NLP Project Report

Title: Fake Job Postings Detection

Members of the group:

1. Aral Açikalin
2. Amey Chandrakant Darekar
3. Braian Olmiro Dias

1 Introduction

The main goal of the project is to design and implement a Natural Language Processing system capable of correctly classifying Fake Job Postings from a dataset containing categorical, text, and binary information such as location, department, company profile, job description, requirements, etc. The given dataset has 18.000 job postings, 17.200 are real (approximately 95%) and 800 are fake (approximately 5%).

The performance of the system will be measured using three metrics: (1) F1-score for the “fake” class, (2) Micro-averaged F1-score and (3) Macro-averaged F1-score. Metric (1) is the main metric and should drive the progress of the project. In order to evaluate these metrics and report final scores, we will have access to train, development and test sets, where the latter does not contain labels and will be evaluated externally by the teaching staff.

The following sections of this report detail the research that was done prior to the design phase, the reasoning behind our approach, and finally, the results and conclusion about our work.

2 Related Work

The “Fake Job Postings” dataset is relatively small (14.000 instances on the training set) for training a language model from scratch. To extract the most from state-of-the-art language models, such as BERT, we need to understand the different approaches for fine-tuning a pre-trained model on a classification task. Sun et al.[6] demonstrated with substantial experiments that fine-tuning a pre-trained model achieves state-of-the-art results on widely-studied text classification datasets such as IMDb and Yelp and documented that pre-trained models on large corpus can be used for other NLP tasks without retraining from scratch. The authors investigated 1) Fine-tuning strategies, 2) Further pre-training and 3) multi-task fine-tuning. Their experiments provided insights into model construction such as how to deal with long texts and which layer from BERT has the best performance for classification.

Qasim et al.[5] propose removing URLs and special characters in the data with pre-trained models since these unknown tokens may not give valuable information about the data in the classification task. Kumar[3] compared GRU networks to ensamble text classification methods like random forest classifiers and logistic regression and showed that self-attention really improves performance for text classification tasks. Additionally, the author claims including the length of text tokens and removing input fields that have more than 60% missing values can be useful in classification performance. Although the research of the author is reasonably new, they did not mention transformer architectures but the techniques discussed can be applied to transformers.

The classifier performance is typically evaluated with predictive accuracy, which is not a useful measure for an imbalanced dataset. Chawla et al. [1] propose an over-sampling approach in which the minority class is over-sampled by creating “synthetic” examples rather than by over-sampling with replacement. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the ‘k’ minority class nearest neighbors. The proposed approach significantly improved performance with several classifiers over imbalanced datasets.

Padurariuab et al. [4] discussed several of the data balancing techniques including resampling methods like random oversampling, SMOTE, SMOTE-SVM, and other techniques like thresholding and cost-based direct methods. Thresholding is used to threshold the output probability of the classifier to fit the imbalance of the data better. The cost-based direct method is used to penalize the model when misclassification of small classes happens. Franks et al. [2] discussed the text classification task and the challenge of working with imbalanced data in a real-world scenario. The author compared many types of statistical classifiers, CNNs and language models, with combination of data sampling with SMOTE and random sampling. The author claimed that while the use of resampling to achieve balance in data improved performance in statistical and Neural network types of approaches, they failed to surpass the performance of Language models even without sampling.

3. Methods

For our implementation we decided to use as the main model a Transformers-based model using CLS token, adding binary and categorical features to the classifier layer. The idea is to concatenate text features as input for the Transformers model, and concatenate the binary and categorical features into the classifier layer. More details about data processing and feature selection, and model architecture are presented below:

