Attacks against Ranking Algorithms with Text Embeddings: A Case Study on Recruitment Algorithms

Anahita Samadi and **Debapriya Banerjee** and **Shirin Nilizadeh**
The University of Texas at Arlington
{anahita.samadi,debapriya.banerjee2}@mavs.uta.edu,
and shirin.nilizadeh@uta.edu

Abstract

Recently, some studies have shown that text classification tasks are vulnerable to poisoning and evasion attacks. However, little work has investigated attacks against decision-making algorithms that use text embeddings, and their output is a ranking. In this paper, we focus on ranking algorithms for the recruitment process that employ text embeddings for ranking applicants' resumes when compared to a job description. We demonstrate both white-box and black-box attacks that identify text items that, based on their location in embedding space, have a significant contribution in increasing the similarity score between a resume and a job description. The adversary then uses these text items to improve the ranking of their resume among others. We tested recruitment algorithms that use the similarity scores obtained from Universal Sentence Encoder (USE) and Term Frequency–Inverse Document Frequency (TF-IDF) vectors. Our results show that in both adversarial settings, on average the attacker is successful. We also found that attacks against TF-IDF are more successful compared to USE.

1 Introduction

Recently some studies have shown that text classification tasks are vulnerable to poisoning and evasion attacks (Liang et al., 2018; Li et al., 2018; Gao et al., 2018; Grosse et al., 2017). For example, some works have shown that an adversary can fool toxic content detection (Li et al., 2018), spam detection (Gao et al., 2018) and malware detection (Grosse et al., 2017) by modifying some text items in the adversarial examples. A recent work (Schuster et al., 2020) showed that applications that rely on word embeddings are vulnerable to *poisoning attacks*, where an attacker can modify the corpus that the embedding is trained on, i.e., Wikipedia and Twitter posts, and modify the meaning of new or existing words by changing their locations in the embedding space. In this work, however, we investigate a new type of attack, i.e., *rank attack*, when the text application utilizes text embedding approaches. In this attack, the adversary does not poison the training corpora but tries to learn about the embedding space, and how adding some keywords to a document can change the representation vector of it, and based on that tries to improve the ranking of the adversarial text document among a collection of documents.

As a case study, we focus on ranking algorithms in a recruitment process scenario. Recruitment process is a key to finding a suitable candidate for a job application. Nowadays, companies use ML-based approaches to rank resumes from a pool of candidates (Sumathi and Manivannan, 2020; Roy et al., 2020). One naive approach for boosting the ranking of a resume can be adding the most words and phrases to his/her resume from the job description. However, this is not the best approach all the time, because: First, the attacker must add a specific *meaningful* word to their resume. For example, they cannot claim they are proficient in some skills while they are not. Second, adding any random keyword from the job description to the resume does not always increase the similarity between the resume and job description. Instead, we demonstrate that the adversary can learn about the influential words and phrases that can increase the similarity score and use this knowledge and decide about the keywords or phrases to be added to their resume.

Therefore, in this work, we investigate: (1) *How can an adversary utilize the text embedding space to extract words and phrases that have a higher impact on the ranking of a specific document (here resume)?* and, (2) *How can an adversary, with no knowledge about the ranking algorithm, modify a text document (here resume) to improve its ranking among a set of documents?*

While we focus on the recruitment application, the same approaches can be employed on

Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 457–467

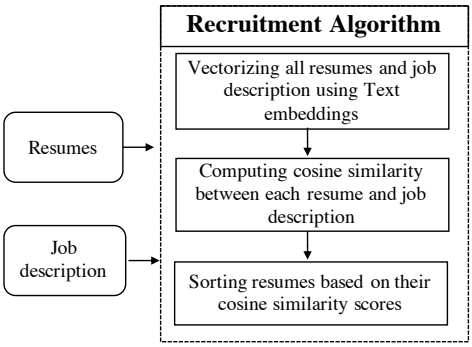


Figure 1: Ranking algorithm for recruitment

other ranking algorithms that use text embeddings for measuring the similarity between text documents and ranking them. For example, such ranking algorithms are applied as part of summarization (Rossiello et al., 2017; Ng and Abrecht, 2015) and question and answering systems (Bordes et al., 2014; Zhou et al., 2015; Esposito et al., 2020).

We consider both white-box and black-box settings, depending on the knowledge of the adversary about the specific text embeddings approach that is used for the resume ranking. In white-box settings, we propose a novel approach which utilizes the text embedding space to extract words and phrases that influence the rankings significantly. We consider both Universal Sentence Encoder (USE) (Cer et al., 2018) and TF-IDF as the approaches for obtaining the word vectors. USE computes embeddings based on Transformer architecture (Wang et al., 2019a), and it captures contextual information while on the other hand TF-IDF does not capture the contextual information. In the black-box setting, we propose a neural network based model that can identify the most influential words/phrases without knowing the exact text embedding approach that is used by the ranking algorithm.

