DCU School of Computing Final Year Project Technical Guide

Project Title: Fake News Detector

CASE4

Student Name:David Talan

Student ID: 14387991

Student E-mail: david.talan2@mail.dcu.ie

Project Supervisor: Yvette Graham

E-mail: yvette.graham@dcu.ie

Motivation

The Rise of Fake News

The basis of Fake News isn’t something that was only invented a few years ago thanks to a certain individual, but has been around since humans learned that they can deceive others simply by lying. Falsified and unreliable information has played a major part in human history and still continue to do so today.

My motivation for this project is due to the recent rise in popularity of the term ‘Fake News’, mainly because of Donald Trump and his advocates. He was/is known to dismiss any type of news that did not fit his agenda, while also spouting falsified himself information in the process.

I was also inspired from my personal experience of being constantly fooled into taking article titles as fact, instead of clicking into it and reading its contents. It would then lead me into telling that false information to others, therefore helping the cause of spreading fake news.

The Dangers of Fake News

The spread of misinformation in general can be a very dangerous thing, especially when dealing with those that do not have immediate access to fact check it. An example would be the spread of misinformation about vaccinations, putting the lives of young children in danger. It can also come in the form of propaganda, used for political gain to sway the opinions of normal people. This could lead to conflict within the country and endanger people.

I wanted to create this tool to prevent the potential threat that fake news can bring if not enough people are aware of it. Anyone who uses the internet and reads articles, or someone who just came across a title they’re 100% convinced by, can use this web application to hopefully guide them to the truth.

Research

There were many goals to the research that I conducted. One of these goals was to find out if other groups have already created fake news classifier to see if it was possible. To my surprise, a good amount of people have already done so, their methods ranging from using Neural Networks, to Sentiment Analysis, and also Text Classification. The big difference is that these projects were done with multiple people, so I had to come up with a way to creatively detect fake news.

Another goal that I set out to do was to look for a large dataset of fake news articles. Scraping the web for these types of articles and labelling would have taken a really long time. By this time, I already decided to use Machine Learning to classify the articles and it meant that I needed a large dataset so I spent some time looking for a suitable one. I did find a dataset from Kaggle which contained thousands of articles and was already labelled. I also found another dataset in the web that also contained thousands of articles, but differed to the Kaggle dataset when it came to the labelling.

Methodology

I based my research from two sources of data; articles from other groups that have done research on the same topic, and blogs.

My search strings included “fake news” and executed them both on Google Scholar and Google’s search engine. The articles and blogs that returned, although relevant, were too general and was more about explaining what it was and its effects. I expanded the search string to “fake news detection/classification” and it resulted in more technical articles and blogs talking about their approach into detecting fake news.

Inclusion/Exclusion Criteria

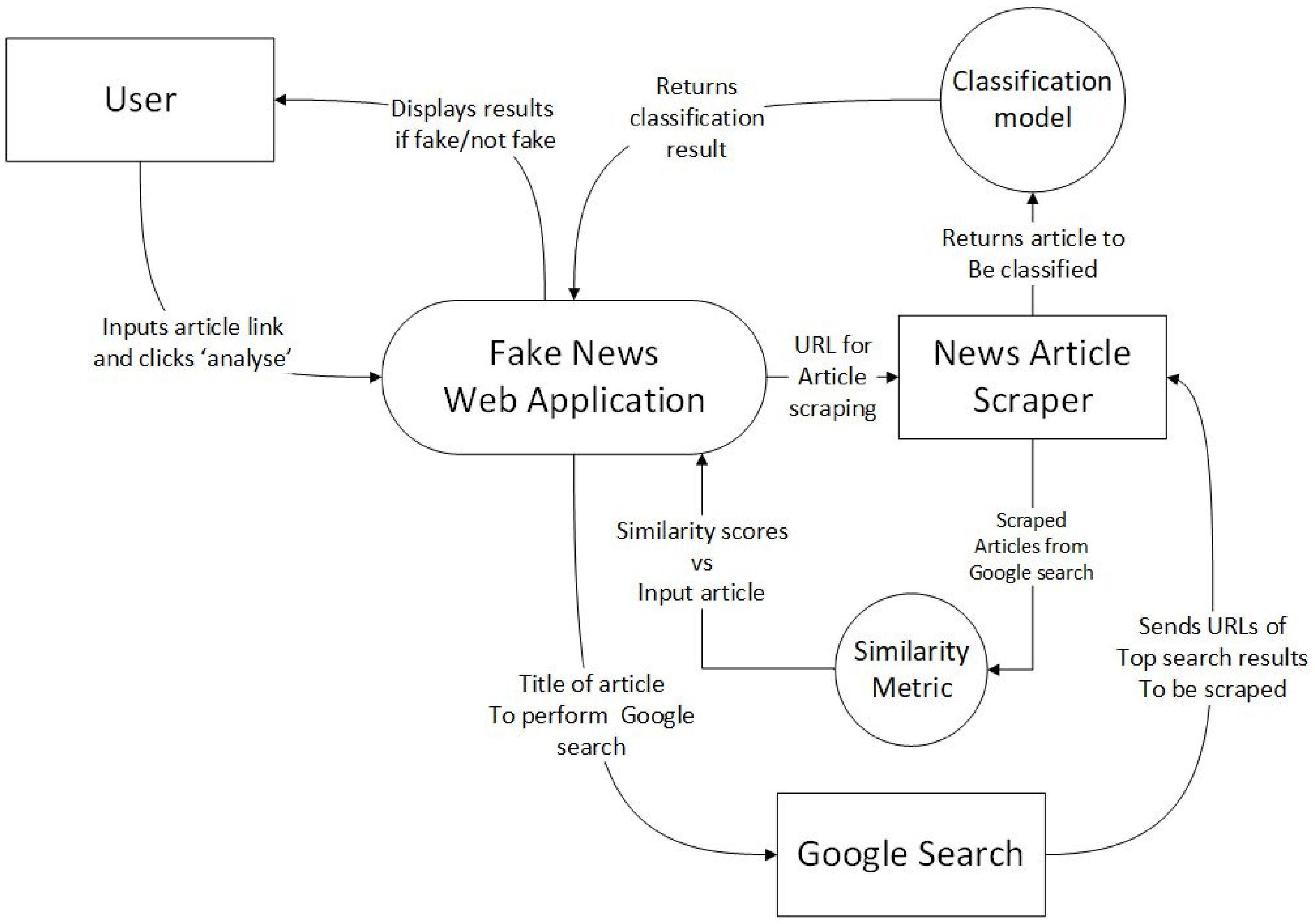
I included the articles and blogs whose processes I was able to understand and technologies I was familiar enough with. At this point, I was thinking of using Text Classification, a subtopic in Machine Learning, for my project. I excluded the articles that only explained the meaning of fake news, and included those that mentioned classification and machine learning. I also included those that were using languages that I’m more familiar to i.e. Python, Java.

