A

Industrial-Oriented

Mini Project On

TWO STAGE JOB TITLE IDENTIFICATION SYSTEM FOR ONLINE JOB ADVERTISMENTS

(Submitted in partial fulfilment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

G. SURAJ
(227R1A0587)

Under the Guidance of

Najeema Afrin

(Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) & 12(B) of the UGCAct.1956,

Kandlakoya(V), Medchal Road, Hyderabad-501401.

June, 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**TWO STAGE JOB TITLE IDENTIFICATION SYSTEM FOR ONLINE JOB ADVERTISMENTS**" being submitted by **G. SURAJ (227R1A0587)** in partial fulfilment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, during the year 2024-25.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Najeema Afrin **Assistant Professor** **INTERNAL GUIDE**	**Dr. Nuthanakanti Bhaskar** **HoD**
Dr. A. Raji Reddy **DIRECTOR**	**Signature of External Examiner**

Submitted for viva voice Examination held on ______________________

ACKNOWLEDGEMENT

We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project, we take this opportunity to express our profound gratitude and deep regard to our guide **Najeema Afrin,** Assistant Professor for her exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by her shall carry us a long way in the journey of life on which we are about to embark.

We take this opportunity to extend our heartfelt appreciation to the Project Review Committee (PRC) Coordinators— **Y Varalaxmi, B Sekhar, D Nageswar Rao, and SVSV Prasad Sanaboina** —for their unwavering support, insightful guidance, and valuable inputs, which played a crucial role in steering this project through its various stages.

My sincere appreciation also goes to **Dr. Nuthanakanti Bhaskar**, Head, for his encouragement and continuous support in ensuring the successful completion of our project.

I am deeply grateful to **Dr. A. Raji Reddy**, Director, for his cooperation throughout the course of this project. Additionally, I extend our profound gratitude to Sri. **Ch. Gopal Reddy**, Chairman, Smt. **C. Vasantha Latha**, Secretary and Sri. **C. Abhinav Reddy**, Vice-Chairman, for fostering an excellent infrastructure and a conducive learning environment that greatly contributed to our progress.

I also acknowledge and appreciate the guidance and assistance provided by the faculty and staff of **CMR Technical Campus**, whose contributions have been invaluable in bringing this project to fruition.

Lastly, I sincerely thank our families for their unwavering support and encouragement. I also extend our gratitude to the teaching and non-teaching staff of CMR Technical Campus for their guidance and assistance. Their contributions, along with the support of everyone who helped directly or indirectly, have been invaluable in the successful completion of this project.

G. SURAJ
227R1A0587

VISION AND MISSION

INSTITUTE VISION:

To Impart quality education in serene atmosphere thus strive for excellence in Technology and Research.

INSTITUTE MISSION:

1. To create state of art facilities for effective Teaching- Learning Process.
2. Pursue and Disseminate Knowledge based research to meet the needs of Industry & Society.
3. Infuse Professional, Ethical and Societal values among Learning Community.

DEPARTMENT VISION:

To provide quality education and a conducive learning environment in computer engineering that foster critical thinking, creativity, and practical problem-solving skills.

DEPARTMENT MISSION:

1. To educate the students in fundamental principles of computing and induce the skills needed to solve practical problems.
2. To provide State-of-the-art computing laboratory facilities to promote industry institute interaction to enhance student’s practical knowledge.
3. To inculcate self-learning abilities, team spirit, and professional ethics among the students to serve society.

ABSTRACT

This project is titled as “Two Stage Job Title Identification System For Online Job Advertisements.”. The proliferation of online job advertisements has introduced significant variability in how job titles are presented, often due to informal language, inconsistent naming conventions, and company-specific jargon. This variability poses a major challenge for automated systems tasked with job classification, search relevance, and candidate-job matching. To tackle this issue, we propose a Two-Stage Job Title Identification System designed to robustly extract and standardize job titles from unstructured job postings.

In the first stage, advanced Natural Language Processing (NLP) techniques are applied to parse job descriptions and accurately extract potential job titles, effectively filtering out unrelated text and contextual noise. This involves a combination of named entity recognition (NER), part-of-speech tagging, syntactic analysis, and contextual embeddings to isolate job-specific terminology.

The second stage focuses on job title standardization and classification. Extracted titles are mapped to standardized taxonomies using a hybrid approach that combines supervised machine learning, rule-based normalization, and unsupervised clustering algorithms. This ensures uniformity in job title representation across diverse job postings, enabling more reliable analysis and categorization.

Our system significantly outperforms traditional keyword-based and rule-driven methods, offering improved accuracy in job title recognition and enhanced consistency for downstream applications. The proposed approach benefits job portals, recruitment platforms, and labor market researchers by enabling more precise job matching, improving HR analytics, and supporting labor trend analysis. Ultimately, this system contributes to more efficient and data-driven recruitment processes in the digital job market.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1. INTRODUCTION

The widespread use of the Internet in many industries due to the digitization of processes and the development of social media has resulted in a large amount of data that needs to be processed and analysed quickly and efficiently to extract valuable insights that can help in decision-making [1]. In this context, data science techniques can be powerful tools for extracting information from large datasets, facilitating the process of classifying different types of data (e.g., text, images, and video) [2], and can also solve many other tasks that are handled in a traditional manner, which is often time and resource consuming. Similarly, the job market shifted from traditional channels to online websites and job portals. This is because employers and recruiters share various job advertisements across different platforms to expand their reach and target more job seekers. This shift represents an opportunity to understand the needs of the job market from the vast amount of data shared daily, which can benefit many stakeholders [3]. In particular, identifying the requirements in terms of skills and occupations can help labor market analysts and policymakers foster employment and also help job seekers and students find suitable jobs and the training needed to successfully transition to the job market [4]. Classifying online job advertisements (ads) is not a trivial task. Indeed, the information contained in a job ad is expressed in plain text in a non-structured or semi- structured format, and the lexicon used by employers in the text is often very different from the occupational classifiers and the databases developed by human resources experts. In addition, job ads often include overly generic information that is not relevant to the position. This adds noise to the process of matching the job advertisement to its corresponding occupation. For instance, a job advertisement may have a title that includes information about the city where the job is located or some salary information. Also, the description can contain information about the company and information about other tasks that do not necessarily relate to the desired occupation. It is therefore necessary to apply advanced techniques for word and document representation and to use novel feature extraction methods to address these challenges. Most of the proposed methods for dealing with occupation normalization classifiers, ranging from traditional machine learning (ML) models to deep learning models, have been proposed for this task, such as support vector machine (SVM) [1], naïve bayes [5], k-nearestneighbor (KNN) [1], [5], artificial neural networks (ANNs) [6] and Bidirectional Encoder Representations from Transformers (BERT) [7]. While some studies used the combination of the title and the description to perform classification, the authors in [8] used only the title and found that 30% of the job offer titles did not contain enough information to identify the occupation. Similarly, the authors in [9] looked at the text of the job description only and found that each job description could correspond to more than one occupation. To the best of our knowledge, no previous study has examined the degree to which the title and description contribute when normalizing job ads. Classifying job ads using an occupational classifier or internal taxonomy has generally achieved satisfying results [6], [10]. However, these methodologies require human- labeled datasets with hundreds of thousands of examples, which is time-consuming and resource-intensive. In addition, updating the occupation description or including a newly created occupation in the occupational classifier is very difficult, as the entire training process must be repeated. Also, most prior work only focuses on English- language job ads and uses specific occupational classifiers such as the Occupational Information Network (O*NET); the extension of existing work to job ads written in other languages is challenging. which makes it extremely difficult to replicate their methodologies in other languages. On the other hand, using unsupervised models to identify the occupation, such as clustering [11] and field similarity [12], avoids training the model with labeled data, which are not always available; this is particularly relevant since we are dealing with a large number of occupations. The majority of previous works have relied on simple techniques for word embedding such as Bag of Words (BOW) [1], [12] or Term Frequency Inverse Document Frequency (TFIDF) [11] to generate word embedding and have applied averaging methods to calculate the document embedding. However, these techniques are considered weak in capturing the semantic relationships between the words, especially when we are dealing with job ads written by multiple employers who are using different lexicons. Therefore, word embedding approaches and feature extraction techniques [13] need to be closely monitored to achieve the highest results, as state-of the-art techniques do not perform well in all cases [11]. In this paper, we propose a job title identification methodology

based on self-supervised and unsupervised machine learning algorithms with minimal labeling and high accuracy that can be replicated on data from other countries to overcome the limitations mentioned above. The proposed methodology consists of two steps: the classification of job ads by sector and the matching of job ads with occupations belonging to the predicted sector. The step of job ads classification is done using several text classifiers such as SVM, Naïve Bayes, Logistic Regression, and BERT to classify job ads into their corresponding sectors (e.g., Information Technology (IT), Agriculture) which will help us focus on the occupations of the predicted sector instead of using all the occupations from the occupational classifier. For the job title identification step, we compare different techniques for vector representation of texts and use several combinations and parameters to propose a customized document embedding strategy. We also test several feature selection methods to extract important keywords from the description and analyze the degree of contribution of the title and the description in improving the results. Finally, we calculate the similarity between the job ad representation and the occupation representations belonging to the predicted sector to choose the closest one. To do this, we collect the French occupational classifier ''Pole Emploi'' and about two hundred thousand job ads from job portals. When used to identify the occupation title on a random sample of job ads, our methodology achieves an overall accuracy of 76.5% and more than 85% for some sectors which is considered high accuracy compared to prior work. Furthermore, the effectiveness of our approach was validated with the help of a team of domain experts who manually labeled a sample of our dataset. Finally, we applied our methodology to a dataset of 248,059 job ads in the French language to get an overview of the Moroccan job market, especially the IT sector. This study allows us to shed light on key sectors and occupations in the Moroccan job market where there is a high demand for IT profiles and Telemarketers which was identified by a previous study on the offshore sector in Morocco [14]. Using this methodology, we can identify emerging occupations that can help decision-makers including universities to take appropriate measures to adapt their programs and curricula, and to also help job seekers and students in their orientation by taking a career path that leads to employment.

