PREDICTION OF THE APPLY RATE OF JOB POSTINGS BASED ON THE JOB CHARACTERISTICS

A Thesis

by

SUMOD PRAMOD BADCHHAPE

Submitted to the College of Graduate Studies
Texas A&M University-Kingsville
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2020

Major Subject: Computer Science

PREDICTION OF THE APPLY RATE OF JOB POSTINGS BASED ON THE JOB CHARACTERISTICS

A Thesis

by

SUMOD PRAMOD BADCHHAPE

Approved as to style and content by:

Ayush Goyal, Ph.D.
(Chair of Committee)

David Hicks, Ph.D.
(Co-chair of Committee)

George Toscano, Ph.D.
(Member of Committee)

Scott Smith, Ph.D.
(Chair of Department)

George Allen Rasmussen, Ph.D.
Vice President
for Research and Graduate Studies

May 2020

ABSTRACT

Prediction of the Apply Rate of Job Postings Based on the Job Characteristics

(May 2020)

Sumod Pramod Badchhape, B.E.

University of Mumbai

Chairman of Advisory Committee: Dr. Ayush Goyal

Job portals and job listings such as Glassdoor, Indeed, and LinkedIn use different data mining techniques and machine learning algorithms to provide the best job recommendations based on a candidate's preferences. Job recommendations are not only based on the preferences set by candidates. Rather, there are other parameters that need to be considered as well, such as – skills required by the job, relocation being provided or not, searched keywords, visa sponsorship, etc. The job recommendations provided by these job recommendation systems play a significant role in the company's growth and the *"Apply Rate"* or *"Click Through Rate"* (CTR) for a particular job posting. CTR is a metric used to measure the success of an online advertisement campaign and how it impacts advertisement rank and quality score. In this research, the apply rate is considered as the measure of whether a user will apply to that particular job or posting based on different characteristics such as job title, job description, popularity of job, job listing matches, location, time period of job listing, etc. The results of this research can be analysed in order to improve future job searches, click rates, and job matches in order to attract more users in an online advertisement system. Amongst the data mining techniques used in this work for prediction of CTR (apply rate), the random forest classifier accuracy was 94.39%, which was higher than the

accuracy values of 93.97% for the decision tree classifier and 93.15% for the K-Nearest Neighbour (KNN) classifier.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Ayush Goyal, and my committee members, Dr. David Hicks, Dr. George Toscano, for their guidance and support throughout the course of this research. Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University-Kingsville a great experience. Finally, thanks to my mother and father for their encouragement.

The author would like to thank the contributor Animesh on Kaggle for providing the dataset used in this study: https://www.kaggle.com/animeshgoyal9/predict-click-through-rate-ctr-for-a-website/version/1 [1].

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Over the past few decades, online job portals and job listings are growing because of the success of machine learning and data science techniques. With the speedy development of internet and web applications, online job postings have become one of the most triumphant business models in the world. Due to this, it has become crucial for job search engines to take into consideration Click Through Rate (CTR) as a performance metric in order to improve not only the company revenue by attracting more users but also, to provide the best job listings based on search queries. In a pay-per-click model, a website only gets paid when a user applies to that job and visits the job advertisement company. Thus, CTR plays a vital and important role in analysing how a certain job posting is attracting users. Inappropriate job listings can lead to a bad user experience and may impact visitors.

In recent years, prediction of CTR for online job listings has attracted widespread attention from researchers in industry and academia. Researchers have proposed many data mining techniques and machine learning models. These models can be classified as linear models, nonlinear models, and fusion models.

In the proposed research in this work, three different machine learning models, viz. decision tree, K-nearest neighbour (KNN), and random forest will be used. This project will be to make predictions for the Apply Rate for a job with a dataset consisting of seven different features. These features provide information about the matching job title, searched query, matched job description, popularity of the job, the location provided by user, and the time period of the job listing posted. This project will analyse and compare the findings and discuss the results obtained through prediction, along with discussion of the accuracy of all three models using different performance

metrics, i.e. classification accuracy, confusion matrix, logarithmic loss, and the area under receiver operating curve (AUC / ROC). It will analyse how the accuracy can be improved by tweaking certain features in the data.

The models will be trained with the help of a popular open-source machine learning framework i.e., scikit-learn. Anaconda and Jupyter Notebook will be used to build the models and to simulate the entire system with the help of a web application built using ReactJS for the front-end and Flask server for the back-end. ReactJS is an open-source framework to build SPAs (single page applications) using component-based HTML DOM (document object model). Flask is a back-end web development framework developed in Python that provides a lightweight server, which handles REST (representational state transfer) calls. Also, this project discusses how the accuracy of predictions for three machine learning models can be increased by taking into consideration the *"position_class_id"* column as a feature to train and test the machine learning models again.

CHAPTER II

LITERATURE REVIEW

In the past decades, the application of data mining techniques in the job and advertisement industry has taken a huge turn with some reassuring advancements. The papers discussed below provide a rundown of applications of different data mining techniques to improve the click through rate by providing the best suited recommendations for job portals in the advertisement industry.

In Gupta 2009 [2], three machine learning algorithms were applied to predict the click through rate for job listings. The machine learning (ML) techniques used in the paper are linear regression, SMO (sequential minimal optimization) regression also known as SVM (support vector machine) regression, and gradient boosted decision tree. In this paper, the procedure of each of the three ML techniques was described and performance juxtaposition of these techniques was carried out using different features. The result of the comparison of all the three techniques was that SMO regression increased the accuracy gain slightly. However, the gradient based decision tree had the highest probability. Furthermore, this researcher also discussed that these models can be further improved by incorporating more features into the dataset [2].

Richardson et al. 2007 [3] proposed a logistic regression model and a boosted regression model to predict the click through rate for an advertisement on the data gathered from the Microsoft web search engine. The result of prediction using the boosted regression model had no significant improvement over the logistic regression model. Also, the logistic regression model was easy to train and achieved 30% reduction in the error [3].

Ma et al. 2013 [4] discussed that the click through rate (CTR) is influenced by the page information and proposed a logistic regression model. By choosing a maximum posterior probability (MAP) estimation rule to compute coefficients of the model, there was a significant

13.15% improvement in the CTR. Rather than calculating the accuracy of the model, researchers focused on the accuracy of the improved model by visualizing Receiver Operating Characteristics (ROC) and Area under the Curve (AUC) [4].

Gupta and Garg 2014 [5] proposed a job recommendation system for candidate job preferences using the application of the decision tree algorithm. The proposed system primarily focuses on providing recommendations on jobs which are best suited according to the candidate profile. In the research, Python was used as the implementation language, along with the Orange data mining library and the results were formulated for CBRS (Content Based Recommender System) and CFRS (Collaborative Based Recommender System) with a classification accuracy of 72% [5].

In other related research papers [6-11], several different methods were discussed to estimate CTR and how predictions on CTR can be made using field aware neural factorization machines, deep learning techniques, and other algorithms that were discussed.

Another set of research papers [12-17] discusses a survey of techniques used in job recommendation systems and how these systems can be improved to provide improved search results in order to increase CTR and increase revenue.

Most of the research in the aforementioned papers discusses results that are based on a linear logistic regression model. However, in the proposed research of this work, three major classification and regression models, viz. decision tree, random forest, and K-Nearest Neighbor (KNN) will be applied for predicting the apply rate for a job posting. The classification accuracy, logarithmic loss, confusion matrix, area under receiver operating curve (AUC ROC) for the three different models will be computed as the performance metrics. The model with the best performance metrics will be evaluated as the one best suited for the prediction of the apply rate for a job.

CHAPTER III

METHODOLOGY

3.1 System Design

The system will consist of the different modules, as shown in Fig. 1.

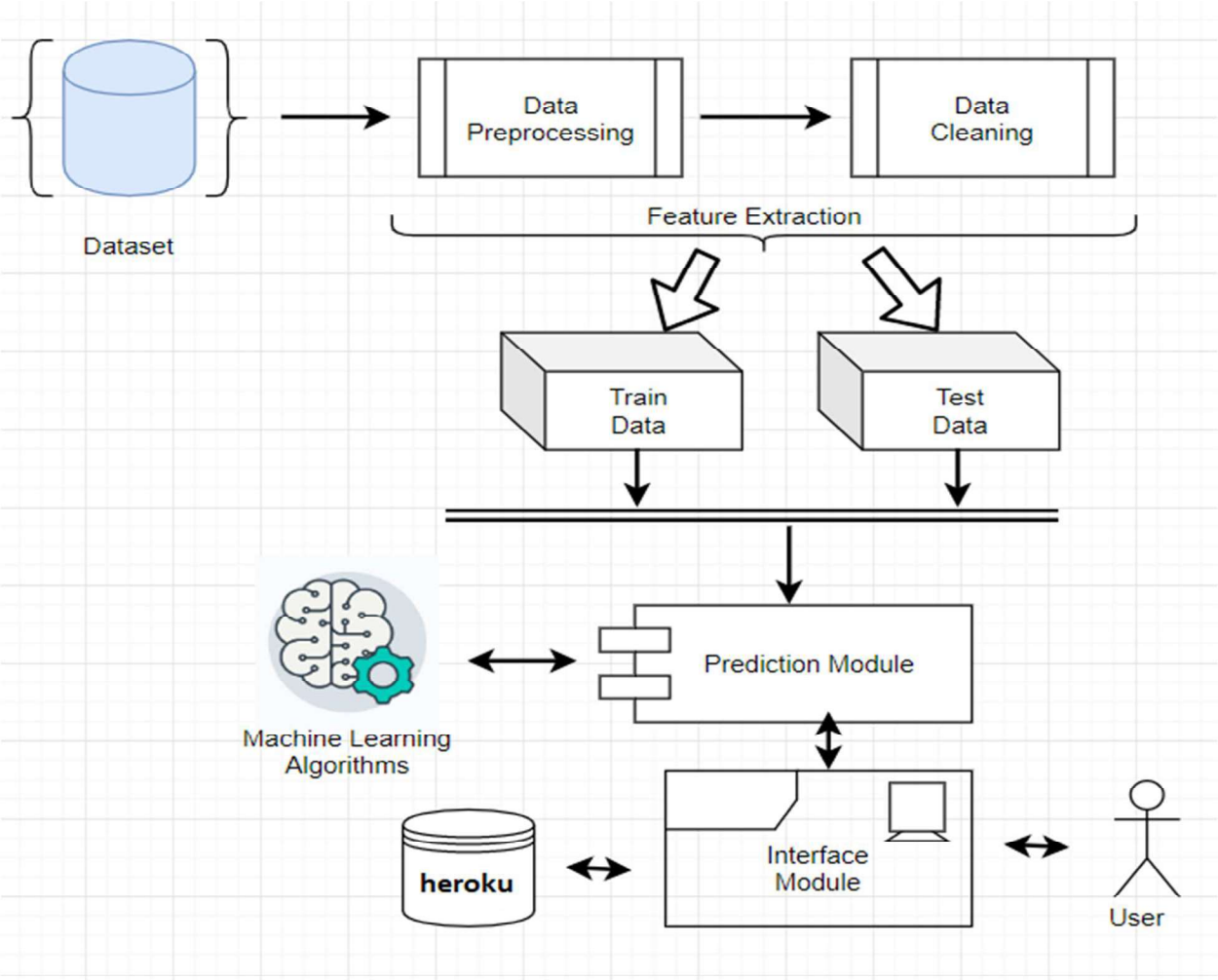

Fig. 1. System architecture.

