DS-GA 1003 Project Final Report

Co-occurrence Embedding of Named Entities

Zhiming Guo
zg758@nyu.edu

Qintai Liu
ql819@nyu.edu

Yicheng Pu
yp653@nyu.edu

Advisor: Kurt Miller

May 11, 2018

Contents

List of Tables

List of Figures

1 Introduction

This project was inspired by the GloVe[9] model used in Natural Language Processing(NLP). GloVe trains word embeddings using a word co-occurrence matrix. The success of such co-occurrence embedding in NLP field motivates us to explore its power in some other tasks.

The specific goal of this project is to identify which industry does an entity belong to using its embedding vector trained from a co-occurrence matrix, such an algorithm would help to improve Named Entity Recognition ideally. The designed pipeline would start from extracting entities from news articles with Named Entity Recognition (NER), building an entity co-occurrence matrix, training entity embeddings with GloVe and then train classifiers with embedding vectors.

During the project, we have also found a lot of valuable knowledge about co-occurrence embedding and some of them could be used beyond this project, which we will talk more about in the next few sections.

2 Data

2.1 Data Source

There are two collections of data we would use. The first data collection (data1), which is a collection of news and wikipedia articles, would be used to train our entity embedding. The data was obtained from 2010's Wikipedia (6GB), scrapped from 2017's Prnewswire(431MB) and 2015-2017's Yahoo News(189MB), each data source would individually generate a word embedding with 150 embedding size.

The second data collection (data2), crawled from Nasdaq, Forbes and U.S. Securities and Exchange Commission (SEC) website is in a dataframe format including people/company names and their industry categories. People and company are separated into two datasets for each source. There are a total number of 57304 company names and 318027 people names from SEC, 4763 company names, and 1760 people names from Forbes. Data2 is prepared to be used in training classifiers.

2.2 Data Issues and Cleaning

Before training classifiers, the first step is to join people datasets and company datasets from data2 together. Then, the industry category inconsistency issue occurred. There are 12 industry categories in Nasdaq dataset, which are: basic industries, capital goods, consumer durables, consumer non-durables, consumer services, energy, finance, healthcare, miscellaneous, public utilities, technology, and transportation. However, there are 20 industry categories in Forbes and over 300 industry categories in SEC due to different definition of segment.

One way to solve this issue is to use the smallest number of categories as standard. In Nasdaq, basic industries and energy are combined to basic industries. Capital goods, consumer durables, consumer non-durables, and miscellaneous are combined to miscellaneous. Also, new categories named “Media” and “Agriculture” are created. The new industry categories would be our basic reference. Then, Forbes and SEC categories can be manually mapped from 20 categories and 300 categories to this new industry categories.

The second step is to inner join data1 and data2. Each name in data2 is supposed to be exactly matched up the name in data1 and therefore the corresponding word vector would be the features of that name. However, the different format of names and category from source makes exact matching difficult. The issues includes that the name format style varies based on different source. For example, company x occurs in both data1 and data2. There are several cases of different format causing matching failure: X has suffix like “.inc” and “.llc” in data1 but doesn’t have suffix in data; x is all upper letters vs x is all lower letters; X includes detailed department name vs X only. For people, besides upper and lower letter issue, order can also be an issue. Some source places the last name in the first position followed by first name; some has middle name but some doesn’t have; some company is also misclassified into people dataset.

To solve the inconsistent format for company name, we keep the name format as easy as possible: all characters are transformed to lower letters and all punctuations are removed; The company names whose length are greater than 50 are also removed; companies’ suffix are removed by regular expression. To solve the inconsistent format for people name, all letters are transformed into lower case. All names with character length greater than 30 or the number of words greater than 3 are removed. Also, for three-word names, the middle word is removed. Then, the words in first position and second position are switched.

The last step is to remove repeated names. There are three reasons why there are repeated name

- We crawled data from different sources, it is possible to have replicated name.
- For people name, we removed middle name, which brought more repetitions.
- There are lots repetitions for popular names, especially in English.