1.1 OBJECTIVE OF THE PROJECT

Data science techniques are powerful tools for extracting knowledge from large datasets. Analyzing the job market by classifying online job advertisements (ads) has recently received much attention. Various approaches for multi-label classification (e.g., self-supervised learning and clustering) have been developed to identify the occupation from a job advertisement and have achieved a satisfying performance. However, these approaches require labeled datasets with hundreds of thousands of examples and focus on specific databases such as the Occupational Information Network (O*NET) that are more adapted to the US job market. In this paper, we present a two-stage job title identification methodology to address the case of small datasets. We use Bidirectional Encoder Representations from Transformers (BERT) to first classify the job ads according to their corresponding sector (e.g., Information Technology, Agriculture). Then, we use unsupervised machine learning algorithms and some similarity measures to find the closest matching job title from the list of occupations within the predicted sector. We also propose a novel document embedding strategy to address the issues of processing and classifying job ads. Our experimental results show that the proposed two-stage approach improves the job title identification accuracy by 14% to achieve more than 85% in some sectors. Moreover, we found that incorporating document embedding-based approaches such as weighting strategies and noise removal improves the classification accuracy by 23.5% compared to approaches based on the Bag of words model. Further evaluations verify that the proposed methodology either outperforms or performs at least as well as the state-of-the-art methods. Applying the proposed methodology to Moroccan job market data has helped identify emerging and high- demand occupations in Morocco.

2. LITERATURE SURVEY

Carotene: A Job Title Classification System for the Online Recruitment Domains

In the online job recruitment domain, accurate classification of jobs and resumes to occupation categories is important for matching job seekers with relevant jobs. An example of such a job title classification system is an automatic text document classification system that utilizes machine learning. Machine learning-based document classification techniques for images, text and related entities have been well researched in academia and have also been successfully applied in many industrial settings. In this paper we present Carotene, a machine learning-based semi-supervised job title classification system that is currently in production at CareerBuilder. Carotene leverages a varied collection of classification and clustering tools and techniques to tackle the challenges of designing a scalable classification system for a large taxonomy of job categories. It encompasses these techniques in a cascade classifier architecture. We first present the architecture of Carotene, which consists of a two-stage coarse and fine level classifier cascade. We compare Carotene to an early version that was based on a flat classifier architecture and also compare and contrast Carotene with a third party occupation classification system. The paper concludes by presenting experimental results on real world industrial data using both machine learning metrics and actual user experience surveys.

ScienceDirect Web Data Extraction Approach for Deep Web using WEIDJ

Data extraction is one of the most prominent areas in data mining analysis that is been extensively studied especially in the field of data requirements and reservoir. The main aim of data extraction with regards to semi-structured data is to retrieve beneficial information from the World Wide Web. The data from large web data also known as deep web is retrievable but it requires request through form submission because it cannot be performed by any search engines. Data mining applications and automatic data extraction are very cumbersome due to the diverse structure of web pages. Most of the previous data extraction techniques were dealing with various data types such as text, audio, video and but research works that are focusing on image as data are still lacking. Document Object Model (DOM) is an example of the state of the art of data extraction technique that is related to research work in mining image data. DOM was the method used to solve semi-structured data extraction from web. However, as the HTML documents start to grow larger, it has been found that the process of data extraction has been plagued with lengthy processing time and noisy information. In this research work, we propose an improved model namely Wrapper Extraction of Image using DOM and JSON (WEIDJ) in response to the promising results of mining in a higher volume of web data from a various types of image format and taking the consideration of web data extraction from deep web. To observe the efficiency of the proposed model, we compare the performance of data extraction by different level of page extraction with existing methods such as VIBS, MDR, DEPTA and VIDE. It has yielded the best results in Precision with 100, Recall with 97.93103 and F-measure with 98.9547.

F. Javed et al., "Carotene: A Job Title Classification System for the Online Recruitment Domain" (2015)

This paper introduces *Carotene*, a machine learning-based system developed to classify job titles within online recruitment data. The authors propose a two-level classification framework:

- **First Level:** Coarse-grained classification (e.g., IT, Healthcare, Finance).
- **Second Level:** Fine-grained classification (e.g., Java Developer, Data Analyst).

Carotene uses a supervised learning approach, particularly Support Vector Machines (SVM), trained on real-world data obtained from job portals. The authors also apply natural language processing (NLP) techniques to extract and preprocess job titles and descriptions. The system demonstrated high accuracy and scalability, making it suitable for large-scale commercial applications. Its contribution is significant in enhancing job search efficiency and improving recruitment recommendation systems.

M. S. Pera et al., "Web-Based Closed-Domain Data Extraction on Online Advertisements" (2013)

This study addresses the problem of extracting structured data from semi-structured online job advertisements. The authors propose a web data extraction framework designed for closed-domain scenarios, particularly focused on employment listings.

The methodology involves:

- **Template detection:** Identifying recurring structures in web pages.
- **Field extraction:** Mapping specific elements such as job title, company name, location.
- **Rule-based heuristics:** Used to enhance extraction accuracy.

The results show that a domain-specific approach can outperform general-purpose information extraction systems in terms of precision. This work laid the foundation for building structured datasets from web advertisements, which are essential for training machine learning models in recruitment analytics.

R. Kessler et al., "A Hybrid Approach to Managing Job Offers and Candidates" (2012)

Kessler and colleagues propose a hybrid recommendation system that combines both rule-based and statistical techniques to match candidates with job offers. The system integrates:

- **Keyword-based filtering:** Using predefined taxonomies of skills and qualifications.
- **Semantic analysis:** For interpreting context and intent in job descriptions and resumes.
- **Statistical learning:** To improve match accuracy over time using user feedback.

This approach balances domain knowledge with adaptive learning, enabling better personalization and relevance in job-candidate matching systems. The hybrid methodology demonstrates higher precision compared to purely rule-based or statistical systems, and it paved the way for intelligent recruitment tools.

Rahhal et al., "Education Path: Student Orientation Based on the Job Market Needs" (2022)

This recent work addresses the need for aligning educational pathways with job market trends. The authors propose the *Education Path* system, which uses labor market analytics to help students make informed decisions about their academic futures.
Key components include:

- **Market demand analysis:** Using real-time labor data to identify trending careers.
- **Curriculum mapping:** Suggesting courses and programs that match future job requirements.
- **Predictive modeling:** Machine learning algorithms forecast future demand for specific skills.

The framework is particularly useful for educational institutions and career counselors to guide students toward professions that are in demand, reducing skill mismatches in the workforce. This work exemplifies the increasing importance of data-driven education planning.

S. Mittal et al., "A Performance Comparison of Machine Learning Classification Techniques for Job Titles Using Job Descriptions" (2020)

In this comparative study, Mittal et al. examine the performance of multiple machine learning classifiers in predicting job titles based on job descriptions. The models evaluated include:

- **Support Vector Machines (SVM)**
- **Naïve Bayes**
- **Decision Trees**
- **Random Forest**
- **K-Nearest Neighbors (KNN)**

The dataset comprises various job descriptions from different sectors. The authors evaluate the models based on accuracy, precision, recall, and F1-score. Findings indicate that Random Forest and SVM consistently outperform other models due to their robustness in handling textual features and noise in job descriptions.

This research is valuable for developers of recruitment platforms and HR analytics tools, offering a benchmark for model selection in job title classification tasks.

R. Boselli et al., "Using Machine Learning for Labour Market Intelligence" (2017)

This study explores how machine learning techniques can be leveraged to generate actionable labor market intelligence from large datasets. The authors use supervised learning algorithms to process job advertisements and extract occupational insights such as demand trends, skill requirements, and geographic distribution.

Key contributions:

- **Classification of job advertisements** into standard occupational codes (ISCO/ESCO).
- **Extraction of high-level labor insights** such as emerging skills and evolving job roles.
- **Use of labeled datasets** and feature engineering to train ML models.

This research is particularly significant for policymakers, government labor agencies, and job market analysts, as it provides a framework for continuously monitoring labor trends using intelligent systems.

3. SYSTEM ARCHITECTURE & DESIGN

3.1 PROJECT ARCHITECTURE

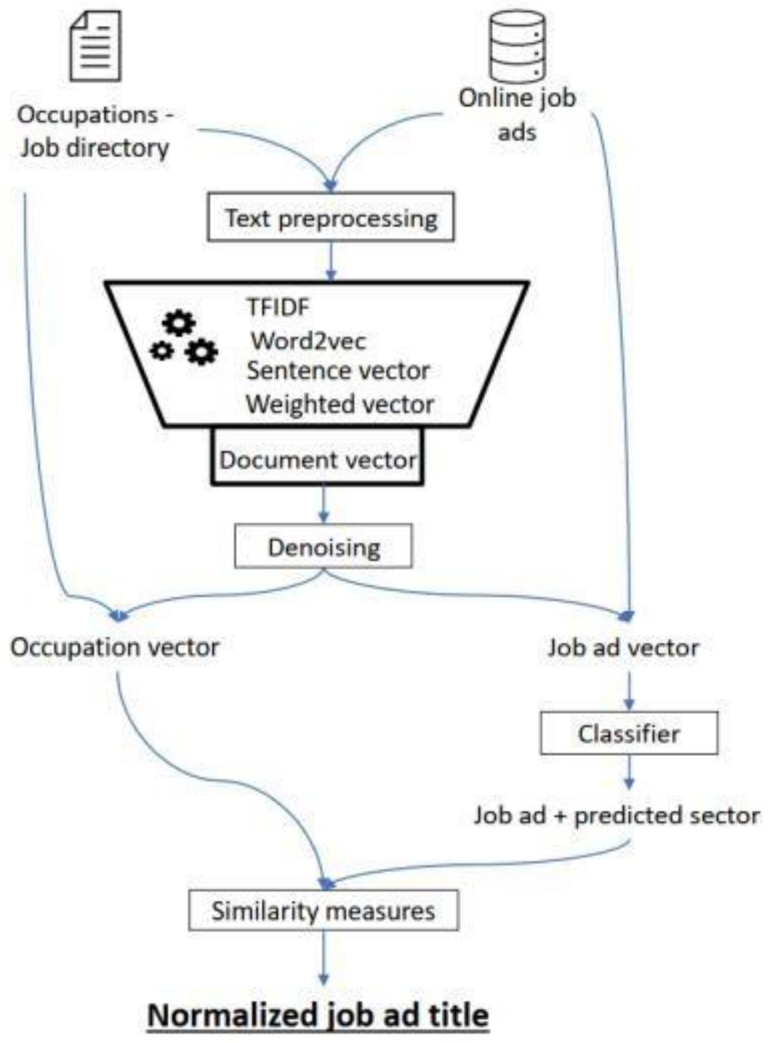


Figure 3.1: Project Architecture of Two Stage Job Title Identification System for Online Job Advertisements

3.2 DESCRIPTION

1. **Input Sources**

- Occupations – Job Directory: Structured data containing standardized job titles or occupational categories.
- Online Job Ads: Unstructured text from job advertisements posted online.

