preprocessing

March 15, 2025

1 Task 1

1.1 Import libraries and load data

[3]:
```
# Import libraries
import pandas as pd
import numpy as np
import re
from cleantext import clean
import matplotlib.pyplot as plt
import nltk
#nltk.download('all')
from nltk.probability import FreqDist
import ast
```

[4]:
```
# Load data as data frame
corpusSample = pd.read_csv("250_rows.csv")
```

1.2 Clean content variable

[5]:
```
# The function clean_text() does this:
#   - all words will be lowercased
#   - tabs, new lines and multiple white spaces will be set to single white␣
 ↪space
#   - numbers, dates, emails, and URLs will be replaced by "\<NUM>", "\<DATE>",␣
 ↪"\<EMAIL>" AND "\<URL>", respectively.
def clean_text(data):

  # Set dates with format DD/MM/YYYY or MM/DD/YYYY to "<date>"
  data = re.sub("[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}", "<date>", data)

  # Set dates with the format DD/MM/YY or MM/DD/YY to "<date>"
  data = re.sub("[0-9]{1,2}/[0-9]{1,2}/[0-9]{2}", "<date>", data)

  # Set dates with the format DD/MM/YYYY or MM/DD/YYYY to "<date>"
  data = re.sub("[0-9]{1,2}-[0-9]{1,2}-[0-9]{4}", "<date>", data)

  # Set dates with the format DD/MM/YY or MM/DD/YY to "<date>"
```

```
    data = re.sub("[0-9]{1,2}-[0-9]{1,2}-[0-9]{2}", "<date>", data)

    # Consider adding other date formats, like "Sept 6", "September 6, 2019", etc.

    # Use clean() for remaining cleaning
    cleaned = clean(data,
        fix_unicode=False,          # fix various unicode errors
        to_ascii=False,             # transliterate to closest ASCII␣
↪representation
        lower=True,                 # lowercase text
        no_line_breaks=True,        # fully strip line breaks as opposed to only␣
↪normalizing them
        no_urls=True,               # replace all URLs with a special token
        no_emails=True,             # replace all email addresses with a special␣
↪token
        no_phone_numbers=False,     # replace all phone numbers with a special␣
↪token
        no_numbers=True,            # replace all numbers with a special token
        no_digits=False,            # replace all digits with a special token
        no_currency_symbols=False,  # replace all currency symbols with a␣
↪special token
        no_punct=False,             # remove punctuations
        replace_with_punct="",      # instead of removing punctuations you may␣
↪replace them
        replace_with_url="<URL>",
        replace_with_email="<EMAIL>",
        replace_with_phone_number="<PHONE>",
        replace_with_number="<NUM>",
        replace_with_digit="0",
        replace_with_currency_symbol="<CUR>",
        lang="en"                    # set to 'de' for German special handling
    )
    return cleaned
```

[6]:
```
content_sample_cleaned = corpusSample['content'].apply(clean_text)
```

1.3 Tokenize content variable

[7]:
```
# Import libraries
from nltk.tokenize import word_tokenize
from nltk.tokenize import MWETokenizer

# This is to make sure that "<num>", "<date>", "<email>" and "<url>" are
# single tokens - and not "<", "num" and ">" etc.
multiWordsTokenizer = MWETokenizer([('<', 'num', '>'), ('<', 'date', '>'),␣
↪('<', 'email', '>'), ('<', 'url', '>')], separator='')
```

[8]:
```
# Function that tokenize a string
def tokenize(data_string):

    # Word tokenize
    data_string = word_tokenize(data_string)

    # MAKE '<', 'NUM' and '>' into '<NUM>'. Same for <DATE>, <EMAIL> and <URL>:
    data_string = multiWordsTokenizer.tokenize(data_string)

    return data_string
```

[9]:
```
content_sample_tokenized = content_sample_cleaned.apply(tokenize)
```

1.4 Remove stop words

[10]:
```
# Import libraries
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

[11]:
```
# Function that removes stop words from a string
def removeStopWords(words):
    filteredWords = []

    for w in words:
        if w not in stop_words:
            filteredWords.append(w)
    return(filteredWords)
```

[12]:
```
# Remove stop words
content_sample_no_stop_words = content_sample_tokenized.apply(removeStopWords)
```

1.5 Perform stemming on content variable

[13]:
```
# Import libraries
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
```

[14]:
```
# Function that performs stemming on a string
def stemming(words):

    stemmedWords = []
    for w in words:
        stemmedWords.append(stemmer.stem(w))

    return(stemmedWords)
```

[15]:
```
content_sample_stemmed = content_sample_no_stop_words.apply(stemming)
```

1.6 Reduction rates

[16]:

```python
# Using FreqDist() we can see the vocabulary as well as the frequence of each␣
↪token
tokens_after_tokenization = [x.strip("'") for l in content_sample_tokenized for␣
↪x in l]
tokens_after_tokenization_vocab = FreqDist(tokens_after_tokenization)

tokens_after_removing_stop_words = [x.strip("'") for l in␣
↪content_sample_no_stop_words for x in l]
tokens_after_removing_stop_words_vocab =␣
↪FreqDist(tokens_after_removing_stop_words)

tokens_after_stemming = [x.strip("'") for l in content_sample_stemmed for x in␣
↪l]
tokens_after_stemming_vocab = FreqDist(tokens_after_stemming)

print(f"Size of vocabulary after tokenization:␣
↪{len(tokens_after_tokenization_vocab)}\n")

print(f"Size of vocabulary after removal of stop words:␣
↪{len(tokens_after_removing_stop_words_vocab)}\n")

print(f"Size of vocabulary after stemming:␣
↪{len(tokens_after_stemming_vocab)}\n")

print(f"Reduction rate of the vocabulary size after removing stopwords:␣
↪{(len(tokens_after_tokenization_vocab)
                                    -␣
↪len(tokens_after_removing_stop_words_vocab)) /
                                    len(tokens_after_tokenization_vocab) *␣
↪100}\n")

print(f"Reduction rate of the vocabulary size after stemming:␣
↪{(len(tokens_after_removing_stop_words_vocab)
                                    - len(tokens_after_stemming_vocab)) /
                                    len(tokens_after_removing_stop_words_vocab)␣
↪* 100}\n")
```

```
Size of vocabulary after tokenization: 16887

Size of vocabulary after removal of stop words: 16752

Size of vocabulary after stemming: 11590

Reduction rate of the vocabulary size after removing stopwords:
0.7994315153668502
```

```
Reduction rate of the vocabulary size after stemming: 30.814231136580705
```

[17]:
```python
# Add preprocessed 'corpus' variable to corpus sample:
corpusSamplePreprocessed = corpusSample
corpusSamplePreprocessed['content'] = content_sample_stemmed
```

2 Task 2

2.0.1 Initial clean up: Remove non-relevant features and rows with invalid values - and remove row duplicates

[18]:
```python
# Load data as data frame. Load either full corpus or sample with 10,000 rows.

corpus = pd.read_csv("995,000_rows.csv")
#corpus = pd.read_csv("10,000_rows.csv")
```