2.3 Dataset Merging

After the above steps, we could match names from data1 and data2 and obtain name’s word representation as their features. For company, there are over 6000 names that can be exactly matched but 300 of them belong to multiple industries, these repetitions are removed randomly . However, for people, there are around 22000 matches with over 4000 repetitions. These repetitions bring lots of noise to data. For example, suppose there are 10 repetitions from 10 industry categories, in machine learning language, this means that there are 10 instances with exact same 150 features but each example belongs to different class. Later on, we decided not to train classifiers for people names due to the terrible data quality. We would train and test classifiers to predict industry categories for company only with around 6000 examples.

3 Named Entity Recognition

Named entity recognition seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, and locations. We apply NERClassifierCombiner[7] developed by Stanford University to obtain the names of persons and organizations within each article in our data source 1. For a concrete example,

```
>>> content = "Combine that with purchases by Warren Buffett, chairman and CEO of
    Berkshire Hathaway (BRKA), who has been gushing over the stock, and the combined
     purchases made up about 9% of Apple daily trading volume."
>>> ner.tag(content.split())

Output:
[('Combine', 'O'), ('that', 'O'), ('with', 'O'), ('purchases', 'O'), ('by', 'O'), ('
    Warren', 'PERSON'), ('Buffett,', 'PERSON'), ('chairman', 'O'), ('and', 'O'), ('
    CEO', 'O'), ('of', 'O'), ('Berkshire', 'ORGANIZATION'), ('Hathaway', '
    ORGANIZATION'), ('(BRKA),', 'O'), ('who', 'O'), ('has', 'O'), ('been', 'O'), ('
    gushing', 'O'), ('over', 'O'), ('the', 'O'), ('stock,', 'O'), ('and', 'O'), ('
    the', 'O'), ('combined', 'O'), ('purchases', 'O'), ('made', 'O'), ('up', 'O'), (
    'about', 'O'), ('9%', 'O'), ('of', 'O'), ('Apple', 'ORGANIZATION'), ('daily', 'O
    '), ('trading', 'O'), ('volume.', 'O')]
```

From the output, we can see that each word in the sentence is classified into either 'Person', 'ORGANIZATION', or 'O'. What we do next is combining neighbors whose category are the same. In the example, the word 'Warren' and the word 'Buffett' would be appended to form 'Warren Buffett'. Also the company 'Berkshire Hathaway' is obtained by the same way. To distinguish person and organization, the name of person is suffixed with '_1' and the name of organization is suffixed with '_2'. Therefore the sentence in the example is finally transformed into the following list:

```
['warren buffett_1', 'apple_2', 'berkshire hathaway_2']
```

Then we use the list to build co-occurrence matrix.

4 Co-occurrence Matrix

The co-occurrence matrix is used to record how many times entity A and entity B are both mentioned in the same article. To reduce the storage size of the co-occurrence matrix, we actually use a nested dictionary to construct the matrix. For example, the co-occurrence matrix 1 could be converted to this nested dictionary 1:

	A	B	C	D	E
A	12	2	4	10	3
B	2	10	3	9	0
...	...	...	...	...	...

Table 1: Co-occurrence Matrix

Listing 1: nested dictionary

```
Co-occurrence = {A: {A:12, B:2,C:4,D:10, E:3},
                 B: {A:2, B:10, C:3, D:9},
                     ...}
```

From the previous co-occurrence, we can see the number of articles mentioned entity A, which is 12. We can also see how many times A and E are both mentioned within same news., which is 3.

5 Baseline Model

We firstly match entities from co-occurrence matrix with entities in data2, then randomly choose 80 percent of matched pairs for training. And the rest 20 percent for testing.