2. **Text Preprocessing**

Both the job directory data and job ads undergo text preprocessing, which may include:

- Tokenization
- Lowercasing
- Removing stop words
- Lemmatization/stemming
- Cleaning special characters or HTML

3. **Document Vectorization**

The preprocessed text is transformed into document vectors using vectorization techniques like:

- TFIDF
- Word2Vec
- Sentence Vector
- Weighted Vector

These methods convert textual content into numerical form for further analysis.

4. **Denoising**

A denoising step is applied to clean the vectors, potentially removing irrelevant information or noise from the data to improve model accuracy.

5. **Branching Paths**

After obtaining the document vectors, the process splits:

a. Occupation Vector Path:

The vectors from the job directory are saved as occupation vectors.

b. Job Ad Vector Path:

The job ad vectors are passed to a classifier which predicts the sector associated with the ad.

The output is a job ad paired with its predicted sector.

6. **Similarity Matching**

Similarity measures (e.g., cosine similarity) are computed between the job ad vectors and occupation vectors.

This helps in matching each job ad to the most similar standardized job title.

7. **Output**

The final result is a normalized job ad title, representing a standardized version of the original job ad.

3.3 Data Flow Diagram

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Dataflow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out in precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

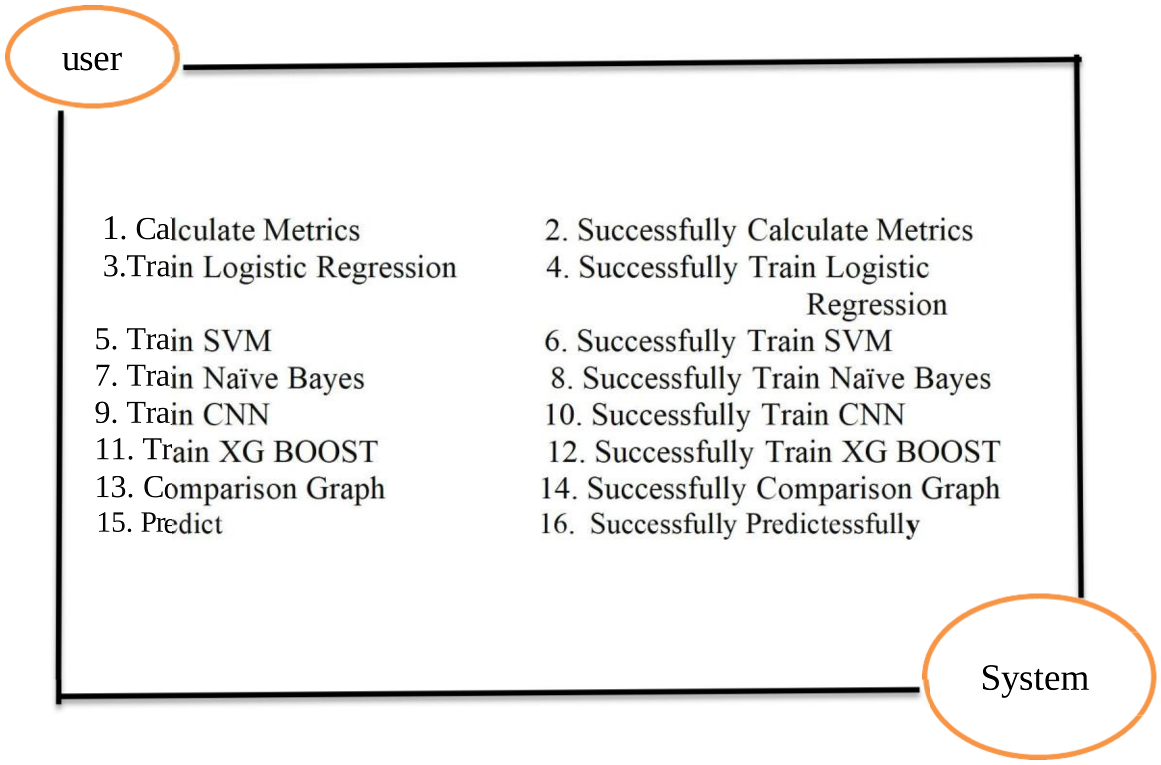

user
1. Calculate Metrics
3.Train Logistic Regression
5. Train SVM
7. Train Naïve Bayes
9. Train CNN
11. Train XG BOOST
13. Comparison Graph
15. Predict
2. Successfully Calculate Metrics
4. Successfully Train Logistic Regression
6. Successfully Train SVM
8. Successfully Train Naïve Bayes
10. Successfully Train CNN
12. Successfully Train XG BOOST
14. Successfully Comparison Graph
16. Successfully Predictessfully
System

Figure 3.2 Dataflow Diagram

4. IMPLEMENTATION

The implementation of the Two-Stage Job Title Identification System is designed to accurately classify job advertisements into standardized job titles using a combination of machine learning and vector-based techniques. The system processes job ad text and occupation descriptions through a structured pipeline involving text preprocessing, vectorization, classification, and similarity analysis. This section outlines the core algorithms employed in building the two-stage architecture and highlights their roles in achieving efficient and scalable job title normalization.

4.1 ALGORITHMS USED

4.1.1 Sentence Embedding using Word2Vec

Word2Vec is a neural network-based technique that learns vector representations of words by analyzing their context in large text corpora. It operates using two main architectures: Continuous Bag of Words (CBOW) and Skip-Gram. These embeddings capture the semantic and syntactic relationships between words, which are critical for understanding job titles and descriptions in context.

Usage in the Project:

In this system, Word2Vec is used to convert both job advertisements and occupation descriptions into document vectors. By averaging or weighting individual word embeddings, a full job post or occupation entry is represented as a single, dense vector. This enables more meaningful comparisons between texts during classification and similarity matching.

Advantages:

- Captures semantic relationships(e.g.,"developer" is close to "programmer").
- Efficient for large text corpora.
- Supports similarity computation with cosine distance.

4.1.2 Support Vector Machine (SVM)

SVM is a supervised learning algorithm ideal for text classification. It finds the optimal hyperplane that maximally separates the classes in high-dimensional space. When linear separation isn’t possible, SVM uses kernel functions to transform data into higher dimension

Usage in the Project:

SVM is applied in the first stage of classification to assign a coarse job sector (e.g., IT, Healthcare, Finance) to each job advertisement vector. By doing so, the search space for fine-level job title matching is reduced, improving both speed and accuracy.

Advantages:

- High performance with sparse, high-dimensional data (like text).
- Effective for binary and multi-class classification.
- Robust to overfitting in lower-dimensional cases.

4.1.3 Dimensionality Reduction with PCA

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of data while preserving as much variance as possible. It transforms correlated variables into a set of linearly uncorrelated components.

Usage in the Project:

After vectorizing the job ad and occupation text, PCA is used to denoise the vector space and eliminate irrelevant features. This step improves computational efficiency and enhances the classifier’s performance by focusing on key features.

Advantages:

- Reduces model complexity.
- Improves training time.
- Helps visualize high-dimensional data.

4.1.4 Cosine Similarity for Vector Matching

Cosine Similarity measures the cosine of the angle between two vectors in a multi-dimensional space. It is widely used for text similarity due to its effectiveness in measuring direction over magnitude.

Usage in the Project:

In the second stage, once the coarse sector is identified, Cosine Similarity is used to compare the job ad vector with all occupation vectors within that sector. The occupation with the highest similarity score is selected as the normalized job title.

Advantages:

- Ideal for comparing document embeddings.
- Not affected by the length of the document.
- Simple and efficient.

