Text Analytics

March 2, 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

[645]:
```
#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
```

[25]:
```
#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]   Unzipping corpora/stopwords.zip.
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/ArshyaSrinivas/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/ArshyaSrinivas/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/ArshyaSrinivas/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[25]:
```
True
```

[254]:
```
#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

[255]:
```
#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)
```

[256]:
```
#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. The code I used to install it is as follows:

[615]:
```
%%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/Lemmatisation/Stemming 6. NER algorithm

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

Step 1: Obtaining the raw text

[502]:
```
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.

[448]:
```
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.

[450]:
```
stop_words=sorted(set(stopwords.words("english")))
```

Step 5: POS tagging and lemmatization

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

[452]:
```
lemmatizer = WordNetLemmatizer()
```

```
[('Why', 'WRB'), ('are', 'VBP'), ('there', 'RB'), ('so', 'RB'), ('many', 'JJ'),
('monkeys', 'NNS'), ('in', 'IN'), ('this', 'DT'), ('area', 'NN'), ('?', '.')]
```

```
Lemmatized sentence:
['Why', 'be', 'there', 'so', 'many', 'monkey', 'in', 'this', 'area', '?']
```

[451]:
```
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

[125]:
```
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 alogrithm. 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.

[562]:
```
#Making sure we are in the right data directory
os.chdir(new_path)
os.listdir(os.getcwd())

person_array_train = []
company_array_train = []
```

```
percent_array_train = []
person_array_feature = []
company_array_feature = []

paths = [os.path.abspath(os.listdir(os.getcwd())[0]),os.path.abspath(os.
 ↪listdir(os.getcwd())[1])] #Getting paths

for path_dir in paths:
    os.chdir(path_dir) #Change the path to a particular folder (2013 or 2014)
    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 fle␣
 ↪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 person_array_train:
                        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 company_array_train:
                        company_array_train.append(ent.text) #Store the label␣
 ↪in the company matrix
                if ent.label_ == "PERCENT": # If the label is a percent
                    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 or "executive" in␣
↪filtered_sent or "founder" in filtered_sent or "chief" in filtered_sent:
                        person_array_feature.append(ent.text)#Check to see if␣
↪the sentence has other features
                        if ent.text not in person_array_feature:
                            person_array_feature.append(ent.text)
                if ent.label_ == "ORG": #If the label is an organization
                    if "price" in filtered_sent or "institution" in␣
↪filtered_sent or "stock" in filtered_sent or "IPO" in filtered_sent or␣
↪"dividend" in filtered_sent or "share" in filtered_sent:
                        if ent.text not in company_array_feature:
                            company_array_feature.append(ent.text)
```

```
  0%|          | 0/364 [00:00<?,
?it/s]
```

```
      131
      132     def predict(self, X):
  --> 133         y, _ = self.begin_update(X, drop=None)
      134         return y
      135

      _parser_model.pyx in spacy.syntax._parser_model.ParserModel.
↪begin_update()

      _parser_model.pyx in spacy.syntax._parser_model.ParserStepModel.
↪__init__()

      ~/anaconda3/lib/python3.7/site-packages/thinc/neural/_classes/
↪feed_forward.py in begin_update(self, X, drop)
       44         callbacks = []
       45         for layer in self._layers:
  ---> 46             X, inc_layer_grad = layer.begin_update(X, drop=drop)
       47             callbacks.append(inc_layer_grad)
       48

      ~/anaconda3/lib/python3.7/site-packages/spacy/_ml.py in flatten(seqs,␣
↪drop)
      795         return ops.unflatten(d_X, lengths, pad=0)
      796
  --> 797     X = ops.flatten(seqs, pad=0)
      798     return X, finish_update
      799

      KeyboardInterrupt:
```

Unfortunately, due to time constraints, the loop was unable to loop through files in both the 2013 and 2014 folders. Hence, only files in the 2013 folder were used.

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.

