Fraudulent Job Posts Detection Using Neural Networks

Shabnam Hajian

Department of Management Science, University of Waterloo

MSCI 623 – Big Data Analytics

Lukasz Golab

August 5,2020

ABSTRACT

Following the revolutionary impact of the internet on the human lifestyle, most of the recruitment process has been changed to the online from the traditional format. This change potentially creates a situation to misuse personal information since the important information can be extracted by fraudulent job position advertise and fake recruitment processes. This employment scam has disadvantages for both organizations and job seekers. In this project, a publicly available dataset has been used to study online recruitment. This project has two main parts. The first part is to use the k-mean algorithm to classify job titles. Then in the second part, two classification models were used to classify fraudulent and legitimate job posts. The results showed by using the document to vector format of all text data in a job post as an input, a weighted feed-forward neural network model improved model performance in comparison to the decision tree model.

INTRODUCTION

The hiring process started a new era by using online advantages as e-recruitment. Although online processes are faster and more cost-efficient, it can create potential security problems. E-recruitment considered as a part of e-HRM and defined as the implementation of recruitment activates by involving the internet and using web solutions to operate and in the end, it has the same goals of the traditional recruitment (Ghazzawi and Accoumeh, 2014). More number of job advertisements and job applicants encouraged both employers and job seekers to use online platforms and processes. Most of the large organizations and many small size companies already use e-recruitment; they posted jobs and accepted resumes online and respond to applicants by e-mails. This online process is continued to grow in the next years (Kapse et al., 2012). The future of hiring is e-recruitment and in spite of many advantages, it costs some new security problems.

The e-recruitment process provided a potential scam situation. Vidros, Kolias and Kambourakis (2016) introduced Online Recruitment Fraud (ORF) for the first time as a form of the employment scam. ORF occurs when job seekers revealing their personal information as a result of trusting on a fake job advertisement on an online platform (Mahbub and Pardede, 2018). A lot of personal information being communicated between employees and employers during the recruitment procedure. Usually, job seekers are comfortable with giving that required information to their future employers in the recruitment process while this information can be used for both good or bad purposes. The requested information can be at any level from the simple email addresses to a more sensitive level as passport data or credit card information. Although all this information can be an essential part of the real recruitment process, this self-confession can cost applicants big security problems if they trapped in the fraudulent job advertisement or fake recruitment process.

Detecting fraudulent job posts is important to prevent ORF. The fraudulent job posts can harm job seekers, by stealing their personal information. It can also damage organizations, by destroying their reputations. Hence, the fraudulent job posts detection is beneficial for both employee and employer. However, it is not easy to detect fraudulent job posts even for humans. This project's goal is to find a reliable classifier that can detect fake job advertisements. This classification model can be used as the early detection of scam employment. By preventing to post fraudulent job advertisements after early detection, the amount of ORF can be reduced.

Although the main part of this project is to detect fraudulent job posts, a clustering task has been done too. For this task, the job titles have been clustered into different groups based on their semantic similarities in an unsupervised task. Different organizations might use different names for a single job position and the clustering can give a general idea about the job position. This clustering can be useful for job seekers to speed up their searches between many job positions. Hence, online recruitment applications can use them as a part of the user interface. This clustering will be done by using text to vector format of the job titles, in order to keep their semantic relations, plus implementing a k-mean algorithm. The clustering decrease variation of job titles in the dataset.

Additionally, there is no association task for the unsupervised learning part of this project. This decision made as a result of the nature of data. There are no clear boundaries between the data of each column. More clearly stated, there are many pieces of information in unrelated columns. To creating association rules, it is needed to have correctly labelled data. This labelling data is possible to be done by more advanced text mining processes, but it was not in the scope of this project. Without deeper text mining, the association rules cannot be trusted. Thus, there is no association rule mining in this project.

In this research, the publicly available dataset named EMSCAD has been used which firstly introduced by Vidros et al. (2016). This data has two properties; firstly, the class data is highly imbalanced and secondly, each job advertisement contains many text information columns with not a restricted structure. As mentioned, the lack of a strict structure for the text data leads to having a mess in data and information in each column. As an example, in some job advertisements, the department of the job post has been mentioned in the job title column, instead of the department column. Thus, job posts data have been distributed between columns and not well labelled. Both of these properties are what is expected in real-life data too. Thus, a practical classification model has to be able to deal with these two properties. The goal of this project is to create a classifier on this dataset considering its specific attributes.

Ultimately there are two main hypotheses for the classification task of this project. These hypotheses are based on the nature of the problem that the classification model addressed to. As mentioned, while each job advertisement contains a lot of data in the format of the text, these data are not well structured and distributed between different columns. Hence, the first hypothesis in this project is that “*combining all text data about the job posts, can be a good input of the classification*”. The other point is that the contribution of fraudulent and legitimate job advertisements is not balanced**,** and the classification model should be able to handle this skewed class distribution. Most models cannot show good performance when they are trained with highly imbalanced training data. The neural network models are a relatively new and powerful approach that can be used as a classification model. The second hypothesis in this project is that “*using the advantages of the Neural Network, the model can handle the imbalance problem in data and improve the performance of detecting fraudulent job posts in comparison to former models like Decision Tree classifier*”. Therefore, the classification model in this project based on these two hypotheses.

RELATED WORKS

The Online Recruitment Fraud (ORF) is relatively a new area of research. However, many valuable works have been done in similar fields. According to Vidros et al. (2016) email spamming, phishing, Wikipedia vandalism, cyberbullying, trolling and opinion fraud are similar problems that have been addressed as improper user behaviours on the web. There are many articles in similar areas and many projects have been done with different approaches. As an instance, Roy and Viswanatham (2016) in spam email detection, Zhang, Hong and Cranor (2007) and Zhang and Yuan (2012) in phishing detection, McKeown and Wang (2010) in Wikipedia vandalism, Dinakar and Reichart (2011) in cyberbullying. Despite many works in those similar areas, there are not many works that addressed ORF.

The ORF first introduced by Vidros et al. in an article in 2016 followed by a complimentary article in 2017. In these two articles, mostly the employment scam and ORF problem were described. These two papers also introduced similar prior areas of research as the start point. Vidros et al. (2017) also introduced the EMSCAD dataset and used a balanced subset of this dataset to model six classifiers of “ZeroR”, “Logistic regression”, “OneR single rule”, “J48 decision trees”, “Naïve Bayes” and “Random forest for 100 trees”. The main problem of these first classifiers was using a balanced subset of data instead of the whole data. Nevertheless, these two articles were a start point to research about ORF and gave a good insight to start.

The next published work has been done by Mahbub and Pardede (2018) by using contextual features. They have studied dataset features and used binary feature vectors to model three classifiers. They performed experiment these three models on two independent subsets of the dataset as A and B. The subset of B contained contextual features of job posts as well as other textual and structural features. The main focus of this paper was about companies’ profile context and all three models have been created based on data on that column. All two experiments performed only applied on a subset of the dataset which has data in the company profile column. Thus, this second paper focused on the company profile data and its contextual features.

The next two works have been done in 2019. In the first paper, Alghamdi and Alharby (2019) used Support Vector Machine (SVM) for feature extraction. Then by using ensemble random forest classifiers for approximately all data, they achieved an accuracy of 97.4%. In the other paper, Lal, Jiaswal, Sardana, Verma, Kaur and Mourya (2019) used the ensemble approach “average vote”, “maximum vote” and “minimum vote” for three classifiers of “Logistic regression”, “J48” and Random forest. They used all data and claimed an accuracy of 95.5%. These two papers were the latest papers in this field.

In this project the focus in using all text information about job posts and applying the “Neural network” model in fraudulent job detection. Considering all text data of each job post is a new approach between prior works. Moreover, none of the prior works on this dataset used neural models. The neural classifier performance will be compared with the Decision Tree classifier to show the improvements. As mentioned, the EMSCAD is a new dataset and ORF is a new area to work. This project used also the new concept of neural models to detect and classify fraudulent job posts.

DATA

The dataset used in this project is named Employment Scam Aegean Dataset (EMSCAD) that first introduced in the paper of Vidros et al. in 2017. The dataset contains real-life job advertisements, posted by Workable between the years 2012 to 2014 in English(Vidros et al., 2017). Based on the Vidros et al. (2017) all data records have been manually labelled by specialized Workable employees as "fraudulent" and "not_fraudulent". The EMSCAD aims to provide a clear picture of the employment scam problem. The dataset is publicly available to be used for scientists who work on the field(University of the Aegean-Laboratory of Information & Communication Systems Security, 2020).

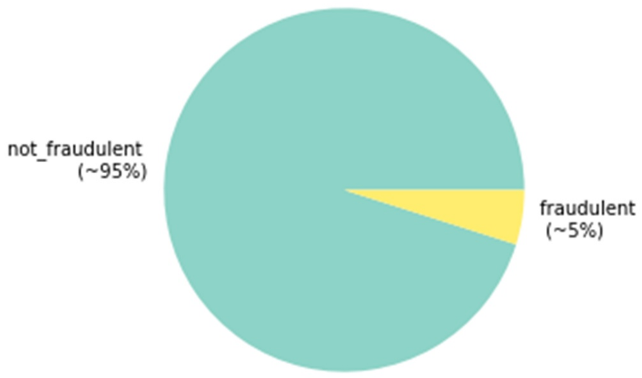
The EMSCAD dataset is in 18 columns and 17880 rows. Each row is one record of the job advertisement and contains job posts information in 18 columns. The below table shows all column names and a short description for data on that column.