```
C:\Users\Krist\AppData\Local\Temp\ipykernel_13112\4127333678.py:3: DtypeWarning:
Columns (0,1) have mixed types. Specify dtype option on import or set
low_memory=False.
  corpus = pd.read_csv("995,000_rows.csv")
```

[19]:
```python
# Remove non-relevant features
corpus = corpus[['domain','type', 'content', 'title', 'authors',␣
↪'meta_description']]
```

[20]:
```python
# Remove rows with invalid values
corpus = corpus.drop(corpus[corpus['type'] == '2018-02-10 13:43:39.521661'].
↪index)
```

[21]:
```python
# Remove data points where either 'type' or 'content' is NaN
corpus = corpus[corpus['type'].notna() & corpus['content'].notna()]
```

[22]:
```python
# Remove duplicates - there are 66.232 duplicated rows in full dataset.
corpus = corpus.drop_duplicates()
```

2.0.2 Preprocessing: Clean, tokenize, remove stop words and perform stemming

[23]:
```python
# Clean 'content' and save data frame as .csv file
corpus['content'] = corpus['content'].apply(clean_text)
corpus.to_csv('corpus_cleaned.csv', index=False)
```

[24]:
```python
# Tokenize 'content' and save data frame as .csv file
corpus['content'] = corpus['content'].apply(tokenize)
corpus.to_csv('corpus_tokenized.csv', index=False)
```

[25]:
```python
# Remove stop words from 'content' and save data frame as .csv file
corpus['content'] = corpus['content'].apply(removeStopWords)
corpus.to_csv('corpus_no_stop_words.csv', index=False)
```

[26]:
```python
# Perform stemming on 'content' and save data frame as .csv file
corpus['content'] = corpus['content'].apply(stemming)
corpus.to_csv('995,000_rows_preprocessed.csv', index=False)
```

3 Task 3 - the three questions

1. Data frames has labeled axes, as opposed to for instance numpy arrays. And it is possible to get a nice spreadsheet representation of the data set.
2. - `Authors` variable has 44% missing values and `meta_description` has 53% - making it hard to use them in a model.
   - The type value `2018-02-10 13:43:39.521661` only has one news article and it looks like the article has been mislabeled (the name is weird and all other domains has at least 8779 articles). Should be removed.
   - 'type' has 47786 missing values and 'content' has 12 missing values. The data points (rows) where either of these two values are missing should be removed.
3. Include for instance: Number of features and data points. Number of missing values for each feature. Number of distinct values for relevant features (and what the categorical values are). Data type of each feature.

4 Task 3 - non-trivial observation

In this section we will see that…

- All texts from a particular domain is of the same type,
- Few domains accounts for a majority of the total number of articles.
- The distribution of article types are very uneven. For instance: There are 25 times as many articles of type 'reliable' than type 'hate',
- The average length of articles (the token count) for each article type varies greatly. 'hate' articles has the highest average token count and 'satire' has the lowest.
- There is a large variation in number of tokens in 'content' in each article. For instance: The longest article has 21730 tokens, and the shortest has just two tokens. However: only very few articles has a very high amount of tokens.

[ ]:
```python
# Load preprocessed corpus (if you want to skip the preprocessing steps above).
# Load either full preprocessed dataset or sample with 15,000 rows

#corpus = pd.read_csv("15,000_rows_preprocessed.csv")
#corpus = pd.read_csv("995,000_rows_preprocessed.csv")
```

[ ]:
```python
# After loading the .csv file as dataframe remember to convert the data type of
# 'content' from string back to list:
```

```
corpus['content'] = corpus['content'].apply(ast.literal_eval)
```

4.0.1 Looking into 'domain'

[28]:
```
print(f"Number of unique values in 'domain': {len(set(corpus['domain']))}")
```

```
Number of unique values in 'domain': 618
```

[29]:
```
# Here we see that all texts from a particular domain is of the same type
boolian_value = True
for x in set(corpus['domain']):
    df_subset = corpus[corpus["domain"] == x]
    if(len(set(df_subset['type']))) != 1:
        print(x)
        boolian_value = False

if boolian_value:
    print("All texts from a particular domain is of the same type!")
```

```
All texts from a particular domain is of the same type!
```

[30]:
```
# Here is the number of news texts for each domain
counts_domain = corpus['domain'].value_counts()
counts_domain_df = counts_domain.rename_axis('unique_values').
 ↪reset_index(name='counts')

