Predicting Job Postings as Real or Fake

Jayme Kirchner – CSC 535 – Deep Learning

TABLE OF CONTENTS

1 ABSTRACT

This project looks to address the problem of fake job postings and tries to predict whether a given job posting fraudulent based on its text. The data was highly imbalanced with only 5% fraudulent postings and required extensive preprocessing to replace missing values. The baseline model was a RNN (Recurrent Neural Network) with an Embedding layer and a Bidirectional LSTM layer, and its results were compared with a BERT (Bidirectional Encoder Representations from Transformers) model. Due to resource constraints, the Small-BERT architecture was applied. Initially, the baseline model was trained on all data (numeric and textual) but the AUC of 50% was no better than a random classifier so the project instead focused solely on the textual data. The twelve text columns in the dataset were split into five groups to be passed as inputs to each model. After five epochs, the baseline RNN resulted in a validation AUC of 0.7974 with 139 False Negatives; and the test data had a final AUC of 0.7864 with 173 False Negatives. The Small-BERT model ran for a single epoch on the same five text groupings and yielded significantly higher results. The validation AUC was 0.9625 with 109 False Negatives and 1 False Positive; and the testing AUC was 0.9637 with 119 False Negatives and 4 False Positives. The ability of both models to distinguish between fraudulent and real postings on heavily imbalanced data is promising, but it highlights the need for job posters to be diligent in creating complete job postings, and for job seekers to read all postings with a critical eye.

2 PROBLEM DEFINITION AND PROJECT GOALS

The focus of my project is to predict whether a job posting is real or fake, based on the words contained in the job posting's text. Typical "red flags" when looking at a job posting include: (i) the company has no online presence [1][2]; (ii) the post contains many grammar errors or lots of capitalization [1][2]; (iii) the posting asks for money or personal information [1][2][3]; (iv) it sounds too good to be true [1][2]; (v) no experience is needed [2]; and (vi) there are little-to-no details provided about the job or company [2]. Common job titles that are prone to scams are Assistants, Receptionists, Delivery Drivers or Chauffeurs, and Warehouse Workers [3]. As a jobseeker looking to enter the field of Computer Science, this topic stood out to me because I have received multiple emails from "recruiters" with what appear to be fake job posts that request personal information such as social security numbers or a photo of government identification documents. Many of the job posts in this dataset do not always have these obvious "red flags" and so I wanted to create a model to try and predict which job postings are authentic so I could apply it to postings that I find online during my future job searches.

The public dataset was downloaded from the Kaggle website [4], and it contains almost 18,000 job postings, of which roughly five percent are fake. There are 18 variables in total, and they can be split into four categories: Text, Categorical, Numeric, and Other; and the table below shows the features that correspond to each category. The 'Salary Range' column is listed in the table as 'Other' because it is a string in the original dataset, but it will be converted into numeric values and split into two separate columns.

Category	Relevant Columns from Dataset
Text	Title, Location, Department, Company Profile, Description, Requirements, Benefits
Categorical	Employment Type, Required Experience, Required Education, Industry, Function
Numeric	Job Id, Has Company Logo, Telecommuting, Has Questions, Fraudulent
Other	Salary Range

Table 1: Data Categories and Relevant Columns

After the data is cleaned and missing values have been removed, I will apply the data on a RNN model with embedding layer, and a Small-BERT to compare the accuracy of each model in predicting the fake job postings. Due to the dataset's imbalance of approximately 5% fraudulent posts to 95% real posts, the primary metrics used to determine the model's success will be the Area Under the Curve (AUC), False Positives (FP) and False Negatives (FN). My aim is for the AUC to be in a range between 80% and 90%, with the lowest possible values for FP and FN.

3 Related Work

As a public dataset on Kaggle [4], there have been sixty-five people who have submitted their solutions to this problem using a variety of models. Many are machine learning models such as Naïve Bayes, Random Forest, and Logistic Regression; but there are four notebooks that utilize at least one of the Deep Learning models that I chose for this project. However, it is important to note that the author of each notebook has manually cleaned the text data to remove stop words and unwanted characters [5][6][7][8]; and two of the notebooks had applied lemmatization [6] and stemming [7]. While these steps may have increased the speed of the model's training, they also could have the effect of removing (potentially important) context from the job posting.

The best result [5] produced an F-1 score of 0.84 for the fraudulent postings and 0.99 for real postings. The author focused on the text from seven columns (Title, Department, Company Profile, Description, Requirements, Benefits, and Function) and applied the Classification Model class from Simple Transformers [9] with BERT on the text.

[6] used all text columns on both a Bidirectional LSTM model and a Large-BERT model. The baseline model was a Sequential Keras model with an Embedding layer, a Dropout layer, and a Bidirectional LSTM layer. There was drastic overfitting during training, but no adjustments were made to reduce it; and the final F-1 score on fraudulent posts was 0.71 (0.99 for real posts). Their second model was Large-BERT (L-24_H-1024_A-16/1) with the max length of tokens set to 160. Since all text is combined into one column, this means that a large amount of data could be lost due to this low number of tokens. Again, the model is overfitting during training; and the F-1 score for fraudulent posts was 0.79 (F-1 score for real posts was unchanged).

[8] applied three models to this dataset: Logistic Regression, Glove, and DistilBERT. While the results are good with AUC scores of 85% for Logistic Regression, 81% for Glove, and 77% with DistilBERT, the fact that the author used a different method in each model to split the data makes it difficult to say whether these results are truly accurate because the models did not necessarily train and test on the exact same rows. The baseline model was Logistic Regression, and the author applied a Count Vectorizer before splitting the data with a stratified k-fold. For Glove, they applied sklearn's train_test_split method (different random state than the baseline model) and then passed the training set into a two-layer Sequential model with Batch Normalization. The final model used PyTorch and DistilBERT before passing the results to a Logistic Regression model; but it is important to note that the data was not stratified to keep the same 95/5 ratio of real postings to fraudulent postings. The author only selected the first 2000 rows, which contained 1948 real postings and only 52 fake postings (approximately 3%). This ratio is even less than the original data imbalance of 5% so it may explain why the AUC dropped for this model compared to the previous two.

The final notebook [7] focused solely on the description column of the data and applied a Sequential model with an Embedding layer and a Bidirectional LSTM layer. Each sentence was reduced to 40 stemmed tokens and the final F-1 score was the lowest at 0.63, which was not surprising due to the lack of text data and size of the embedding vectors passed to the model.