The pseudocode for our baseline model is shown on next page [1]. The inputs for the baseline are the training set and testing set, both of which are dictionary whose item is company and industry pair, and the co-occurrence matrix. The basic idea of the baseline is that for each company C in the testing set, we will create a list $C_{industries}$ whose length is equal to the number of all kinds of industries we have, and all of values are zeros. Each index of $C_{industries}$ corresponds an industry. We will also find all companies that are mentioned together with C and how many times they appear in the same article with C by simply looking up the co-occurrence matrix. Suppose the result is denoted as C_{info}, which is a dictionary whose item is a pair of the name of a company $C_{cooccur}$ and an integer $C_{cotimes}$ which represents how many times C and $C_{cooccur}$ are mentioned in the same news article. For each pair of $C_{cooccur}$: $C_{cotimes}$ in the C_{info}, we will find whether $C_{cooccur}$ is in the training set. If it is, we find which industry $C_{cooccurIndustry}$ it belongs to and update the element in $C_{industries}$ whose index is corresponding to $C_{cooccurIndustry}$ by adding $C_{cotimes}$. After all of the items of C_{info} has been iterated, the C will be predicted to belong to the industry whose corresponding value in $C_{industries}$ has the greatest value.

6 Embedding

In order to overcome the limitation of co-occurrence matrix and capture more relations between entities, we decide to train an embedding matrix use GloVe algorithm. We choose GloVe based on two reasons: 1. Both GloVe and our data input are co-occurrence matrix. 2. It is widely used in Natural Language Processing field and achieved good performance.

Algorithm 1 BaselineModel

```
 1: procedure BaselineModel(co_maxtrix, training, testing,industryToIndex)
 2:     co_matrix: the co-occurrence matrix
 3:     training: dictionary whose each of item's key is the name of company and the value
                is the industry it belongs to
 4:     testing: dictionary whose each of item's key is the name of company and the value
                is the industry it belongs to
 5:     industryToIndex: dictionary whose each of item's key is one type of industry and
                value is corresponding index
 6:     $indexToIndustry \leftarrow$ swap $industryToIndex$.key with $industryToIndex$.value
 7:     $CS_{unknown} \leftarrow$ list($testing$.keys)
 8:     $num_{industries} \leftarrow$ len($industryToIndex$.keys)
 9:     $preds \leftarrow$ dict{}
10:     while TRUE do
11:         $num\_before_{unknown} \leftarrow$ len($CS_{unknown}$)
12:         for $C \in CS_{unknown}$ do
13:             $C_{industries} \leftarrow [0]*num_{industries}$
14:             $C_{info} \leftarrow co\_matrix[C_{industries}]$
15:             for $C_{cooccur}, C_{cotimes} \in C_{info}$.items do
16:                 if $C_{cooccur}$ exists in $training$.keys then
17:                     $C_{cooccurIndustry} \leftarrow training[C_{cooccur}]$
18:                     $key \leftarrow industryToIndex[C_{cooccurIndustry}]$
19:                     $C_{industries}[key] \leftarrow C_{industries}[key] + C_{cotimes}$
20:                 end if
21:             end for
22:             if sum($C_{industries}$) $> 0$ then
23:                 $index_{max} \leftarrow$ argmax($C_{industries}$)
24:                 $pred \leftarrow indexToIndustry[index_{max}]$
25:                 $training$.add({$C$:$pred$})
26:                 $preds$.add({$C$:$pred$})
27:                 $CS_{unknown}$.remove($C$)
28:             end if
29:         end for
30:         if len($CS_{unknown}$) $>=$ $num\_before_{unknown}$ then
31:             break
32:         end if
33:     end while
34:     return $preds$
35: end procedure
```

6.1 Original GloVe

The Global Vectors for Word Representations is a well-known embedding Algorithm. It is designed to train word embedding based on the co-occurrence of word pairs. Its objective function is listed below:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - \log(X_{ij}))^2$$

Where X_{ij} is the co-occurrences number of $word_i$ and $word_j$, w_i is the embedding vector of $word_i$, and b_i represents the bias of $word_i$. Note that in GloVe's paper the authors used two embedding matrices W and $\tilde{W}$ to introduce more randomness and regularize the training process, and took the sum of two embedding matrices in the end. The function $f(X_{ij})$ is a weight function as another regularizer.

$$f(x) = \begin{cases} \frac{x}{x_{max}}^{\alpha}, & \text{if } x < x_{max} \\ 1, & \text{otherwise} \end{cases}$$

Here they took alpha as 3/4 from empirical results, and $x_{max} = 100$.