# The domains with the most articles
print(f"The 10 domains with the most articles:\n{counts_domain_df[0:10]}\n")

# The domains with the fewest articles
print(f"The 10 domains with the fewest articles:\n{counts_domain_df[-10:]}")
```

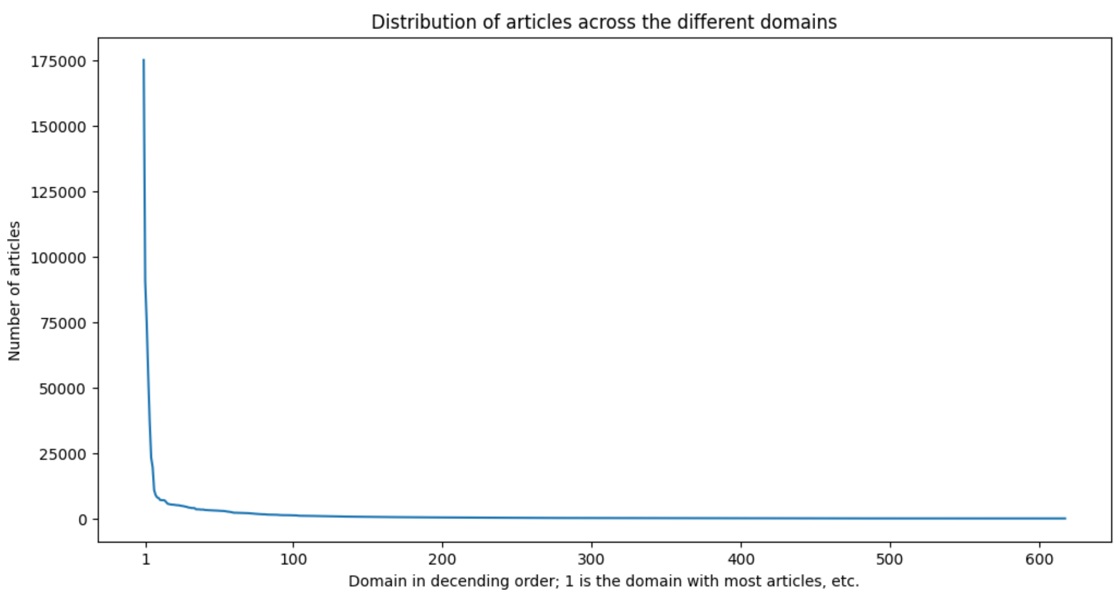
```
The 10 domains with the most articles:
            unique_values  counts
0             nytimes.com  175276
1        beforeitsnews.com   90842
2              dailykos.com   75018
3            express.co.uk   54927
4          sputniknews.com   37143
5             wikileaks.org   23249
6       abovetopsecret.com   19556
7                 pravda.ru   10802
8             lifezette.com    8918
9  investmentwatchblog.com    8039

The 10 domains with the fewest articles:
              unique_values  counts
608          dailynews10.com       1
```

```
609              channel18news.com       1
610              ushealthylife.com       1
611                   goneleft.com       1
612                     uspoln.com       1
613                 madpatriots.com       1
614           dailypoliticsusa.com       1
615        livefreelivenatural.com       1
616  silentmajoritypatriots.com       1
617                 newsmagazine.com       1
```

[31]:

```
# Plot of the distribution of articles across the different domains
plt.figure(figsize=(12,6))
plt.xlabel('Domain in decending order; 1 is the domain with most articles, etc.
↪')
plt.ylabel('Number of articles')
plt.title('Distribution of articles across the different domains')
plt.xticks([1,100,200, 300, 400, 500, 600])
plt.plot(counts_domain_df.index, counts_domain_df['counts'])
plt.show()
```



4.0.2 Looking into 'type'

[32]:

```
print(f"Number of unique values in 'type': {len(set(corpus['type']))}\n")

# Here is the number of news texts for each domain
counts_type = corpus['type'].value_counts()
```

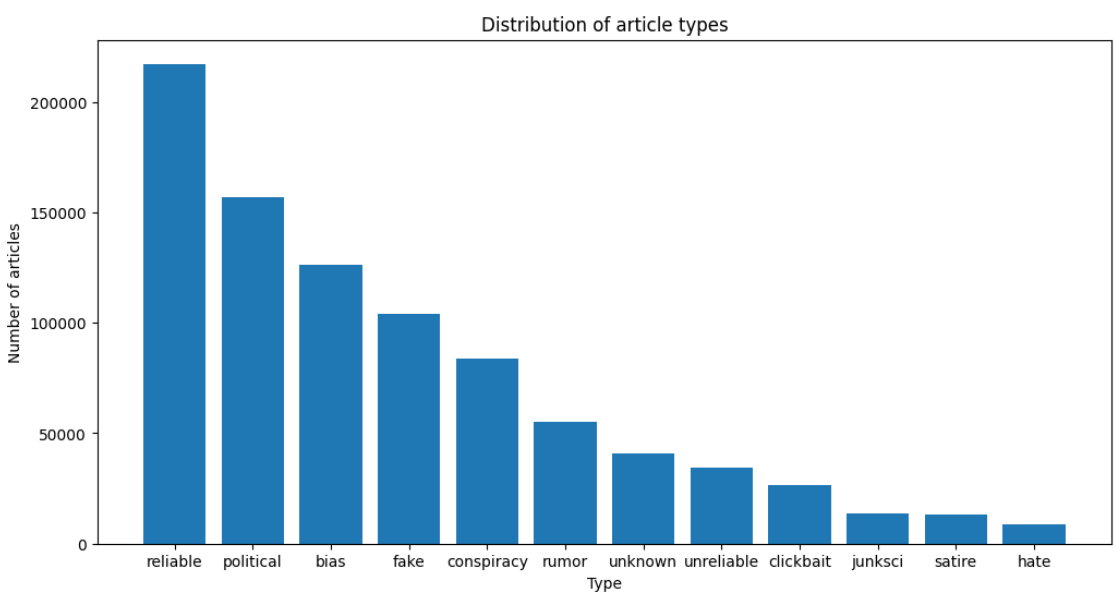
```
counts_type_df = counts_type.rename_axis('unique_values').
  ↪reset_index(name='counts')

