# STOR 565 Final Project

The Machine Learners: Anna Hattle, Raphael Kim, Joseph Son, Jade Wu

## Summary

We wanted to assess the predictive ability of user reviews for major tech companies in the United States: Apple, Amazon, Google, Facebook, Microsoft, and Netflix. More specifically, we wanted to predict whether the employee was former or current, the rating the employee gave, and what company the employee worked at.

We looked at reviews written by the employees of these companies on GlassDoor.com. The reviews were formatted as text which can not be directly inputted into our statistical models. In order to numerically represent the text we primarily investigated two Natural Language Processing techniques: Bag of Words (BOW) and Word2Vec/Doc2Vec.

Using our various data representation models, we applied what we felt were the most appropriate classification techniques to try to answer our initial questions. We used logistic regression as a baseline for all of the questions we were answering (with a penalty since our predictor set is very large). In addition to that model we used Multinomial Naive Bayes for our Bag of Words predictions and random forests for our word2vec models.

Overall, our user review data encoded by BOW or word2vec did not offer much predictive power for predicting former and current. This makes sense since what an employee thinks of a company does not directly relate to former or current. For example, an employee could be former for a number of reasons other than not liking the company.

When predicting ratings, we found the user review data offered notable predictive power. To our surprise, BOW did better than our word2vec embedding. This was most likely because our review data was overall very similar, and the max wise component method removed a lot of the small nuances in the data.

Finally when predicting company, we found that the BOW method performed the best while the Word2Vec embeddings produced nearly indistinguishable boundaries between the different companies.

## Introduction:

Our dataset includes over 67K reviews for Google, Amazon, Facebook, Apple, and Microsoft. These reviews were retrieved from the website GlassDoor, where current and former employees anonymously review companies and their management, and was compiled on Kaggle. The dataset contains the following features for each review:

- Text data: Summary of Employee Review, pros, cons, advice to management
- Categorical variables: Overall Rating (1-5), work/life balance (1-5), Culture and Values (1-5), Comp & Benefits (1-5), Senior Management (1-5), Job Title (current or former employee)
- Continuous variables: Helpful Review Count

As prospective employees of these companies, our main motivation for picking this dataset was to better understand what employee opinion was on each company.

In addition, we were interested in learning more about natural language processing (NLP) methods, and how powerful of a tool it can be in prediction. Using NLP methods, we tried predicting what company the employee worked at, whether the employee rated the company as good or bad, and whether the employee was former or current.

Since we are primarily interested in the text data and what information it can offer us, the primary challenge in our investigation was finding the best way to encode our natural language data while still capturing the meaning and semantics of the text.

**Methods:**

**Data Pre-processing**

Before performing any analysis, we first preprocessed the data. For each review, we applied the following preprocessing techniques:

1. Converted all words to lowercase
2. replaced "/" with " "
3. Remove stopwords (ie, "the", "he", "a", etc.)
4. Included company names in list of stopwords to be removed
5. Performed lemmatization
6. Tokenized words

Lemmatization is the process of converting different forms of words to the same stem. For example, "were", "being", and "are" would all be converted into "be." Tokenization involves taking a text or set of text and breaking it up into its individual words.

**NLP Techniques:**

Once we processed the data, we moved into using two popular NLP methods: bag of words and word2vec. Bag of words represents text by encoding (1) a vocabulary of known words and (2) a count of of the presence of these known words for each data sample. Word2Vec is an alternative, often more robust, framework for vectorizing text. Once a dimension of interest ($S$) is specified, word2vec finds a real valued S-length vector representation of the each word in the input vocabulary. By the nature of the algorithm, it embeds semantically similar words near one another in this vector space of size $S$. ****see Appendix for a more thorough explanation of Bag of Words or Word2Vec.*

Using these methods, we converted this preprocessed data into encodings or embeddings.

**Classification Algorithms:**
With our encodings from bag of words, we tried logistic regression and multinomial naive bayes to predict reviews. Logistic regression acted as a baseline for evaluating model performance. Multinomial naive bayes made sense because of the nature of bag of words only utilizing frequency counts.

