df\_clean\_ceo=df\_clean\_copy.copy()

X\_train, X\_test= train\_test\_split(df\_clean\_ceo, test\_size=0.4)

print (X\_train.shape, X\_test.shape)

ceo\_pos\_list=ceo\_pos\_df['fullname'].unique().tolist()

(405349, 2) (270233, 2)

stop\_wordsceo\_raw = stopwords.words('english')

commonstopword=['San Francisco','Los Angeles','New York','United States','President','Members','On','Twitter','In','Police','The','London','City','China','Japan','Europe','European','Chinese','Governor','Chicago','Hong Kong','White House','Jackson Hole']

ceostopwordceo=['Goldman Sachs','Morgan Stanley','Wells Fargo','Google Finance','Thomson Reuters','Business Insider','Yahoo Finance','Time Warner','Wall Street','Deutsche Bank','Credit Suisse','Elliott Management','Federal Open','Merrill Lynch','Capital Management']

ceostopword=['Justice','Department','Federal Reserve','Central Bank','Communist','Vice Chairman','Vice President','University','Bank','Investment','Company','School']

stop\_wordsceo\_raw.extend(commonstopword)

stop\_wordsceo\_raw.extend(ceostopwordceo)

stop\_wordsceo\_raw.extend(ceostopword)

print(stop\_wordsceo\_raw)

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't", 'San Francisco', 'Los Angeles', 'New York', 'United States', 'President', 'Members', 'On', 'Twitter', 'In', 'Police', 'The', 'London', 'City', 'China', 'Japan', 'Europe', 'European', 'Chinese', 'Governor', 'Chicago', 'Hong Kong', 'White House', 'Jackson Hole', 'Goldman Sachs', 'Morgan Stanley', 'Wells Fargo', 'Google Finance', 'Thomson Reuters', 'Business Insider', 'Yahoo Finance', 'Time Warner', 'Wall Street', 'Deutsche Bank', 'Credit Suisse', 'Elliott Management', 'Federal Open', 'Merrill Lynch', 'Capital Management', 'Justice', 'Department', 'Federal Reserve', 'Central Bank', 'Communist', 'Vice Chairman', 'Vice President', 'University', 'Bank', 'Investment', 'Company', 'School']

train\_pos\_feature=get\_feature\_ready\_step1(X\_train\_raw\_pos)

train\_pos\_feature['bothnnp']=train\_pos\_feature.apply(if\_two\_words\_are\_non,axis=1)

train\_pos\_feature['first\_name\_length']=train\_pos\_feature.apply(return\_first\_len,axis=1)

testa=train\_pos\_feature.apply(if\_sent\_contains\_chief\_executive\_officer,axis=1)

testdf=testa.apply(pd.Series)

train\_pos\_feature['contain\_chief']=testdf.iloc[:,0]

train\_pos\_feature['contain\_exec']=testdf.iloc[:,1]

train\_pos\_feature['contain\_officer']=testdf.iloc[:,2]

len(train\_pos\_feature)

0

10000

Out[18]: 26765

resultlen=train\_pos\_feature['y'].value\_counts().to\_frame()

print (resultlen)

posnum=int(resultlen.iloc[1])

negnum=int(resultlen.iloc[0])

cleaned\_pos\_train\_sample=train\_pos\_feature.loc[train\_pos\_feature['y']==1]

cleaned\_neg\_train\_sample=train\_pos\_feature.loc[train\_pos\_feature['y']==0]

if posnum<negnum:

neg\_new\_size=posnum

negative\_updated\_sample=cleaned\_neg\_train\_sample.iloc[:neg\_new\_size]

train\_shrink=pd.concat([cleaned\_pos\_train\_sample,negative\_updated\_sample])

train\_shrink\_copy=train\_shrink.copy()

train\_shrink\_copy.head(10)

y

0 19857

1 6908

Out[20]:

word string contain\_ceo\_word sentence\_index words\_index y bothnnp \

1 Lloyd Blankfein 1 1 1 1 True

3 Bill Ackman 0 2 0 1 True

4 Pershing Square 0 2 5 1 True

8 Angelo Mozilo 1 3 19 1 True

12 Angelo Mozilo 1 3 19 1 True

15 Alexandra Damsker 1 7 7 1 True

20 Mark Hurd 1 12 18 1 True

25 Steve Schwarzman 1 14 15 1 True

30 Bill Gross 0 19 2 1 True

33 Bill Gross 0 19 2 1 True

first\_name\_length contain\_chief contain\_exec contain\_officer

1 5 0 0 0

3 4 0 0 0

4 8 0 0 0

8 6 0 0 0

12 6 0 0 0

15 9 0 0 0

20 4 1 0 0

25 5 0 0 0

30 4 0 0 0

33 4 0 0 0

#----logistic regression

model\_ceo=LogisticRegression(random\_state=0,solver='lbfgs',multi\_class='multinomial').fit(scaledx\_ceo,Y\_ceo)

y\_pred=model\_ceo.predict(scaledx\_test\_ceo) #testdata

b=accuracy\_score(Y\_test\_ceo,y\_pred)

a = precision\_recall\_fscore\_support(Y\_test\_ceo, y\_pred, average='binary')

