
Sri Lanka Institute of Information Technology

**Machine Learning**

Semester 01 – Year 04 – 2020

# Fake Job Posting Prediction Using Random Forest Classifier Algorithm

**Assignment 01**

**Submitted by:**

M. D. P. Peiris
IT17123150
(Weekend SE batch)

April 20, 2020

# Table of Content

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Problem Statement

As an undergraduate senior, I spend considerable time seeking new job opening posts, mostly in LinkedIn and in other sites. Even though they all look genuine, there are millions of fake job opening posts out there.

It is a tremendous embarrassment that well-reputed companies also tend to post fake job openings. Reasons for fake job posting are as below [1]:

- Employers can measure the current talent pool, which means this is a better way to evaluate how in-demand a position is than to post an advertisement to fill that same position.
- Those companies can get a back-up for your position and keep resumes on file which is called pipelining.
- Those people can add you to a list and later send spam mail kind of things to you.
- From these fake job opening posts, the criminal (especially the cybercriminals) steal your personal and bank information. Then they swipe your identity or engage in deviant behavior that involves your good name and result in considerable financial bankrupts [1][2].
- By applying for these fake job openings, you let people copy your resume. Resume plagiarism is accepted. When you search for 'resume templates', you can see thousands of resumes floating around on the web. Your resume can be one of them.

By considering the above-mentioned facts, it is necessary to be beware of fake job posts. Plenty of features in a job opening post can be used to recognize whether that particular post is a real one or a fake one. The chosen dataset consists of those features. So the problem is to construct a machine learning classification model by delivering the ability to predict the given post is fake or real.

# 2 Methodology

## 2.1 Data Collection

### 2.1.1 Dataset

The relevant dataset is selected from the Kaggle website. Data is provided by the Laboratory of Information & Communication Systems Security of the University of the Aegean [3]. Data is stored in comma-separated value file(.csv). This dataset contains 18K job descriptions out of which about 800 are fake. The data consists of both textual information and meta-information about the jobs. The dataset can be used to create classification models that can learn the job descriptions which are fraudulent [3].

| | |
|---|---|
| Source of the dataset | [3] |
| Number of instances | 17880 |
| Number of attributes | 18 |
| Number of classes | 2 |

*Table 2.1.1.1: Basic details of the dataset*

### 2.1.2 Description of the Dataset

All the attributes in the dataset are described in detail as follow:

| Attribute | Description | Data Type |
|---|---|---|
| Job_id | Unique job ID | Integer |
| Title | The title of the job ad entry | String |
| Location | The geographical location of the job ad | String |
| Department | Corporate department(e.g. sales) | String |
| Salart_range | Indicative salary range(e.g. $50,000-$60,000) | String |
| Company_profile | A brief company description | String |
| Description | The details description of the job ad | String |
| Requirements | Enlisted requirements for the job opening | String |
| Benefits | Enlisted offered benefits by the employer | String |
| Telecommuting | True for a telecommuting position | Integer |
| Has_company_logo | True if company logo is present | Integer |
| Has_questions | True if screening questions are present | Integer |
| Employement_type | Full-time, Part-time, Contract, etc | String |
| Required_experience | Executive, Entry level, Intern, etc | String |
| Required_education | Doctorate, Master's Degree, Bachelor, etc | String |
| Industry | Automation, IT, Health care, Real estate, etc | String |
| Function | Consulting, Engineering, Research, Sales, etc | String |
| Fraudulent | Target – Classification attribute | Integer |

*Table 2.1.2.1: Description of attributes*

According to the mentioned information, the classification attribute of this dataset is fraudulent which has values as 0 and 1. If it has 0, that means that a particular job opening post is a genuine one and if it is 1, that means that a particular job opening post is a fraudulent one.

## 2.2 Data Preprocessing

Before applying any machine learning algorithm, the data in the selected dataset is needed to be preprocessed.

