Text Classification for Fake Job Posting

Jiaqi Jiang, Jingyu Zou, Yewon Kim

Abstract

Our project focuses on using classification models such as Logistic Regression, Random Forest, LSTM (Long short-term memory) and neural network in conjunction with various vectorization and embedding techniques to identify fake job posts based on textual information and meta-information of the job postings. We paid close attention to using an appropriate resampling technique for the imbalanced dataset during model implementation, and tested whether bringing in external reference data would improve the overall performance of our model. In addition, we implemented the semi-supervised learning technique to assess whether bringing in additional unlabeled dataset would improve the overall performance of our model. We came up with two neural network models using different vectorization techniques to solve the classification problem. Both models obtained f1-scores of around 0.79, which outperformed our baseline models.

The code is hosted in the following link: https://github.gatech.edu/jzou63/nlp_project

The recording of our presentation is in the following link: https://youtu.be/njlcYCikvLk

1 Introduction & Related Work

Nowadays, job seekers can easily find job postings online as many companies use online and automated systems in their hiring processes. It allows job seekers to find jobs more quickly and efficiently, but on the other hand, it also puts them at risk of online recruitment fraud. Such fraud not only damages job seekers, but also harms the reputation of organizations that have stolen their names. Our motivation is to build an effective classification model to identify fake job postings to prevent any potential risk of the frauds. The prior studies on fake job postings was mostly done with simple word embedding techniques like Bag-of-Words, and more basic classification models like Naive Bayes and Logistic Regression. [1]

For this project, we used more advanced word embedding techniques and neural network models and bring in additional predictors to identify fake job postings more effectively. Specifically, ELMo has shown great success in language modeling tasks and goes beyond the traditional word embedding approach by producing context-sensitive features in a bi-directional manner. [2] Our team decides to experiment with both word-based and context-based embedding in our models to compare the performance of each. We would also like to compare models' performance between methods that use neural networks versus non-neural network models. For the neural network models, we experimented with added layers, and fine-tuned the hyperparameters for optimal performance using cross validation. In addition to modeling techniques, we are also bringing external data to test realisticity of the postings by comparing the given salary with state averages. It allows us to evaluate a qualitative side of our numerical data, and improve the overall performance of classification.

Due to the imbalanced nature of our dataset, we want to test our re-sampling techniques, and specifically augment our data by generating additional data samples for minority classes. In addition, we are also going to test two different techniques to deal with imbalanced data: 1) Semi-supervised learning, and 2) Text generation. Referring to Keyvanpour who demonstrated his semi-supervised algorithm significantly improves the results of text classification in his paper [3], we are going to implement a semi-supervised model to test whether bringing in additional unlabeled data would boost the overall performance of the model. Text generation is another well-known method to improve text classification models with imbalanced labels. We

got an idea from Li, Guo, and Zhang's paper on lexicon generation for sentiment classification with imbalanced labels [4], and Sharifirad, Jafarpour, and Matwin's findings on boosting sexist tweet classification with text generation. [5] Based on these related studies, we are going to generate new text and test if it could improve the performance of our model.

2 Methods

2.1 Data

Our project uses the "Real or Fake: Fake Job Description Prediction" dataset from Kaggle, which contains 18 thousands job descriptions across 18 features with disk size of 50.1 MB. [6] In addition, we used the government census data, which contains the average employment and salary across different states, as the ground truth for location and salary range. [7] This file has 1380 rows with a disk size of 190KB for nationwide data and 36900 rows with a disk size of 5.73MB for statewide data. The data pre-processing process includes data cleaning, text formatting, stop words removal, missing value imputation, categorical variable encoding, and data aggregation, detailed descriptions are included in the midway report.

Since the midway report, we decided to bring in two additional sets of data for further analysis. For semi-supervised learning, we brought in an additional unlabeled dataset "US jobs on Monster.com" dataset from kaggle, which includes 22,000 US-based job listings with disk size of 65 MB. [8] We also bring in synthetically generated data to be labelled as fake in order to deal with imbalanced labels of our dataset. We generated 5000 new texts for description and company profile to get more balance between the labels.

Our data consists of many categorical variables with information on job location, employment type, and required experience, etc. If we simply convert those categorical variables to binary variables, we would get an extremely large matrix and the models are likely to overfit. Therefore, we performed variable selection on the numerical dataset by fitting a random forest classifier against the labels, and only kept features that have an importance of more than 0.005. As a result of feature selection, we reduced the dimension of numerical variables from over 600 to 55.

