Generating Adversarial Examples for Recruitment Ranking Algorithms

by

Anahita Samadi

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

University of Texas at Arlington
December 2020

Abstract

Generating Adversarial Examples for Recruitment Ranking Algorithms

Supervisor: Dr. Shirin Nilizadeh

Committee Members: Dr. Dajiang Zhu, Dr. Jiang Ming

There is no doubt that recruitment process plays an important role for both employers and applicants. Based on huge number of job candidates and open vacancies, recruitment process is expensive, time consuming and stressful for both applicants and companies. In today's world so many recruitment processes are based on machine learning techniques. Therefore, it is very important to ensure security of these algorithms. Adversarial examples are proposed to examine vulnerability of machine leaning algorithms. Many research studies have been done on evaluating the resistance of artificial intelligence-based systems, in computer vision and text classification, against adversarial examples. However, to the best of our knowledge, there is no other work evaluating the robustness of NLP-based ranking algorithms that are used in recruitment process. In this study, we proposed an attack model for generating adversarial texts and evaluate its success rate on a set of real-world recruitment applications. We carried out our study into two settings: white-box and black-box. In white-box setting, we proposed a new approach for keyword extraction, and we applied our technique to change the target resume into an adversarial example. Through extensive experiments, we examined our approach for different recruitment algorithms, and we found that on average adversarial examples have significant rank improvements. In black-box setting, we assumed that the adversary has no knowledge about recruitment process and matching algorithms. We proposed a neural network architecture to determine the proper keywords to be added to the adversarial resumes. The keywords that were predicted by our proposed neural network were tested in two different settings: (1) simple setting where recruitment is a classification task for accepting and rejecting the resumes, and (2) more complex setting where recruitment algorithm is a ranking algorithm that ranks the resumes. We found that in setting (1) number of accepted resumes increased significantly after adding predicted keywords and in setting, over 95% present of resume got accepted (2) most of the resumes experienced great rank improvement after predicted keywords were applied for example over 50% of resumes them got over 150 number rank improvement. This study shows that ranking algorithms that use very popular embedding algorithms, such as TF-IDF, and USE are vulnerable to adversarial examples.

Acknowledgements

I would like to express my sincere gratitude to my dear supervisor, Dr. Shirin Nilizadeh, whose insight, and knowledge steered me through mater’s journey and made this success possible with her persistent support and guidance and provision. I would like to extend my gratitude toward my committee members, Dr. Dajiang Zhu and Dr. Jiang Ming, for their time and constructive feedback. In addition, a thank to Ms. Peace Ossom Williamson, my supervisor in RDS team. She was truly a friend throughout this path and always encouraged me and supported me to overcome all the hardships.

I would like to thank my parents, Roshanak Talebian and Morteza Samadi and my sister, Katayoun Samadi who were always there for me and have been my best friends. They always reminded me of my strengths, goals, and the bright future I am building with my hard work.

I would like to thank my dear husband Mohammadvaghef Ghazvinian for his support and patience. Finally, my deepest gratitude goes to my little darling daughter, Hannah, for bringing warmth and hope to our life and soon became the most important reason for working hard for a better future.

Table of Contents

List of Figures

List of Acronyms

NLP	Natural Language Processing
ML	Machine Learning
USE	Universal Sentence Encoder

Chapter 1.

Introduction

Recruitment procedure is highly important in areas including job applications. Companies seek to recruit candidates that fit better in the position, and job seekers try to be recognized among the large number of applicants. According to (https://code.org), there were over 500,000 open computing positions in the US in 2017 and it is expected that in 2020, the available seats will exceed by millions of job opportunities. Recruitment process is very competitive and stressful for both applicants and recruiters.

Over the last few years Natural Language Processing (NLP) methods has been widely used in different tasks of human being daily life. Since machine leaning approaches has been successful in many fields of study, NLP method has utilized ML to automize its related tasks. Despite Machine learning model's success, they are vulnerable against adversarial attacks [45]. One application of using machine learning methods in NLP in recent years is in recruitment process for identifying the best candidates for specific job descriptions. Companies use from simple algorithms to complicate ML based models to identify qualified candidates to save time and money. In Figure 1-1 shows how recruitment process works. Recruitment process mostly works based on two approaches, the first is keyword matching, which analyses the resume and focuses to find relevant information to job description by using NLP techniques. The second applies machine learning techniques to match the resume and job description.

In this work, we are creating adversarial examples against recruitment applications that are based on Machine Learning approach and non-Machine Learning approaches.

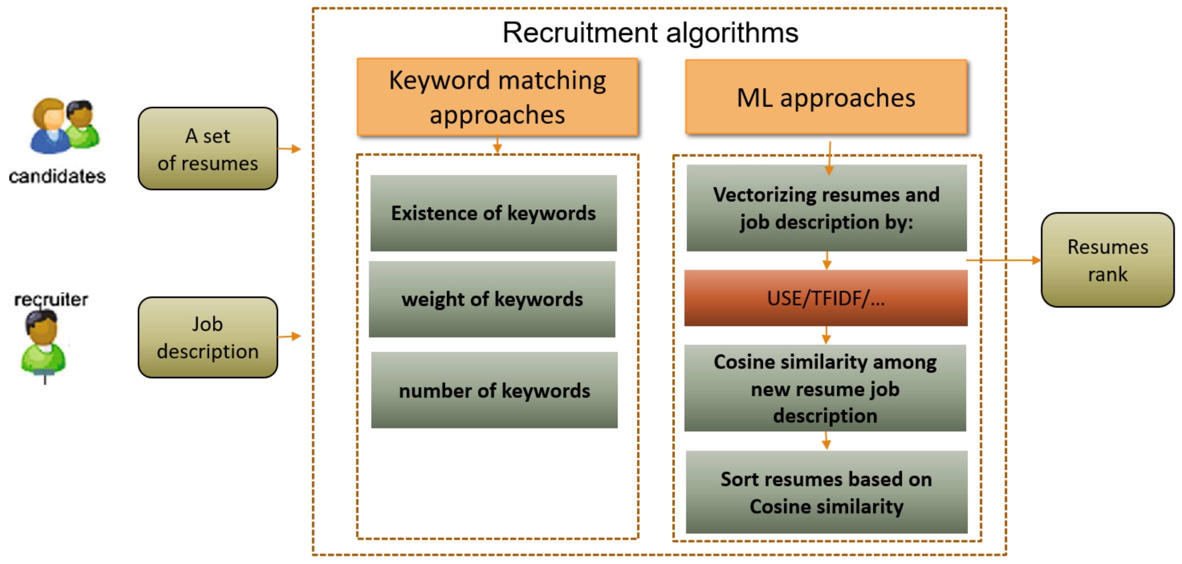


Figure 1-1 Recruitment algorithm

We pursue our study in two categories: white-box and black-box.

White-box approach: We assume that the adversary has little information on how recruitment process works, and his/her aim is to improve its resume's ranking for a specific job position automatically. After analyzing applicant's resume by employing NLP methods, we demonstrate that for a job description by adding a few keywords to resume, its rank can be improved significantly among other applicants. We proposed and applied a new approach to find best keywords that improve the ranking of applicant's resume. In particular, in this work and for the white-box approach we use:

1- job description keyword extraction: since recruiters declare their requirements based on their job description. We assumed we could have better rank for reach resume if we had job description keywords in target resume. To this end, we applied new approach for identifying best keywords based on Universal Sentence Encoder.

First, we vectorized job description by using Universal Sentence Encoder method then we tokenized job description and step by step removed keywords, each time we calculated cosine similarity between resulted job description with main job description to calculate word importance. For the next step, we fitted the keywords from phase one in the resume and we sorted keywords by their cosine similarity increasing rate. Experiments were repeated for bigram and three-gram keywords and for adding different number of keywords combination, we also tried many job descriptions and many resumes to have accurate deduction.

2- We inserted keywords that we extracted in step one to target resume. To rank resumes we assumed that recruiter was using Universal Sentence encoder as document embedding algorithm and applied cosine similarity to calculate semantic similarity among resume and job description.
3- To assure that our method in step one works great in other recruitment algorithms, we applied another approach for recruitment algorithm. We used TF-IDF to vectorize resumes and job description then cosine similarity to calculate semantic similarity among them.

Black-box approach: in this phase, we assume that we have no information about the recruitment process. To find the best keywords we designed multilabel Neural Network architecture. To have proper input for our network, we coded resumes as one-hot method, and designed the output to be keywords that lead to rank improvement for targeted resumes.

Output vector was one-hot the same as input vector. We developed black-box approach for two different settings based on recruitment algorithms:

1- **simple setting (binary classifier):** in this setting, recruitment algorithm used some rules to accept or reject applicants, the aim of our proposed neural network was to find the keywords that satisfied that hidden rules and made the target resume be accepted. We lowered the input dimension vector to simplify the problem.
2- **complex setting (recruitment algorithm is a ranking algorithm):** we considered Universal Sentence Encoder as recruiter approach. Since we applied broad number of words in this model, input vector of neural network is high in dimension, and recruiter output was rank of resume. We aimed the neural network could find best keywords that if they were added to resume, resume's rank would improve in this recruitment process.

Results: in this work we had several experiments for white-box and Black-box settings. In both settings, we had generated successful adversarial attack to fool recruitment process. In this regard, we proposed new approach for keyword extraction for White-box. By adding the extracted keywords to target resume, the resume rank improved among different recruitment algorithms. For Black-box, keywords that are generated in proposed neural network, performed a great impact on resume rank improvement.

Chapter 2.

Background and Related work

Employment procedure plays an important role for both companies and job seekers. Companies seek to recruit candidates that fit better in the position, and job seekers seek to be recognized and obtain the job among the large pool of applicants. Recruitment Industry is big, according to (www.Code.org), there were over 500,000 open computing positions in the US in 2017 [1]. It is expected that in 2020, the available seats will exceed by millions of job opportunity. Recruitment process is very competitive and stressful for both applicants and recruiters. Based on Glassdoor statistics, the average job opening attracts 250 resumes [2]. A recent survey by the Society of Human Resource Management (SHRM) found that the average cost per hire is just over $4,000 [3]. Companies use simple to very complicated algorithms to rank resumes based on their requirements [4]. Zaman et al. [5] used big data analysis techniques which is branch of data mining field of study to choose resumes, where they created a platform to evaluate an applicant by the information they provided in social media, like Linked In and checked if it satisfied job criteria. They assigned score for each feature in job and then classified the resumes by using decision tree. Word Embedding is the neural network-based approach where words and phrases are mapped to vectors [6]. Fernández-Reyes et al. [7] applied information retrieval models to retrieve relevant resume to the requested job. They represented each resume and job description by using hybrid word embedding, then they built corpora with Word2Vec model for fast training then and reduced the dimension of resulted vector was reduced. To retrieve best resume they used cosine similarity as their ranking function. In Keenan et al. [8] users' resumes were kept in online dataset with predefined features, so the employers were able to choose among them by considering their multi-criteria in resumes. In Faliagka et al. [9] for recruitment process applicant was chosen by filling out online forms. Some machine learning methods, such as linear regression, and support vector regression were proposed. First applicant score was calculated based on the features that were mentioned in the resume, then by using mentioned machine learning algorithms best resumes were chosen. In Apatean et al. [10] classifiers, such as LDA, KNN, NB and tree, were applied to classify and choose among resumes. To create the dataset, the authors gathered all information from employees that already fitted in specific position and used that as training set. Test set included applicants who filled out the online application. The score of each resume was calculated based on the features that individuals entered in predetermined fields. Koh et al. [11] focused more on ontology concepts. First, they looked for specific information in resumes, which requested in job vacancies. Since there were too many data to match against, they created some criteria to capture some necessary information in resume and job vacancies then assigned weight for resumes, this weight showed relevancy between job and resume. Furthermore, they created a mechanism to autofill information that are missed unintentionally in the resumes; this helped candidates to have more chance to get the appropriate job. Khairina et al. [12] categorized recruitment as Fuzzy Multiple Attribute Decision Making (FMADM) then applied Weighted Product Methods and allocated weights for each key attribute in job position then checked them among applicants and then chose those applicants with a better rank. Sarda et al. [13] applied two-way relevancy ranking for every candidate after finding important attributes in job requirement. In order to find attributes, they used data mining methods. To match applicants for a job, they used Gradient Boost Decision Trees, and took advantage of online websites like LinkedIn to find better understanding of candidate qualifications.

In the following we will review important concepts of natural language processing and security concerns about that.

With board application of machine learning techniques, it is very important to ensure security of the related algorithms [14]. To understand vulnerability of machine leaning algorithms adversarial examples are proposed. Adversarial examples are served as input of machine leaning models to deliberately cause them make mistake and fool machine learning models [15]. Successful attacks illustrate bugs of machine learning models and based on that, proposing defense against such attacks helps machine leaning be more reliable. Hence, implementation of adversarial examples and defense techniques ae popular and hot topics in machine leaning and security field of study [14].

Since it is hard to distinguish between two text formats, type of possible adversarial example for NLP differs from other types of adversarial attacks for machine learning tasks [16]. Adversarial examples in natural language processing includes many different categories from character level to sentence and document level. Some of the categories are mentioned as below: Character level, which attacker, insert, or delete a specific character in the token of text. Token level, which specific token is deleted, inserted, or replaced by another token. Sentence level, which attacker changes the whole sentence.

Based on the problem, description of adversarial example can be proposed in two different settings: White-box and Black-box. In the white-box setting attacker access to the target model architecture, target parameters and input features. In the back-box setting attacker does not have access to the model specification, but it can send input to the model and receive output of the model as response [17].