Unlike the above examples, I used all text columns. Initially, the text was split into 512 tokens, but resources were inadequate to run the BERT model on this many tokens so both models instead received inputs of 128 tokens. I also wanted to minimize manual modifications of the text because neither model requires extensive manual preprocessing; thus, the only modifications made were to separate combined words (i.e., PinterestLoves changed to Pinterest Loves), and to change the country "US" into "USA". The AUC for the text-only test data was 0.7864 for the baseline RNN model (FN: 173; FP: 0), and .9637 for the Small-BERT model (FN: 119; FP: 4).

4 Data Exploration and Preprocessing

After first downloading the data, I removed the Job Id column and renamed the Telecommuting column to Work Remote. Figure 1 lists with the name, number of non-null values, and data type of each column in the dataset; and Figure 2 shows the distribution of fraudulent and real job postings. The dataset has many missing values, and the data is very imbalanced with only 866 fraudulent postings (approximately 5%).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17880 entries, 0 to 17879
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   title                17880 non-null  object
 1   location             17534 non-null  object
 2   department           6333 non-null   object
 3   salary_range         2868 non-null   object
 4   company_profile      14572 non-null  object
 5   description          17879 non-null  object
 6   requirements         15185 non-null  object
 7   benefits             10670 non-null  object
 8   work_remote          17880 non-null  int64
 9   has_company_logo     17880 non-null  int64
 10  has_questions        17880 non-null  int64
 11  employment_type      14409 non-null  object
 12  required_experience  10830 non-null  object
 13  required_education   9775 non-null   object
 14  industry             12977 non-null  object
 15  function             11425 non-null  object
 16  fraudulent           17880 non-null  int64
```

Figure 1: Dataframe at a Glance

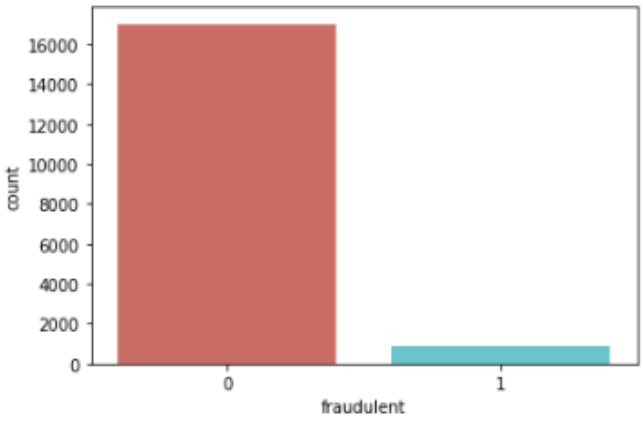

Figure 2: Distribution of Job Postings

The following steps were taken to preprocess the data prior to running the models:

1) Handling missing values.
2) Split salary range and replace missing values.
3) Final text cleaning.
4) Update column order.
5) One-hot-encoding categorical variables.
6) Splitting the data into train, validation, and test sets

Further details about each step are provided in the sections below.

4.1 HANDLING MISSING VALUES

When the data is initially downloaded, there are only five columns that have no missing values: Title, Work Remote, Has Company Logo, Has Questions, and Fraudulent (the target variable). The Description column only has one missing value, and the Location column is missing 346. As the table shows below, the remaining columns have thousands of missing values that must be replaced, with Salary Range having the most at over 15,000.