3.1.1 Data pre-processing module

The dataset consists of 10 different attributes, which are as follows:

1) *title_propinquity_tf_idf*: estimates the value of the proximity of the search question and position heading.

2) *description_propinquity_tf*: estimates the value of the proximity of the search question and position explanation.
3) *main_question_tf*: estimates score related to a user search question, which provides the proximity of the position heading and position description.
4) *question_score*: estimates the popularity of a search question and position listing pair.
5) *query_heading_score*: estimates the popularity of a search question and position heading pair.
6) *location_match*: specifies whether the position listing matches the location of the user.
7) *position_age_days*: specifies the number of days that the job listing has been posted.
8) *Apply*: specifies if the user has applied for this position listing.
9) *search_date*: start date of action.
10) *position_class_id*: specifies the unique id for that position.

Data pre-processing is the first and the most crucial step involved in analysing the dataset to understand what useful information is described in the dataset. Data pre-processing involves gathering data from the internet on open-source machine learning repositories or open-source communities, such as GitHub, etc. The data may consist of CSV (comma separated values), XLS (excel sheets), image files (JPG, JPEG, PNG), etc. In this project, the dataset is a CSV (comma separated values) file collected from Kaggle. Kaggle is an online community of machine learning engineers and data scientists. The dataset in this project consist of 1200890 rows × 10 columns, i.e. 12008900 samples [1].

By performing serialization on the dataset, data pre-processing reconstructs the dataset in such a way that, it can be easily fed to a machine learning model and the required results of the predictions can be obtained. Data pre-processing boosts up the accuracy and performance of

machine learning algorithms and data mining techniques significantly by neglecting noisy data, unpredictable variables, attributes, and NaN (Not a Number) values.

Pandas is an open-source data visualisation, data manipulation and data analysis library available in Python. Pandas provides a quick way to open and visualize the dataset in the Anaconda Jupyter notebook. It also provides different functions to partition the dataset into a data frame object. Pandas is used in this project to split the dataset into data frame objects. Further, the data frame objects can later be split or flattened into small data frames or arrays according to the features, which contain dependent variables and independent variables. The data pre-processing module utilizes the function *pd.read_csv('path')* provided by Pandas. This function is used to load and open the dataset in the Jupyter notebook. This function takes a path as an argument and then, looks for the dataset file in the given path.

3.1.2 Data cleaning module

A dataset gathered from repositories may contain null values or NaN (Not a Number) values, dirty data, and noisy data. Dirty data means incomplete, inconsistent, or inaccurate data, that could have missing values or be in the wrong data type. Noisy data means corrupted or garbage data. If that data is not handled properly before feeding the dataset to a machine learning model, the predictions generated by the machine learning model may be incorrect to a certain extent. It may also result in underfitting or overfitting the model in certain cases. Also, if the noisy data is not removed and NaN (Not a Number) values are not handled correctly, it significantly impacts the training accuracy, testing accuracy, and speed for training the machine learning model. In the dataset, there are 252,571 missing values in the "*title_propinquity_tf_idf*" column, 252,571 missing values in the "*description_propinquity_tf*" column, and 256,555 missing values in the "*location_match*" column, with a total of 761,697 missing values.

The data cleaning or data cleansing module handles the missing values in the dataset by overwriting the missing values for that column with the mean or average value of that column. It is always the best practice to rewrite the missing value with the average or mean value of that column instead of overwriting them with 0. Ignoring the missing values is not recommended as it affects the data quality of the input data frame used in training the machine learning model.

The data cleaning module makes use of the function *df.fillna(df.mean())* from Pandas. This function fills all the NaN (Not a Number) values in that column with the average value of that column. Here, *df.mean()* function calculates the average value of that column and passes that value as an argument to the *df.fillna()* function. And *df* refers to the data frame, which contains the loaded dataset.

3.1.3 Feature extraction module

Feature extraction is the process of selecting and extracting the necessary features from the dataset. Feature extraction is a dimensionality reduction technique in which unnecessary variables and attributes are removed from the dataset. After data pre-processing and data cleaning techniques, the feature extraction technique is applied on the dataset. After feature extraction, the resulting dataset is smaller in size in comparison to the original dataset.

Only the necessary set of features are extracted from the original dataset in order to improve the efficiency of the trained model. Features of a dataset are classified as a list of independent variables (inputs) and a list of dependent variables (outputs). The dataset used in this project contains the following set of independent variables: *1) "title_propinquity_tf_idf", 2) "description_propinquity_tf", 3) "main_question_tf", 4) "question_score", 5) "query_heading_score", 6) "location_match", 7) "position_age_days", 8) "search_date", and 9) "position_class_id".* The only dependent variable is *"Apply"*.

The *"search_date"* column represents the date when a job search was conducted and the other values were collected. The data is collected between the dates 01/21/2018 and 01/27/2018. In this research, prediction of the *"Apply"* column and analysis for the *"Apply_rate"* dataset is done in two separate parts, described below. Also, the data between the dates 01/21/2018 to 01/26/2018 is used as training data and predictions for *"Apply"* column are made for date 01/27/2018 for testing the model. The data frame object of the test data contains the data belonging to the date 01/27/2018. Pandas provides a dataframe.drop() function which is used to remove the unnecessary columns from the data frame object. This function takes in the column name as an argument and removes the entire column from the data frame object.

In the first part of the analysis, *"title_propinquity_tf_idf", "description_propinquity_tf", "main_question_tf", "question_score", "query_heading_score", "location_match", "position_age_days", and "position_class_id"* are considered as the independent variables or the set of suitable features for prediction of the *"Apply"* column. The *"search_date"* column is used to slice up the dataset into a data frame which contains data from 01/21/2018 to 01/26/2018, which is used as the training dataset. The data frame is thus split into two separate data frames, the training data frame and the testing data frame. Here, in the Python code, the equal ("==") operator was used to slice the dataset based on the *"search_date"* column. After this, the two columns in the data frame object - *"position_class_id"* and *"search_date"* become redundant and are dropped. The resulting data frame is now properly structured, organized, errorless and ready to be fed to a machine learning or data mining technique for prediction.

In the second part of the analysis, all the features that were taken into consideration in the first part were considered again along with the last column *"position_class_id"*. Considering *"position_class_id"* allows filtering out rows of the dataset that do not contain data belonging to

a certain specific position of a selected job posting. In this way, data is extracted out only from the rows of the dataset that belong to the job posting into a separate data frame. This data frame is comparatively much smaller to the data frame obtained in the first analysis. Training the machine learning model on only the specific data related to a certain *"position_class_id"* guarantees an increase in efficiency in training and testing the model, along with an increase in accuracy of predictions.

Finally, outputs of predictions and accuracy of machine learning models can be observed and evaluated by four different performance metrics, viz. classification accuracy, confusion matrix, area under receiver operating curve (AUC / ROC), and logarithmic loss.

3.1.4 Prediction module

Once the data is pre-processed, it is stored in a cloud environment, and machine learning algorithms are implemented on the data. The prediction module will use three different machine learning algorithms to predict if a user will apply to that job or not, i.e., "1" or "0", respectively. Finally, a relative analysis of four different performance metrics will be performed on the predictions made by the three machine learning algorithms:

i) Decision Tree.

ii) Random Forest.

iii) K-Nearest Neighbour (KNN).

3.1.4.1 Decision tree

Decision tree is a powerful and popular supervised machine learning algorithm frequently used for prediction and classification. The decision tree algorithm uses an if-else condition rule to make a prediction, utilizing a flow diagram structure in which a node denotes a test on an attribute and each branch denotes an outcome of the test. In the field of computer science, a binary tree is a

fundamental data structure with a variety of applications in storing and retrieving data in an efficient manner, implementation of priority queues, etc. The decision tree algorithm constructs a binary decision tree or binary decision forest by applying the divide and conquer strategy and making use of heuristic functions such as ID3 (Iterative Dichotomiser 3) and C4.5 (an extension of the ID3 algorithm) on the given input dataset.

Data items are classified into a limited number of predefined classes by the decision tree. The root node in the decision tree consists of the attribute which is most relevant to the output variable. To classify a dataset, the decision tree algorithm asks a set of binary questions and traverses through the decision tree based on the answers. As traversal meets the stopping criterion, the data item is considered as classified.

To understand how a decision tree classifies the input data, consider Fig. 2, which depicts a decision tree for an example problem called *playTennis*. The problem of interest here is to predict whether the weather conditions are suitable enough to play tennis or not. The following are the set of rules generated by the decision tree:

i) If the outlook is overcast, then play tennis.

ii) If the outlook is rainy and there is wind, then check if wind is strong or weak.

   a. If wind is weak, play tennis.

   b. If wind is strong, then don't play tennis.

iii) If the outlook is sunny, then check for humidity.

   a. If humidity is high, then don't play tennis.

   b. If humidity is normal, then play tennis.