2.2 Baseline Models- With and Without External Data

Prior research done on recruitment fraud detection utilizes the same dataset as we used. [1] Researchers decided to create a balanced dataset by randomly selecting 450 real postings and 450 fake postings that contains the most information (with little to no missing values). After careful consideration, our team believes that 900 data points don't necessarily represent the extent of information contained in the large dataset. Hence, we didn't find the undersampling method appropriate. The models that performed well set forth in the paper are Logistic Regression and Random forest. Therefore, we have decided to implement both as baseline models for performance comparison. In order to balance the training and testing set so that both splits contain the same percentage of majority versus minority class. We tokenized the texts using the bag-of-word method and concatenated the text and numerical columns to feed into the model. The performance is shown below.

Model	F1-Score	Precision	Recall	Accuracy
Logistic Regression	0.24	0.15	0.68	0.79
Random Forest	0.63	1	0.47	0.97

Table 1: Baseline Model- Data without Salary

Model	F1-Score	Precision	Recall	Accuracy
Logistic Regression	0.25	0.15	0.68	0.80
Random Forest	0.65	1	0.48	0.98

Table 2: Baseline Model- Data with Salary

For the salary column, prior work has mostly analyzed the salary within the given dataset as numerical values. Upon that, we decided to analyze the salary further by comparing with the external data and checking if the given salary was in a reasonable range. We used the government statistics of national/state occupation and salary published in 2019 to validate the salary for US job postings based on the national/state average. We matched two datasets by state and department, and then indicated if the difference is more than twice using a binary column. If state location is not provided, we used the national average to compare instead.

Another binary column was used to indicate if the salary is missing in the government statistics. After adding salary related data, we ran our baseline models again with the same hyperparameters. Results are shown above in **Table 2**. Compared to **Table 1**, which doesn't contain any salary information, the f1-score for the models increased, indicating that bringing in external data for reference increased the overall performance of our model.

2.3 Vectorization Result

In this project, we experimented with three different vectorization methods. First, we utilized the Tokenizer method in the Keras package to tokenize the corpus by turning each text into a sequence of integethers with each integer being the index of that token in the dictionary. We also used the CountVectorizer function in the sklearn package to convert text corpus to matrices of token counts. Furthermore, we implemented TF-IDF to transform word count to reflect the importance of words to a document in a corpus.

For each method, we outputed the f1-scores using both logistic regression and random forest model to compare whether one outperforms the other. Surprisingly, the regular bag-of-words tokenization performed the worst amongst three while Countvectorizer performed the best. We suspect that specific word counts present much more explanatory power than how words are ordered in job postings. We decided to move forward with building our neural net models using both bag-of-words tokenization and countvectorizer.

Model/Vectorization		**Tokenizer**	**Counter Vectorizer**	**TF-IDF**
Company Profile	Logistic Regression	0.23	0.38	0.38
	Random Forest	0.38	0.38	0.38
Description	Logistic Regression	0.13	0.61	0.55
	Random Forest	0.60	0.71	0.70

Table 3: Vectorization Result

2.4 Result Comparison- Hypothesis

2.4.1 Generated Fake Postings

Text generation is done with a python module called *textgenrnn*, which was built upon Keras/TensorFlow using RNN models to generate text. We took 606 texts for description and company profile that are labeled as fake from our training data to train, and generated 5000 texts based on these training sets.

For numerical data, we used SMOTE to resample our dataset to balance the labels. Resampling is done on our training data with 6:4 ratio between real and fake postings. Then we took 5000 data points from the generated data, and randomly combined with the generated texts. We tested our new training data with these generated fake postings using our Logistic Regression baseline model.

We noticed that the performance of our baseline model has decreased to the f1-score of 0.44. There could be two possible reasons: 1) Sample size was too small, and 2) Significant linguistic context and semantics might be ignored. Considering complex and diverse nature of variables in each job posting, a sample size of 606 could be too small to perform synthetic generation well. Also, *textgenrnn* generates text only in char-level or word-level, without taking account of any semantics or context of entire sentences. This could lead to loss of significance of the texts.

2.4.2 Semi-Supervised Learning

Aiming to enrich our database with more fake job postings, our group brings in the additional dataset, the Monster.com job posting samples dataset. Since the new dataset and our original dataset both have job descriptions, we decide to use this column to test whether semi-supervised learning will improve the baseline model's F1 score.