Table 1- EMCAD columns description

Column name	Description
job_id	Index - Table key
title	The title of the job
location	The geographic location of the job
department	Internal department of the organization
salary_range	Indicative the salary range
company_profile	A brief description about the company
description	A description of the job
requirements	Requirements of the job
benefits	Offered benefits of the job
telecommuting	Binary - Whether position is a telecommunication position or not
has_company_logo	Binary - Whether the post contains company logo or not
has_questions	Binary - Whether the post has questions when applying or not
employment_type	Type of employment (full-time, part-time, contract, etc.)
required_experience	Required level of experience (Executive, entry level, intern, etc.)
required_education	Required level of education (Master’s, bachelor’s degree, etc.)
industry	Specific industry (IT, healthcare, etc.)
function	Specific area of functionality (Engineering, research, sales, etc.)
fraudulent	Class label (1 as fraudulent job and 0 as legitimate job advertisements)

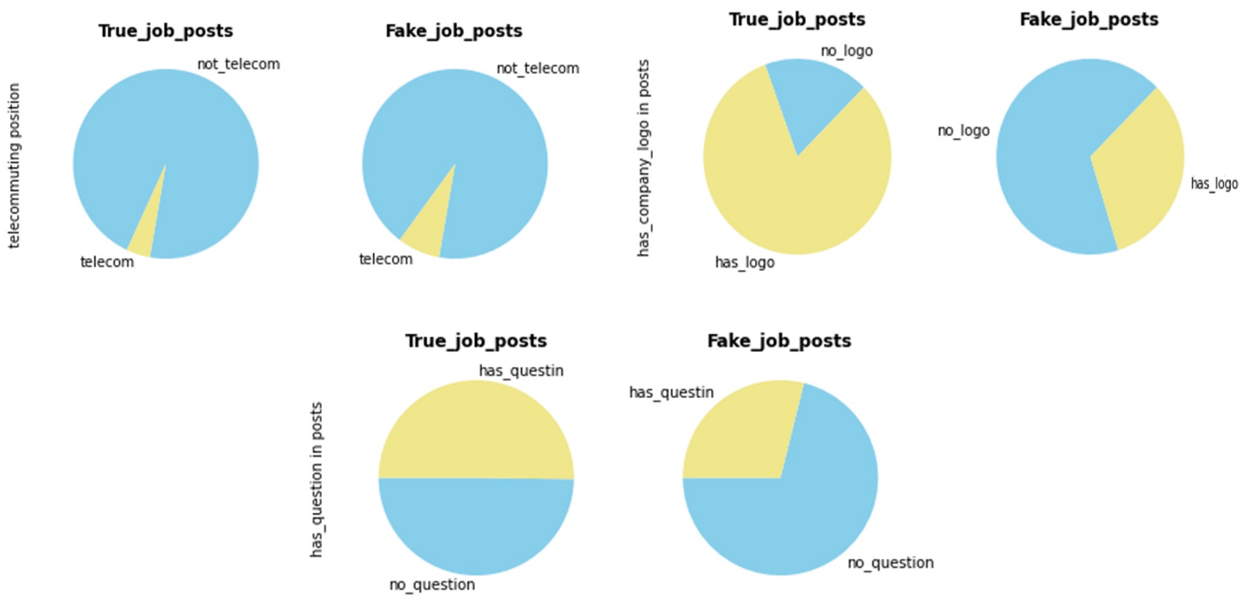
The EMSCAD dataset contains a total of 17,880 job posts which 17,014 of that is legitimate and 866 is fraudulent. The below figure shows the share of fraudulent and legitimate job posts among the dataset. From these numbers, only 4.84 % of total data is fraudulent which means that data is severely imbalanced. In this project, all 17,880 data will be used. Hence, the model should handle skewed class distribution.

Figure 1- Share of fraudulent and legitimate data in EMCAD



There are three binary variables as numeric variables in this dataset (except the class variable) which can be studied separately. The first point is the share of data in these three columns based on the value of the class variables. As figure 2 illustrates, while there are no huge differences in the share of telecommuting positions in fraudulent and legitimate job posts, the share of data for the company logo and having a question when applying are different between two classes. The share of having a company logo in the advertisement is bigger when the job post is legitimate, in comparison to the fake job posts. Moreover, the share of job posts that have a question when applying is 50% for true jobs, but this percentage is smaller if the job post is fake. The share of data in two classes is a point to study for binary variables in the dataset.

Figure 2- Share of three binary variables based on the value of the class variable

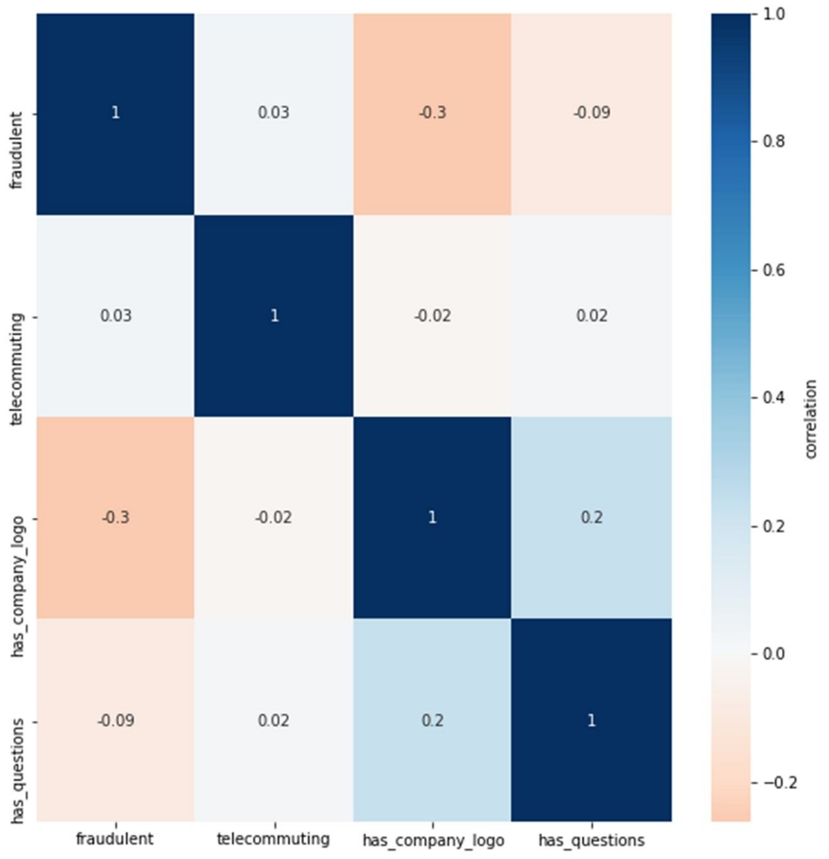


The other thing that can be checked about numeric variables is the correlation between these values. Figure 3 shows the correlation matrix of three binary columns of “telecommuting”, “has_company_logo”, “has_question” and the target column of “fraudulent”. In this picture, the darker cell means a higher correlation. Additionally, the blue colour shows a positive correlation and the orange colour shows a negative correlation. From figure 3 below points can be concluded:

1. There is no high correlation between these numeric columns (> |0.5|).
2. Having the "has_company_logo" and "has_question" show a negative correlation with fraudulent job posts, while the "telecommuting" positions shows a positive association.
3. " has_company_logo" shows the highest correlation with the class variable (-0.3).
4. "telecommuting" positions shows the lowest correlation with the class variable (0.03).
5. "has_company_logo" and "has_question" shows the highest correlation among all variables except the class variable (0.2).
6. "telecommuting" positions shows the exact same amount of correlation with "has_company_logo" and "has_question", whereas this correlation is negative with the "has_company_logo" and is positive with "has_question".

The important note about all above notes is these relations are only association and the correlation does not imply causation.

Figure 3- Correlation matrix of numeric variables and the class variable



The next step is to study missing information in the dataset. The dataset contains too many missing values. The below table shows the percentage of missing information based on the target label. The table percentages are based on the total amount of legitimate and fraudulent data in each column. As table 2 demonstrates, 68% of fraudulent job posts have no company profile data, while this number is 16%, when the job advertisement is legitimate. The other important insight is about “salary_range”; this column shows more than 84% missing information. Generally, considering all data (17,880 rows *18 columns) the amount of missing information is about 22%.

Table 2- Percentage of missing information based on the classes

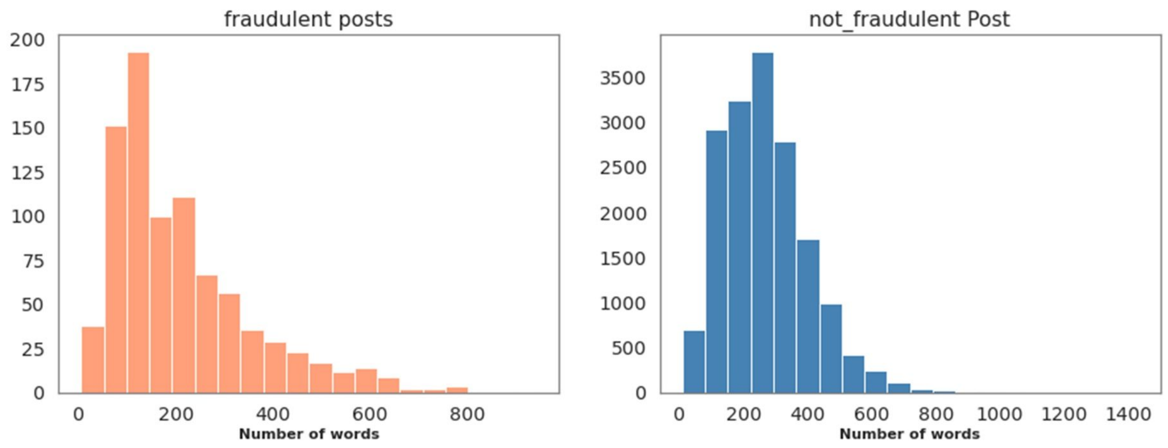
Class	Legitimate	Fraudulent	Total
job_id	0%	0%	0%
title	0%	0%	0%
location	2%	2%	2%
department	65%	61%	65%
salary_range	84%	74%	84%
company_profile	16%	68%	19%
description	0%	0%	0%
requirements	15%	18%	15%
benefits	40%	42%	40%
telecommuting	0%	0%	0%
has_company_logo	0%	0%	0%
has_questions	0%	0%	0%
employment_type	19%	28%	19%
required_experience	39%	50%	39%
required_education	45%	52%	45%
industry	27%	32%	27%
function	36%	39%	36%

This missing value should be considered based on the project goals. this project has two purposes. The unsupervised learning of clustering and the supervised learning of classification job posts into two classes of fraudulent and legitimate(not_fraudulent). For clustering job titles, the only needed column is the "title" column. which shows no missing value based on table 2. For the classification, as mentioned, all text data of the job advertisements are needed. Hence, all text information of the job posts will be combined and considered as a unit of information. By merging all 12 columns of “title”, “location”, “department”, “company_profile”, “description”, “requirements”, “benefits”, “employement_type”, “required_experience”, “required_education”, “industry” and “function” as the “information” column, there will be no missing data for job posts classification. Moreover, based on the share of missing value in the “salary_range” column, it will not be considered in this project. Therefore, by these changes, there are no missing data for both clustering and classification tasks.

Considering the “information” column as a unit of data, in addition to solving the missing value problems, is essential, based on the lack of determinative structure in entering text data. As mentioned, there is no restricted limitations about importing text data on in job advertisement. As an instance, there are many job titles that included other data like the location information. “*Account Executive - Washington DC*” is a job title example with this data entry problem. Another instance is where the content of the “company profile” column mentioned in the “description” column such as "*about the company, this is an amazing job opportunity with one of the most robust companies in the energy industry!*" in the "description" column. The personal opinion or some other patterns will affect the place of importing information. For example, McKeown and Wang (2010) pointed, fraudulent job posts tend to mention the salary in the job title column in order to catch more attention. Hence, when there is no limiting structure to import text information based on the nature of the data, combining all text information is a good solution to deal with this problem.

