

Text Analytics

March 5, 2020

0.1 Introduction

The purpose of this assignment was to scrape BusinessInsider articles to extract CEO names, company names and all numbers involving percentages.

0.2 Importing the packages

```
[4]: #Importing packages
import re
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize, RegexpTokenizer
from nltk import pos_tag
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import pandas as pd
import numpy as np
from nltk.tokenize.punkt import PunktSentenceTokenizer, PunktParameters
from tqdm import tqdm
import spacy
import en_core_web_sm
import os
from word2number import w2n
from fractions import Fraction
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

[5]: #Downloads
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]      /Users/ArshyaSrinivas/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
[nltk_data] Downloading package punkt to
[nltk_data]      /Users/ArshyaSrinivas/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]      /Users/ArshyaSrinivas/nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data]      date!
[nltk_data] Downloading package wordnet to
[nltk_data]      /Users/ArshyaSrinivas/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

[5]: True

```
[7]: #Importing training datasets
ceo_train = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/Junior_
→year/WINTER 2019/IEMS 308/text_analytics/ceo.csv", encoding="latin-1",
→header = None)
company_train = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/
→Junior year/WINTER 2019/IEMS 308/text_analytics/companies.csv",
→encoding="latin-1", header = None)
percent_train = pd.read_csv("/Users/ArshyaSrinivas/Google Drive/Engineering/
→Junior year/WINTER 2019/IEMS 308/text_analytics/percentage.csv",
→encoding="latin-1", header = None)
```

The training data needs to be cleaned in the following way: 1. Remove duplicates in the CEO training data 2. Remove duplicates in the company training data 3. Clean the percentage data such that text is converted to numbers

```
[8]: #Removing na and duplicates for CEO training data

ceo_train = ceo_train[ceo_train.iloc[:,0].notna()]
ceo_train = ceo_train[ceo_train.iloc[:,1].notna()]
ceo_concat = []

for row in range(0,len(ceo_train)-1):
    name = ceo_train.iloc[row,0] + " " + ceo_train.iloc[row,1]
    if name not in ceo_concat:
        ceo_concat.append(name)
```

```
[9]: #Removing na and duplicates for CEO training data

company_train = company_train[company_train.iloc[:,0].notna()]
company_concat = []

for row in range(0,len(company_train)-1):
    name = company_train.iloc[row,0]
    if name not in company_concat:
        company_concat.append(name)
```

To clean the percent data, I used the word2number module.

```

[234]: %%capture
# Cleaning the percentage data
percent_update = []

for row in range(0, len(percent_train)-1):
    value = str(percent_train.iloc[row,0]) #Convert the value to a string for
    ↪easier cleaning

    if "to" in value:
        continue

    if "-" in value:
        continue

    if "," in value : # If there is no percent
        continue

    if "\" in value:
        continue

    if "/" in value:
        try:
            if "percent" in value:
                value = value.replace(" percent", "")
                value = str(value) + "%"
                percent_update.append(value)
                continue

            else:
                value = float(Fraction(value)) * 100
                value = str(value) + "%"
                percent_update.append(value)
                continue

        except:
            continue

    if "half" in value:
        value = "0.5%"
        percent_update.append(value)
        continue

    if "quarter" in value:
        value = "0.25%"
        percent_update.append(value)
        continue

```

```

if value == "half a percent":
    value = "0.5%"
    percent_update.append(value)
    continue

if "." in value and "%" not in value:
    if "percentage" in value:
        value = value.replace(" percentage", "")
        value = str(value) + "%"
        percent_update.append(value)
        continue

    elif "percent" in value:
        value = value.replace(" percent", "")
        value = str(value) + "%"
        percent_update.append(value)
        continue

    elif float(value) >= 0.001:
        value = str(value) + "%"
        percent_update.append(value)
        continue

if "percentage" in value:
    value = value.replace(" percentage", "")
    value = str(value) + "%"
    percent_update.append(value)
    continue

if "percent" in value: #If there is a percent in the string
    try:
        value = value.replace(" percent", "") #Remove the percent and the
→ space behind it
        value = w2n.word_to_num(value) #Convert to number using word2num
        value = str(value) + "%" #Convert to string using str
        percent_update.append(value)
        continue
    except:
        continue

if "%" not in value: # If there is no percent

    try:
        if float(w2n.word_to_num(value)) < 100:
            value = str(value) + "%" #Add a percentage sign at the end
            percent_update.append(value)

```

```

        continue
    except:
        continue
    #value = int(value) * 100 #Multiply by 100

    else:
        try:
            value = w2n.word_to_num(value) #Convert to number using word2num
            value = str(value) + "%" #Convert to string using str
            percent_update.append(value)
            continue
        except:
            continue

```

0.3 Analyzing the text

The process for analyzing the text are as follows: 1. Obtain the raw text 2. Sentence segmentation 3. Tokenization 4. Remove stop words 5. Normalization/Lemmatization/Stemming 6. NER algorithm

All of this needs to be conducted in a large for loop. However, some preparation is required for these steps to occur.

Step 1: Obtaining the raw text

```

[11]: path = "/Users/ArshyaSrinivas/Google Drive/Engineering/Junior year/WINTER 2019/
        ↳ IEMS 308/text_analytics/2013/2013-01-02.txt"

new_path = "/Users/ArshyaSrinivas/Google Drive/Engineering/Junior year/WINTER_
        ↳ 2019/IEMS 308/text_analytics/data"

```

Step 2: Sentence segmentation

For sentence segmentation, I will be using the function `sent_tokenize` in the larger loop.

Step 3: Tokenization

To implement the `word_tokenize` command on `regex`, the following function was created. With this function, a loop can eventually be created to tokenize each sentence in the array with each sentence.