| | job_id | title | location | department | salary_range | company_profile | description | requirements | benefits | telecommuting | has_ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | Marketing Intern | US, NY, New York | Marketing | NaN | We're Food52, and we've created a groundbreaki... | Food52, a fast-growing, James Beard Award-winn... | Experience with content management systems a m... | NaN | 0 | |
| 1 | 2 | Customer Service - Cloud Video Production | NZ, , Auckland | Success | NaN | 90 Seconds, the worlds Cloud Video Production ... | Organised - Focused - Vibrant - AwesomeIDo you... | What we expect from you:Your key responsibilit... | What you will get from usThrough being part of... | 0 | |
| 2 | 3 | Commissioning Machinery Assistant (CMA) | US, IA, Wever | NaN | NaN | Valor Services provides Workforce Solutions th... | Our client, located in Houston, is actively se... | Implement pre-commissioning and commissioning ... | NaN | 0 | |
| 3 | 4 | Account Executive - Washington DC | US, DC, Washington | Sales | NaN | Our passion for improving quality of life thro... | THE COMPANY: ESRI – Environmental Systems Rese... | EDUCATION: Bachelor's or Master's in GIS, busi... | Our culture is anything but corporate—we have ... | 0 | |
| 4 | 5 | Bill Review Manager | US, FL, Fort Worth | NaN | NaN | SpotSource Solutions LLC is a Global Human Cap... | JOB TITLE: Itemization Review ManagerLOCATION:... | QUALIFICATIONS:RN license in the State of Texa... | Full Benefits Offered | 0 | |

*Figure 2.2.1: Data in the dataset before preprocessed*

Data preprocessing is done using a few steps and some realistic assumptions. As the first step, assumptions have made to drop some of the attributes because they look little difficult to handle when selecting a feature to apply a machine learning algorithm [8]. They are mentioned as below:

| **Attribute** | **Reason to drop** |
|---|---|
| Job_id | It is unique for each record.<br>It is hard to get useful information using this. |
| Title | It is irrelevant because when it is notorious that a lot of job titles are superfluous and fluff. |
| Location | The model which is going to be created does not need to care about the location and keep performance generalized for a job anywhere in the world. |
| Department | This is the same as the *title* attribute.<br>This can be varied too much by posting and not be meaningful enough to draw anything from. |
| Industry | Same reason as *title* and *department.* |
| Function | Same reason as *title* and *department.* |

*Table 2.2.1: Reasons for dropping attributes*

Then the next step is to create a new string type attribute(combined_text) combining all the string type attributes except the ones that have chosen to drop in the first step. Both selected attributes to drop and the attributes that used to make the new string have been dropped after that [8][10].

```
#Data Preprocessing

#Combine separate string type columns into one column by removing all the non-relevant features for the prediction like job_id,
title, location, department, salary_range, industry and function
jobs.fillna(" ",inplace = True)
jobs['combined_text']=jobs['company_profile']+" "+jobs['description']+" "+jobs['requirements']+" "+jobs['benefits']+" "+jobs['em
ployment_type']+" "+jobs['required_experience']+" "+jobs['required_education']
jobs.head()
```

*Figure 2.2.2: Combine String*

In the next step, the new attribute has gone through a data cleaning process. In this process, first, it is needed to create a list. Then each expression has replaced with a space except a-z and A-Z. Then it has converted from UPPERCASE to lowercase. Then each word in the string has separated and returned the base(lemma) form of the word. Then every word in the string has broken down using a loop. Then the separated words have combined by putting a space between them again. At the end of this data cleaning process, all the strings have gathered into the created list [10].

```
#Cleaning data
string_list = []
for string in jobs_df.combined_text:
    string = re.sub("[^a-zA-Z]"," ",string)             #replace all expressions except a-z and A-Z with a space
    string = string.lower()                             #convert string from UPPERCASE to lowercase
    string = nltk.word_tokenize(string)                 #seperate each word in the string
    lemma = nlp.WordNetLemmatizer()                     #returns the base(lemma) form of the word
    string = [lemma.lemmatize(word) for word in string] #breakdown every word in the string using a loop
    string = " ".join(string)                           #combine the separated words by putting a space between them again
    string_list.append(string)                          #gather all the strings into the created list
```

*Figure 2.2.3: Cleaning Data*

The final step of the data preprocessing stage is to make the feature ready to apply a machine learning algorithm. In this case, most used 150 words have selected by avoiding non-English words. Then the method has fitted on the string and made the list result [10].

```
# Bag of Words:
max_features = 150                                                          #most used 150 words in the string
count_vectorizer = CountVectorizer(max_features=max_features, stop_words = "english") #delete non-English words using stop_words

sparce_matrix = count_vectorizer.fit_transform(string_list).toarray()       #fit the method on string and make a list
result

print("The most used {} words :{}".format(max_features,count_vectorizer.get_feature_names()))
X = sparce_matrix
Y = jobs_df.iloc[:,3].values
```

*Figure 2.2.4: Bag of Words*

| | telecommuting | has_company_logo | has_questions | fraudulent | combined_text |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | We're Food52, and we've created a groundbreaki... |
| 1 | 0 | 1 | 0 | 0 | 90 Seconds, the worlds Cloud Video Production ... |
| 2 | 0 | 1 | 0 | 0 | Valor Services provides Workforce Solutions th... |
| 3 | 0 | 1 | 0 | 0 | Our passion for improving quality of life thro... |
| 4 | 0 | 1 | 1 | 0 | SpotSource Solutions LLC is a Global Human Cap... |

*Figure 2.2.5: Data in the dataset after preprocessed*

## 2.3 Random Forest Classifier

### 2.3.1 Justification for selecting the algorithm

Random Forest is a great machine learning algorithm for producing a prediction model for both classification and regression problems. Its default hyperparameters already return great results and the system is great at avoiding overfitting [4].