After combining text data in the "information" column, the length of data can be studied based on the class label. The below bar charts are the histograms of the "information" column data length in two groups of true and fraudulent job advertisements. As figure 4 shows, the fraudulent job posts have shorter "information". The maximum length of the "information" column for true job posts is 1430, while in fake job posts, the maximum number of words in the "information" column is 942. This conclusion is sensible, as the amount of missing information on the fraudulent job posts were more.

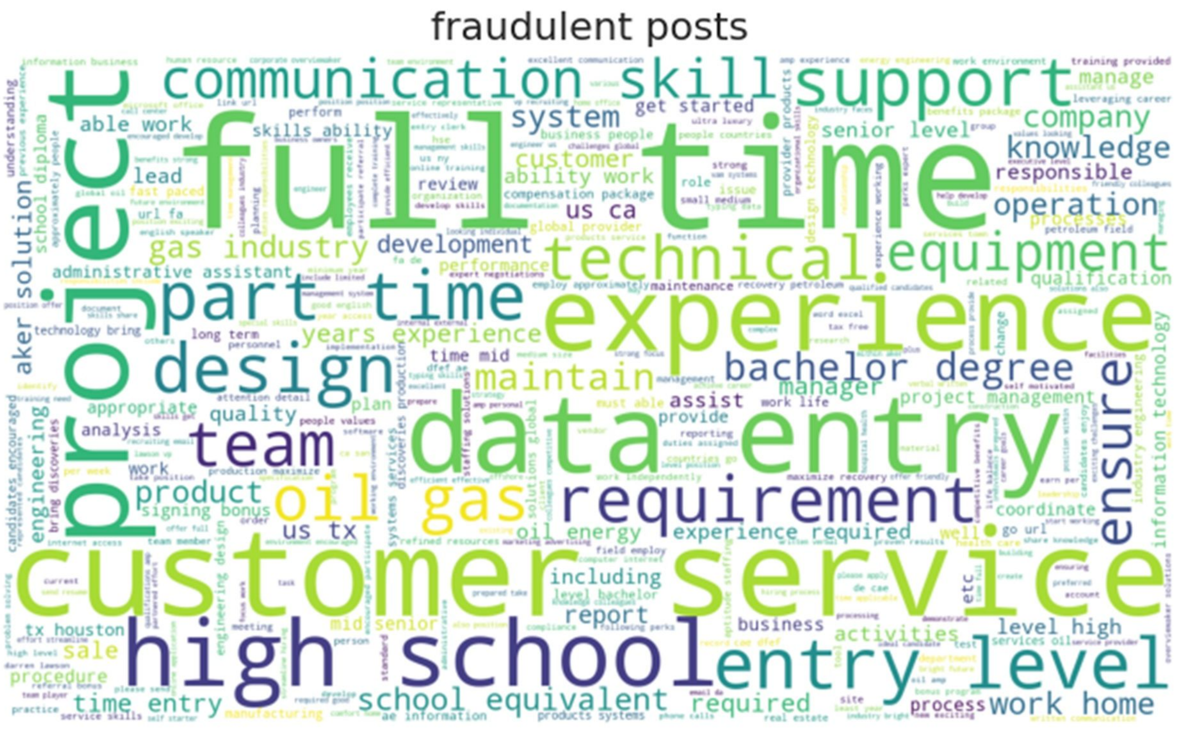
Figure 4- Length of information column based on the class variable

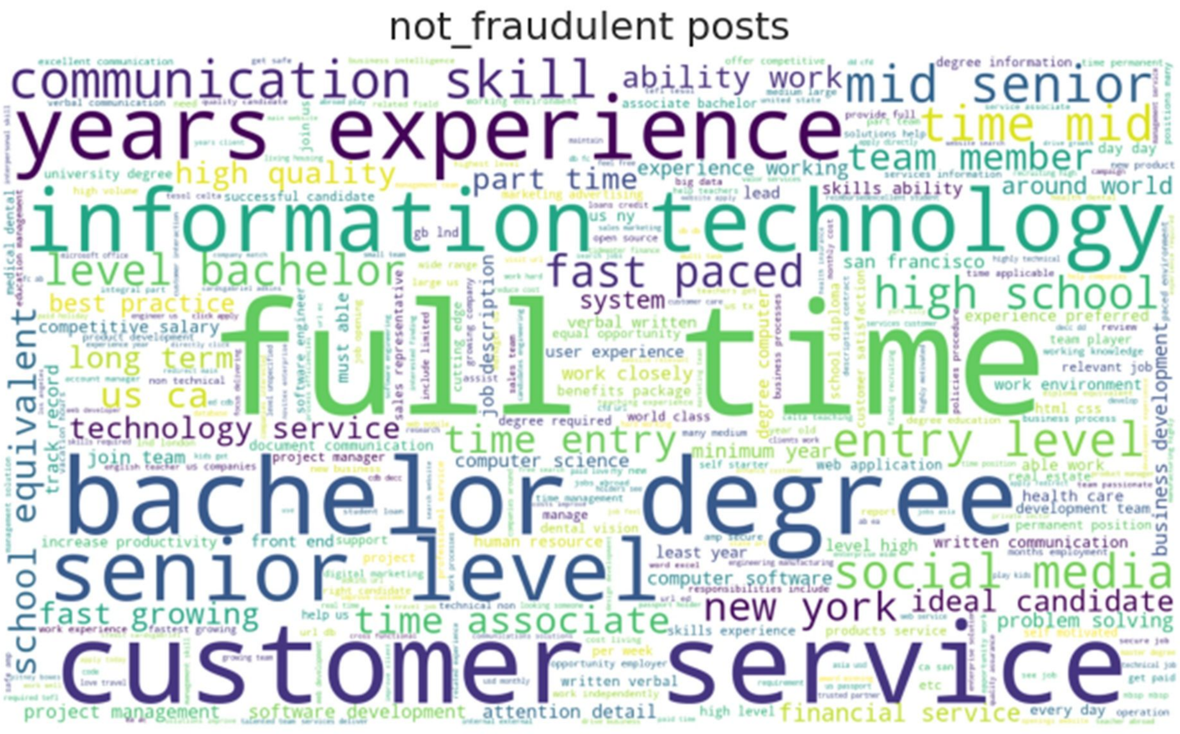


Before start modelling, to work with the text data, the text cleaning process is essential. For all text data in the "title" column and the "information" column, a text cleaning function has been defined and applied to these columns. In this function, all data first changed to the lowercase format, then "text regularization" has been applied and only alphabetic text data kept. Then the "stop words" removed from texts. This "stop word" removal is done by implementing an NLP package in python named "nlkt". For all other steps such as data vectorization, this clean text data is used.

The last step in exploring text data is to check the common words in the "information" column based on the target value. An easy way of visualizing the common words in the text data is using "Word Clouds" in Natural Language Processing (NLP). In this structure, the font size is based on the frequency of a word which means that the more frequency the word has, the bigger the font size it will have. The below figures are the word clouds of the "information" column in two classes of fraudulent and legitimate. As figure 5 shows, words such as "full time" or "customer services" are common words in both classes while word such as "project" is only common in fraudulent job posts. Since the “information” column is in text format, Word Cloud tool is used to show 500 most common tokens in fraudulent and not_fraudulent job posts.

Figure 5- The most common words in fraudulent and legitimate job posts





RESULTS

The results of this project will be explained in two parts of unsupervised learning and supervised learning.

Unsupervised learning

The supervised learning of this project is the clustering of the job titles by using the “K-mean” algorithm. Although the k-mean algorithm is good in clustering a large amount of data, it has the limit of working only with numeric variables (Huang, 1997). The job title data is in the text format. To deal with the limitation of the k-mean algorithm and apply it to the job title data, the text data needs to be converted into a vector format. There are many approaches to convert text data into a vector format, such as “bag of words” or “TFIDF” vectorization. In this project, the Document to Vector (doc2vec) model is used for the text vectorization. By converting text data into the vector format, it will be possible to use k-mean algorithm for clustering job titles.

The Doc2vec model is a derivation of the Word to Vector (word2vec) model, that first introduced by Mikolov, Chen, Corrado and Dean in 2013 as a method to convert text data into a vector format. The advantage of the word2vec approach over similar prior methods was its ability to keep semantic relation between words. The doc2vec is an approach introduced by Le and Mikolov (2014) a year after introducing the word2vec model. Similar to the word2vec, doc2vec method also has the ability to keep semantic relations, but it is useful for paragraphs. The doc2vec model has no limitation of same-sized input and this makes it a better model for document vectorization. The doc2vector model, gives a vector format of the inputted paragraph in a specific size, though the inputted paragraph might have different lengths. This fixed-size vector of the text as the output of the doc2vec model saved the semantic relations, just like what word2vec models do for words.

Doc2vec model is an unsupervised learning task that needs tagged data as input, based on the structure of the model (Le, Mikolov, 2014), and returns the vector format of text data in a specified size. The text vectorization of the job titles has been done in three steps. First, all job titles in data have to be tagged; then the tagged data be used as an input of a doc2vec model, and the model be trained with that data. The last step is to find vectors of each job title. This unsupervised task in three steps turn text data of the job titles into the vector format.