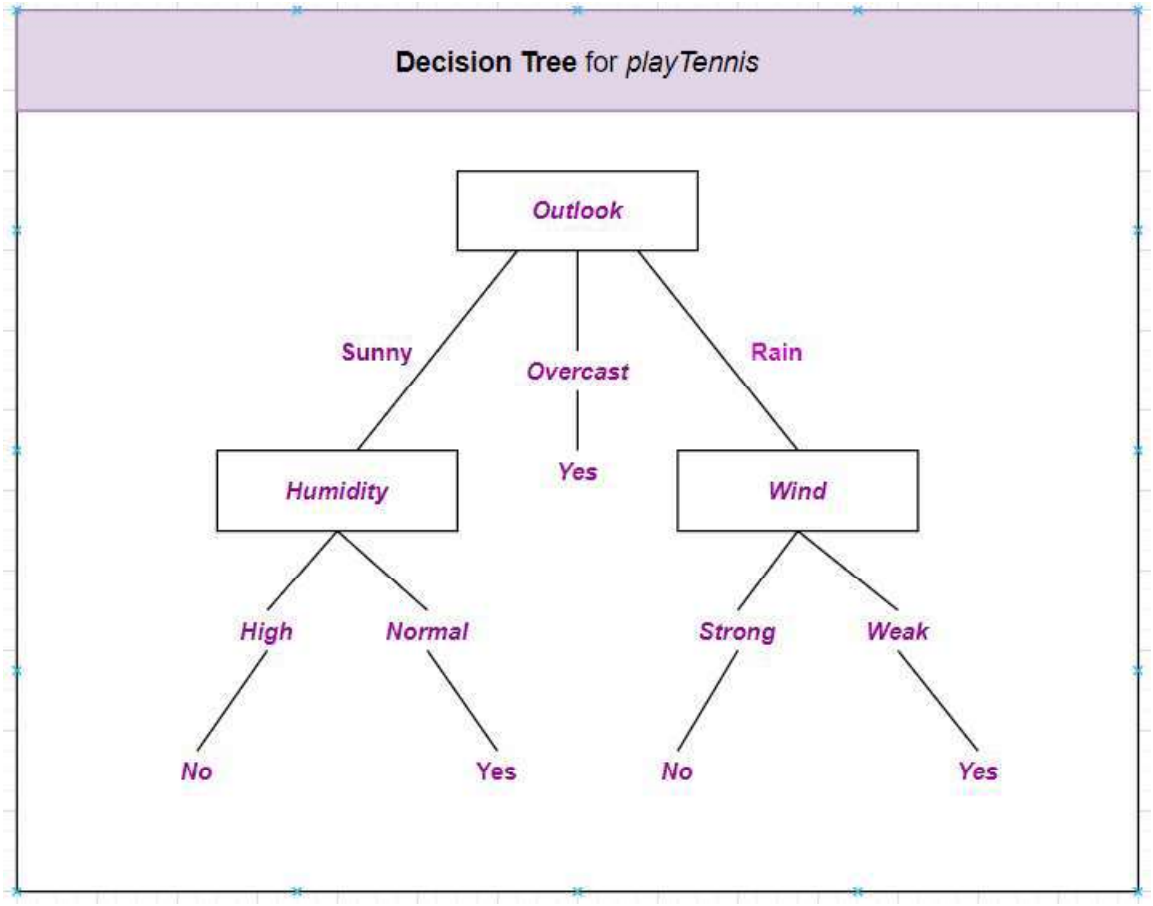

Fig. 2. Famous play tennis problem as a decision tree.

Scikit-learn is an open-source machine learning library and framework that provides the *"DecisionTreeClassifier"* class. This class can be instantiated to form a decision tree classifier object. The decision tree object provides two methods, viz., fit and predict. The fit method takes in two arguments. The first argument is a data frame containing the columns of the independent or input variables. The second argument is the array of the dependent or output variable. The independent (input) and dependent (output) variables used here as the input arguments in the fit method will be from the training dataset. Using these two arguments, the fit method fits a decision tree classifier to the training data to construct the decision tree object. The predict method takes the data frame containing only the columns of the independent variables and classifies this input data. This independent variable data frame given as the input argument to the predict method will

be from the testing dataset. The training dataset is used to train the decision tree classifier object using the fit method and predictions are made on the testing dataset using the predict method. Once the predictions are made, the generated decision tree object classifier can be evaluated by comparing the predictions with actual values. Decision trees are easy to build, implement, and deploy.

3.1.4.2 Random forest

Random forest or random decision forest is a type of supervised learning algorithm which uses an ensemble learning method for classification. Random forest is an extended version of a decision tree. Instead of constructing a single decision tree, it constructs multiple decision trees and conducts a voting among them to find the best-suited result. Better predictive performance can be achieved by the use of ensemble methods, rather than the use of a single decision tree.

Certain number of classification trees get fit to a dataset by the random forest method, and then predictions are combined and compared to generate the best result. Constructing multiple decision trees with big samples of data results in more stable classification. Trees are constructed in a top-down approach and each tree is fully grown and used for classification. All trees constructed using the random forest have the same distribution. The process of constructing a tree involves extracting a bootstrap sample from an n number of observations in the training dataset. The rest of the observations, i.e. p observations are used to estimate the error of every single tree constructed by the random forest in the testing phase. As the name suggests, k random variables are chosen as a decision at a certain node.

Random forests can handle missing data as well. Whenever the random forest encounters a missing value, the algorithm finds the median value for that column and substitutes the missing value with the median value. Fig. 3 shows the depiction of trees constructed by a random forest.

The following section gives the advantages and disadvantages of the random forest method over decision trees.

The advantages of the random forest method over decision trees are as follows:

i) Random forest works really well as compared to decision tree in regards to a large set of data.

ii) Improved accuracy as compared to a single decision tree.

iii) Trees constructed by random forest can be saved for future use on other data.

iv) Overfitting problem is handled by averaging the prediction results of different decision trees.

The disadvantages of the random forest method over decision trees are as follows:

i) Time complexity is one main disadvantage of random forests.

ii) Large computational power is required to implement random forest.

iii) Time taken for prediction is high in comparison with decision trees.

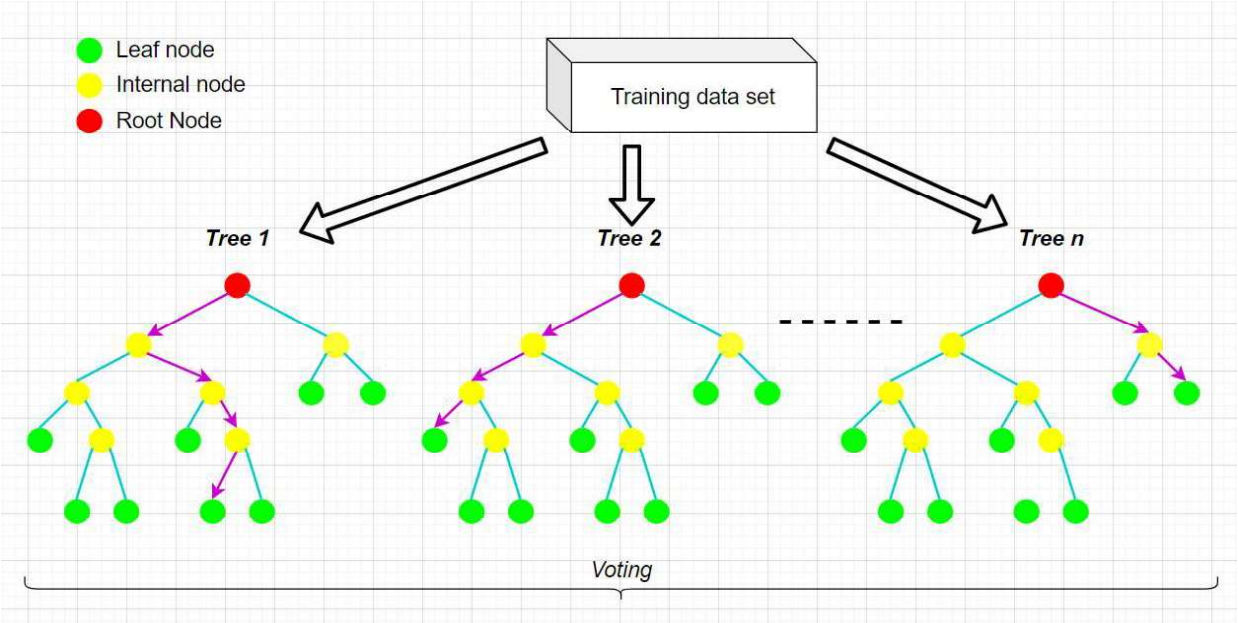

Fig. 3. Depiction of trees generated by a random forest.

Scikit-learn provides a random forest classifier class interface which can be used to create a random forest classifier object. This object is used to construct *N* number of trees trained using the input training dataset. The trees are tested on testing data and their performance can be evaluated using a performance matrix, containing the specificity, confusion matrix, etc.

3.1.4.3 K-nearest neighbour (KNN)

K-nearest neighbour (KNN) is one of the simplest supervised techniques in machine learning and data mining, commonly used in classification problems. KNN classifies data points on the basis of how its neighbours are classified. The number *k* is a parameter that denotes the number of nearest neighbours needed to be considered during the classification. For example, if $k = 5$, then the data point is classified by a majority of votes from 5 nearest neighbours.

In order to choose the correct value for *k*, KNN uses a process of parameter tuning for better accuracy of classification. The KNN algorithm uses the concept of feature similarity. The accuracy of KNN depends on the value of *k*. If the value for *k* selected is too low, then the KNN algorithm considers that data is too noisy, i.e. data points are too close to each other. If the value for *k* is selected too high, then the KNN algorithm becomes time consuming to process all the neighbouring data points and the algorithm might run into processing and resource issues. There are two different methods for selecting the value of *k*. The first and very common method is to select the value for *k* by taking the square root of the total number of data points. The second method suggests that the value for *k* should be odd to avoid indecision between the two classes of data. The KNN algorithm can be effectively applied when:

i) The data is labelled.

ii) Data is noise free.

iii) Dataset is small and precise.

iv) The output variable value consists of only a few classes, such as two or three.