2.1. Machine learning based text classification algorithms

Most of the classifiers works on analysis text in different levels of text, some of them are mentioned below:

- **Document level:** classifier considers the whole text document and categorizes it base on all sentences included in the document.
- **word level:** classifier breaks documents into word element of each document and categorizes based on words [18].

We mention some popular text classifications hereunder.

2.1.1. Word Embedding

Considering semantic meaning of the words has a great effect to better analysing the text. Since bag-of-word models do not include semantic information of the context and do not respect the order of the words, they have serious difficulty to understand the sentences. In this regard, Skip-gram and continuous bag-of-words (CBOW) models are proposed to overcome the problems of bag of word model [20]. Word embeddings are one of the few successful applications of unsupervised learning. Word embedding is a method that maps each word in a context to N-dimensional vector. This method can be categorized as feature learning, which learns to represent words with similar meaning with similar vectors [23]

Some of the most popular word embedding are mentioned in below.

Word2Vec

The Word2vec [20] is using word embedding concept to represent each word with a vector. Its' architecture is a neural network with two hidden layers, continuous bag-of-words (CBOW), and the Skip-gram model that enables Word2vec to present high dimension vector for each word. There are some pre-trained vectors for Word2Vec format, one of the most important ones is Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases.

Global Vectors for Word Representation (GloVe)

Glove concept is similar to Word2Vec. They both present a word in high dimensional vector [21]. GloVe captures both global statistics and local statistics of a corpus, in order to come up with word vectors [22]. GloVe is based on matrix factorization techniques on the word-context matrix. A large matrix that includes text information is constructed and then count each "word", and the column is how frequently we see this word in the text. Some well-known pretrained GloVe word vectorization datasets are Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download) and Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download).

FastText

Facebook introduced FastText for embedding words [24]. In the FastText, words are divided into bag of n-gram character. By using FastText one can train supervised and unsupervised representations of words and sentences. These embeddings can be used for various applications including, data compression, candidate selection, or as initializers for transfer learning [25].

BERT

BERT is another approach that applies bidirectional training to model the text. This means that BERT learns from both the left and the right side of a context during the training. This helps better understanding of the text [26]. BERT was pretrained on Wikipedia that includes over 2,500 million words. There are two BERT variants available:

- BERT Base: 12 layers, 12 attention heads, and 110 million parameters
- BERT Large: 24 layers, 16 attention heads and, 340 million parameters [27]

2.1.2. Document Embedding

There is many real-world applications that need to understand text as a whole and not only consider single words. Document embedding is an approach to create a relevant vector of a context, this will give an approximation of that document's content, and in this way, one can compare and classify two text. There are many different ways to find a vector representation of a document. In this study we applied tf-idf weighting and Universal Sentence Encoder, we elaborated them in sections 4.2 and 4.3.

Chapter 3.

Data Collection

3.1. Data bases

We needed resumes and job descriptions for our experiments. To have more accurate results, we collected 100 real applicant resumes. These resumes are public resumes and we collected them among LinkedIn public job seeker resumes, GitHub, and personal websites. To have equal chance for applicants and make our experiences closer to real world recruitment procedures, we only considered resumes related to computer science. Resumes are chosen to be in different levels of education, skills, experiments, gender, etc. We stored these resumes in text format files.

For job descriptions, we developed a web scraper in python to extract data science jobs from indeed website (*https://www.indeed.com/*). Our dataset includes over ten thousand records, which are extracted from different areas of united states.

For black-box settings, our neural network architecture needs a huge amount of data to be trained well. To have enough training set we augmented our data for our models. For simple setting model, we created five thousand records by concatenated one hundred resumes to fifty selected job description. For more complex setting, we split resumes into half, then joined upper part of each resume to others resumes lowers part. With this approach, we could maintain the structure of each resume and each resume could have common resumes information, such as skills, education, work experiment, etc. We did this procedure for all possible combinations and created a database with ten thousand resumes.

3.2. Text tuning

To have successful text analysis, it was crucial to clear out text from noise and unnecessary words. These unwanted words had adverse effect on system performance. In this section we mentioned some techniques that were applied for text preprocessing:

Tokenization: in tokenization phrase or sentence broken into meaningful part of sentence called tokens. In this study, we focused on bag of word format, so we aimed to investigate the place of each word (token) into document [18].

Stop Words: there are some words that are repeated many times in the text but they did not carry out specific meaning for document analysing, e.g., “a”, “about”, “above”, “across”, “after”, “afterwards”, “again”, etc. We chose to remove stop words form our texts [18].

Noise Removal: presence of some characters, such as punctuations did not have any positive effect for our text analysing algorithm. In this study we removed unnecessary characters [18].

Chapter 4.

White-Box approach

In the white-box setting, we assume that the adversary, who is a job seeker, has knowledge about the ranking algorithm, and tries to improve his/ her rank among all applicants by modifying his/ her resume. Adversary also needs to have knowledge about the resumes of the other applicants to better evaluate its own resume. This analysis also helps with understanding in the worst-case scenario how much the adversary can improve their ranking, and later we study black-box settings when the adversary has no knowledge about the algorithm and other applicants' resumes. In the Figure 4-1, we demonstrated our approach for adversarial attack in white box. First recruitment process receive resume and job description and rank the resume based on its ranking algorithm, in the next step adversary manipulate the resume based on the method that we proposed in sections 4.2.1. and 4.2.1., finally improved resume considered as an input of recruitment process and new resume rank calculated.

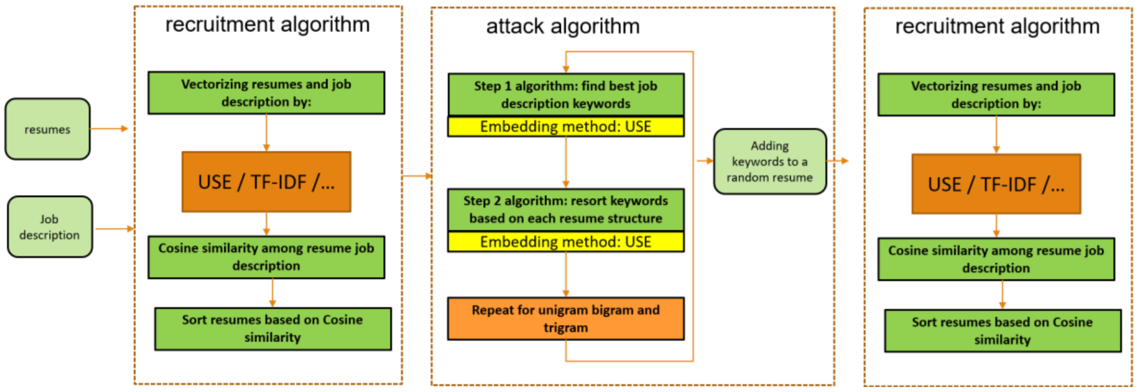

recruitment algorithm
resumes
Job description
Vectorizing resumes and job description by:
USE / TF-IDF /...
Cosine similarity among resume job description
Sort resumes based on Cosine similarity
attack algorithm
Step 1 algorithm: find best job description keywords
Embedding method: USE
Step 2 algorithm: resort keywords based on each resume structure
Embedding method: USE
Repeat for unigram bigram and trigram
Adding keywords to a random resume
recruitment algorithm
Vectorizing resumes and job description by:
USE / TF-IDF /...
Cosine similarity among resume job description
Sort resumes based on Cosine similarity

Figure 4-1 White-Box approach block diagram

4.1. Simple resume ranking algorithms

Resume Ranking Algorithms: We first examine the possibility of generating adversarial examples for non-deep learning-based ranking algorithms. We designed five resume ranking algorithms, which rank the resumes based on a list of keywords that are selected by the employer, although these could also be a result of a separate job description parser. These algorithms assign a score to each resume and then rank all the resumes based on their scores, where higher score shows a better fit to the job.

- Algorithm 1: The score is computed as the percentage of keywords in the list that are present in the resume. For example, if employers list of keywords includes {python, software development, and computer science} and a resume include only computer science, then the score for this resume would be 0.33.
- Algorithm 2: The score is computed as the percentage of keywords in the list that are present in the resume at least a specified minimum number of times. For example, if employers list of keywords includes {python, software development, and computer science}, applicant with higher count number of appearances of

these keywords in their resume had the better rank. in the other words, we for each appearance of each keywords we added one to applicant score.

- Algorithm 3: The score is computed as the number of occurrences of each keyword in the resume.

Algorithm 4 and Algorithm 5: are variations of Algorithm 1 and 4 that assign weights to keywords, when some keywords have higher weight than others.

```
**Algorithm 1** Simple
  $matches \leftarrow 0$
  $number\_of\_keywords \leftarrow length(keywords)$
  **for all** word in keywords **do**
    **if** keyword_count[word] >= 1 **then**
      $matches \leftarrow matches + 1$
    **end if**
  **end for**
  score = matches/number_of_keywords
  **return** score
```

Figure 4-2 Simple recruitment algorithm

```
**Algorithm 2** min_count
  $matches \leftarrow 0$
  $number\_of\_keywords \leftarrow length(keywords)$
  **for all** word in keywords **do**
    **if** keyword_count[word] >= keyword_mincount[word] **then**
      $matches \leftarrow matches + 1$
    **end if**
  **end for**
  score = matches/number_of_keywords
  **return** score
```

Figure 4-3 Min_Count recruitment algorithm

```
Algorithm 3 simple_weighted
  $points \leftarrow 0$
  $max\_points \leftarrow sum(keyword\_weights)$
  for all word in keywords do
    if keyword_count[word] >= 1 then
      $points \leftarrow points + keyword\_weight[word]$
    end if
  end for
  score = points/max_points
  return score
```

Figure 4-4 Simple_Weighted recruitment algorithm

```
Algorithm 4 count
  $score \leftarrow 0$
  for all word in keywords do
    $score \leftarrow keyword\_count[word]$
  end for
  return score
```

Figure 4-5 Count recruitment algorithm

```
Algorithm 5 count_weighted
  $score \leftarrow 0$
  for all word in keywords do
    $score \leftarrow score + keyword\_count[word] \times keyword\_weight[word]$
  end for
  return score
```

Figure 4-6 Count_Weighted recruitment algorithm

4.1.1. generating adversarial resumes

We added five random keywords (work attributes) to one resume and did not modify the other resumes. Then found the rank improvement due to these additions. We repeated this procedure for all resumes for all algorithms.

Here are the keywords that are dedicated for each algorithm:

For Simple algorithm and Count algorithm: {Python, AngularJS, Spark, Java, C#, PHP, CSS, html5, AI, Linux, Machine, bachelor, master, statistic, PhD, Publications, amazon, Algorithm, Microsoft, Awarded, research, scholarship, Stanford, Harvard, TensorFlow, Robotics, Reinforcement, git, thesis, OpenCV}

For Min_Count algorithm. in this set, corresponding number to each word indicated the minimum required time for each keyword to be present in each resume to be accepted: {Python, 2, AngularJS, 1, Spark, 1, Java, 1, PHP, 1, CSS, 1, AI, 1, Linux, 2, Machine, 3, bachelor, 1, master, 1, Algorithm, 1, Awarded, 1, research, 1, scholarship, 1, TensorFlow, 1, git, 1, thesis, 1}

For Simple_Weighted and Count_Weighted algorithms. in this set, corresponding number to each word indicated the weight of the word: {Python, 30, AngularJS, 10, Spark, 5, Java, 10, C#, 10, PHP, 5, CSS, 5, html5, 5, AI, 15, Linux, 15, Machine, 20, bachelor, 10, master, 15, statistic, 5, PhD, 20, Publications, 10, amazon, 5, Algorithm, 5, Microsoft, 5, Awarded, 5, research, 5, scholarship, 5, Stanford, 30, Harvard, 30, TensorFlow, 5, Robotics, 5, Reinforcement, 5, git, 5, thesis, 5, OpenCV, 5}

The random keywords that we added to the target resume: {AngularJS, Python, Java, Harvard, Machine}

The following plots show the rank improvement in X axis when adding the “random” keywords to an adversarial resume.

Following plots demonstrate the number of resumes that had rank improvement by adding 5 random keyworks. For example, in simple algorithm plot above twenty resumes had three rank improvement and for count_weighted algorithm five resumes had 32 rank improvement that is significant result.