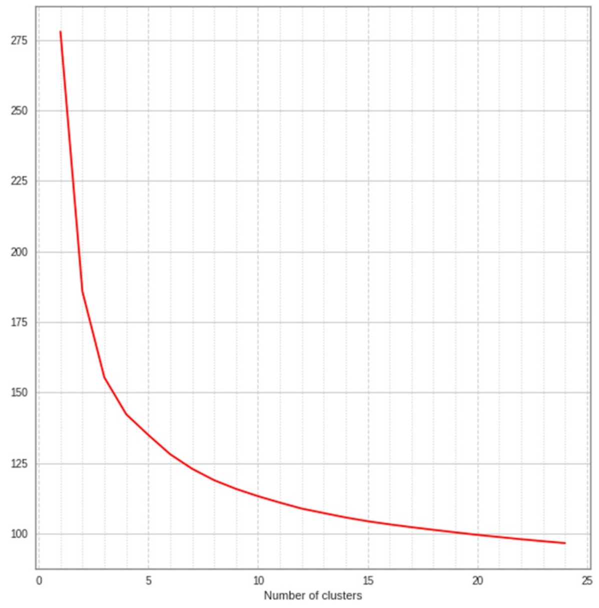
The other important part of training a doc2vec model is to define the output vectors size. In order to choose the best vector size for the job titles, it is needed to have again more knowledge about the number of words in the job title column of data. Based on the below data about the title column of this dataset, the vector size of 20 has been chosen:

- Minimum = 3
- Maximum = 138
- Mode = 22
- 99 percentiles = 63

Now that the data is in the numeric format, we would be able to use the k-mean algorithm for classification. For the k-mean algorithm, the number of clusters(k) must be specified. To find the right amount of k, the visual approach of the “Elbow curve” can be used. This visual approach is the oldest approach and mostly use to find the correct value of the k in the k-mean algorithm (Naeem and Wumaier, 2018). To apply the elbow method, different amounts of k will be tested and for each k, the amount of SSE will be calculated. By plotting different amounts of SSE by increasing the amount of k, an arm shaped line will be generated. For smaller amounts of k, the SSE drops dramatically, until a point that increasing one unit of k will not change SSE sharply and changes slow down. The elbow of the curve is the amount of k that the slope of the curve changes and the marginal gain of increasing the number of clusters decreased (Naeem and Wumaier, 2018). Since the target of clustering is to find the minimum of k as the true number of clusters, the elbow point is the best number of clusters for the k-mean algorithm and can be used for clustering.

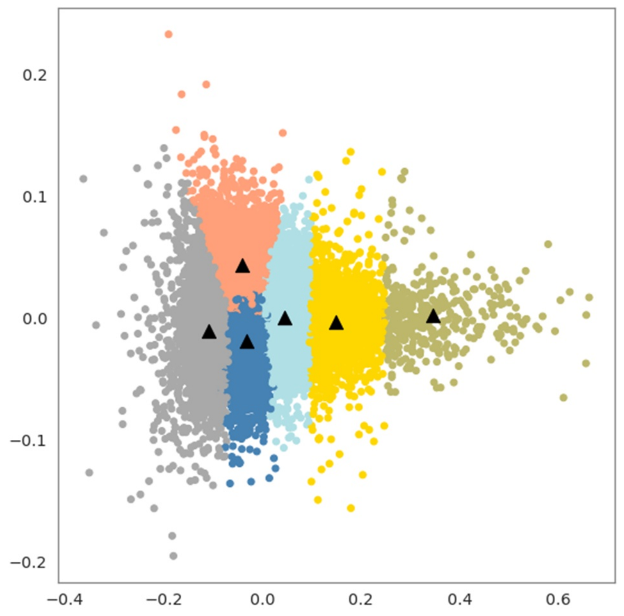
Figure 6 shows the elbow curve for the vectors of the job titles. The curve shows the true number of clusters is 6. Thus, k=6 will be considered as a correct amount in the k-mean algorithm. The x-axis of this graph shows different amounts of k in the range between 1 to 25 and the y-axis is the mean of the sum of squared distances between each observation and its closest centroid in the model.

Figure 6- The elbow curve for job title vectors



By considering the true amount of k as 6, the k-mean model created six clusters for the job title vectors. The below picture provides a visual picture of clusters. In figure 7, the Principal Component Analysis (PCA) transform had been applied to show results in a two-dimensional graph. Different colors used to show six different clusters and the black triangle shows the center of each cluster.

Figure 7- Six clusters of the k-mean algorithm



Supervised learning

For the supervised learning part, two models have been used to classify job posts into fraudulent and legitimate classes and results have been compared together. For both models, a doc2vector format of the information column has been used. For these classifications, all text data of each job post have been used which have been stored in the information column. Like unsupervised learning, the doc2vec model has been used for text vectorization. In this second doc2vec model the vector size has been set to 100 based on the length of the data in the information column. The first model of this project is a simple Decision Tree classifier as the baseline model. The second model is a weighted neural network. Both models used the same input and the results of these models have been compared.

For supervised learning, three subsets of data are needed for training, hyperparameters tuning (validation) and testing. Thus, all dataset has been split into three subsets of training, validation and, test subsets. The training subset has been used for model training purposes, the validation subset has been used for the hyperparameters tuning purposes and it is only needed in the neural model. The last subset of data is a test set that has been used to check the model performance. Hence, all 17880 job posts randomly split into 80% as training (16,092 job posts), 10% as validation (1,788 job posts) and 10% as test (1,788 job posts).

The other important note is about the “Accuracy Paradox” in this dataset. This simply means that when there is a large class imbalance in the dataset, further inspection is essential for model evaluation. Since the dataset has been used in this project have severely skewed class distribution, considering the model accuracy to evaluate and compare models is not enough. For instance, in this project, if a model falsely considered all job posts as legitimate, based on the contribution of the fraudulent and legitimate job posts, the model reaches the accuracy of 95%. However, in reality, this model performed badly and did not specify any fraudulent job posts. As a result, it is important to investigate model performance more accurately especially when the data set is imbalanced. Since it is possible that model with lower accuracy was preferred over the model with higher accuracy.

There are some traditional metrics to evaluate model performance which will be used in this project:

- Accuracy: total number of correct predictions made divided by the total number predictions
- Precision: the number of truly positive predictions divided by the total number of positive predicted.
- Recall: the number of positive predictions divided by the total number of positive class values.
- F1 Score: the balance score between the precision and the recall.
- Confusion matrix: a table that presents the class distribution in the data set and the classifiers predicted class distribution.

In this project, the accuracy paradox caused by the imbalanced class data exists. Hence, both classification reports and confusion matrix used to evaluate models' performance. The classification report is used to have more detailed information about the accuracy, precision and f1-score for each class. Moreover, the confusion matrix will present clear and unambiguous picture of the results of the model. Both the confusion matrix and classification report together can be used to evaluate models and comparing the performances.

The decision tree model has been used as the baseline model in this project. This model has two attributes that needed to be tuned in this project. First, since data is highly imbalanced, “class_weight” has to set as 'balanced'. By this regulation, the model calculated the class weights and consider them in the classification. The other hyperparameter that needed to be tuned is the selection measure. The “entropy” used as “criterion” in this project. By changing these two attributes the model is ready for the training section. After training, the model has been used to predict the classes of the test set. The results of this model prediction on the test set, have been considered as the baseline results.

Table 3 is the classification report of the decision tree model on the test set. This classification report provides detailed data of precision, recall and f1-score for both classes. The f1-score considered as the balanced score between the precision and the recall and has been used to indicate model performances in this report. The differences between this measure in two classes illustrate the differences in the model performance based on the target value. From table 3, the model has the f1-score of 96% when the true class is legitimate while this score is 0.27 when the job post is fraudulent. This distance between these two f1-scores means that the model performance is relatively poor when the job post is fraudulent while the accuracy is 92%. This table also has a column as support. The support column indicated that the table results are based on how many instances. To conclude, as mentioned the accuracy is not enough and the f1-score of both classes has to be considered for evaluation.

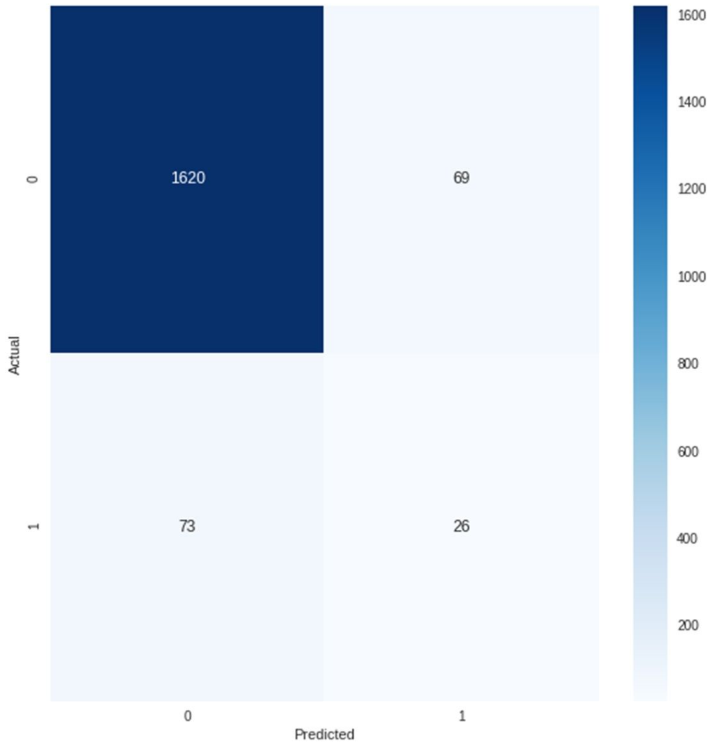
Table 3- Decision Tree classification report

Classification report	precision	recall	f1-score	support
0	0.96	0.96	0.96	1689
1	0.27	0.26	0.27	99
accuracy	-	-	0.92	1788
macro avg	0.62	0.61	0.61	1788
weighted avg	0.92	0.92	0.92	1788

As mentioned, in addition to the classification report, the confusion matrix can picture the breakdown of model performance. In a confusion matrix, both true class labels and predicted class labels will be shown. Furthermore, the numbers in the matrix indicated the number of predictions. For a binary classification like this project, the matrix will have two columns and two rows. Each row is for one true class label and each column is for one of the classifier possible predictions. There are two important cells as True Positive (TP) and True Negative (TN). True Positive is when the correct label is “one” and the model predicts it as “one” too. True Negative has a similar meaning for the “zero” class data, which means that when the true class is “zero” and the classifier predicts “zero” for it correctly. The bigger number in TP and TN mean better model performance. Two other cells in the matrix are False Negative (FN), which is when the true class is “one” but the classifier label it falsely as “zero”, and False Positive (FP), which happens when the true class is “zero” but the classifier output is “one” for that. Despite TN and TP cells, the fewer records in FN and FP cells is a sign of the ability of the classifier in predicting true classes. Thus, the confusion matrix helps to have a better understanding of the results of the classification report and model performance.

Figure 8 shows the decision tree classifier confusion matrix. In this picture, darker blue means a greater number of data, regardless of the correctness of the prediction. The Y-axis shows the correct class based on the EMCAD dataset and the X-axis shows the predicted class of the classifier. As the picture shows, when the job post is fraudulent (the actual class label is 1) the model only labels 26 out of 99 job posts correctly as a fraudulent (predicting 1 as the class label) and predicts the incorrect label of legitimate for 73 fraudulent job advertisements. Thus, the confusion matrix of figure 8 confirms the results of the classification report of the decision tree classifier.