```
title                    0
location               346
department           11547
salary_range         15012
company_profile       3308
description              1
requirements          2695
benefits              7210
work_remote              0
has_company_logo         0
has_questions            0
employment_type       3471
required_experience   7050
required_education    8105
industry              4903
function              6455
fraudulent               0
```

Figure 3: Missing Data (per column)

Due to the high number of missing values, it would not be prudent to simply remove these rows. For the five categorical columns (Employment Type, Required Experience, Required Education, Industry, Function), missing values can be replaced by using the mode of the column or by creating a new label. I chose to replace missing values with a new label of "No Data" to distinguish between the complete absence of data and the already-present labels of "Not Applicable" or "Unknown" because the absence of data can be a telling factor in determining that a job posting is fraudulent [3].

Now it was time to look at the seven remaining text columns with missing values (Location, Department, Salary Range, Description, Company Profile, Requirements, Benefits); and I wanted to see how many rows had all missing values for these columns because they could potentially be removed from the dataset. Since Description only had a single missing value, there is only one possible row where all text data aside from the title was missing and it could be dropped from the dataset. To my surprise, there were no rows that had all text data missing except for the title; this row was missing a description but had a location and so it remained in the dataset because it is one of the 866 fraudulent postings. Looking at the six remaining text columns, only 19 rows were missing values in all text columns aside from Title and Description. Although this is a small number relative to the total amout of rows, the fact that 12 of these 19 rows were fraudulent postings supports [3] that the absence of data is a strong indicator of a fake job posting; thus, I chose to keep them in the data frame and instead replaced all missing text values with an empty space and created a new column, Full Text, which combined these preprocessed text columns into a single paragraph.

There were no missing values for the binary variables (Work Remote, Has Questions, Has Company Logo, Fraudulent), but the Salary Range column only had 2868 cells that were not missing values. The next section (4.2) details the steps taken to clean up the Salary Range column.

4.2 THE SALARY RANGE COLUMN

4.2.1 Splitting the Salary Range

In the original dataset, the Salary Range values are strings instead of numeric integers and so the first step was to split the data into two columns, indicating the distinct values for Low Salary and High Salary. However, because they were strings, some values that were less than 12 were interpreted as months of the year. For example, a salary range of "12-15" (indicating "12,000 to 15,000") were interpreted as "Dec-15" in the dataset. To correct this, I created a dictionary with the months of the year as keys and the corresponding number as the value (i.e., {"December": 12}); and then looped through the data to update the values. Once completed, the values of each column were set to numeric integers and the columns were added back to the original dataset in preparation for the next step of replacing its missing values.

4.2.2 Replace Missing Values

Now that the salary values are numeric, the mean could be calculated based on each employment type (Contract, Full-Time, No Data, Other, Part-Time, and Temporary). It is important to note that the mean was calculated based on the training data only and the mean() method used only non-null values. First, the two salary columns were split into train, validation, and test sets (80-20 split, random seed used for reproducibility); and the table below shows the resulting means grouped by employment type. The values for the 'Full-Time' label are unrealistic but it could indicate that the values present may also be unrealistic and possibly have come from fraudulent job postings.

Employment Type	Salary Low	Salary High
Contract	35562	55320
Full-time	886959	1397585
No Data	51559	81863
Other	18826	335284
Part-time	15085	27513
Temporary	14091	17356

Table 2: Mean Salary Values Grouped by Employment Type

