Identifying Fraudulent Job Postings with Machine Learning

Alexandra Stepanenko

MSc Data Science

1 Introduction

This data set comprises of data relating to job advertisements posted online. The task is to predict whether these postings are genuine or fraudulent (i.e. this is a supervised binary classification task).

There are a total of 17880 postings, but only 866 of these are fraudulent, making this data set severely class imbalanced. A significant class imbalance presents a difficultly in many machine learning models as it is easier for the model to learn to predict the dominant class with high accuracy, leading to models which predict the dominant class well yet the minority class relatively poorly.

The sixteen feature variables comprise of a mix of text features such as job title and description, binary variables such as whether there was a company logo, categorical variables such as industry, and structured string data such as salary range. Using text as a feature in machine learning models often leads to high dimensionality. In addition, some of the non-text variables in this data set (such as industry or location) have a large number of unique values. High dimensionality (otherwise known as the curse of dimensionality) often leads to data sets being very sparse, which reduces a model's capability to learn generalisations within the data, and can lead to overfitting.

Within this work I will try to address both of these issues to create classifiers which are effective at classifying both classes.

2 Methodology

The following steps were taken:

1. Split into test and train data sets.
2. The data was explored and visualised to identify patterns within the data and potentially predictive variables. Data types, number of unique values and percentage of missing data for each variable were identified.

3. Preprocessing: each variable was processed according to its data type.
4. Feature selection
5. Resampling for class balancing
6. Creating models and hyperparameter tuning
7. Testing the models on the test data

2.1 Test-train split

After an initial look at the data, it was identified that the target variable was imbalanced. A stratified split of the target variable classes was used. The test and train sets have nearly identical proportion of fraudulent and non-fraudulent jobs postings.

2.2 Data exploration and visualisation

The data set comes with 18 variables - one unique identifier,'job_id', which we will not use for prediction, 16 columns which could be used as predictive features, and one target, 'fraudulent', which is a binary variable. The 16 potentially predictive features have a number of different data types:

2.2.1 Free-text

There are 6 free-text variables: title, department, company_profile, description, requirements, benefits. These fields need natural language processing (NLP) techniques to extract features such that they can be used in models. These variables have between 0.008% (description) and 64.38% (department) missing data. Whether or not data is missing can be treated as a feature as data being missing in certain variables may itself be predictive in whether a job posting is fraudulent or not.

2.2.2 Structured

Two variables are a string data type but are a structured format:

- location - appears that multiple locations can be selected from a list, separated by a comma. Usually used to create a granulated location, e.g. 'CA, ON, Toronto'. 1.85% of data is missing. The list of most common locations for genuine job postings was somewhat varied from the list of most common locations for fraudulent postings.
- salary_range - format is 1234-1234 where 1234 is any value, or NaN for missing data. 83.98% of job postings are missing a salary range. Salary range is more commonly reported on in fraudulent job postings, with 15.6% of genuine job postings having a salary range compared with 24.8% of fraudulent job postings. Whether salary_range is missing was identified as a potentially useful feature.

2.2.3 Binary

There are three binary variables in addition to the target: telecommuting, has_company_logo and has_questions. Binary variables do not require further processing. All binary values have a value of 0 or 1, there is no missing data.

The distribution for genuine versus fraudulent job postings was very similar for telecommunting (suggesting this will not be a predictive variable), very different for has_company_logo (unsurprisingly, most genuine companies had a logo, and most fraudulent companies did not - this is likely to be predictive) and somewhat different for has_questions.

2.2.4 Categorical

There are five categorical variables, all of which have between 19% and 45% missing data:

- employment_type - 5 unique values, 19.57% missing data
- required_experience - 7 unique values, 39.68% missing data
- required_education - 13 unique values, 45.89% missing data
- industry - 131 unique values, 27.93% missing data
- function - 37 unique values, 36.47% missing data

These variables will all be treated as nominal. Although required_experience and required_education have some ordering, missing data and values such as 'Not Applicable' means we cannot treat them as ordinal variables. The categorical values will be one-hot encoded, with missing data being its own category.

Data exploration showed that generally across categorical variables, some categories showed a different distribution across fraudulent and genuine postings, while others were relatively consistent.

2.3 Preprocessing

2.3.1 Binary and categorical variables, salary range

Binary variables did not need any further processing.

Categorical variables, which we identified as nominal, were one-hot encoded.