With our word embeddings for each sample by word2vec, we then predict outcomes of interest with logistic regression with the ridge penalty and random forests. Logistic regression with ridge penalty was used as it is a typical base line. We decided to penalize to guard against overfitting. We decided to use the ridge penalty over the lasso penalty since a sparse model does not really make sense when using word embeddings as features. Random forests were also used since they are good for protecting against overfitting, and industry seems to identify this as one of the top performing machine learning algorithms. Hyperparameters for each method were selected by 5-fold cross validation from a predetermined set.

We did consider LDA and QDA as possible classification schemes, but after performing the Kolmogorov-Smirnov test on the features in the vectors, we found that the features were not normally distributed. We concluded that LDA/QDA would not be robust models.

Again, the outcomes of interest were binarized reviews (good is 1 when review >= 3, and 0 otherwise), the company (of a possible 6), and whether the employee was former or current.

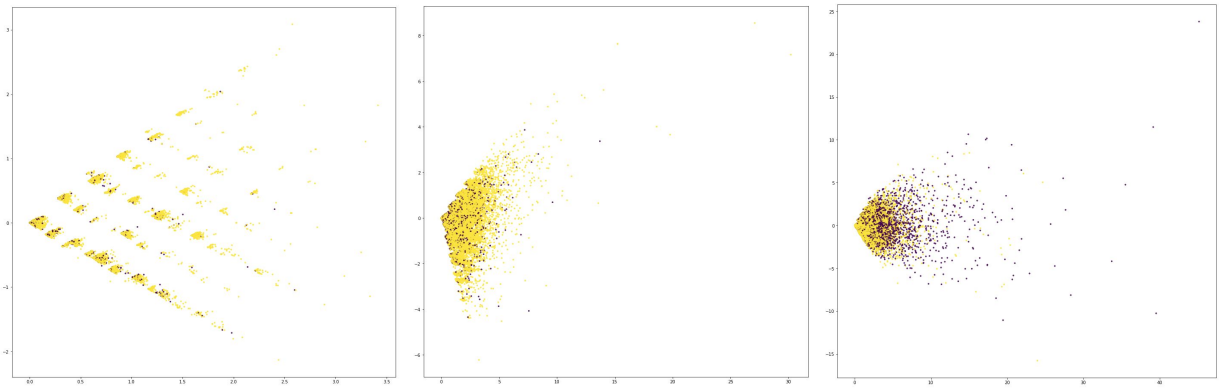
## Analysis and Results:

**Predicting ratings:**
We performed some preliminary analysis on the robustness of this model by predicting the overall rating of a company based on the company, pros, and cons tokenized text. A “good” review was considered to be 4 or above, and a “bad” review 3 or lower.

| | **tokenized_summary** | | **tokenized_pros** | | **tokenized_cons** | |
|---|---|---|---|---|---|---|
| | accuracy | precision | accuracy | precision | accuracy | precision |
| **Logistic** | 0.722 | 0.725 | 0.669 | 0.696 | 0.716 | 0.724 |
| **Multinomial Naive Bayes** | 0.756 | 0.747 | 0.725 | 0.710 | 0.746 | 0.734 |

As we can see from the results above, the multinomial naive bayes model outperformed logistic regression. Since the bag of words does not take into account the order of words, the assumption that the conditional probabilities of the features must be independent of each other is met. It is also worth considering which of the three text corpuses, summary, pros, or cons, is best at predicting overall ratings. To represent the embeddings in interpretable plots, we performed dimension reduction through PCA to more clearly see the separation between classes.

**Plot 1.** PCA plots of bag of words vector. From left to right: summary text, pros text, and cons text. (yellow = good, purple = bad)



As shown in the plots shown above, the summary and cons plot show a much clearer separation between the two classes than pros. This is confirmed in our accuracy table, where both logistic and naive bayes perform better by predicting overall reviews using summary and cons. Summary and cons have the same accuracy, but cons has a slightly higher precision rate.