The means were saved as a dictionary with the key being the employment type, and the salary columns were added to the original data frame. To fill in the missing salary values in the original data frame based on employment type, I created a loop that would find all rows with that employment type, fill in the missing high and low salary values using the relevant saved means, and then update the original data frame with these new values. Since NaN values are floating point values, the final step was to cast all salary values as numeric integers to remove all decimals.

4.3 Final Pre-Processing Steps

4.3.1 Minor Text Cleaning

Now that all missing values have been replaced, the dataset is almost ready for the models. The built-in preprocessing steps for the two chosen models, BERT and RNN, will make all words lower case and remove unwanted characters. To distinguish the country "US" from the word "us" when put into lower case, I changed all instances of "US" (where both letters were capitalized) into "USA" so that I could distinguish it when viewing the list of lowercase tokens. I had also noticed some instances with two separate words had been combined (i.e., "PinterestLoves") and so I created a function to separate them if a lowercase letter was next to an uppercase letter. In this way, the words are as distinct as possible, which may help to improve the model's predictions.

4.3.2 Update Column Order

After all text is updated, the Full Text column is brought to the front of the dataset and the Fraudulent column is moved to the end. This dataset was copied as a new dataset to be applied later to the text-only models.

4.3.3 One-Hot-Encoding Categorical Variables

In preparation for the variables to be passed to the baseline RNN model, the five categorical text variables (Required Education, Required Experience, Function, Industry, Employment Type) were one-hot-encoded to be passed with the other numeric variables of Work Remote, Has Questions, Has Company Logo, Salary Low, and Salary High.

4.4 Splitting the Data into Train, Validation, and Test Sets

The data had previously been split to calculate the mean for each employment type in the training data; but after pre-processing all data, the dataset is re-split into train, validation, and test sets. The same 80/20 ratio and random seed that had been used previously to calculate the salary means of the training set were again used for these splits so that no changes were made to the rows assigned to each dataset. The final training set, train_x, had 10,889 real posts and 554 fraudulent posts. While further exploring the train_x dataset on which the models will be trained, I found that the training set only has real postings for five levels in the Industry columns: Libraries, Military, Package/Freight Delivery, Shipbuilding, and Wine and Spirits. This is important because the models will only be trained on real job postings and so any job posting for these industry levels will be marked as a real posting and increase the resulting False Positive value.

5 Data Analysis

The baseline model for this project was a Recurrent Neural Network (RNN) with a Text Vectorization layer, an Embedding Layer, and a Bidirectional LSTM Layer; and the results were compared with those from Small-BERT. Due to the data imbalance, predicting "accuracy" is misleading because it is guaranteed to be 95% just by always predicting that a job post is authentic. Thus, the metrics used were False Positives (FP), False Negatives (FN), and Area Under the Curve (AUC). Binary Accuracy (accuracy) was also displayed but was not used as an indicator of good model performance. I used Binary Cross-Entropy for the loss function since the posting is either fraudulent or not; and the optimizer was Adam for RNN, and AdamW for Small-BERT. The RNN models ran for five epochs in training and testing, while the Small-BERT models ran for one epoch.

5.1 BASELINE RNN MODEL

5.1.1 Text and Numeric Inputs

5.1.1.1 Building the Model

The baseline model would take as input all text from the Full Text column and all numeric inputs. The text inputs would pass through a TextVectorization layer to be broken into 128 tokens; then passed to an Embedding layer with dimensions set to 128; and finally, to a Bidirectional LSTM layer with 128 units to learn the context of each token. The numeric inputs were passed through a Dense layer of 128 units and Relu activation; a Dropout layer set to 0.1; and finally, to a second Dense layer with 128 units and Relu activation. Both inputs were then combined and passed together through a final Dense layer with 32 units and Relu activation. The output layer consists of 1 unit with Sigmoid activation. In total, this model has 334,529 total parameters and all are trainable. Figure 4 shows the graph of its architecture.