```

[12]: def word_tokenize_regex(text):
        return re.findall(r'\w+|[\;\.\!\?\:\]\|\'\w+',text)

```

Step 4: Removing stop words

A set of stop words can be compiled as seen below.

```

[13]: stop_words=sorted(set(stopwords.words("english")))

```

Step 5: POS tagging and lemmatization

The following function was created to ensure that both lemmatization and pos tagging is done before further analysis

```

[14]: lemmatizer = WordNetLemmatizer()

```

```

[15]: def lemmatize_pos(lemmatizer,tokens):
        out = []

```

```

for token,tag in pos_tag(tokens):
    if tag[0] == 'N':
        # noun
        tmp = lemmatizer.lemmatize(token,"n")
    elif tag[0] == "V":
        # verb
        tmp = lemmatizer.lemmatize(token,"v")
    elif tag[0] == "J":
        # adjective
        tmp = lemmatizer.lemmatize(token,"a")
    elif tag[0] == "R":
        # adverb
        tmp = lemmatizer.lemmatize(token,"r")
    else:
        tmp = token

    out.append(tmp)

return out

```

Step 6: NER

The NER algorithm that I will be using is the spaCy algorithm

```
[16]: nlp = en_core_web_sm.load()
```

0.4 Creating the arrays with names using a for loop

Within the for loop, I did two main things. The first thing I did was add all entity texts into the “train” arrays using the spaCy algorithm. The second thing I did was add entity texts into the “feature” arrays if a subset of features were present in the same sentence the entity was. I deemed the following features useful for CEO and company:

CEO: 1. partner 2. executive 3. founder 4. chief

Company 1. price 2. institution 3. stock 4. IPO 5. dividend 4. share

The purpose of doing this second step is to determine whether these features affect whether the entities are CEOs or companies respectively. This can be used in classification, by looking at whether the names in the training data are also present in the features.

In the first two loops, I extract the 2013 data, which I will be using for training data. The training data sets have a label “A” at the start of the data names. I will be using the 2014 data as a testing data set; these data sets have a label “B” at the start of the data names.

```
[17]: os.chdir(new_path)
      os.listdir(os.getcwd())

      path_train = os.path.abspath(os.listdir(os.getcwd())[0])
      path_test = os.path.abspath(os.listdir(os.getcwd())[1])

```

```
[20]: #Making sure we are in the right data directory
      A_person_array_train = []
      A_company_array_train = []

```

```

A_percent_array_train = []

A_person_partner = []
A_person_exec = []
A_person_founder = []
A_person_chief = []

A_person_partner_name = []
A_person_exec_name = []
A_person_founder_name = []
A_person_chief_name = []

A_comp_price = []
A_comp_int = []
A_comp_stock = []
A_comp_div = []
A_comp_ipo = []
A_comp_share = []

A_comp_price_name = []
A_comp_int_name = []
A_comp_stock_name = []
A_comp_div_name = []
A_comp_ipo_name = []
A_comp_share_name = []

#Getting paths
os.chdir(path_train)
files = os.listdir(os.getcwd()) #Get the list of files in the particular folder

for file_index in tqdm(range(0,len(files)-1)): #For each folder
    text_file_path = os.path.abspath(os.listdir(os.getcwd())[file_index]) #Get
    →the file path

    string_values = ""
    with open(text_file_path, "r", errors = 'ignore') as f: #Open the file and
    →read it in
        for line in f.readlines():
            string_values = string_values + str(line)

    sent_seg = sent_tokenize(string_values)

    for sent in sent_seg: #For each individual sentence in the segmented
    →sentence array
        tokenized_sent = word_tokenize_regex(sent) #Tokenize the sentence
        filtered_sent = [word for word in tokenized_sent if word not in
    →stop_words] #Removing the stop words

```

```

    lemmatized = lemmatize_pos(lemmatizer, filtered_sent) #Conducting POS_
    →tagging and lemmatizing using lemmatize_pos
    doc = nlp(sent)

    for ent in doc.ents:
        if ent.label_ == "PERSON": #If the label is a person
            if ent.text not in A_person_array_train:
                A_person_array_train.append(ent.text) #Store the text in_
    →the person matrix
            if ent.label_ == "ORG": #If the label is an organization
                if ent.text not in A_company_array_train:
                    A_company_array_train.append(ent.text) #Store the label in_
    →the company matrix
            if ent.label_ == "PERCENT": # If the label is a percent
                A_percent_array_train.append(ent.text) #Store the label in the_
    →percent matrix


    for ent in doc.ents:
        if ent.label_ == "PERSON": #If the label is a person
            if "partner" in filtered_sent:
                if ent.text not in A_person_partner_name:
                    A_person_partner.append(1)
                    A_person_partner_name.append(ent.text)

            if "executive" in filtered_sent:
                if ent.text not in A_person_exec_name:
                    A_person_exec.append(1)
                    A_person_exec_name.append(ent.text)

            if "founder" in filtered_sent:
                if ent.text not in A_person_founder_name:
                    A_person_founder.append(1)
                    A_person_founder_name.append(ent.text)

            if "chief" in filtered_sent:
                if ent.text not in A_person_chief_name:
                    A_person_chief.append(1)
                    A_person_chief_name.append(ent.text)

        if ent.label_ == "ORG": #If the label is an organization
            if "price" in filtered_sent:
                if ent.text not in A_comp_price_name:
                    A_comp_price.append(1)
                    A_comp_price_name.append(ent.text)

```