**Predicting former/current:**

We first tried to predict former or current employee status with bag of words.

| Method | Accuracy | Precision |
|---|---|---|
| Log Reg. Ridge | .583 | .595 |
| MN Naive Bayes | .635 | .603 |

We then tried predicting whether the employee was a former or current employee. We first tried predicting with the word2vec embeddings.

| Method | Accuracy | Precision |
|---|---|---|
| Log Reg. Ridge | .630 | .500 |
| Random Forest | .630 | .539 |

We then tried adding the other variables like 'work-balance-stars', 'culture-values-stars', 'career-opportunities-stars', 'comp-benefit-stars', and 'senior-management-stars'. This marginally improved prediction.

| Method | Accuracy | Precision |
|---|---|---|
| Log Reg. Ridge | .632 | .615 |

| | | |
|---|---|---|
| Random Forest | .632 | .611 |

Overall these results make sense because more often than not, information from a review or what a reviewer thought of the company does not seemly strongly correlated with whether the employee is former or current. An employee could be former or current for a number of reasons, not just because they did not like the company.

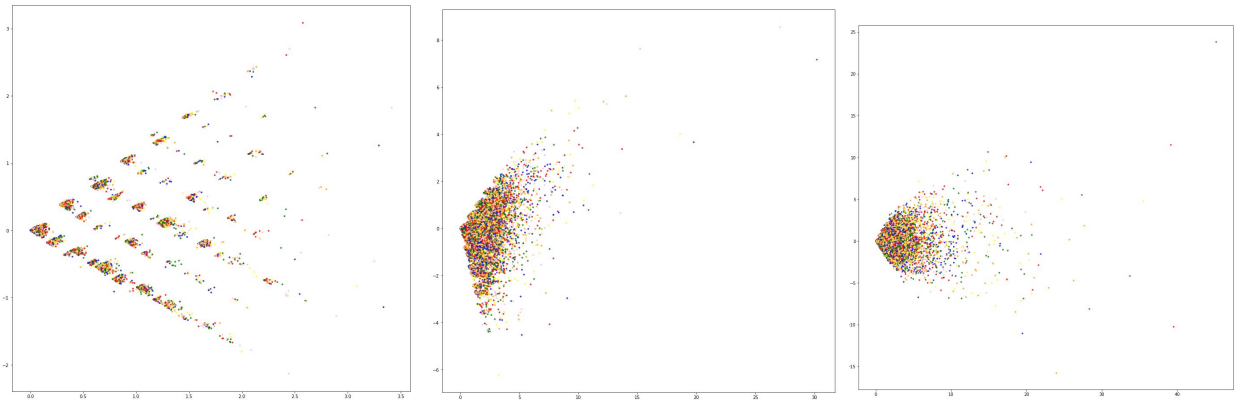
### Predicting Companies

How does the bag of words model fare when predicting the company? With the bag of words model, it is safe to assume, that multinomial bayes performs better than a logistics model; thus, for predicting company, we will only use the multinomial bayes classification scheme. The results are shown in the table below.

| | **tokenized_summary** | | **tokenized_pros** | | **tokenized_cons** | |
|---|---|---|---|---|---|---|
| | accuracy | precision | accuracy | precision | accuracy | precision |
| **Multinomial Bayes** | 0.526 | 0.524 | 0.596 | 0.590 | 0.595 | 0.586 |

Below we show the PCA plots for the bag of words predictors color coded by company.

**Plot 2.** From left to right: summary text, pros text, and cons text. (key: orange = amazon, blue = google, pink = facebook, red = microsoft, yellow = netflix, green = apple)



The naive bayes model remains a suitable scheme for bag of words, since the lack of consideration of context matches the assumption of independent conditional probabilities in naive bayes. The accuracies and precision values for predicting companies is significantly lower than the accuracies and precisions for predicting overall ratings. Additionally, based on the PCA plots, there is not a clear separation between the six classes. The question follows, why does our bag of words/multinomial bayes model perform better at classifying overall ratings than companies?

This becomes clear when looking at the most frequently used words in our bag. The most popular words in reviews included “good,” “great,” “amazing,” and other descriptive adjectives that clearly correspond to a positive or negative review/rating. At the same time, when you separate the data by company, the most popular words are still the same sort of generic adjectives described above. When generic words like “good” appear frequently in reviews, it is more difficult to discern which company is being described in those reviews, and much easier to discern the sentiment.

