Disseminating fake information

Project Goal

Steps

1. Loading the data
2. Retrieving information
3. Pre-processing
4. Feature Importance
5. ML Models
6. Choosing the Classifier

Loading the data

sklearn
+
pandas

libraries initially used

This is where we are retrieving our information from

Hidden

85% - 15% split

Retrieving information

Most important words, hashtags, and mentions

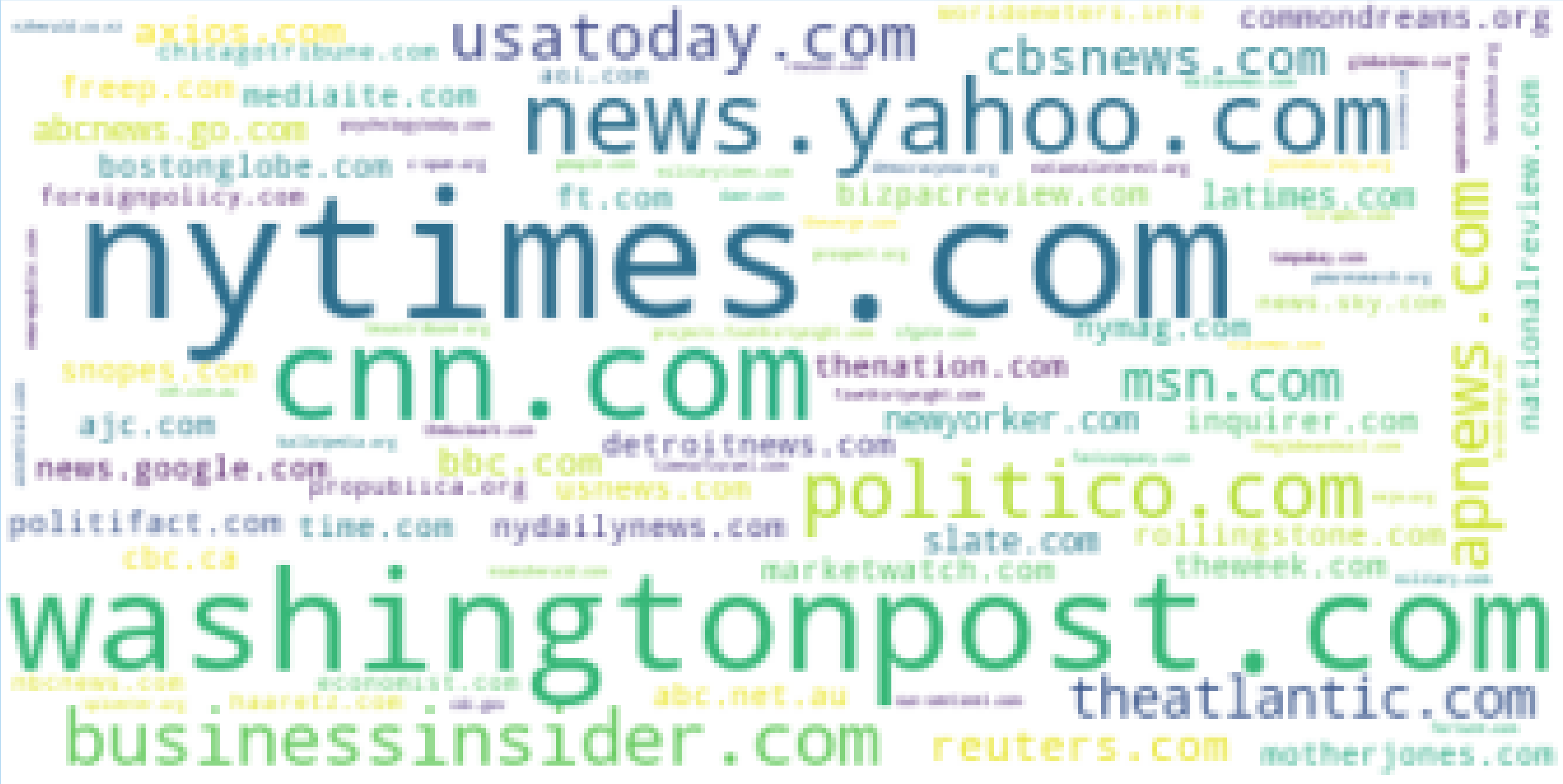
1. Separate the tweets

```
true_tweets = tweets.loc[tweets['questionable_domain'] == 0]
fake_tweets = tweets.loc[tweets['questionable_domain'] == 1]
```