```

if "institution" in filtered_sent:
    if ent.text not in A_comp_int_name:
        A_comp_int.append(1)
        A_comp_int_name.append(ent.text)
if "stock" in filtered_sent:
    if ent.text not in A_comp_stock_name:
        A_comp_stock.append(1)
        A_comp_stock_name.append(ent.text)
if "IPO" in filtered_sent:
    if ent.text not in A_comp_ipo_name:
        A_comp_ipo.append(1)
        A_comp_ipo_name.append(ent.text)
if "dividend" in filtered_sent:
    if ent.text not in A_comp_div_name:
        A_comp_div.append(1)
        A_comp_div_name.append(ent.text)
if "share" in filtered_sent:
    if ent.text not in A_comp_share_name:
        A_comp_share.append(1)
        A_comp_share_name.append(ent.text)

```

```

0%|      | 0/364 [00:00<?, ?it/s]
0%|      | 1/364 [00:12<1:13:50, 12.20s/it]
1%|      | 2/364 [00:32<1:28:45, 14.71s/it]
1%|      | 3/364 [00:49<1:32:11, 15.32s/it]
1%|      | 4/364 [01:00<1:24:30, 14.08s/it]
1%|      | 5/364 [01:12<1:20:21, 13.43s/it]
2%|      | 6/364 [01:26<1:21:36, 13.68s/it]
2%|      | 7/364 [01:44<1:29:00, 14.96s/it]
2%|      | 8/364 [01:58<1:27:03, 14.67s/it]
2%|      | 9/364 [02:12<1:24:35, 14.30s/it]
3%|      | 10/364 [02:23<1:19:34, 13.49s/it]
3%|      | 11/364 [02:35<1:16:22, 12.98s/it]
3%|      | 12/364 [02:51<1:22:03, 13.99s/it]
4%|      | 13/364 [03:00<1:11:43, 12.26s/it]
4%|      | 14/364 [03:10<1:07:45, 11.61s/it]
4%|      | 15/364 [03:20<1:05:46, 11.31s/it]
4%|      | 16/364 [03:27<58:03, 10.01s/it]
5%|      | 17/364 [03:40<1:01:54, 10.70s/it]
5%|      | 18/364 [03:59<1:17:15, 13.40s/it]
5%|      | 19/364 [04:05<1:04:20, 11.19s/it]
5%|      | 20/364 [04:24<1:17:25, 13.50s/it]
6%|      | 21/364 [04:32<1:07:36, 11.83s/it]
6%|      | 22/364 [04:44<1:06:27, 11.66s/it]
6%|      | 23/364 [04:57<1:08:32, 12.06s/it]

```

```

99%|| 360/364 [1:14:46<00:36, 9.17s/it]
99%|| 361/364 [1:14:53<00:25, 8.45s/it]
99%|| 362/364 [1:14:59<00:15, 7.97s/it]
100%|| 363/364 [1:15:04<00:06, 6.85s/it]

```

I had noticed something was wrong for 2 arrays, so I redid the analysis for both of these arrays.

```

[38]: A_comp_price_name = []

A_comp_ipo_name = []

#Getting paths
os.chdir(path_train)
files = os.listdir(os.getcwd()) #Get the list of files in the particular folder

for file_index in tqdm(range(0,len(files)-1)): #For each folder
    text_file_path = os.path.abspath(os.listdir(os.getcwd())[file_index]) #Get
    ↳the file path

    string_values = ""
    with open(text_file_path, "r", errors = 'ignore') as f: #Open the file and
    ↳read it in
        for line in f.readlines():
            string_values = string_values + str(line)

    sent_seg = sent_tokenize(string_values)

    for sent in sent_seg: #For each individual sentence in the segmented
    ↳sentence array
        tokenized_sent = word_tokenize_regex(sent) #Tokenize the sentence
        filtered_sent = [word for word in tokenized_sent if word not in
    ↳stop_words] #Removing the stop words
        lemmatized = lemmatize_pos(lemmatizer, filtered_sent) #Conducting POS
    ↳tagging and lemmatizing using lemmatize_pos
        doc = nlp(sent)

    for ent in doc.ents:

        if ent.label_ == "ORG": #If the label is an organization
            if "price" in filtered_sent:
                if ent.text not in A_comp_price_name:
                    A_comp_price.append(1)
                    A_comp_price_name.append(ent.text)

            if "IPO" in filtered_sent:

```

```

if ent.text not in A_comp_ipo_name:
    A_comp_ipo.append(1)
    A_comp_ipo_name.append(ent.text)

```