Fig. 4 depicts the steps involved in the KNN algorithm.

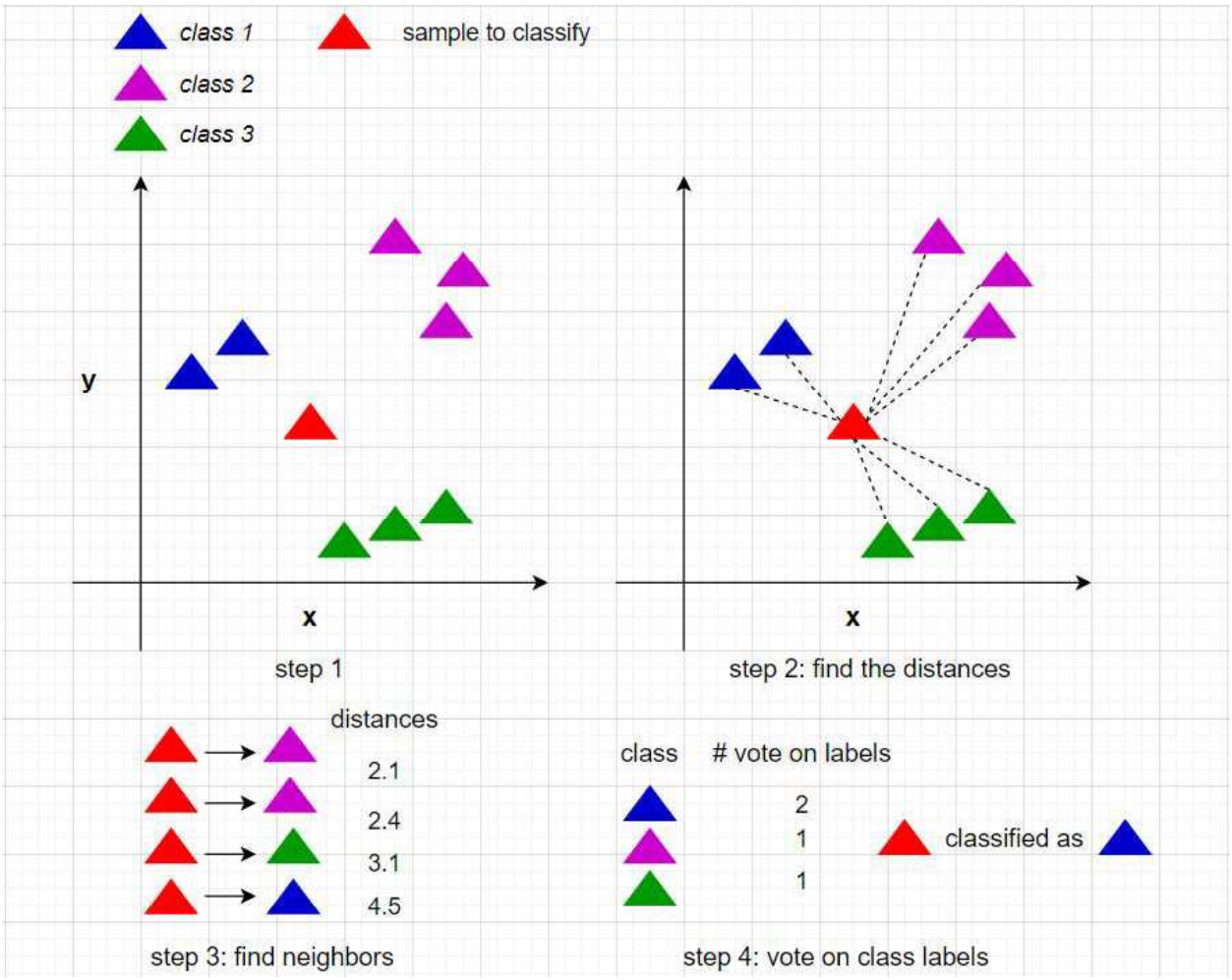

Fig. 4. Steps involved in the KNN algorithm.

To find the nearest neighbours, three different distance functions can be used. The distance functions are as follows:

i) Euclidean distance –

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{1}$$

In Eq. (1), (x, y) is the position of a data point in the 2D plane.

ii) Manhattan distance –

$$\sum_{i=1}^{k} |x_i - y_i| \tag{2}$$

In Eq. (2), (x, y) is the position of a data point in the 2D plane.

iii) Minkowski distance –

$$\left(\sum_{i=1}^{k} (|x_i - y_i|)^q \right)^{\frac{1}{q}} \tag{3}$$

In Eq. (3), q is a positive integer and (x, y) is the position of a data point in the 2D plane.

3.1.5 Interface module

The interface module is a web application which serves as the interface to the system and shows the predictions based on the dataset provided to the system. The web application is a SPA (Single Page Application), a simple web application designed using ReactJS for the frontend and Flask for the backend. The interface module consists of a login page and a dashboard page. The login page contains a web form for authentication. The dashboard page contains a web form where, a user can put values of features related to the dataset and can observe the prediction as a result.

CHAPTER IV

IMPLEMENTATION

The implementation of this project consists of a frontend web interface developed with ReactJS and a backend server developed using Flask.

4.1 Web Interface

The web interface follows a client-server architecture and consists of two pages, i.e. a login page (see Figure 5) and a dashboard page (see Figure 6). Both of the pages are designed using ReactJS and Bootstrap 4. ReactJS is a front-end framework designed by Facebook, which follows component-based design patterns for modern web applications. Bootstrap 4 is a front-end user interface framework developed for providing a better user experience. The login page consists of a web form where a user enters the login details. Once the user clicks on the login button, the login page sends the login details to an API (Application Programming Interface) endpoint with the "/login" using an HTTP (Hyper Text Transfer Protocol) request. If the login details are valid, the user gets redirected to the dashboard page.

The dashboard page consists of a web form where the user will enter all the values of features and can observe the predictions made by the machine learning algorithm. Figures 5-8 show the interface of the system. Once the user clicks the predict button, all the form values get sent over to the Flask back-end server at an API endpoint "/api/predict" and results are received as an HTTP response in JSON (JavaScript Object Notation) format. Results are rendered on the webpage in the results section of the dashboard page (see Figures 7-8).

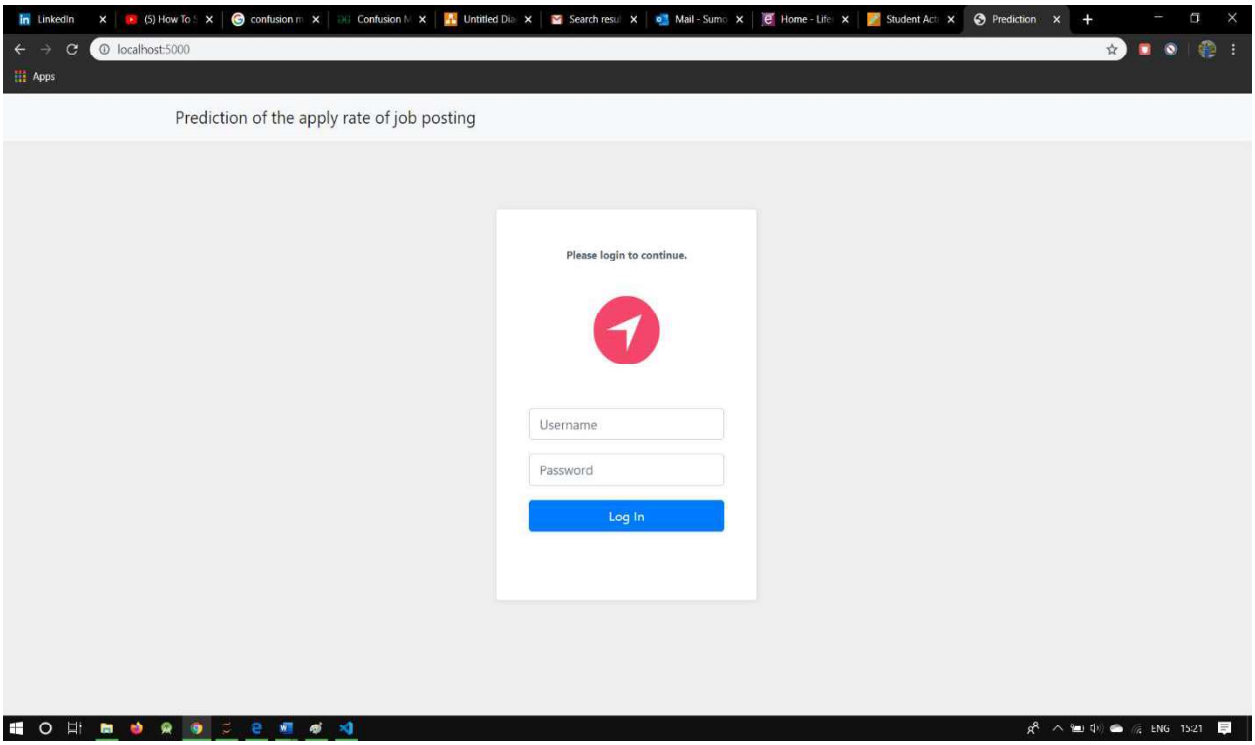

Fig. 5. Login page.