Although intuitive and simple to implement, this method has some drawbacks. The model is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not *where* in the document. Thus, any semantic or contextual meaning of a word is lost. To capture some of the semantic meaning of the words, we decided to consider another framework for analysis, Word2Vec.

## GoogleNews Data Word2Vec Embeddings

Initially, the easiest way to add some of the benefits of Word2Vec to our model were to use word embeddings from a model pre-trained by the creators of Word2Vec on a large set of GoogleNews articles. We then used those embeddings to represent the words in our input corpus.

We then performed logistic regression with a penalty term and got the following results:

| | **Predicting Ratings** | | **Predicting Companies** | |
|---|---|---|---|---|
| | accuracy | precision | accuracy | precision |
| **Logistic Reg w/penalty** | 0.601 | 0.642 | 0.242 | 0.389 |

For both prediction options, the model performed slightly better than guessing. Since these results did not look particularly promising, before moving further with exploring the predictive capacity of this model, we made a decision to train our own word2vec model on the Glassdoor reviews. We chose to do this because word2vec encodes information about words in the input text based on the context in which it is found. The context of company reviews is extremely specific, so we hypothesized that we could get better results from a model trained for this.

We can see how the two models encode entirely different information by examining the word “culture” and the words with the highest cosine similarity to it under each model. In our Glassdoor word2vec model (hereafter referred to as our word2vec model), “culture” is close to these words: *colleague, atmosphere, peer, organization, coworker, workplace, vision, value, ethic, strong*. In the GoogleNews model, “culture” is close to: *insane, cultural, Libyan_reformer_Fathi, Universit_de_Montreal, Fikr, Arabes, Jaysh, Ihya, Medio_Oriente.* It is clear that our word2vec model encodes information about words specific to employment that are much more useful in our situation.

## Glassdoor Data Word2Vec Embeddings

### Predicting Ratings

| | tokenized_summary | |
|---|---|---|
| | accuracy | precision |
| **Logistic Regression** | 0.7017621797719532 | 0.7116762989986553 |
| **Random Forests** | 0.7294535761883607 | 0.7263685222577734 |

We find that reviews are able to be predicted decently with our word embeddings, but it did not perform as well as bag of words. This result was somewhat unexpected to us. We thought that a representation of words that encoded their relationship to each other and their meaning would allow us to do a better job at predicting the class of good and bad ratings, but there are a few possible reasons why this was not the case. One reason could be that our max-wise component scheme for combining word embeddings is flawed and contorts the data so it has less predictive value. Another possible reason could be that context is not as important as we thought to this classification. Regardless, this model still performs decently well.

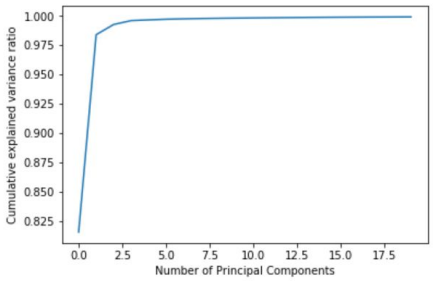
### Predicting Companies

Now we will attempt to predict the company based off the word embeddings for the user's review summary, pros, cons, and advice using our Word2Vec embeddings. A preliminary glance at the word embeddings for a sample of our dataset show that classifying companies may be troublesome.

| wordEmbeddingtokenized_advice96 | wordEmbeddingtokenized_advice97 | wordEmbeddingtokenized_advice98 | wordEmbeddingtokenized_advice99 | company |
|---|---|---|---|---|
| 3.35185 | 3.95689 | 3.133237 | 2.565454 | amazon |
| 3.35185 | 3.95689 | 3.133237 | 2.565454 | amazon |
| 3.35185 | 3.95689 | 3.133237 | 2.565454 | google |
| 3.35185 | 3.95689 | 3.133237 | 2.565454 | apple |