```

0%|          | 0/364 [00:00<?, ?it/s]
0%|          | 1/364 [00:10<1:03:18, 10.47s/it]
1%|          | 2/364 [00:25<1:11:37, 11.87s/it]
1%|          | 3/364 [00:36<1:09:12, 11.50s/it]
1%|          | 4/364 [00:45<1:04:09, 10.69s/it]
1%|          | 5/364 [00:54<1:01:22, 10.26s/it]
2%|          | 6/364 [01:04<1:01:29, 10.31s/it]
2%|          | 7/364 [01:18<1:07:33, 11.35s/it]
2%|          | 8/364 [01:27<1:03:57, 10.78s/it]
2%|          | 9/364 [01:43<1:12:05, 12.19s/it]
3%|          | 10/364 [01:57<1:14:33, 12.64s/it]
3%|          | 11/364 [02:07<1:10:34, 12.00s/it]
3%|          | 12/364 [02:21<1:13:10, 12.47s/it]
4%|          | 13/364 [02:28<1:03:14, 10.81s/it]
4%|          | 14/364 [02:35<57:16, 9.82s/it]
4%|          | 15/364 [02:44<55:07, 9.48s/it]
4%|          | 16/364 [02:48<46:17, 7.98s/it]
5%|          | 17/364 [02:58<48:37, 8.41s/it]
5%|          | 18/364 [03:15<1:03:14, 10.97s/it]
5%|          | 19/364 [03:19<51:21, 8.93s/it]
5%|          | 20/364 [03:32<58:53, 10.27s/it]
6%|          | 21/364 [03:39<52:29, 9.18s/it]
6%|          | 22/364 [03:48<52:31, 9.21s/it]
6%|          | 23/364 [04:00<56:10, 9.88s/it]
7%|          | 24/364 [04:06<49:50, 8.80s/it]
7%|          | 25/364 [04:16<51:18, 9.08s/it]
7%|          | 26/364 [04:34<1:06:25, 11.79s/it]
7%|          | 27/364 [04:52<1:17:39, 13.83s/it]
8%|          | 28/364 [05:06<1:16:46, 13.71s/it]
8%|          | 29/364 [05:17<1:12:23, 12.97s/it]
8%|          | 30/364 [05:20<55:42, 10.01s/it]
9%|          | 31/364 [05:31<57:13, 10.31s/it]
9%|          | 32/364 [05:37<49:36, 8.96s/it]
9%|          | 33/364 [05:44<46:02, 8.35s/it]
9%|          | 34/364 [05:50<42:56, 7.81s/it]
10%|         | 35/364 [05:58<42:05, 7.68s/it]
10%|         | 36/364 [06:15<57:11, 10.46s/it]
10%|         | 37/364 [06:31<1:06:19, 12.17s/it]
10%|         | 38/364 [06:43<1:06:27, 12.23s/it]
11%|         | 39/364 [06:50<56:57, 10.51s/it]
11%|         | 40/364 [07:00<57:06, 10.58s/it]
11%|         | 41/364 [07:15<1:03:22, 11.77s/it]

```

```

91%| | 330/364 [58:38<04:44, 8.36s/it]
91%| | 331/364 [58:48<04:50, 8.81s/it]
91%| | 332/364 [59:01<05:27, 10.24s/it]
91%| | 333/364 [59:13<05:33, 10.75s/it]
92%| | 334/364 [59:32<06:36, 13.20s/it]
92%| | 335/364 [59:44<06:11, 12.80s/it]
92%| | 336/364 [59:58<06:10, 13.23s/it]
93%| | 337/364 [1:00:10<05:47, 12.89s/it]
93%| | 338/364 [1:00:28<06:11, 14.28s/it]
93%| | 339/364 [1:00:33<04:50, 11.60s/it]
93%| | 340/364 [1:00:38<03:53, 9.74s/it]
94%| | 341/364 [1:00:42<02:59, 7.80s/it]
94%| | 342/364 [1:00:47<02:34, 7.03s/it]
94%| | 343/364 [1:00:51<02:08, 6.12s/it]
95%| | 344/364 [1:00:54<01:45, 5.27s/it]
95%| | 345/364 [1:01:13<02:56, 9.31s/it]
95%| | 346/364 [1:01:25<03:00, 10.02s/it]
95%| | 347/364 [1:01:35<02:54, 10.25s/it]
96%| | 348/364 [1:01:47<02:51, 10.74s/it]
96%| | 349/364 [1:02:01<02:55, 11.69s/it]
96%| | 350/364 [1:02:13<02:41, 11.56s/it]
96%| | 351/364 [1:02:25<02:32, 11.73s/it]
97%| | 352/364 [1:02:32<02:03, 10.33s/it]
97%| | 353/364 [1:02:40<01:47, 9.79s/it]
97%| | 354/364 [1:02:54<01:48, 10.85s/it]
98%| | 355/364 [1:03:05<01:40, 11.16s/it]
98%| | 356/364 [1:03:17<01:29, 11.16s/it]
98%| | 357/364 [1:03:20<01:00, 8.70s/it]
98%| | 358/364 [1:03:22<00:41, 6.92s/it]
99%| | 359/364 [1:03:40<00:50, 10.03s/it]
99%| | 360/364 [1:03:49<00:39, 9.90s/it]
99%| | 361/364 [1:03:56<00:27, 9.07s/it]
99%| | 362/364 [1:04:03<00:16, 8.41s/it]
100%| | 363/364 [1:04:08<00:07, 7.31s/it]

```

```

[53]: B_person_array_train = []
      B_company_array_train = []
      B_percent_array_train = []