4.2 SAMPLE CODE

```
#import require python classes and packages
from string import punctuation
from nltk.corpus import stopwords
import nltk
from nltk.stem import WordNetLemmatizer
import numpy as np
import pandas as pd
import pickle
from nltk.stem import PorterStemmer
import os
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sentence_transformers import SentenceTransformer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer #loading tfidf vector
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression


from sklearn.naive_bayes import GaussianNB
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2 #loading CHI2 class
from sklearn.tree import DecisionTreeClassifier
from scipy.spatial import distance
from numpy import dot
from numpy.linalg import norm
from keras.utils.np_utils import to_categorical
from keras.layers import  MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D
from keras.models import Sequential, Model, load_model
from keras.callbacks import ModelCheckpoint
import seaborn as sns
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
#define object to remove stop words and other text processing
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
ps = PorterStemmer()
#define global variables to store labels, title and title + description
title_textdata = []
desc_textdata = []
labels = []
#define function to clean text by removing stop words and other special symbols
def cleanText(doc):
    tokens = doc.split()
    table = str.maketrans('', '', punctuation)


 tokens = [w.translate(table) for w in tokens]
    tokens = [word for word in tokens if word.isalpha()]
    tokens = [w for w in tokens if not w in stop_words]
    tokens = [word for word in tokens if len(word) > 1]
    tokens = [ps.stem(token) for token in tokens]
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    tokens = ' '.join(tokens)
    return tokens
#load & display dataset values
dataset = pd.read_csv("Dataset/JobsDataset.csv")
class_label, count = np.unique(dataset['Query'], return_counts=True)
dataset
#find and plot graph of different jobs found in dataset
count = count[0:5]
class_label = class_label[0:5]
height = count


bars = class_label
y_pos = np.arange(len(bars))
plt.figure(figsize =(6, 3))
plt.bar(y_pos, height)
plt.xticks(y_pos, bars)
plt.xlabel("Job Titles")
plt.ylabel("Count")
plt.xticks(rotation=90)
plt.show()
#function to get job label of give name
def getLabel(name):
    class_id = -1
    for i in range(len(class_label)):
        if class_label[i] == name:
            class_id = i
            break
    return class_id
#now read job details dataset and then convert to TFIDF vector and BERT vector
if os.path.exists("model/bert_title_desc.npy"):
    bert_desc_X = np.load("model/bert_title_desc.npy")#load bert title + description data
    Y = np.load("model/label.npy") #load training labels
    bert_title_X = np.load("model/bert_title.npy") #load bert title
    with open('model/desc_tfidf.txt', 'rb') as file:
        tfidf_desc_vector = pickle.load(file)
    file.close()
    with open('model/title_tfidf.txt', 'rb') as file:
        tfidf_title_vector = pickle.load(file)
    file.close()
    tfidf_title_X = np.load("model/tfidf_title_X.txt.npy")
    tfidf_desc_X = np.load("model/tfidf_desc_X.txt.npy")
else:
    for i in range(len(dataset)):
        label = dataset.get_value(i, 'Query')#loop all job details from dataset
        label = getLabel(label)
        if label < 5:
            title = dataset.get_value(i, 'Job Title')#get title
            desc = dataset.get_value(i, 'Description')#get description
            title = title.strip().lower()
            title = cleanText(title)#clean titke
            desc = desc.strip().lower()
            desc = cleanPost(desc)#clean description
            title_textdata.append(title)
            desc_textdata.append(title+" "+desc)#append both title and description
            labels.append(label)


    print(label)
    bert = SentenceTransformer('nli-distilroberta-base-v2'
    embeddings = bert.encode(desc_textdata, convert_to_tensor=True)
    X = embeddings.numpy()
    np.save("model/bert_title_desc", X)
    Y = np.asarray(labels)
    np.save("model/label", Y)
    embeddings = bert.encode(title_textdata, convert_to_tensor=True)
    X = embeddings.numpy()
    np.save("model/bert_title", X)
    #===============create TFIDF vector
    tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words, use_idf=True, ngram_range=(1,
1), smooth_idf=False, norm=None, decode_error='replace', max_features=768)
    tfidf = tfidf_vectorizer.fit_transform(desc_textdata).toarray()
    np.save("model/tfidf_desc_X.txt",tfidf)
    with open('model/desc_tfidf.txt', 'wb') as file:
        pickle.dump(tfidf_vectorizer, file)
    file.close()
    tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words, use_idf=True, ngram_range=(1,
1), smooth_idf=False, norm=None, decode_error='replace', max_features=768)
    tfidf = tfidf_vectorizer.fit_transform(title_textdata).toarray()
    np.save("model/tfidf_title_X.txt",tfidf)
    with open('model/title_tfidf.txt', 'wb') as file:
        pickle.dump(tfidf_vectorizer, file)
    file.close()
print("BERT and TFIDF vector generated")
print("BERT Vector : "+str(bert_desc_X))
print("TFIDF Vector : "+str(tfidf_desc_X))
#now preprocess dataset such as normalization and features selection usng CHI2 weights
scaler1 = MinMaxScaler((0,1))
scaler2 = MinMaxScaler((0,1))
scaler3 = MinMaxScaler((0,1))
scaler4 = MinMaxScaler((0,1))
```

```
bert_desc_X = scaler1.fit_transform(bert_desc_X)#normalized TFIDF and BERT Title and
description
bert_title_X = scaler2.fit_transform(bert_title_X)
tfidf_desc_X = scaler3.fit_transform(tfidf_desc_X)
tfidf_title_X = scaler4.fit_transform(tfidf_title_X)
selected1 = SelectKBest(score_func = chi2, k = 300)#select best top to 300 features usinggv
CHi2
bert_desc_X = selected1.fit_transform(bert_desc_X, Y)
selected2 = SelectKBest(score_func = chi2, k = 300)
bert_title_X = selected2.fit_transform(bert_title_X, Y)

selected3 = SelectKBest(score_func = chi2, k = 300)
tfidf_desc_X = selected3.fit_transform(tfidf_desc_X, Y)
selected4 = SelectKBest(score_func = chi2, k = 300)
tfidf_title_X = selected4.fit_transform(tfidf_title_X, Y)
print("Preprocessing completed")
#now split both BERT and TFIDF dataset into train & Test
bert_desc_X_train, bert_desc_X_test, bert_desc_y_train, bert_desc_y_test =
train_test_split(bert_desc_X, Y, test_size=0.2)
bert_title_X_train, bert_title_X_test, bert_title_y_train, bert_title_y_test =
train_test_split(bert_title_X, Y, test_size=0.2)
tfidf_desc_X_train, tfidf_desc_X_test, tfidf_desc_y_train, tfidf_desc_y_test =
train_test_split(tfidf_desc_X, Y, test_size=0.2)
tfidf_title_X_train, tfidf_title_X_test, tfidf_title_y_train, tfidf_title_y_test =
train_test_split(tfidf_title_X, Y, test_size=0.2)
print()
print("Dataset train & test split as 80% dataset for training and 20% for testing")
print("Training Size (80%): "+str(bert_desc_X_train.shape[0])) #print training and test size
print("Testing Size (20%): "+str(bert_desc_X_test.shape[0]))
print()
#define global variables to store accuracy and other metrics
precision = []
```

```
recall = []
fscore = []
accuracy = []
#function to calculate various metrics such as accuracy, precision etc
def calculateMetrics(algorithm, predict, testY):
    p = precision_score(testY, predict,average='macro') * 100
    r = recall_score(testY, predict,average='macro') * 100
    f = f1_score(testY, predict,average='macro') * 100
    a = accuracy_score(testY,predict)*100
    print(algorithm+' Accuracy  : '+str(a))
    print(algorithm+' Precision   : '+str(p))
    print(algorithm+' Recall      : '+str(r))
    print(algorithm+' FMeasure    : '+str(f))
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    conf_matrix = confusion_matrix(testY, predict)
    plt.figure(figsize =(5, 4))
    ax = sns.heatmap(conf_matrix, xticklabels = class_label, yticklabels = class_label, annot = 
True, cmap="viridis" ,fmt ="g");
    ax.set_ylim([0,len(class_label)])
    plt.title(algorithm+" Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()
#now train SVM algorithm of TFIDF features
svm_cls = svm.SVC() #create SVM object
svm_cls.fit(tfidf_desc_X_train, tfidf_desc_y_train)#train SVM on training data
predict = svm_cls.predict(tfidf_desc_X_test) #predict on test data
calculateMetrics("SVM", predict, tfidf_desc_y_test)#calculate accuracy and other metrics
#now train naive bayes algorithm
```