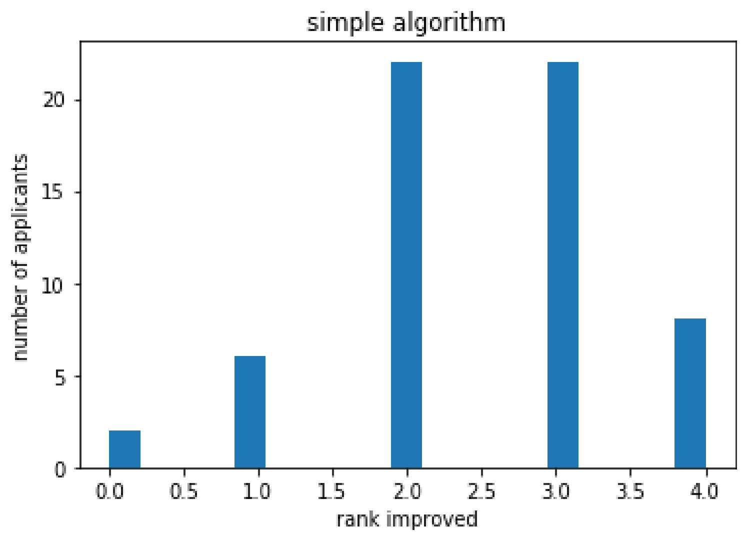


Figure 4-7 Simple algorithm rank improvement result

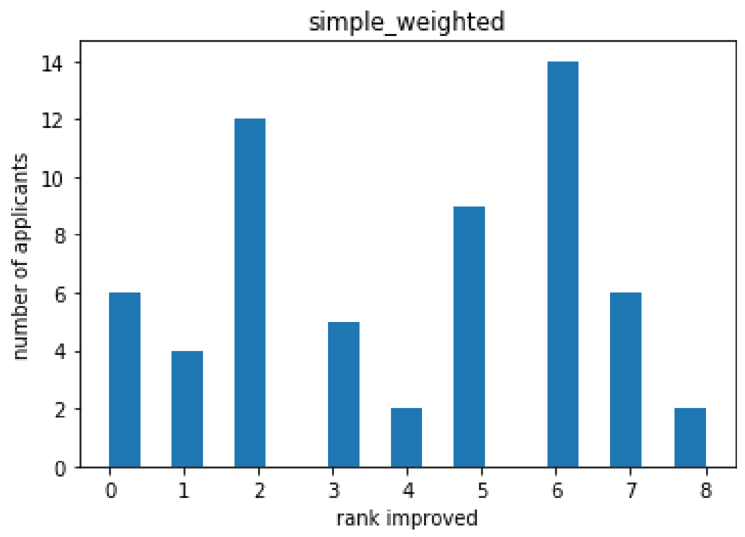


Figure 4-8 Simple_Weighted algorithm rank improvement result

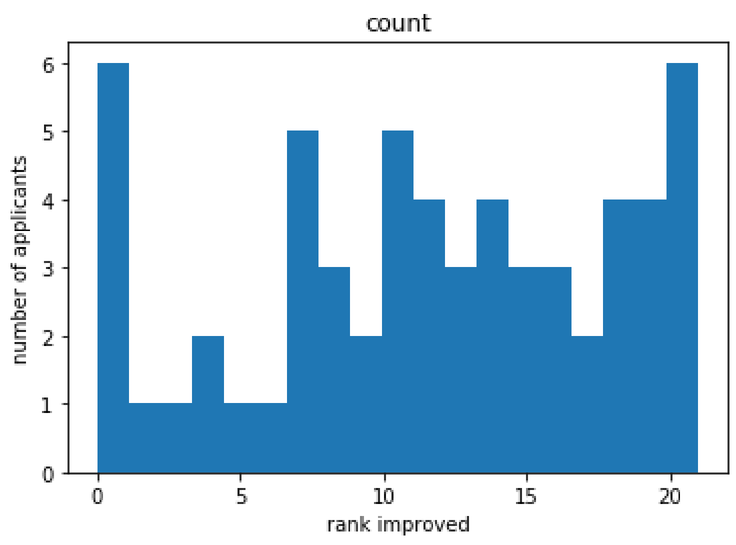


Figure 4-9 Count algorithm rank improvement result

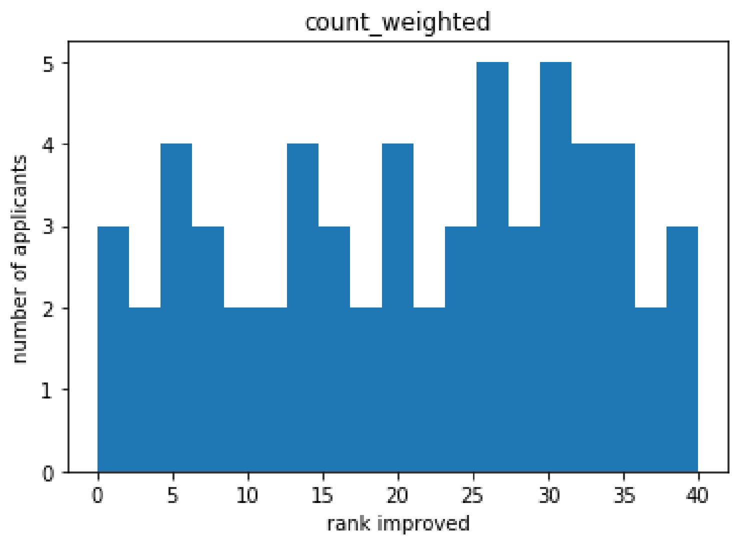


Figure 4-10 Count_Weighted algorithm rank improvement result

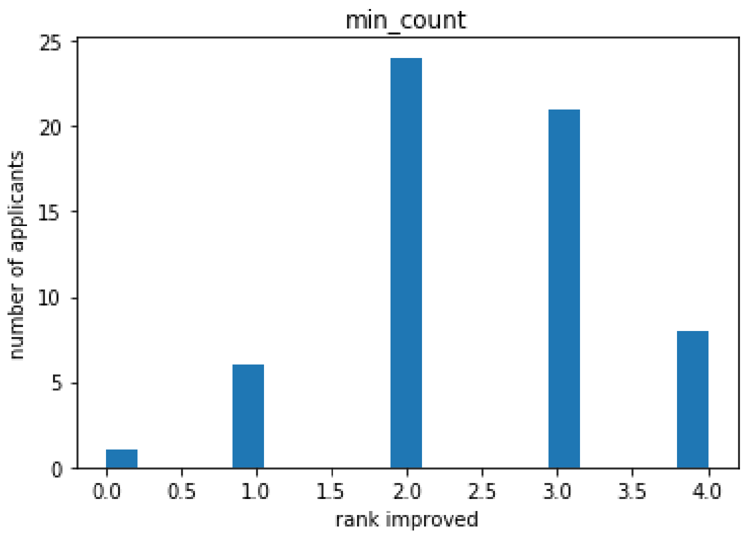


Figure 4-11 Min_Count algorithm rank improvement result

4.2. Universal Sentence Encoder

Universal sentence encoder is a well-known method that uses an encoder to convert given text to a fixed-length 512-dimensional vector. This embedding has been used to solve multiple tasks, such as semantic search [43], text classification [44], and many other NLP tasks. it will return only the most informative features and will not consider noises [29].

Figure 4-12 shows how universal sentence encoder used for Semantic similarity among sentences. Semantic similarity is a measure of the degree to which two pieces of text carry the same meaning. The metric for calculating Semantic similarity is cosine similarity. It has been shown that after embedding sentences, sentences that have closes meaning carry out higher cosine similarity, it is because they have more resampling vectors. [29][44]

We used USE pretrained model, which applies STS benchmark. STS Benchmark is a selection of the English datasets used in the STS tasks organized in the context of SemEval between 2012 and 2017. The datasets, that are used in STS, are from different types, including image captions, news headlines and user forums [29][30].

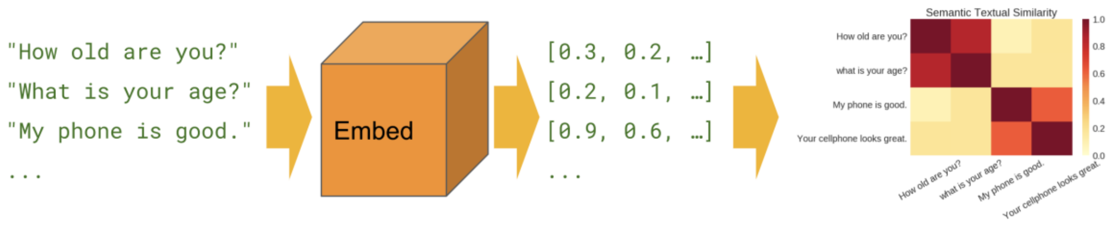


Figure 4-12 Universal Sentence Encoder sample

4.2.1. Phase one: New approach for keyword extraction

In this study we propose a new approach for keyword extraction enjoying Universal Sentence Encoder algorithm. To have successful adversarial attack for recruitment process, first we need to analyse job description as the only source from recruiter. Based on the job description's most relevant keywords and skills, the adversaries can change their resume to better match the job description. Their objection is to add best keywords into the resume and increase the resume's chance of getting through to the next round of recruitment.

In this project to evaluate similarity between resume and job description we embedded both texts. The adversary manipulated the resume and then submitted it to recruitment process. Recruitment algorithm in our project was based on universal Sentence encoder, it considered document as a whole and despite of other approached like Doc2Vec that are based on bag of words, it returned more accurate results.

Regarding to the nature of the job description, we have not received decent results from traditional keyword extraction algorithms like TF-IDF and RAKE [28], to have relevant keywords from job description we need to consider job description document as a whole not only choosing words with higher frequency as keyword. For this purpose, we enjoyed Universal Sentence Encoder algorithm to embed whole job description document. We implemented following steps regarding to extract job description keywords:

Step1: Text preprocessing: in this step we remove unnecessary symbols and words such as stop words. In many algorithms, especially algorithms that work with statistics and machine learning, noise and unnecessary attributes can have adverse effects on the performance, therefore we try to minimize their effects by removing them from the descriptions.

Step2: USE embedding: we embed the whole original job description into a vector using Universal sentence encoder. We applied one of the universal family models family pretrained model [29] The input of this model is English text and the output is a 512-dimensional vector. The model is applied to the STS benchmark [30] for semantic similarity, the universal sentence encoder model is trained with a deep averaging network (DAN) encoder.in this version default inference function now returns the Tensor instead of a dictionary. [29] we store this vector into a variable $USE_{OriginalJob}$.

Step3: Tokenizing and deleting tokens: in this step we break the original job description documents into tokens which are meaningful part of the sentence. We want to investigate the importance level of each word in the job description. We delete single token $Token_i$ from job description and save new job description that is excluded from $Token_i$ as $JobDescription_{new}$. Next, we need to embed $JobDescription_{new}$base of Universal

Sentence Encoder formant that mentioned in step2. We store this vector in variable USE_{Newjob}.

Step4: Scoring keywords: until now, we have $Token_i$ as a keyword that we want to know its place among all other tokens of job description. We need a metric to measure the similarity between two vectors USE_{Newjob} and $USE_{OriginalJob}$. For this purpose, we applied Cosine Similarity. This metric measures the cosine of the angle between two vectors which are multi-dimensional. Mathematical formula of Cosine Similarity is hereunder:

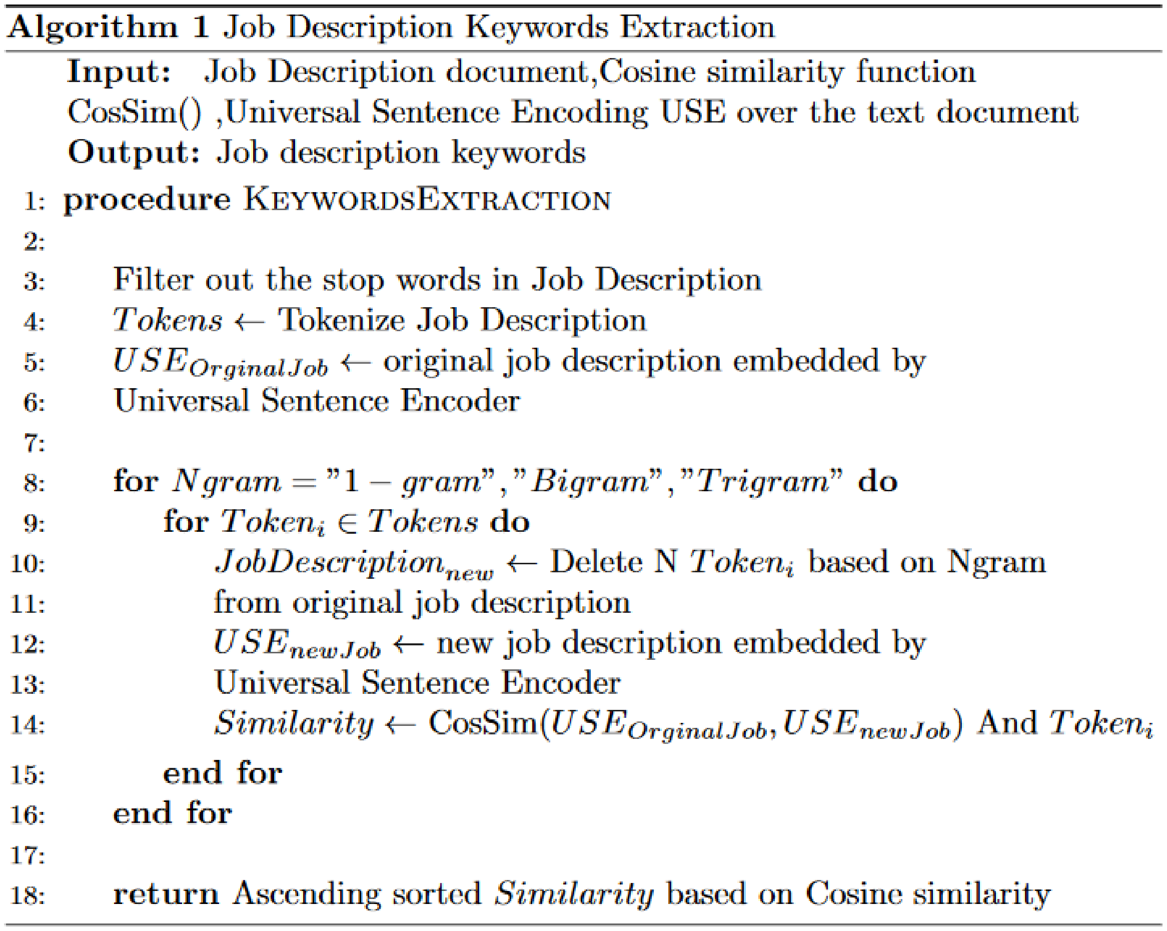
$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \ \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \ \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

The result of cosine similarity among USE_{Newjob} and $USE_{OriginalJob}$ illustrate the importance of deleted word $Token_i$. **In this regard, lower cosine similarity expressed the fact that deleted token, caused impressive change in the job description contents therefore it might be an important keyword.**

Step5: repeating all the procedure: we need to repeat step 3 and step 4 for all tokens in the job description. This procedure gives us dictionary data structure which the keys are tokens, and the values are related cosine similarities. To obtain best keywords, we sorted them in ascending order and selected the 50 best keywords among all words of job description.

