Job Posting Classifier

Classifying Job Postings as Real or Fake for Jobbies Inc.

Joe Wiley

Capstone Project #2

The problem:

My goals are to answer these questions:

How accurately can we predict whether a job listing is real or fake? What factors are most important for prediction, and what models give the best results?

Background:

Jobbies Inc., a job search company, wants to remove fake job postings in order to increase credibility and employer relations. We have a database of job postings provided by Kaggle where the postings are labeled as ‘real’ or ‘fake’. We aim to use NLP techniques and implement a few different models to predict whether a given job posting is real or not.

Dataset from Kaggle here:
https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction

My approach:

Feature Selection and Engineering:

The largest hurdle in this project was manipulating the data into a form that would not only be usable in the modeling stage, but to extract as much signal as possible.

To give an example, the ‘title’ feature (job title), was not structured in any way and depended entirely on how individuals decided to enter their data. I didn’t want to throw out the column entirely, but it would also be unacceptable to essentially have a unique job title category from each row of my data. I managed to extract the most common words like ‘engineer’ and ‘manager’. The ‘title’ feature could then be split into a few important features corresponding to the most common words found in it. For example, ‘title_engineer’ indicates whether the word engineer appears in the title.

Each one of my initial 17 features required extensive feature engineering as described above, and in the end that expanded into 86 total features! Three of those features- ‘company_profile’, ‘description’, and ‘requirements’- were text that could further be manipulated and expanded when needed.

Modeling:

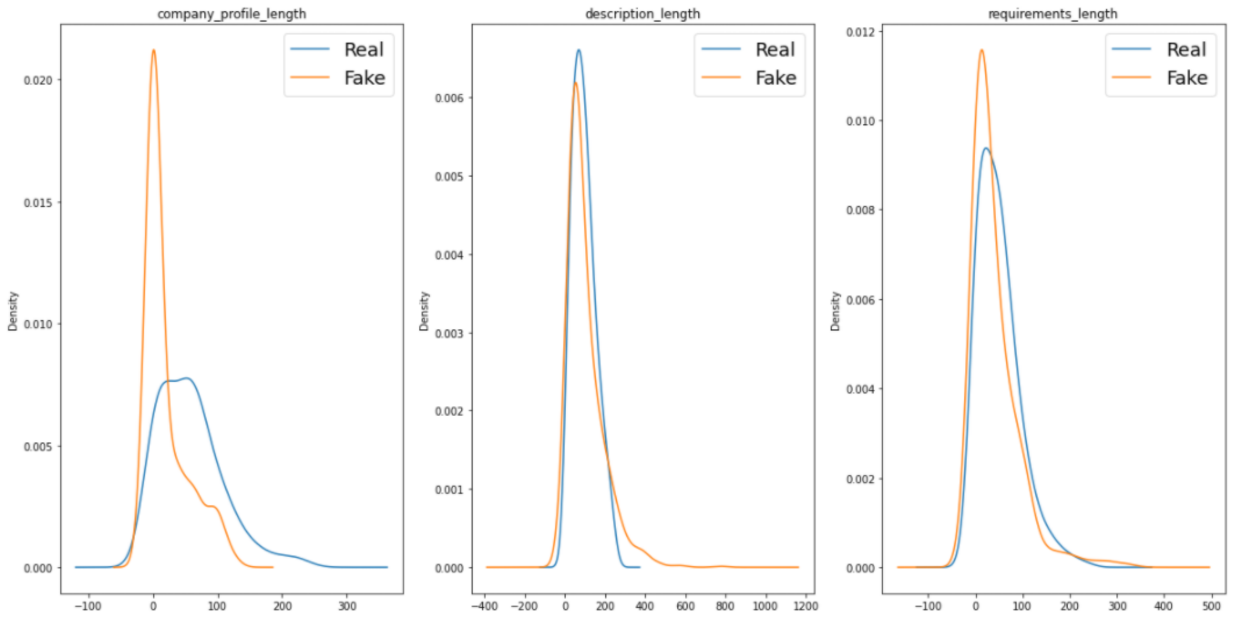
When modeling, I needed to account for unbalanced classes in my dataset- 95% of my data was real postings and only 5% was fake. I decided to use F1 score as it gives a more balanced representation of a model’s success when the data is unbalanced. Accuracy, for example would give a 95% score to a dummy classifier that only predicted real.