With the collected articles and blogs from my research, I was able to get a good idea on how I would approach my project and the technologies I would end up using.

Design

High Level Design

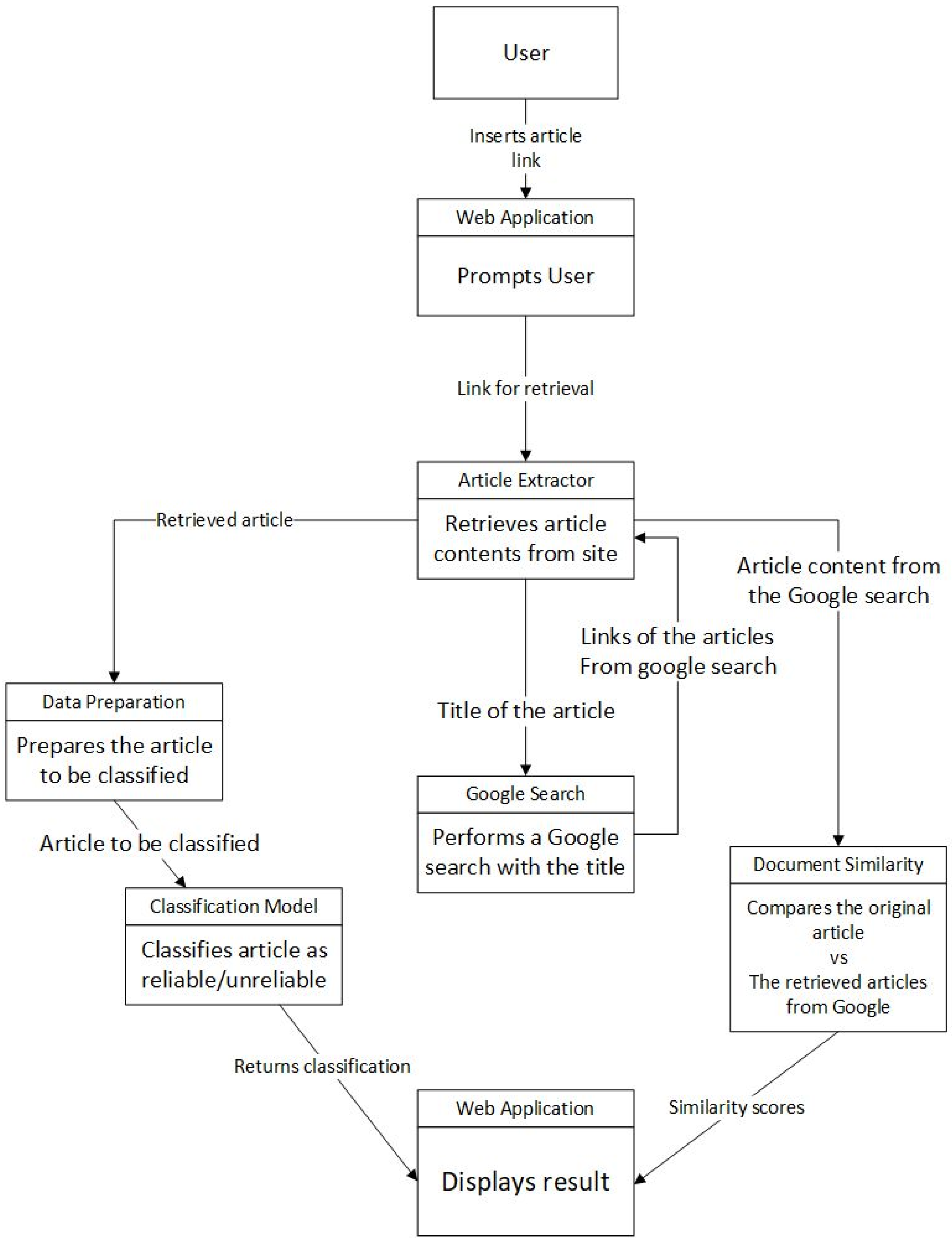
Figure 1.1 - Context Diagram


User
Displays results if fake/not fake
Returns classification result
Classification model
Returns article to Be classified
Inputs article link and clicks 'analyse'
Fake News Web Application
URL for Article scraping
News Article Scraper
Similarity scores vs Input article
Scraped Articles from Google search
Sends URLs of Top search results To be scraped
Similarity Metric
Title of article To perform Google search
Google Search

This context diagram shows a high level overview of the system and all the entities and components that interact with it.

Compared to my initial context diagram, you can clearly see the removal of the database, which I planned to store the previous queries and and classification results so they can be retrieved quickly. I initially thought that classifying an article to either be fake or not would take longer but it was actually done pretty quickly.