Their paper used an adaptive sub-gradient descent method called Ada-grad[4]. Ada-grad modifies the learning rate η for a parameter with respect to the parameter's past updates .

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

Here $\theta_{t,i}$ represents the parameter i at time step t, η is the learning rate, $g_{t,i}$ is the gradient of $\theta_{t,i}$. G_t is a diagonal matrix such that $G_{t,ii}$ is the sum of all updates θ_i had before. Epsilon is a smoothing parameter.

With Ada-grad, infrequent word embeddings would be updated more and frequent word embeddings would be updated less, we see it as another penalty for frequent entity pairs.

We firstly trained our embedding using original GloVe algorithm. But when we check the embedding matrix by looking at nearest neighbors of well-known entities like Apple, Microsoft and Steve Jobs, the neighbors are not relevant. Also, the average pair cosine similarity is around 0.5, which is pretty high.

6.2 Some Rethinking

We then went back to the algorithm. We found the GloVe algorithm applies high penalties to pairs with high co-occurrence, $\log(X_{ij})$ and $f(X_{ij})$ were designed to prevent popular co-occurrences from influencing the embedding too much. And using two-embedding matrices makes the final embedding more unstable. We compared word embedding task with our entity embedding task: For word-embedding, there are lots of words like “a” or “the” that have high co-occurrences with many different kinds of words. However, for entity embedding, it is pretty hard to find an entity with such annoying attribute. In other words, high co-occurrence pairs, which are treated as noise in word embedding, should not be treated as noise in entity embedding.

6.3 A New Algorithm

Considering the difference between word embedding and entity embedding, we designed a new objective function:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - X_{ij})^2$$

$$f(x) = \begin{cases} x^{1.5}, & \text{if } x < 100 \\ 100^{1.5}, & \text{otherwise} \end{cases}$$

In the new objective function, we removed the $\log(X_{ij})$ regularization from original GloVe and change the weight function $f(X_{ij})$ from concave to convex. We believe those pairs with high co-occurrences are actually more valuable than pairs with low co-occurrences, thus should be assigned with more weights. However, to prevent overflow in training process, we set a upper limit of $100^{1.5}$ for the weight. In our training process we also replaced Ada grad with naive SGD.

We trained new embeddings with the revised algorithm. This time we successfully find similar entities by looking at their embeddings' nearest neighbors, but the average similarity between entities pair are still pretty high. In our assumption, if each embedding's dimension represents some attribute of the entity, then two irrelevant entities should have large difference in most dimension, which results in a low cosine similarity.

Here we see the high cosine similarity as the model's correspondence to the loss function $(W_i W_j - b_i - b_j - X_{ij})^2$, which pushes not so relevant pairs' product high. To fight with this issue, we add a new regularizer so the final objective function becomes:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - X_{ij})^2 + \lambda(w_i^T w_j + b_i + b_j)^2$$

Here we choose λ to be 5, since the average co-occurrence count is around 4.

This is the final version of our embedding algorithm, the new embedding trained with it finally meets our expectation.

7 Machine Learning

7.1 Unsupervised Learning

We implemented KNN[1] and K-means[2] to check the quality of our embedding matrix.