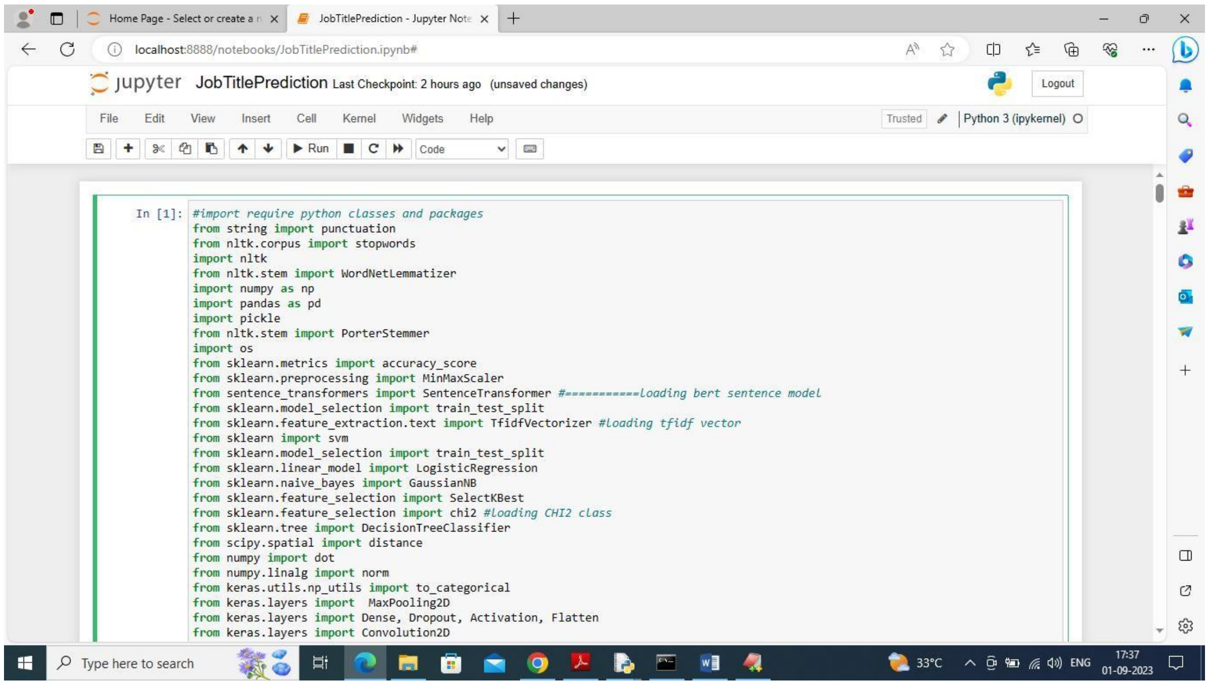
```
nb_cls = GaussianNB() #create SVM object
nb_cls.fit(tfidf_desc_X_train, tfidf_desc_y_train)#train Naive Bayes on training data
predict = nb_cls.predict(tfidf_desc_X_test) #predict on test data
calculateMetrics("Naive Bayes", predict, tfidf_desc_y_test)#calculate accuracy and other
metrics
#now train Logistic Regression algorithm
lr_cls = LogisticRegression() #create LR object
lr_cls.fit(tfidf_desc_X_train, tfidf_desc_y_train)#train LR on training data
predict = lr_cls.predict(tfidf_desc_X_test) #predict on test data
calculateMetrics("Logistic Regression", predict, tfidf_desc_y_test)#calculate accuracy and
other metrics
#train propose BERT model on BERT data using euclidean distance function to match
predicted label with highest similarity
#to avoid incorrect prediction
predict = []
for i in range(len(bert_desc_X_test)):#loop all test data
    max_value = 0
    pred = 0
    for j in range(len(bert_desc_X)):#loop all bert train data
        #calculate euclidean distance between bert train and test data
        Dst=dot(bert_desc_X_test[i],bert_desc_X[j])    /(norm(bert_desc_X_test[i])    *
norm(bert_desc_X[j]))
        #choose predicted label with max similarity
        if dst > max_value and dst != 1:
            max_value = dst
            pred = Y[j]
    #save max similarity label in predict array
    predict.append(pred)
calculateMetrics("Propose BERT Model", predict, bert_desc_y_test)#calculate accuracy and
other metrics
#now train extension CNN model using convolution 2D neural network as this algorithm
filtered features at multiple
```

```
#neurons iterations to train model with best features and this best features help CNN in getting
high accuracy
bert_desc_X_train = np.reshape(bert_desc_X_train, (bert_desc_X_train.shape[0], 10, 10, 3))
bert_desc_X_test = np.reshape(bert_desc_X_test, (bert_desc_X_test.shape[0],

10, 10, 3))
bert_desc_y_train = to_categorical(bert_desc_y_train)
bert_desc_y_test = to_categorical(bert_desc_y_test)
#define object
extension_model = Sequential()
#add CNN2d layer with 32 neurons to filter features 32 times
extension_model.add(Convolution2D(32, (3 , 3), input_shape = (bert_desc_X_train.shape[1],
bert_desc_X_train.shape[2], bert_desc_X_train.shape[3]), activation = 'relu'))
#max layer collected filtered features from CNN layer
extension_model.add(MaxPooling2D(pool_size = (2, 2)))
#defining another filter
extension_model.add(Convolution2D(32, (3, 3), activation = 'relu'))
extension_model.add(MaxPooling2D(pool_size = (2, 2)))
extension_model.add(Flatten())
#defining output layer
extension_model.add(Dense(units = 256, activation = 'relu'))
extension_model.add(Dense(units = bert_desc_y_train.shape[1], activation = 'softmax'))
#compile and train the model
extension_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['accuracy'])
if os.path.exists("model/cnn_weights.hdf5") == False:
   model_check_point = ModelCheckpoint(filepath='model/cnn_weights.hdf5', verbose = 1,
save_best_only = True)
   hist = extension_model.fit(bert_desc_X_train, bert_desc_y_train, batch_size = 16, epochs =
50, validation_data=(bert_desc_X_test, bert_desc_y_test), callbacks=[model_check_point],
verbose=1)
   f = open('model/cnn_history.pckl', 'wb')
```

```
    pickle.dump(hist.history, f)
    f.close()
else:
    extension_model = load_model("model/cnn_weights.hdf5")
#perform prediction on test data
predict = extension_model.predict(bert_desc_X_test)
predict = np.argmax(predict, axis=1)
y_test1 = np.argmax(bert_desc_y_test, axis=1)
calculateMetrics("Extension CNN2d Model", predict, y_test1)#calculate accuracy and other metrics
#all algorithms performance graph
df = pd.DataFrame([['SVM','Precision',precision[0]],['SVM','Recall',recall[0]],['SVM','F1 Score',fscore[0]],['SVM','Accuracy',accuracy[0]],
                ['Naive Bayes','Precision',precision[1]],['Naive Bayes','Recall',recall[1]],['Naive Bayes','F1 Score',fscore[1]],['Naive Bayes','Accuracy',accuracy[1]],

           ['Logistic Regression','Precision',precision[2]],['Logistic Regression','Recall',recall[2]],['Logistic Regression','F1 Score',fscore[2]],['Logistic Regression','Accuracy',accuracy[2]],
                ['Propose Bert','Precision',precision[3]],['Propose Bert','Recall',recall[3]],['Propose Bert','F1 Score',fscore[3]],['Propose Bert','Accuracy',accuracy[3]],
                ['Extension Bert with CNN2D','Precision',precision[4]],['Extension Bert with CNN2D','Recall',recall[4]],['Extension Bert with CNN2D','F1 Score',fscore[4]],['Extension Bert with CNN2D','Accuracy',accuracy[4]],
                ],columns=['Parameters','Algorithms','Value'])
df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar',figsize =(6, 3))
plt.title("All Algorithms Performance Graph")
plt.show()
#showing all algorithms with scenario A and B performance values
columns = ["Algorithm Name","Precison","Recall","FScore","Accuracy"]
values = []
algorithm_names = ["SVM", "Naive Bayes", "Logistic Regression", "Propose BERT",
'Extension BERT CNN2D']
for i in range(len(algorithm_names)):
    values.append([algorithm_names[i],precision[i],recall[i],fscore[i],accuracy[i]])

temp = pd.DataFrame(values,columns=columns)
temp
#create bert object
bert = SentenceTransformer('nli-distilroberta-base-v2')
print("Bert model created")
#now predict job title from job description
dataset = pd.read_csv("Dataset/testData.csv",encoding = "ISO-8859-1")#read test data
dataset = dataset.values
for i in range(len(dataset)):#loop all jobs description
    data = dataset[i,0]
    data = cleanText(data)#clean job description
    temp = []
    temp.append(data)#add message to array
    embeddings = bert.encode(data, convert_to_tensor=True)#convert message review to bert
vector
    X = embeddings.numpy()#convert vector to numpy
    X = X.reshape(1, -1)
    X = scaler1.transform(X)#normaize bert fetures
    X = selected1.transform(X)#select features using CHI2 selected objeect
    X = np.reshape(X, (X.shape[0], 10, 10, 3))
    predict = extension_model.predict(X)#now predict job title from job description features
using extension model
```

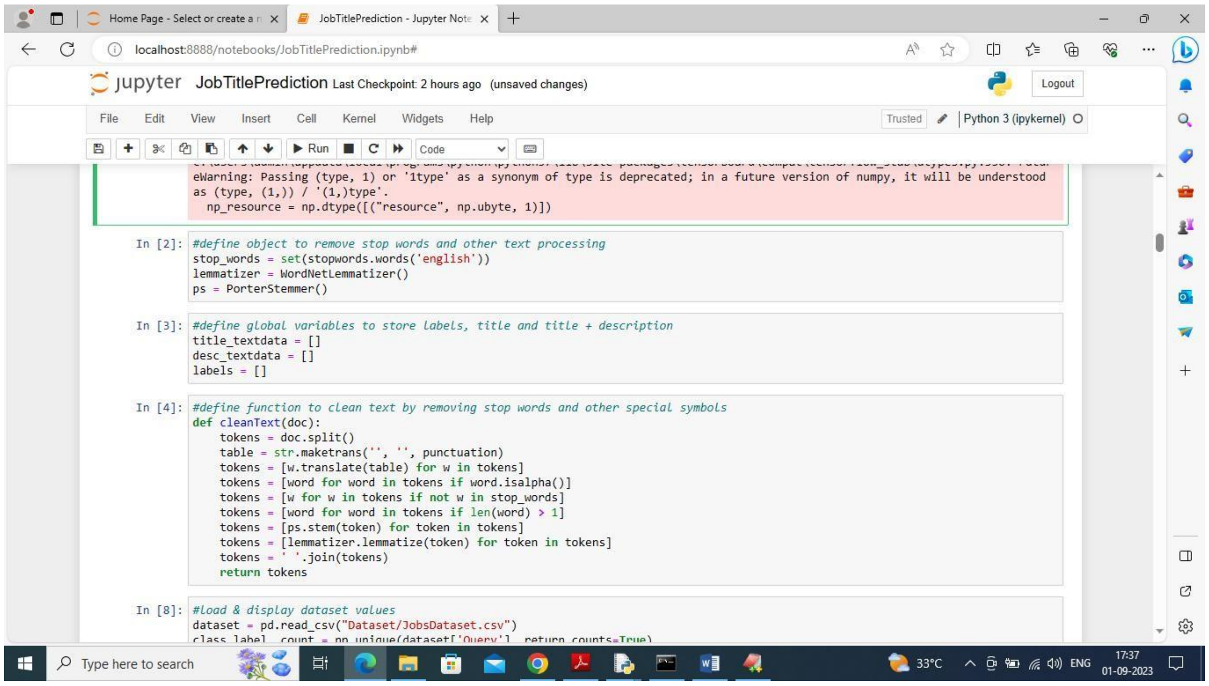
5. RESULTS & DISCUSSION

We have coded this project using JUPYTER notebook and below are the code and output screens with blue colour comments



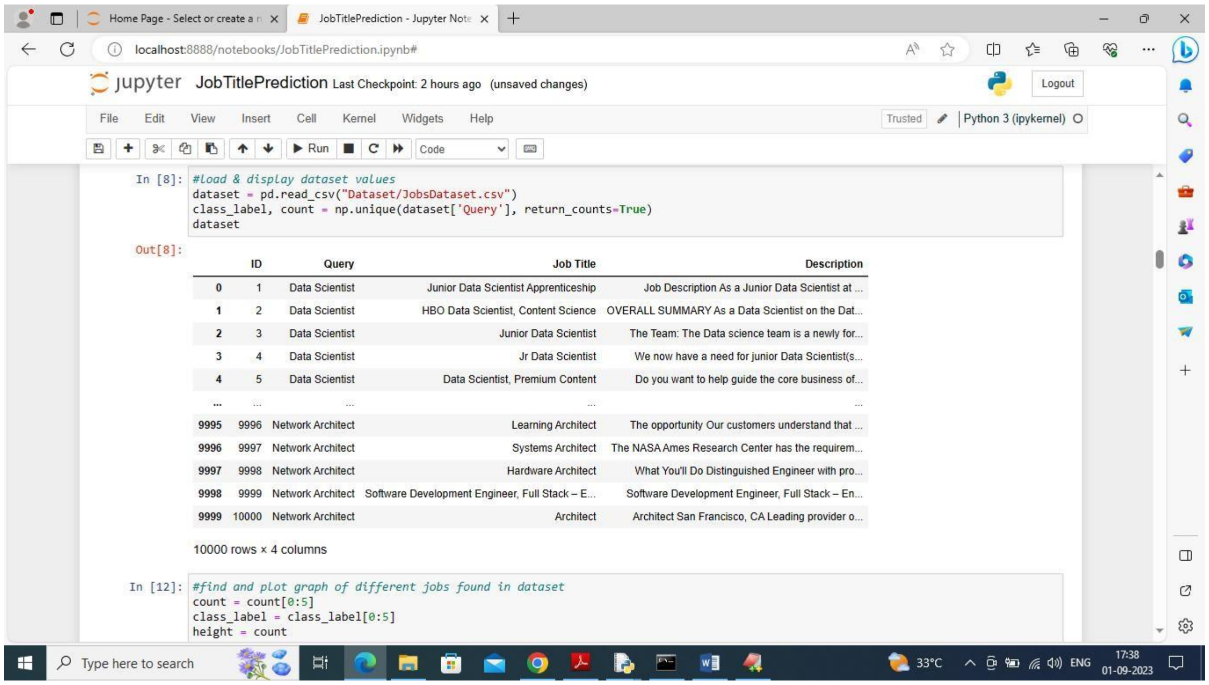
```
In [1]: #import require python classes and packages
        from string import punctuation
        from nltk.corpus import stopwords
        import nltk
        from nltk.stem import WordNetLemmatizer
        import numpy as np
        import pandas as pd
        import pickle
        from nltk.stem import PorterStemmer
        import os
        from sklearn.metrics import accuracy_score
        from sklearn.preprocessing import MinMaxScaler
        from sentence_transformers import SentenceTransformer #===========loading bert sentence model
        from sklearn.model_selection import train_test_split
        from sklearn.feature_extraction.text import TfidfVectorizer #loading tfidf vector
        from sklearn import svm
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.naive_bayes import GaussianNB
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2 #loading CHI2 class
        from sklearn.tree import DecisionTreeClassifier
        from scipy.spatial import distance
        from numpy import dot
        from numpy.linalg import norm
        from keras.utils.np_utils import to_categorical
        from keras.layers import  MaxPooling2D
        from keras.layers import Dense, Dropout, Activation, Flatten
        from keras.layers import Convolution2D
```