2 System and Threat Model

The recruitment process helps to find a suitable candidate for a job application. Based on Glassdoor statistics, the average job opening attracts approximately 250 resumes (Team), and a recent survey found that the average cost per hire is just over $4,000 (Bika).

To Limit work in progress many companies use Machine Learning to have more efficiency in ranking resumes.A successful approach to rank resumes is calculating similarity among resume and job description leveraging NLP techniques (Excellarate). In this study, the recruitment algorithm is based on a ranking algorithm. This algorithm takes input from the resume and the job description and finds similarities based on their matching score. We use universal sentence encoder (USE) as the text embedding approach to vectorize each resume and the job descriptions. As it is shown in Figure 1, our algorithm includes three steps: (1) all resumes and the job description are vectorized using USE text embedding approach; (2) the similarity between each job description and an individual resume is computed. Cosine similarity is used as a metric to compute the similarity score; and (3) the resumes are sorted based on their similarity scores computed in the second step in such a way that the resume with highest similarity score appears at the top of the list and on the other hand the resume with least similarity score appears at bottom.

Threat Model. Adversaries have a huge motivation to change the rankings provided by some algorithms, when they are used for decision-making, e.g., for recruitment purposes. Adjusting a resume based on the job description is a well-known approach for boosting the chance of being selected for the next rounds of recruitment (Team, 2021). In this work, we show how an adversary can automatically generate adversarial examples specific to a recruitment algorithm and text embeddings. We define this attack as a *rank attack*, where the adversary adds some words or phrases to its document to improve its ranking among a collection of documents. We consider white-box and black-box settings. In a white-box setting, we assume the attacker has complete knowledge about the ranking algorithm. In a black-box setting, however, the attacker has no knowledge of the recruitment process but has limited access to the recruitment algorithm and can test some resumes against the algorithm.

3 Background

USE Text Embedding. This embedding has been used to solve tasks, such as semantic search, text classification, question answering (Rossiello et al., 2017; Ng and Abrecht, 2015; Bordes et al., 2014; Zhou et al., 2015; Esposito et al., 2020). USE uses an encoder to convert given text to a fixed-length 512-dimensional vector. It has been shown that after embedding sentences, sentences that have closer meaning carry out higher cosine similarity (an, 2018). We used USE pretrained model, which is trained on the STS (Semantic Textual Sim-

ilarity) benchmark (Agirre).

TF-IDF. TF-IDF or term frequency-inverse document frequency is a widely used approach in information retrieval and text mining. TF-IDF is computed as a multiplication of *TF*, the frequency of a word in a document, and *IDF*, the inverse of the document frequency.

4 Data Collection

We collected 100 real public applicant resumes from LinkedIn public job seeker resumes, GitHub, and personal websites. To have an 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 (bachelor, master and Ph.D. with equal distribution), skills, experiments (entry level, mid level, senior level), and gender (around fifty percent men and fifty percent women). We also developed a web scraper in python to extract computer science jobs from the Indeed website.[1] Our dataset includes over 10,000 job descriptions, extracted randomly from cities in the USA. We randomly chose 50 job descriptions and used them in our experiments.

For black-box settings, our neural network architecture needs a huge amount of data to be trained on. To have enough training sets we augmented our data for our models. For a simple setting model, we created 5,000 records by concatenating 100 resumes to 50 selected job descriptions to augment our data for recruitment algorithms. Note that the adversary only needs to obtain or create a set of resumes and they do not need to use the job descriptions. For a more complex setting, we split resumes into half, then joined the upper part of each resume to other resumes' lower parts. With this approach, we could maintain the structure of each resume and each resume could have common resume information, such as skills, education, work experince, etc. We did this procedure for all possible combinations and created a database with 10,000 resumes.

5 White-Box Setting and a Recruitment Algorithm that Employs USE

In white-box settings, we assume the adversary has knowledge about the recruitment process and the use of universal sentence encoder (USE) or term frequency–inverse document frequency (TF-IDF) for obtaining the embedding vectors. We propose a novel approach which utilizes the knowledge about the text embedding space, here USE, to identify keywords in the job description that can be added to the adversarial resume and increase its similarity score with the job description. Our approach as it is shown in Figure 2 consists of three phases: in *Phase 1*, the adversary tries to identify the important words in job description; in *Phase 2*, the adversary tries to rank the effectiveness of those words based on its own resume, i.e., identifying those words that can increase the similarity between his/her resume and the job description. After identifying the most effective words, then in *Phase 3*, the adversary modifies its resume based on his/her expertise and skill set and decides to add N of those identified words and phrases. Note that in this attack scenario the adversary does not need to have any knowledge about the other candidates' resumes. The details of phase 1 and 2 in this attack are depicted in Algorithm 1 and Algorithm 2, respectively, and we explain them in the following.