3.1 Data preprocessing and feature selection:

The initial training data available contains 14.000 observations of 18 features, as shown in Figure 1 below.

```
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Unnamed: 0           14304 non-null  int64
 1   job_id               14304 non-null  int64
 2   title                14304 non-null  object
 3   location             14029 non-null  object
 4   department           5095 non-null   object
 5   salary_range         2311 non-null   object
 6   company_profile      11626 non-null  object
 7   description          14303 non-null  object
 8   requirements         12127 non-null  object
 9   benefits             8558 non-null   object
 10  telecommuting        14304 non-null  int64
 11  has_company_logo     14304 non-null  int64
 12  has_questions        14304 non-null  int64
 13  employment_type      11496 non-null  object
 14  required_experience  8613 non-null   object
 15  required_education   7789 non-null   object
 16  industry             10354 non-null  object
 17  function             9125 non-null   object
 18  fraudulent           14304 non-null  int64
```

Figure 1: description of the training data.

After analyzing the training data we decided to perform the following steps to preprocess the dataset:

- Remove features with more than 60% of missing data, plus useless features (e.g: job-id).
- Concatenate all text features into one large text using the [SEP] token between each individual text. We also clip text features which are too long: “description” contains a maximum of 200 tokens, and “requirements” and “company_profile” maximum of 30. The remaining text features "title", "location","employment_type","industry" and "function" are not clipped.
- Concatenate binary and categorical features from the dataset into the classifier layer of the network. From the dataset we used features "has_company_logo", "telecommuting", "has_questions", plus an engineered feature “text_length” containing the number of tokens of the final text feature.

As the tokenizer for the text feature, the uncased version of the BERT tokenizer is selected. Before tokenizing the text it is cleaned so that the more meaningful text will be present for the model. Non-ASCII characters and tags like #E-MAIL, #URL, and #PHONE were removed from the text.

3.2. Model architecture

We will use BERT as a baseline for our solution, as Padurariuab et al. and Qasim et al.[5] proposed. We use a pre-trained BERT model available on Hugging Face and fine-tune all layers of the model using the concatenated text feature described earlier.

Since our data analysis showed categorical and binary features have some predictive power, we will concatenate them into the classification layer, as follows:

- Linear layer with BERT output plus 4 binary and categorical features as input (BERT size +4, BERT size + 4).
- ELU activation.
- Dropout.
- Linear layer (BERT size +4, 2)

The training procedure will use as input both text and other features and Cross Entropy Loss as loss function.

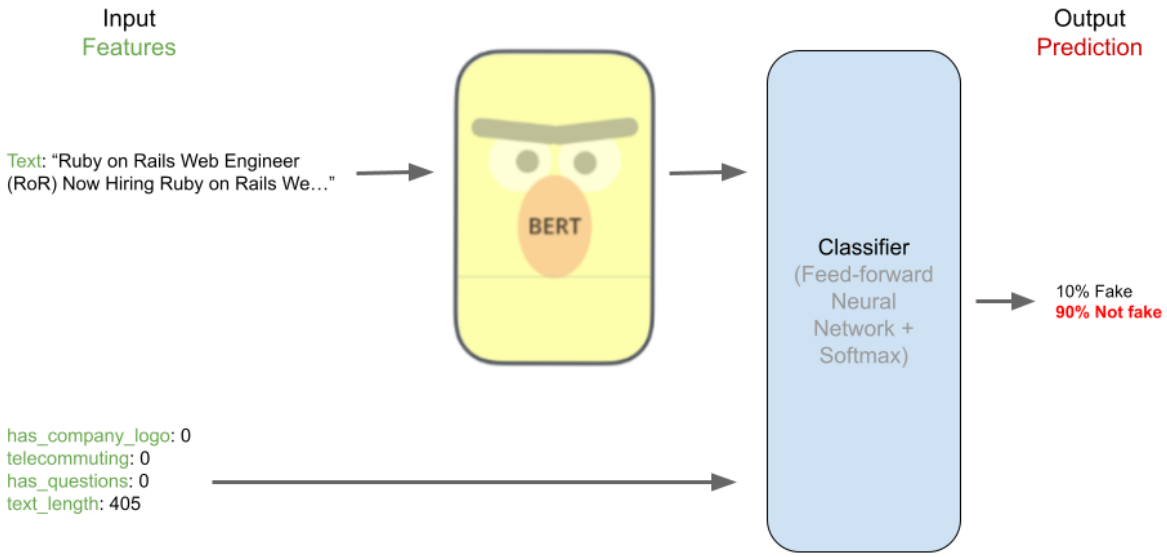

Figure 2: Our proposed architecture.