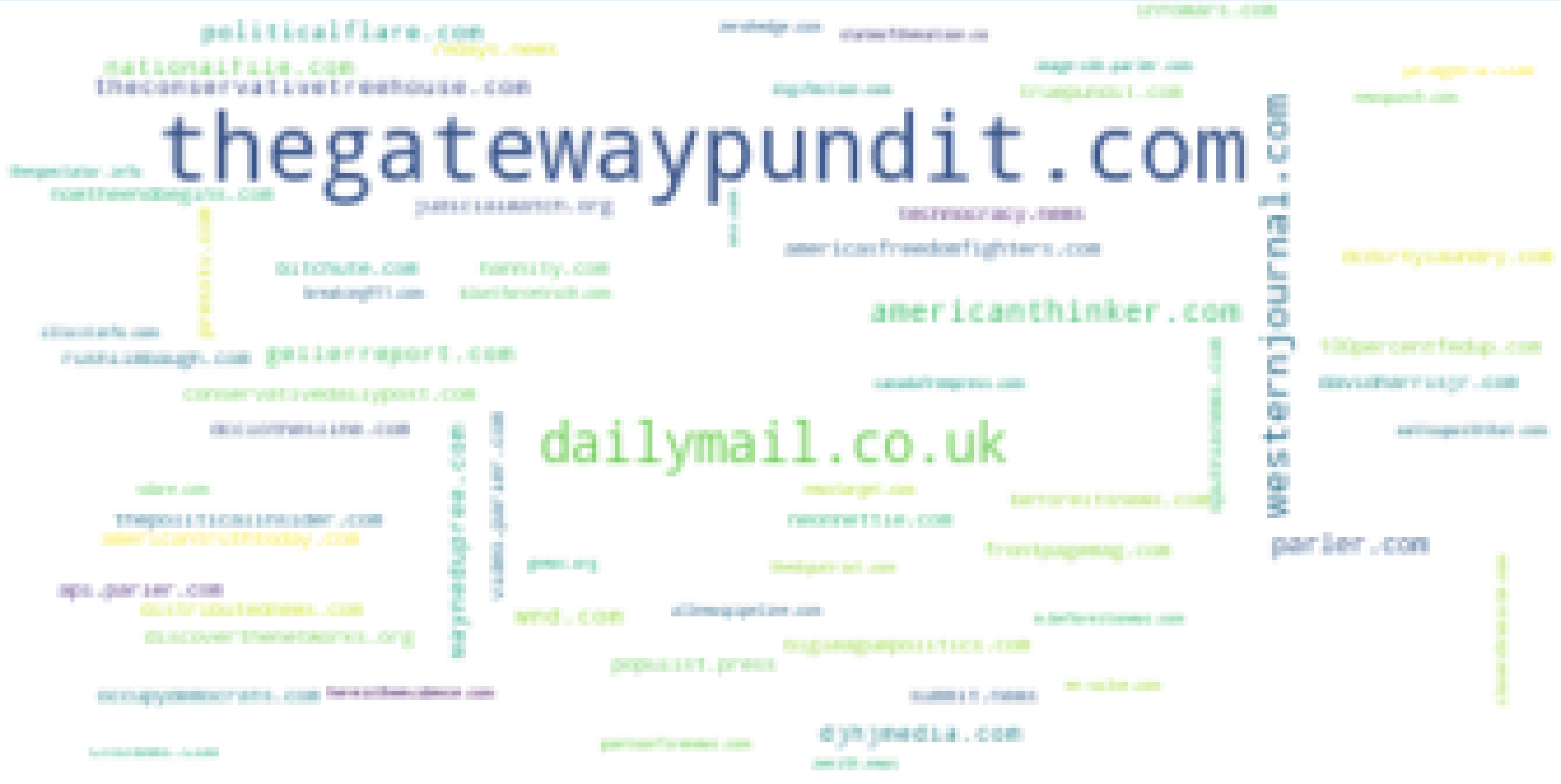
2. Retrieve all the words, hashtags, and mentions from each group
    2.1. Make a top n most used words, hashtags or mentions for each group, based on its frequency among the tweets of each group
3. Retrieve all links from each group
    3.1. For each tweet-like link, convert it to visible domain
    3.2. For each visible domain, retrieve the root domain