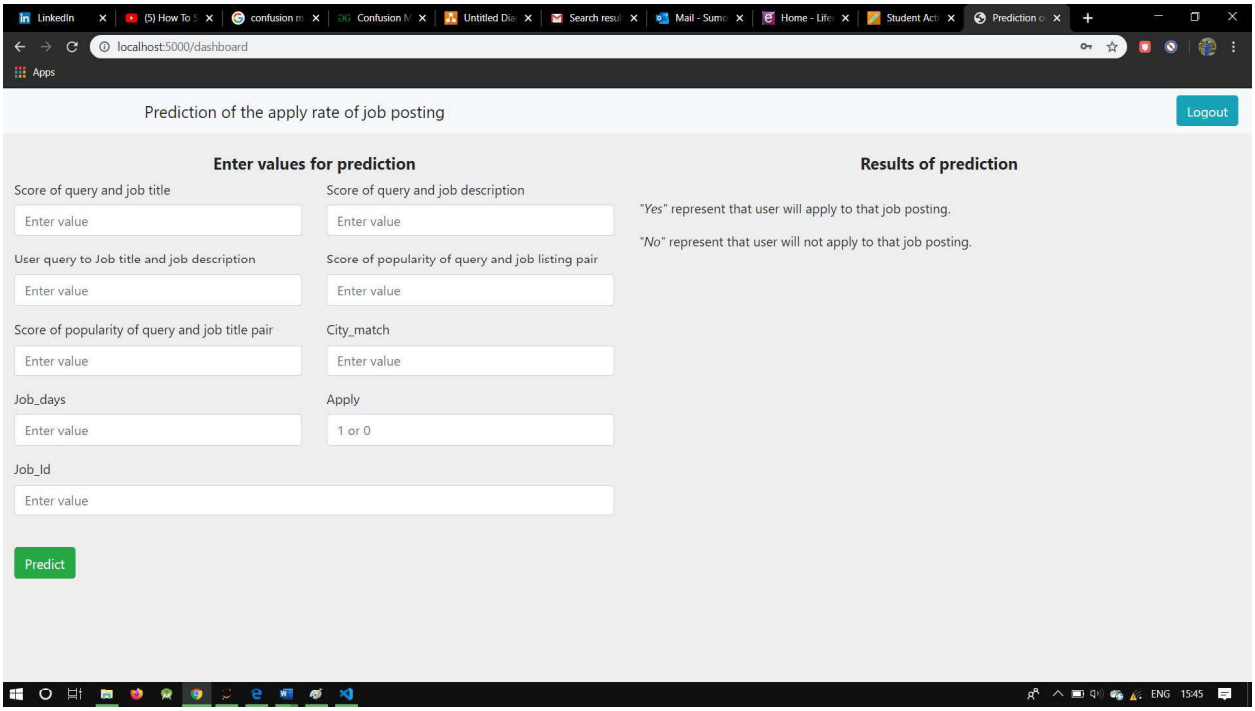

Fig. 6. Dashboard page.

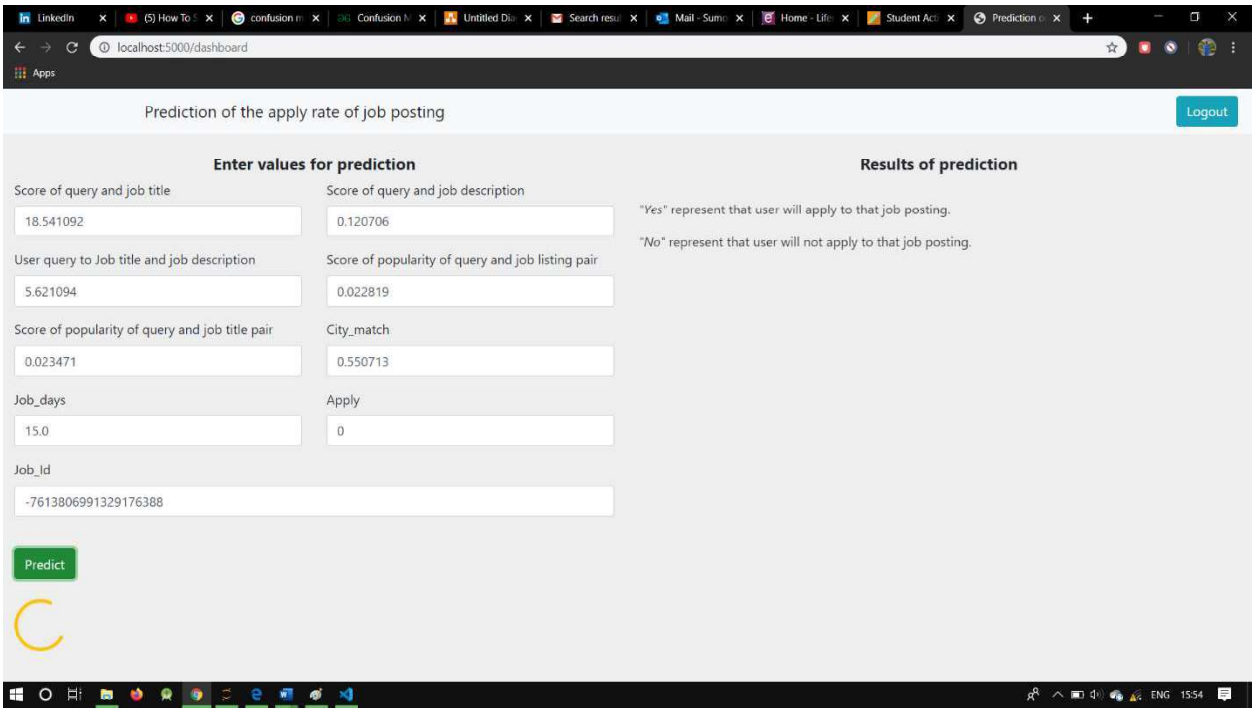

Fig. 7. Flask back-end server processing.

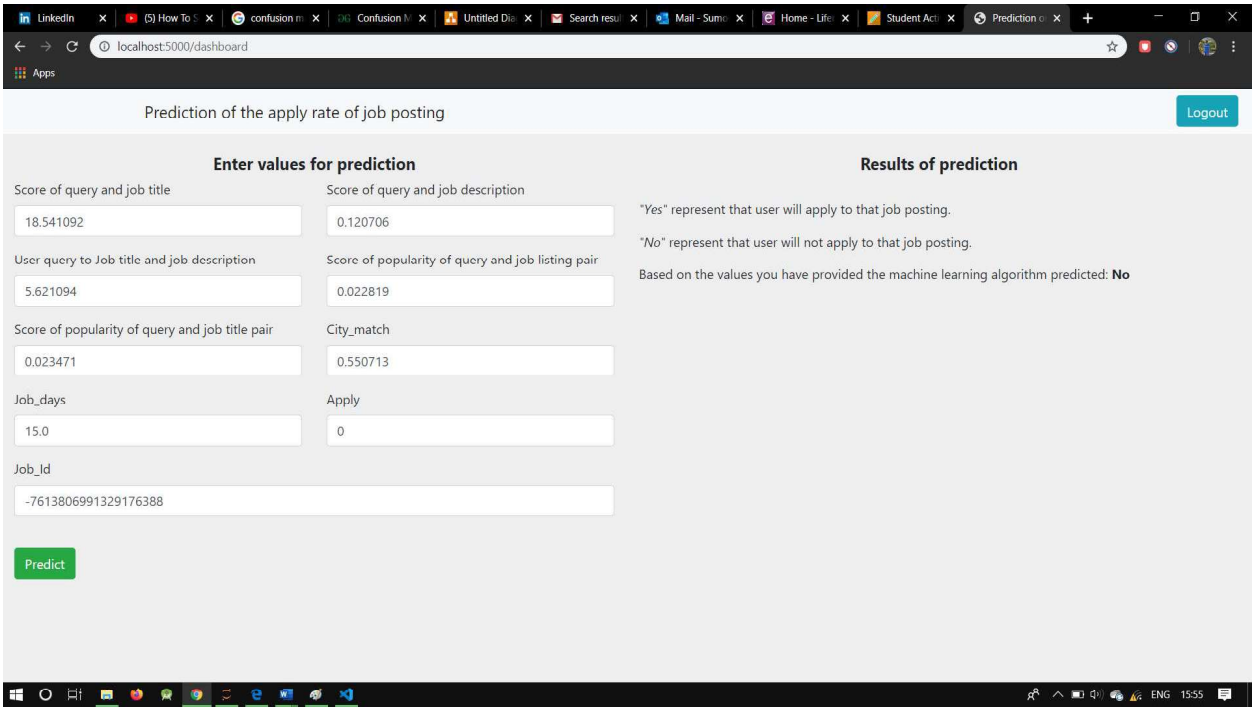

Fig. 8. Results obtained from Flask back-end server.

4.2 Flask Back-end Server

Flask is a back-end web development framework used by Python developers. Flask provides a light-weight server for testing API (Application Programming Interface) endpoints. Flask back-end provides routes for different URLs. In this project, three URL routes are configured using Flask. Accessing "/login" routes redirects the browser to the login page. The "/dashboard" page is a protected route and can only be accessed if the valid session variable is set. If the session is valid, the Flask back-end server redirects the browser to the dashboard page.

The "/api/predict" route is an API endpoint and serves the machine learning model as an API. This route listens to HTTP (Hyper Text Transfer Protocol) POST requests and returns a response of predictions in the JSON (JavaScript Object Notation) format.

CHAPTER V

RESULTS AND DISCUSSION

The research analysis for this project has been done in two separate parts. In the first part, the results of predictions of three machine learning algorithms are computed on the *"Apply_rate"* dataset without taking the *"position_class_id"* column into consideration in the input training dataset. Results of predictions have been computed using four different performance metrics, viz., accuracy score, confusion matrix, area under receiver operating curve (AUC ROC), and logarithmic loss, to evaluate each machine learning model.

The second part of the research analysis for this project demonstrates how accuracy for all three machine learning models was improved by using the *"position_class_id"* in the input training dataset.

The next section describes the performance metrics used to evaluate each machine learning model in terms of how well it can make the predictions.

5.1 Performance Metrics

5.1.1 Classification accuracy

Classification accuracy is the number of correct predictions made by the model over all kinds of predictions made, as shown in Eq. (4) below:

$$\text{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions\ made} \tag{4}$$

5.1.2 Confusion matrix

The confusion matrix or error matrix is a matrix which describes the performance of a classifier on input test data. The rows in the confusion matrix correspond to what the machine learning algorithm predicted and columns in the confusion matrix correspond to the actual values of the input test data. Figure 9 shows the confusion matrix.