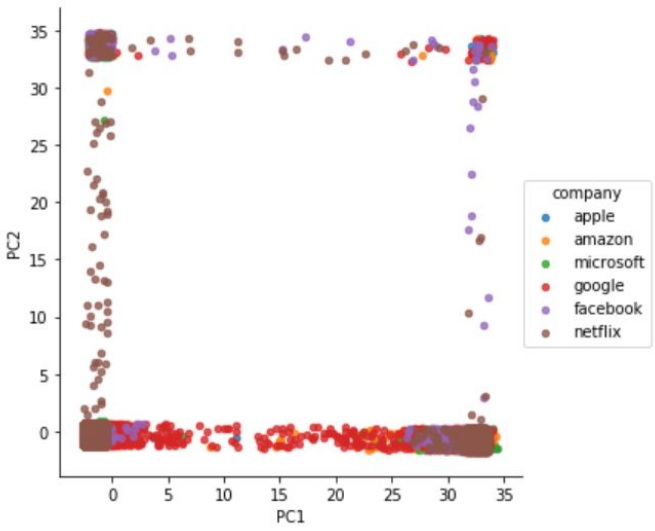
We also notice that there is a severe imbalance of classes. When attempting to predict the company, when there is this class balance, we found that the classifier always predicts Amazon since it is the largest class (Amazon: 26430 v. Netflix: 810) by far. We can either balance the classes by undersampling or oversampling. We chose to oversample instead of undersample in order to avoid losing over 40,000 observations in our training set.

We divided full dataset into training and test sets. Each set has class distributions proportional to the original. Used SMOTE on the training set to interpolate observations for the minority classes. This gave us 21144 observations for each class in the training set. We then fit PCA on the non-standardized oversampled training set. 2 components explained 95% of the variance. We chose




not to standardize the data, because Word2Vec produced each word embedding on the same range.

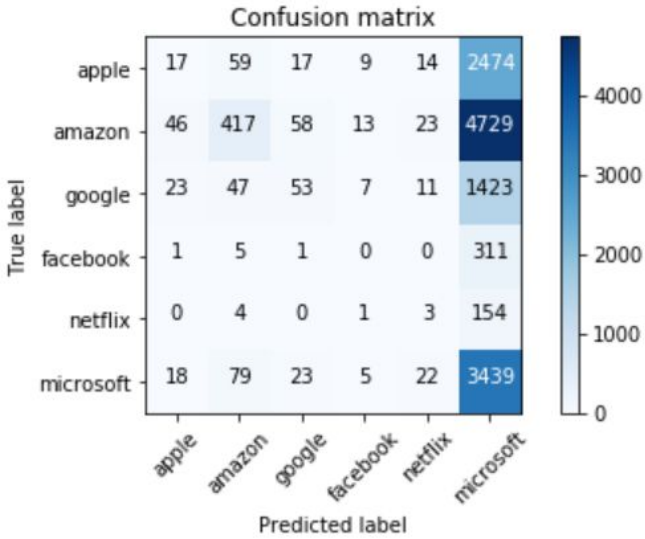
*Percent Variance Explained by Principal Components*




*PC1 and PC2 scores of our Word2Vec embeddings labeled by company*

Next, we applied the PCA model to transform the features of the training and test observations to their respect PC-1 and 2 scores. We note here that a lot of the transformed word embeddings are the same. When we look back at the original embeddings, we also see that a lot of the observations are the same across companies.

Based off the principal components projection, it looks like there exists nonlinear boundaries between pc scores of the observations of different classes. Therefore, we chose to assess the performance of a nonlinear classifier such as Random Forest.




The Random Forest Classifier almost always predicted Microsoft, giving 29% accuracy. Because of this poor performance and the observation that many observations overlap, it seems that our Word2Vec embeddings are not very informative in predicting the company an Employee review is associated.

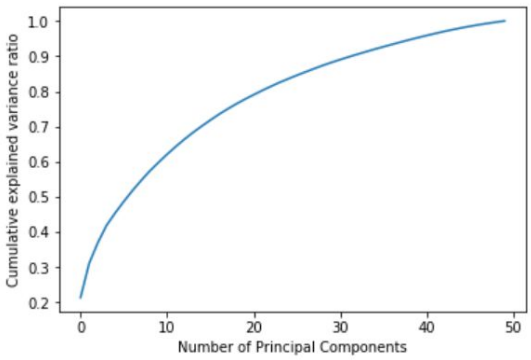
## Doc2Vec

Regarding embeddings, we also tried using doc2vec. We guess that doc2vec could do better than word2vec because each natural language col: pros, cons, advice-to-management, summary tends to have similar words in each. By using word2vec with schemes like average/max wise component, we lose a lot of the small nuances to the word information. Doing doc2vec may preserve these differences better.