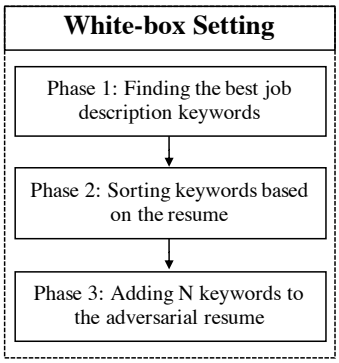


Figure 2: White-box setting

Algorithm 1 Job Description Keywords Extraction

Input: Job Description document

Output: An list of keywords in ascending order based on their similarity score

procedure PHASE1($OrginalJob$)
  $OrginalJob \leftarrow FilterStopWords(OrginalJob)$
  $Tokens \leftarrow Tokenize(OrginalJob)$
  $USE_{Job} \leftarrow USE(OrginalJob)$

  $Token\_Sim \leftarrow \{\}$
  for $Token_i \in Tokens$ **do**
    $newJob \leftarrow DeleteToken(JobDescription, Token_i)$
    $USE_{newJob} \leftarrow USE(newJob)$
    $Token\_Sim \leftarrow Token\_Sim + \{Token_i, CosSim(USE_{Job}, USE_{newJob})\}$
  end for

  return $AscendingSortByValue(Token\_Sim)$

[1] https://www.indeed.com/

5.1 Phase one: Identifying the Influential Keywords in the Job Description

In this phase, the adversary focuses on identification of the most important keywords from job description. Based on the use of cosine similarity between the vectors of resumes and job descriptions in the recruitment algorithm (depicted in Figure 1), the highest similarity can be achieved if the resume is the same as the job description. We propose to remove words/phrases from the job description and then examine its impact on the similarity score. A substantial decrease in the similarity score when a specific word/phrase is removed demonstrates the importance of the keyword in the word embedding space corresponding to this job description. In the white-box setting, the adversary is aware of the details of algorithms. Therefore, they employ USE text embedding to obtain the vector and use cosine similarity to calculate the similarity scores.

In addition, to examine the importance of phrases instead of individual words, the adversary can try to remove phrases with one word (unigram), two words (bigram), three words (trigram), etc., and then compute the similarity score. Algorithm 1 shows the details of *phase 1* which consists of the following steps: *(1) Text pre-processing:* the keywords of the job description are obtained as a bag of words, and the stop words are removed to lower the dimensional space. *(2) USE embedding:* The embedding vector is obtained for the original job description using universal sentence encoder (USE). *(3) Token removal:* measures the importance of each word in the job description. A single token is deleted from the job description, and the new job description is created. Next, the embedding vector for $NewJob$ is obtained by passing to USE. *(4) Scoring keywords:* Cosine similarity is calculated to measure the similarity between two vectors USE_{newJob} and $USE_{OrginalJob}$. *In this regard, lower cosine similarity expresses the fact that the deleted token has caused an impressive change in the job description content, therefore it might be an important keyword. (5) Repetition:* Steps three and four are repeated for all tokens in the job description. This procedure provides a dictionary, where the keys are tokens, and the values are their corresponding cosine similarity scores. *(6) Sorting keywords:* Finally, extracted keywords are sorted based on the similarity score in ascending order.

N-gram phrases: Moreover, we extended the code in Algorithm 1 so that it can also identify the influential phrases, i.e., bigrams and trigrams. In that case, in each repetition, instead of one individual word, repeatedly 2 or 3 neighbor words are removed from the job description and the similarity score is computed.

5.2 Phase two: Re-sorting the Influential Keywords based on a Specific Resume

The previous phase helps identify the words and phrases in the job description that in USE embedding space have a higher impact on providing a larger similarity score. However, each resume is unique and adding the most influential words obtained from the job description might not have the same impact on all the resumes. In *Phase 2*, we try to identify the best words and phrases that can boost the similarity score between a specific resume and job description. Algorithm 2 shows the details of *phase 2* which consists of the following steps:

(1) Adding Keywords: A keyword from the list of fifty keywords obtained from *phase 1* is added to the adversarial resume. *(2) Obtaining the Embedding Vector*: The embedding vector for the adversarial resume is obtained using USE. *(3) Calculating the Similarity:* The cosine similarity between adversarial and job description is computed. *Higher cosine similarity expresses the fact that the deleted token caused an impressive change in the job description contents, therefore, it might be an important keyword. (4) Repetition:* These steps are repeated for all fifty keywords. This procedure provides a dictionary, where the keys are the fifty keywords, and the values are their corresponding cosine similarity scores. The list of keywords are sorted based on their cosine similarity scores.

N-gram phrases: Algorithm 2 is also extended to get the sorted list of bigrams or trigrams, and added to the adversarial resume.