4. Results

With our architecture, we tried multiple hyperparameters and base pre-trained models. Mostly we experimented with dropout, learning rate, and the base BERT pre-trained model. For dropout, we experimented with the range 0.15-0.75 and for learning rate, we experimented with 1e-5 and 2e-5, and for base pre-trained models we experimented with distilbert-base-uncased and bert-base-uncased. However, our best results were with 0.15 dropout, 1e-5 learning rate, and distilbert-base-uncased combination. To decide on the best model we used the dev set and metrics like accuracy and different F1 scores.

Model Configuration			Outcome			
Model	Learning Rate	Drop-out Rate	Dev accuracy	Dev f1 macro	Dev F1 Fake class	Dev F1 micro
Distilbert-base-uncased	**1e-5**	**0.15**	**0.995**	**0.969**	**0.940**	**0.995**
Bert-base-uncased	1e-5	0.15	0.993	0.955	0.914	0.993
Distilbert-base-uncased	2e-5	0.5	0.988	0.915	0.837	0.988

Table 1. Dev set results

After training the model Distilbert-base-uncased, 0.15 dropout, 1e-5 learning rate we thought training full BERT with the same hyperparameters might give slightly better results, however, full BERT actually decreased the performance of Distil-BERT. Distil-BERT usually loses 3% performance compared to full BERT however in this case full BERT loses performance. This might be happening because of the imbalance in the data. Because of the imbalance of the data, the model tends to overfit the data and bigger models like full BERT might be affected by this more. Test set results of the best model can be seen in the below table.

Model Configuration			Outcome		
Model	Learning Rate	Drop-out Rate	F1 macro	F1 Fake class	F1 micro
Distilbert-base-uncased	**1e-5**	**0.15**	**0.991**	**0.880**	**0.937**

Table 2. Held-out set results

5 Discussions

For these results we did not do any balancing of the data but rather focused on cleaning the data. For future steps, we may also apply some balancing techniques discussed by Chawla et al.[1] and Padurariuab et al.[4] and compare with our results. We did not pre-train the base BERT model with a similar dataset as described in experiments proposed by Qasim et al[5]. We decided to only fine-tune our model with the given Fake Job Postings dataset since this option provided a good baseline, however we acknowledge a complete pipeline with pre-train and fine-tune can lead to better results.

6 Conclusions

The imbalance in the data is the biggest problem of this task however pre-trained language models are powerful enough that they can provide good results with these settings. The second problem with the data is that the text fields are very long and exceed the limits of pre-trained models. Cleaning the data can shorten these text fields and help the model get a more meaningful text. But we can't say for certain that cleaning the data helps and it needs more investigation. To get around the pre-trained model's input limit another approach can be tried. Training different BERT models for each text field of the data can allow the model to have more text input. However, using multiple BERT models will decrease the model's inference and training time performance significantly. This can be researched in the future.

7 References

1. Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.

2. Franks, Jason. "Text Classification for Records Management." Journal on Computing and Cultural Heritage (JOCCH) (2022).

3. Kumar, Ankit. "Self-Attention GRU Networks for Fake Job Classification."

4. Padurariu, Cristian, and Mihaela Elena Breaban. "Dealing with data imbalance in text classification." Procedia Computer Science 159 (2019): 736-745.

5. Qasim, Rukhma, et al. "A Fine-Tuned BERT-Based Transfer Learning Approach for Text Classification." Journal of healthcare engineering 2022 (2022).

6. Sun, Chi, et al. "How to fine-tune bert for text classification?." China national conference on Chinese computational linguistics. Springer, Cham, 2019.