Random Forest can handle many types of features such as binary, categorical and numerical. It requires less amount of preprocessing and the data does not have to be rescaled or transformed. This makes Random Forest is impressive in versatility [5].

It can give results with a faster computation time by splitting the process to multiple machines to run and it is known as the parallelizability of the Random Forest algorithm. Since it works with subsets of data, this algorithm is great with high dimensional data and faster to train than decision trees. Prediction speed is faster than the training speed in this algorithm [5].

The ability to handle unbalanced data, missing values, and non-linear features make this algorithm special. And also it has low bias and a moderate variance model [5].

### 2.3.2 Basic concepts of Decision Tree

Decision Tree is a predictive model used in machine learning, statistics, and data mining. It has a tree structure starting with some observation about an item(data) to conclusions (target values). It represents possible paths(branches) to make a prediction based on a series of decisions along the branches of the tree. Decision Tree can be used for both classification and regression. Univariate, multivariate, binary and n-ary are the different types of Decision Trees [6].

Induction and Pruning are the two steps of creating Decision Tree models. Construction of the Decision Tree from class labeled training samples is the basic function of the Induction step. This uses the common top-down approach. It starts with a training set of samples associated with class labels and recursively split the training set into smaller subsets as the tree is being built. Splitting is done based on a test (splitting rule) on an attribute or variable of the dataset. Pruning is the technique used to reduce the unnecessary complexity of the constructed Decision Tree by removing last-reliable branches. This addresses the problem of overfitting the data and improves accuracy [6].

### 2.3.3 Implementation of Classifier

As the first step of the implementation of random forest classifier, the dataset has been split by 9:1 for training and testing using random selection. The training set is to train the model and perform optimization. The testing set is to assess the model performance of the model. 'random_state' increases the processing speed by making the model's output replicable.

```
#Seperating dataset for training and testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, random_state = 42)

print('Training Features Shape : ',X_train.shape)
print('Training Labels Shape   : ',Y_train.shape)
print('Testing Features Shape  : ',X_test.shape)
print('Testing Labels Shape    : ',Y_test.shape)
```

```
Training Features Shape :  (16092, 150)
Training Labels Shape   :  (16092,)
Testing Features Shape  :  (1788, 150)
Testing Labels Shape    :  (1788,)
```

*Figure 2.3.3.1: Split the dataset into training and testing sets*

Then Standard Scaling has been applied to get the optimized results. Then using 'fit_transform()', the values of the training set and testing set have been calculated and applied on actual data and then finally given the normalized value.

```
#Applying Standard scaling to get optimized result
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

*Figure 2.3.3.2: Applying Standard Scaling*

Random Forest Classifier has been built by using 1000 decision trees and the model fitted by the training data. Then the model has been tested using 10% of the job posting dataset and then the given job opening post will be predicted as a genuine one or a fraudulent one by the model.

```
#Applying Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=1000)
rfc.fit(X_train, Y_train)                    #build the classifier
pred_rfc = rfc.predict(X_test)               #predict using test data
```

*Figure 2.3.3.3: Building Random Forest Classifier*

## 2.4 Testing

Once the classification model is built and trained, testing that the trained model is the only thing remaining to complete. Four testing techniques have been used for the testing purpose of this model.

1. Accuracy Score

This is the most basic testing method in learning models. It is a ratio between the total positive predictions vs. the total number of predictions. Following is the formula for calculating the accuracy score [7].

*Accuracy = Total Positive Prediction / Total Number of Prediction*

Accuracy Score of this trained model = 0.97

```
#Get the accuracy score
print('Accuracy Score : ', str(accuracy_score(Y_test, pred_rfc)))
```

```
Accuracy Score :  0.977069351230425
```

*Figure 2.4.1: Accuracy Score*

2. Classification Report

This testing method provides important classification metrics for each category as below:

| Metric | Description |
|---|---|
| Precision | This identifies the prediction frequency of a positive class by the model.<br>*Precision = True Positives / (True Positives + False Positives)* |
| Recall | This gives the percentage of correctly identified actual True Positive class.<br>*Recall = True Positives / (True Positives + False Negatives)* |
| F1 – score | This gives a score between 0 and 1 where 1 means the model is perfect and 0 mean useless.<br>*F1-score = 2 * ( (Precision * Recall) / (Precision + Recall) )* |
| Support | This provides the number of actual class occurrences in the specific dataset. |

*Table 2.4.1: Description of metrics in the classification report*

Following is the classification report of this model.