5.3 Experimental Setup

We implemented the *rank attack* in a white-box setting and tested all combinations of 100 resumes and 50 job descriptions. The attack is shown in Algorithm 3. Then in each experiment, we assumed that a resume is adversarial and therefore the best keywords for that specific resume are added to it. To investigate the impact of the number of keywords on the ranking of the resume, we tested with $n \in (1, 2, 5, 10, 20, 50)$ of keywords. We also repeated these experiments for bigram and trigram

Algorithm 2 Resorting the extracted job description keywords based on a resume

Input: Job description document, Resume document, A list of tokens

Output: An ordered list of keywords

procedure PHASE2($JobDescription$, $Resume$, $OrderedTokens$)
  $USE_{JobDescription} \leftarrow USE(JobDescription)$
  $Keyword\_Sim \leftarrow \{\}$
  for $keyword_i \in OrderedTokens$ **do**
    $adversarialResume \leftarrow AddToken(Resume, keyword_i)$
    $USE_{advResume} \leftarrow USE(adversarialResume)$
    $Keyword\_Sim \leftarrow Keyword\_Sim + \{Keyword_i, CosSim(USE_{advResume}, USE_{JobDescription})\}$
  end for
  return $SortByValue(Keyword\_Sim)$

phrases. Note that in practice, the adversary does not need to, and cannot obtain the ranking of its resume among all other resumes. However, in our experiments we show how using this attack they can improve their position.

Algorithm 3 White Box Adversarial Attack

Input: Job Description documents, Resume documents

Output: Ranking

procedure WHITEBOX($Jobs$, $Resumes$)
  for $job_i \in Jobs$ **do**
    $Tokens \leftarrow Phase1(job_i)$
    for $resume_j \in Resumes$ **do**
      $Rank \leftarrow GetRanking(job_i, resume_j, Resumes)$
      $words \leftarrow Phase2(JobDescription, Resume, Tokens)$
      for $n \in (1, 2, 5, 10, 20, 50)$ **do**
        $AdvResume \leftarrow AddWords(resume_j, n, words)$
        $Resumes2 \leftarrow Resumes - resume_j$
        $Resumes2 \leftarrow Resumes2 + AdvResume$
        $Rank_{new} \leftarrow Ranking(job_i, Resumes2)$
        $RankChange_{job_i, resume_j, n} \leftarrow Rank_{new} - Rank$
      end for
    end for
  end for
  return $RankRankChange$

5.4 Experimental Results

Figure 3a shows the histogram of average rank improvements for 100 resumes and 50 random job descriptions. All resumes had rank improvement, and in most of them the rank improvement is significant. For example, on average an adversarial resume moved up 16 positions in rank improvement, about 6 resumes had an average of moving up 28 ranking positions, and more than 65 resumes moved up more than 10 ranking positions.

Figure 3b shows the average rank improvement based on the number of words or phrases (bigrams and trigrams) added to the adversarial resume. We see a similar trend when adding unigrams, bigrams, and trigrams, i.e., adding more words and phrases increases the average rank improvement. For example, while adding 2 bigrams improves the ranking of the adversarial resume on average by 6, adding 20 bigrams improves the ranking of the adversarial resume by 30, among 100 resumes. However, interestingly we see that adding too many words/phrases might not have the same effect, e.g., adding 50 bigrams shows a rank improvement by only about 28. This shows there might be an optimal number of words/phrases that can help ranking of a document.

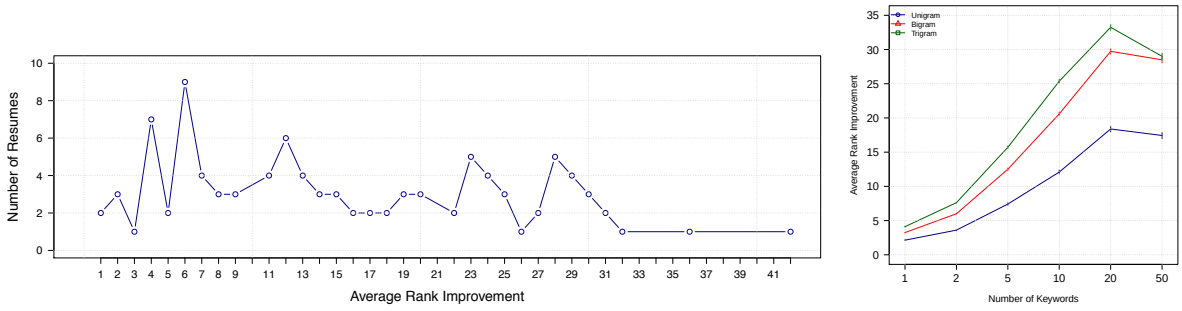
Comparing the addition of unigrams, bigrams, and trigrams, we see that trigrams provide better rank improvement. For example, adding 10 unigrams, bigrams, and trigrams, we see a rank improvement of 12, 21 and 25, respectively. This finding can be explained by USE being a context-aware embedding approach, which takes into account the order of words in addition to their meaning.

6 White-box Setting and a Recruitment Algorithm that Employs TF-IDF

We also investigate the effectiveness of the attacks in white-box setting, when the recruitment algorithm uses TF-IDF vectors to compute the similarity between resumes and job descriptions. The attack is the same. The only difference is in the recruitment algorithm. Since the adversary has the knowledge that the recruitment algorithm uses TF-IDF vectors, then they compute TF-IDF vectors instead of USE vectors.