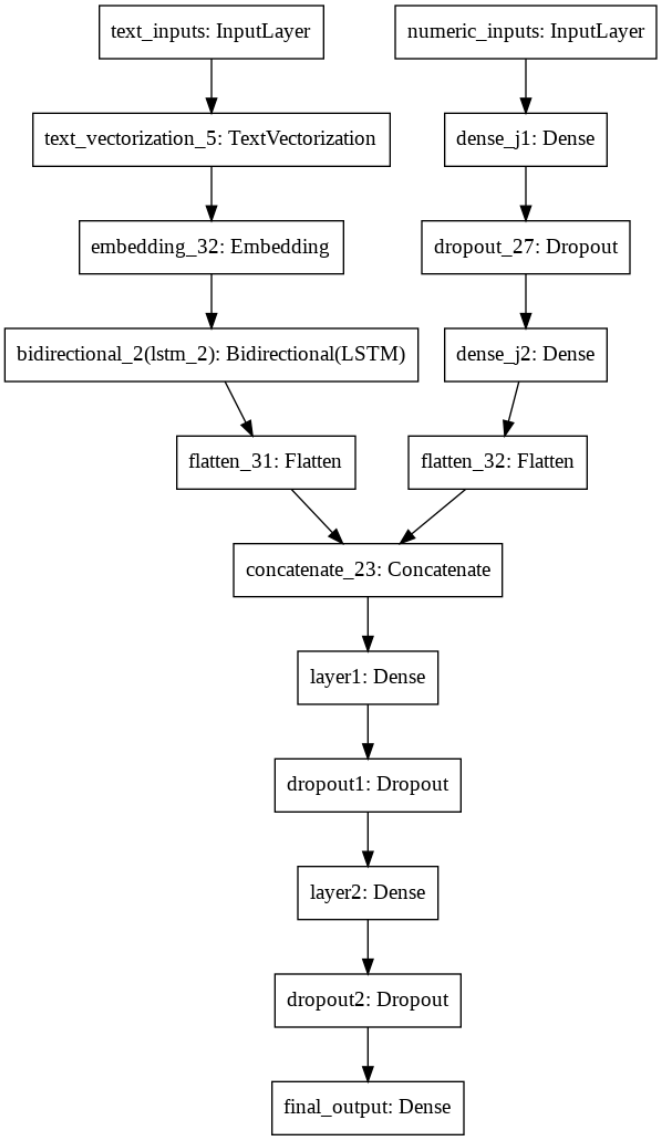

Figure 4: Baseline RNN Model Architecture (Text & Numeric Inputs)

5.1.1.2 Training the Model with Text and Numeric Data

The final model was compiled using Adam optimizer with a learning rate of 1e-5, and Binary Cross-Entropy loss; and it measured the metrics of Loss, Accuracy, AUC, False Positives, and False Negatives. After running for five epochs, the model produced a training AUC of 0.5124 (FN: 515; FP: 492) and a validation AUC of 0.5054 (FN: 137;

FP: 10). The model did not overfit, but its performance was no better than a random classifier. The graphs in Figure 5 display the results of each metric for the training and validation data throughout the five epochs.

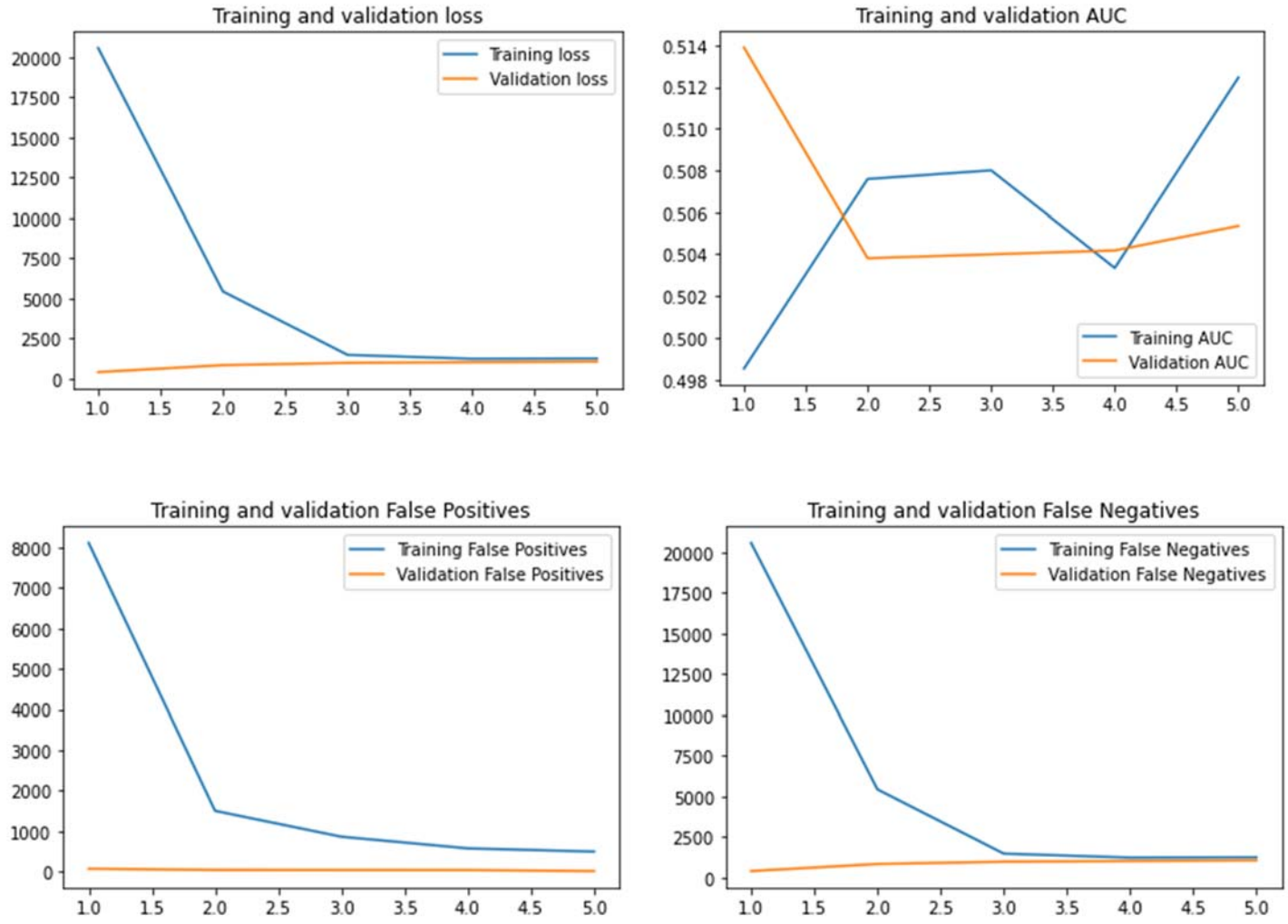

Figure 5: Graph Results for Metrics

5.1.1.3 Under-Sampling the Training Data

To improve the model's baseline performance, I used under-sampling on the training data and reduced the number of real job postings by 50%. The training set for the previous model had 10,889 real job postings and 554 fraudulent postings. The under-sampled training set has the same number of fraudulent postings (554), but the real postings are reduced to 5444. The model was then recompiled using the same optimizer, loss function, and metrics as the previous model; but the training inputs were the under-sampled text and numeric data. Validation data never changed from the initial model.

After running for five epochs, the model's performance did not significantly improve; the training AUC was 0.5069 (FN: 511; FP: 446) and the validation AUC was 0.5010 (FN: 139; FP: 0.0). The model did not overfit, but once again, its performance was no better than a random classifier. The graphs in Figure 6 display the results of each metric for the under-sampled training data with the original validation data throughout the five epochs.

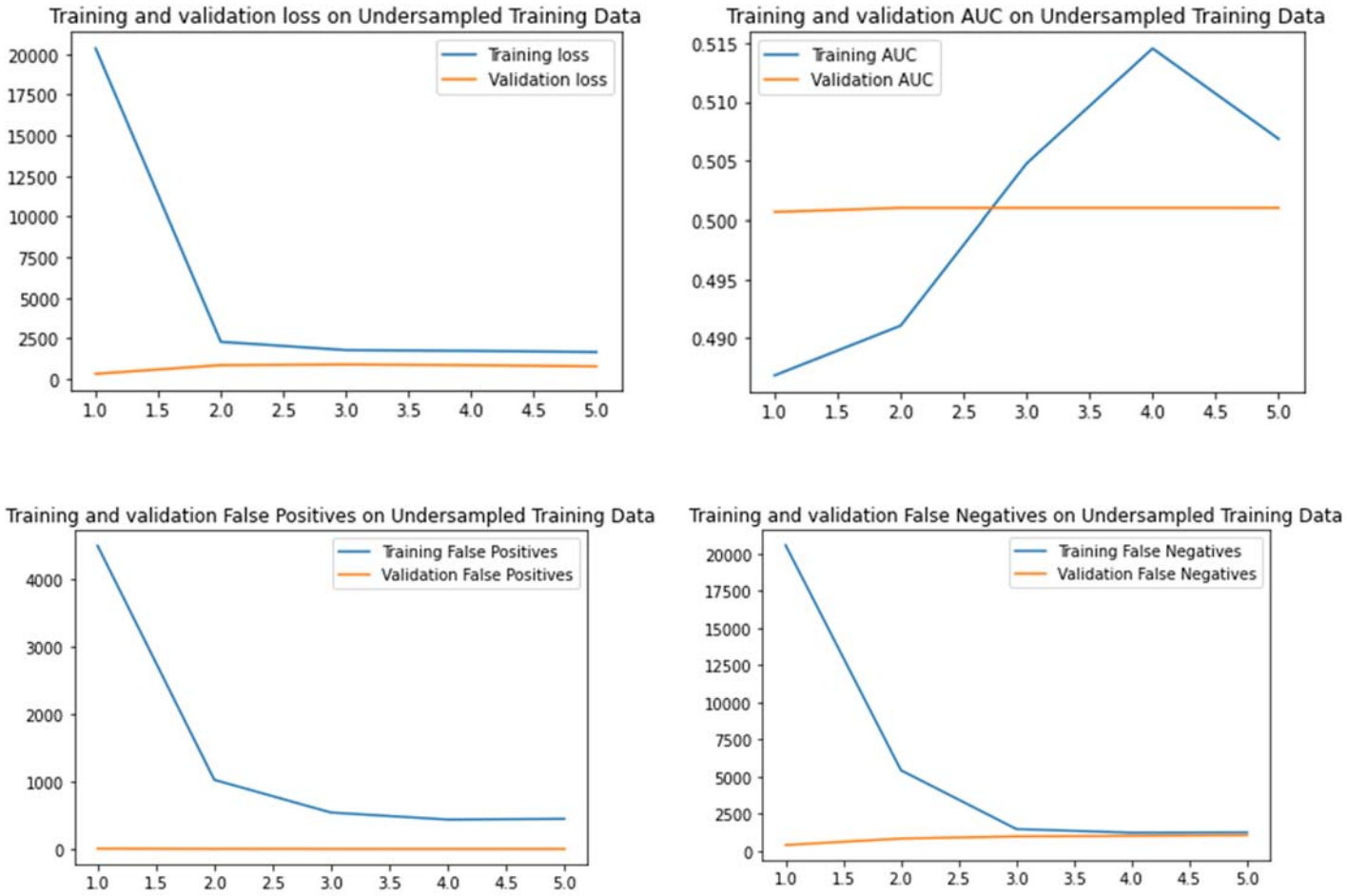

Figure 6: Graph Results for Metrics on Under-Sampled Training Data

5.1.2 Text-Only Data

5.1.2.1 Creating the Text-Only Datasets

Due to the poor AUC performance of both RNN models using text and numeric data, it was decided to focus solely on the text data. Unlike the previous models, the text input would not be 128 tokens from the Full Text column but instead much of the text would be kept as separate columns with 128 tokens from each column being passed as inputs to the model. The text-only dataset was copied from the preprocessed data frame that had been saved before the categorical variables were one-hot-encoded. All numeric variables were dropped and only the text columns remained. After combining some of the shorter text columns together, there were five inputs total to be passed to the models: title_loc_description (Title, Location, Description); cat_vars (Department, Employment Type, Required Experience, Required Education, Industry, Function); benefits; company_profile; and requirements. The data was split again into new train, validation, and test sets using the same ratio and random seed as the previous splits so the same rows would be selected.

5.1.2.2 Building the Text-Only Model

Raw text cannot be passed to the models so first each text group needed to be vectorized into 128 tokens. Each input passes through its own Text Vectorization layer, Embedding layer, and Bidirectional LSTM layer; and all layers have 128 units. All five inputs are then concatenated and passed through a Dense layer with Relu activation. The output layer consists of 1 unit with Sigmoid activation. In total, this model has over 1 million parameters (1,438,785) and all are trainable. Figure 7 shows the graph of its architecture.

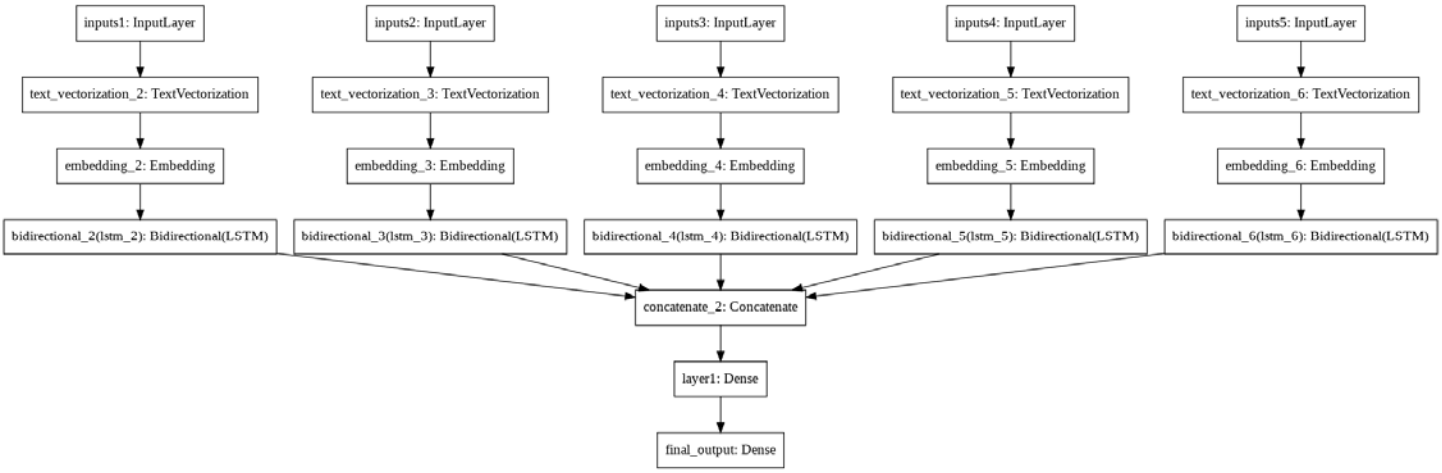

Figure 7: Baseline RNN Model Architecture (Text-Only Inputs)

5.1.2.3 Training the Text-Only Model

The model was again compiled using Adam optimizer with a learning rate of 1e-5, and Binary Cross-Entropy loss; and it measured the metrics of Loss, Binary Accuracy, AUC, False Positives, and False Negatives. Each text group was passed as an input to the model and after running for five epochs, the model produced a training AUC of 0.7930 (FN: 554; FP: 0) and a validation AUC of 0.7972 (FN: 139; FP: 0). The graphs in Figure 8 display the results of each metric for the training and validation data throughout the five epochs. The model did not overfit, and the validation results were better than the training results.