Retrieving information

URL Word Clouds



"Good" URLs



"Bad" URLs

Retrieving information

Hashtags Word Clouds

"Good" Hashtags

"Bad" Hashtags

Pre-processing

1 Define auxiliary functions

To process text:

- count words
- count words in CAPS
- retrieve hashtags
- retrieve root domains
- retrieve mentions
- retrieve the most common words
- evaluate emotion levels
- sentiment analysis

libraries used for pre-processing:
- text2emotion
- nltk
- re

Pre-processing

evaluate emotions level

Anger

Fear

Happiness

Sadness

Surprise

Pre-processing

Sentiment Analysis – few steps to get there:

1. Lower case every word
2. Remove punctuation
3. Remove Stop-Words
4. Lemmatization

→ **Score** $\in [-1,1]$

Pre-processing

2 Define a Pre-processing fuction

Using every auxiliary function previously defined we **mounted** some sort of **a pipeline for pre-processing** any data frame that had the same structure.

It goes from **removing unused features** like the identifier and duplicated features in content, to **performing** some **NLP Techniques** in order **to perform** both **Emotion Extraction** and **Sentiment Analysis**.

This is where feature engineering is done: **feature extraction** and **feature selection**!

Feature Importance

We used a classifier named ExtraTreesClssifier from **sklearn** library to perform a feature importance measurement.

We did just like we were to fit a model, but we used the **feature_importances_** method that comes with this classifier.

```
X = data.loc[:, data.columns != 'questionable_domain']

y = data['questionable_domain']

# feature extraction
model = ExtraTreesClassifier(n_estimators=10)
model.fit(X, y)

importance_list = list(model.feature_importances_)
```

Feature Importance

```
a = features_names[1:]
b = importance_list

c = zip(a,b)
c = list(c)

d = sorted(c, key = lambda x: x[1])

to_delete = []
for i in range(len(d)):
  if d[i][1] <= 0.01:
    to_delete.append(d[i][0])

tweets = tweets.drop(to_delete, axis=1)
```

We zipped the features' names and their score of feature importance so we could know which features to remove, in order to perform **dimensionality reduction**.

We, then, **sorted** the list and **filtered** it to have only features that had an importance score **greater than 1%**.

A list named **to_delete** was created to facilitate the drop features step.

ML Models

Naive Bayes

AdaBoost

Random Forrest

Voting Classifier

ML Models

Before fitting the models:

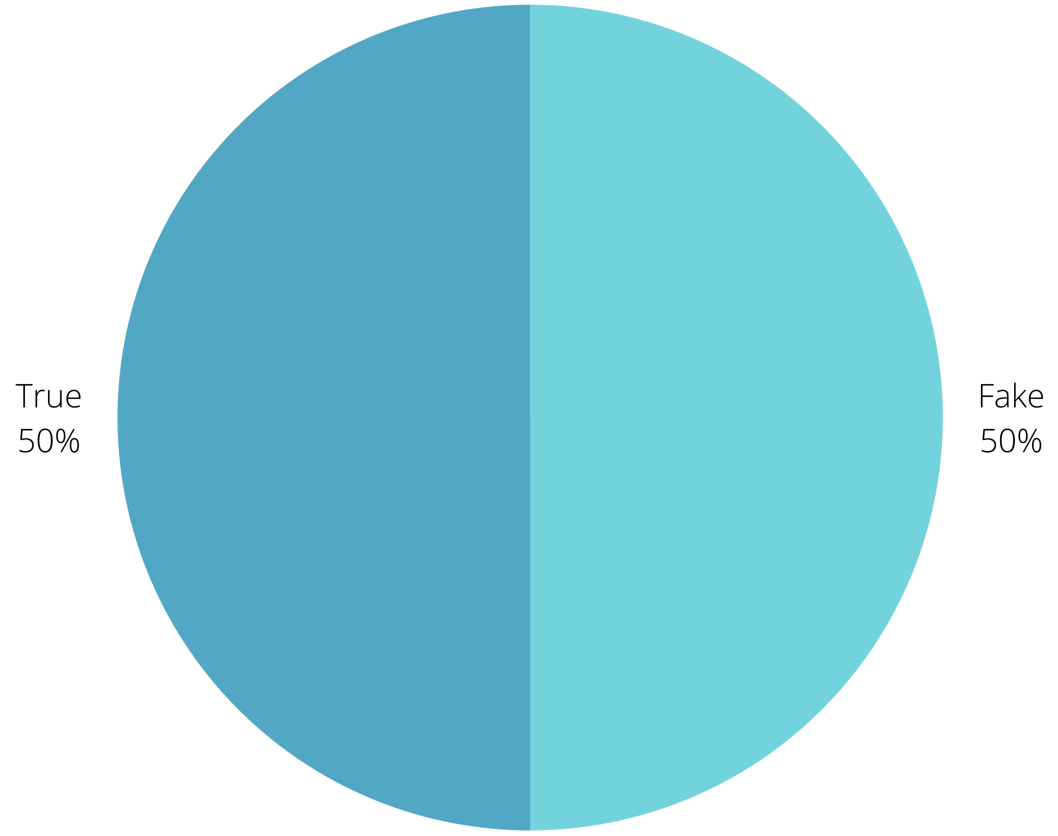
Imbalanced Data

```
#check out how many fake and true cases we have:
print(tweets['questionable_domain'].value_counts())
```

```
0     12720
1      2537
```

random under-sampling → majority class

random over-sampling → minority class



ML Models

Evaluating the models:

1 X 10 CROSS VALIDATION

Naive Bayes

```
accuracy: 0.6207621550591327
precision: 0.6069025654781756
recall: 0.7543721508495648
f1: 0.6611756459412084
```

AdaBoost

```
accuracy: 0.8169513797634691
precision: 0.8730174228175306
recall: 0.7419242989363172
f1: 0.8020081100018412
```

Random Forrest

```
accuracy: 0.9195795006570302
precision: 0.951277281667301
recall: 0.8856858682138418
f1: 0.9166526091862528
```

Voting Classifier

```
accuracy: 0.8629434954007884
precision: 0.8943853852335609
recall: 0.8263157894736841
f1: 0.8595269458416139
```

Choosing the Classifier

The classifier with the best performance, according to the chosen evaluation method is **Random Forrest**.

Now, it's time to fit/train our chosen model and use it in our **hidden** data frame (the 15% split we did at the beginning).

This step is done with an already balanced train data frame, so our model can recognize as many true cases as fake cases.

Choosing the Classifier

One other "small" detail that we cannot forget about it:

The **hidden** data frame **is not** yet **pre-processed** at this stage, and this is why it was so convenient to create a pre-processing function.

At this point, we just need to pre-process it and we're all set to go!

```
hidden_after_preProcessing = preProcessing(hidden)
```

Choosing the Classifier

After the pre-processing on hidden is done, we separate the target variable from the rest, and we let our model predict the **unseen** tweets.

The results are as follow:

```
RandomForestClassifier() accuracy: 0.9220200519866321
RandomForestClassifier() precision: 0.7962529274004684
RandomForestClassifier() recall: 0.734341252699784
RandomForestClassifier() f1: 0.7640449438202248
```