Due to the very high proportion of missing data in salary_data, it did not seem prudent to extract and attempt to use the salary data itself. However, as noted during the data exploration stage, a salary range being present was more common in fraudulent postings, and may be predictive. A binary variable was created to indicate whether or not salary range was given.

2.3.2 Location

It was noted in the data exploration stage that some locations may be more common for fraudulent job postings.

Individual locations were extracted for each posting and one-hot encoded if they appeared ten or more times. This was to avoid increased dimensionality and overfitting, as almost half of locations only appeared once in the whole data set.

2.3.3 Text

Data was cleaned (see Jupyter notebook for details), stop words were removed to reduce dimensionality (these are common words such as 'and' or 'the' which generally do not aid prediction), and words were lemmatised (convert to base form - this helps for both dimensionality reduction and retention of context).

For basic text processing, NLTK and spaCy are both effective. I used spaCy for stop word removal and lemmatisation: it has faster performance which is important as we have a large amount of data, and spaCy has a much longer list of stop words which we can remove for dimensionality reduction.

Vectorisation of text There are a number of ways to represent text in vector form. The simplest way is through the Bag of Words model, which is similar to one-hot encoding in that each unique token is a separate feature. When using a count vectoriser, the value for each term for each sample is how many times a term appears in that sample (term frequency). There is a more advanced version of this Bag of Words model, which encodes *Term Frequency Inverse Document Frequency* (TF-IDF) rather than term frequency. This accounts for not only the frequency of a word in a sample, but how many samples that word appears in. For example, if the word 'legal' appears four times in a sample but appears in very few other samples, this is more significant than the word 'and', which occurs many times in many samples - TF-IDF encodes this significance.

The draw back of Bag of Words models are:

- This form of vectorisation results in sparse, high dimensional vectors
- No context is retained between each word

An alternative to the Bag of Words model are word embeddings, which are dense vector representations of terms. Not only are word embeddings dense and can be low in dimension, they also retain context. For example, if two words, 'almost' and 'nearly' are mostly interchangeable in a sentence, they will be close together in an embedding space.

I chose to use word embeddings with my neural network models, and TF-IDF for other models.

Word embeddings Word embeddings in neural networks can be trained as weights during training, or pre-trained embeddings can be set as weights.

I chose to set pre-trained word embeddings (one for each of the five text features), which I created by training a Word2Vec model on the corpus of each feature. Chollet in Deep Learning with Python [1] notes that models with pre-trained word embeddings usually perform better as they are not subject to random initialisation. It is further noted that word embedding models trained on a specific corpus are preferable as this allows for the specific contexts of the corpus to be embedded.

Before creating the Word2Vec models, I detected bigrams within the corpus using the genism library.

2.4 Feature selection

There were two stages of feature selection to reduce the dimensions of the data:

- Assessing correlation between variables for removing highly correlated variables (this was used for locations which are correlated, e.g. 'Washington' and 'DC')
- As all of the variables which were assessed at feature selection stage were binary or categorical, I opted to use a Chi-square test to assess which variables are most associated with the target. Features with a Chi-squared p-value of more than 0.05 (i.e. a 95% confidence that the feature is not significantly associated with the target) were excluded from the data set.

2.5 Resampling for class balancing

One method of addressing the severe class imbalance is to resample the data [2][3].

Undersampling is the practice of reducing the number of samples belonging to the dominant class to match the number of samples in the minority class.

Oversampling is the practice of sampling the minority class with replacement to match the number of samples in the dominant class.

There are pros and cons to both methods - the downside of undersampling is a lot of information is lost. However, with oversampling there can be many duplicates, which in some cases may lead to optimistic training results when using cross-validation [2].

Advanced sampling techniques such as SMOTE (Synthetic Minority Oversampling Technique) exist, which synthesise realistic new minority class samples (by editing vectors, for example through use of K-Nearest Neighbour), however, much of our data is text so we will not use this technique.

I used a simple random over-sampler and random under-sampler from the Imbalanced Learn library to create balanced data sets. I tested my models with undersampled, oversampled and unbalanced (original) data to compare the results of these resampling methods.

2.6 Models

Neural networks have become an increasingly popular model for processing text due to advances in deep learning and increase in GPU speed and availability [4]. In addition, the ability to incorporate word embeddings into neural networks to greatly improve the way text is represented in vectors is also advantageous. I chose to use neural networks as a model to develop due to the advantages of word embeddings over sparse vectors.