Figure 1.2 - Data Flow Diagram

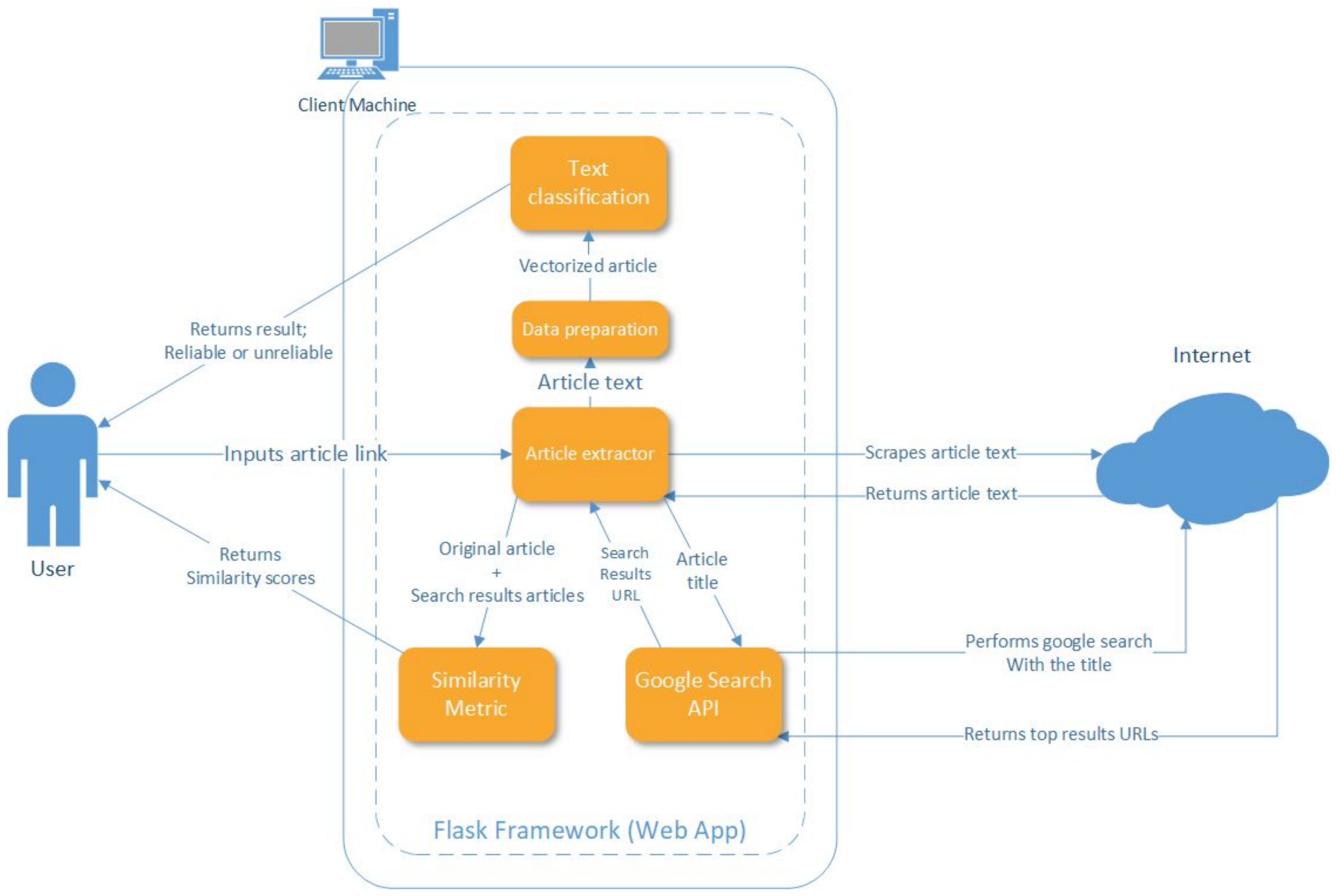

User
Inserts article link
Web Application
Prompts User
Link for retrieval
Article Extractor
Retrieves article contents from site
Retrieved article
Article content from the Google search
Links of the articles From google search
Title of the article
Data Preparation
Prepares the article to be classified
Article to be classified
Google Search
Performs a Google search with the title
Document Similarity
Compares the original article vs The retrieved articles from Google
Classification Model
Classifies article as reliable/unreliable
Returns classification
Similarity scores
Web Application
Displays result

The data flow diagram represents the flow of all sorts of data between the components and ending at the web application displaying the results.
The biggest difference between this and my initial data flow diagram is the removal of both the database and the NLP Tool. I was planning on using Natural Language Processing for the topic segmentation. However, after further research, I decided on a different approach without NLP, using Term Frequency - Inverse Document Frequency (TF-IDF) with text classification to determine the article's “fakeness”.

After working on the system, it was made clear to me that the article extraction process contains the most data flowing in and out of it. This diagram shows the two main processes that is happening in the backend; the classification of the article and the calculation of how similar the articles from Google search are to the user input article.

System Architecture

Figure 1.3



Like the previous two diagrams, the architecture diagram also underwent some changes. The web app is run locally on a Flask framework and accesses the internet when scraping the articles and searching for them. The database was also removed, thinking I was gonna store the classification model in the cloud. However, I found out that I'm able to deploy the machine learning model onto Flask so I didn't feel the need to implement one.

Implementation

Front End