Step 6: N-gram phrases: by pursuing step1 to step 5 we will obtain 1-gram bag of words. We run the experiment to have best keywords for 2-grams and 3-grams as well. For 2-grams we deleted two consequence tokens and for 3-grams we deleted three consequence tokens in step3. The rest of procedure is the same and will have dictionary of 2-grams and 3-grams in step5 that we choose 50 best keywords among them.



Algorithm 1 Job Description Keywords Extraction

```
    Input: Job Description document,Cosine similarity function
    CosSim() ,Universal Sentence Encoding USE over the text document
    Output: Job description keywords
 1: procedure KEYWORDSEXTRACTION
 2:
 3:     Filter out the stop words in Job Description
 4:     $Tokens \leftarrow$ Tokenize Job Description
 5:     $USE_{OrginalJob} \leftarrow$ original job description embedded by
 6:     Universal Sentence Encoder
 7:
 8:     for $Ngram = "1-gram", "Bigram", "Trigram"$ do
 9:         for $Token_i \in Tokens$ do
10:             $JobDescription_{new} \leftarrow$ Delete N $Token_i$ based on Ngram
11:             from original job description
12:             $USE_{newJob} \leftarrow$ new job description embedded by
13:             Universal Sentence Encoder
14:             $Similarity \leftarrow$ CosSim($USE_{OrginalJob}, USE_{newJob}$) And $Token_i$
15:         end for
16:     end for
17:
18:     return Ascending sorted $Similarity$ based on Cosine similarity
```

Figure 4-13 keyword extraction phase one pseudo code

4.2.2. Why Phase two is required

The adversary's objective is to improve his/her resume to fool recruitment process. We aim to show that without manipulating resume structure we can have impressive ranking improvement among other resumes just by adding some important keywords from job description. We noticed by adding best keywords of job description to resume best results would not be achieved, therefore we performed the following experiments:

- we added an irrelevant sentence to each pair of resume and job description: We found that the similarities between a pair of resume and job description did not change significantly.
- we added test sentence ("today is rainy day") of job description for five times to resumes: We found that similarity between the resume and job description got decreased.
- we made a fake sentence with the best job description keyword (just one word): We found that the similarities increased only for some while for the others it decreased. We added this sentence five times: we found that the similarity increased for some and decreases for some others. We added the best keyword five times instead of a sentence with the keywords to examine the importance of a sentence: We found that the similarity results did not change significantly from test4.
- we added the whole job description to the resumes: We found that similarity of some resumes increased while others decreased.

Following conclusions have be achieved based on the mentioned experiments:

- Adding a keyword have different effects among different resumes
- Adding a lot of keywords is useless and keywords needs to be relevant to resume structure, since, we added whole job description to the end of resumes and some resumes even got worse similarity to job description.

Based on the following results we developed phase two that Resort the best keywords in phase two for each resume to finds best keywords that fits in the specific resume.

Phase 2: New approach for keyword extraction We noticed just by adding best keywords of job description to resumes, best results would not be achieved, and each resume based on its previous information and structure needs specific keywords. In this regard, we developed phase two that resorts the best keywords of phase one for each resume, our objective is to find best keywords that fits in a specific resume. We performed phase two for each resume and job description by following steps:

Step1: we added best fifty keywords of phase one, one by one to the resume.

Step2: we embedded the resume by Universal Sentence Encoder after adding each keyword and we calculated cosine similarity between resume after adding keyword and job description. Then we calculated rank improvement of the resume after adding keyword. Now we resort keywords in descending order, based on the rank improvement. At this point we have list of sorted keywords for each resume.

Step3: In this step we added N numbers of best keywords that we achieved in step2 to original form of resume and we store this resume in variable $Resume_{jnew}$. We vectorized new resume and job description based on Universal Sentence Encoder then we calculate cosine similarity before and after adding keywords to calculate rank improvement.

Step4: We repeated these steps for fifty job descriptions, one hundred resumes and for group numbers of 1, 2, 5, 10, 20, 50 keywords and used unigram, bigram and trigram keywords.

```
Algorithm 2 White Box Adversarial Attack
    Input: Job Description documents,Cosine similarity function
    CosSim() ,Universal Sentence Encoding USE over the text document
    Resumes ,keywords Extraction procedure
    Output: new resumes rank

procedure WHITE BOX ADVERSARIAL ATTACK

    $Jobs \leftarrow$ all descriptions embedded by
    Universal Sentence Encoder
    $Resumes \leftarrow$ all resumes embedded by
    Universal Sentence Encoder

    for $job_i \in Jobs$ do
        $OldScores \leftarrow$ Calculate CosSim for $job_i$ and all the resumes
        $OldRanks \leftarrow$ Calculate ranks of each resumes
        $keywords_{job_i} \leftarrow$ Store 50 best $job_i$ keywords that
        extracted with KeywordsExtraction procedure
        for $Ngram \in ("1-gram","Bigram","Trigram")$ do
            for $resume_j \in Resumes$ do
                $keywords_{job_i resume_j} \leftarrow$ sort 50 best keywords
                of $job_i$ for $resume_j$ based on the fact that which one
                increase CosSim$(job_i, resume_j)$ more
            end for

            for $N \in (1, 2, 5, 10, 20, 50)$ do
                for $resume_j \in Resumes$ do
                    $resume_{jnew} \leftarrow$ add $N$ best keywords from
                    $keywords_{job_i resume_j}$ to $resume_j$
                    $Similarity_{new} \leftarrow$ calculate CosSim$(job_i, resume_{jnew})$
                    $Rank_{new} \leftarrow$ calculate new rank for $resume_j$
                end for
            end for
        end for
    end for

    return new $Rank$ and $Similarity$ for each resume after adding
    $N$ keywords of job description
```

Figure 4-14 keyword extraction phase two pseudo code

4.2.3. Results

In this section we applied Universal Sentence Encoder as part of matching algorithm. In this regard, we developed various experiment in our proposed mothed. In Fig 4-14 we chose 25 random resumes and three job descriptions, and we checked rank improvement when we added different numbers of keywords to each resume. We aimed to investigate the effect of adding different number of keywords on rank improvement. Experiments demonstrate that rank improved significantly by adding more keywords. For example, with adding 10 keywords for resume one, we get about 20 rank improvement, while adding 50 keywords for same resume and jobs will have about 50 rank improvement.

In Fig 4-15 we chose 25 random job description and calculated average rank improvement among all resumes with different N-gram setup. In most of the job descriptions we can see that average rank improvement increased after adding more keywords. For example, with adding 10 keywords for job one, we get on average 25 average rank improvement, while adding 50 keywords for same job and jobs will have 30 average rank improvement.

In Fig 4-16 our objective was to show how many resumes' ranks improve in average of all settings included all job description, different number of added keywords and different type of N-grams. As a result, we can see all resumes had rank improvement and in most of them rank improvement is significant. For example, in average of all combination, about six resumes had twenty-six average rank improvement.

In Fig 4-17 we aimed to show that when we added keywords how uni-gram, bi-gram and trigram bag of words effected average rank improvement. In this regard, we calculated average rank improvement for all combinations of different job descriptions. In this plot we can see that when our ranking algorithm is Universal Sentence Encoder, we will achieve better results for tri-grams bag of words and by increasing adding the number of keywords, rank improvement for unigram, bigram and trigram approach to each other. For example, by adding 20 keywords to resumes, in all job description combination, we will have 17,20,34 average rank improvement for unigram, bigram and trigram consequently.