      B_person_partner = []
      B_person_exec = []
      B_person_founder = []
      B_person_chief = []

      B_person_partner_name = []
      B_person_exec_name = []
      B_person_founder_name = []

```

```

B_person_chief_name = []

B_comp_price = []
B_comp_int = []
B_comp_stock = []
B_comp_div = []
B_comp_ipo = []
B_comp_share = []

B_comp_price_name = []
B_comp_int_name = []
B_comp_stock_name = []
B_comp_div_name = []
B_comp_ipo_name = []
B_comp_share_name = []

#Getting paths

os.chdir(path_test)
files = os.listdir(os.getcwd()) #Get the list of files in the particular folder
for file_index in tqdm(range(0,len(files)-1)): #For each folder
    text_file_path = os.path.abspath(os.listdir(os.getcwd())[file_index]) #Get the file path

    string_values = ""
    with open(text_file_path, "r", errors = 'ignore') as f: #Open the file and read it in
        for line in f.readlines():
            string_values = string_values + str(line)

    sent_seg = sent_tokenize(string_values)

    for sent in sent_seg: #For each individual sentence in the segmented sentence array
        tokenized_sent = word_tokenize_regex(sent) #Tokenize the sentence
        filtered_sent = [word for word in tokenized_sent if word not in stop_words] #Removing the stop words
        lemmatized = lemmatize_pos(lemmatizer, filtered_sent) #Conducting POS tagging and lemmatizing using lemmatize_pos
        doc = nlp(sent)

        for ent in doc.ents:
            if ent.label_ == "PERSON": #If the label is a person
                if ent.text not in B_person_array_train:
                    B_person_array_train.append(ent.text) #Store the text in the person matrix

```

```

        if ent.label_ == "ORG": #If the label is an organization
            if ent.text not in B_company_array_train:
                B_company_array_train.append(ent.text) #Store the label in
→the company matrix
            if ent.label_ == "PERCENT": # If the label is a percent
                B_percent_array_train.append(ent.text) #Store the label in the
→percent matrix

for ent in doc.ents:
    if ent.label_ == "PERSON": #If the label is a person
        if "partner" in filtered_sent:
            if ent.text not in B_person_partner_name:
                B_person_partner.append(1)
                B_person_partner_name.append(ent.text)

        if "executive" in filtered_sent:
            if ent.text not in B_person_exec_name:
                B_person_exec.append(1)
                B_person_exec_name.append(ent.text)

        if "founder" in filtered_sent:
            if ent.text not in B_person_founder_name:
                B_person_founder.append(1)
                B_person_founder_name.append(ent.text)

        if "chief" in filtered_sent:
            if ent.text not in B_person_chief_name:
                B_person_chief.append(1)
                B_person_chief_name.append(ent.text)

    if ent.label_ == "ORG": #If the label is an organization
        if "price" in filtered_sent:
            if ent.text not in B_comp_price_name:
                B_comp_price.append(1)
                B_comp_price_name.append(ent.text)
        if "institution" in filtered_sent:
            if ent.text not in B_comp_int_name:
                B_comp_int.append(1)
                B_comp_int_name.append(ent.text)
        if "stock" in filtered_sent:
            if ent.text not in B_comp_stock_name:
                B_comp_stock.append(1)
                B_comp_stock_name.append(ent.text)

```

```

if "IPO" in filtered_sent:
    if ent.text not in B_comp_ipo_name:
        B_comp_ipo.append(1)
        B_comp_ipo_name.append(ent.text)
if "dividend" in filtered_sent:
    if ent.text not in B_comp_div_name:
        B_comp_div.append(1)
        B_comp_div_name.append(ent.text)
if "share" in filtered_sent:
    if ent.text not in B_comp_share_name:
        B_comp_share.append(1)
        B_comp_share_name.append(ent.text)

```

```

0%|          | 0/365 [00:00<?, ?it/s]
0%|          | 1/365 [00:36<3:41:06, 36.45s/it]
1%|          | 2/365 [00:52<3:03:05, 30.26s/it]
1%|          | 3/365 [01:05<2:31:55, 25.18s/it]
1%|          | 4/365 [01:44<2:56:01, 29.26s/it]
1%|          | 5/365 [01:51<2:16:13, 22.71s/it]
2%|          | 6/365 [02:28<2:41:36, 27.01s/it]
2%|          | 7/365 [03:03<2:55:21, 29.39s/it]
2%|          | 8/365 [03:17<2:26:05, 24.55s/it]
2%|          | 9/365 [03:50<2:40:37, 27.07s/it]
3%|          | 10/365 [04:11<2:30:42, 25.47s/it]
3%|          | 11/365 [04:42<2:38:58, 26.94s/it]
3%|          | 12/365 [05:06<2:33:51, 26.15s/it]
4%|          | 13/365 [05:31<2:31:43, 25.86s/it]
4%|          | 14/365 [05:40<2:02:16, 20.90s/it]
4%|          | 15/365 [06:10<2:17:18, 23.54s/it]

```

```

96%|| 352/365 [1:59:37<03:28, 16.00s/it]
97%|| 353/365 [1:59:54<03:15, 16.29s/it]
97%|| 354/365 [2:00:08<02:51, 15.56s/it]
97%|| 355/365 [2:00:26<02:44, 16.46s/it]
98%|| 356/365 [2:00:37<02:13, 14.84s/it]
98%|| 357/365 [2:00:53<02:01, 15.22s/it]
98%|| 358/365 [2:01:14<01:58, 16.86s/it]
98%|| 359/365 [2:01:26<01:31, 15.26s/it]
99%|| 360/365 [2:01:34<01:06, 13.35s/it]
99%|| 361/365 [2:01:51<00:56, 14.17s/it]
99%|| 362/365 [2:02:08<00:45, 15.18s/it]
99%|| 363/365 [2:02:25<00:31, 15.83s/it]
100%|| 364/365 [2:02:36<00:14, 14.36s/it]

```

Cleaning the NER percent data

The same process of cleaning the percent data has to be conducted on the percent_array_train data. Since the NER data has a lot more variations, only a simple cleaning will be done.