Figure 5.1: Require python classes and packages



```
eWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood
as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])

In [2]: #define object to remove stop words and other text processing
        stop_words = set(stopwords.words('english'))
        lemmatizer = WordNetLemmatizer()
        ps = PorterStemmer()

In [3]: #define global variables to store labels, title and title + description
        title_textdata = []
        desc_textdata = []
        labels = []

In [4]: #define function to clean text by removing stop words and other special symbols
        def cleanText(doc):
            tokens = doc.split()
            table = str.maketrans('', '', punctuation)
            tokens = [w.translate(table) for w in tokens]
            tokens = [word for word in tokens if word.isalpha()]
            tokens = [w for w in tokens if not w in stop_words]
            tokens = [word for word in tokens if len(word) > 1]
            tokens = [ps.stem(token) for token in tokens]
            tokens = [lemmatizer.lemmatize(token) for token in tokens]
            tokens = ' '.join(tokens)
            return tokens

In [8]: #load & display dataset values
        dataset = pd.read_csv("Dataset/JobsDataset.csv")
```

Figure 5.2: Code to remove stop words, special symbols etc.



```
In [8]: #load & display dataset values
        dataset = pd.read_csv("Dataset/JobsDataset.csv")
        class_label, count = np.unique(dataset['Query'], return_counts=True)
        dataset
```

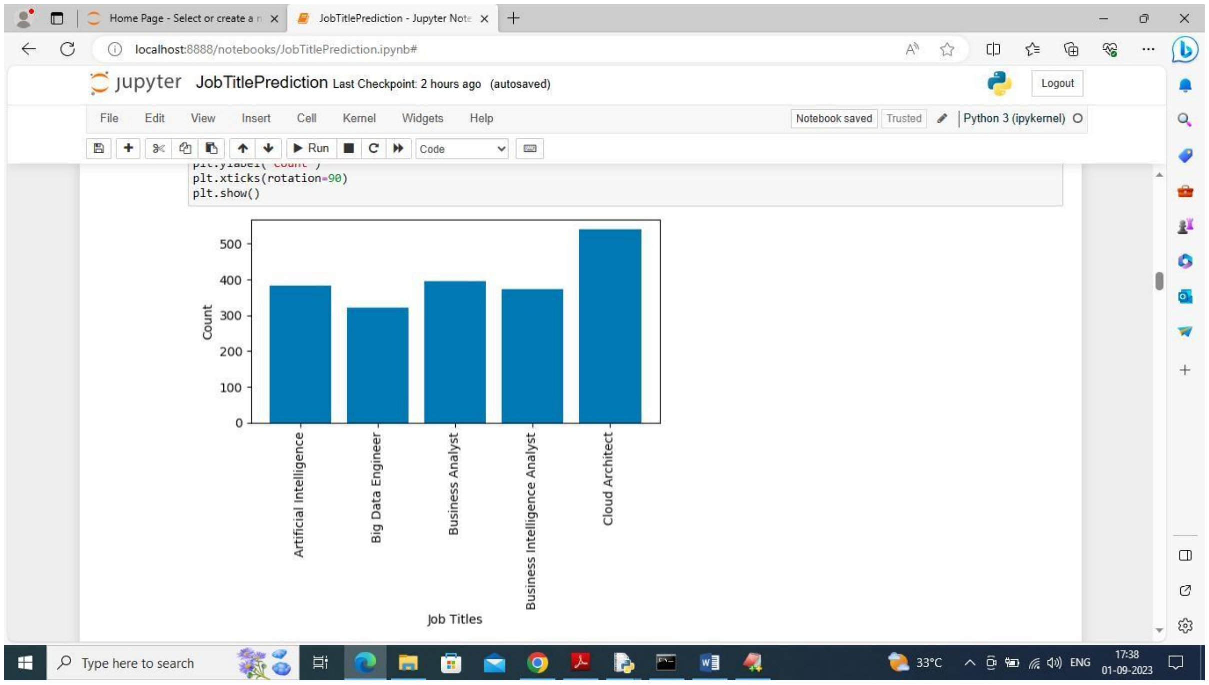
Out[8]:

	ID	Query	Job Title	Description
0	1	Data Scientist	Junior Data Scientist Apprenticeship	Job Description As a Junior Data Scientist at ...
1	2	Data Scientist	HBO Data Scientist, Content Science	OVERALL SUMMARY As a Data Scientist on the Dat...
2	3	Data Scientist	Junior Data Scientist	The Team: The Data science team is a newly for...
3	4	Data Scientist	Jr Data Scientist	We now have a need for junior Data Scientist(s...
4	5	Data Scientist	Data Scientist, Premium Content	Do you want to help guide the core business of...
...	...	...	...	...
9995	9996	Network Architect	Learning Architect	The opportunity Our customers understand that ...
9996	9997	Network Architect	Systems Architect	The NASA Ames Research Center has the requirem...
9997	9998	Network Architect	Hardware Architect	What You'll Do Distinguished Engineer with pro...
9998	9999	Network Architect	Software Development Engineer, Full Stack – E...	Software Development Engineer, Full Stack – En...
9999	10000	Network Architect	Architect	Architect San Francisco, CA Leading provider o...

10000 rows × 4 columns

```
In [12]: #find and plot graph of different jobs found in dataset
         count = count[0:5]
         class_label = class_label[0:5]
         height = count
```

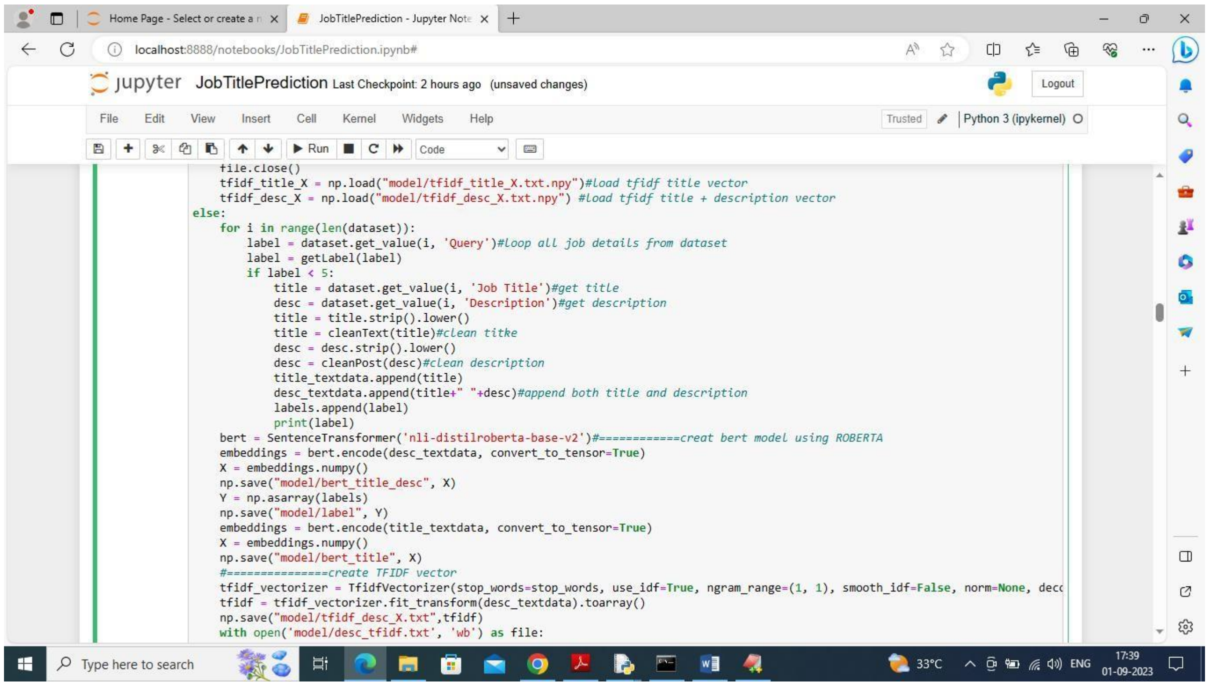
Figure 5.3: Code to remove stop words, special symbols etc.



```
plt.xticks(rotation=90)
plt.show()
```

