Despite very high cross validation results, for some reason, the outputted files have an incredibly large number of false positives. For this reason, I only submitted the output for the first document and I will spend a majority of the report explaining my methodology (I believe there must have been some sort of indexing error that would account for this because I believe I have sound methodology).

I will then explain future improvements that may solve this problem, but could not be implemented because of time frame issues by the time I realize my model has too many false positives.

**Metholology/code structure:**

First, I loaded the labelled positive files and parsed them for understanding. I broke down the number of words nltk would recognize in each of the samples and outputted the results below:







The purpose of generating histograms of word length is to determine which length grams I will need to build models to identify. Since a vast majority of samples are 4 words or under, I will only build models to identify unigrams, bigrams, trigrams, and 4grams.

Additionally, I wanted to search if there are any trailing words of the samples that repeat a large number of times. Repeated words could be used to identify the word type.

Top words at the end of each word type:

[('Corporation', 758), ('Corp', 639), ('Inc', 491), ('Group', 398), ('Ltd', 82), ('Capital', 40), ('Management', 37), ('Co', 28), ('.', 21), ('Bank', 20), ('Apple', 15), ('America', 15), ('Sachs', 14), ('Partners', 14), ('Advisors', 14), ('Company', 14), ('Holdings', 14), ('Facebook', 13), ('Microsoft', 13), ('Stanley', 12)]

[('Smith', 6), ('Miller', 6), ('Johnson', 5), ('Brown', 5), ('Rometty', 5), ('Cook', 4), ('Goldberg', 4), ('Handler', 4), ('Doctoroff', 3), ('Benioff', 3), ('Thompson', 3), ('Ullman', 3), ('Fuld', 3), ('Lewis', 3), ('Ackman', 3), ('Corbat', 3), ('Holmes', 3), ('Lynch', 3), ('Simon', 3), ('Williams', 3)]

[('%', 2154), ('percent', 1417), ("''", 100), ('percentage', 23), ('4', 7), ('point', 6), ('5', 6), ('0.2', 5), ('1.5', 5), ('3', 5), ('2', 5), ('6', 5), ('0.7', 5), ('2.4', 5), ('2.3', 5), ('1.3', 5), ('1', 5), ('0.1', 4), ('2.5', 4), ('13', 4)]

These repeated words show that there are commonly repeated endings for percents and companies that should be used as features.

The model samples were created by collecting features for each word in the ngram sample and then adding features for the 4 trailing and preceding words. This means that a unigram, bigram and trigram will all have a different number of observations per sample (because of the additional words in the observation), so different models must be created for each word size.

**The features collected for each word are as follows:**

**-Length of the word**

**-is the word in all caps?**

**-is the word all lowercase?**

**-is the word capitalized?**

**-does the word contain % sign**

**-does the word contain the word percent**

**-does the word contain the word point**

**-is the word all numeric?**

**-is the word all ascii characters?**

**-does the word contain a –**

**-does the word contain a .**

**-does the word contain one of the common trailing words for a company?**

**-pos tag**

**Each sample includes these features for each word in the ngram, in addition to the 4 preceding and trailing words**

**After testing random forests, svms, and gradient boosted trees, I found that gradient boosted trees had the higher cross validation accuracy. Accuracy is a valid measure for this classification because I took many steps to ensure an even dataset between positive and negative samples**

I then went through and collected features for each of the words in the labelled dataset according to their appropriate word length. These features include features of the trailing and preceding words as well. Now I needed to generate negative samples. I did this by randomly selecting words from the documents to sample according to the distribution of word lengths seen in labelled document. Random selection should be a valid method for negative sample collection because the word types in question make up a small portion of the overall words. Additionally, the CEO name identifier should use other names in the text as negative samples. The intent was to use the ceo names with random samples to build a separate name classifier. This name classifier could then be used to generate all names from the document and I would then be able to classify these names as CEO or non=ceo. This classification was to be done by first checking the list of labelled CEOs, and then doing a quick websearch of the name and scraping the top result for executive related keywords.

Unfortunately, the size of the positive dataset is not well represented by the keys because many of the keys are used in multiple articles. This resulted in a much smaller negative sample set than positive, which I accounted for by building an ensemble model.

To create the ensemble model, I separated the dictionaries of features into smaller dictionaries that would match the negative feature dict size. This created 3 dicts for the ceo names, 10 for the company names, and 26 for the percent dictionary, which must have had a large number of repetitions between articles.

I then saved the dictionaries to pickle for easy access later because this data took a long time to collect. Unfortunately, the percent data would not save to pickle due to some OS error, so I was not able to save this data and needed to continuously recollect it (which happened frequently because the percent data liked to crash the kernel as well)

I reorganized the dicts of features to be a simple list of observation features that could be indexed by length of word (1,2,3, or 4) so that the samples in each category have the same number of columns.

I then generated a loop that would train models for each of the smaller positive sample dictionaries. Since these smaller positive sample dictionaries were created to be of equal size to the negative dict, there will be an equal number of positive and negative observations. To ensure that this is true for each length word, I built in a mechanism to resample negative samples if there are too few (this should prevent too many false positives). Since some dictionaries might have few observations for a specific word length, I only generated models if there are over 100 positive and negative observations (each). Since the feature collection results in hundreds of binary features, I also scaled the data and did pca with 10 components.

I collected cross validated accuracy scores for each of the models created and averaged them for each of the classification types. Accuracy is a reasonable metric for this classification because I made sure that the dataset is balanced between positive and negative samples. This method of model generation resulted in the following average cross validated accuracy rates:

**cross validation accuracy of ceo names trainer is 0.9771019911724171**

**cross validation accuracy of company trainer is 0.9527491909050892**

I then use these models for an ensemble prediction for each word. This is done by classifying a word as positive if a majority of the smaller models classify the sample as positive.

Predictions are then aggregated across the word lengths and outputted to a csv file

**Improvements**

I believe a large part of the massive numbers of false positive is from insufficient negative samples. Instead of sampling negatives to make the number and distribution of keys, negative samples should be sampled to make the number and distribution of the actual number of samples taken across all words (this accounts for repeated words). Unfortunately, the sampling of negative features took a very long time to get evenly distributed across article types, so I was not able to repeat the process once I learned this might be a concern. Instead, I built secondary measures to protected against imbalanced classes, but this may not have been sufficient.

The data might have also been easier to process if there had been more data cleaning done. I did not want to do this initially because I was worried it could change the subtleties of capitalization in proper nouns, but this may have been useful in eliminating stop words and extraneous symbols that occasionally appear in the articles and throw off the pos tagger.