6.1 Experimental Setup

For these experiments, we implemented the recruitment algorithm that ranks resumes based on the similarity scores that are calculated between the TF-IDF vectors of resumes and the job description. We implemented the rank attack and tested on all combinations of the 100 resumes and 50 job descriptions. For each job description, we first obtained the ranking of the original resumes, and then in each experiment, we assumed that a resume is adversarial and therefore the best keywords for that specific resume are added to it. To investigate the impact of the number of keywords on the ranking of the resume, we tested with $n \in (1, 2, 5, 10, 20, 50)$ of keywords. We also repeated these experiments for bigram and trigram phrases.



(a) Histogram of average rank improvement

(b) Average rank improvement

Figure 3: Average rank improvement for 100 resumes and 50 job descriptions, in white-box setting when recruitment algorithm employs USE embeddings

6.2 Experimental Results

Figure 4a shows the histogram of average rank improvements for 100 resumes and 50 random job descriptions. Most resumes had rank improvement and in most of them rank improvement was significant. For example, on average, an adversarial resume moved up about 25 ranking positions, and more than 85 resumes moved up more than 10.

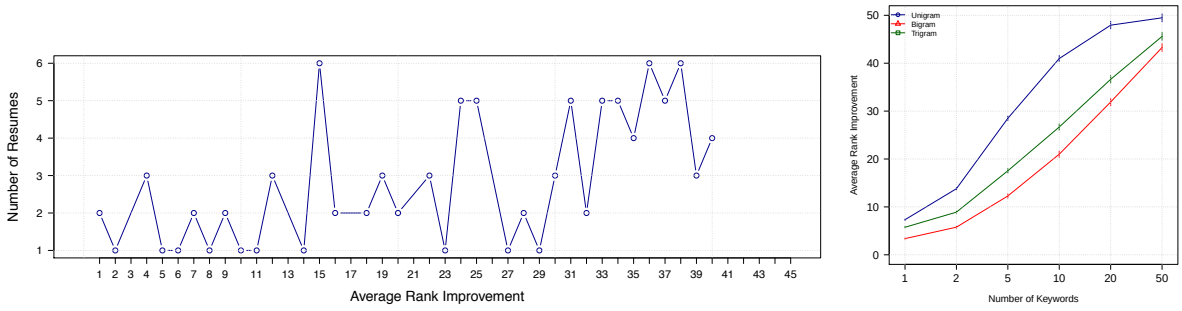
In Figure 4b, we investigated the effect of adding bigram and trigram bags of words on average rank improvement. Interestingly, in contrast to our results for recruitment algorithms that employ USE embedding, in these experiments we achieved better results for unigrams compared to bigrams and trigrams. Note that TF-IDF approach is only based on word similarity and does not consider the context. This might explain the results. No matter what n-gram is used, by increasing the number of keywords, the ranks also improve. Comparing results in Figure 3b and Figure 4b also shows that it is easier for the adversary to attack a recruitment algorithm that employs TF-IDF compared to USE, as the rank improvement is larger for attacks against the the recruitment algorithm that employs TF-IDF, specially in the case of unigrams. For example, by adding 20 keywords to attack the recruitment algorithm that employs *USE*, we observe an average rank improvement of 18, 29, and 33 for unigram, bigram and trigram, respectively. However, by adding 10 keywords to attack the recruitment algorithm that employs *TF-IDF*, they are 48, 32, and 37 for unigram, bigram and trigram, respectively.

7 Black-Box Setting

In the back-box setting, an adversary does not have access to the model specification but has access to the recruitment algorithm as an oracle machine, which receives an input, e.g., here a resume, and then generates a response, e.g., here accept/ reject, or the ranking among all the resumes in its pool. Therefore, the adversary is able to send some resumes to the recruitment algorithm and obtain the output for them, which then can be used to train a neural network model. This model then identifies the features (i.e., keywords) that help resumes to be accepted or obtain better ranking. The adversary then can boost its chance of acceptance by adding some or all of the identified keywords.

The input vector of our neural network model is the resumes, and the output is a vector that demonstrates the best keywords. In our experiments, we examine attacks against two recruitment algorithms: a binary classifier, which label a resume as *accept* or *reject*, and a ranking algorithm that provides a ranking based on similarity scores between the job description and the resumes, where they are vectorized by USE embeddings. The attacks against both algorithms have two phases: *phase 1: pre-processing* which prepares the data for the neural network, and *phase 2:* a neural network model.

Phase 1: Pre-processing. To provide an acceptable format for the neural network input, we applied one-hot encoding (Harris and Harris, 2012) following these steps: *(1) Tokenization:* The job description and resumes are tokenized and a dictionary of words is created, where key is the token/word, and value is the frequency of the words in these documents. *(2) Vectorization of Resumes:*



(a) Histogram of average rank improvement

(b) Average rank improvement

Figure 4: Average rank improvement in white-box setting when recruitment algorithm employs TF-IDF vectors

Tokens/words are encoded as a one-hot numeric array, where for each word a binary column is created. As a result, a sparse matrix is returned, which represents the resumes in rows and the words in columns. If a resume includes some word, the entry for that row and column is 1, otherwise it is 0.