For the user interface, I decided on a design that is appealing, easy to use, easy to navigate, and has clear instructions on how to use it. An input box, both for links and text respectively, prompts the user to enter either the URL of the article they want to check, or copy and paste the body of the article into the box below it. The home page also provides a quick explanation on how the article is deemed reliable or unreliable, and also a section about myself.

The tools and technologies I used for the front end are: **HTML, CSS, Javascript** and is built on a **Flask framework**.

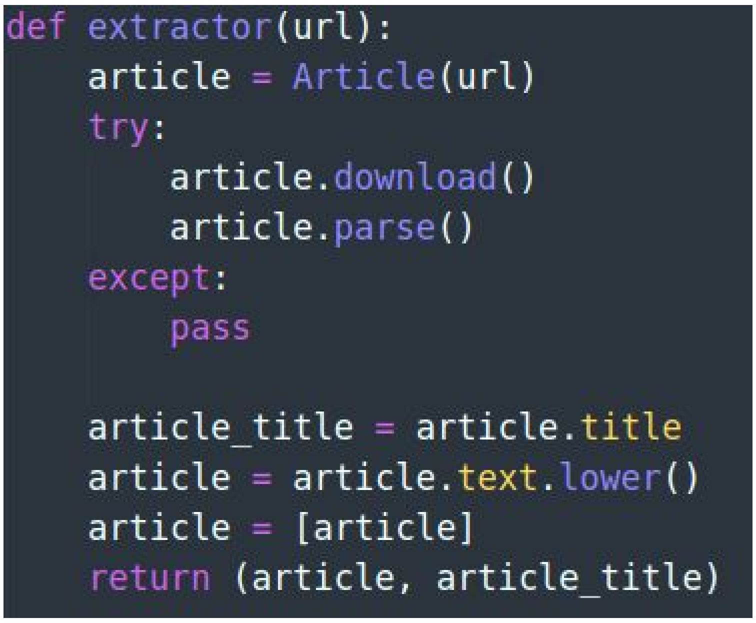
HTML, CSS and Javascript are all languages that I'm familiar with as I've been using them since first year. Initially, I was thinking of using Django as my web framework as I had implemented it for my 3rd year project the year prior. However, I realised that my web application wouldn't really use too many of the features that Django provides. I needed to build a simple web app; no user sign-up or log-in needed as I wanted anyone to be able to use it, and Flask was simple and easy enough to learn. Flask also uses Jinja2 as their template engine for Python which made my life easier.

Back End

Back end of the system is written in **Python**, which resides on the Flask framework. The machine learning aspects were done using **scikit-learn,** and **Pandas** was used to handle the large datasets.

The Extractor

With the nature of my project, I started by figuring out how to extract the articles from different news site. I realised quickly that writing code to extract the article bodies from various news site would be very difficult. The tags that encapsulate the article bodies will vary from site to site, meaning that writing my own extractor would only work in certain places. I searched for multiple web scrapers, and often came across APIs that cost money. Finally, I found and opted to use an API called **Newspaper3k**, which was exactly what I needed. All I needed to do was provide a URL and it downloads and parses the articles, and I'm able to use its many functions like extracting the title (which is later used for the Google search), and of course, the body of the article itself.