Our initial method is to use logistic regression to pseudo label the unlabeled data and train the model again with the unlabeled data which has a high probability (0.6) of being fake. The result shows a huge decrease in F1 score. We went back to the baseline model and found out the logistic regression which used to pseudo label the data performs poorly, the model might poorly pseudo-labeled the unlabeled postings.

We solved this problem by using the better performed baseline model, the Random Forest Model. However, using the Random Forest Model with threshold 0.5, we were only able to find 2 fake postings. Therefore we did not incorporate semi-supervised learning in subsequent models. It is very hard to find fake postings, especially from the job postings from career boards such as Monster.com as they might already screen out most of the fake

postings. A much larger numbers of job postings is required to find more potential fake postings for semi-supervised learning purpose.

3 Model/Analysis

While prior research used traditional machine learning models such as random forest and logistic regression, our team proposes to use neural networks to address this classification problem. We believe that neural network models have the ability to extract meaningful variables with significant explanatory power that could further improve the performance of our model. We proposed to experiment with different embedding techniques, activation functions, and hyper-parameters to construct a more accurate model compared to baseline established.

3.1 Experiment Setup

We first split our data into training and testing sets using the 70%:30% ratio in a stratified fashion. Due to the imbalanced nature of our data, we further split the training set into 4 folds in a stratified fashion for cross validation purposes of our data. For more complicated neural net models with higher runtime, we allocated 20% of the training set for validation purposes. The validation split is also stratified to ensure the same percentage of minority class in each set.

Within our data, only 866 out of 17880 job postings are fake, making our dataset extremely imbalanced. Without using any resampling techniques, we've been getting extremely high accuracy scores, but low f1-scores, as the models are more likely to predict job postings to be real rather than fake.While the undersampling method yielded relatively high f1-scores, we believe that only training on 1600 data points could potentially lead to information loss and poor performance when tested with new data. Therefore, we decided to adopt the oversampling technique. Throughout model implementation, we decided to try adjusting class weights so as to balance the data. We also experimented with using the RandomOverSampler function in imblearn package to oversampling minority classes during training. When building the neural net models, we set the optimizer to "adam", gradient-based optimization of stochastic objective functions. This optimizer has a default learning rate of 0.001. We tried setting the loss function of our neural network models to crossentropy-loss as well as f1-loss during tuning.

3.2 Result Comparison- Neural Network

Generic word embeddings capture both semantic and syntactic information about words in a corpus, where words are represented by dense vectors projected into a vector space [9]. Embeddings can be trained from text data and even reused in other projects. Recent advancement has shed light on context-based embeddings, including ELMo and BERT, aiming at capturing word semantics in different contexts. Researchers have found context-based embeddings outperforming traditional word embeddings in NLP tasks [9].

In this project, we implemented neural network models using both traditional word embedding and context based embedding to compare the performance of each for fake job classification class. For traditional word embedding, we decided to use embedding layers in keras to train the embeddings for words in our corpus. For context based embedding, we chose to implement ELMo embedding, developed at ALLEN NLP and learned from the internal state of a bidirectional LSTM to represent contextual information of the text [10]. For each approach, we implemented a neural network model using both text and numerical data, experimented with different layer structure and hyperparameters using the held-put test set split on stratified data.

4 Results

4.1 Neural Network Model

As we found success in using Countvectorizer to transform the text corpus, we decided to utilize a neural network model, incorporating counts for both company profile and description columns, in order to perform classification. The concatenate function within Keras allows us to train separate neural network models on company profile, description, and numerical data against the label and combine the outputs in a concatenated model for final prediction. We implemented cross validation and tested the following sets of hyperparameters, shown in **Table 4**. Overall we ran the model on 18 different hyperparameter combinations using cross validation and recorded the combinations with the highest average f1-score on the validation set.