Neural Network Architecture. Neural networks have been applied in many real-world problems (Nielsen, 2015; Salakhutdinov, 2014; Hassanzadeh et al., 2020; Ghazvinian et al., 2021; Krizhevsky et al., 2012). We propose a deep neural network architecture which consists of an input layer, three dense layers as hidden layers, and an output layer representing the labels. In the output vector, ones indicate an index of words in the dictionary that adding them to a resume will increase the rank of the resume. For the first two hidden layers we used rectified linear unit (ReLU) (Nair and Hinton, 2010) as the activation function. Due to their unique formulations, ReLUs provide faster training and better convergence relative to other activation functions (Nair and Hinton, 2010). 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 (Han and Moraga, 1995). As our problem was a multilabel problem, we applied binary cross-entropy loss. 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.

For the optimization of loss function (training the neural network), we used a stochastic gradient descent-based optimization algorithm (Adam; (Kingma and Ba, 2015)). For the regularization technique, to avoid over-fitting (Salakhutdinov, 2014), we tested dropout with different rates [0 to 0.5]. Dropout was applied for all hidden layers, and our assessment showed that the dropout rate (0.1) yielded better results.

7.1 Experiments for the Binary Classifier

This is a simpler model, where the recruitment process is defined as a binary classification algorithm, and a resume is accepted based on some rules. In our experiments, we defined simple rules, e.g., if *python* as a skill is in the resume. After tokenization of the resume and the job description, instead of generating the one-hot encoding for all the words obtained, we chose 20 of the most frequent words of all resumes and job descriptions. Also, our proposed neural network will predict a maximum number of 20 keywords that will enhance resume rank. This is to test with a low dimension vector. We then concatenated the vectors of the resume and job description.

Creating the groundtruth dataset. We employed two steps: (1) Xtrain: 5000 records of 40-dimension vectors, each vector is a resume and coded by one-hot format. In Section 4 we explained the method for generating these resumes. (2) Ytrain: 5000 records of 20-dimension vectors. If the value of a word index is set to one then adding this word to resume makes the resume be accepted.

The attack can be more successful if the adversary obtains a larger training dataset. This might be seen as a bottle-neck for this type of attack, since the adversary needs to create/ obtain many resumes and submit them for evaluation without being suspicious. However, this scenario can be practical. First of all, it is easy to obtain resumes online and the attacker can obtain relevant resumes and use them. Also, the adversary does not need to send all the resumes for the target position, but they can divide and send them to multiple positions in the

same company assuming that the company uses the same algorithm or set of rules for their recruitment for all their positions.

Model Training. 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 to 100. Our results show that the model is trained well through epochs. After 50 epochs, the recall, precision and F1-score of this model over the validation set reached their maximum, which are 0.6, 0.7 and 0.82, respectively.

We examined the performance of the trained neural network model on test data. We added predicted words in the neural network to each related resume and submitted each resume to the recruitment algorithm and obtained the response from the recruitment oracle in the form of a binary value. Figure 6 shows that the success rate of getting accepted significantly increases by using suggested keywords. While without adding the keywords, the acceptance rate was 20%, after the attack the acceptance rate increased to about 100%.

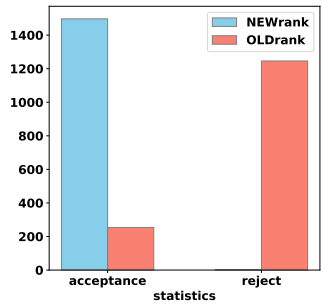


Figure 5: Attack success in black-box simple setting

7.2 Experiments for the Ranking Algorithm

This model is more complicated where the recruitment process is defined as a ranking algorithm. The goal is similar to the previous setup, i.e., identifying the best 50 words that can improve the ranking of a resume. For this setting, we considered higher dimension input vectors with 10,000 words.

Creating the groundtruth dataset. We employed these steps: (1) Xtrain: 10,000 records of 9054-dimension vectors, each vector is a resume that is coded by one-hot format. The 9054-dimension vector represents all the words (after cleaning and removing stop words) in all the resumes. (2) Ytrain: 10,000 records of 50-dimension vectors. For creating the output vectors, we selected a random job description, and employed the same technique proposed in white-box approach (Section 5.1) and identified the 50 most influential words in the job description. We assumed the recruitment algorithm uses the USE embedding, however the attacker can try this approach with other text embedding algorithms, or even choose the most frequent words. Therefore, in practice, the adversary does not need to know about the recruitment algorithm. We used 50 words to be consistent with our white-box attack and have enough choices of words. However, this is a parameter that can be defined during the time of attack. After identifying the influential words for the target job description, the adversary adds each of the words to each of the resumes in the training set, and then queries the black-box algorithm asking if the position of resume is improved. If it is improved then the value for that word and that resume in Ytrain would be 1, otherwise it would be 0. Therefore, the output label is an encoded vector by one-hot format.