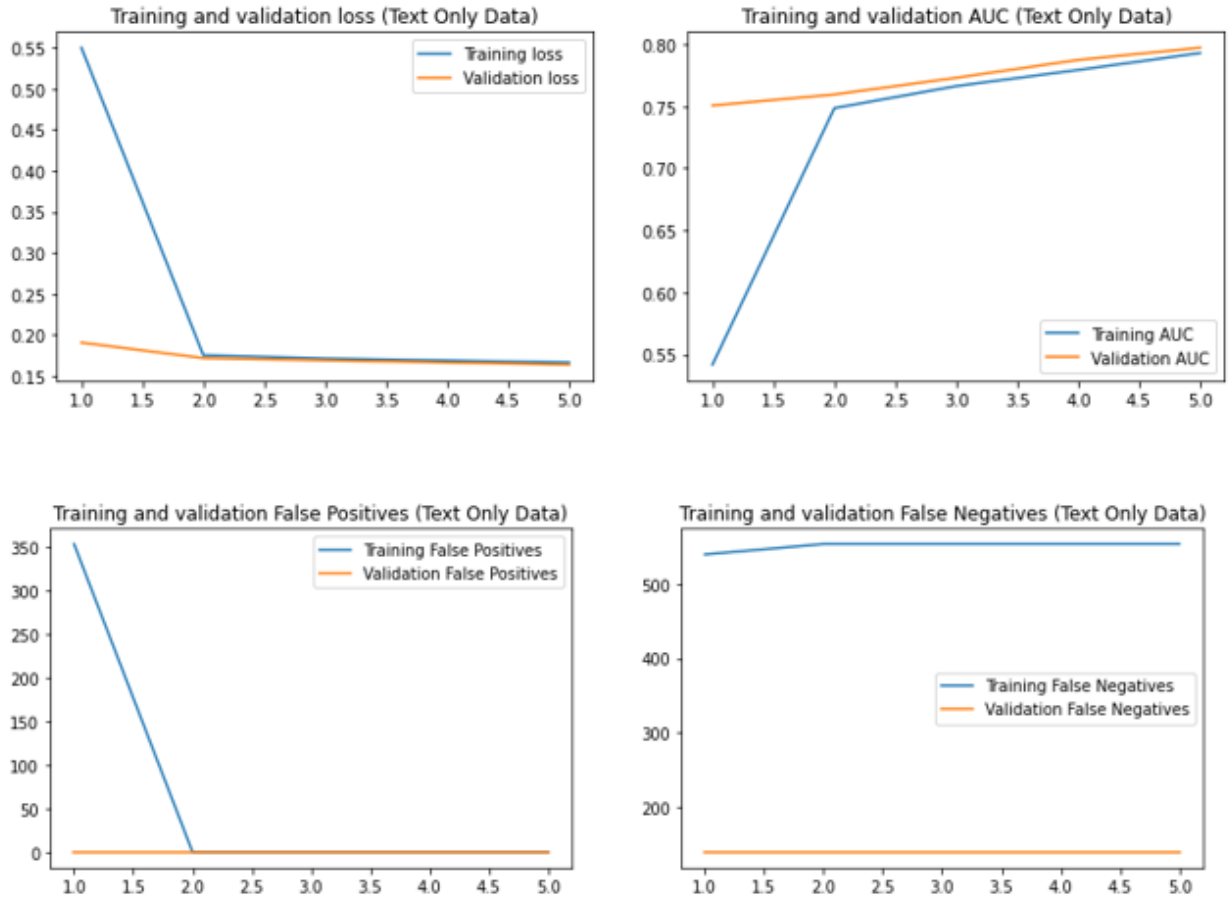

Figure 8: Graph Results for Metrics on Training Text-Only Data

5.2 BERT MODEL

Due to the high performance of the baseline RNN model on the text-only data, I focused solely on the text data for the BERT model and did not apply its outputs to any other model. The table below shows the possible model architectures for BERT Miniatures [10].

	H=128	H=256	H=512	H=768
L=2	2/128 (BERT-Tiny)	2/256	2/512	2/768
L=4	4/128	4/256 (BERT-Mini)	4/512 (BERT-Small)	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512 (BERT-Medium)	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	12/768 (BERT-Base)

Figure 9: Graph of BERT Miniatures [10]

5.2.1 Defining the Small-BERT Model

Every BERT model has a specific encoder and preprocessing layer that must be used to produce a RaggedTensor of shape input word ids, input mask, and input type ids. For this project, I used the uncased Small-Bert encoder [11] and its corresponding BERT preprocessor [12].

Initially a preprocess model was designed based on [13] to create tokens with the maximum length of 512. However, the available resources to train the model on 512 tokens per input resulted in multiple crashes of the system so the preprocessing model was revised to only produce the default of 128 tokens per input. Each text group was passed to the preprocessing model and its 3D ragged tensor was returned for future use. The graph on the left in Figure 10 displays the preprocessing layers that each input will pass through; the tokenizer layer will split the text into segments of 128 tokens, and the packer layer will perform the required preprocessing of text that BERT requires (i.e., adding the start and end token ids).

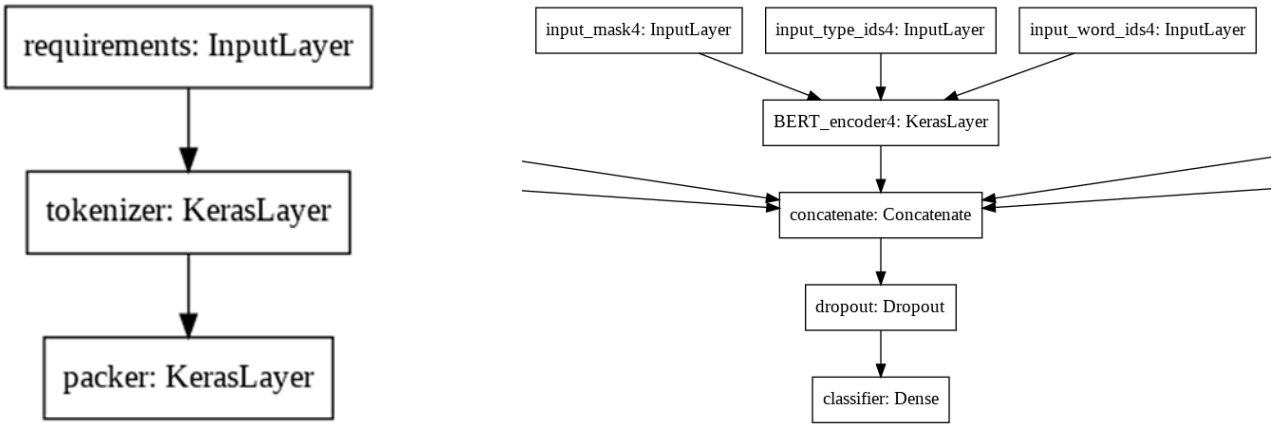

Figure 10: Graph of Preprocessing Layer for Requirements Input (left) and Truncated Graph of Small-Bert Model (right)

To build the BERT model, each input must be converted into a 3D Ragged Tensor of shape (input word ids, input mask, input type ids). The tensor is then passed as input to the encoder, which outputs the text's word embeddings as a "pooled output". These pooled outputs for all input texts are combined and then passed through a single Dropout layer (0.1). The model's final output layer has 1 neuron with Sigmoid activation. This general architecture was inspired by the Tensorflow tutorial [14] and the image on the right in Figure 10 displays part of the model's overall architecture to show how the 3D tensor is passed through the model. In total, there are over 1 million parameters (1,438,785) and all are trainable except the initial five 3D input tensors.

5.2.2 Fine-Tuning BERT

The same loss function and metrics that had been used with the RNN baseline model were again applied to Small-BERT. Due to the resources available, the model ran for only one epoch, but this is oftentimes enough to sufficiently fine-tune BERT. Guided by [15], I compiled the BERT model with a Warmup Learning Scheduler using the AdamW optimizer. Once the model was finished, it produced a training AUC of 0.8055 (FN: 520; FP: 27) and a validation AUC of 0.9625 (FN: 109; FP: 1). The model did not overfit and so no additional tuning of the hyper-parameters was done.

6 Evaluating, Tuning, and Improving the Model

6.1 Baseline RNN Model

The initial parameters to the baseline RNN model were influenced by previous course assignments and [16]. For the model with text and numeric inputs, no additional tuning was done because of its low AUC of roughly 50%. Instead, the input data was focused on the text components of the job postings which produced significantly higher results (see graphs in Figure 8).

There was no overfitting during training and so the same parameters were applied to the testing set (after having combined the training and validation data) and passed into the model. The model again ran for five epochs; the final training AUC was 0.8009 (FN: 693; FP: 0). and the final testing AUC was 0.7864 (FN: 173; FP: 0). Although the test data's AUC was slightly lower than that from the combined training & validation data, there was a significant reduction in the number of False Negatives. The graphs in Figure 11 display the results of each metric for the five epochs.

