because we will use them in a future use.

**TOP API:**

Yes, an application programming interface for our machine learning model. Why? Because we want to show the world out project that solves a big problem in the life. We used fastAPI platform to deploy TOPAPI

app = FastAPI(

    title="TOP API",

    docs\_url="/api/doc",

    description="A Saas machine learning model API developed to help in classifying suspicious and fake news articles all over the web"

)

We initialized an instance from FastAPI that contain our api name, description, url of the api documentation.

class article(BaseModel):

    text: str

vectorizer = pickle.load(open('vectorizer.pkl', 'rb'))

transformer = pickle.load(open('transformer.pkl', 'rb'))

model = pickle.load(open('dt.pkl', 'rb'))

we created a basemodel class that has an attribute text (news article to be classified). And we loaded the pickle files that we have saved earlier.

@app.get("/")

def root():

    return {"Health": "OK"}

a root decorator with get method to test the api, it return a JSON object says OK.

@app.post("/classify", tags=["Classify a news article"])

def classify(input: article):

the important endpoint in the whole file. The one that takes the input with type article (the basemodel class), to apply on it the preprocessing techniques and then applies TF-IDF vectorizer and transformer that we loaded from pickle file, then predicts the answer and return it as JSON with attribute “prediction” and has the value of the prediction (0 or 1), it returns from predict function as an array so we select the first index as seen below.

    counts = vectorizer.transform(text)

    tfidf = transformer.transform(counts)

    prediction = model.predict(tfidf)

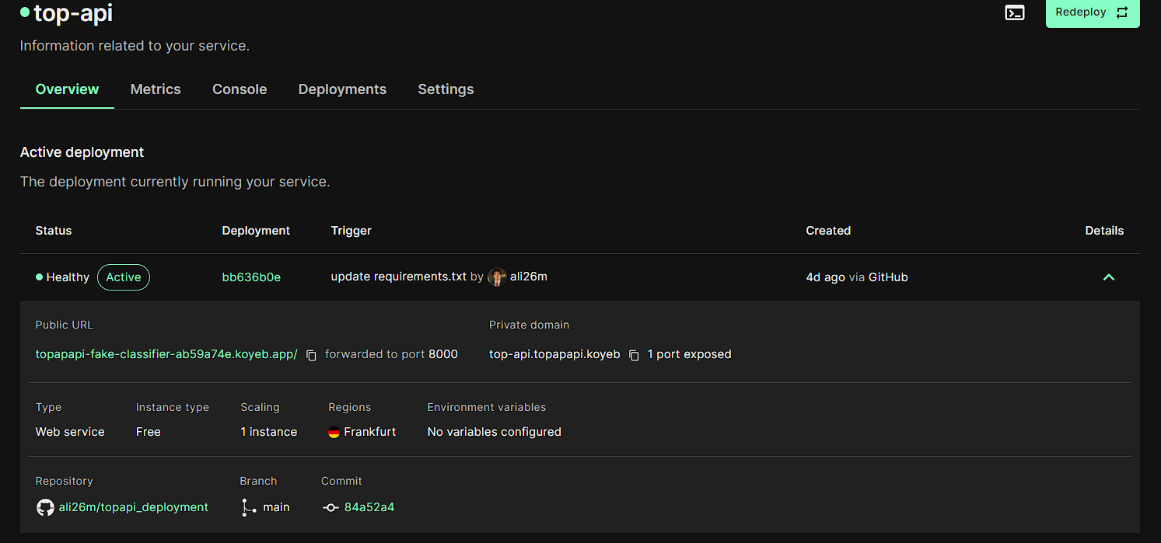
    return {"prediction": str(prediction[0])}