After choosing a single score to measure my models against each other, I ran my data through a few models: Naive Bayes, Random Forest, and Logistic Regression. Naive Bayes was chosen because it is the ‘classic’ model for classifying on text data. Random Forest is known to perform very well out of the box, which I needed since I didn’t have time to tune hyperparameters. Logistic Regression was chosen as an easy and fast to implement classifier that could be compared to the other two.

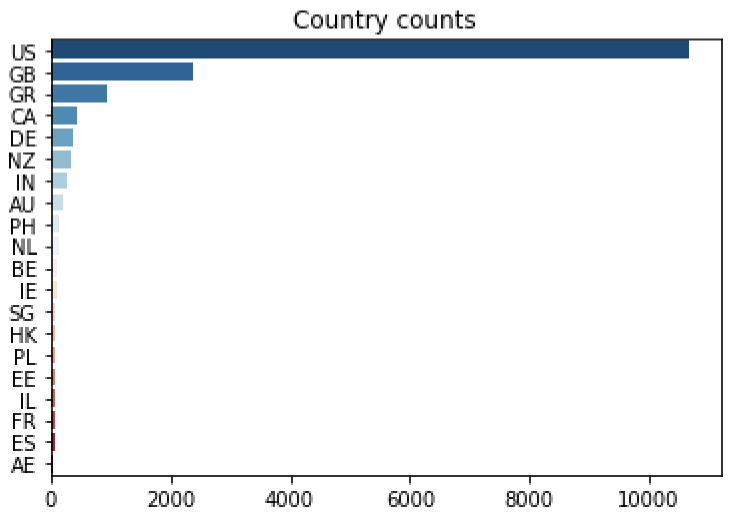
Findings:

EDA:

The first variable I investigated was word length of the text features. I found the average word length differed between the two classes- real posts tended to be longer, fake posts shorter, and a large number of fake posts had no text at all in these features (‘company_profile’, ‘description’, and ‘requirements’).



I also had some interesting findings when breaking the posts down by country, and state. The US had the majority of the data- 10,000 out of a total of 17,000 postings. I see the US rate of fake postings as a nice benchmark to compare other countries to, or individual state rates to. For reference, the US rate is 6.8% fake. See the Country counts diagram below for an idea of where the data is coming from:



The country with the highest rate was Malaysia at 57%! Malaysia only had 21 total postings, so maybe it's not as important as some other countries and states. Some examples of high-posting countries with surprising results: Great Britain has 2,000 postings with only 1% fake rate, and Greece has 1,000 postings with 0% fake rate! That's literally no fake postings out of 1000 from Greece. I suspect both Greece and Great Britain have already been filtered before reaching this data. For surprising states, I found that Texas has 1,000 postings with a fake rate of 15.6%- over double the national rate. I'm going to read my Texas job postings more carefully next time. See the tables below:

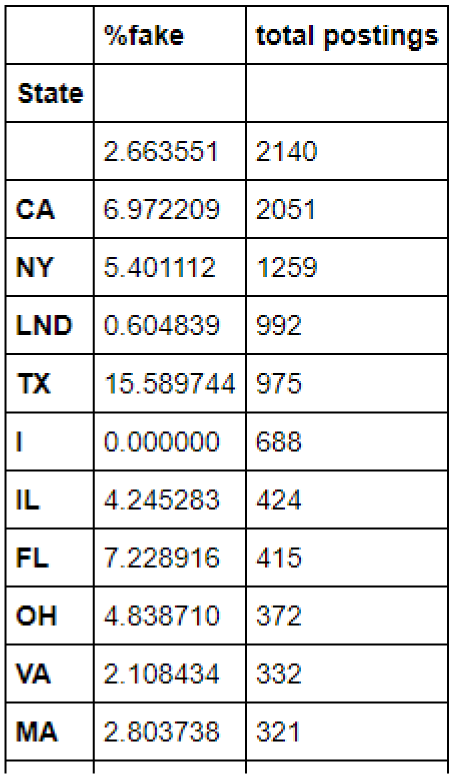
Country, sorted by %fake:

	%fake	**total postings**
Country		
MY	57.142857	21
BH	55.555556	9
TW	50.000000	4
QA	28.571429	21
AU	18.691589	214
ID	7.692308	13
US	6.850601	10656

Country, sorted by total postings:

	%fake	**total postings**
Country		
US	6.850601	10656
GB	0.964765	2384
GR	0.000000	940
CA	2.625821	457
DE	0.000000	383
NZ	0.000000	333
IN	1.449275	276
AU	18.691589	214
PH	0.757576	132
NL	0.000000	127
BE	0.000000	117

State, sorted by total postings:

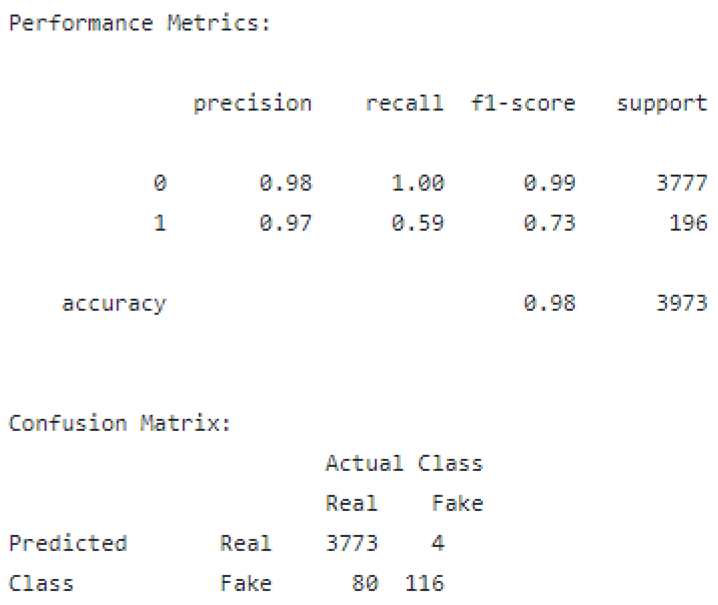


	%fake	total postings
State		
	2.663551	2140
CA	6.972209	2051
NY	5.401112	1259
LND	0.604839	992
TX	15.589744	975
I	0.000000	688
IL	4.245283	424
FL	7.228916	415
OH	4.838710	372
VA	2.108434	332
MA	2.803738	321

Note that the top state is blank, because many entries had no state listed.