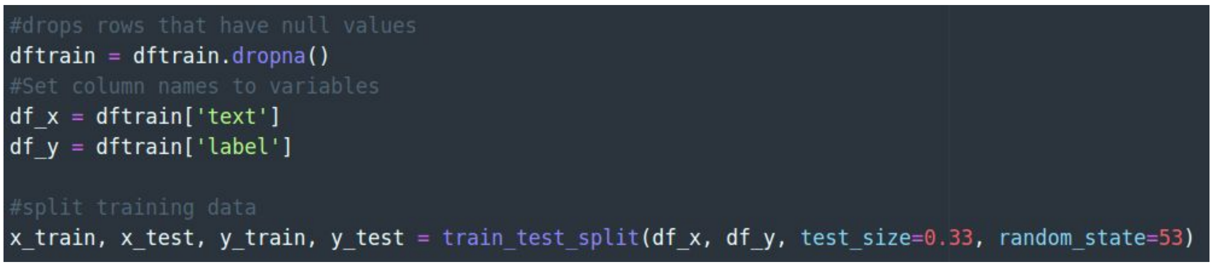
```python
def extractor(url):
    article = Article(url)
    try:
        article.download()
        article.parse()
    except:
        pass

    article_title = article.title
    article = article.text.lower()
    article = [article]
    return (article, article_title)
```

Text Classification

The next step was finding a suitable text classifier that I can train with the dataset I have acquired. After the research that I have conducted, I decided to try out classifiers such as Multinomial Naive Bayes (MultinomialNMB) and Passive Aggressive Classifier (PAC) in sk-learn.

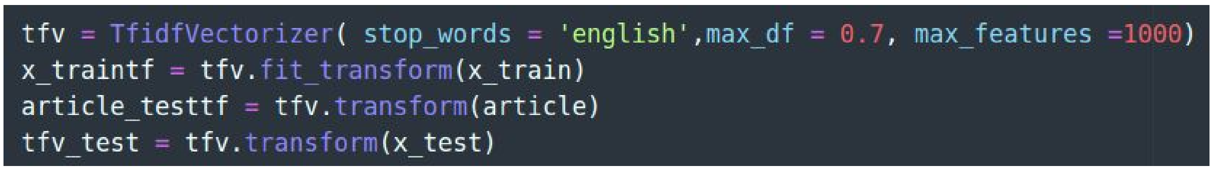
Using Pandas, I did some cleaning by dropping rows that were empty and split the dataset into test and training data respectively.



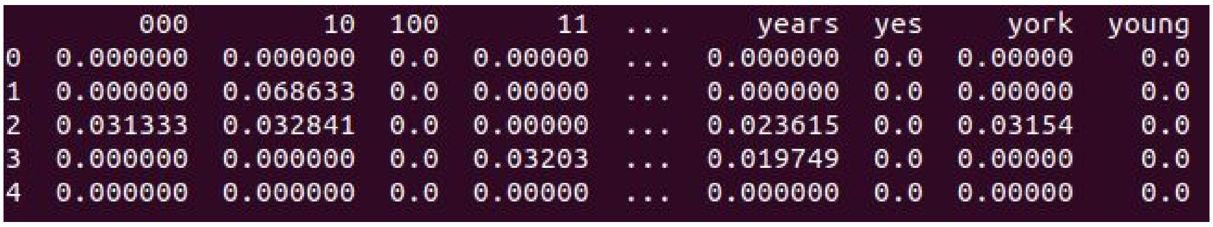
```python
#drops rows that have null values
dftrain = dftrain.dropna()
#Set column names to variables
df_x = dftrain['text']
df_y = dftrain['label']

#split training data
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.33, random_state=53)
```

Using the **TfidfVectorizer** function, I fit_transform/fit the various articles. This converts the articles into a matrix of TF-IDF features.



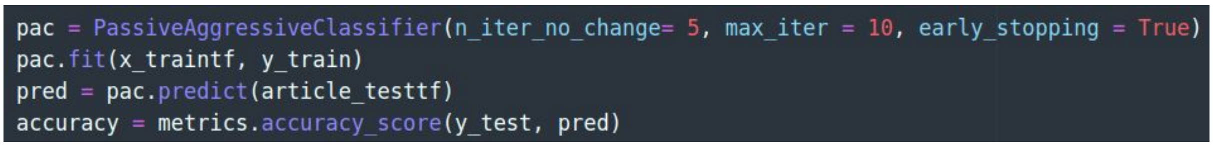
```python
tfv = TfidfVectorizer( stop_words = 'english',max_df = 0.7, max_features =1000)
x_traintf = tfv.fit_transform(x_train)
article_testtf = tfv.transform(article)
tfv_test = tfv.transform(x_test)
```