### Predicting Companies

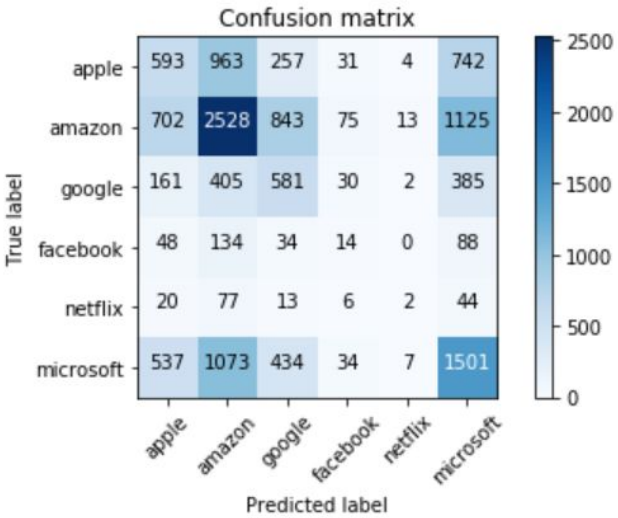
We can immediately see that the embeddings are more varied than those constructed from Word2Vec.

| docEmbed9 | ... | docEmbed41 | docEmbed42 | docEmbed43 | docEmbed44 | docEmbed45 | docEmbed46 | docEmbed47 | docEmbed48 | docEmbed49 | company |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.015113 | ... | -0.021760 | 0.012323 | 0.010084 | 0.017389 | 0.001595 | -0.003047 | -0.021168 | 0.017058 | 0.007600 | google |
| 0.028908 | ... | -0.093214 | 0.007856 | 0.006215 | 0.004991 | 0.011382 | -0.057320 | -0.083939 | -0.029888 | -0.009353 | microsoft |
| 0.037896 | ... | 0.001112 | -0.032382 | -0.001314 | 0.014601 | 0.054912 | 0.035140 | -0.124574 | -0.041021 | 0.027987 | google |
| -0.081151 | ... | 0.027262 | 0.046853 | -0.000891 | 0.017433 | -0.121763 | 0.009998 | -0.105419 | 0.019758 | -0.025534 | microsoft |
| -0.091945 | ... | 0.056995 | -0.136366 | -0.061134 | 0.010618 | 0.016770 | 0.027838 | 0.058722 | -0.155882 | -0.012531 | amazon |
| -0.066547 | ... | 0.018996 | 0.003877 | 0.038321 | 0.033050 | 0.006022 | -0.052027 | 0.027167 | -0.022724 | -0.004574 | microsoft |
| 0.021977 | ... | -0.041790 | -0.036135 | -0.009828 | 0.029498 | -0.002676 | 0.017794 | -0.093245 | 0.092141 | 0.016433 | amazon |
| -0.005247 | ... | -0.005591 | -0.026288 | 0.023169 | 0.018288 | -0.017974 | -0.054278 | 0.024781 | -0.022911 | 0.047491 | amazon |




*Percent Variance Explained by Principal Components*

40 principal components explain 95% of the Doc2Vec variance. Because the first two components only explain around 30% of the variance of data, we will not pursue visualizing it.