```

[57]: A_percent_array_update = []

for row in range(0, len(A_percent_array_train)-1):
    value = str(A_percent_array_train[row]) #Convert the value to a string for
    →easier cleaning

    if "/" in value:
        try:
            if "percent" in value:
                value = value.replace(" percent", "")
                value = str(value) + "%"
                A_percent_array_update.append(value)
                continue

        else:

```



```

        value = float(Fraction(value)) * 100
        value = str(value) + "%"
        A_percent_array_update.append(value)
        continue
    except:
        continue

    if "half" in value:
        value = "0.5%"
        A_percent_array_update.append(value)
        continue

    if "quarter" in value:
        value = "0.25%"
        A_percent_array_update.append(value)
        continue

    if value == "half a percent":
        value = "0.5%"
        A_percent_array_update.append(value)
        continue

    if "." in value and "%" not in value:
        try:
            if "percentage" in value:
                value = value.replace(" percentage", "")
                value = str(value) + "%"
                A_percent_array_update.append(value)
                continue

            elif "percent" in value:
                value = value.replace(" percent", "")
                value = str(value) + "%"
                A_percent_array_update.append(value)
                continue

            elif float(value) >= 0.001:
                value = str(value) + "%"
                A_percent_array_update.append(value)
                continue
        except:
            continue

    if "percentage" in value:
        value = value.replace(" percentage", "")
        value = str(value) + "%"

```

```

A_percent_array_update.append(value)
continue

if "percent" in value: #If there is a percent in the string
    try:
        value = value.replace(" percent","") #Remove the percent and the
→ space behind it
        value = w2n.word_to_num(value) #Convert to number using word2num
        value = str(value) + "%" #Convert to string using str
        A_percent_array_update.append(value)
        continue
    except:
        continue

if "%" not in value: # If there is no percent

    try:
        if float(w2n.word_to_num(value)) < 100:
            value = str(value) + "%" #Add a percentage sign at the end
            A_percent_array_update.append(value)
            continue
        except:
            continue
        #value = int(value) * 100 #Multiply by 100

    else:
        try:
            value = w2n.word_to_num(value) #Convert to number using word2num
            value = str(value) + "%" #Convert to string using str
            A_percent_array_update.append(value)
            continue
        except:
            continue

```

Now we can extract the percentages in the data

```

[235]: trained_percent = []
for p in A_percent_array_update:
    if p in percent_update:
        trained_percent.append(p)

```

Evaluating the accuracy of the training models

To evaluate the accuracy of the training models, I will compare the values entities selected through my feature selection with the training models in the beginning. I can then compile a summary statistic to determine how accurate the classification model is.

```

[58]: #Checking if NER data selection of "PERSON" is in the training data

```

```

A_name_in_train_ceo = []
A_name_in_train_comp = []

for name in A_person_array_train:
    if name in ceo_concat:
        A_name_in_train_ceo.append(1)
    else:
        A_name_in_train_ceo.append(0)

for name in A_company_array_train:
    if name in company_concat:
        A_name_in_train_comp.append(1)
    else:
        A_name_in_train_comp.append(0)

#Checking if NER data selection of "PERSON" is in the feature data

A_name_in_partner = [0]*len(A_person_array_train)
A_name_in_exec = [0]*len(A_person_array_train)
A_name_in_founder = [0]*len(A_person_array_train)
A_name_in_chief = [0]*len(A_person_array_train)

A_name_in_price = [0]*len(A_company_array_train)
A_name_in_int = [0]*len(A_company_array_train)
A_name_in_stock = [0]*len(A_company_array_train)
A_name_in_ipo = [0]*len(A_company_array_train)
A_name_in_div = [0]*len(A_company_array_train)
A_name_in_share = [0]*len(A_company_array_train)

for ceo in range(0,len(A_person_array_train)-1):
    if A_person_array_train[ceo] in A_person_partner_name:
        A_name_in_partner[ceo] = 1
    if A_person_array_train[ceo] in A_person_exec_name:
        A_name_in_exec[ceo] = 1
    if A_person_array_train[ceo] in A_person_founder_name:
        A_name_in_founder[ceo] = 1
    if A_person_array_train[ceo] in A_person_chief_name:
        A_name_in_chief[ceo] = 1

for comp in range(0,len(A_company_array_train)-1):
    if A_company_array_train[comp] in A_comp_price_name:
        A_name_in_price[comp] = 1
    if A_company_array_train[comp] in A_comp_int_name:
        A_name_in_int[comp] = 1
    if A_company_array_train[comp] in A_comp_stock_name:
        A_name_in_stock[comp] = 1

```