[576]:
```
percent_array_update = []

for row in range(0, len(percent_array_train)-1):
    value = str(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) + "%"
                percent_array_update.append(value)
                continue

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

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

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


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

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

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

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

Evaluating the accuracy of the training models

To evaluate the accuracy of the training models, I will compare the values of the labeled entities selected using NER with the training models in the beginning. I can then compile a summary

statistic to determine how accurate the selection is by comparing how many additional features are added using the feature selection

[607]:
```
#CEO data
trained_ceo = []

for name in person_array_train:
    if name in ceo_concat:
        trained_ceo.append(name)

#Company data

trained_comp = []

for name in company_array_train:
    if name in company_concat:
        trained_comp.append(name)

#Percent data

trained_percent = []

for p in percent_array_update:
    if p in percent_update:
        trained_percent.append(p)
```

[591]:
```
ceo_feature = []

for name in person_array_train:
    if name in person_array_feature:
        ceo_feature.append(name)

company_feature = []

for name in company_array_train:
    if name in company_array_feature:
        company_feature.append(name)
```

Here it was noticed that the number of values in the company_feature was 0. This was because of an indentation error in the code for the loop. I have fixed the indentation error, but because the loop took 8 hours to run, and because of time constraints, I was unable to run the for loop again. If the code were to run now, it would will the company_feature array. Code to test against the training data is presented below, however it will not have run.

0.5 Classification model

I used a logistic regression model to predict how well the features I chose describe the training set and the NER selection of the entity. This can be seen below:

[634]:
```
training_ceo = ceo_concat

NER_ceo = []

for name in ceo_concat:
    if name in person_array_train:
        NER_ceo.append(1)
    else:
        NER_ceo.append(0)


Feature_ceo = []

for name in ceo_concat:
    if name in person_array_feature:
        Feature_ceo.append(1)
    else:
        Feature_ceo.append(0)


data = pd.DataFrame({'name': ceo_concat, 'train': np.ones(len(ceo_concat),␣
 ↪dtype = int),'NER': NER_ceo, 'Feature': Feature_ceo})
```

[646]:
```
X_reg = data.loc[:, "train":"NER"]
Y_reg = data.loc[:,"Feature"]

X_train, X_test, y_train, y_test = train_test_split(X_reg, Y_reg, test_size=0.
 ↪33, random_state=1)
```

[650]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
```

```
/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)
```

[649]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91       266
           1       0.00      0.00      0.00        52

    accuracy                           0.84       318
   macro avg       0.42      0.50      0.46       318
```

```
weighted avg       0.70      0.84      0.76       318


/Users/ArshyaSrinivas/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

From these results, it can be seen that the model is poor in terms of identifying which people are CEOs, but better at identitifying which people are not CEOs. This indicates that the features chosen may not necessarily describe individual and in the future, more feature engineering can be done.

This same training and testing process can be done for the company data as well.

[ ]:
```
training_company = company_concat

NER_company = []

for name in company_concat:
    if name in company_array_train:
        NER_company.append(1)
    else:
        NER_company.append(0)


Feature_company = []

for name in company_concat:
    if name in company_array_feature:
        Feature_company.append(1)
    else:
        Feature_company.append(0)


data = pd.DataFrame({'name': company_concat, 'train': np.
 ↪ones(len(company_concat), dtype = int),'NER': NER_company, 'Feature':␣
 ↪Feature_company})
```

[ ]:
```
X_reg = data.loc[:, "train":"NER"]
Y_reg = data.loc[:,"Feature"]

X_train, X_test, y_train, y_test = train_test_split(X_reg, Y_reg, test_size=0.
 ↪33, random_state=1)
```

[ ]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
predictions
```

[ ]:
```
print(classification_report(y_test,predictions))
```

0.6 Saving arrays

[613]:
```
verified_ceo = []

for name in trained_ceo:
    if name in person_array_feature:
        verified_ceo.append(name)

verified_company = []

for name in trained_comp:
    if name in company_array_feature:
        verified_company.append(name)
```

[616]:
```
ceo = open("ceo", "w")
for name in verified_ceo:
    ceo.write(name + '\n')
ceo.close()
```

[617]:
```
perc = open("percentage", "w")
for name in trained_percent:
    perc.write(name + '\n')
perc.close()
```

[ ]:
```
comp = open("company", "w")
for name in verified_company:
    comp.write(name + '\n')
comp.close()
```