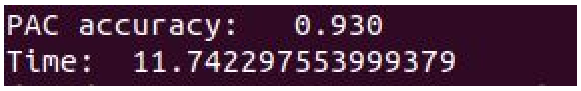
```
        000        10  100       11  ...     years  yes     york  young
0  0.000000  0.000000  0.0  0.00000  ...  0.000000  0.0  0.00000    0.0
1  0.000000  0.068633  0.0  0.00000  ...  0.000000  0.0  0.00000    0.0
2  0.031333  0.032841  0.0  0.00000  ...  0.023615  0.0  0.03154    0.0
3  0.000000  0.000000  0.0  0.03203  ...  0.019749  0.0  0.00000    0.0
4  0.000000  0.000000  0.0  0.00000  ...  0.000000  0.0  0.00000    0.0
```

Term Frequency-Inverse Document Frequency and the corresponding function in sk-learn is used to assign weights to each word and is a measure of how important that word is to a document in a collection. The importance however is offset by the frequency of the word in the corpus, which helps to adjust words that appear more frequently than others. The idea is that since the dataset has been labelled as fake or real, the more often a word appears in that labelled document, then there will be an increase in “importance” towards it and the weights are adjusted accordingly. This means if a certain word appeared multiple times in the fake articles in the dataset, then the higher likelihood that any document to be tested that contains that word will be classified as fake.

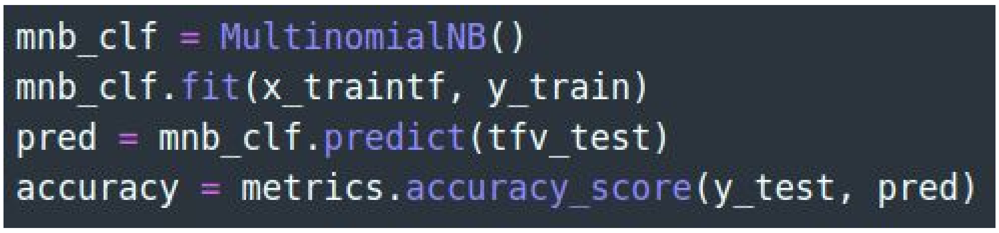
With the dataset prepared for training, I tested both MultinomialNMB and PAC to see see which works best with classifying the articles. MultinomialNMB returned with ~87% accuracy while PAC returned with ~92% accuracy. Seeing this, I opted to use PAC for classifying the articles.



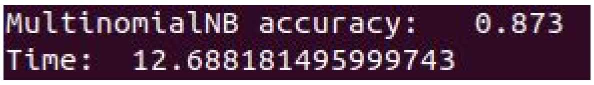
```
pac = PassiveAggressiveClassifier(n_iter_no_change= 5, max_iter = 10, early_stopping = True)
pac.fit(x_traintf, y_train)
pred = pac.predict(article_testtf)
accuracy = metrics.accuracy_score(y_test, pred)
```



```
PAC accuracy:    0.930
Time:  11.742297553999379
```



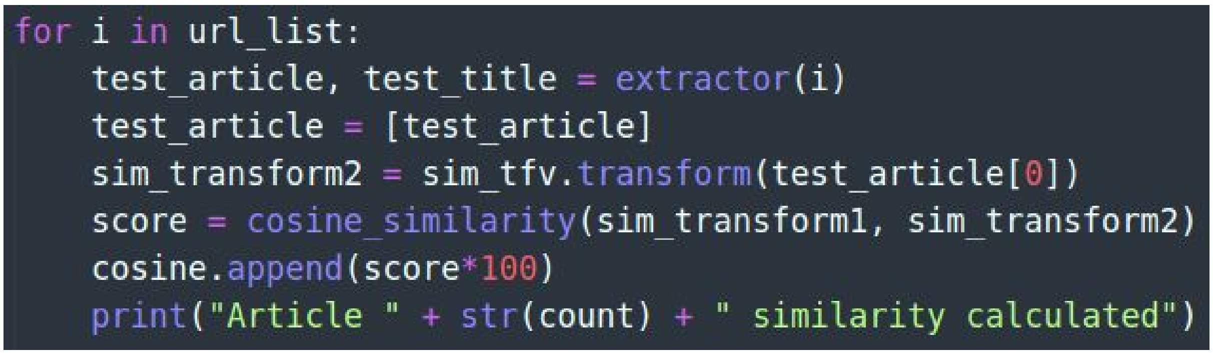
```
mnb_clf = MultinomialNB()
mnb_clf.fit(x_traintf, y_train)
pred = mnb_clf.predict(tfv_test)
accuracy = metrics.accuracy_score(y_test, pred)
```



```
MultinomialNB accuracy:    0.873
Time:  12.688181495999743
```