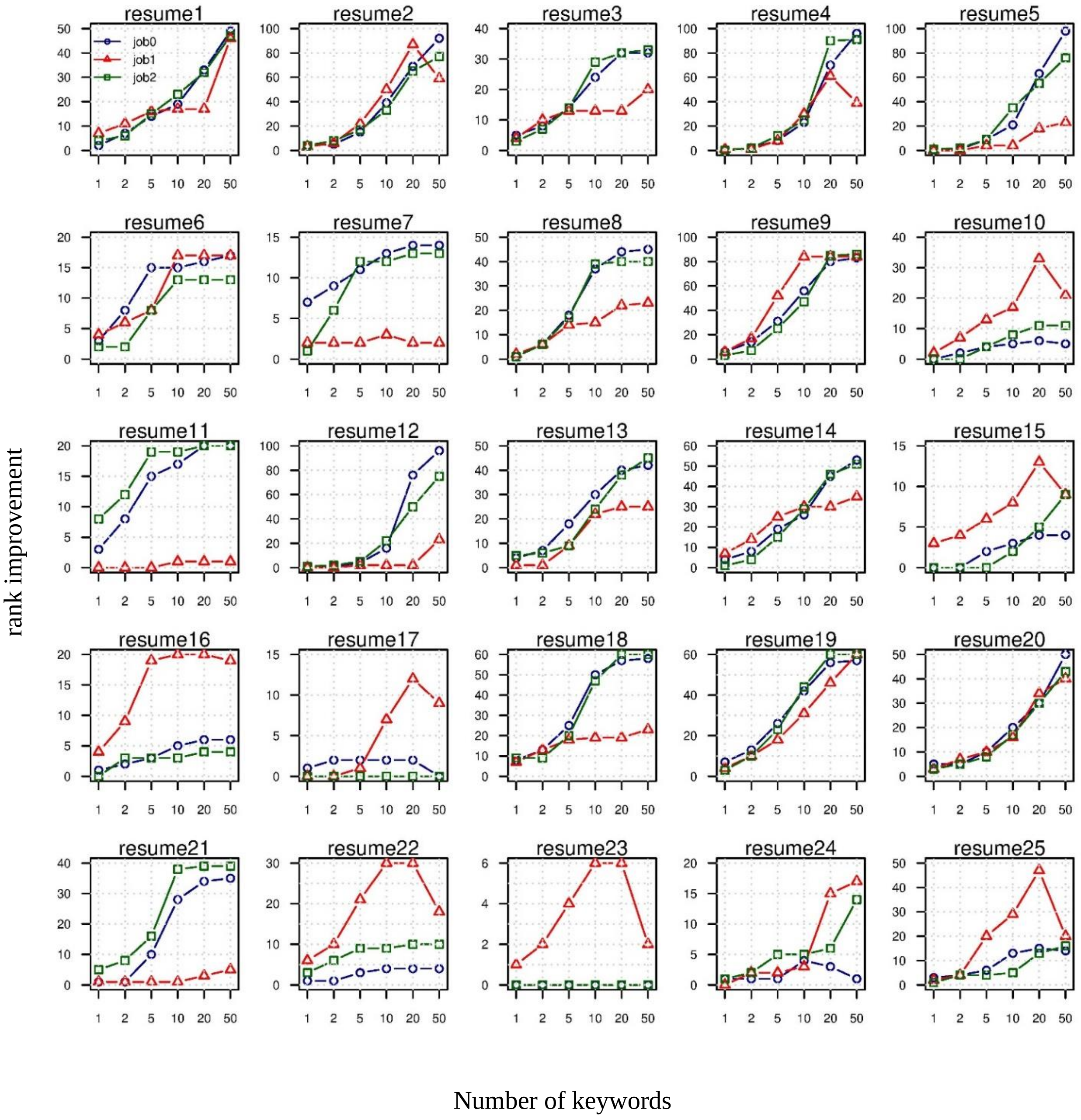


Figure 4-15 rank improvement for three random job

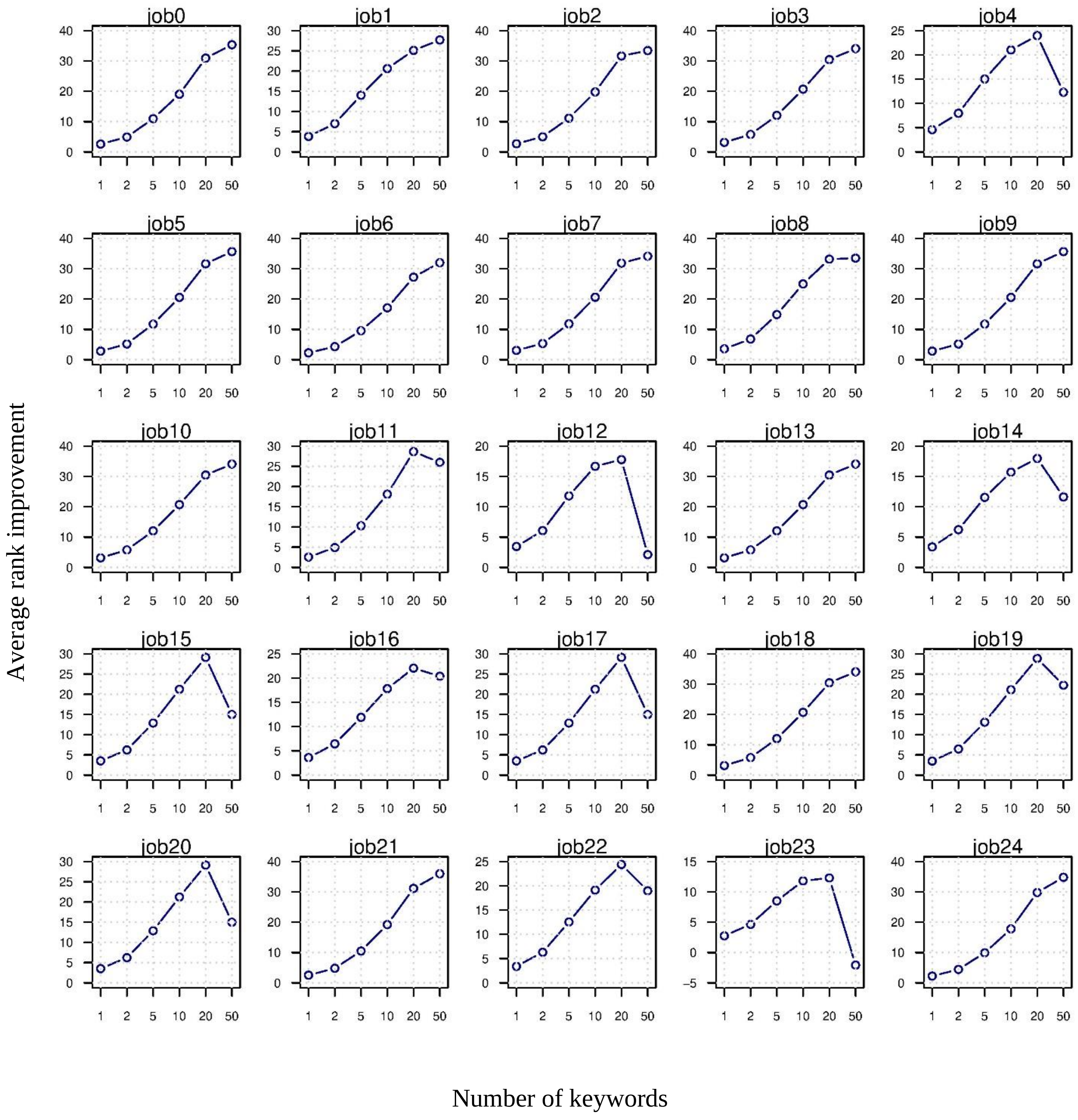


Figure 4-16 Average rank improvement for all resumes

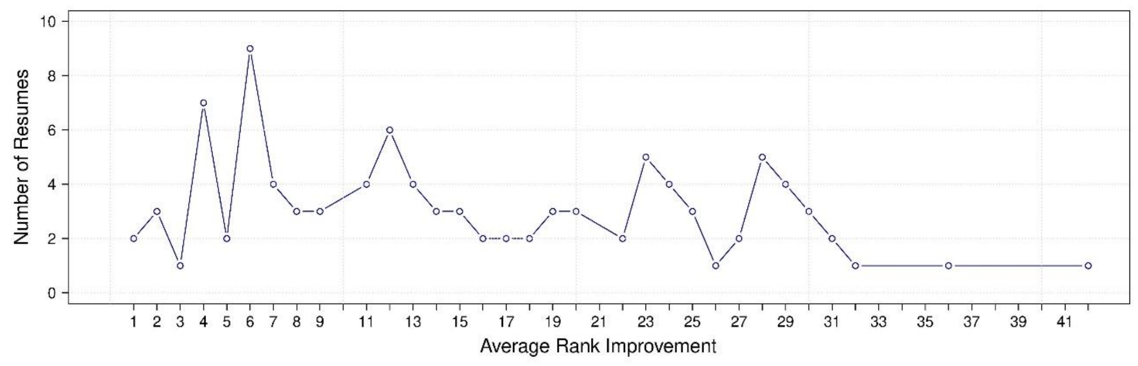


Figure 4-17 average rank improvement for all possible combinations

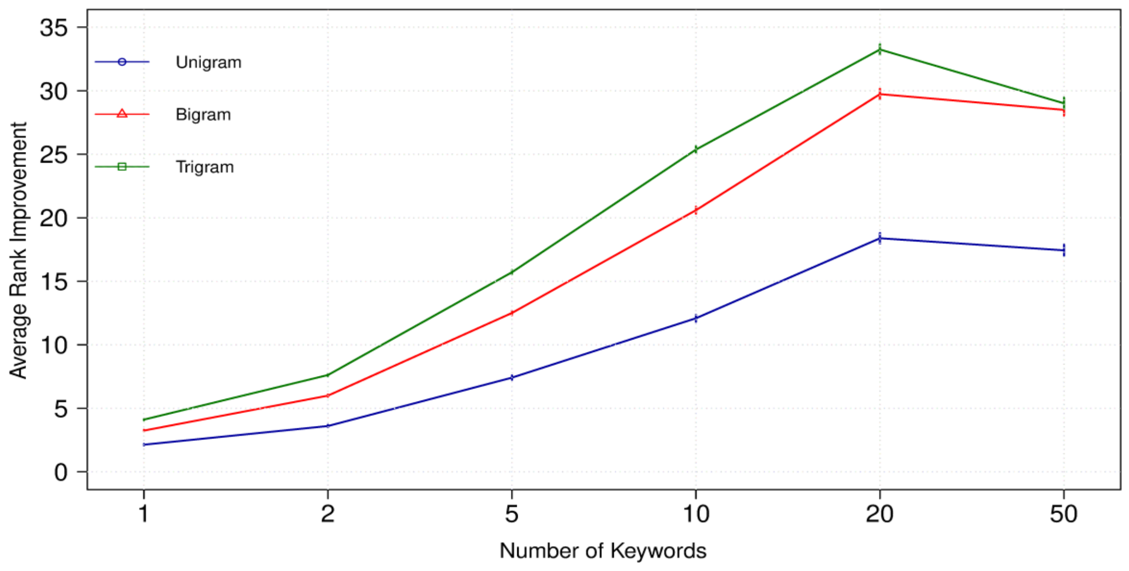


Figure 4-18 average rank improvement for ngrams

4.3. TFIDF

TF-IDF *"term frequency-inverse document frequency"* is a numerical statistic that is designed to reflect how important a word is in a collection or corpus [33].

The goal of using TF-IDF instead of the raw frequencies of a token is to reduce the impact of tokens that occur very frequently which make them less informative than features that occur in a small portion of the training corpus.

TF: Term Frequency reflects how often a term occurs in a document as every document has different length, they may occur more often in longer documents than shorter ones. Thus, TF is:

$$\mathrm{TF}_{(w,j)} = \frac{\text{(Number of times term w appears in a document)}}{\text{(Total number of terms w in the document)}}$$

IDF: Inverse Document Frequency measures how important a term is. In TF, all terms are considered similarly important. therefore, we need to cut down the frequent terms while increasing the frequency of the rare ones, by following equation:

$$\mathrm{IDF}_{(w)} = \log \frac{\text{Total number of documents}}{\text{Number of documents with term w in it}}$$

product of two weights, the TF and the IDF weight results the *TF-IDF*

However, the fundamental of TF-IDF remains the same: *TF-IDF results in larger numbers for less frequent words and is high when both IDF and TF values are high*.

In general, until 2015, TF-IDF was the traditional method to assigning values to words that emphasize the relevance in documents compared to the Bag of words way that treats every word similarly important. It is still effectively used in solving NLP problems [32].

4.3.1. Adversarial example against TFIDF based recruitment process

In this step we added keywords to resumes from our novel approach in 6.2.1, we assumed that recruiter used *TF-IDF* algorithm for ranking resumes. We vectorized resumes and job description based on TF-IDF then we calculated cosine similarity before and after adding keywords to calculate rank improvement. Our goal is to show our new approach is reliable even with deferent ranking algorithms. We repeated this step for fifty job descriptions and for group numbers of 1,2,5,10,20,50 keywords.

4.3.2. Results

In this section we applied TF-IDF as matching algorithm. To this end, we developed various experiments. In Fig 4-18 we chose 25 random resumes and three random job descriptions and checked rank improvement when added different numbers of keywords to each resume. We aimed to investigate the effect of adding different number of keywords to resume rank improvement. Experiments demonstrated that for most resumes and job descriptions rank improvement increased significantly by adding more keywords.

In Fig 4-19 we chose 25 random job description and calculated average rank improvement among all resumes with different N-gram setup. In all of the job description we can see that average rank improvement increased after adding more keywords.

In Fig 4-20 our objective was to show how many resumes ranks improved in average of all settings included all job description, different number of added keywords and different type of N-grams. As a result, we can see most resumes had rank improvement and in most of them rank improvement is significant.

In Fig 4-21 we wanted to show that when we added keywords how uni-gram, bi-gram and trigram bag of words effected average rank improvement. In this regard, we calculated average rank improvement for all combinations of different job descriptions. In this plot we can see that when our ranking algorithm TF-IDF we will achieve better results for tri-grams bag of words and by increasing adding the number of keywords, rank improvement for unigram, bigram, and trigram approach to each other.