Model Training. To train our neural network model, we split our data into 70% train set and 30% test set. To evaluate our training results, we allocated 30% of training set for validation and trained on 4,900 samples, and validated on 2,100 samples. Our results showed that the model is trained well through epochs. After 10 epochs, the recall, precision and F1-score of this model over the validation set reached their maximum values, i.e., 0.62, 0.68 and 0.75, respectively.

To test the performance of our trained neural network model, we ran it on our testing set and obtained the 50-dimention vector for each resume in it. This vector shows which words among the 50 influential words help the resume for this job description. Then, we added the identified words to the resumes, submitted each resume to the recruitment algorithm, and obtained the response in the form of a ranking score. Note that the adversary does not need to obtain these rankings for conducting the attack. Figure 6 shows that most of the resumes have a significant rank improvement. For example, more than 200 resumes out of 3000 resumes in the testing set had a rank improvement of more than 400, while more than 400 resumes had a rank improvement between 150 and 200 after adding the suggested keywords.

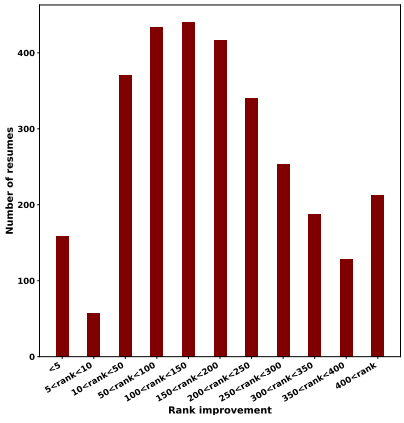


Figure 6: Rank improvement (complex setting)

8 Related Work

Attacks against text retrieval systems. Most of studies attack the deep neural networks for *text classification tasks*, such as sentiment analysis (Li et al., 2018), toxic content detection (Li et al., 2018), spam detection (Gao et al., 2018) and malware detection (Grosse et al., 2017). These works proposed methods for identifying text items that have a significant contribution in text classification task, then generate adversarial text by modifying those text items, e.g., modifying a word's characters (Liang et al., 2018; Ebrahimi et al., 2018b,a), adding or removing words (Liang et al., 2018), replacing words arbitrarily (Papernot et al., 2016; Sun et al., 2018), or substituting words with synonyms (Alzantot et al., 2018; Ren et al., 2019; Wang et al., 2019b; Blohm et al., 2018; Gong et al., 2018).

Adversarial examples for non-classification tasks. There are a couple of articles on crafting adversarial text for non-classification tasks. Jia and Liang (Jia and Liang, 2017) attacked machine comprehension and showed that the Stanford question answering dataset (Rajpurkar et al., 2016) is susceptible to black-box attacks, where adversarial questions are generated by appending distracting sentences at the end of the paragraph. A recent work (Cheng et al., 2020) attacked a text summarization, that is based on seq2seq models. In our paper, however, we propose crafting an adversarial text for ranking algorithms.

Attacks against resume search. A recent work (Schuster et al., 2020) showed that applications that rely on word embeddings are vulnerable to *poisoning attacks*, where an attacker can modify the corpus that the embedding is trained on and modify the meaning of new or existing words by changing their locations in the embedding space. However, our attack is not about poisoning the corpus, but instead it learns the words and phrases, based on some text embedding algorithm. In addition, we focus on the recruitment process that employs cosine similarity for ranking applicants' resumes compared to a job description.

9 Limitations and Future Work

This study examines the recruitment algorithms that use text embeddings and a similarity function to rank the resumes. However, examining more complex ranking algorithms is left for future work. In addition, the analysis can be expanded to other types of text embeddings. In the black-box setting, we did not investigate the trade-off between the number of queries that the adversary needs to send to the recruitment algorithm to create the training set, and the performance of the attack. It might be possible that the adversary can obtain good results using even a smaller training set. We will explore this in the future. We also have not studied the types of words that make it easy to game the system. In the future, we will examine the identified words in experiments in terms of their frequency, semantics, and other linguistic properties. Moreover, investigating the graybox attack remains for future work, when the attacker has this knowledge that the recruitment algorithm uses some kind of text embedding but does not know its type. This attack setting can help us to investigate the use of one text embedding approach for attacking a ranking algorithm that uses another text embedding approach. In other words, we can examine the transferability of one approach for another approach.

10 Conclusion

In this project we found that an automatic recruitment algorithm, as an example of ranking algorithms, is vulnerable against adversarial examples. We proposed a successful adversarial attack in two settings: white-box and black-box. We proposed a new approach for keyword extraction based on USE. We observed the majority of resumes have significant rank improvements by adding more influential keywords. Finally, in a black-box setting, we proposed multilabel neural network architecture to predict the proper keyword for each resume and job description.

References

Eneko Agirre. STSbenchmark. https://ixa2.si.ehu.eus/stswiki/index.php/STSbenchmark.

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium. Association for Computational Linguistics.