```

if A_company_array_train[comp] in A_comp_ipo_name:
    A_name_in_ipo[comp] = 1
if A_company_array_train[comp] in A_comp_div_name:
    A_name_in_div[comp] = 1
if A_company_array_train[comp] in A_comp_share_name:
    A_name_in_share[comp] = 1

```

[59]: *#Checking if NER data selection of "ORG" is in the training data*

```

B_name_in_train_ceo = []
B_name_in_train_comp = []

```

```

for name in B_person_array_train:
    if name in ceo_concat:
        B_name_in_train_ceo.append(1)
    else:
        B_name_in_train_ceo.append(0)

```

```

for name in B_company_array_train:
    if name in company_concat:
        B_name_in_train_comp.append(1)
    else:
        B_name_in_train_comp.append(0)

```

#Checking if NER data selection of "ORG" is in the feature data

```

B_name_in_partner = [0]*len(B_person_array_train)
B_name_in_exec = [0]*len(B_person_array_train)
B_name_in_founder = [0]*len(B_person_array_train)
B_name_in_chief = [0]*len(B_person_array_train)

```

```

B_name_in_price = [0]*len(B_company_array_train)
B_name_in_int = [0]*len(B_company_array_train)
B_name_in_stock = [0]*len(B_company_array_train)
B_name_in_ipo = [0]*len(B_company_array_train)
B_name_in_div = [0]*len(B_company_array_train)
B_name_in_share = [0]*len(B_company_array_train)

```

```

for ceo in range(0,len(B_person_array_train)-1):
    if B_person_array_train[ceo] in B_person_partner_name:
        B_name_in_partner[ceo] = 1
    if B_person_array_train[ceo] in B_person_exec_name:
        B_name_in_exec[ceo] = 1
    if B_person_array_train[ceo] in B_person_founder_name:
        B_name_in_founder[ceo] = 1
    if B_person_array_train[ceo] in B_person_chief_name:

```

```

B_name_in_chief[ceo] = 1

for comp in range(0, len(B_person_array_train)-1):
    if B_company_array_train[comp] in B_comp_price_name:
        B_name_in_price[comp] = 1
    if B_company_array_train[comp] in B_comp_int_name:
        B_name_in_int[comp] = 1
    if B_company_array_train[comp] in B_comp_stock_name:
        B_name_in_stock[comp] = 1
    if B_company_array_train[comp] in B_comp_ipo_name:
        B_name_in_ipo[comp] = 1
    if B_company_array_train[comp] in B_comp_div_name:
        B_name_in_div[comp] = 1
    if B_company_array_train[comp] in B_comp_share_name:
        B_name_in_share[comp] = 1

```

[134]: *#Making the training data frame*

```

data_train_ceo = pd.DataFrame({'NER': A_person_array_train, 'train':␣
    ↳A_name_in_train_ceo, 'partner': A_name_in_partner, 'exec': A_name_in_exec,␣
    ↳'founder': A_name_in_founder, 'chief': A_name_in_chief})
data_train_comp = pd.DataFrame({'NER': A_company_array_train, 'train':␣
    ↳A_name_in_train_comp, 'price': A_name_in_price, 'institution':␣
    ↳A_name_in_int, 'stock': A_name_in_stock, 'ipo': A_name_in_ipo, 'dividend':␣
    ↳A_name_in_div, 'share': A_name_in_share})

```

[201]: *#Making the testing data frame*

```

data_test_ceo = pd.DataFrame({'NER': B_person_array_train, 'train':␣
    ↳B_name_in_train_ceo, 'partner': B_name_in_partner, 'exec': B_name_in_exec,␣
    ↳'founder': B_name_in_founder, 'chief': B_name_in_chief})
data_test_comp = pd.DataFrame({'NER': B_company_array_train, 'train':␣
    ↳B_name_in_train_comp, 'price': B_name_in_price, 'institution':␣
    ↳B_name_in_int, 'stock': B_name_in_stock, 'ipo': B_name_in_ipo, 'dividend':␣
    ↳B_name_in_div, 'share': B_name_in_share})

```

0.5 Classification model

I used a logistic regression model to predict how well the features I chose describe the names in the training set. These models are seen below.

[69]: `data_train_ceo.to_csv(r'/Users/ArshyaSrinivas/Google Drive/Engineering/Junior␣
 ↳year/WINTER 2019/IEMS 308/text_analytics/train_ceo.csv')`

[131]: `data_test_ceo.to_csv(r'/Users/ArshyaSrinivas/Google Drive/Engineering/Junior␣
 ↳year/WINTER 2019/IEMS 308/text_analytics/test_ceo.csv')
data_train_comp.to_csv(r'/Users/ArshyaSrinivas/Google Drive/Engineering/Junior␣
 ↳year/WINTER 2019/IEMS 308/text_analytics/test_ceo.csv')`

```
data_test_comp.to_csv(r'/Users/ArshyaSrinivas/Google Drive/Engineering/Junior_
→year/WINTER 2019/IEMS 308/text_analytics/test_ceo.csv')
```

```
[214]: logmodel = LogisticRegression()
```

```
[215]: X_train_ceo = data_train_ceo.loc[:, "partner":"chief"]
Y_train_ceo = data_train_ceo.loc[:, "train"]

X_train_comp = data_train_comp.loc[:, "price":"share"]
Y_train_comp = data_train_comp.loc[:, "train"]