7.1.1 KNN

For KNN, instead of using it as a supervised learning algorithm, we only used its KD-tree structure to find the nearest neighbors of certain entities. The results are listed below in Table 2 and Table 3. All three embeddings are trained on 2010 Wikipedia dataset, "Original" embedding was trained with original GloVe algorithm, "Revised" was trained was our algorithm without the regularization part,"Regularized" was trained with our regularized algorithm.

neighbor	Original	Revised	Regularized
neighbor1	united states. lewis	ibm	novell
neighbor2	lenin military academy	intel	gpl
neighbor3	paris warren. paris jazz warren	apple	iso
neighbor4	aerovivianda	university of british columbia	unicode

Table 2: Nearest Neighbors of Microsoft in three embeddings

neighbor	Original	Revised	Regularized
neighbor1	metallica	university of cambridge	nyu
neighbor1	berklee college of music	bmw	city college
neighbor1	mountaineers	university of massachusetts	columbia university
neighbor1	pba	nhs	columbia college

Table 3: Nearest Neighbors of New York University in three embeddings

7.1.2 K-means

We also used K-means to divide the embeddings into 9 clusters and then manually match these clusters to 9 industries. The best results we got is only accruacy = 0.159 . The poor performance of this unsupervised algorithm meets our expectation because our embedding has pretty high dimension and embeddings belong to a single industry does not necessarily gather together.

7.2 Supervised Learning

We started doing supervised learning with multi-class classification. We matched 6025 training data from 1,307,082 data we collected from SEC and Nasdaq with exact matching. Since our data is highly unbalanced, we dropped two class (media and agriculture) that contain very few instances. We implemented Logistic Regression, SVM[3] and Random Forest[5] in a OneVsAll way using Scikit-learn[8], the result is not ideal.

We thought it could be some issues with using Scikit-learn, so we implemented The OneVsAll architecture by ourselves, but only found limited improvement. Table 4 summarizes our results, Scikit-learn is the result we got using Scikit-learn and Self Implementation is the result we got with our own implementation.

Algorithm	Scikit-learn(accuracy rate)	Self Implementation(accuracy rate)
Logistic Regression	0.1486	0.2325
SVM	0.1506	0.1233
Random Forest	0.1448	0.1881

Table 4: Multi-class Classification Results

We then tried many different approaches to improve the result, including tuning hyper-parameters, stacking models and building other models but none of them really worked.

On the other side, we found the results of binary classification is surprisingly good. We used SVM and Random Forest to classify whether a company belong to one industry or not, and the AUC for all industries are above 0.8. The clear contrast between binary classification result and multi-class classification result attracted our attention.

We implemented the binary classification again, and surprisingly, this time the AUC for all industries fluctuate around 0.5. After we checked potential causes like data leakage, we found the only difference between these two implementations is the choice of train/test data.

8 Error Analysis

8.1 Data Quality

The reason hides in plain sight: Our data is actually sorted by data quality. For example, we have more than 1,300,000 entities, and most of the big companies we know are in the first 10,000 entities. Index of Facebook is 342, index of Google is 198 and index of general electric is 2631. It is because our sparse co-occurrence matrix was stored as a 2-layer nested dictionary like this:

```
{
    k_1:{k_2:v_1,k_3:v_2,k_4:v_3,...k_1233:v_1232},
    k_1234:{k_1235:v_1234,.....},
    .....
    }
```

When the dictionary was converted to a list of keys, it follows the order from k_1 to k_n. So a popular company, which has co-occurred with many entities, would be listed way above a company in obscurity.

8.2 Binary Classification

In our first binary classification implementation, we unconsciously used top quality data by sub-setting data with top index. But in our second classification, we used a mixture of data with both top index and bottom index, that's why the performance dropped.

Since such "sorting" still involves some randomness, we use the sum of co-occurrence to sort our embedding again. Then we performed a dimension reduction using t-SNE[6], comparing the visualization of data with different qualities. Here we use Technology industry as an example, we took three sets of Technology embeddings: The top 5%, top 20% and top 50% data, compare them with a sample of Non-Tech embeddings. As Figure 1 shows, the classification becomes harder when more low-quality data is brought in.