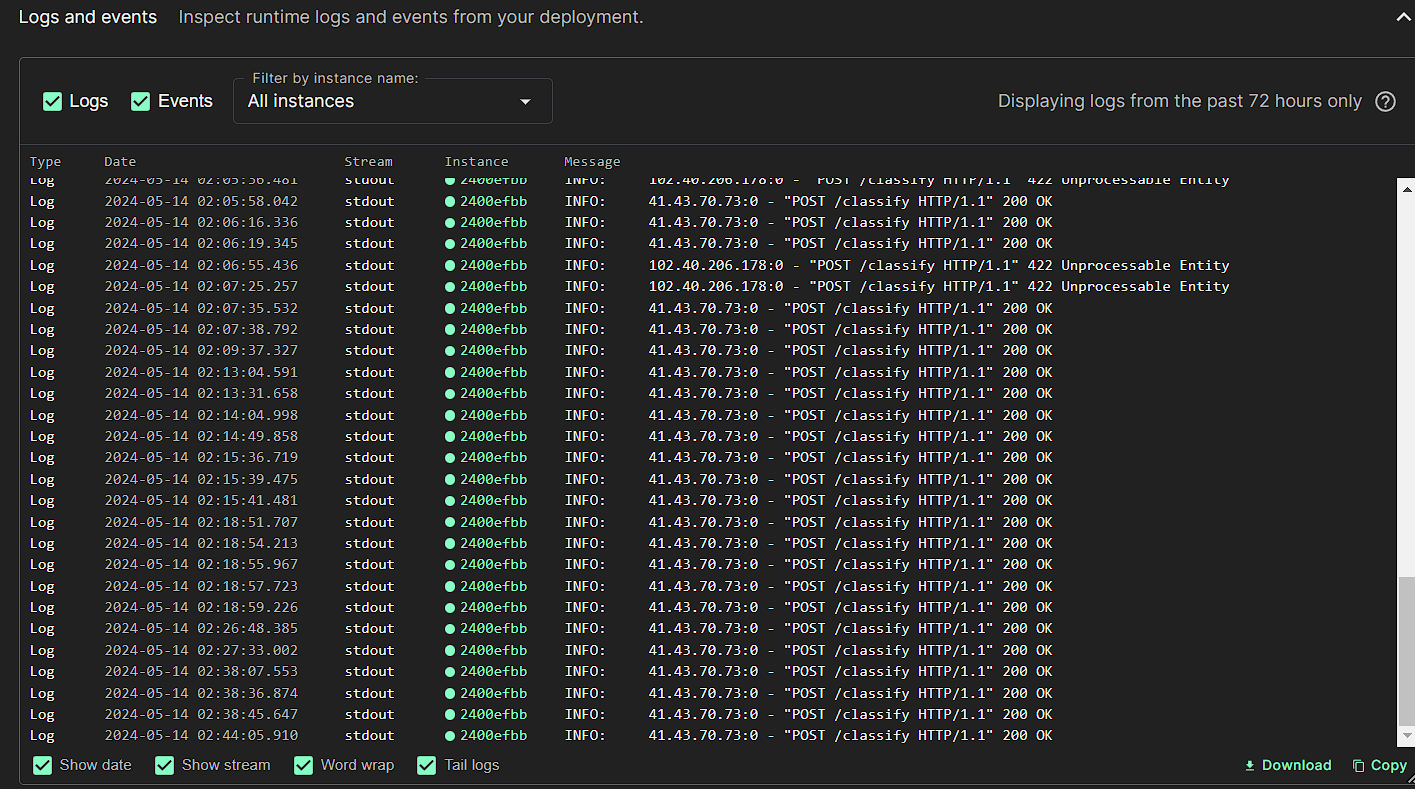
then we try to run the api locally using uvicorn by typing this command in the terminal

uvicorn topapi:app --reload

topapi is the name of the file, app is the name of fastAPI object, --reload means to auto refresh the api when there is a change saved in the python file. It gives us a local url to test the endpoints of the api.

fastAPI gives us automatic documentation for our api, it’s a great feature. You can access it by typing /api/doc at the end of api url.



We didn’t give up at this point we deployed the api in the cloud. Yes, its running right now and healthy. We choose KOYEB, platform as a service that deploy the api in free plan, it requires the files to be on github repo. And then with a simple click we deployed it.

The logs of the running instances show the most recent logs.