Figure 5.4: Plotting graph of various JOBS found in dataset

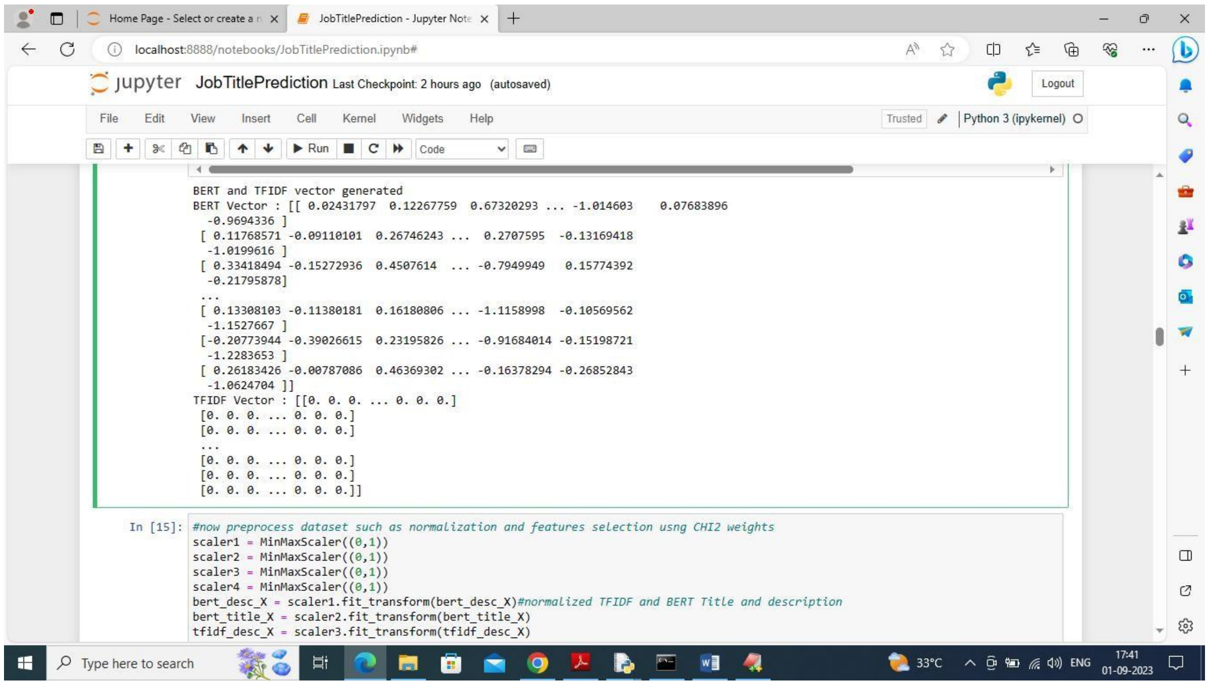
In above screen finding and plotting graph of various JOBS found in dataset where x-axis represents JOB TITLE and y-axis represents counts.



```
        file.close()
        tfidf_title_X = np.load("model/tfidf_title_X.txt.npy")#load tfidf title vector
        tfidf_desc_X = np.load("model/tfidf_desc_X.txt.npy") #load tfidf title + description vector
else:
    for i in range(len(dataset)):
        label = dataset.get_value(i, 'Query')#loop all job details from dataset
        label = getLabel(label)
        if label < 5:
            title = dataset.get_value(i, 'Job Title')#get title
            desc = dataset.get_value(i, 'Description')#get description
            title = title.strip().lower()
            title = cleanText(title)#clean titke
            desc = desc.strip().lower()
            desc = cleanPost(desc)#clean description
            title_textdata.append(title)
            desc_textdata.append(title+" "+desc)#append both title and description
            labels.append(label)
            print(label)
    bert = SentenceTransformer('nli-distilroberta-base-v2')#============creat bert model using ROBERTA
    embeddings = bert.encode(desc_textdata, convert_to_tensor=True)
    X = embeddings.numpy()
    np.save("model/bert_title_desc", X)
    Y = np.asarray(labels)
    np.save("model/label", Y)
    embeddings = bert.encode(title_textdata, convert_to_tensor=True)
    X = embeddings.numpy()
    np.save("model/bert_title", X)
    #===============create TFIDF vector
    tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words, use_idf=True, ngram_range=(1, 1), smooth_idf=False, norm=None, decc
    tfidf = tfidf_vectorizer.fit_transform(desc_textdata).toarray()
    np.save("model/tfidf_desc_X.txt",tfidf)
    with open('model/desc_tfidf.txt', 'wb') as file:
```

Figure 5.5: Reading each JOB description and then cleaning

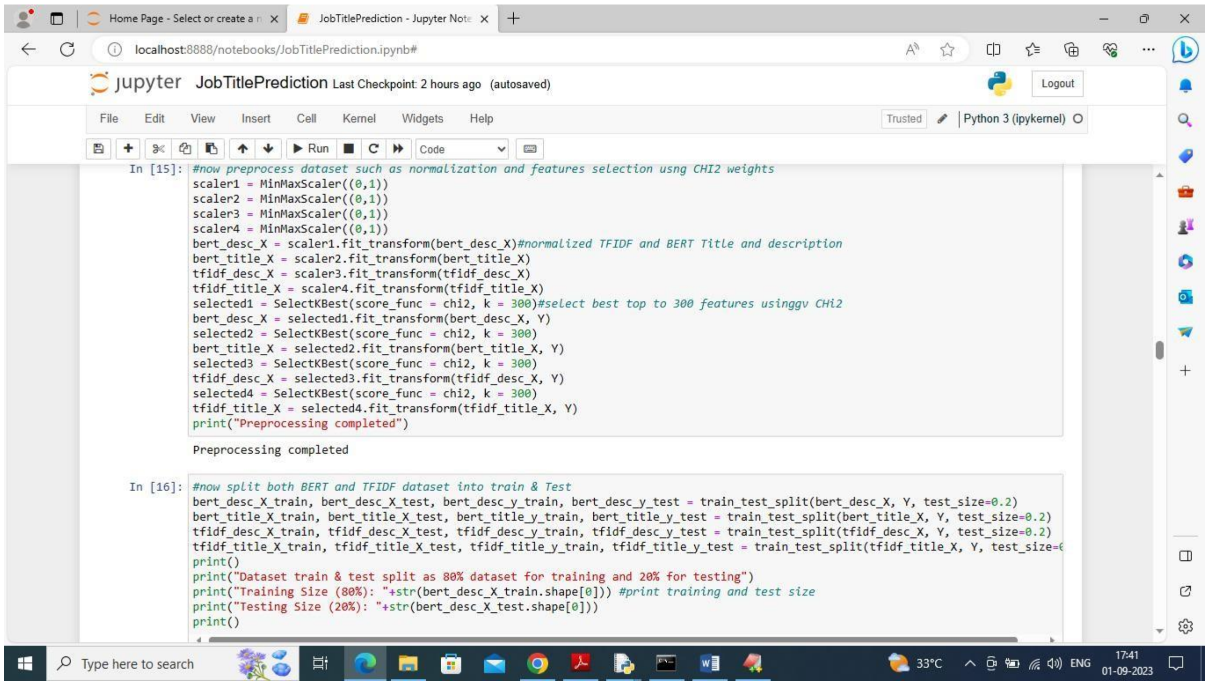
In above screen we are reading each JOB description and then cleaning and adding to array variable and then creating BERT and TFIDF object to convert all JOB description into numeric vector. In above screen =========== before dashed lines you can see we are creating BERT and TFIDF vectors and after executing above block will get below vector



```
BERT and TFIDF vector generated
BERT Vector : [[ 0.02431797  0.12267759  0.67320293 ... -1.014603    0.07683896
  -0.9694336 ]
 [ 0.11768571 -0.09110101  0.26746243 ...  0.2707595  -0.13169418
  -1.0199616 ]
 [ 0.33418494 -0.15272936  0.4507614  ... -0.7949949   0.15774392
  -0.21795878]
 ...
 [ 0.13308103 -0.11380181  0.16180806 ... -1.1158998  -0.10569562
  -1.1527667 ]
 [-0.20773944 -0.39026615  0.23195826 ... -0.91684014 -0.15198721
  -1.2283653 ]
 [ 0.26183426 -0.00787086  0.46369302 ... -0.16378294 -0.26852843
  -1.0624704 ]]
TFIDF Vector : [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
In [15]: #now preprocess dataset such as normalization and features selection usng CHI2 weights
scaler1 = MinMaxScaler((0,1))
scaler2 = MinMaxScaler((0,1))
scaler3 = MinMaxScaler((0,1))
scaler4 = MinMaxScaler((0,1))
bert_desc_X = scaler1.fit_transform(bert_desc_X)#normalized TFIDF and BERT Title and description
bert_title_X = scaler2.fit_transform(bert_title_X)
tfidf_desc_X = scaler3.fit_transform(tfidf_desc_X)
```

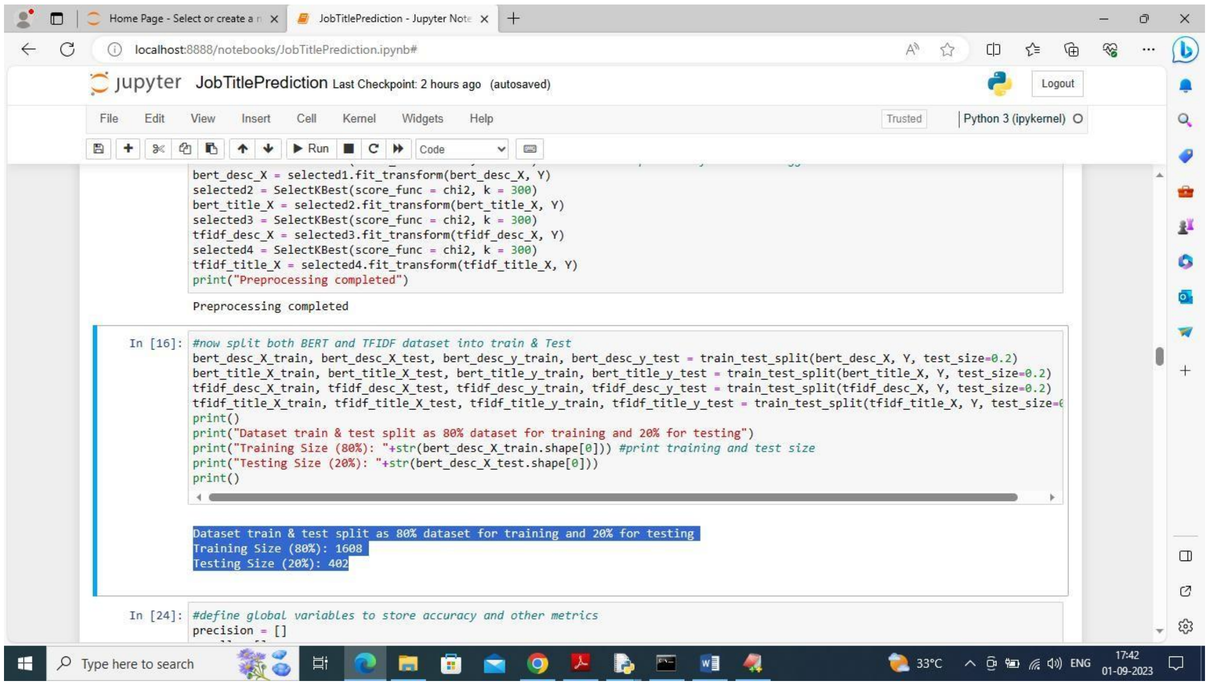
Figure 5.6: BERT and TFIDF vector created.



```
In [15]: #now preprocess dataset such as normalization and features selection usng CHI2 weights
         scaler1 = MinMaxScaler((0,1))
         scaler2 = MinMaxScaler((0,1))
         scaler3 = MinMaxScaler((0,1))
         scaler4 = MinMaxScaler((0,1))
         bert_desc_X = scaler1.fit_transform(bert_desc_X)#normalized TFIDF and BERT Title and description
         bert_title_X = scaler2.fit_transform(bert_title_X)
         tfidf_desc_X = scaler3.fit_transform(tfidf_desc_X)
         tfidf_title_X = scaler4.fit_transform(tfidf_title_X)
         selected1 = SelectKBest(score_func = chi2, k = 300)#select best top to 300 features usinggv CHi2
         bert_desc_X = selected1.fit_transform(bert_desc_X, Y)
         selected2 = SelectKBest(score_func = chi2, k = 300)
         bert_title_X = selected2.fit_transform(bert_title_X, Y)
         selected3 = SelectKBest(score_func = chi2, k = 300)
         tfidf_desc_X = selected3.fit_transform(tfidf_desc_X, Y)
         selected4 = SelectKBest(score_func = chi2, k = 300)
         tfidf_title_X = selected4.fit_transform(tfidf_title_X, Y)
         print("Preprocessing completed")

         Preprocessing completed

In [16]: #now split both BERT and TFIDF dataset into train & Test
         bert_desc_X_train, bert_desc_X_test, bert_desc_y_train, bert_desc_y_test = train_test_split(bert_desc_X, Y, test_size=0.2)
         bert_title_X_train, bert_title_X_test, bert_title_y_train, bert_title_y_test = train_test_split(bert_title_X, Y, test_size=0.2)
         tfidf_desc_X_train, tfidf_desc_X_test, tfidf_desc_y_train, tfidf_desc_y_test = train_test_split(tfidf_desc_X, Y, test_size=0.2)
         tfidf_title_X_train, tfidf_title_X_test, tfidf_title_y_train, tfidf_title_y_test = train_test_split(tfidf_title_X, Y, test_size=
         print()
         print("Dataset train & test split as 80% dataset for training and 20% for testing")
         print("Training Size (80%): "+str(bert_desc_X_train.shape[0])) #print training and test size
         print("Testing Size (20%): "+str(bert_desc_X_test.shape[0]))
         print()
```