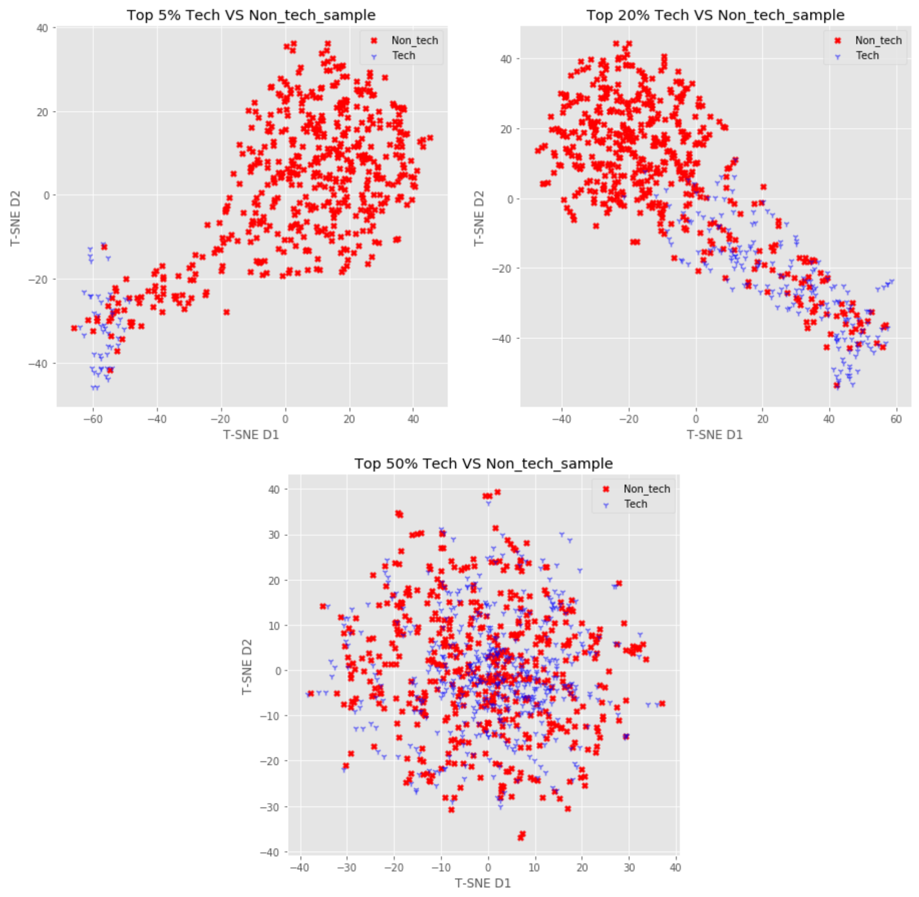

Figure 1: t-SNE Visualization of data with different quality

We did more experiments by training and testing Random Forest classifiers with different data quality. Using technology embeddings as example, we call the top 35% data "Good" data, from top 35% to top 65% "Neutral" data, and the rest "Bad" data. Our experiments are listed in Table 5 with results.

Tech Train	Tech Test	Non_Tech Train	Non_Tech Test	AUC
Good	Good	Bad	Bad	0.94
Bad	Bad	Good	Good	1.0
Good	Good	Good	Good	0.99
Bad	Bad	Bad	Bad	0.50
Good	Bad	Good	Bad	0.46
Bad	Good	Bad	Good	0.45

Table 5: Binary classification on difference qualities of data

The conclusion is clear: "Good" data is extremely important for training, only a classifier trained with "Good" data has the prediction power. However, even a classifier trained with "Good" data would not be able to distinguish "Bad" technology embeddings from "Bad" non-technology embeddings.

8.3 Multi-class Classification

We also find the reason why Multi-class classification has such a poor performance. Figure 1 shows the train/test split for our very first version of Binary Classification,which has an AUC of 0.96. We can see it is actually similar to use "Good" training data, "Bad" and "Neutral" Tech testing data and "Good" Non-Tech testing data.