Melanie Tosik an. 2018. Debunking fake news one feature at a time. *ArXiv preprint*, abs/1808.02831.

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

Matthias Blohm, Glorianna Jagfeld, Ekta Sood, Xiang Yu, and Ngoc Thang Vu. 2018. Attention-based convolutional and recurrent neural networks: Success and limitations in machine reading comprehension. Association for Computational Linguistics.

Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014. Open question answering with weakly supervised embedding models. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 165–180. Springer.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174, Brussels, Belgium. Association for Computational Linguistics.

Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. 2020. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3601–3608. AAAI Press.

Javid Ebrahimi, Daniel Lowd, and Dejing Dou. 2018a. On adversarial examples for character-level neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 653–663, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018b. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia. Association for Computational Linguistics.

Massimo Esposito, Emanuele Damiano, Aniello Minutolo, Giuseppe De Pietro, and Hamido Fujita. 2020. Hybrid query expansion using lexical resources and word embeddings for sentence retrieval in question answering. *Information Sciences*, 514:88–105.

Excellarate. Resume Ranking using Machine Learning. https://medium.com/@Excellarate/resume-ranking-using-machine-learning-implementation-47959a4e5d8e.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56. IEEE.

Mohammadvaghef Ghazvinian, Yu Zhang, Dong-Jun Seo, Minxue He, and Nelun Fernando. 2021. A novel hybrid artificial neural network - parametric scheme for postprocessing medium-range precipitation forecasts. *Advances in Water Resources*, 151:103907.

Zhitao Gong, Wenlu Wang, Bo Li, Dawn Song, and Wei-Shinn Ku. 2018. Adversarial texts with gradient methods. *ArXiv preprint*, abs/1801.07175.

Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pages 62–79. Springer.

Jun Han and Claudio Moraga. 1995. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer.

David Harris and Sarah Harris. 2012. *Digital design and computer architecture (2nd ed.)*. Morgan Kaufmann.

Yousef Hassanzadeh, Mohammadvaghef Ghazvinian, Amin Abdi, Saman Baharvand, and Ali Jozaghi. 2020. Prediction of short and long-term droughts using artificial neural networks and hydrometeorological variables.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *ArXiv preprint*, abs/1812.05271.

Bin Liang, Hongcheng Li, Miaoqiang Su, Pan Bian, Xirong Li, and Wenchang Shi. 2018. Deep text classification can be fooled. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 4208–4215. ijcai.org.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814. Omnipress.

Jun-Ping Ng and Viktoria Abrecht. 2015. Better summarization evaluation with word embeddings for ROUGE. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1925–1930, Lisbon, Portugal. Association for Computational Linguistics.

Michael A Nielsen. 2015. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA.

Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. 2016. Crafting adversarial input sequences for recurrent neural networks. In *MILCOM 2016-2016 IEEE Military Communications Conference*, pages 49–54. IEEE.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy. Association for Computational Linguistics.

Gaetano Rossiello, Pierpaolo Basile, and Giovanni Semeraro. 2017. Centroid-based text summarization through compositionality of word embeddings. In *Proceedings of the MultiLing 2017 Workshop on Summarization and Summary Evaluation Across Source Types and Genres*, pages 12–21, Valencia, Spain. Association for Computational Linguistics.

Pradeep Kumar Roy, Sarabjeet Singh Chowdhary, and Rocky Bhatia. 2020. A machine learning approach for automation of resume recommendation system. *Procedia Computer Science*, 167:2318–2327.

Ruslan Salakhutdinov. 2014. Deep learning. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, page 1973. ACM.

Roei Schuster, Tal Schuster, Yoav Meri, and Vitaly Shmatikov. 2020. Humpty dumpty: Controlling word meanings via corpus poisoning. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1295–1313. IEEE.

D Sumathi and SS Manivannan. 2020. Machine learning-based algorithm for channel selection utilizing preemptive resume priority in cognitive radio networks validated by ns-2. *Circuits, Systems, and Signal Processing*, 39(2):1038–1058.

Mengying Sun, Fengyi Tang, Jinfeng Yi, Fei Wang, and Jiayu Zhou. 2018. Identify susceptible locations in medical records via adversarial attacks on deep predictive models. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 793–801. ACM.

Glassdoor Team. glassdoor. https://www.glassdoor.com/employers/blog/50-hr-recruiting-stats-make-think.

Indeed Editorial Team. 2021. How To Tailor Your Resume To a Job Description. https://www.indeed.com/career-advice/resumes-cover-letters/tailoring-resume.

Chenguang Wang, Mu Li, and Alexander J. Smola. 2019a. Language models with transformers.

Xiaosen Wang, Hao Jin, and Kun He. 2019b. Natural language adversarial attacks and defenses in word level. *ArXiv preprint*, abs/1909.06723.

Guangyou Zhou, Tingting He, Jun Zhao, and Po Hu. 2015. Learning continuous word embedding with metadata for question retrieval in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 250–259, Beijing, China. Association for Computational Linguistics.