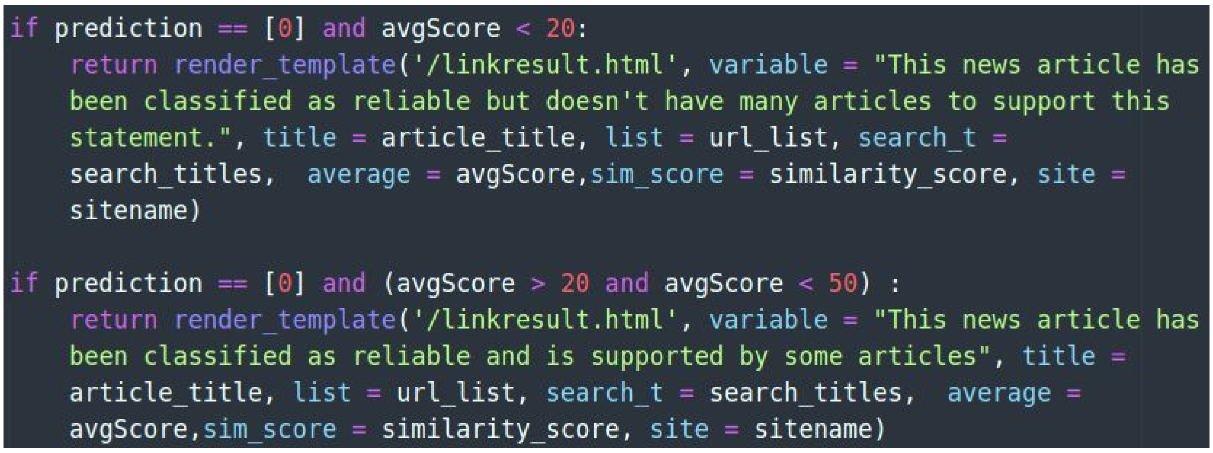
Similarity Score

When I presented the idea to the approval panel early in the semester, I was given the idea to check the similarity of other articles that that may be talking about the same thing as the target article being classified. To achieve this, I needed to perform a Google search using the article’s title and extract the body of the top results from the search result. This was achieved using a googlesearch api and the Newspaper3k api mentioned earlier.



```
for i in url_list:
    test_article, test_title = extractor(i)
    test_article = [test_article]
    sim_transform2 = sim_tfv.transform(test_article[0])
    score = cosine_similarity(sim_transform1, sim_transform2)
    cosine.append(score*100)
    print("Article " + str(count) + " similarity calculated")
```

The contents of these articles are then compared using Cosine Similarity. Cosine Similarity calculates similarity by measuring the cosine of angle between two vectors. TF-IDF is used once again to transform both the target article and the articles pulled from the Google search executed. The similarity is calculated and the average is found between list of pulled articles. I set the value of <20% similarity as “unreliable” with not many supporting articles, >20% but < 50% similarity as “supported by some articles” and anything above 50% similarity as “reliable” and “supported by multiple articles”.



```
if prediction == [0] and avgScore < 20:
    return render_template('/linkresult.html', variable = "This news article has
    been classified as reliable but doesn't have many articles to support this
    statement.", title = article_title, list = url_list, search_t =
    search_titles,  average = avgScore,sim_score = similarity_score, site =
    sitename)

if prediction == [0] and (avgScore > 20 and avgScore < 50) :
    return render_template('/linkresult.html', variable = "This news article has
    been classified as reliable and is supported by some articles", title =
    article_title, list = url_list, search_t = search_titles,  average =
    avgScore,sim_score = similarity_score, site = sitename)
```

Problems Encountered/Solved

100% similarity

One of the many problems I encountered occurred during the calculation of similarity score. Recall that the articles are pulled from Google search using the article title and in a lot of cases, as you would expect, the top result is the target article itself. When the similarity is calculated, it would then obviously return with 100% similarity.

I started with the solution in a for loop of **“if i not in url_list”,** only executing the similarity calculation if it’s a unique entry in the list. However, in some cases, a website showing the same article would have a variance of “ww.bbc...” and “https://www.bbc...”, therefore it still gets added into the list as it is seen as unique.

I overcame this by using “urlparse” function from urllib which lets you get the hostname of the site.