We picked the hyperparameter combination with the highest averaged f1-score from cross validation (Batch size=30, Epoch=10, Loss function set to f1-loss) and reran the model on the combined train

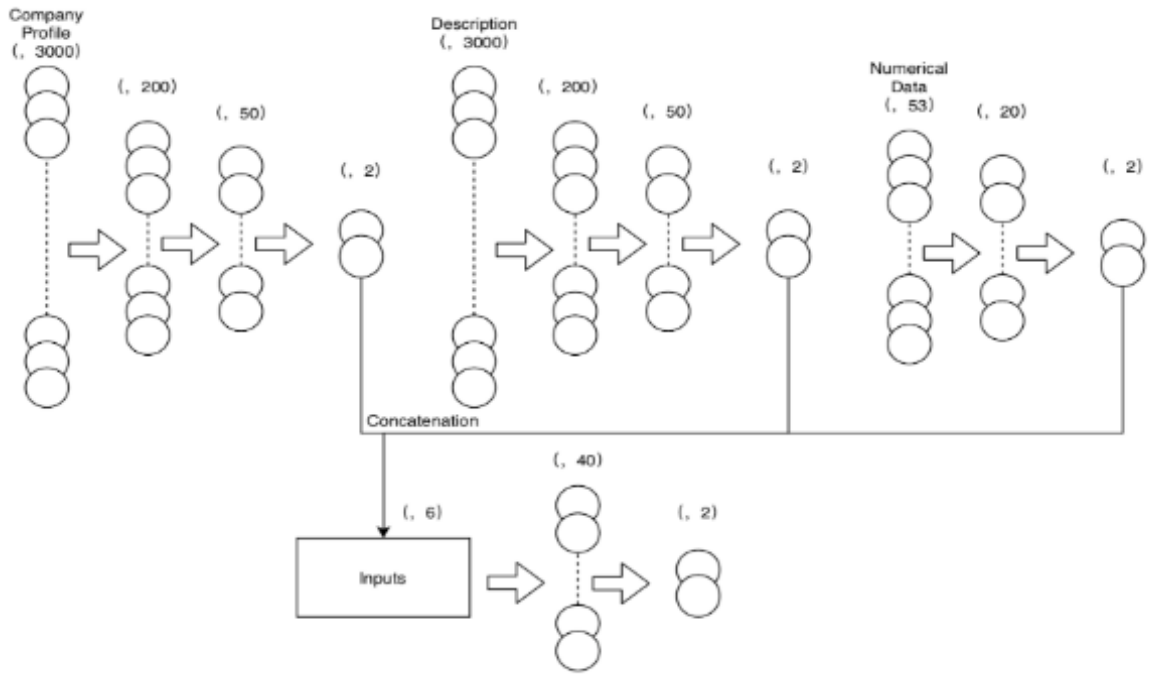

Figure 1: Structure of Neural Net

and validation set. The final model has a f1-score of 0.79. The structure of the neural net is shown in **Figure 1**.

4.2 Neural Network Model with LSTM

While the neural network model with Countvectorizer yielded good performance, we also implemented a LSTM (long short term memory network) using Keras embedding for the classification task. We tuned our model with a stratified validation set and tested 12 hyper-parameter sets, shown in **Table 5**. During tuning, we realized that increasing the number of embedding dimension allowed the model to learn additional feature from the text