print(f"Distribution of articles type:\n{counts_type_df}")
```

```
Number of unique values in 'type': 12

Distribution of articles type:
   unique_values  counts
0       reliable  217268
1      political  156862
2           bias  126260
3           fake  104052
4     conspiracy   83688
5          rumor   55389
6        unknown   40749
7     unreliable   34667
8      clickbait   26761
9        junksci   13492
10        satire   13072
11          hate    8720
```

[33]:
```
# Plot of the distribution of articles across the different types
plt.figure(figsize=(12,6))
plt.title('Distribution of article types')
plt.xlabel('Type')
plt.ylabel('Number of articles')
plt.bar(counts_type_df['unique_values'], counts_type_df['counts'])
plt.show()
```



4.0.3 Looking into 'content'

[34]:
```
# Calculate lengths of 'content' strings
corpus['content_length'] = corpus.content.str.len()
```

[ ]:
```
content_length = corpus['content_length'].sort_values(ascending=False)
print(f"The articles with most tokens has {content_length.iloc[0]} tokens.")
print(f"The articles with fewest tokens has {content_length.iloc[-1]} tokens.")
```

```
The articles with most tokens has 21730 tokens.
The articles with fewest tokens has 2 tokens.
```

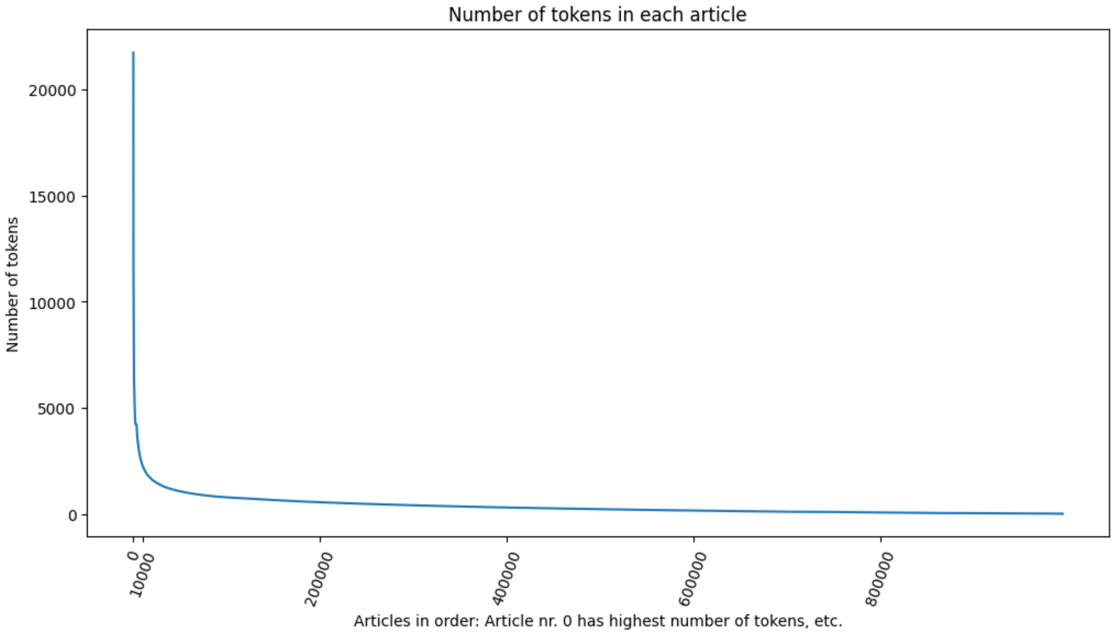
[56]:
```
print(f"Only 5,000 articles (or {round(5000/corpus.shape[0] * 100, 2)}% of the␣
 ↪articles) has more than {content_length.iloc[5000]} tokens.")
```