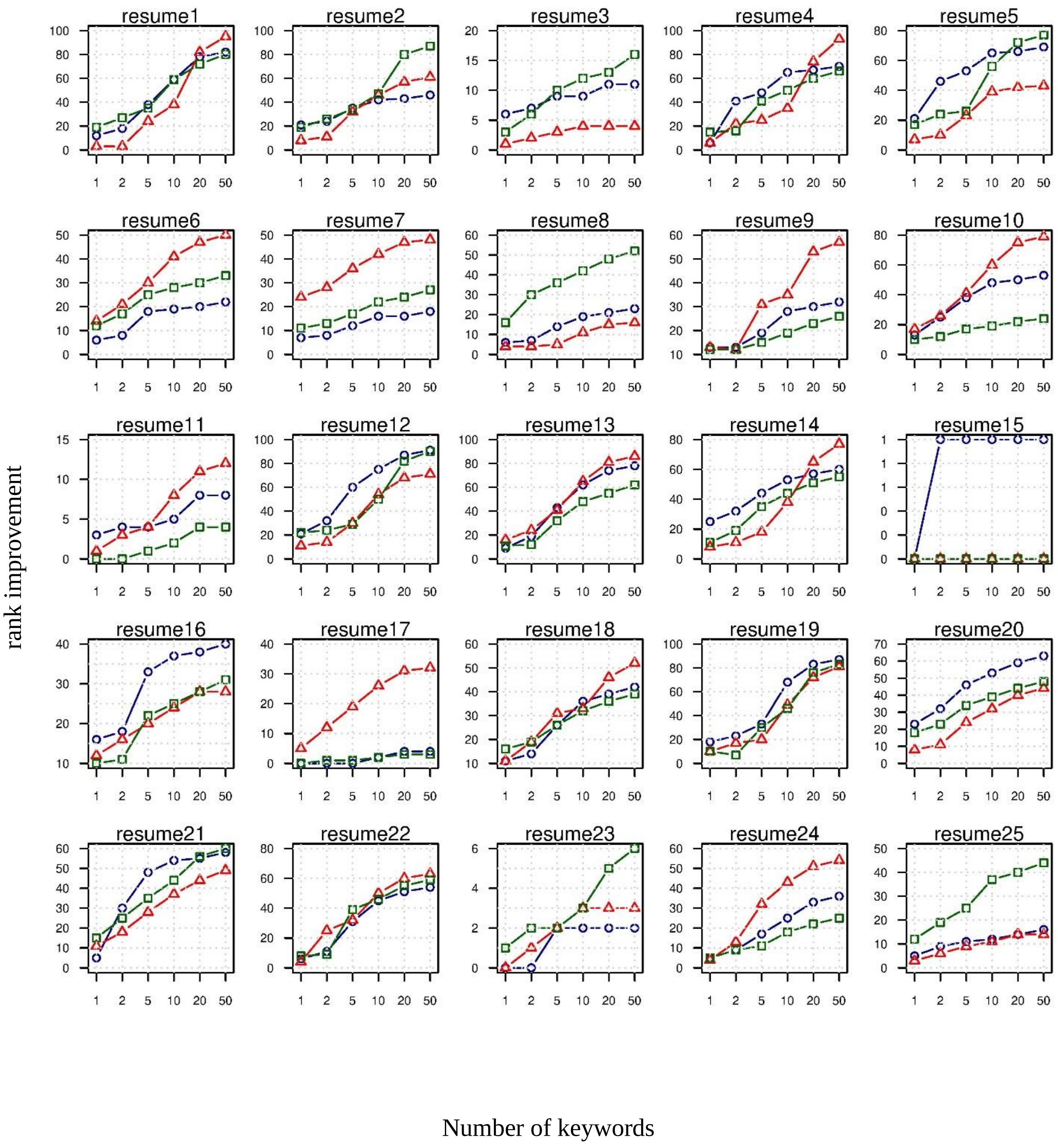


Figure 4-19 rank improvement for three random job

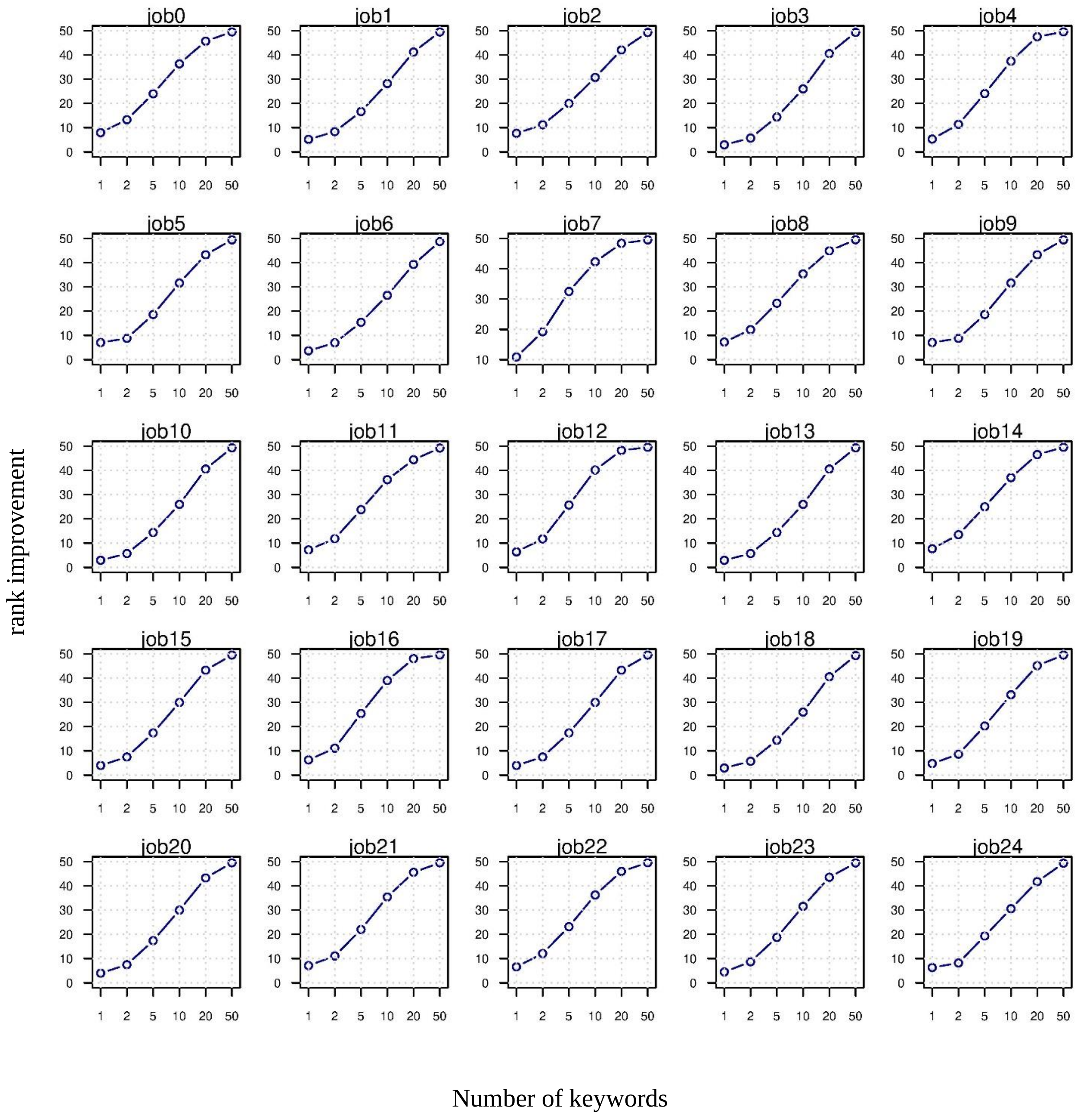


Figure 4-20 Average rank improvement for all resumes

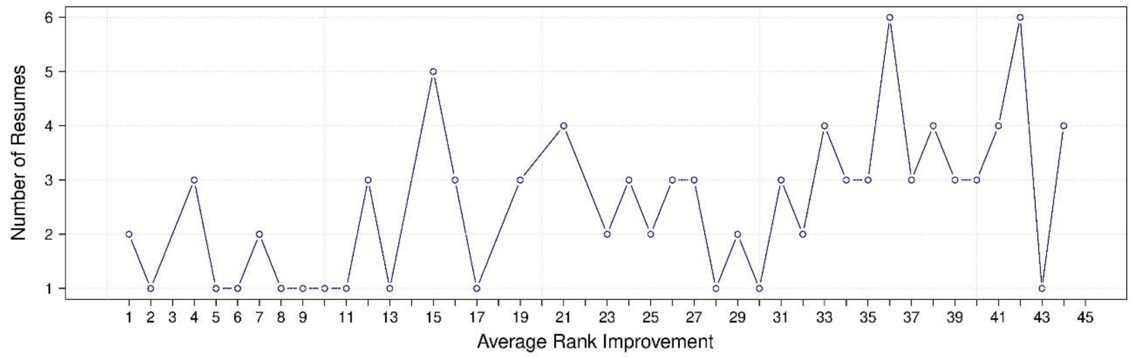


Figure 4-21 average rank improvement for all possible combinations

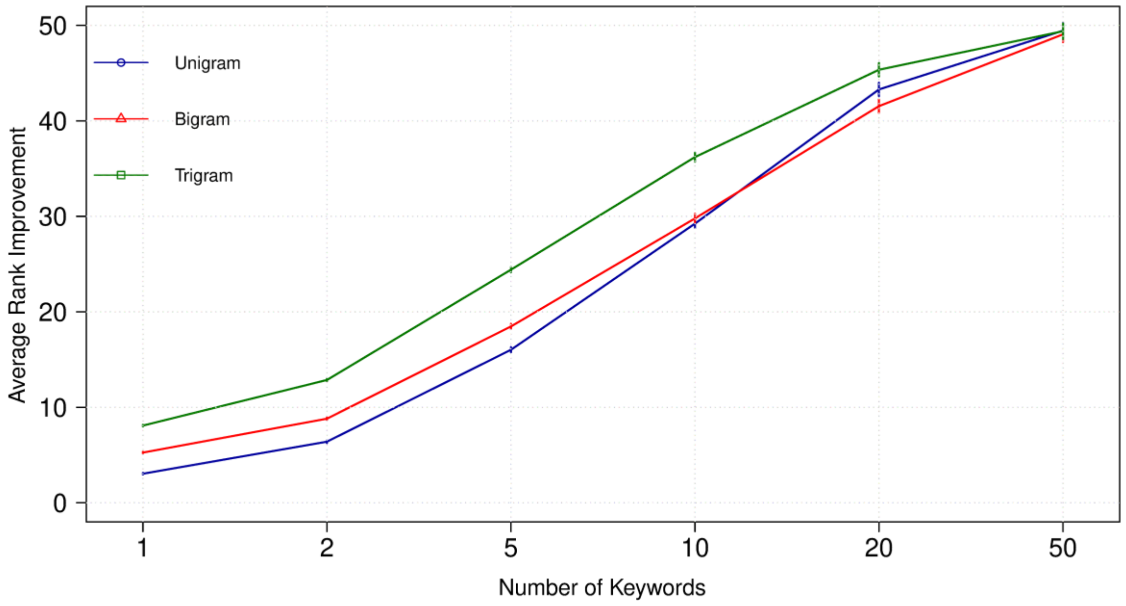


Figure 4-22 average rank improvement for Unigram,Bigram,Trigram

Chapter 5.

Black-Box approach

In the back-box setting, attacker does not have access to the model specification, but can send input to the model and receive output of the model as response.

5.1. Neural network

Neural networks are known to be able to approximate arbitrary nonlinear functions. Recent advancements in neural network architectures and GPU technology have resulted in scalable and effective algorithms leading to growing their popularity in a variety of disciplines. Neural networks have been applied in many real-world problems including business, education, and environmental prediction [31][39][19]. Neural network models have been used in NLP tasks, such as text classification, natural language inference, machine translation, and question answering [42][46][47]. In this study we applied neural network to predict keywords that improve resumes ranking. To provide an acceptable format for neural network input, we applied one-hot style [34]. Choosing proper domain of relevant words plays crucial role to code resume and job description so we tokenized all job descriptions and resumes and created dictionary of words of all resumes and job descriptions, where key is the token and value is the frequency of the word that has been repeated in all resumes and job description that are available. In the next step we need to vectorize resumes, we check each dictionary word in resume. If the word exists in resume it appends one to resume vector, else appends 0. That is how one-hot style works for our vectorization process.

Figure 5-1 shows an example where ones in the vector show the presence of those words in a sentence.

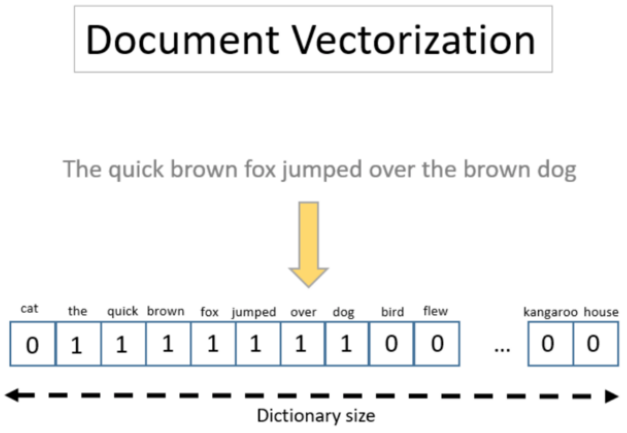


Figure 5-1 one-hot example

In this figure, dictionary size is included some words and in sample sentence. As can be seen, values of dictionary of words that are presented in the sample sentence are set to one and rest of dictionary words are remained zero [35].

In this study, we proposed neural network model where input vector is the resumes, and output is a vector that demonstrate the best keywords. In the output vector, ones indicate index of words that adding them to resume will increase the rank of resume. For recruitment algorithms we considered two different approaches that we will elaborate in the following sections. Figure 5-2 shows the architecture of our proposed neural network model.

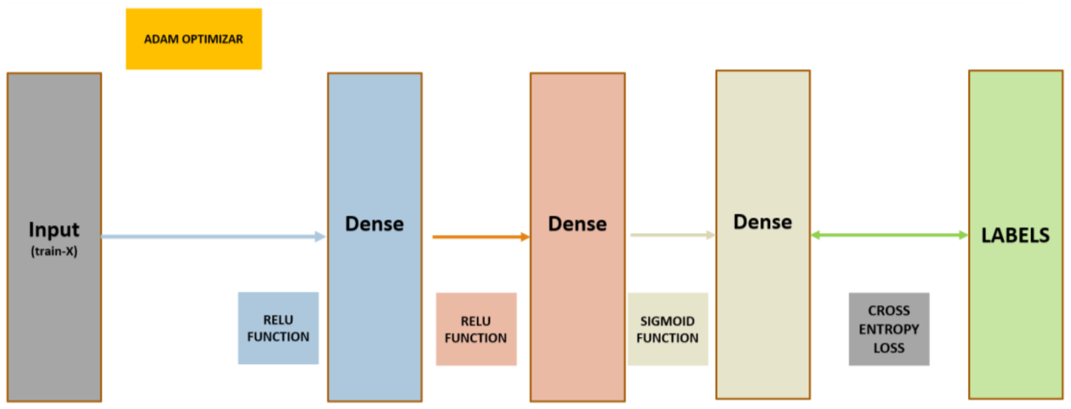


Figure 5-2 Neural Network architecture

Our proposed neural network architecture is shallow network. It consists of input layer, two dense layers as hidden layers and output layer representing the labels. For the first two hidden layers we used rectified linear unit (ReLU [37]) [y = max(0, x)]as the activation function. ReLUs are the most commonly used activation functions in neural networks models [39]. Due to their unique formulations, ReLUs provide faster training and better convergence relative to other activation functions [37]. For the output layer we used sigmoid activation function to map the output to lie in the range [0,1] i.e., actual probability values [38].

As our problem was multilabel problem we applied Binary Cross-Entropy Loss (also known as Sigmoid Cross-Entropy) defined as, $H(y,p) = -\sum_i y_i \log(p_i)$. Where y_i refers to binary indicator zeros which is equal to one for correct class and 0 otherwise and p_i refers to estimated probability for each class. It is used when node activations can be perceived as representing the probability that each hypothesis might be true, i.e., when the output is in the form of a probability distribution [39].

In contrast to softmax cross entropy loss, binary Cross-Entropy is independent in terms of class, i.e., the loss measured for every class is not affected by other classes. Thus, it is useful for multi-label classification, where a parameter belonging to a particular class should not impact the decision for the other class [36]. We used this method to predict proper keywords in recruitment process, by adding these keywords to resumes their rank for recruitment would be increased. For the optimization of loss function (training the neural network), we used stochastic gradient descent-based optimization algorithm (Adam;[40]). Adam is an adaptive learning rate method, which computes individual learning rates for different values and uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network [40].

As the regularization technique to avoid overfitting [39][41] we tested dropout with different rates [0 to 0.5]. In dropout at each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Dropout was applied for all hidden layers and our assessment showed that dropout rate (0.1) yielded better results.

5.1.1. Examining a simple setting

In this model recruitment process is defined as a binary classification algorithm (hire or not), our objection is finding simple rules in job description. We assigned following rules as recruitment conditions.

- if "python" is in resume
- if "university" is in job description applicant should be "master “or “PhD”

The Neural network input: in this model our objective was testing our model on a low dimension and basic vector to generalize it to a more complicated model in the next section. We chose 20 random words among most frequent words of all resumes and job descriptions we already extracted. We coded the resumes and job descriptions as one-hot, then concatenated resume and job descriptions together.

The Neural network labels: the value of word index is set to 1 if adding this word to resume makes resume to be accepted. For example, if resume does not have word "masters" in it and the recruitment algorithm looks for word "masters," then the value for "masters" index in y-train would be 1.