Figure 8- Confusion matrix of the decision tree classifier

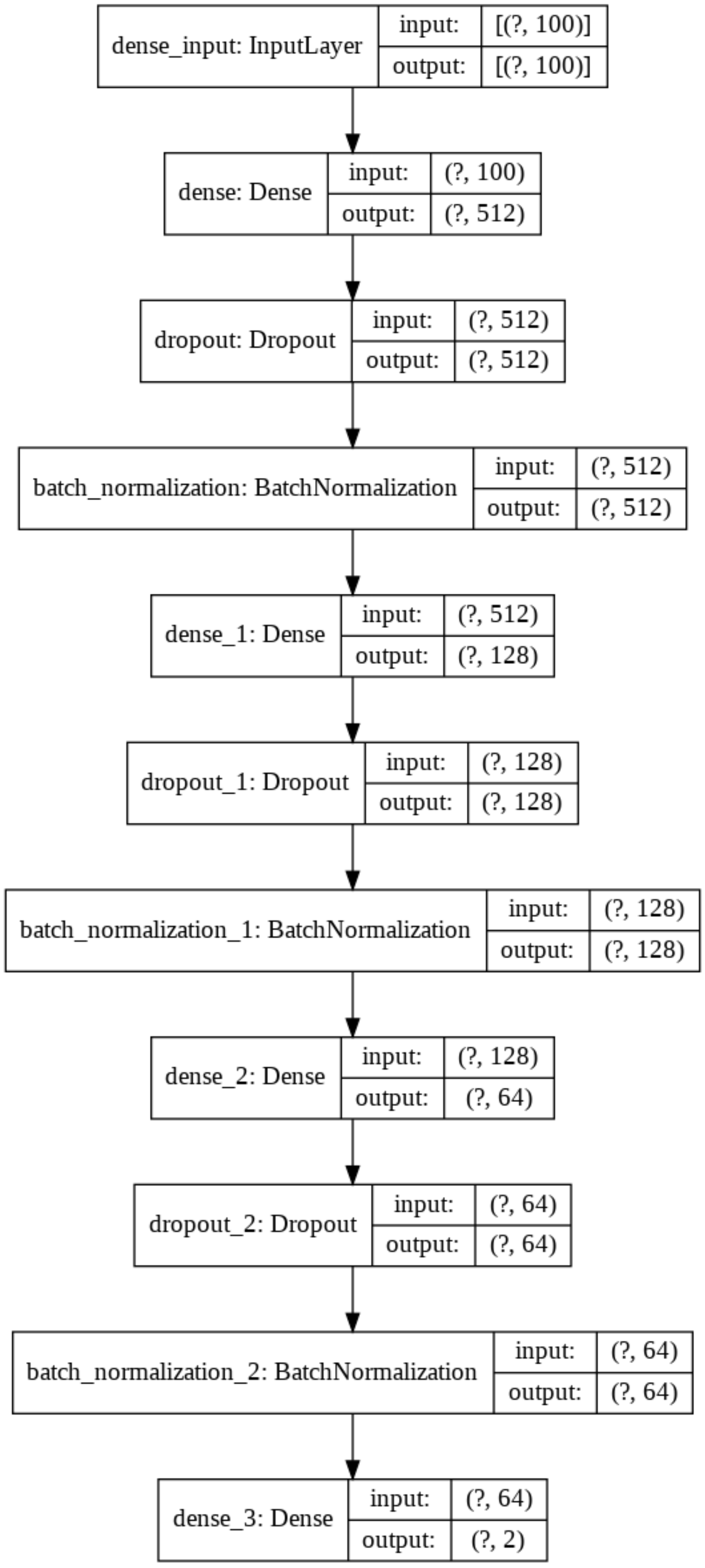


As the next step, a relatively new concept of supervised learning as the Neural network model used to classify job posts. The feed-forward neural networks can be imagined as a function approximation machines, that achieve some generalize insights from data (Goodfellow, Bengio and Courville, 2016, p. 169). Function approximation is a technique that uses available observation, and historical data to estimate an unknown function underlying in data. This process is somehow like the function that is known from the brain and the name of this technique is based on this similarity. Nowadays, this powerful new approach is used to solve complex classification. Hence, in this project, the feed-forward neural model has been chosen for classification.

The neural model used in this project is the sequential model that used the vector format of data as an input and return probabilities as output. The input of the model is doc2vec vectors of the information with the size of 100. The output of the model is two probabilities for each record. Since there are two classes, the model will return two probabilities, one for each class label. These probabilities have been defined as the probability, that the job post belongs to each class based on the learned features. The final predicted class is the class that shows the highest probability. Thus, the model accepts a vector and define the class label based on the probabilities.

Figure 9 shows the specific arrangement of the layers and nodes in the network. This picture shows the input layer, hidden layers and the output layer. It also shows the number of nodes in each layer. In addition to the input layer, hidden layers and the output layer, there are two other layers as dropout and batch normalization in the model architecture. The below picture shows both model's layers and layers' nodes to visualize the model structure.

Figure 9- Neural network model architecture



To solve the problem of having imbalanced class data in the dataset, the Weighted Neural Network is used in this project. The standard neural network algorithm does not support imbalanced classification. While in this dataset, class data is skewed. A cost-sensitive learning method can be used to solve the imbalanced data problem (Wang, Liu, Wu, Cao, Meng and Kennedy, 2016). The class_weight is a hyperparameter of the neural model that maps class values to each class in order to reflect their importance. In general, the best practice of class weighting is to use the inverse of the class distribution present in the training dataset. There are other methods to deal with imbalance data. For example, up-sampling examples from the minority class or down-sampling examples from the majority class to make data with balanced class data, and then use the balanced data to train the model. Based on Wand et al. (2016) if imbalance data had not been considered in the training step and model be trained with balance data, there is no guarantee for the good performance of the model on imbalance data after that. Hence, the Cost-Sensitive Neural Network by heuristic approaches for defining weighting has been chosen as a solution to deal with skewed class distribution in this project.

The dropout layer helps this model to prevent overfitting and increases model generalization power. Having large weights in the neural model means a more complex model and increases the risk of overfitting. Neural network models are powerful tools to define relations. As a result, these models need some regularizations to prevent overfitting. Dropouts help the model to prevent overfitting by the cost of creating a noisier training process (Srivastava, Hinton, Krizhevsky, Sutskever and Salakhutdinov, 2014). As Srivastava et al. (2014) mentioned, by applying dropout in some nodes, all incoming and outgoing connections of that nodes temporarily be removed from the network. Hence, dropout layers reduce model capacity and a wider model is needed when using dropouts based on the Srivastava et al. (2014) findings. Dropouts applied to a layer by using a fixed probability independently from other layers. Probabilities of 50% for dropout of hidden units and 20% for dropout of visible units give good results in a wide range of networks and tasks (Hinton, Srivastava, Krizhevsky, Sutskever and Salakhutdinov, 2012). The more model becomes overfitted, the less generalization ability it will have. In this weighted neural model, dropout regularization applied to prevent overfitting.

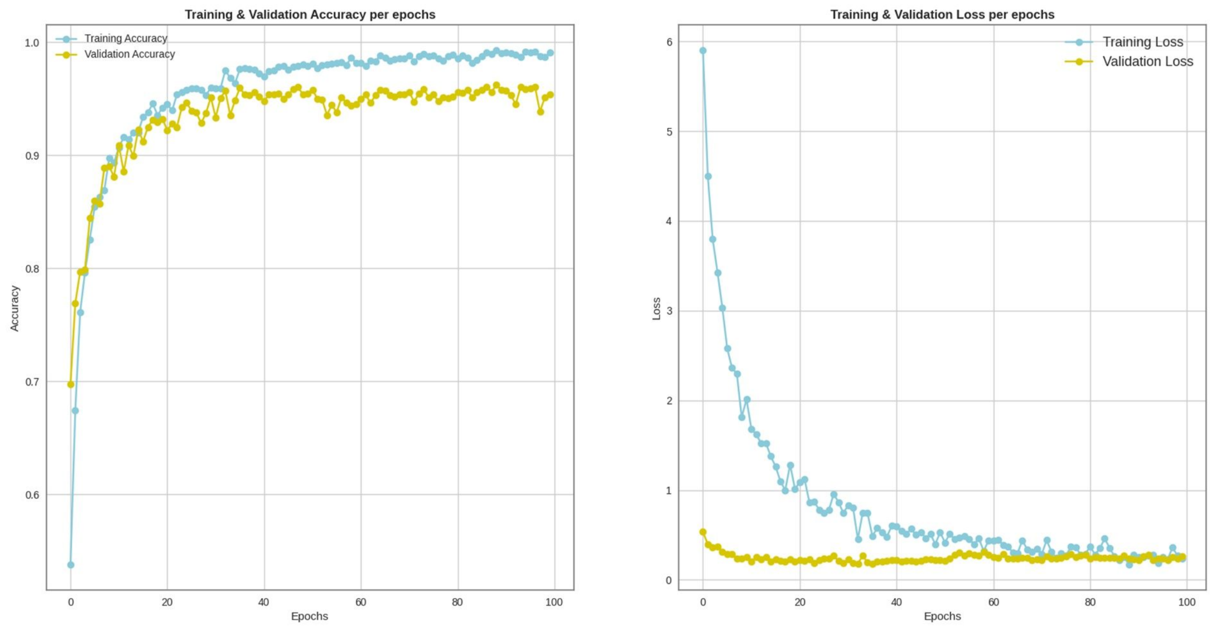
Furthermore, the batch normalization layer can be implemented to speed up the training process of the model (Ioffe and Szegedy, 2015). The batch normalization layer can also act as regularizer based on the paper of Ioffe and Szegedy (2015). This layer can be used before and after the activation function of the layer (Ioffe and Szegedy, 2015). The batch normalization is automatically standardizing the inputs to a layer. This technique is designed to accelerate the learning of the neural model. in this project also, the batch normalization layer has been used to speed up learning process of the model.

In this model Rectified Linear activation Unit (ReLU) used as the activation function. The ReLU activation function is a linear activation that returns the input value directly if it were above zero and returns zero if the input value were 0 or less. Implementing ReLU activation improves the performance of the feed-forward network in comparison to the sigmoid activation function since ReLU behavior is closer to linear (Goodfellow, Bengio and Courville, 2016, p. 194). In this project, using the ReLU activation function decrease the amount of training loss and increases the training set accuracy during the training process. Hence, the neural model of this project has been used ReLU activation function for input and hidden layers.

The learning rate is another hyperparameter that affects model performance. Base on the Goodfellow et al. (2016, p. 429) the learning rate is the most important hyperparameter to tune. The learning rate is the amount that the weights are updated during training. On one hand, large learning rates result in unstable training and on the other hand, small rates lead to fail in training. As a result, the learning rate must define cautiously. In this project also, several learning rates have been experimented to find the learning rate which performs well in the model.