The active url of our topAPI:

<https://topapapi-fake-classifier-ab59a74e.koyeb.app/>

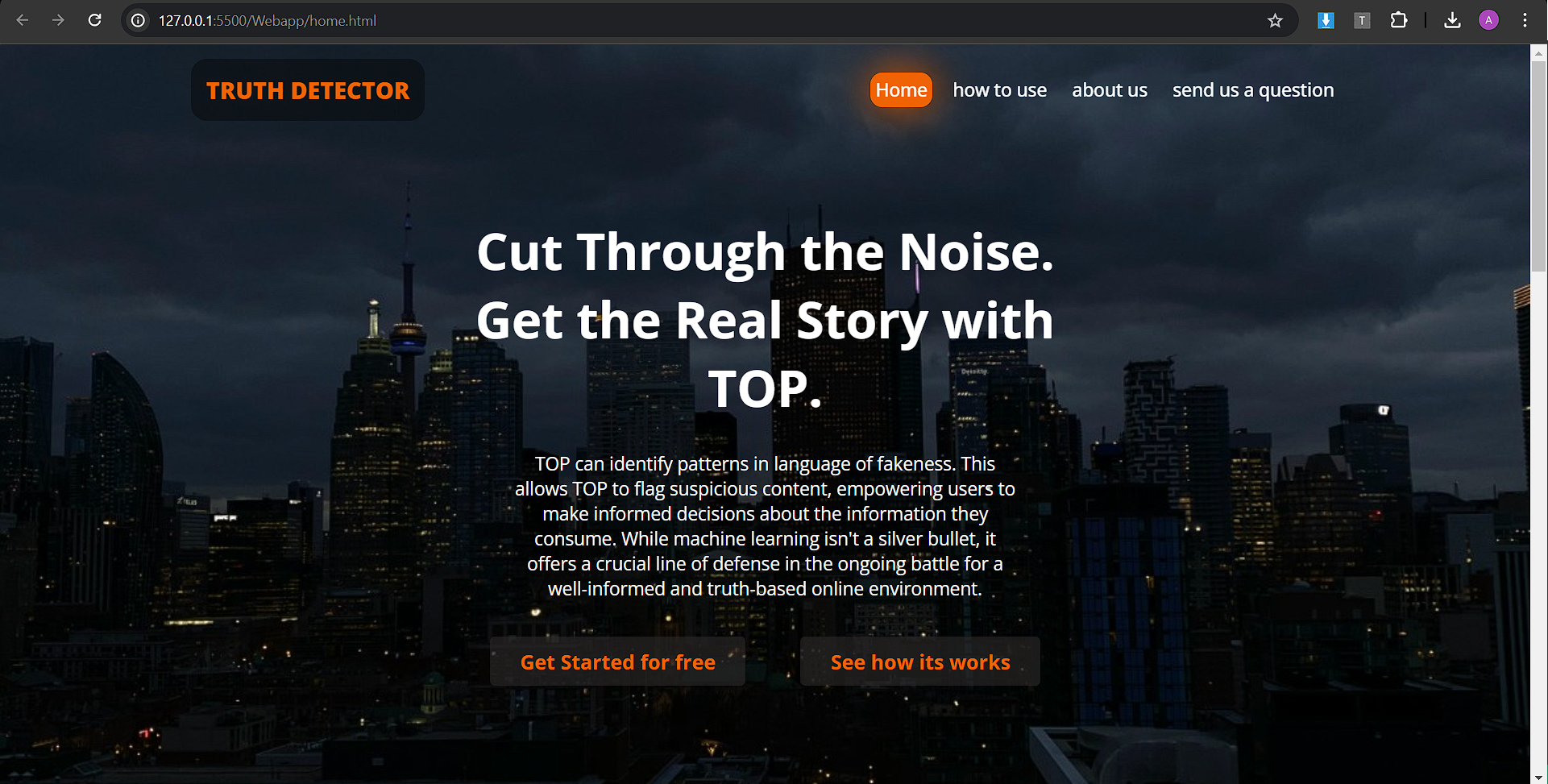
To make TOPAPI more accessible we added it on RapidAPI a platform that has many many apis for use. It make you add logo or description and pricing (but we are free don’t worry).