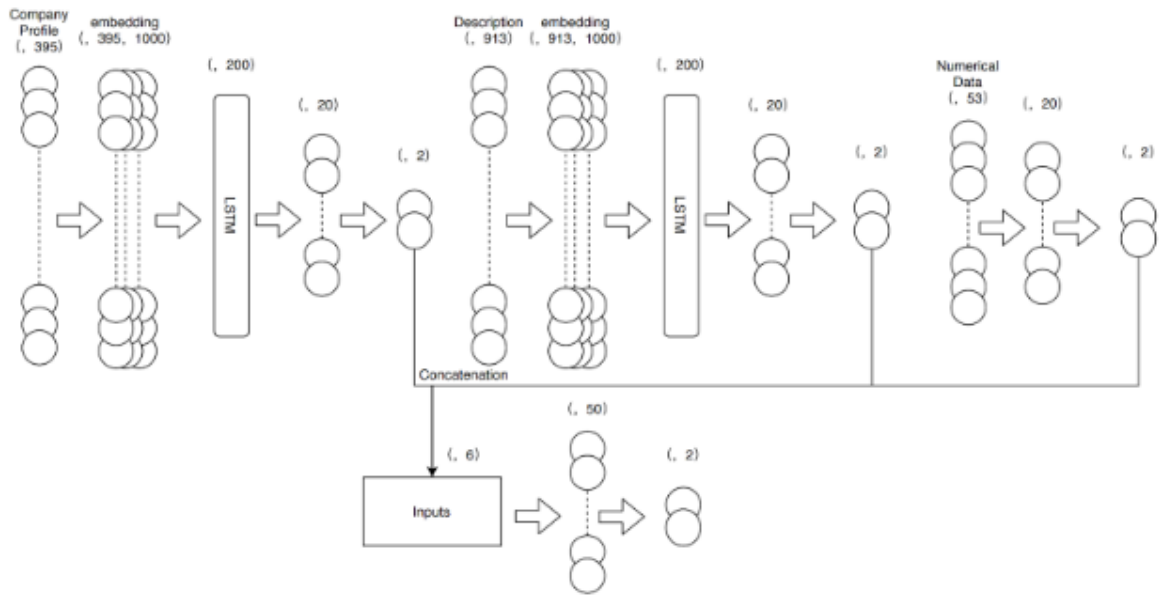

Figure 2: Structure of Neural Net with LSTM

Batch Size	Epoch	Loss Function	Averaged F1 -Valid. set
30	10	categorical crossentrophy	0.6516
20	4	categorical crossentrophy	0.6332
30	7	categorical crossentrophy	0.6261
30	4	categorical crossentrophy	0.6260
20	7	f1-loss	0.5912
20	10	f1-loss	0.5995

Table 4: Hyperparameter Tuning - NN

data, and including a dropout rate enabled model performance to be more stable and helped avoid overfitting. LSTM has been proven to be a well-suited model for classification tasks as it is insensitive to gap length and can capture a wider range of information in a sequence.

Similarly, we picked the hyper-parameter combination with the highest averaged f1-score from cross validation (Batch size=40, Epoch=10, Loss function set to categorical cross entropy loss and no dropouts) and reran the model on the combined train and validation set. The final model has a f1-score of 0.7842. The structure of the neural net is shown in **Figure 2**.

Batch Size	Epoch	Embed Dim.	Dropout	Loss Function	F1-Valid
20	8	500	No	categorical crossentropy	0.7570
20	8	500	0.2	categorical crossentropy	0.7570
20	8	500	No	f1	0.7417
20	10	500	No	categorical crossentropy	0.7586
20	10	1000	No	categorical crossentropy	0.7444
40	10	1000	No	categorical crossentropy	**0.7644**
40	10	1000	0.2	categorical crossentropy	0.7444

Table 5: Hyperparameter Tuning - NN with LSTM

4.3 Neural Network Model with ELMo

Lastly, we found the best hyper parameters for Elmo is (Batch size=20, Epoch=10, Loss function set to categorical cross entropy loss and no dropouts). We reran the model on the combined train and validation set. The final model has a f1-score of 0.7364. ELMO does not perform as good as the NN model and the LSTM model.

Batch Size	Epoch	Loss Function	F1-Valid. set
20	5	categorical crossentrophy	0.5573
20	10	categorical crossentrophy	0.6516
50	10	categorical crossentrophy	0.6169
20	5	f1-loss	0.5912
20	10	f1-loss	0.5995

Table 6: Hyperparameter Tuning - NN with ELMo

5 Error Analysis Model Comparison

NN			LSTM		
	Predicted 0	Predicted 1		Predicted 0	Predicted 1
True 0	5082	22	True 0	5082	22
True 1	77	183	True 1	71	189
F1-score: 0.79			F1-score: 0.78		
Precision: 0.89			Precision: 0.85		
Recall: 0.70			Recall: 0.73		
Accuracy: 0.98			Accuracy: 0.98		

Table 7: Error Analysis

NN and LSTM have similar F1 score and accuracy, the LSTM model has a slightly higher Precision and lower recall compared to the NN model. Looking at the confusion matrix shown in **Table 7**, the Neural Network model with LSTM seems to do a better job identifying fake job posts, with less false negatives. Training loss plots for both models (**Figure 3**, **Figure 4**) indicate that as the number of epoch increases, training loss approximates to zero. Given that the highest f1-score for our baseline models is 0.63, we conclude that both models outperform the baseline models.