In the end, by considering all the above notes a feed-forward neural network have been trained to classify job posts. The validation set was used to tune the hyperparameters of the model. The below table shows the changes in accuracy and loss of training set and validation set per 100 epochs by using a batch size of 64. As figure 10 shows, the training loss decreases considerably per epochs. Moreover, as the accuracy plot illustrates, the training accuracy is more than the validation accuracy which can be a sign of overfitting. However, using early stoppings or training models for fewer numbers of epochs reduced the model accuracy in the test set. This contradiction comes from the amount of loss in the training set. As mentioned before, a weighted model is more complicated, and the model needs enough epochs to learn weights.

Figure 10- Accuracy and loss history the Neural model per epochs for the training set and validation set



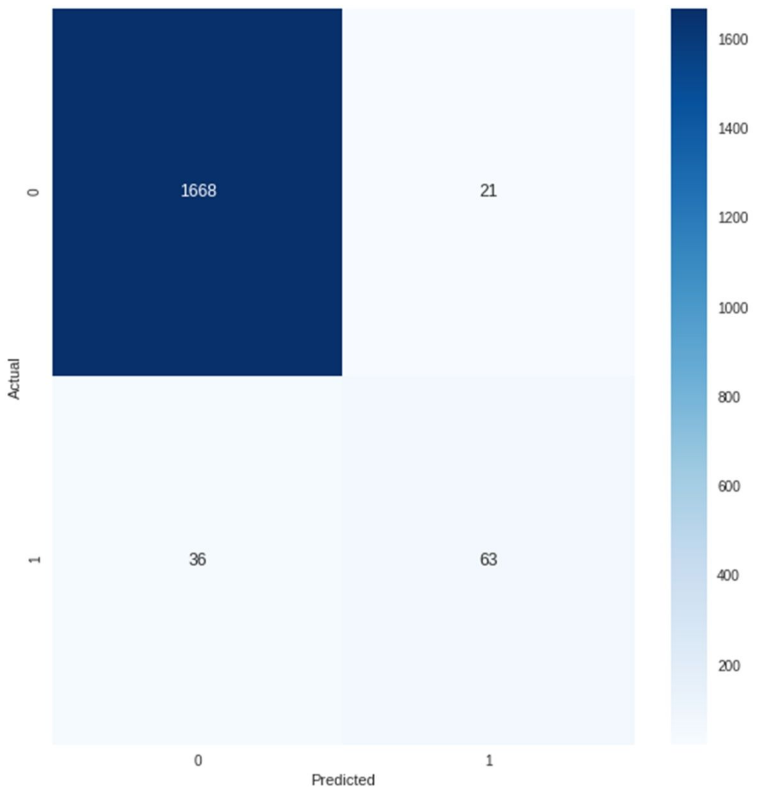
To check the model performance, the results of this classification has been compared with the decision tree classifier. First, a classification report is needed. Table 4 shows the F1 score of fraudulent job posts as 69% which is an improvement from the baseline model. However, the F1 score for legitimate job posts is almost the same. This means that by keeping approximately the same performance for the fraudulent class job posts, the model performance for the legitimate class has been increased. Hence, it can be concluded that this model is a better classifier in comparison to the decision tree model.

Table 4- Neural model classification report

Classification report	precision	recall	f1-score	support
0	0.98	0.91	0.95	1689
1	0.75	0.64	0.69	99
micro avg	0.97	0.90	0.93	1788
macro avg	0.87	0.77	0.82	1788
weighted avg	0.97	0.90	0.93	1788
samples avg	0.90	0.90	0.90	1788

From the general perspective of classification reports, the feed-forward neural model performed better. To have more detailed information about the model, figure 11 illustrates the confusion matrix of the neural network classifier. Same as figure 8, darker blue means a greater number of data, regardless of the correctness of the prediction. From the picture, when the job posts were legitimate, only 21 job advertisements have been mislabeled by the model. Moreover, when there were fraudulent job posts, the model has been correctly predicted the class label of 63 job advertisements as fraudulent and there were only 36 out of 99 mislabeled job posts from the model. By comparing figure 8 and figure 11, it can be concluded that the neural model has been classified both fraudulent and legitimate job posts more accurately. Thus, it can be concluded that the neural model is a better classifier and using a neural model for classification is an advance over decision tree classifier.

Figure 11- Confusion matrix of neural network classifier



CONCLUSIONS

In this project, two tasks of clustering and classification have been done. Data used for both of these two tasks were in the text format and needed vectorization. For both tasks, the document to vector format of the text data has been used. For the clustering, data of the “title” column, and for the classification, the combined column of all text data as the “information” column has been used. Based on the data of each column, different vector sizes have been selected in each task. Two parts of this project can be summarized and concluded as below:

Unsupervised learning: The clustering has been done by using the k-mean algorithm. To find the best amount of k in this algorithm, the visual approach of the elbow curve has been implemented. The results of this clustering can be used in the user interface of online job advertisements systems to speed up job seekers searches.

Supervised learning: The idea was, using all text data of a job post, regardless of the column that data entered, for classification. Then, the powerful approach of Neural Networks has been tested. The feed-forward cost-sensitive neural model designed for the classification and its results showed improvement over a decision tree classifier as an older approach. This classification can be used as an auto-detection for fraudulent job advertisements to reduce the amount of ORF. The other advantage of this model is the more input data it has, the more precise detection it would have. This feature of the neural model makes it even better model to be used in online platforms, with continuous learning, to help both job seekers and organizations.

Areas for improvement

As the last part of this project, it worth mentioning some areas for improvement for these two tasks that have been done in this project. About the clustering task, Since text data of this project is messy and with no structure, more advanced text cleaning can result in better clusters. The next point is about the vector size of the document to vector model. Dai, Olah, and Le, (2015) mentioned, the vector size in the doc2vec models can affect the performance of the vectorization. hence, a better vector size can perform better. In general, better input gives better output for clustering. Moreover, the elbow approach can be ambiguous in choosing the correct number of clusters and using other approaches can lead to more precise clusters. The last idea comes from an article by Huang (1997). This paper presents an algorithm named k-mode as an extension of the k-mean algorithm for categorical data. After making some changes in data, this approach can also be tested for clustering.

About the classification, Neural models have tons of hyperparameters that tuning them can improve model performance. Additionally, as a general idea, just like what mentioned for the clustering part, more advanced text clean and experimenting with the vector size of the doc2vec model can lead to a better performance of the model by providing better input for the model. Moreover, it can be also a good experiment to compare the neural model with more other classification models rather than only the decision tree classifier.

REFERENCES

Alghamdi, B., & Alharby, F. (2019). An Intelligent Model for Online Recruitment Fraud Detection. *Journal of Information Security*, *10*(3), 155-176.

Dadvar, M., Trieschnigg, D., Ordelman, R., & de Jong, F. (2013, March). Improving cyberbullying detection with user context. In European Conference on Information Retrieval (pp. 693-696). Springer, Berlin, Heidelberg.

Dai, A. M., Olah, C., & Le, Q. V. (2015). Document embedding with paragraph vectors. arXiv preprint arXiv:1507.07998.

Dinakar, K., Reichart, R., & Lieberman, H. (2011). Modeling the Detection of Textual Cyberbullying.

Ghazzawi, K., & Accoumeh, A. (2014). Critical success factors of the e-recruitment system. Journal of Human Resources Management and Labor Studies, 2(2), 159-170.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Huang, Z. (1997). A fast clustering algorithm to cluster very large categorical data sets in data mining. DMKD, 3(8), 34-39.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Kapse, A. S., Patil, V. S., & Patil, N. V. (2012). E-recruitment. International Journal of Engineering and Advanced Technology, 1(4), 82-86.

Lal, S., Jiaswal, R., Sardana, N., Verma, A., Kaur, A., & Mourya, R. (2019, August). ORFDetector: Ensemble Learning Based Online Recruitment Fraud Detection. In *2019 Twelfth International Conference on Contemporary Computing (IC3)* (pp. 1-5). IEEE.

Le, Q., & Mikolov, T. (2014, January). Distributed representations of sentences and documents. In International conference on machine learning (pp. 1188-1196).

Mahbub, S., & Pardede, E. (2018). Using contextual features for online recruitment fraud detection.

McKeown, K., & Wang, W. (2010). “Got You!”: Automatic vandalism detection in wikipedia with web-based shallow syntactic-semantic modeling.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Naeem, S., & Wumaier, A. (2018). Study and implementing K-mean clustering algorithm on English text and techniques to find the optimal value of K. International Journal of Computer Applications, 182(31), 7-14.

Roy, S. S., & Viswanatham, V. M. (2016). Classifying spam emails using artificial intelligent techniques. In *International Journal of Engineering Research in Africa* (Vol. 22, pp. 152-161). Trans Tech Publications Ltd.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

University of the Aegean - Laboratory of Information & Communication Systems Security (2020). EMCAD dataset. Retrieved from http://emscad.samos.aegean.gr/

Vidros, S., Kolias, C., & Kambourakis, G. (2016). Online recruitment services: Another playground for fraudsters. *Computer Fraud & Security*, *2016*(3), 8-13.

Vidros, S., Kolias, C., Kambourakis, G., & Akoglu, L. (2017). Automatic detection of online recruitment frauds: Characteristics, methods, and a public dataset. *Future Internet*, *9*(1), 6.

Wang, S., Liu, W., Wu, J., Cao, L., Meng, Q., & Kennedy, P. J. (2016, July). Training deep neural networks on imbalanced data sets. In 2016 international joint conference on neural networks (IJCNN) (pp. 4368-4374). IEEE.

Zhang, N., & Yuan, Y. (2012). Phishing detection using neural network. *CS229 lecture notes*.

Zhang, Y., Hong, J. I., & Cranor, L. F. (2007, May). Cantina: a content-based approach to detecting phishing web sites. In Proceedings of the 16th international conference on World Wide Web (pp. 639-648).

Appendix

MSCI623- Big Data Analytics

University of Waterloo - spring 2020

These codes have two goals:

1. Using the K-mean algorithm to clustering job titles - *Unsupervised learning*
2. Using a neural network to detect "fraud" and "not_fraud" job posts - *Supervised learning*

by implementing python language.

Preparing coding environment

These codes have been run in **Google Colab** and the coding environment has been set based on that.