The confusion matrix depicts the different ways in which the machine learning algorithm gets "confused" when it makes predictions. The following are terms used in the confusion matrix:

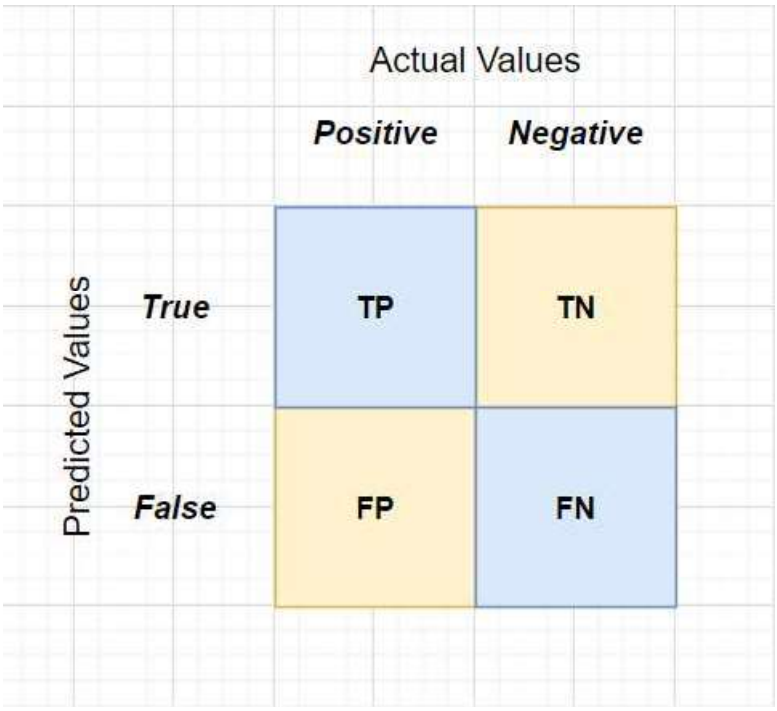

Fig. 9. Confusion Matrix.

i) True positive (TP) – Correct value is positive and predicted value is positive.

ii) True negative (TN) – Correct value is negative and predicted value is negative.

iii) False positive (FP) – Correct value is negative and predicted value is positive.

iv) False negative (FN) – Correct value is positive and predicted value is negative.

5.1.3 Area under the curve

Area under the curve (AUC) measures the logistic model's degree of effective classification. The AUC makes it easier to contrast one model with another's ROC curve.

5.1.4 Logarithmic loss

The logarithmic loss or loss of log penalizes the incorrect classifications. This fits well for the grouping of multiple classes. The classifier will assign a likelihood for all samples to each class when operating with log loss. Suppose, there are *N* samples belonging to *M* classes, then the log loss is calculated as below in Eq. (5):

$$\frac{-1}{N} \sum_{\substack{0 \le i \le n \\ 0<j<m}} y_{ij^*} \log\left(p(i,j)\right) \tag{5}$$

In Eq. (5), y_{ij} indicates whether sample i belongs to class j or not and P_{ij} indicates the probability of sample i belonging to class j.

5.2 Decision Tree Results

5.2.1 Decision tree classification accuracy

Table 1 and Table 2 show the classification accuracies for the training dataset and testing dataset for the decision tree classifier.

Table 1. Classification accuracy of the decision tree classifier omitting *"position_class_id"*.

Decision Tree Classifier Classification Accuracy (exact match subset accuracy)	
Training Dataset	91.44 %
Test Dataset	90.47 %

Table 2. Classification accuracy of the decision tree classifier including *"position_class_id".*

Decision Tree Classifier Classification Accuracy (exact match subset accuracy)	
Training Dataset	91.44 %
Test Dataset	93.97 %

As can be observed in Table 2, the accuracy for the testing dataset is improved by 3% when including the *"position_class_id"* as a feature in the input or independent variable data frame. Classification accuracy for the testing dataset is definitely increased, because, by neglecting rows or data not related to the certain specific position id, the overfitting of the classifier can be prevented, resulting in better predictions.

5.2.2 Confusion matrix for decision tree classifier

Figure 10 shows the confusion matrix for the decision tree classifier when omitting the *"position_class_id"* column from the input or independent variable data frame. Figure 11 shows the confusion matrix for the decision tree classifier when including the *"position_class_id"* column in the input or independent variable data frame.

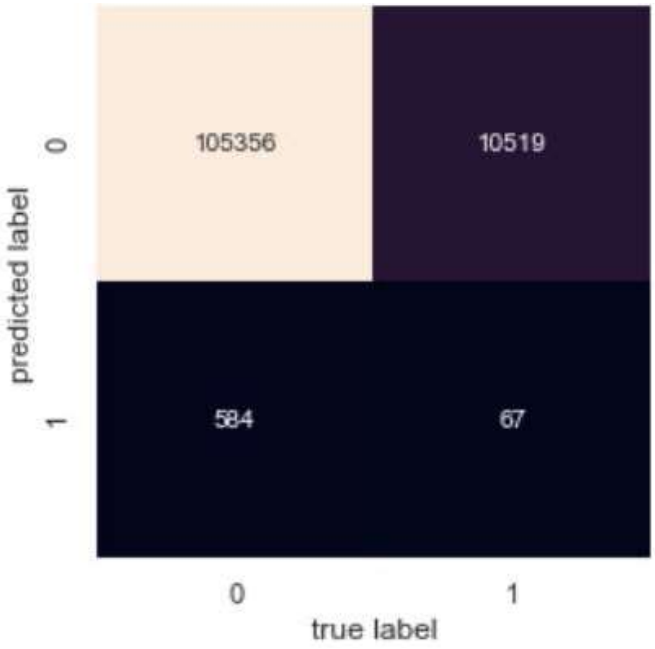

Fig. 10. Decision tree confusion matrix when omitting the *"position_class_id"* column.

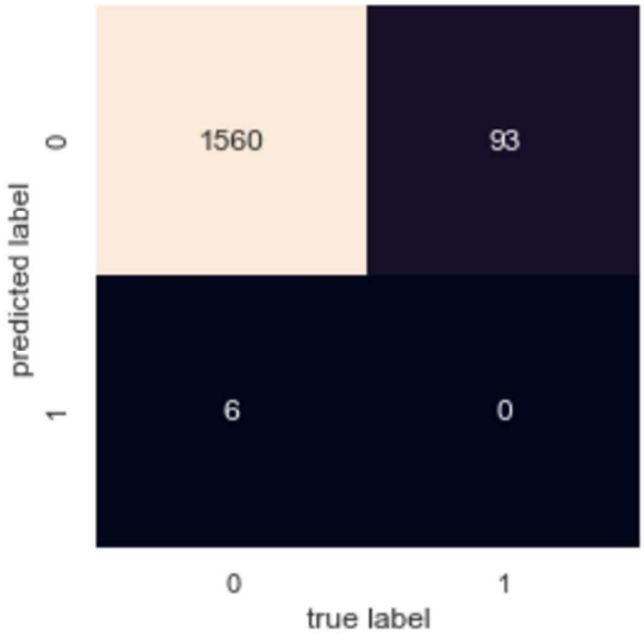

Fig. 11. Decision tree confusion matrix when including the *"position_class_id"* column.

Looking at the confusion matrix for the decision tree without considering *"position_class_id"* as a feature in the training dataset, the decision tree classifier predicted 105,356 TP (True Positive) cases and 67 TN (True Negative) cases with a total of 105,423 correct predictions. A TP (True Positive) case indicates the user did actually apply to that job and the model also predicted that the user will apply to the job. A TN (True Negative) case indicates the user did not actually apply to that job and the model also predicted that the user will not apply to the job. The confusion matrix also indicates that the decision tree classifier incorrectly classified 10,519 FP (False Positive) cases and 584 FN (False Negative) cases with a total of 11,103 incorrect predictions. Thus, the number of correct predictions is much higher than the number of incorrect predictions.

In the confusion matrix obtained for the decision tree considering *"position_class_id"* as a feature for the training dataset, the decision tree predicted 1,560 TP (True Positive) cases and 0 TN (True Negative) cases with a total of 1560 correct cases. The decision tree classifier also predicted 93 FP (False Positive) cases and 6 FN (False Negative) cases with a total of 99 incorrect predictions. Now when considering the *"position_class_id"* as a feature for the training dataset, the number of correct predictions is even higher than the number of incorrect predictions.

5.2.3 Area under receiver operating curve

Table 3 shows the area under the receiver operating curve for the decision tree classifier when it was trained on a dataset without including the *"position_class_id"* column. Table 4 shows the area under the receiver operating curve for the decision tree classifier when it was trained with a dataset including the *"position_class_id"* column.

Table 3. AUC ROC value of the decision tree classifier when omitting *"position_class_id"*.

Area under receiver operating curve	
Train	0.54
Test	0.50

Table 4. AUC ROC value of the decision tree classifier when including *"position_class_id"*.

Area under receiver operating curve	
Train	0.49
Test	0.55

The value for the AUC ROC (area under receiver operating curve) here represents how well the decision tree classifier can distinguish between classes. The AUC value can range between 0 – 1. Here, the decision tree classifier's AUC value is between 0.49 and 0.55. There is a slight improvement of 0.05 in the decision tree classifier's area under the curve for the test data when including the *"position_class_id"*.

5.2.4 Logarithmic loss

Table 5 shows the logarithmic loss of the decision tree classifier when it is trained on a dataset not including the *"position_class_id"* column. Table 6 shows the logarithmic loss of the decision tree classifier when it is trained on a dataset including the *"position_class_id"* column.

Table 5. Logarithmic loss of the decision tree classifier when omitting *"position_class_id"*.

Logarithmic Loss	
Train	2.95
Test	3.28

Table 6. Logarithmic loss of the decision tree classifier when including *"position_class_id"*.

Logarithmic Loss	
Train	2.95
Test	2.08

The logarithmic loss value should be close to 0 for a good classifier (it ranges between 0 - 1). The logarithmic loss for the decision tree classifier without considering *"position_class_id"* for training the classifier is 2.95 for the training data and is 3.28 for the test data. Considering *"position_class_id"* as a feature decreases the logarithmic loss to 2.08 for the test data, which is a significant improvement in accuracy for the decision tree classifier.