Figure 5-3 illustrates input and label setting of our neural network.

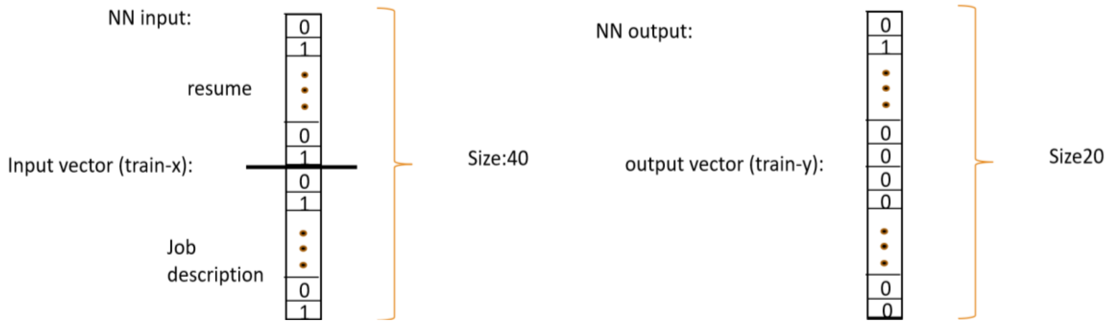


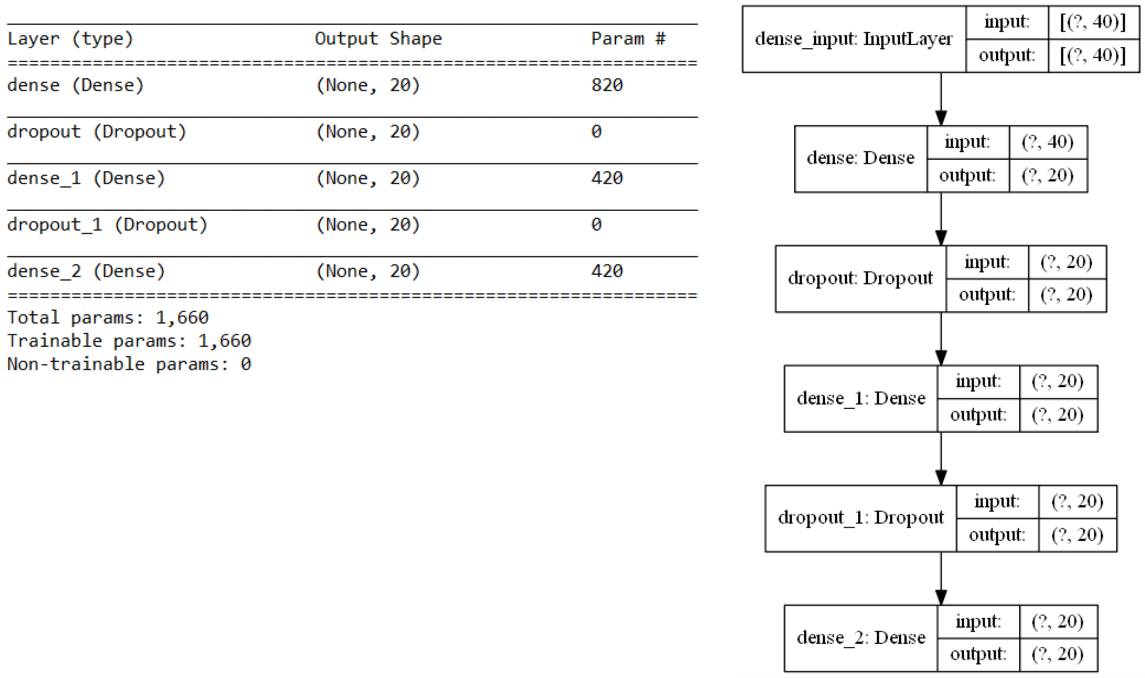
Figure 5-3 input and label setting of neural network

We designed this experiment as following:

- Ranking algorithm: simple rule detection
- Creating the ground truth: we added random words from our dictionary domain (20 words) to resumes and received the response from recruitment algorithm
- Xtrain: 5000 records of 40-dimension vectors, each vector is a resume that is concatenated to job description and is coded by one-hot format.

- Ytrain: 5000 records of 20-dimension vector
- Goal: finding the best 20 keywords that improve the rank of each resume.

To train our neural network model, we split our data into train and test (70% train set, 30% test set), to evaluate our training results, we used validation set approach (we allocated 30% of training set for validation) and trained on 2,450 samples, validate on 1,050 samples. We also set batch size equal to 50 and number of epochs equal to100. Figure 5-4 shows our model architecture that demonstrate each layer input numbers, output numbers and parameters.



```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 20)                820
_________________________________________________________________
dropout (Dropout)            (None, 20)                0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                420
_________________________________________________________________
dropout_1 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                420
=================================================================
Total params: 1,660
Trainable params: 1,660
Non-trainable params: 0
```

Figure 5-4 NN layers configuration in simple setting

To check the performance of our model settings we used recall, precision and F1 over validation set. Our results showed that the model is trained well through epochs (see, Figure 5-5).

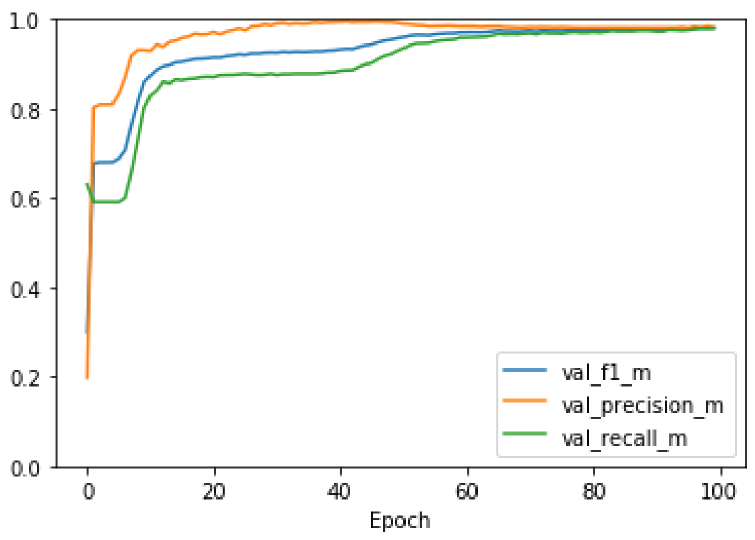


Figure 5-5 NN simple setting model performance

Finally, we need to check how the trained neural network model works for out of sample data (test data). To test the performance of our trained neural network model, we added predicted words in neural network to each related resume and submitted each resume to recruitment algorithm. In the Figure 5-6, it is shown that the success rate of getting hired increases by using suggested NN keywords.

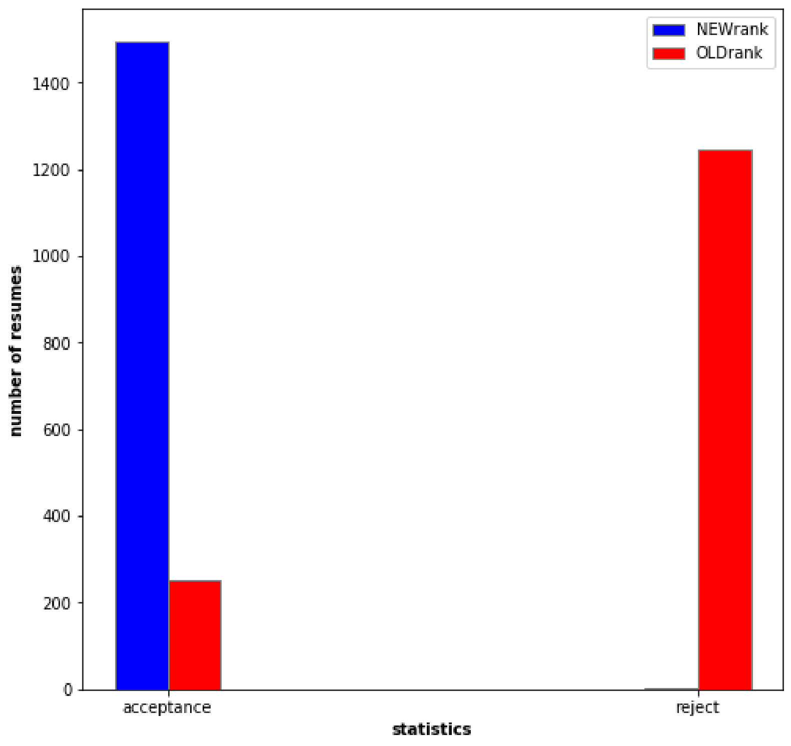


Figure 5-6 simple setting recruitment result

5.1.2. Examining a more complex setting

This model is more complicated and used about 10000 words to model the input resumes vector to have more accurate results.

Recruitment: is based on universal sentence encoder algorithm that we have in white-box. It is for one random job description. And our focus is finding 20 unigram keywords.

Objective: we expect NN determines keywords that improve resumes in recruitment process.

Input: one-hoted 10,000 resumes.

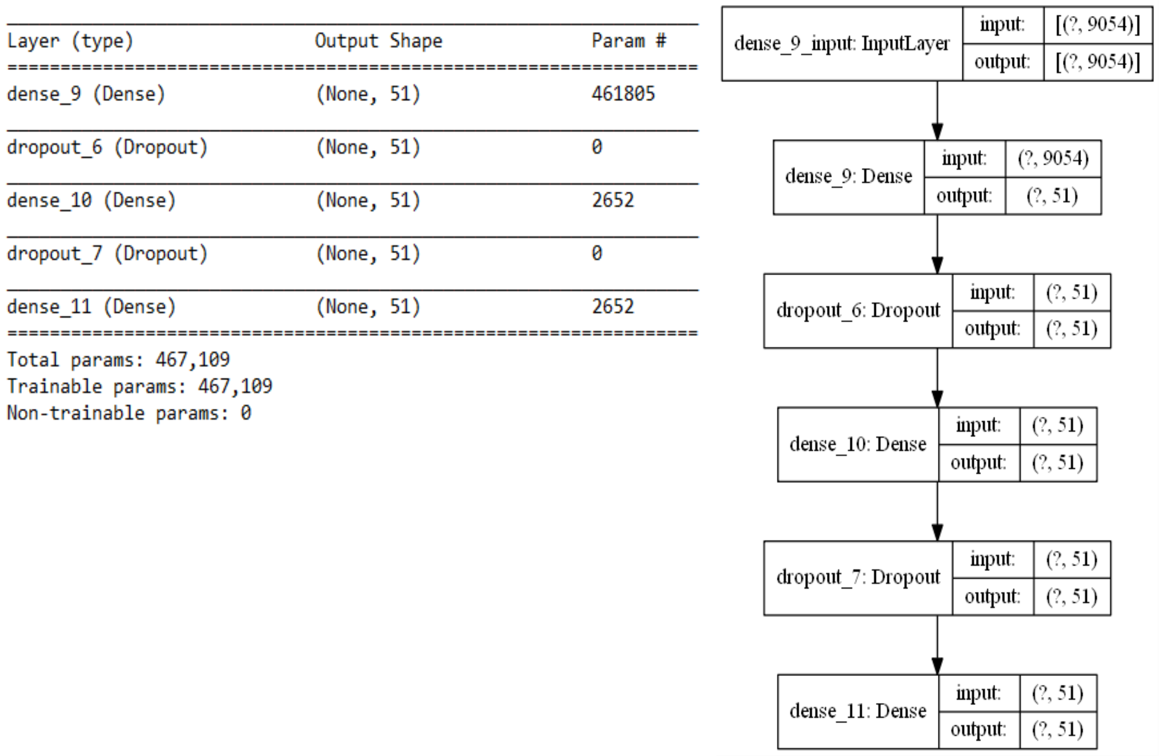
TrainY: one-hoted keywords that improves resume.

- We designed this experiment as following: Ranking algorithm is set to USE.

- Creating the ground-truth: we added random words to resumes and received the response from the black box matching algorithm.
- Xtrain: 10000 records of 9054-dimension vectors, each vector is a resume that is coded by one-hot format.
- Ytrain: 10000 records of 51-dimension vector
- Goal: finding the best 20 keywords that improves the rank of each resumes

To train our neural network model we split our data into train and test (70% train set, 30% test set). To evaluate our training results, we used we allocated 30% of training set for validation and trained on 4,900 samples, validate on 2,100 samples. We also set batch size equal to 50 and number of epochs equal to 100.

Figure 5-7 shows our model architecture that demonstrates each layer input shapes, output shapes and parameters.



```
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 51)                461805
_________________________________________________________________
dropout_6 (Dropout)          (None, 51)                0
_________________________________________________________________
dense_10 (Dense)             (None, 51)                2652
_________________________________________________________________
dropout_7 (Dropout)          (None, 51)                0
_________________________________________________________________
dense_11 (Dense)             (None, 51)                2652
=================================================================
Total params: 467,109
Trainable params: 467,109
Non-trainable params: 0
```

dense_9_input: InputLayer | input: [(?, 9054)] | output: [(?, 9054)]
dense_9: Dense | input: (?, 9054) | output: (?, 51)
dropout_6: Dropout | input: (?, 51) | output: (?, 51)
dense_10: Dense | input: (?, 51) | output: (?, 51)
dropout_7: Dropout | input: (?, 51) | output: (?, 51)
dense_11: Dense | input: (?, 51) | output: (?, 51)

Figure 5-7 NN layers configuration in complex setting

To check the performance of our model settings we used recall, precision and F1 over validation set. Fig. 5-8 shows that this Model is trained well through epochs.