However, for Multi-class it is a very different situation: We need to use a lot of high-quality training data for each industry, which seriously influenced the quality of our testing data, as shown in Figure 2. We can definitely spare high quality data for testing but that means we need to sacrifice training data. In general, the shortage of high-quality data results in the failure of multi-class classification. To verify our assumption, we did another experiment: implement the Multi-class classification for

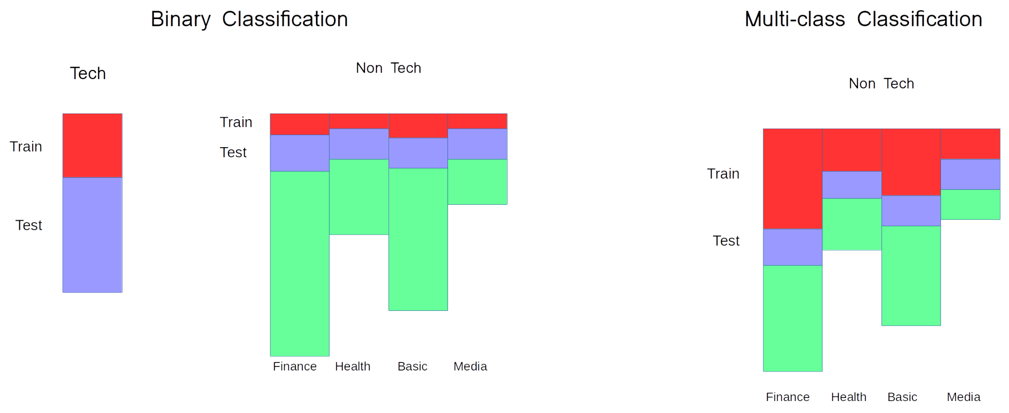

Figure 2: Test/Train Split in Binary and Multi-Class Classification

different number of classes, results are summarized in Table 6. And it is clear that average AUC decreases as number of class increases, as we assumed.

Num of Classes	Single Class AUC
One	0.99
Two	0.64, 0.76
Three	0.58, 0.68, 0.66
Four	0.54, 0.60, 0.65, 0.62

Table 6: The relationship between the performance of single binary classifier and the number of classes

9 Conclusion

Although the embedding model does not outperform baseline model in Multi-class classification because of poor data quality, it shows strong advantage in Binary classification when the data quality is well controlled. We compared baseline model with our embedding model using top 50% data of every class and a test/train split of 0.2/0.8, the results are listed in Table 7:

Industry	Baseline Accuracy	Model Accuracy	Model AUC
Basic Industries	0.2154	0.9922	1.0
Consumer Services	0.6842	0.9966	1.0
Finance	0.4568	0.9937	1.0
Health Care	0.2917	0.9896	1.0
Miscellaneous	0.2439	1.0	1.0
Public Utilities	0.1000	1.0	1.0
Technology	0.4946	0.9844	1.0

Table 7: Baseline V.S. Model on Binary classification

It proves that when the data quality is good enough, embedding algorithms could successfully project entities into high-dimensional space and overcome the limitation of co-occurrence matrix, like the disconnected components or noise from connecting with other classes.

Except the model performance, our findings about GloVe model, like the weight function and non-uniform distribution of embedding qualities could also be valuable for other relevant projects.

10 Future Work

For this task, there are still a lot of things left for us to explore, like effect of “OneVsOne” architecture and other classifying algorithms. For our modified GloVe algorithm, the effect of the regularizer we add would also be interesting to explore. Besides from models, since we see the potential of our embedding algorithm and its dependency on high-quality data, in the future we will try to collect more high quality data and see if there is any significant improvement on performance. Also, generally speaking our algorithm could easily be apply to any kind of co-occurrence matrix, which makes it possible to transfer this project to a different task, like recommender systems. We believe our algorithm would work well especially when the data source is comprehensive, so we would try to implement our algorithm on such a new task.

References

[1] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[2] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[5] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[6] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[7] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[9] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.