#----randomeforest

from sklearn.ensemble import RandomForestClassifier

rfceo=RandomForestClassifier(n\_estimators=500)

rf\_fit=rfceo.fit(scaledx\_ceo,Y\_ceo)

rf\_pred\_test=rf\_fit.predict(scaledx\_test\_ceo)

rf\_score=rf\_fit.score(scaledx\_test\_ceo,Y\_test\_ceo)

rf\_metrics=precision\_recall\_fscore\_support(Y\_test\_ceo,rf\_pred\_test,average='macro')

Logistic Regression for ceos: accuracy score is 0.6253979212211866

Logistic Regression for ceos: precision is 0.3813474941918354, recall is 0.6654119009700304, and f1 score is 0.48483569808534205.

Random Forest for ceos: accuracy score is 0.6854984083151152

Random Forest for ceos: precision is 0.6709376920956758, recall is 0.718524219773442, and f1 score is 0.6612408868765318.

notcapitalizedf=company\_df\_sample.loc[company\_df\_sample['capitalized']==False]

print(notcapitalizedf)

company names capitalized

227 the Human Rights Foundation False

507 eBay False

534 20th Century Fox False

1014 twenty-something False

1049 interim False

1052 iSectors LLC False

3985 2100 Xenon False

4106 eBay False

companysamplelist=company\_df\_sample['company names'].tolist()

df\_clean\_company=df\_clean\_copy.copy()

X\_train\_company, X\_test\_company= train\_test\_split(df\_clean\_company, test\_size=0.4)

print (X\_train\_company.shape, X\_test\_company.shape)

(405349, 2) (270233, 2)

whole\_feature\_filter\_copy.head(10)

Out[27]:

index word string sentence\_index y contain\_company \

0 3 Both Silver 3 0 0

1 8 Cumulative Travel 8 0 0

2 10 Pomona College 10 0 0

3 12 Sergeant Wardaddy 12 0 0

4 16 Minsky Moment 16 0 0

5 22 Mitsumi Electric Co Ltddown 22 0 0

6 30 The Scotch 30 0 0

7 32 Virginia Realtor 32 0 0

8 42 US Bancorp 42 1 0

9 48 Lorenzo Cremonesi 48 0 0

contain\_keyword first\_word\_index number\_of\_words num\_of\_letters

0 0 18 2 10

1 0 0 2 16

2 0 17 2 13

3 0 20 2 16

4 0 24 2 12

5 1 20 4 24

6 0 0 2 9

7 0 2 2 15

8 0 14 2 9

9 0 19 2 16

resultlen\_cop=whole\_feature\_filter\_copy['y'].value\_counts().to\_frame()

print (resultlen\_cop)

posnum\_cop=int(resultlen\_cop.iloc[1])

negnum\_cop=int(resultlen\_cop.iloc[0])

cleaned\_pos\_train\_sample\_cop=whole\_feature\_filter\_copy.loc[whole\_feature\_matrix['y']==1]

cleaned\_neg\_train\_sample\_cop=whole\_feature\_filter\_copy.loc[whole\_feature\_matrix['y']==0]

if posnum\_cop<negnum\_cop:

neg\_new\_size\_2=posnum\_cop

negative\_updated\_sample\_cop=cleaned\_neg\_train\_sample\_cop.iloc[:neg\_new\_size\_2]

train\_shrink\_copmany=pd.concat([cleaned\_pos\_train\_sample\_cop,negative\_updated\_sample\_cop])

train\_shrink\_company\_copy=train\_shrink\_copmany.copy()

y

0 62910

1 1618

train\_shrink\_company\_copy.shape

Out[31]: (3017, 9)

rf\_pred\_test\_comp=rf\_fit\_comp.predict(scaledx\_comptest)

rf\_score=rf\_fit\_comp.score(scaledx\_comptest,Y\_test\_comp)

rf\_metrics=precision\_recall\_fscore\_support(Y\_test\_comp,rf\_pred\_test\_comp,average='macro')

Random Forest for companies: accuracy score is 0.9690986858418051

Random Forest for companies: precision is 0.652647600375614, recall is 0.6023880721944882, and f1 score is 0.6220283068146786.