The url of RapidAPI of our TopAPI:

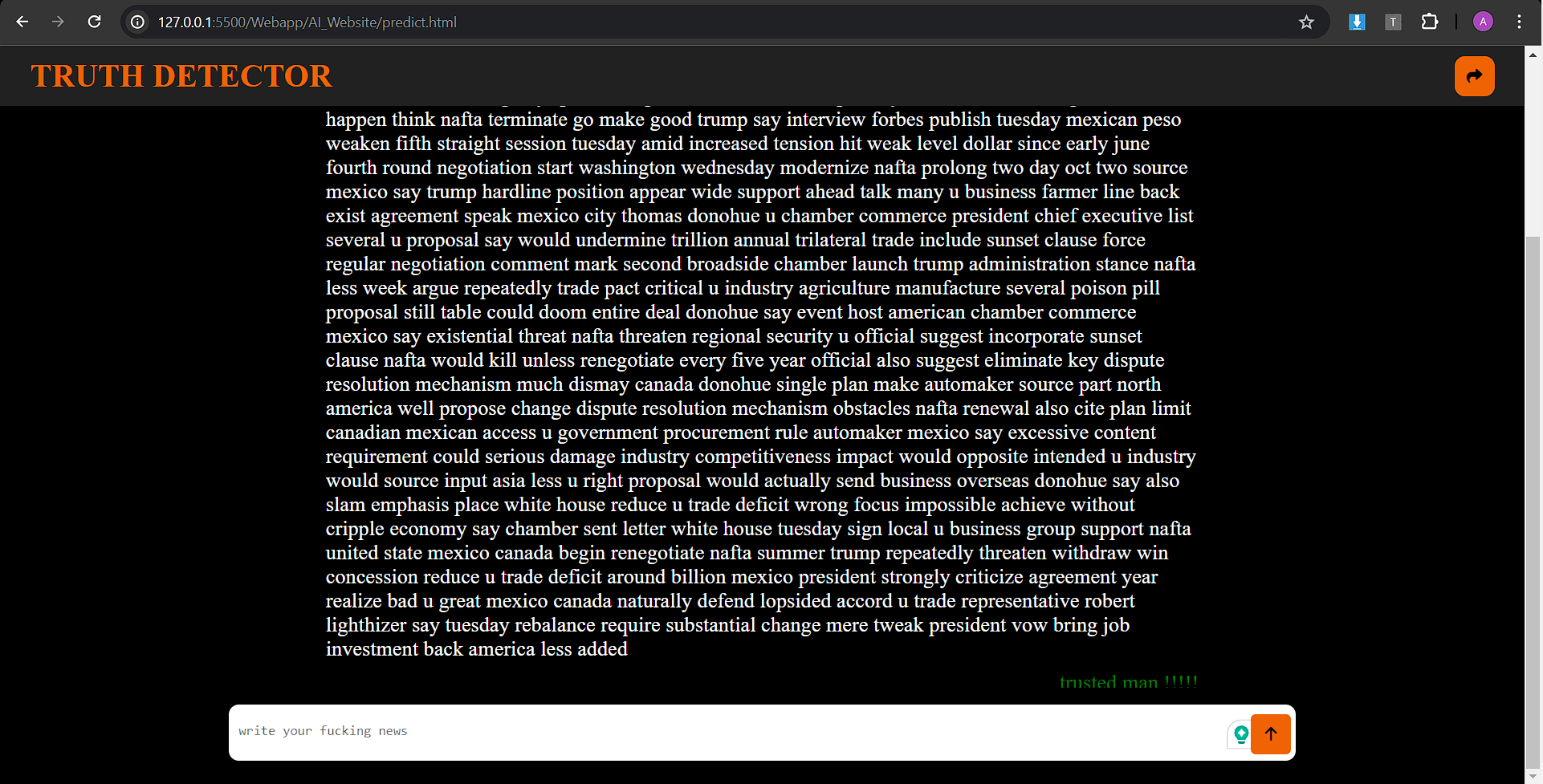
<https://rapidapi.com/alihisham26m/api/top9>

after this important step we are able to use the endpoints as we wish. So we developed a web app for the model. It acts like a chatbot.