To compare another model with neural networks, I chose Random Forests as it is a model that performs well on high-dimensional data because each tree in the Random Forest only considers a subset of the features.

2.6.1 Random Forest

F1 was chosen as the scoring metric to maximise both recall and precision.

Hyperparameters were tuned using RandomizedSearchCV. A randomised search was performed for the unbalanced set of data, the undersampled set and the oversampled set, to compare performance.

I chose to use random search methods to tune both the Random Forest and neural network hyperparameters as research by Bergstra and Bengio (2012) [5] suggests random search is more effective for tuning hyperparameters than grid search, as the nature of grid search means the optimum value for an important hyperparameter may lie between the limited number of values selected to search over.

2.6.2 Neural Networks

The neural networks were constructed using the keras functional API with 6 different inputs: the five different text features with their own embedding layers, and the non-text data.

Although the use of LSTM (Long Short-Term Memory) and Convolutional layers is common with text data, I chose to use Dense layers. This is because, as Chollet notes in Deep Learning with Python [1], in cases where the order of words is not crucially important, LSTM and ConvNets usually do not perform substantially if at all better than neural networks using dense layers, yet are significantly more resource and memory intensive.

I created a function which built and ran a model based on certain hyperparameters and architectural decisions (for example whether to include dropout layers). This was then used in a loop which randomly selected hyperparameters (similar to RandomizedSearchCV), ran the model and saved the results. A range of different metrics were measured, including binary accuracy, AUC (area under the ROC curve), precision and recall. Early stopping was implemented based on validation loss.

As with the Random Forest model, models were fit on all three of the unbalanced, undersampled and oversampled data to compare performance.

3 Results

3.1 Random Forest

3.1.1 Training

RandomizedSearchCV was used on each of the unbalanced, undersampled and oversampled sets, and the best parameters and F1 score were extracted. The balanced data sets performed significantly better than the unbalanced data set.

Best F1 score for:

- Unbalanced data: 0.692410
- Undersampled data: 0.928276
- Oversampled data: 0.928103

We see the model trained on undersampled data gave the best F1 score. Although it is only marginally better than the best F1 trained on oversampled data, we must take into consideration that cross-validation as used, and cross-validation scores with oversampled data can be overly optimistic, as some of the data which is seen in training may also be seen in the validation set due to duplication.

The best performing model had the hyperparameters: 'n_estimators': 350, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_depth': None, 'bootstrap': False

3.1.2 Test

On unseen test data, the best performing model produced an F1 score of 0.5738095238095238, which is significantly lower than in the during validation. This suggests the model overfit during training.

3.2 Neural Networks

3.2.1 Training

60 neural networks were built and run. Each of these models (whose hyperparameters were randomly selected) was trained on either the oversampled, undersampled or unbalanced data set.

The models which achieved the highest validation accuracy and precision were those trained on unbalanced data, however these models generally had very poor validation recall scores.

The models which achieved the highest validation recall scores were trained on a mix of oversampled and undersampled data, although these models generally had lower accuracy than other models.

The models which achieved the highest validation AUC scores were those trained on the oversampled data set. The maximum validation AUC achieved by a model was 0.991163. I chose the model which achieved this score as the

most successful model out of the 60, as it had a combination of a high AUC, accuracy, precision and recall score. The generalisation error of this model was calculated using unseen test data.

The variation between how models performed on different metrics based on the data set they were trained on can be seen in Figure 1 (overleaf).