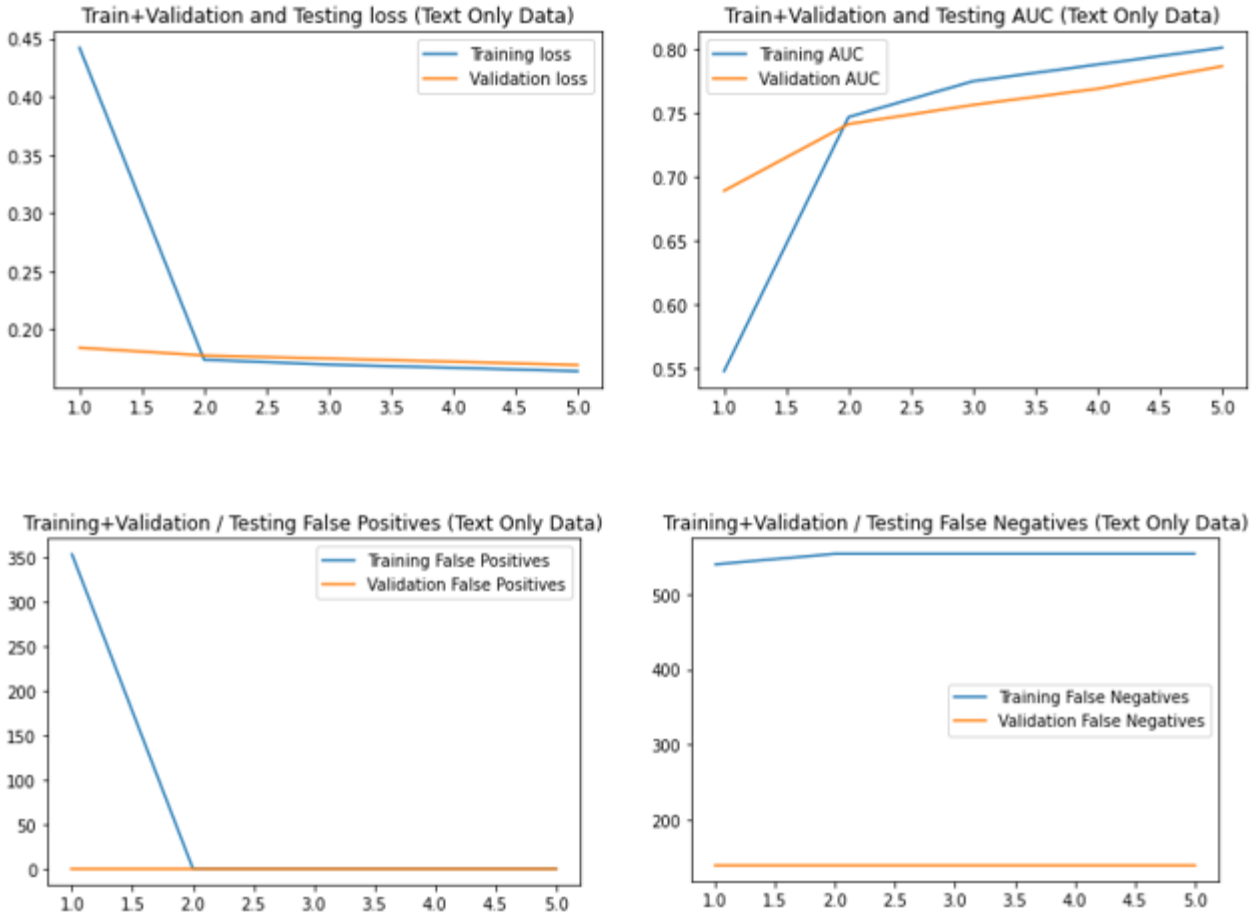

Figure 11: Graph Results for Metrics on Training Text-Only Data

6.2 Small-BERT Model

For both the preprocessing function, and the function to build the BERT model, the initial parameters were inspired by official Tensorflow tutorials [12][13][14][15]. Training the model for a single epoch resulted in a training AUC of 0.8055 but a validation AUC of 0.9625. These results are significantly higher than the RNN baseline model, so no additional changes were made to the initial parameters. Once trained, the test data was passed into the model using the evaluate() method and resulted in a final AUC of 0.9637(FN: 119; FP: 4).

7 Expanding the Project

While the models produced satisfactory results when predicting fraudulent job postings from a very imbalanced dataset, there are ways to further explore the data and potentially improve these results.

7.1 Augmenting Data with Back Translation

Back translation is when a text is translated to a source language and back via a second language. It is often used to verify the accuracy of translated legal documents but for this project, it could also be applied to over-sample the fraudulent posts in the training data. I would use the BackTranslation package from PyPi[17] to translate the fraudulent job postings from English into French, Spanish, Chinese, and Russian (and back to English). Applying back translation with four languages will result creating a ratio where the fraudulent postings are approximately 45% of the dataset. With the first language, only the original 866 fraudulent postings in English are used; and then each subsequent language will be applied to all existing augmented posts, doubling the number of augmented posts with each run of back translation.

7.2 Generating Fake Job Postings with GPT-2

I would like to extend the findings from BERT and train a GPT-2 model to create a realistic fake job posting based on its training data. This newly created job posting would then be passed back into the existing RNN and BERT models to see how they would classify it. This idea was inspired by a guided project on Coursera [18], but additional research would need to be done to learn more about GPT-2 and how to apply its output to the baseline RNN and BERT models.

7.3 Combine Numeric Features and Text Groups for Model Inputs

The inputs to the initial RNN baseline model were the numeric features and 128 tokens from the Full Text column; and the resulting AUC was no better than a majority classifier. With the significant improvement in both models' performance when the text data was separated into five groups, I would like to explore whether the models' performance would further improve by providing the numeric features (Work Remote, Has Questions, Has Company Profile, Salary Low, Salary High) as additional inputs, either grouped together or kept as separate columns.

7.4 Apply Classic BERT (BERT-Base)

This project could only run the data through a Small-BERT model due to a lack of resources in Colab, but I would like to explore whether Classic BERT (BERT-Base) would provide better AUC scores than Small-BERT; and then evaluate the test data on the best-performing BERT model.

7.5 Increase Number of Tokens

BERT allows a maximum of 512 tokens but only 128 tokens could be passed to the models due to insufficient resources with larger token inputs. However, I would like to see whether a model with a larger number of tokens for each input would improve on the results of this project. Applying larger token sizes to each model could also be combined with the idea presented in Section 7.3 – using text-only inputs or combining text & numeric inputs.

7.6 Hyper-Parameter Tuning

Although the models were not overfitting during the training data, the hyper-parameters could be tuned to further separate the validation loss from the training loss. It is important that sufficient resources be available to tune the model's hyper-parameters because without tuning, it took approximately 30 minutes per epoch to train the RNN baseline model, and the single epoch to train Small-BERT took approximately 3.5 hours. This time and resources required will significantly increase with an increase in input token size and the number of total inputs to the model.

8 Conclusion