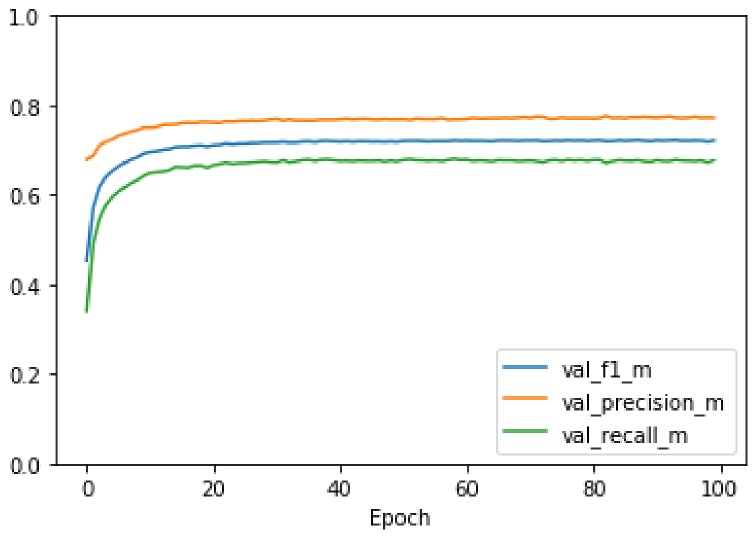


Figure 5-8 NN complex setting model performance

Finally, we set to check how the trained neural network model works with the test set. To test the performance of our trained neural network model, we added predicted words in neural network to each related resume and submitted each resume to recruitment algorithm. In the Figure 5-9 we added predicted words to each resume, we can see that number of getting hired increases significantly by using suggested NN keywords. For example, above 200 resumes rank increased by more than 400 after adding keywords while over 400 resumes rank increased by between 150 and 200 after adding keywords.

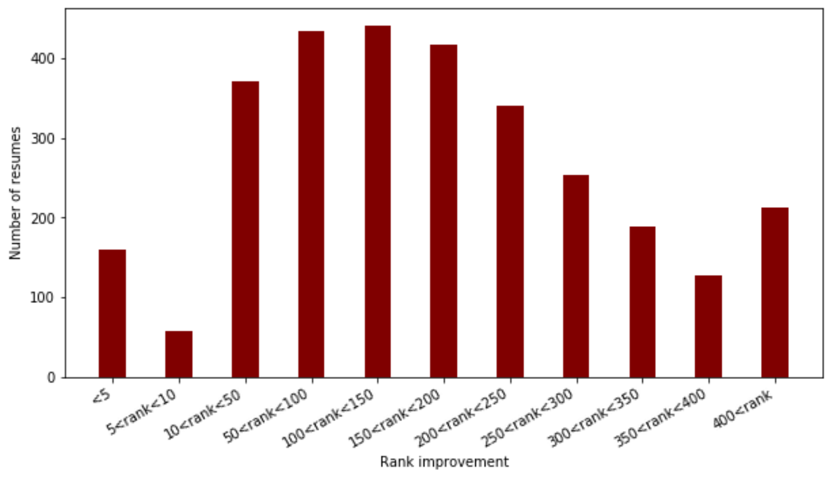


Figure 5-9 complex setting recruitment result

Chapter 6.

Conclusion and Future works

In this project we found that automatic recruitment process, machine learning based, and non-machine learning based, are vulnerable against adversarial examples. We proposed successful adversarial attack in two settings: white-box and black-box. We analysed one hundred real applicant resumes and then thousand job descriptions based on NLP techniques. In this study, we proposed a new approach for keyword extraction based on Universal Sentence Encoder. We extracted meaningful and relevant bag of words keywords of job description in unigram, bigram, and trigram. We ran different experiments for white-box, then we repeated our experiments for two different recruitment approaches. We noticed majority of resumes have significant rank improvements by adding more relevant keywords.

Finally, in black box setting, we proposed multilabel neural network architecture to predict proper keyword for each resume and job description. We applied our model into augmented datasets with five thousand and ten thousand records of resumes consequently for two different settings. After testing our neural network model and predicting the proper keyworks, we added keywords to related resumes and we found over 95% of resumes getting accepted in first setting. For the second setting we examined three thousand resumes as test set. The findings showed that over 85% of those resumes experienced rank improvements above 5 and over 50% of the resumes experienced rank improvements above150.

Future Work

In this study we focused on basic recruitment algorithms. However, in well-known companies, typically more constrains exist and complicated approaches might be used. In our future works we will ask for companies for the approaches they are using for recruitment and compare those approaches efficiencies with proposed schemes. In addition, we will ask for real data and resumes and expand our project to use large scale data. In this regard, we can offer defend solutions alongside analyzing their recruitment algorithm and creating adversarial attack.

In the black-box section, we will optimize our neural network to have better performances. Extensive grid search is one of the works can be performed to set the neural network hyperparameters.

In this study, we considered access to other resumes data set to evaluate adversarial attack in the white box setting. For our future work, we will explore an adversary who has access to the model, but we will limit their access to the resume database.

Bibliography

[1] (Jan 2019). Talent Shortage of Software Developers. https://fullscale.io/talent-shortage-software-developers/

[2] Glassdoor Team . (January 20, 2015). 50 HR & Recruiting Stats That Make You Think. https://www.glassdoor.com/employers/blog/50-hr-recruiting-stats-make-think/

[3] Bika, N. Recruiting costs FAQ: Budget and cost per hire. https://resources.workable.com/tutorial/faq-recruitment-budget-metrics

[4] Shehu, M. A., & Saeed, F. (2016). An adaptive personnel selection model for recruitment using domain-driven data mining. Journal of Theoretical and Applied Information Technology, 91(1), 117. ISSN 1992-8645

[5] Zaman, E. A. K., Kamal, A. F. A., Mohamed, A., Ahmad, A., & Zamri, R. A. Z. R. M. (2018, August). Staff Employment Platform (StEP) Using Job Profiling Analytics. In International Conference on Soft Computing in Data Science (pp. 387-401). Springer, Singapore. http://doi-org-443.webvpn.fjmu.edu.cn/10.1007/978-981-13-3441-2_30

[6] Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137-1155.

[7] Fernández-Reyes, F. C., & Shinde, S. (2019). CV Retrieval System based on job description matching using hybrid word embeddings. Computer Speech & Language, 56, 73-79. https://doi.org/10.1016/j.csl.2019.01.003

[8] Keenan, P., McGarraghy, S., McNamara, C., Phelan, M., & Schools, U. B. (2004). Human resource management DSS. In International Conference DSS2004 (pp. 525-534). Corpus ID: 1638639

[9] Faliagka, E., Ramantas, K., Tsakalidis, A., & Tzimas, G. (2012, May). Application of machine learning algorithms to an online recruitment system. In Proc. International Conference on Internet and Web Applications and Services. ISBN: 978-1-61208-200-4

[10] Apatean, A., Szakacs, E., & Tilca, M. (2017). MACHINE-LEARNING BASED APPLICATION FOR STAFF RECRUITING. Acta Technica Napocensis, 58(4), 16-21.

[11] Koh, M. F., & Chew, Y. C. (2015). Intelligent job matching with self-learning recommendation engine. Procedia Manufacturing, 3, 1959-1965. https://doi.org/10.1016/j.promfg.2015.07.241

[12] Khairina, D. M., Asrian, M. R., & Hatta, H. R. (2016, October). Decision support system for new employee recruitment using weighted product method. In 2016 3rd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE) (pp. 297-301). IEEE. DOI: 10.1109/ICITACEE.2016.7892459

[13] Sarda, V., Sakaria, P., & Nair, S. (2014). Relevance ranking algorithm for job portals. International Journal of Current Engineering and Technology, 4(5), 3157-3160.

[14] Ren, K., Zheng, T., Qin, Z., & Liu, X. (2020). Adversarial attacks and defenses in deep learning. Engineering. https://doi.org/10.1016/j.eng.2019.12.012

[15] Goodfellow, I, Papernot, N, Huang, S, Duan, R, Abbeel P, Clark ,A. (February 24, 2017). Attacking Machine Learning with Adversarial Examples. Goodfellow, 2013. https://openai.com/blog/adversarial-example-research/

[16] Morris, J, (Aug 28 2020). What are adversarial examples in NLP?. https://towardsdatascience.com/what-are-adversarial-examples-in-nlp-f928c574478e

[17] Alshemali, B., & Kalita, J. (2020). Improving the reliability of deep neural networks in NLP: A review. Knowledge-Based Systems, 191, 105210. https://doi.org/10.1016/j.knosys.2019.105210

[18] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. Information, 10(4), 150. arXiv:1904.08067

[19] Hassanzadeh Y, Ghazvinian M, Abdi A, Baharvand S, Jozaghi A (2020) Prediction of short and long-term droughts using artificial neural networks and hydro-meteorological variables. arXiv:2006.02581 [physics.ao-ph]

[20] Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. arXiv 2013, arXiv:1301.3781

[21] Pennington, J.; Socher, R.; Manning, C.D. Glove: Global Vectors forWord Representation. In Proceedings of the 2014 Conference on Empirical Methods in

Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; Volume 14, pp. 1532–1543. DOI: 10.3115/v1/D14-1162

[22] Ganegedara, T. (May 5, 2019). Intuitive Guide to Understanding GloVe Embeddings. https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

[23] Hui, J. (Oct 21, 2019). NLP — Word Embedding & GloVe. https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6

[24] Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching word vectors with subword information. arXiv 2016, arXiv:1607.04606.

[25] Subedi, N. (Jul 6, 2018). FastText: Under the Hood. https://towardsdatascience.com/fasttext-under-the-hood-11efc57b2b3

[26] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[27] Zaki Rizvi, M.(September 25, 2019). Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework. https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/

[28] Rose, S., Engel, D., Cramer, N., & Cowley, W. (2010). Automatic keyword extraction from individual documents. Text mining: applications and theory, 1, 1-20. https://doi.org/10.1002/9780470689646.ch1

[29] Google. (Dec 16 2020). universal-sentence-encoder. https://tfhub.dev/google/universal-sentence-encoder/4

[30] (2017). STSbenchmark. http://ixa2.si.ehu.es/stswiki/index.php/STSbenchmark

[31] Nielsen, M. A., 2015: Neural Networks and Deep Learning. Determination Press, http://neuralnetworksanddeeplearning.com/

[32] Rajaraman, A.; Ullman, J.D. (2011). "Data Mining" (PDF). Mining of Massive Datasets. pp. 1–17. doi:10.1017/CBO9781139058452.002. ISBN 978-1-139-05845-2.

[33] Siva, S. Jun 12 2020. “Sklearn’s TF-IDF” vs “Standard TF-IDF”. https://towardsdatascience.com/how-sklearns-tf-idf-is-different-from-the-standard-tf-idf-275fa582e73d

[34] Harris, David and Harris, Sarah (2012-08-07). Digital design and computer architecture (2nd ed.). San Francisco, Calif.: Morgan Kaufmann. p. 129. ISBN 978-0-12-394424-5.

[35] Silipo, R. Nov 21, 2019. Text Encoding: A Review. https://towardsdatascience.com/text-encoding-a-review-7c929514cccf

[36] Sadowski, P. (2016). Notes on backpropagation. homepage: https://www. ics. uci. edu/pjsadows/notes. pdf (online).

[37] Vinod, N, Hinton,G , (2010). Rectified Linear Units Improve Restricted Boltzmann Machines (PDF). ICML. ISBN:9781605589077

[38] Han, Jun; Morag, Claudio (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Mira, José; Sandoval, Francisco (eds.). From Natural to Artificial Neural Computation. Lecture Notes in Computer Science. 930. pp. 195–201. doi:10.1007/3-540-59497-3_175. ISBN 978-3-540-59497-0.

[39] Goodfellow, I., Y. Bengio, and A. Courville, (2016). *Deep Learning*. MIT Press. pp. 180–184. *ISBN 978-0-26203561-3*.

[40] Diederik, Kingma; Ba, Jimmy (2014). "Adam: A method for stochastic optimization". arXiv:1412.6980

[41] Hinton, Geoffrey E.; Srivastava, Nitish; Krizhevsky, Alex; Sutskever, Ilya; Salakhutdinov, Ruslan R. (2012). " Improving neural networks by preventing co-adaptation of feature detectors". arXiv:1207.0580

[42] Krizhevsky, A., I. Sutskever, and G. E. Hinton, 2012: Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105). Available online at: http://www.csri.utoronto.ca/~hinton/absps/imagenet.pdf.

[43] Conneau, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. arXiv preprint arXiv:1705.02364.

[44] Cer, D., Yang, Y., Kong, S. Y., Hua, N., Limtiaco, N., John, R. S., ... & Sung, Y. H. (2018). Universal sentence encoder. arXiv preprint arXiv:1803.11175.

[45] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., & Tygar, J. D. (2011, October). Adversarial machine learning. In Proceedings of the 4th ACM workshop on Security and artificial intelligence (pp. 43-58). https://doi.org/10.1145/2046684.2046692

[46] Devlin, J., M. W. Chang, K. Lee, and K. Toutanova, 2018: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. https://arxiv.org/abs/1810.04805.

[47] Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa, 2011: Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research, 12:2493-2537. Available online at: https://www.jmlr.org /papers/volume12 /collobert11a /collobert11a.pdf.