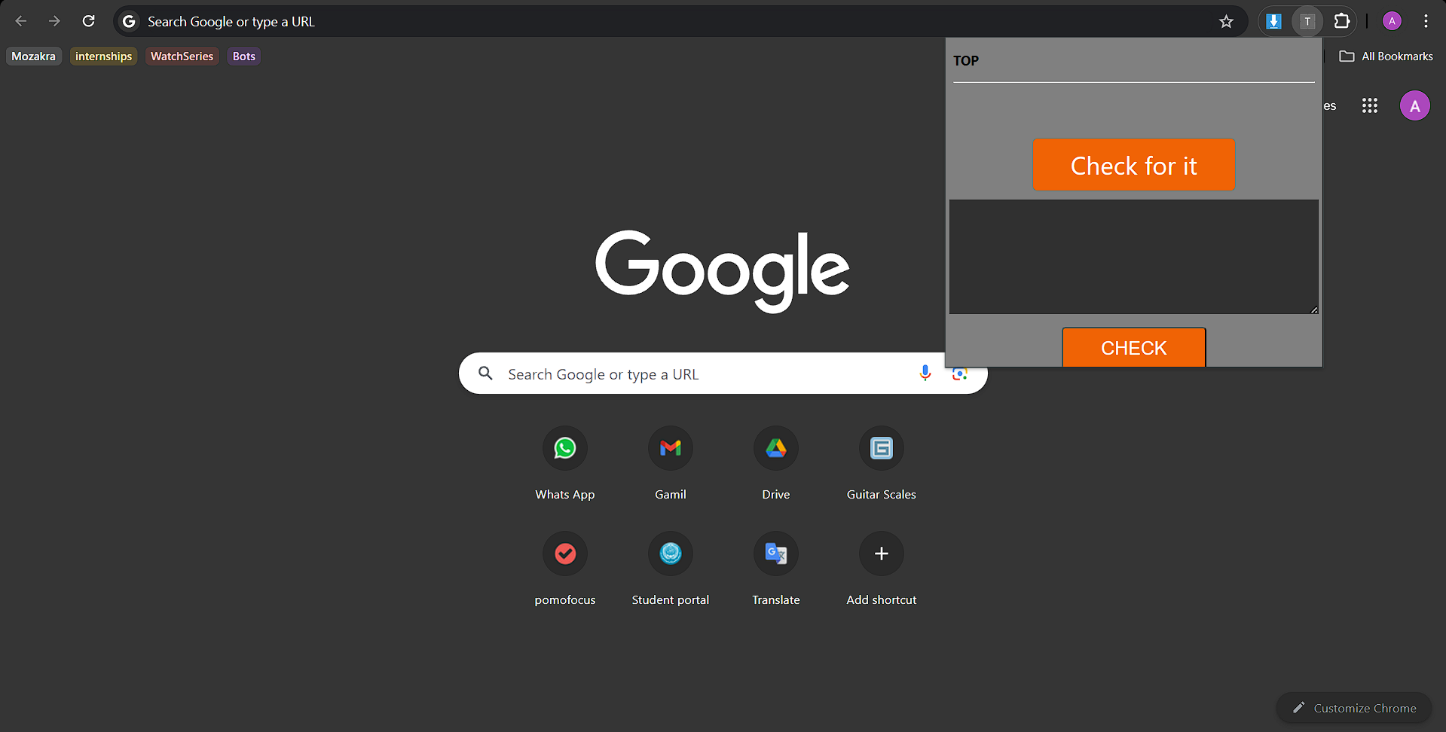
A web app to access it and detect the truth of an article you saw. With homepage that has tutorial videos, contact us, and showcasing our products and solutions.



Enter the chat area to begin the journey of classification and talk to our model using api and it will response with the ugly truth.



Also we developed a chrome extension. To make it easy on the user to access our model’s api with just simple steps.



Just copy the article you saw, paste it in the textarea, hist sheck and the result will display.

Thank you for reading.