The following is the result of predicting company using Doc2Vec with a Random Forest Classifier with 100 estimators and max depth 20. This performed much better than on Word2Vec as we expected.

| | precision | recall | f1-score |
|---|---|---|---|
| amazon | 0.48 | 0.49 | 0.48 |
| apple | 0.23 | 0.29 | 0.25 |
| facebook | 0.04 | 0.07 | 0.06 |
| google | 0.37 | 0.27 | 0.31 |
| microsoft | 0.42 | 0.39 | 0.40 |
| netflix | 0.01 | 0.07 | 0.02 |

## Conclusions

Using the Word2Vec embeddings trained on GoogleNews data to predict the company, Logistic Regression produced 24.2% accuracy. With the first 2 principal components of Word2Vec embeddings trained on our data, the Random Forest classifier produced a 29% accuracy. However, this accuracy was only mainly obtained by only predicting Microsoft. On the other hand, the first 40 principal components of Doc2Vec embeddings produced a 39% accuracy. Even though the accuracy is still quite low, we see a large improvement in correctly guessing each company. Bag of Words performed the best when only using the tokenized pros data to produce around 60% accuracy and precision.

Based off the visualizations and classification performance metrics, it seems that the text to numerical vectorization methods created ambiguous and overlapping boundaries for the company classes. However, Bag of Words Multinomial Naive Bayes seemed to perform the best. This leads us to believe that the context which words occur may not actually be that significant in distinguishing reviews for different companies. Rather, the occurrences of certain words for each company in their reviews are most informative. Even still, merely considering the occurrences of certain words is inherently ambiguous and nontrivial. In our exploratory data analysis, the most frequently occuring words for each company were often the same.

In conclusion, user reviews will not be a distinguishable metric for these “top” tech companies. We probably should not only use GlassDoor reviews to construct our perception of each company as each company may be more similar than they are different. In other words, the differences in the experiences one has at each company may be more nuanced than what we can see by other people’s opinions and experiences.

Regarding prediction of former and current, data encoded by BOW or word2vec did not offer much predictive power. This makes sense since what an employee thinks of a company does not directly relate to former or current. For example, an employee could be former for a number of reasons other than not liking the company.

When predicting ratings, we found the user review data offered notable predictive power. To our surprise, BOW did better than our word2vec embedding. This was most likely because our review data was overall very similar, and the max wise component method removed a lot of the small nuances in the data.

## Future Directions:

Initial word frequency analysis by company did not show much difference between words used by employees in each company. However, like bag of words, this word frequency analysis, eliminates available semantic meaning. To better understand various parts of the company we were interested in like worklife, opportunity, and culture, we could build a word2vec model for data subsetted by company and for example, analyze what words are most similar to worklife in the context of amazon employees and compare that to words semantically similar to worklife in the context of google employees.

As shown by our data visualization on PCA or the poor evaluation metrics, there is not very clear separability in predicting companies or former/current. Thus it seems worthwhile to explore

prediction with neural networks (as by the universal approximation theorem, with enough data, it should be able to produce incredibly accurate decision boundaries.

We were also hoping to use a blackbox explainer like LIME to increase the interpretability of our word2vec model. Lime provides insight on how the model makes decisions by perturbing inputs and returning the words that change the predictions more significantly. This would have allowed us to answer questions like which words in a review seem most likely to associate it with Amazon.

Also, since our Doc2Vec model did a decent job at correctly classifying Microsoft and Amazon reviews, it would be worthwhile to investigate boosting to see if that would increase classification accuracy for the other companies.

Additionally, because of time constraints and because we were prioritizing investigating as many models as possible for this report, we did not do a thorough job of understanding the accuracy/error in a more general sense for each model. In an ideal world, we would have trained and evaluated models hundreds of times to obtain a distribution of errors instead of just evaluating the error once.

# Appendix

**Bag of Words**

The first framework we considered for natural language processing and analysis was bag of words. A representation of the text, it involves (1) a vocabulary of known words and (2) a count of of the presence of these known words.


In the simple example shown above, we can see that each sentence is represented by an n-dimensional vector, where n is the size of the dictionary and p in (1...n) is the number of times the p-th word in the dictionary appears in the sentence. This can be extended to our dataset of company reviews, where each sentence is replaced with a review.