Based on the results of this project, the BERT model is the best option among those tested when trying to predict fraudulent job postings from a highly imbalanced dataset, using only the text components of the posting. Although the final test AUC was 0.9637, there are still improvements that can be made to further reduce the values for False Negatives and False Positives.

The fact that the model predicted 109 real job postings as being fake indicates to the job poster that the details provide in the job posting share many characteristics with fraudulent posts. It is important when posting a job that the posting is as complete as possible, so the job seeker understands the position's requirements and responsibilities. It is also important that the company has an online presence (company website, social media accounts, etc.) to support its authenticity as a legitimate company; and if the salary range is provided, that it is realistic and clear.

Looking at the number of False Positives, the model predicted 4 fraudulent postings as being real. While this may not seem as problematic as the False Negatives, having a more authentic fraudulent posting can greatly affect the job seeker. If the job posting does not raise any red flags and they send a resume, the scammer who posted the fraudulent job now has access to that job seeker's personal job and contact details from their submitted resume. Because the job seeker believes it to be a legitimate company and job position, any future requests for additional personal information (i.e., sending copies of government documentation), or even requests for money to "speed up the application process", would be more believable to the job seeker, resulting in possible identity or financial theft.

9 REFERENCES

[1] J. T. O'Donnell. "6 warning signs a job posting is fake." LinkedIn.com. https://www.linkedin.com/pulse/6-warning-signs-job-posting-fake-jt-o-donnell (accessed May 7, 2021).

[2] P. Jones. "How to spot a fake job posting." TheJobNetwork.com. https://www.thejobnetwork.com/how-you-can-spot-a-fake-job-posting (accessed May 8, 2021).

[3] "How to identify scam job titles on Indeed." Indeed.com. https://support.indeed.com/hc/en-us/articles/360050449471-How-To-Identify-Scam-Job-Titles-on-Indeed (accessed May 8, 2021).

[4] S. Bansal. "[Real or fake] fake job posting prediction." Kaggle.com. https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction (accessed Apr 11, 2021).

[5] "Text classification using Bert (99% acc, 98.7% f1)." Kaggle.com. https://www.kaggle.com/ahmed0sultan/text-classification-using-bert-99-acc-98-7-f1 (accessed May 8, 2021).

[6] "Bert vs bidirectional LSTM." Kaggle.com. https://www.kaggle.com/niduttnb/bert-vs-birdirectional-lstm (accessed May 8, 2021).

[7] "Real or fake job-postings with bidirectional LSTM." Kaggle.com. https://www.kaggle.com/gauravsahani/real-or-fake-job-postings-with-bi-directional-lstm (accessed May 8, 2021).

[8] "Fake job post prediction: Countvec, Glove, Bert." Kaggle.com. https://www.kaggle.com/vikassingh1996/fake-job-post-prediction-countvec-glove-bert (accessed May 8, 2021).

[9] T. Rajapakse. "Classification Models." SimpleTransformers.ai. https://simpletransformers.ai/docs/classification-models/ (accessed May 8, 2021).

[10] "BERT miniatures." Huggingface.co. https://huggingface.co/google/bert_uncased_L-4_H-512_A-8 (accessed May 8, 2021).

[11] "small_bert/bert_en_uncased_L-4_H-512_A-8." Tfhub.com. https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/2 (accessed May 6, 2021).

[12] “bert_en_uncased_preprocess.” Tfhub.com. https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3 (accessed May 6, 2021).

[13] “Solve GLUE tasks using Bert on tpu.” Tensorflow.org. https://www.tensorflow.org/tutorials/text/solve_glue_tasks_using_bert_on_tpu#preprocess_the_text (accessed Apr 28, 2021).

[14] “Classify text with Bert.” Tensorflow.org. https://www.tensorflow.org/tutorials/text/classify_text_with_bert (accessed Apr 28, 2021).

[15] “Fine-tuning a Bert model.” Tensorflow.org. https://www.tensorflow.org/official_models/fine_tuning_bert (accessed Apr 28, 2021).

[16] “Text classification with an RNN.” Tensorflow.org. https://www.tensorflow.org/tutorials/text/text_classification_rnn (accessed Apr 26, 2021).

[17] Z. Wu. “BackTranslation 0.3.1.” PyPi.org. https://pypi.org/project/BackTranslation/ (accessed Apr 30, 2021).

[18] A. Anastassiou. “Generating new recipes using GPT-2.” Coursera.org. https://www.coursera.org/projects/generating-new-recipes-python (accessed Apr 20, 2021).