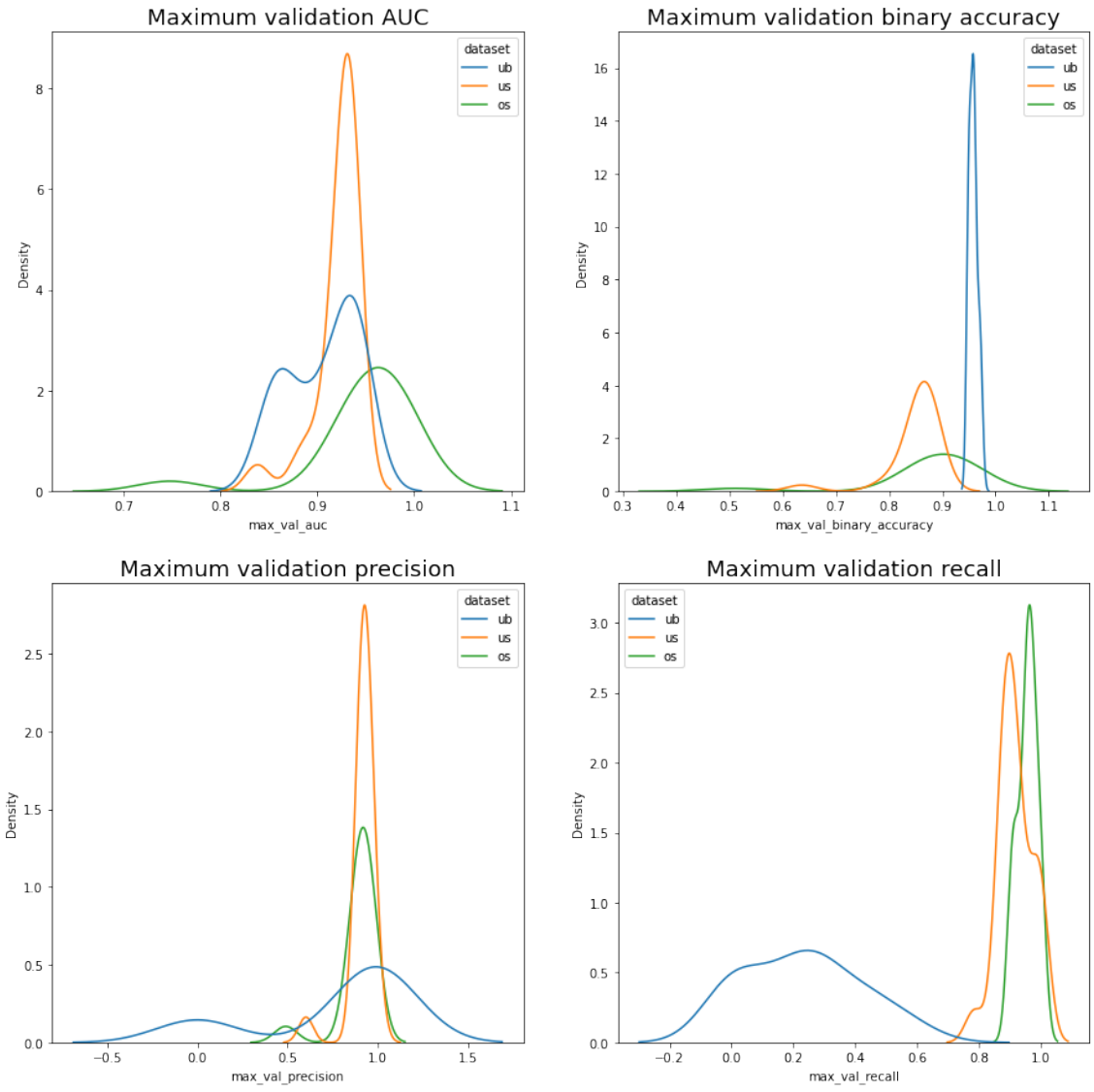

Figure 1: Metrics by models trained on unbalanced, undersampled and oversampled data sets

3.2.2 Test

Best validation metrics for the chosen model were:

- max_val_binary_accuracy 0.956129
- max_val_auc 0.991163
- max_val_precision 0.956747
- max_val_recall 0.995708

The test metrics are:

- binary_accuracy: 0.982289
- auc: 0.845904
- precision: 0.982456
- recall: 0.646153

We note that while recall is significantly lower on unseen data, and AUC is somewhat lower, both precision and accuracy are higher on unseen data. We note that we trained the model on oversampled data, so this may be a situation where minority class duplicates within the data set have lead to the model appearing to predict the minority class better (on validation data) than it actually does on unseen data.

4 References

[1] F. Chollet, Deep learning with Python. Shelter Island, NY: Manning Publications Co., 2018.

[2] M. Kuhn and K. Johnson, Applied predictive modeling. New York: Springer, 2013.

[3] J. Han, M. Kamber, and J. Pei, Data mining concepts and techniques. Amsterdam; London: Morgan Kaufmann, 2012.

[4] J. Eisenstein, Introduction to natural language processing. Cambridge, Massachusetts: The MIT Press, 2019.

[5] Bergstra, J., Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. J. Mach. Learn. Res., 13, 281-305.