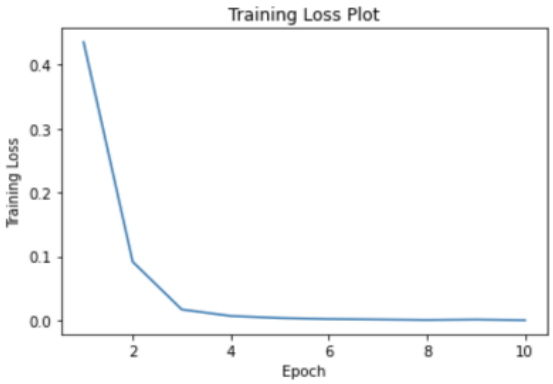

Figure 3: Training Loss for NN

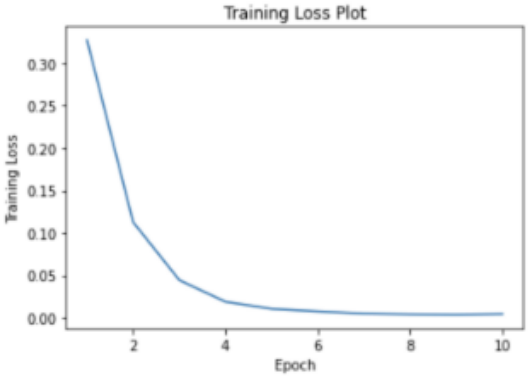

Figure 4: Training Loss for NN with LSTM

6 Work Division

All team members have contributed a similar amount of effort. Our tasks were divided as shown in **Table 8**.

Task	Member Responsible
Data cleaning	Yewon Kim, Jiaqi Jiang
Exploratory data analysis, Implement baseline model	Jingyu Zou
Feature engineering, Implement Semi-supervised learning	Jiaqi Jiang, Jingyu Zou
Data augmentation, Implement BERT/ELMo for classification	Yewon Kim
Finetune hyperparameters, Set up code repository	All
Finish final report	All

Table 8: Work Division

7 Conclusion

Our project focused on a real world problem faced by job seekers: fake job posts identification. We expanded on prior research and applied supervised and semi-supervised learning methods and explored different classification techniques using Logistic Regression, Random Forest, Neural Network, LSTM in conjunction with different vectorization and embedding methods [1]. As expected, our proposed neural network models outperformed the baseline models.

To our surprise, word-based embedding outperformed context-based embedding (derived using ELMo framework) for this particular problem. As a sampling approach, we generated additional fake job posts by building a language model on our minority datasets in order to enrich our imbalanced dataset. However, the performance of our baseline model slightly decreased after additional data being brought in. We believe this is due to the fact that our data is extremely imbalanced, and the number of minority classes present didn't provide us with enough information to generate additional data.

We also experimented with semi-supervised learning approach. We hoped that by bringing in additional unlabeled data, our model can learn a better prediction rule than simply training on labeled data alone. However, using the labeled data we have, we were only able to generate a very small number of postings classified as fake with their pseudo labels. Therefore, we concluded that bringing in this particular unlabeled dataset [8] did not aid in our analysis.

For potential future study, we would suggest the researcher look for another set of unlabeled dataset, ideally in a similar format to our job posting dataset so as to reach better results. Due to limited time, we didn't get to experiment with a lot of feature engineering for the scope of this project. Researchers might find success by extracting additional features from data to better explain the response labels.

References

[1] Kolias C. Kambourakis G. Vidros, S. and L. Akoglu. *Automatic Detection of Online Recruitment Frauds: Characteristics, Methods, and a Public Dataset*. Number 9(1). Future Internet, 2017.

[2] K. Aggarwal and A. Sadana. *Propaganda Detection from News Articles using Transfer Learning*. SAP Labs, 2019.

[3] M. Keyvanpour and M Bahojb Imani. *Semi-supervised text categorization: Exploiting unlabeled data using ensemble learning algorithms*. Intelligent Data Analysis, 2013.

[4] Guo H. Zhang Q. Gu M. Li, Y. and J. Yang. *Imbalanced text sentiment classification using universal and domain-specific knowledge*. Knowledge-Based Systems, 2018.

[5] Jafarpour B. Sharifirad, S. and S. Matwin. *Boosting Text Classification Performance on Sexist Tweets by Text Augmentation and Text Generation Using a Combination of Knowledge Graphs*. Association for Computational Linguistics, 2018.

[6] S Bansal. *[Real or Fake] Fake Job Posting Prediction*. 2020.

[7] U.S. Bureau of Labor Statistics. *May 2018 State Occupational Employment and Wage Estimates*. 2019.

[8] PromptCloud. *US jobs on Monster.com*. Kaggle, 2017.

[9] Goldberger J. Melamud, O. and I. Dagan. *context2vec: Learning Generic Context Embedding with Bidirectional LSTM*. Association for Computational Linguistics, 2016.

[10] Neumann M. Iyyer M. Gardner M. Clark C. Lee K. Peters, M. and L. Zettlemoyer. *ELMo: Deep contextualized word representations*. 2018.