domain = urlparse(target).hostname

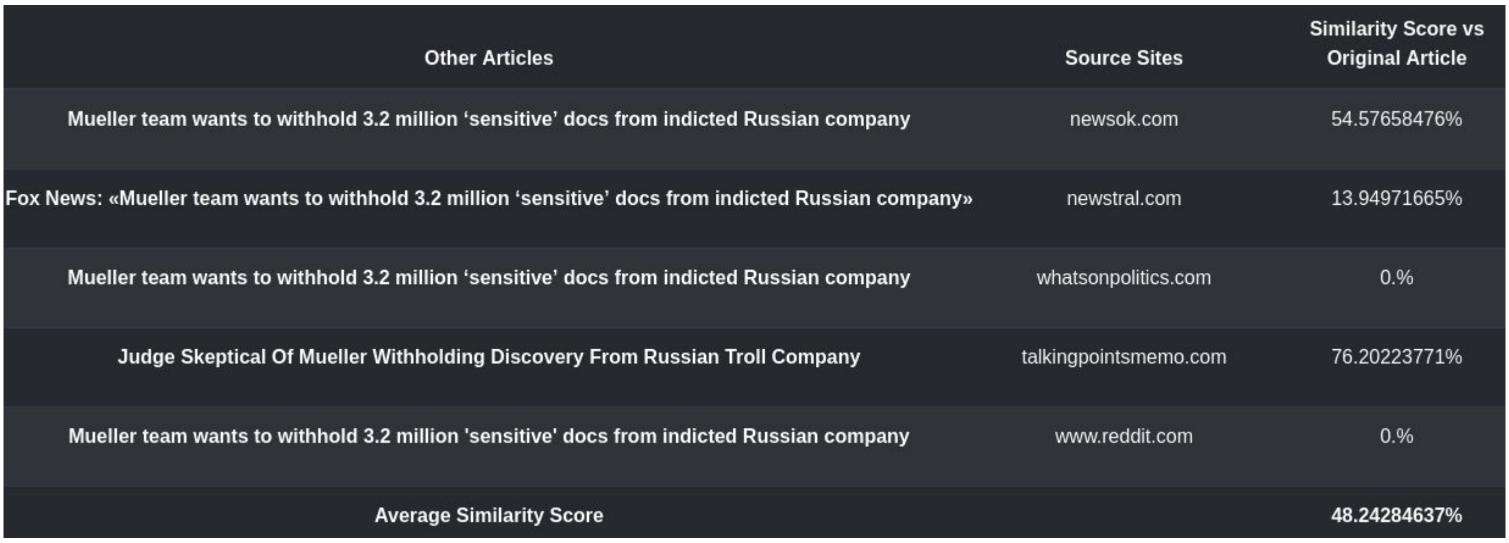
This ensures that the only the domain name is being added into the list and therefore stopped the addition of duplicates and the 100% similarity.

0% Similarity or very low similarity but still relevant

Unfortunately, this problem still occurs in the web app and is mainly caused by a few things;

- The article only consists of embedded media elements i.e. tweets, instagram post
- It’s a video
- The link is in an aggregator site and doesn’t contain the article itself
- There’s a paywall i.e. you have to subscribe to see the article

I was only able to get around the problem with the video. As it happens, most search results that appear in video form come from one domain (Youtube) so all I had to do was include the keyword “youtube” in the for loop I was using.



Other Articles	Source Sites	Similarity Score vs Original Article
Mueller team wants to withhold 3.2 million ‘sensitive’ docs from indicted Russian company	newsok.com	54.57658476%
Fox News: «Mueller team wants to withhold 3.2 million ‘sensitive’ docs from indicted Russian company»	newstral.com	13.94971665%
Mueller team wants to withhold 3.2 million ‘sensitive’ docs from indicted Russian company	whatsonpolitics.com	0.%
Judge Skeptical Of Mueller Withholding Discovery From Russian Troll Company	talkingpointsmemo.com	76.20223771%
Mueller team wants to withhold 3.2 million 'sensitive' docs from indicted Russian company	www.reddit.com	0.%
Average Similarity Score		**48.24284637%**

However, as for the rest of cases in that list, the article scraper that I use does not have the capability to bypass a paywall or detect that it’s an aggregator site. Sites that have embedded media elements may still be relevant to the target article, but the scraper can’t scrape multimedia.

In the image above, both sites where there is 0% similarity score and the one with 13% score are all link aggregator sites that are actually linking the original article itself, hence why it came up in the Google search.

Satire Websites

Satire websites like theOnion and Ireland’s very own WaterfordWhispererNews are one of the biggest challenges in classifying fake news. They are written in a humorous manner that machine learning models find hard to classify. Yes the information in them aren’t real but machine learning models can’t distinguish it from the legitimate article because of the way they are written. This might be solvable using sentiment analysis but I can imagine that it’ll run into the same problem.

Speed

A big problem with this web app is it is quite slow when it comes to calculating the similarity score. The classification of the article is actually very quick, almost instantaneous (can be seen when using the text box classifier). However, it really suffers when it comes to transforming the body of multiple articles pulled from the Google search. This also depends largely on the how big the article is. The more detailed and talked about the subject, the longer it would take to transform the document into vectors.

This is a problem however that I can’t solve as it is to do with the TfidfVectorizer.fit_transform/transform functions within sk-learn.

Results

Overall, the web app works well. It is able to classify real news very often and after trying some satirical news sites, it was able to classify some it as unreliable. The dataset might be a bit more biased towards the political ongoings in America. This could lead to wrongly classifying any legitimate news about Trump for example, as fake, due to the terms associated with him.

The similarity score works pretty well and I think is a good way of self evaluation for the user. If they think the classifier is wrong, they can just look through the list of articles and their scores to see if enough of them can support their judgement or prove them wrong otherwise.

Future Work

In the future, if I do come back to this project, I'll probably implement sentiment analysis and train it to recognise how fake news is written. Maybe even train it to recognise satirical articles as it is quite difficult. I do like the idea of comparing the similarity of other online articles as a sort of validation metric for the classifier so and in the future, more articles being extracted and compared would be great.

In terms of research on the topic of fake news, I think it's important that we should keep developing techniques to identify and help prevent the spread of misinformation. People are quick to assume and believe everything they see on the internet, and with how well connected the world is, it's a forest fire waiting to happen.