X_test_ceo = data_test_ceo.loc[:, "partner":"chief"]
Y_test_ceo = data_test_ceo.loc[:, "train"]

X_test_comp = data_test_comp.loc[:, "price":"share"]
Y_test_comp = data_test_comp.loc[:, "train"]
```

```
[216]: logmodel.fit(X_train_ceo, Y_train_ceo)
predictions_ceo = logmodel.predict(X_test_ceo)
```

```
/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)
```

```
[217]: from sklearn.metrics import classification_report
print(classification_report(Y_test_ceo, predictions_ceo))
```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	32682
1	0.23	0.00	0.01	611
accuracy			0.98	33293
macro avg	0.61	0.50	0.50	33293
weighted avg	0.97	0.98	0.97	33293

```
[115]: logmodel = LogisticRegression()
```

```
[116]: logmodel.fit(X_train_comp, Y_train_comp)
predictions_comp = logmodel.predict(X_test_comp)
```

```
/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)
```

```
[117]: from sklearn.metrics import classification_report
print(classification_report(Y_test_comp, predictions_comp))
```

	precision	recall	f1-score	support
0	0.97	1.00	0.99	41564
1	0.48	0.04	0.07	1184
accuracy			0.97	42748
macro avg	0.73	0.52	0.53	42748
weighted avg	0.96	0.97	0.96	42748

The results above are quite interesting, as it suggests that CEOs are correctly predicted to be CEOs 23% of the time while non-CEOs are correctly predicted to be non-CEOs 98% of the time. Additionally, companies are correctly predicted to be companies 48% of the time while non-companies are correctly predicted to be non-companies 97% of the time.

However, it is important to look at the sensitivity and the specificity of the results. The results for specificity, i.e. the ability to determine the true negative rates are extremely high. Here, it is important to remember that most of the results were initially 0 (i.e. the feature in that column was not present for a particular name) to begin with, due to the design of this experiment. If we were to guess every single value to be 0, then our specificity would also be high. Hence, we should focus more on the sensitivity of the experiment.

The classification model does a better job at classifying whether a company is a company than if a CEO is a CEO when measured against the training data. This could be because I used more features to describe company than I did to describe CEOs. In the future, it might be more beneficial looking at whether more features in a sentence could describe a person to be CEO.

0.6 Saving arrays

First, we will extract the CEO names that were predicted accurately with the model

```
[220]: data_test_ceo.insert(6, "predictions", predictions_ceo , True)
```

```
[222]: ones = data_test_ceo['train'] == 1
new_data_ceo = data_test_ceo[ones]
```

```
[223]: #Save empty array of indexes that need to be dropped
index_drop = []

#If the predicted value is not the same as the training value

for i in tqdm(range(0, len(new_data_ceo))):
    if new_data_ceo.iloc[i, 6] == 0:
        index_drop.append(i)
```

0%| | 0/611 [00:00<?, ?it/s]

```
[225]: #Drop all indicies that do not match
new_data_ceo.drop(new_data_ceo.index[index_drop], inplace=True)
```

/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:3940: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>
errors=errors)

```
[230]: ceo = open("ceoname_extract", "w")
for name in new_data_ceo["NER"]:
    ceo.write(name + '\n')
ceo.close()
```

Now, we will extract the company names that were predicted accurately with the model

```
[202]: data_test_comp.insert(8, "predictions", predictions_comp , True)
```

```
[203]: ones = data_test_comp['train'] == 1
new_data_comp = data_test_comp[ones]
```

```
[204]: #Save empty array of indexes that need to be dropped
index_drop = []

#If the predicted value is not the same as the training value

for i in tqdm(range(0,len(new_data_comp))):
    if new_data_comp.iloc[i,8] == 0:
        index_drop.append(i)
```

0%| | 0/1184 [00:00<?, ?it/s]

```
[205]: #Drop all indicies that do not match
new_data_comp.drop(new_data_comp.index[index_drop], inplace=True)
```

```
[211]: new_data_comp["NER"]
```

```
[211]: 0          Reuters
6          JPMorgan
7      Goldman Sachs
9      Morgan Stanley
13     Business Insider
15          Tesla
32     McDonald's
33          NYSE
```



```

34         Google
38         SEC
43         Yahoo
54         Bloomberg
65         UBS
79         Nasdaq
82         Amazon
83         Coca-Cola
99         Microsoft
101        Facebook
128        Wells Fargo
229        Apple
237        Chipotle
345        BlackRock
356        Alibaba
360        CNBC
401        NASDAQ
586        Verizon
622        Ford
623        Fiat
715        Goldman
724        Credit Suisse
802        Deutsche Bank
876        Samsung
913        Thomson Reuters
981        eBay
1124       Bank of America
1241       Citigroup
1637       GM
1644       Chrysler
1943       Ferrari
2650       Visa
2712       Starbucks
2762       AOL
2766       Fannie Mae
2864       Tesco
3533       Barclays
3935       SoftBank
Name: NER, dtype: object

```

```

[212]: comp = open("compname_extract", "w")
for name in new_data_comp["NER"]:
    comp.write(name + '\n')
comp.close()

```

And finally, we will extract all the percentage values.

```
[240]: perc = open("percentage_extract", "w")  
       for name in trained_percent:  
           perc.write(name + '\n')  
       perc.close()
```

```
[ ]:
```