```
#Check model performance
print(classification_report(Y_test, pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1693
           1       1.00      0.57      0.72        95

    accuracy                           0.98      1788
   macro avg       0.99      0.78      0.86      1788
weighted avg       0.98      0.98      0.97      1788
```

*Figure 2.4.2: Classification Report*

3. Confusion Metrix

The confusion matrix is a square matrix table of N*N where N is the number of classes that the model needs to classify [7]. Following is the confusion matrix of this model:

```
#Confusion matrix for the random forest classification
print(confusion_matrix(Y_test, pred_rfc))
```

```
[[1693    0]
 [  41   54]]
```

*Figure 2.4.3: Confusion Metrix*

The model has predicted genuine job posts (0) as 1693, where it actually is 1693 and fraudulent job posts (1) as 54, where it actually is 95.

4. Cross-Validation Score

This is a technique where the datasets are split into multiple subsets and learning models are trained and evaluated on these subset data [7].

Cross-validation score of this model = 0.97

```
#Cross Validation Score for random forest

#Evaluation for random forest model using cross validation.
rfc_eval = cross_val_score(estimator = rfc, X = X_train, y = Y_train, cv = 10)
rfc_eval.mean()
```

```
0.9751430038332517
```

*Figure 2.4.4: Cross-Validation Score*

# 3 Evaluation

## 3.1 Result of the project

Since the accuracy score is 0.97, Random Forest Classification is more suitable for the prediction regarding the real or fake job posting dataset. Accuracy score can be increased by using more missing value replacements and more data balancing methods.

## 3.2 Lessons learned

- Learned about machine learning concepts, algorithms, the various application that uses them and the benefits of using machine learning algorithms.
- Learned about the libraries that use when building prediction models. e.g. pandas, seaborn, matplotlib, sklearn, nltk, numpy, etc.
- Learned few data preprocessing methods.
- Learned how to build a support vector machine, logistic regression, random forest regression, and random forest classification.
- Learned about Decision Trees and its real-world applications and pros and cons.
- Learned how much it is important to use testing techniques like accuracy score, classification report, confusion matrix, and cross-validation score when testing the build and trained model.
- Learned about how much troubles that can be caused by fake job opening posts.

## 3.3 Future work

- Apply more complex algorithms to build a prediction model for this dataset.
- Proceed more in Exploratory Data Analysis (EDA) and get more knowledge about data and find relationships between the real posts and fake posts like what is the word count in a post. Then use them to preprocess the data.
- Use various libraries, check the differences and select the best ones.

# 4 References


[1] Rosen, A., 2017. "*The Dirty Truth: Why Employers Post Fake Jobs" - Jobacle.Com*. [online] Jobacle.com. Available at: https://www.jobacle.com/blog/the-dirty-truth-why-employers-post-fake-jobs.html. [Accessed 19 April 2020].

[2] Crane, C., 2020. "*Fake Jobs: Cybercriminals Prey On Job Seekers Via Fake Job Postings" - Hashed Out by The SSL Store™*. [online] Hashed Out by The SSL Store™. Available at: https://www.thesslstore.com/blog/fake-jobs-cybercriminals-prey-on-job-seekers-via-fake-job-postings/. [Accessed 19 April 2020].

[3] Bansal, S., 2020. "*[Real Or Fake] Fake Jobposting Prediction"*. [online] Kaggle.com. Available at: https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction. [Accessed 19 April 2020].

[4] Albert, S., 2018. "*Why Random Forest Is The Greatest!*". [online] Medium. Available at: https://medium.com/diogo-menezes-borges/random-forests-8ae226855565. [Accessed 19 April 2020].

[5] Kho, J., 2018. *Why Random Forest Is My Favorite Machine Learning Model*. [online] Medium. Available at: https://towardsdatascience.com/why-random-forest-is-my-favorite-machine-learning-model-b97651fa3706. [Accessed 19 April 2020].

[6] Courseweb lecture materials.

[7] Billa, M., 2019. "*Testers Guide For Testing Machine Learning Models"*. [online] Medium. Available at: https://medium.com/analytics-vidhya/testers-guide-for-testing-machine-learning-models-e7e5cea81264. [Accessed 19 April 2020].

[8] Albert, N., 2020." *Is This Job For Real?"*. [online] Kaggle.com. Available at: https://www.kaggle.com/nikitaalbert/is-this-job-for-real#Introduction. [Accessed 19 April 2020].

[9] Singh, V., 2020. "*Fake_Job_Prediction"*. [online] Kaggle.com. Available at: https://www.kaggle.com/vashistnarayansingh/fake-job-prediction. [Accessed 19 April 2020].

[10] Cevik, M., 2020. "*Job Postings Is It True(Accuracy Logreg, Nb, Knn)"*. [online] Kaggle.com. Available at: https://www.kaggle.com/mahmutevik/job-postings-is-it-true-accuracy-logreg-nb-knn. [Accessed 19 April 2020].

# 5 Appendix – Source Code