5.3 Random Forest Results

5.3.1 Random forest classification accuracy

Table 7 and Table 8 show the classification accuracies for the training dataset and testing datasets for the random forest classifier when omitting *"position_class_id"* and including *"position_class_id"*, respectively. From Table 8, it can be observed that the accuracy for the test dataset is improved by around 4% when including the *"position_class_id"*. In other words, the

classification accuracy for the random forest classifier is thus increased by around 4% when considering *"position_class_id"* as a feature for training the classifier.

Table 7. Classification accuracy of the random forest classifier omitting *"position_class_id".*

Classification Accuracy (exact match subset accuracy)	
Train	91.38 %
Test	90.78 %

Table 8. Classification accuracy of the random forest classifier including *"position_class_id".*

Classification Accuracy (exact match subset accuracy)	
Train	91.38%
Test	94.39 %

5.3.2 Confusion matrix for the random forest

Figure 12 shows the confusion matrix for the random forest classifier when *"Position_id"* is not included as a feature for the training dataset. In this scenario, the random forest classifier predicted 105,749 TP (True Positive) cases and 38 TN (True Negative) cases with a total of 105,787 correct predictions. The confusion matrix also indicates that the random forest classifier incorrectly classified 10,548 FP (False Positive) cases and 191 FN (False Negative) cases with a total of 10,739 incorrect predictions. Figure 13 shows the confusion matrix for the random forest classifier when *"Position_id"* is included as a feature for the training dataset.

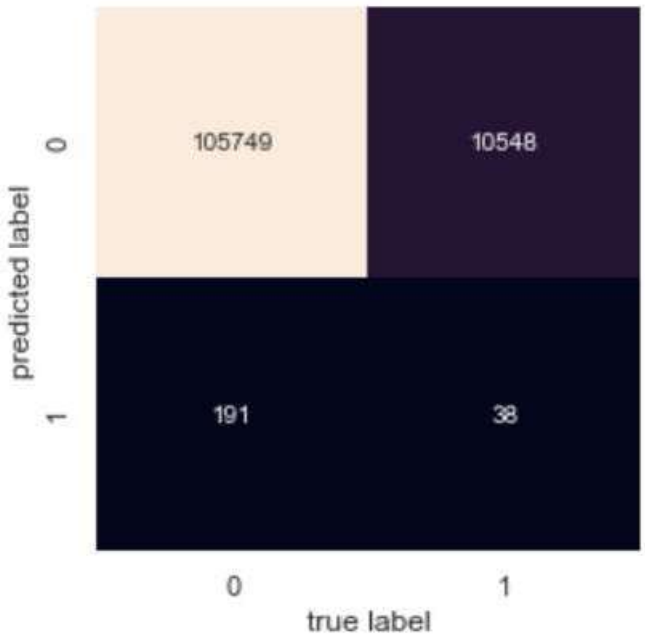

Fig. 12. Random forest confusion matrix when omitting *"position_class_id"* column.

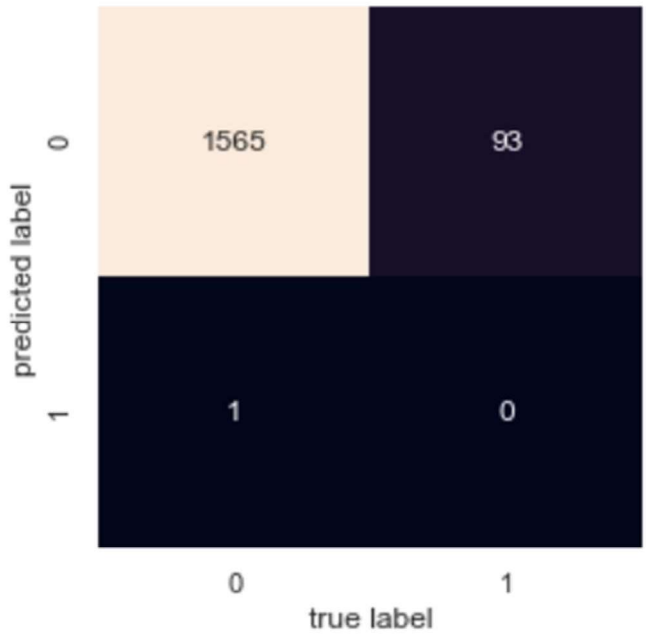

Fig. 13. Random forest confusion matrix when including *"position_class_id"* column.

When including *"position_class_id"* as a feature, the random forest classifier predicted 1565 TP (True Positive) cases and 0 TN (True Negative) cases with a total of 1,565 correct predictions, and 93 FP (False Positive) cases and 1 FN (False Negative) case with a total of 94 false predictions.

5.3.3 Area under receiver operating curve

Table 9 shows the AUC ROC value of the random forest classifier when *"position_class_id"* is not included in the training dataset. Table 10 shows the AUC ROC value of the random forest classifier when *"position_class_id"* is included in the training dataset.

Table 9. AUC ROC value of random forest classifier omitting *"position_class_id".*

Area under receiver operating curve	
Train	0.65
Test	0.50

Table 10. AUC ROC value of random forest classifier including *"position_class_id".*

Area under receiver operating curve	
Train	0.65
Test	0.52

The value for the AUC ROC (area under receiver operating curve) lies between 0.5 and 0.65 for the random forest classifier. There is a slight improvement of 0.02 in the random forest classifier's

AUC ROC value for test data when considering *"position_class_id"* as a feature in the training dataset.

5.3.4 Logarithmic loss

Table 11 shows the logarithmic loss of the random forest classifier when *"position_class_id"* is not included as a feature in the training dataset. Table 12 shows the logarithmic loss of the random forest classifier when *"position_class_id"* is included as a feature in the training dataset.

Table 11. Logarithmic loss of random forest classifier omitting *"position_class_id"*.

Logarithmic Loss	
Train	2.17
Test	3.18

Table 12. Logarithmic loss of random forest classifier including *"position_class_id"*.

Logarithmic Loss	
Train	2.97
Test	1.93

The logarithmic loss for the random forest classifier when the *"position_class_id"* is not included for training the classifier is 2.17 for training data and 3.18 for test data. Including

"position_class_id" as a feature decreases the logarithmic loss to 1.93 for test data, which is a significant improvement in accuracy for the random forest classifier.

5.4 KNN Results

5.4.1 KNN classification accuracy

Table 13 and Table 14 show the classification accuracies for the KNN algorithm for the training dataset and testing dataset when omitting and including the *"position_class_id"* column, respectively. In Table 14, it can be observed that the accuracy for the testing dataset is improved by around 3% when including *"position_class_id"* as a feature. The classification accuracy for KNN is thus improved by around 3% when considering *"position_class_id"* as a feature for training the classifier.

Table 13. Classification accuracy of KNN classifier when omitting *"position_class_id".*

Classification Accuracy (exact match subset accuracy)	
Train	91.09 %
Test	90.59 %

Table 14. Classification accuracy of KNN classifier when including *"position_class_id".*

Classification Accuracy (exact match subset accuracy)	
Train	91.09%
Test	93.15%

5.4.2 KNN confusion matrix

Figure 14 shows the confusion matrix for the KNN algorithm when omitting *"position_class_id"* as a feature for the training dataset. When omitting this column, the KNN classifier predicted 105,522 TP (True Positive) cases and 50 TN (True Negative) cases with a total of 105,572 correct predictions. The confusion matrix also indicates that KNN incorrectly classified 10,536 FP (False Positive) cases and 418 FN (False Negative) cases with a total of 10,954 incorrect predictions. The number of correct predictions is much greater than the incorrect predictions.

Figure 15 shows the confusion matrix obtained for KNN when considering *"position_class_id"* as a feature for the training dataset. When including this column, the KNN algorithm predicted 1,562 TP (True Positive) cases and 0 TN (True Negative) cases with a total of 1,562 correct predictions, and 93 FP (False Positive) cases and 4 FN (False Negative) cases with a total of 97 false predictions. The number of correct predictions is much higher than the number of incorrect predictions when the *"position_class_id"* is included as a feature for the training dataset.

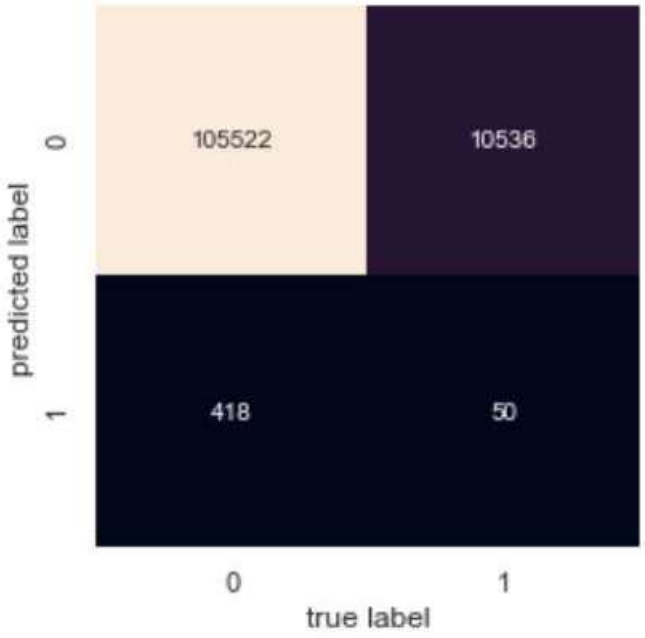

Fig. 14. KNN confusion matrix when omitting *"position_class_id"* column.