1. Importing essential libraries and download some extra components of packages(need to upgrade tensorflow library)
2. Setting random states to have consistency in answers every time of running codes
3. Turning warnings off
4. Uploading data
5. Saving data as a dataframe
6. Setting a drive on google drive to be able to save models, pictures and graphs
7. Checking system GPU
8. Running personalized functions that will be used in the rest of the codes

```
# Upgrade tensorflow library
!pip install tensorflow
!pip install --upgrade tensorflow
!pip install tf-nightly
```

```
# Import libraries
import io
import os
import warnings
import random
import statistics
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import re
import nltk
import gensim
from nltk.tokenize import word_tokenize,sent_tokenize
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from wordcloud import WordCloud
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from sklearn.tree import DecisionTreeClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization
from  tensorflow.keras.utils import plot_model
from sklearn.metrics import classification_report,confusion_matrix

print('All required libraries were imported')
```

```
# Downloading essential components of nltk library and set them
nltk.download('stopwords')
stopwords = set(nltk.corpus.stopwords.words("english"))
nltk.download('punkt')
print('\n', '*'*15, 'All extra packages downloaded and set', '*'*15)
```

```
# set random seeds
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)
PYTHONHASHSEED=123
print('Random seeds set')
```

```
# Turning warnings off
warnings.filterwarnings("ignore")
print('All warnings turned off')
```

```
# Upload data file and wait untill become 100% done. This might takes some time
from google.colab import files
uploaded = files.upload()
```

```
# Saving data on dataframe
job_posts = pd.read_csv('fake_job_postings.csv')
print('CSV file has been saved in pandas dataframe as "job_posts"')
```

```
# Set a folder to save images and tabels
from google.colab import drive
drive.mount('/content/gdrive')
os.chdir("gdrive/My Drive/MSCI 623-Project")
print(os.getcwd())
print('MSCI623-Project in google drive set as repository')
```

```
# Checking the GPU of running codes (If using google colab GPU the output should be "/devi
tf.test.gpu_device_name()
```

```
# This function clean text data and return it as text
def clean_text(text):
  text = text.lower()  # lower case the text
  text = re.sub(r"[^a-z]+", ' ', text)  # remove all signs and punctuation and numbers
  words = word_tokenize(text)  # Tokenize data
  wordsFiltered = []
  for w in words:
    if w not in stopwords:
      wordsFiltered.append(w)

  return '{}'.format(' '.join(wordsFiltered))

print('Personalized function have been loaded')
```

Exploratory data analysis (EDA)

Next codes are to explore the original data.

```
# To check data shape
print('# of columns:', job_posts.shape[1])
print('# of rows:', job_posts.shape[0])
```

```
# To check dataframe
job_posts.head()
```

```
# Understanding numeric columns of data
job_posts.describe()
```

The "job_id" is table key and the other four numeric columns as "telecommuting", "has_company_logo", "has_question" and "fraudulent" are binary variables.
The "fraudulent" column is the target column.

```
# Checking share of the target value
fake_count = (job_posts["fraudulent"]==1).sum()
true_count=(job_posts["fraudulent"]==0).sum()
total = fake_count + true_count
print('# of fake posts:',fake_count, '({:4.2f}'.format(fake_count*100/total),'% of total)'
print('# of true posts:',true_count, '({:4.2f}'.format(true_count*100/total),'% of total')
```

```
# Visualizing share of the target value
fig, axes = plt.subplots()
```

```
fig, axes = plt.subplots()
plt.tight_layout()

job_posts["fraudulent"].value_counts().plot(kind='pie',  labels=['not_fraudulent \n (~95%)
axes.set_ylabel(' ')
plt.title('Fraudulent and legitimate(not_fraudulent) share of data', fontsize=13)

fig.savefig('Images/datashare.png')
plt.show()
```

Data is considerably imbalanced.

```
# Visualizing share of other three binary variables considering the target value
fig, ax = plt.subplots(1,2)
plt.tight_layout()

tab1 = pd.crosstab(job_posts['fraudulent'],job_posts['telecommuting'])
tab1.rename(columns={1: 'telecom', 0: 'not_telecom'},inplace=True)
for i,cat in enumerate(tab1.index):
    tab1.loc[cat].plot.pie(ax=ax[i],startangle=260, colors= np.array(['skyblue','khaki']))
    ax[0].set_title('True_job_posts', fontweight='bold')
    ax[1].set_title('Fake_job_posts', fontweight='bold')
    ax[0].set_ylabel('telecommuting position')
    ax[0].yaxis.labelpad=20.0
    ax[1].set_ylabel('')
fig.savefig('Images/telecom_pos_share.png')
# plt.show()

fig, ax = plt.subplots(1,2)
plt.tight_layout()
tab2 = pd.crosstab(job_posts['fraudulent'],job_posts['has_company_logo'])
tab2.rename(columns={1: 'has_logo', 0: 'no_logo'},inplace=True)
for i,cat in enumerate(tab2.index):
    tab2.loc[cat].plot.pie(ax=ax[i],startangle=45, colors= np.array(['skyblue','khaki']))
    ax[0].set_title('True_job_posts', fontweight='bold')
    ax[1].set_title('Fake_job_posts', fontweight='bold')
    ax[0].set_ylabel('has_company_logo in posts')
    ax[0].yaxis.labelpad=20.0
    ax[1].set_ylabel('')
fig.savefig('Images/Logo_share.png')
# plt.show()

fig, ax = plt.subplots(1,2)
plt.tight_layout()
tab3 = pd.crosstab(job_posts['fraudulent'],job_posts['has_questions'])
tab3.rename(columns={1: 'has_questin', 0: 'no_question'},inplace=True)
for i,cat in enumerate(tab3.index):
    tab3.loc[cat].plot.pie(ax=ax[i],startangle=180, colors= np.array(['skyblue','khaki']))
    ax[0].set_title('True_job_posts', fontweight='bold')
    ax[1].set_title('Fake_job_posts', fontweight='bold')
    ax[0].set_ylabel('has_question in posts')
    ax[0].yaxis.labelpad=20.0
    ax[1].set_ylabel('')
fig.savefig('Images/question_share.png')
```

```
# plt.show()
```

While most real job posts show the company logo, the big portion of fake job posts has no logo. Moreover, a smaller share of fake job posts has questions.

```
# To check the correlation between three binary variables and target value
cor_data = job_posts[['fraudulent','telecommuting','has_company_logo','has_questions']]
corr = cor_data .corr(method="pearson")

fig, axis= plt.subplots(figsize=(8, 8))
plt.tight_layout()

sns.heatmap(corr, cmap= "RdBu", center=0.00, annot=True, fmt='.1g',cbar_kws={'label': 'cor
sns.set(font_scale=1.3)

fig.savefig('Images/binary_cor.png')
plt.show()
```

Though there is no high correlation between variables, telecommuting position shows a bigger positive correlation with target value in comparison to the other two.

```
# Checking missing values in the data set
NA = job_posts.isna().groupby(job_posts.fraudulent).sum().T
NA['Total']= NA.sum(axis=1)
NA.to_csv('data/NA_table.csv', index=True, encoding='utf-8')
NA
```

The data set has a lot of missing information. However, based on the text mining approach in this project, all text columns will be considered as a package of text information. Thus, the only concern will be the salary range column, which could be eliminated.

Cleaning data

All 12 columns as "title", "location", "department", "company_profile", "description", "requirements", "benefits", "employement_type", "required_experience", "required_education", "industry" and "function" will be combined in information column and the original columns will be removed, except for "title" column. The "title" column is needed for unsupervised learnin(clustering).

```
# To keep the main dataset and make some changes to the new one
data = job_posts.copy()
data.fillna(" ",inplace = True)

data.drop(['job_id'], axis=1, inplace=True)
data['information'] = data['title']  + ' ' + data['location'] + ' '  + ' ' + data['departm
```

```
data.drop(['location', 'department', 'company_profile','description', 'requirements', 'ben
data.drop(['telecommuting','has_company_logo','has_questions'],axis=1,inplace=True)
data.drop(['salary_range'], axis=1, inplace=True)
data=data[['title','information','fraudulent']]
data.head()
```

The new data set has no missing value in the remaining columns and data is ready for the next steps.

Unsupervised learnin - Clustering

The **k-means clustering** algorithm will be used to clustering **job titles** for unsupervised learning. The k_means algorithm expect numeric data as input. Thus, the **doc2vec** approach will be applied to convert alphabetic "title" data to a numeric format.

```
# To cleaning text data and check it(this might takes some time)
df=data.copy()
df.information=df.information.apply(lambda x: clean_text(x))
df.title=df.title.apply(lambda x: clean_text(x))
df.head()
```

Doc2vec for job titles

Document to vector(doc2vec) model of "gensim" library will be implemented to convert titles to vectors.

```
title_len = list(df.title.apply(lambda x : len(x)))
print('"Title length information"')
print(' Minimun:', int(np.min(title_len)) )
print(' Mode:', int(statistics.mode(title_len)) )
print(' 99 percentile:', int(np.percentile(title_len, 99)) )
print(' Maximum:', int(np.max(title_len)) )
```

```
# To prepare data for training doc2vec model for titles
titles = list(df.title.apply(lambda x: word_tokenize(x)))
tagged_titles= [TaggedDocument(doc, [i]) for i, doc in enumerate(titles)]
print('Title data are ready to be used to train doc2vec model')
```

```
# Create doc2vec model for titles (this might takes some time)
d2v_titels = Doc2Vec(vector_size=20, min_count=2, epochs=10)
d2v_titels.build_vocab(tagged_titles)
d2v_titels.train(tagged_titles, total_examples=d2v_titels.corpus_count, epochs=d2v_titels.
print('Doc2Vec model for titles trained')
```

```
d2v_titels.save('data/d2v_titels.model')
print('Doc2vec model for titles saved in the data folder')
```

```
# Extracting the doc2vec vectors for the job titles
titles_vec = d2v_titels.docvecs.vectors_docs
print('All',len(titles_vec),'job titles converted to their vectors.')
```

k_mean clustering

To find the optimal number of clusters, the visual approach of "Elbow method" will be used.

```
# Testing different amount of k for k-mean model and calculate SSE for them (this might ta
distortations = {}
for k in range(1,25):
  kmeans = KMeans(n_clusters=k,init='k-means++')
  kmeans.fit(titles_vec)
  distortations[k] = kmeans.inertia_
print('Data for different amount of K has been saved')
```

```
# Plotting elbow curve
fig, axes = plt.subplots(figsize=(8, 8))
plt.tight_layout()

plt.plot(list(distortations.keys()),list(distortations.values()), color='red')
plt.title('Elbow curve to find number of clusters for titles')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
axes.set_facecolor('white')
plt.grid(b=True, which='major', color='lightgrey', linestyle='--', axis='x')
plt.grid(b=None, which='minor', color='lightgrey', linestyle=':', axis='x')
plt.minorticks_on()
axes.spines['bottom'].set_color('0.5')
axes.spines['top'].set_color('0.5')
axes.spines['right'].set_color('0.5')
axes.spines['left'].set_color('0.5')

fig.savefig('Images/elbow.png')
plt.show()
```