```
#Importing required packages.
import pandas as pd
import seaborn as sns
import gc
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
import re
import nltk
from nltk.corpus import stopwords
import nltk as nlp
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
import nltk as nlp
%matplotlib inline

#Load dataset
jobs = pd.read_csv('F:\\Datasets\\ML\\fake job prediction\\fake_job_postings.csv')

#Check how data is initially distributed
jobs.head()

#Check the shape of the data
jobs.shape

# 0 is legitimate(not fake) and 1 is fraudulent(fake)
#Check for Imbalance
jobs['fraudulent'].value_counts()

sns.countplot(jobs['fraudulent'])

#Information about the dataset
jobs.info()

#Statistical analysis of the dataset
jobs.describe()
```

```
#Data Preprocessing
#Combine separate string type columns into one column by removing all the non-relevant features for the prediction like job_id, title, location, department, salary_range, industry and function
jobs.fillna(" ",inplace = True)
jobs['combined_text']=jobs['company_profile']+" "+jobs['description']+" "+jobs['requirements']+" "+jobs['benefits']+" "+jobs['employment_type']+" "+jobs['required_experience']+" "+jobs['required_education']
jobs.head()

#Remove all the unwanted columns : non-relevant columns, columns that have combined in the previous step
drop_columns = ['job_id', 'title', 'location', 'department', 'salary_range', 'company_profile', 'description', 'requirements', 'benefits', 'employment_type', 'required_experience', 'required_education', 'industry', 'function']
jobs_df = jobs.drop(drop_columns, axis=1)

del jobs
gc.collect()

#Check the distribution of data before the data cleaning step
jobs_df.head()

#Cleaning data
string_list = []
for string in jobs_df.combined_text:
   string = re.sub("[^a-zA-Z]"," ",string)          #replace all expressions except a-z and A-Z with a space
   string = string.lower()                          #convert string from UPPERCASE to lowercase
   string = nltk.word_tokenize(string)              #seperate each word in the string
   lemma = nlp.WordNetLemmatizer()                  #returns the base(lemma) form of the word
   string = [lemma.lemmatize(word) for word in string]  #breakdown every word in the string using a loop
   string = " ".join(string)                        #combine the separated words by putting a space between them again
   string_list.append(string)                       #gather all the strings into the created list


# Bag of Words:
max_features = 150                                  #most used 150 words in the string
```

```
count_vectorizer = CountVectorizer(max_features=max_features, stop_words = "english")
#delete non-English words using stop_words

sparce_matrix = count_vectorizer.fit_transform(string_list).toarray()              #fit the method
on string and make a list result

print("The most used {} words
:{}".format(max_features,count_vectorizer.get_feature_names()))
X = sparce_matrix
Y = jobs_df.iloc[:,3].values


#Check the distribution of data after preprocessing the data
jobs_df.head()

#Seperating dataset for training and testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, random_state = 42)

print('Training Features Shape : ',X_train.shape)
print('Training Labels Shape   : ',Y_train.shape)
print('Testing Features Shape  : ',X_test.shape)
print('Testing Labels Shape    : ',Y_test.shape)

#Applying Standard scaling to get optimized result
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)

#Applying Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=1000)
rfc.fit(X_train, Y_train)              #build the classifier
pred_rfc = rfc.predict(X_test)         #predict using test data

#Get the accuracy score
print('Accuracy Score : ', str(accuracy_score(Y_test, pred_rfc)))
#Check model performance
print(classification_report(Y_test, pred_rfc))

#Confusion matrix for the random forest classification
print(confusion_matrix(Y_test, pred_rfc))
```

```
#Cross Validation Score for random forest

#Evaluation for random forest model using cross validation.
rfc_eval = cross_val_score(estimator = rfc, X = X_train, y = Y_train, cv = 10)
rfc_eval.mean()
```