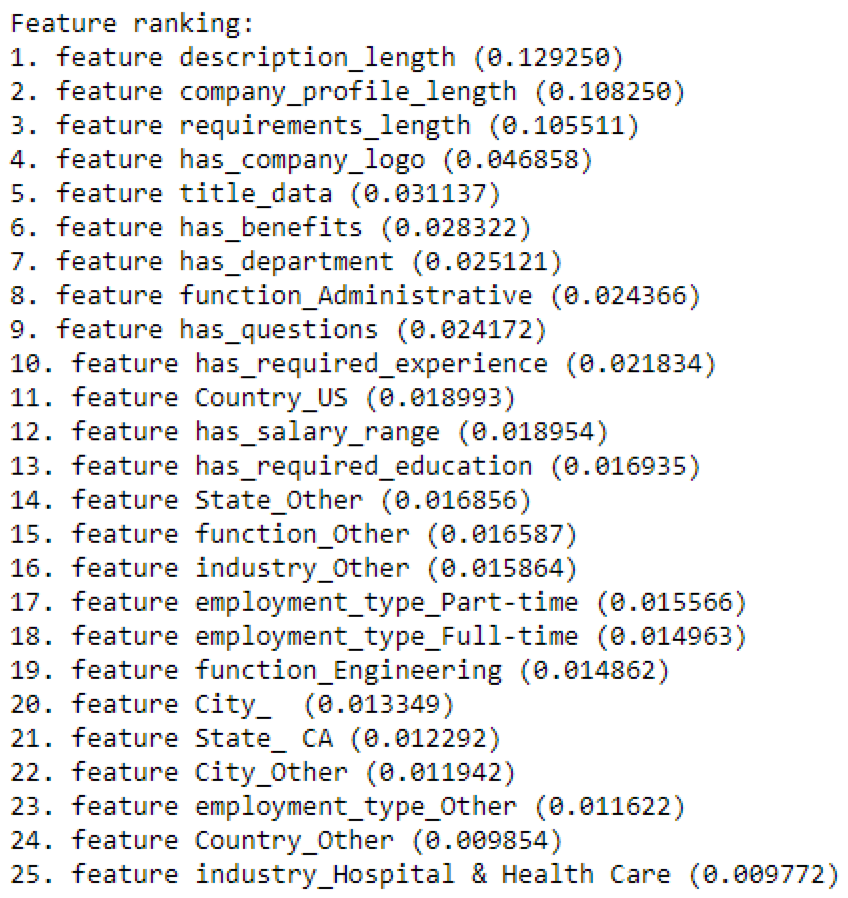
Modeling:

Of my three models, I found that the Random Forest model performed the best under my chosen metric, F1. Random Forest had F1=.73. For reference, my best Naive Bayes model had F1=.45 and my Logistic Regression model had F1=.34. My final scores are displayed below:



```
Performance Metrics:

              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3777
           1       0.97      0.59      0.73       196

    accuracy                           0.98      3973


Confusion Matrix:
                        Actual Class
                        Real    Fake
Predicted       Real    3773    4
Class           Fake      80  116
```

Random forest model also gives feature importances, displayed below:



```
Feature ranking:
1. feature description_length (0.129250)
2. feature company_profile_length (0.108250)
3. feature requirements_length (0.105511)
4. feature has_company_logo (0.046858)
5. feature title_data (0.031137)
6. feature has_benefits (0.028322)
7. feature has_department (0.025121)
8. feature function_Administrative (0.024366)
9. feature has_questions (0.024172)
10. feature has_required_experience (0.021834)
11. feature Country_US (0.018993)
12. feature has_salary_range (0.018954)
13. feature has_required_education (0.016935)
14. feature State_Other (0.016856)
15. feature function_Other (0.016587)
16. feature industry_Other (0.015864)
17. feature employment_type_Part-time (0.015566)
18. feature employment_type_Full-time (0.014963)
19. feature function_Engineering (0.014862)
20. feature City_  (0.013349)
21. feature State_ CA (0.012292)
22. feature City_Other (0.011942)
23. feature employment_type_Other (0.011622)
24. feature Country_Other (0.009854)
25. feature industry_Hospital & Health Care (0.009772)
```

Interesting to note that the lengths of my text columns were all found to be the most important features.

Further Research:

When I return to this project, the most important thing to implement is to extract more signal from the text features. I would like to retry all my previous models with tf-idf applied to the text features, instead of the bag of words that I used here. I would also like to try an embedding, like word-to-vec to extract more meaning from the text columns. Finally, I would like to try a deep learning model to this data. Deep learning models are potentially extremely powerful and I don't have experience with them, so I'd like to try it out.

Recommendations to Jobbies Inc.

I would recommend that Jobbies employs my Random Forest model to filter out fake job postings. My model filtered out 59% of fake postings, while only removing .1% of real postings! The .1% of false positives can be mitigated, for example with increased customer service. A 59% reduction in fake postings is an improvement for job seekers and will increase trust with the company and therefore users, and revenue.