**Word2Vec**

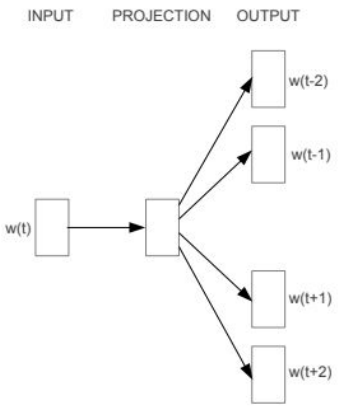
Word2Vec is an alternative, often more robust, framework for vectorizing text. Once a dimension of interest (*S*) is specified, word2vec works to find a real valued vector representation of the word that is of length *S*. By the nature of the algorithm, it embeds semantically similar words near one another in this vector space of size *S*.

Word2vec can refer to two types: skip gram (SG), continuous bag of words (CBOW). Both use a 2 layer neural network to either

1. Take a middle word and predict the context of that word (SG)
2. Take the context of a word and predict the middle word (CBOW)

The context of the word refers to the words around the middle word and is often denoted by *C*. With a context size of *C*, that means we look at the *C* words before and after the middle word.

We used skip gram since we want to better understand




Skip-gram

Src: https://arxiv.org/pdf/1301.3781.pdf 'Efficient Estimation of Word Representations in Vector Space'

More specifically, the middle word $w$ is inputted as a one hot encoding vector of size |V|, the size of the vocabulary; we denote the input by *I*. Then the first set of weights $W \in R^{|V| x S}$ is applied to our input vector to get the hidden layer of size S. From there, $W' \in R^{S x |V|}$ is applied to the hidden layer 2*C* times to obtain 2*C* vectors of size |V| that denote scores for each word occurring in

each context. From there, the softmax function is applied to each of the 2*C* vectors to get the relevant probabilities each word occurs in a given context.

So our equations are as follows:

$h = W^t I$

$o_i = W'^t h, \ \forall i \ \in \{1, ..., 2c\}$

$y_i^{'} = softmax(o), \forall i \ \in \{1, ..., 2c\}$

Where $y_i'$ is a given vector of size v that has probabilities that each word occurs in context i.

The way this works to predict the context is shown by the loss function $L$.

We seek to optimize the probability of seeing the context we see or the $P(W_C|w)$, where $W_C = \ w_C, ..., w_1, w, w_{-1}, ..., w_{-C}$ or the 2*C* words around word $w$.

$L = -log(P(W_C|w)) = -log(\prod_i P(w_i|w)) = -log(\prod_i \frac{exp(o_{j,i}^*)}{\Sigma_j exp(o_{j,i})})$, where $o_{i,j}^*$ is the word we actually see in context i in the training corpus; the training corpus is our set of reviews, each considered as separate sentences.

Following backpropagation, we can then perform gradient descent to determine an optimum value for these weights.

The word embeddings are then chosen as either $W$, $W'$ or $\frac{W+W'}{2}$.

So in the end, we have a set of embeddings $E \in R^{|V|xS}$ where the $i^{th}$ row is the embedding for word i in the vocabulary. To represent the sentence in the data set with our embeddings, we can either take the average of all the word embeddings for each word that occurs in the sentence or take a component wise max of all the word embeddings that occur in the sentence. We use the component wise max or max pooling as described by (https://www.aclweb.org/anthology/C18-1154; https://arxiv.org/pdf/1805.09843.pdf) since it was shown to perform decently in these sorts of applications, and by nature of our reviews, there were many similar words so we wanted to extract extreme or high signal features.

For concreteness, a short example is shown. Suppose we have a vocabulary of size 2, where V={'food','good'}, and we have the following word embeddings:

- 'food': [0.1, 0.23]
- 'good': [0.05, 0.15]

Then when trying to represent sample i: 'food is good', we apply preprocessing to obtain {'food', 'good'}. By our component-wise max of word embeddings to approximate sentences:

Embedding i = [max(0.1, 0.05), max(0.23, 0.15)] = [0.1, 0.15].

When using these embeddings in the algorithm, we used embeddings pre-trained on a google news corpus and our own embeddings trained specifically on our training text.

**Doc2vec**

Doc2vec is simple extension of word2vec. Doc2vec works by training a vector for each sentence/document in combination with the regular word2vec scheme; more specifically, the document ID acts as an input for the corpus during the regular word2vec procedure. This training process is repeated for each sample corpus so it learns an embedding for each sentence or document.