Thus, the best number of clusters could be 6.

```
# To model K-mean
km_model = KMeans(n_clusters=6, init='k-means++')
km_model.fit(titles_vec)
labels=km_model.labels_.tolist()
print('k-mean clustering model has been created with k=6')
```

The "Principal Component Analysis(PCA)" will be used to plot the clusters and show how the model performs visually. PCA makes it easier to see clusters.

```
# To visualize the clustering model
predict = km_model.fit_predict(titles_vec)
pca = PCA(n_components=2).fit(titles_vec)
datapoint = pca.transform(titles_vec)

fig, axes = plt.subplots(figsize=(8, 8))
plt.tight_layout()

color_theme = ['darkgray','lightsalmon','powderblue','steelblue','gold', 'darkkhaki','turq
color = [color_theme[i] for i in predict]

plt.scatter(datapoint[:, 0], datapoint[:, 1], c=color)
centroids = km_model.cluster_centers_
centroidpoint = pca.transform(centroids)
plt.scatter(centroidpoint[:, 0], centroidpoint[:, 1], marker='^', s=150, c='black')
plt.title('K-Means Classification')
axes.set_facecolor('white')
axes.grid()
axes.spines['bottom'].set_color('0.5')
axes.spines['top'].set_color('0.5')
axes.spines['right'].set_color('0.5')
axes.spines['left'].set_color('0.5')

fig.savefig('Images/clusters.png')
plt.show()
```

Now we will add labels to our dataset.

```
# # To check the dataset
df['title_cluster']= labels
df=df[['title','title_cluster','information','fraudulent']]
df.head()
```

Take a look at the number of data in each clusters:

```
# To count number of titles in each cluster
c_table = df.groupby('title_cluster').count()
c_table.drop(['information','fraudulent'],axis=1,inplace=True)
c_table
```

Supervised learning - Classification

Two classifier will be used for classificatioin, **Decision tree** and **Neural networks**.

Data preperation

To apply text mining approaches to the information column, it is good to get a better insight into the texts in that column.

```
# To preparing data for classification
cdata = df.copy()
cdata['tokenized_info']= cdata['information'].apply(lambda x: word_tokenize(x))
cdata.drop(['title','title_cluster'],axis=1,inplace=True)
cdata = cdata[['information','tokenized_info','fraudulent']]
cdata.head()
```

```
# Getting an idea about the information column
fake_info_len = list(cdata[cdata["fraudulent"]==1]['tokenized_info'].apply(lambda x: len(x
true_info_len = list(cdata[cdata["fraudulent"]==0]['tokenized_info'].apply(lambda x: len(x
all_info_len = [*true_info_len, *fake_info_len]
print('number of fake sentences:',len(fake_info_len))
print('number of true sentences:',len(true_info_len))
print('Maximum number of words in information part for fake posts:',max(fake_info_len))
print('Maximum number of words in information part for true posts:',max(true_info_len))
```

```
# visualizing the length of data in the information column
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(15, 5))

ax1.hist(fake_info_len,bins = 20,color='lightsalmon')
ax1.set_title('fraudulent posts')
ax1.set_xlabel('Number of words', fontweight='bold',fontsize=11)
ax1.set_facecolor('white')
ax1.spines['bottom'].set_color('0.5')
ax1.spines['top'].set_color('0.5')
ax1.spines['right'].set_color('0.5')
ax1.spines['left'].set_color('0.5')

ax2.hist(true_info_len, bins = 20,color='steelblue')
ax2.set_title('not_fraudulent Post')
ax2.set_xlabel('Number of words', fontweight='bold',fontsize=11)
ax2.set_facecolor('white')
ax2.spines['bottom'].set_color('0.5')
ax2.spines['top'].set_color('0.5')
ax2.spines['right'].set_color('0.5')
ax2.spines['left'].set_color('0.5')

fig.suptitle('Number of words in information part',fontweight='bold',fontsize=12)

fig.savefig('Images/info_len.png')
plt.show()
```

The "word cloud" is a famous visual method in natural language processing to get an insight into text data.

```
# Using visualization to see the most frequent tokens
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(15, 5), dpi=100)
plt.tight_layout()

wc_fake = WordCloud(width = 1400 , height = 800 , max_words = 500 , background_color ='whi
ax1.imshow(wc_fake)
ax1.set_title('fraudulent posts')
ax1.axis('off')

wc_true = WordCloud(width = 1400 , height = 800 , max_words = 500 , background_color ='whi
ax2.imshow(wc_true)
ax2.set_title('not_fraudulent posts')
ax2.axis('off')

plt.axis("off")
plt.grid(b=None)

fig.savefig('Images/wordclouds.png')
plt.show()
```

```
# To check data types
print('Dataset data types')
print(cdata.dtypes)
```

For unsupervised learning, data has been split into three parts. "Training" data will be used to train the model, "Validation" will be applied for tuning the model's hyperparameters and "Test set" will be used to check the performance of the model.

```
# Splitting data into three sets
x= list(cdata.tokenized_info)
y= to_categorical(np.array(cdata.fraudulent))

train_x , val_x ,train_y , val_y = train_test_split( x , y , test_size = 0.1 , random_stat
train_x , test_x ,train_y , test_y = train_test_split( x , y , test_size = 0.1 , random_st

print('Data randomly splited to the training set(80%), validation set(10%) and test set(10
print('*'*20)
print('training set size=', len(train_x))
print('validation set size=', len(val_x))
print('testing set size=', len(test_x))
```

Doc2vec for job posts text data

Document to vector model will be applied to the information column which contains all text information about the job posts.

```
# Prepare training data to be used  in doc to vec model
tagged_info= [TaggedDocument(doc, [i]) for i, doc in enumerate(train_x)]
print('Training data is ready to be used for doc2vec model')
```

```
# Building Doc2vec model for information column
d2v_info = Doc2Vec(vector_size=100, min_count=1, epochs=20)
d2v_info.build_vocab(tagged_info)
d2v_info.train(tagged_info, total_examples=d2v_info.corpus_count, epochs=d2v_info.epochs)
print('Doc2Vec model has been trained for information column')

d2v_info.save('data/d2v_info.model')
print('Doc2vec model for information column saved in data folder')
```

Pre Processing data

All three data sets information have to be converted into vectors to be used as an input of the neural network.

```
# Pre_processing data (this might takes some time)
train_vec= np.array([d2v_info.infer_vector(item) for item in train_x])
val_vec =  np.array([d2v_info.infer_vector(item) for item in val_x])
test_vec = np.array([d2v_info.infer_vector(item) for item in test_x])
print('pre-processing data is finished')
```

Classification models

Decision tree Model

```
# To create and fit the decision tree model
dt_model = DecisionTreeClassifier(class_weight='balanced', criterion = 'entropy')
dt_model.fit(train_vec,np.argmax(train_y,axis=1))
```

Testing decision tree model performance

```
# To apply model on test set
dt_pred= dt_model.predict(test_vec)
dt_report = classification_report(np.argmax(test_y,axis=1),dt_pred,target_names = ['0','1'
print(dt_report)
```

```
# To graph the confusion matrix
```

```
dt_cm=confusion_matrix(np.argmax(test_y,axis=1),dt_pred)
dt_cm = pd.DataFrame(dt_cm)
dt_cm.index.name = 'Actual'
dt_cm.columns.name = 'Predicted'

fig, axis= plt.subplots(figsize=(8, 8))
plt.tight_layout()

sns.heatmap(dt_cm ,cmap= "Blues",annot = True, fmt='')

fig.savefig('Images/dt_confusion_matrix.png')
plt.show()
```

Neural network - Sequential mode

```
# Defining model
model = Sequential(name='Neural_Model')
model.add(Dense(512, activation='relu', input_dim=100))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(2, activation='sigmoid'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='categorical_c

model.summary()
```

```
# To plot the model
plot_model(model, to_file='Images/Neural_model.png', show_shapes=True, show_layer_names=Tr
```

```
# Fitting model (this takes some times)
weights = {0:5, 1:100}
history = model.fit(train_vec, train_y, class_weight=weights, batch_size=64, epochs=100, v
print('\n','*'*20,'Model trained','*'*20)
```

```
# Saving sequentioal model
model.save('data/sequential_model')
print('Sequential model has been saved')
```

```
# Plotting the accuracy and loss of the training and validation sets during epochs
fig, (ax1,ax2)= plt.subplots(ncols=2, figsize=(10, 5), dpi=100)

epochs = [i for i in range(100)]
train_acc = history.history['accuracy']
train_loss = history.history['loss']
```

```
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
fig.set_size_inches(20,10)

ax1.plot(epochs , train_acc , 'co-' , label = 'Training Accuracy')
ax1.plot(epochs , val_acc , 'yo-' , label = 'Validation Accuracy')
ax1.set_title('Training & Validation Accuracy per epochs',fontsize=12, fontweight='bold')
ax1.legend()
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Accuracy")
ax1.set_facecolor('white')
ax1.spines['bottom'].set_color('0.5')
ax1.spines['top'].set_color('0.5')
ax1.spines['right'].set_color('0.5')
ax1.spines['left'].set_color('0.5')
ax2.plot(epochs , train_loss , 'co-' , label = 'Training Loss')
ax2.plot(epochs , val_loss , 'yo-' , label = 'Validation Loss')
ax2.set_title('Training & Validation Loss per epochs', fontsize=12, fontweight='bold')
ax2.legend( fontsize= 13)
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Loss")
ax2.set_facecolor('white')
ax2.spines['bottom'].set_color('0.5')
ax2.spines['top'].set_color('0.5')
ax2.spines['right'].set_color('0.5')
ax2.spines['left'].set_color('0.5')
fig.suptitle('Fitting model history per epoches',fontweight='bold')

fig.savefig('Images/learning_loss.png')
plt.show()
```

Testing Neural model performance

```
# To apply model on test set
pred_prop =model.predict(test_vec)
pred=np.around(pred_prop , decimals = 0)
report = classification_report(test_y,pred)
print(report)
```

```
# To graph the confusion matrix
cm=confusion_matrix(np.argmax(test_y,axis=1),np.argmax(pred,axis=1))
cm = pd.DataFrame(cm)
cm.index.name = 'Actual'
cm.columns.name = 'Predicted'

fig, axis= plt.subplots(figsize=(8, 8))
plt.tight_layout()

sns.heatmap(cm ,cmap= "Blues",annot = True, fmt='')

fig.savefig('Images/confusion_matrix.png')
plt.show()
```