```
Only 5,000 articles (or 0.57% of the articles) has more than 3137 tokens.
```

[ ]:
```
print(f"Average tokens per article is {round(content_length.mean(), 2)}")
```

```
Average tokens per article is 361.39.
```

[35]:
```
# Plot of the number tokens in each article
plt.figure(figsize=(12,6))
plt.title('Number of tokens in each article')
plt.xlabel('Articles in order: Article nr. 0 has highest number of tokens, etc.
 ↪')
plt.ylabel('Number of tokens')
plt.xticks([0,10000, 200000,400000, 600000, 800000])
plt.xticks(rotation=70)
plt.plot(corpus.index, corpus['content_length'].sort_values(ascending=False))
plt.show()
```

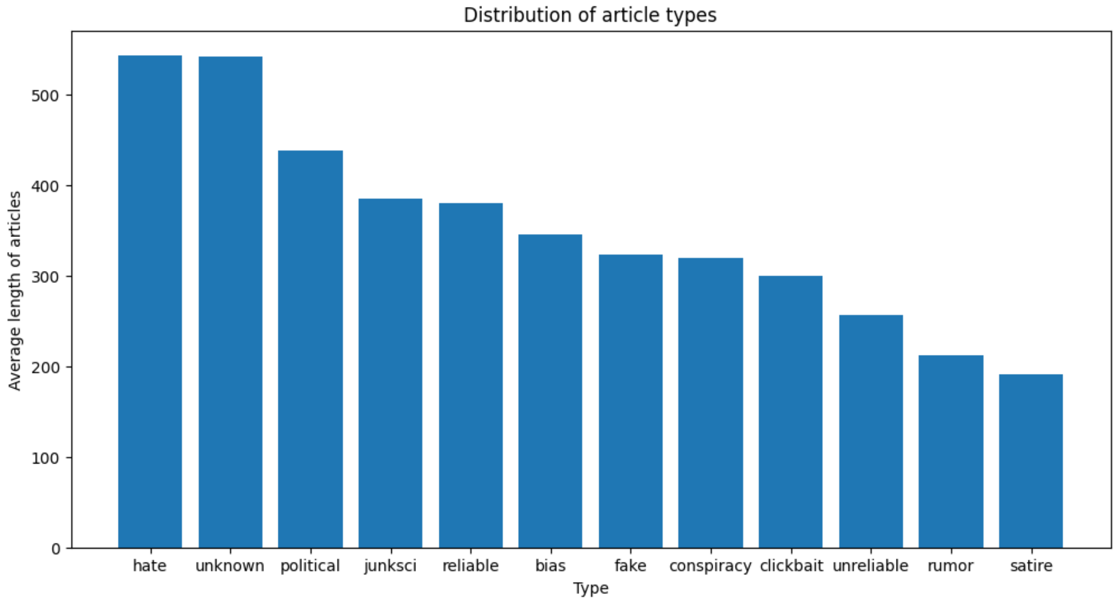


[36]:
```
# Find average length of articles (token count) for each article type.
types = set(corpus['type'])
types_df = pd.DataFrame(types)
types_df.columns = ['type']
types_df["average_token_count"] = np.nan


for index, row in types_df.iterrows():
    df_temp = corpus.loc[corpus['type'] == row['type']]
    types_df.at[index, 'average_token_count'] = sum(df_temp['content_length'])/
 ↪df_temp.shape[0]

types_df = types_df.sort_values("average_token_count", ascending=False)
```

[37]:
```
# Plot the average length of articles (the token count) for each article type.
plt.figure(figsize=(12,6))
plt.title('Distribution of article types')
plt.xlabel('Type')
plt.ylabel('Average length of articles')
plt.bar(types_df['type'], types_df['average_token_count'])
plt.show()
```



5 Task 4

[ ]:
```
train, valid, test = np.split(corpus.sample(frac=1, random_state=42), [int(0.
↪8*len(corpus)), int(0.9*len(corpus))])
```