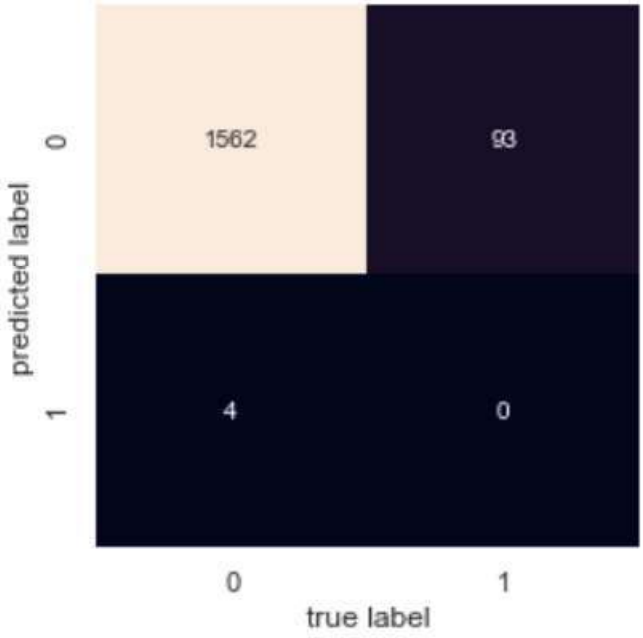

Fig. 15. KNN confusion matrix when including the *"position_class_id"* column.

5.4.3 Area under receiver operating curve

Table 15 shows the AUC ROC value of the KNN classifier when the *"position_class_id"* is not included as a feature in the training dataset. Table 16 shows the AUC ROC value of the KNN classifier when the *"position_class_id"* is included as a feature in the training dataset.

Table 15. AUC ROC value of the KNN classifier when omitting *"position_class_id"*.

Area under receiver operating curve	
Train	0.52
Test	0.50

Table 16. AUC ROC value of the KNN classifier when including *"position_class_id"*.

Area under receiver operating curve	
Train	0.52
Test	0.49

The value for the AUC ROC (area under receiver operating curve) lies between 0.49 and 0.52 for the KNN algorithm. As opposed to the previous algorithms, there is no improvement for the AUC ROC value in the KNN classifier when including *"position_class_id"* as a feature.

5.4.4 Logarithmic loss

Table 17 shows the logarithmic loss of the KNN classifier when *"position_class_id"* is not included as a feature in the training dataset. Table 18 shows the logarithmic loss of the KNN classifier when the *"position_class_id"* is included as a feature in the training dataset.

Table 17. Logarithmic loss of the KNN classifier when omitting *"position_class_id"*.

Logarithmic Loss	
Train	3.07
Test	3.24

Table 18. Logarithmic loss of the KNN classifier when including *"position_class_id"*.

Logarithmic Loss	
Train	3.07
Test	2.02

The logarithmic loss for the KNN classifier when *"position_class_id"* is not included for training the classifier is 3.07 for the training data and 3.24 for the test data. Including *"position_class_id"* as a feature decreases the logarithmic loss to 2.02 for the test data, which is an improvement in accuracy for the KNN algorithm.

5.4 Final Comparison Results

Table 19 shows the final comparison results of the three machine learning methods in this work. As can be observed, the random forest method results in the highest prediction accuracy for the test data both when omitting and when including the *"position_class_id"* as a feature in the training dataset. Also, including the *"position_class_id"* as a feature improves the accuracy of the prediction in all the three methods. So, in this work, it was observed that the best-case scenario is to include the *"position_class_id"* as a feature in the training dataset and use the random forest classifier for making a prediction model.

Table 19. Final comparison results of the classification accuracies (calculated as exact match subset accuracies) for all three machine learning methods.

	Decision Tree	**Random Forest**	**KNN**
Training dataset omitting *"position_class_id"*	91.44 %	91.38 %	91.09 %
Test dataset omitting *"position_class_id"*	90.47 %	90.78 %	90.59 %
Training dataset including *"position_class_id"*	91.44 %	91.38%	91.09%
Test dataset including *"position_class_id"*	93.97 %	94.39 %	93.15%

Table 20 shows the final comparison results of the AUC value for the three machine learning methods in this work. The lower AUC value represents that a greater number of samples had the apply rate of "0" in the training dataset, and a smaller number of samples had the apply rate of "1". Looking at the confusion matrices for the three machine learning algorithms, it can be observed that, in the training dataset, a greater number of samples had an apply rate of "0", and a smaller number of samples had an apply rate of "1". That is the reason the value of AUC, for all three machine learning algorithms, lies between 0.50 to 0.65.

Table 20. Final comparison results of the AUC ROC value for all three machine learning methods.

	Decision Tree	**Random Forest**	**KNN**
Training dataset omitting *"position_class_id"*	0.54	0.65	0.52
Test dataset omitting *"position_class_id"*	0.50	0.50	0.50
Training dataset including *"position_class_id"*	0.49	0.65	0.52
Test dataset including *"position_class_id"*	0.55	0.52	0.49

CHAPTER VI

CONCLUSION AND FUTURE WORK

This project was applying machine learning algorithms for building models to accurately predict the CTR or "Apply Rate" for a job posting. A training dataset containing several features of the job posting was used in training the machine learning models. A web application was built that allowed predicting the "Apply Rate" in real time for a given set of parameters about a job posting. The motivation or application of this work is to facilitate employers or advertisers by allowing them to observe how their job advertisement apply rate can be predicted and improved. Previous research in past published literature has used linear machine learning algorithms for the prediction of the apply rate. However, this project has improved upon the previous work and successfully implemented nonlinear and hierarchical machine learning algorithms and compared their results.

By analyzing the results obtained through four different performance metrics, it was observed that the random forest algorithm performed the best amongst the three methods. Also, from the results it has been observed that the accuracy for all three algorithms can be improved by considering the *"position_class_id"* as a feature in the input training dataset. When including the *"position_class_id"* as a feature, for the test data, the random forest classifier accuracy was 94.39%, which was higher than the accuracy values of 93.97% for the decision tree classifier and 93.15% for the KNN classifier.

The dataset used in this work consists of nine features. However, there are certain features such as salary, visa sponsorship, required experience, etc., that were missing in this dataset. In the future, the accuracy of the machine learning models can be further improved by considering these additional features as well.

REFERENCES

1. Animesh, June 2019, Predict Click Through Rate (CTR) for a Website. https://www.kaggle.com/. https://www.kaggle.com/animeshgoyal9/predict-click-through-rate-ctr-for-a-website/version/1. Accessed 10th April 2020.
2. Gupta, Manish. "Predicting click through rate for job listings." In Proceedings of the 18th international conference on World wide web, pp. 1053-1054. ACM, 2009.
3. Richardson, Matthew, Ewa Dominowska, and Robert Ragno. "Predicting clicks: estimating the click-through rate for new ads." In Proceedings of the 16th international conference on World Wide Web, pp. 521-530. ACM, 2007.
4. Ma, Jing, Xian Chen, Yueming Lu, and Kuo Zhang. "A click-through rate prediction model and its applications to sponsored search advertising." (2013): 500-503.
5. Pan, Zhen, Enhong Chen, Qi Liu, Tong Xu, Haiping Ma, and Hongjie Lin. "Sparse factorization machines for click-through rate prediction." In 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 400-409. IEEE, 2016.
6. International Conference on Data Mining (ICDM), pp. 400-409. IEEE, 2016.

   Regelson, Moira, and D. Fain. "Predicting click-through rate using keyword clusters." In Proceedings of the Second Workshop on Sponsored Search Auctions, vol. 9623, pp. 1-6. 2006.
7. Zhang, Li, Weichen Shen, Jianhang Huang, Shijian Li, and Gang Pan. "Field-Aware Neural Factorization Machine for Click-Through Rate Prediction." IEEE Access 7 (2019): 75032-75040.
8. Gupta, Anika, and Deepak Garg. "Applying data mining techniques in job recommender system for considering candidate job preferences." In 2014 International Conference on

Advances in Computing, Communications and Informatics (ICACCI), pp. 1458-1465. IEEE, 2014.

9. Wang, Qianqian, Shuning Xing, Xiaohui Zhao, and Tianlai Li. "Research on CTR Prediction Based on Deep Learning." IEEE Access 7 (2018): 12779-12789.
10. S. Zheng, W. Hong, N. Zhang, F. Yang, “Job Recommender Systems: A Survey”, 7th International Conference on Computer Science & Education (ICCSE), Melbourne, VIC, pp. 920 – 924, 14-17 Jul. 2012.
11. S. T. Al-Otaibi, M. Ykhlef, “A survey of Job Recommender Systems”, International Journal of the Physical Sciences, Vol. 7 (29), pp. 5127-5142, 26 Jul. 2012.
12. Fruergaard, Bjarne Ørum, and Lars Kai Hansen. "Compact Web browsing profiles for click-through rate prediction." In 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), pp. 1-6. IEEE, 2014.
13. Zhou, Feng, Hua Yin, Lizhang Zhan, Huafei Li, Yeliang Fan, and Liu Jiang. "A Novel Ensemble Strategy Combining Gradient Boosted Decision Trees and Factorization Machine Based Neural Network for Clicks Prediction." In 2018 International Conference on Big Data and Artificial Intelligence (BDAI), pp. 29-33. IEEE, 2018.
14. Liping, Zhu, Ji Lianen, and Guo Wensheng. "Heterogeneous Data Mining in Search Advertisement Click Rates." In 2009 International Conference on Web Information Systems and Mining, pp. 128-132. IEEE, 2009.
15. Wang, Rui, Ping Guo, Xin Fan, Biao Li, Wei Zhang, and Xin Xin. "Listwise Click-Through Rate Prediction with Item-Item Interactions." In 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 4417-4423. IEEE, 2018.

16. Jain, Harsh, and Misha Kakkar. "Job Recommendation System based on Machine Learning and Data Mining Techniques using RESTful API and Android IDE." In 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 416-421. IEEE, 2019.
17. Ta, Anh-Phuong. "Factorization machines with follow-the-regularized-leader for CTR prediction in display advertising." In 2015 IEEE International Conference on Big Data (Big Data), pp. 2889-2891. IEEE, 2015.

VITA

Sumod Pramod Badchhape completed his bachelor of engineering degree in Computer Engineering from Datta Meghe College of Engineering affiliated to University of Mumbai, Maharashtra, INDIA in 2018. He has been a graduate student in the Computer Science Department at Texas A&M University, Kingsville, TX, USA since August 2018. He was a Full Stack Developer with ScriptChain in Boston, MA during the August 2019 to December 2019. He received his Master of Science degree in Computer Science at Texas A&M University, Kingsville, TX, USA in May 2020.