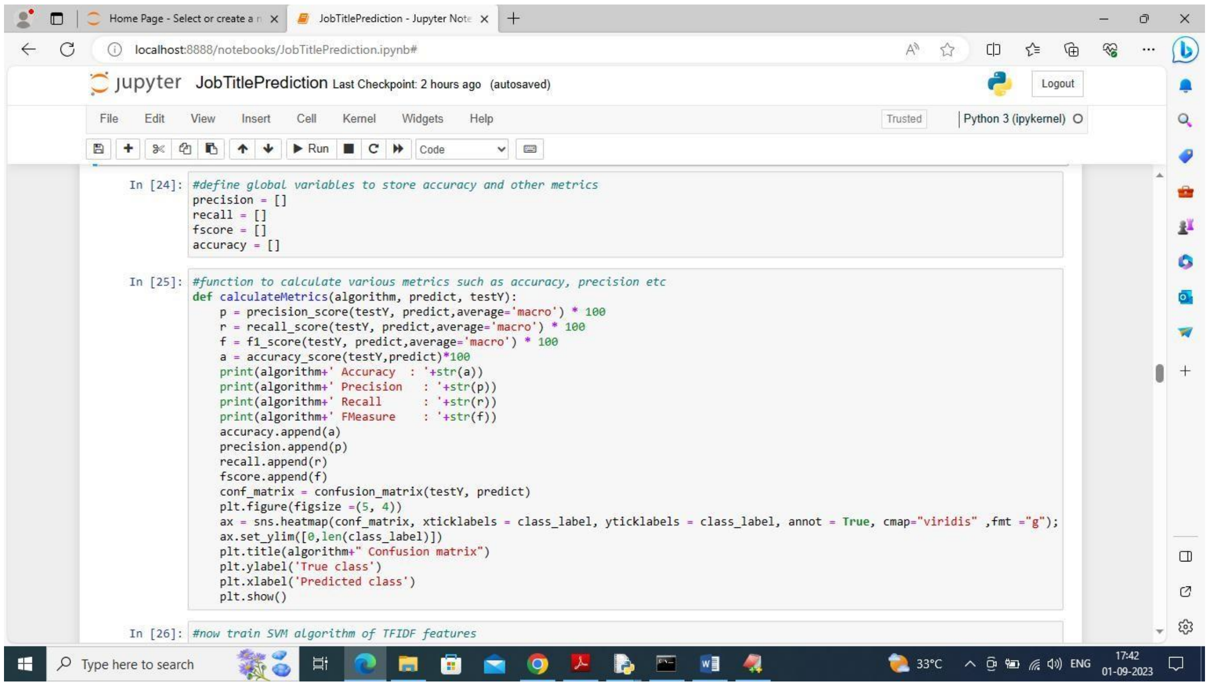
Figure 5.7: Normalizing and applying CHI2 algorithm

In above screen normalizing and applying CHI2 algorithm on both BERT and TFIDF vector.



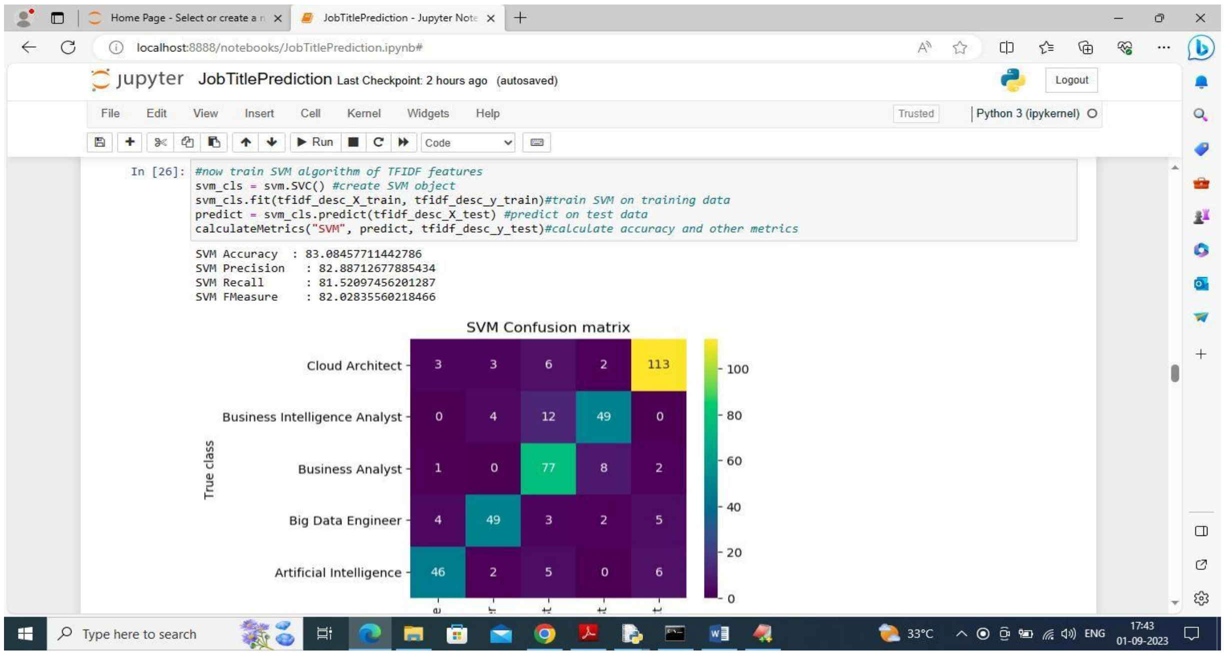
```
         bert_desc_X = selected1.fit_transform(bert_desc_X, Y)
         selected2 = SelectKBest(score_func = chi2, k = 300)
         bert_title_X = selected2.fit_transform(bert_title_X, Y)
         selected3 = SelectKBest(score_func = chi2, k = 300)
         tfidf_desc_X = selected3.fit_transform(tfidf_desc_X, Y)
         selected4 = SelectKBest(score_func = chi2, k = 300)
         tfidf_title_X = selected4.fit_transform(tfidf_title_X, Y)
         print("Preprocessing completed")

         Preprocessing completed

In [16]: #now split both BERT and TFIDF dataset into train & Test
         bert_desc_X_train, bert_desc_X_test, bert_desc_y_train, bert_desc_y_test = train_test_split(bert_desc_X, Y, test_size=0.2)
         bert_title_X_train, bert_title_X_test, bert_title_y_train, bert_title_y_test = train_test_split(bert_title_X, Y, test_size=0.2)
         tfidf_desc_X_train, tfidf_desc_X_test, tfidf_desc_y_train, tfidf_desc_y_test = train_test_split(tfidf_desc_X, Y, test_size=0.2)
         tfidf_title_X_train, tfidf_title_X_test, tfidf_title_y_train, tfidf_title_y_test = train_test_split(tfidf_title_X, Y, test_size=
         print()
         print("Dataset train & test split as 80% dataset for training and 20% for testing")
         print("Training Size (80%): "+str(bert_desc_X_train.shape[0])) #print training and test size
         print("Testing Size (20%): "+str(bert_desc_X_test.shape[0]))
         print()

         Dataset train & test split as 80% dataset for training and 20% for testing
         Training Size (80%): 1608
         Testing Size (20%): 402

In [24]: #define global variables to store accuracy and other metrics
         precision = []
```

Figure 5.8: Splitting data into train and test.



```
In [24]: #define global variables to store accuracy and other metrics
         precision = []
         recall = []
         fscore = []
         accuracy = []

In [25]: #function to calculate various metrics such as accuracy, precision etc
         def calculateMetrics(algorithm, predict, testY):
             p = precision_score(testY, predict,average='macro') * 100
             r = recall_score(testY, predict,average='macro') * 100
             f = f1_score(testY, predict,average='macro') * 100
             a = accuracy_score(testY,predict)*100
             print(algorithm+' Accuracy  : '+str(a))
             print(algorithm+' Precision   : '+str(p))
             print(algorithm+' Recall      : '+str(r))
             print(algorithm+' FMeasure    : '+str(f))
             accuracy.append(a)
             precision.append(p)
             recall.append(r)
             fscore.append(f)
             conf_matrix = confusion_matrix(testY, predict)
             plt.figure(figsize =(5, 4))
             ax = sns.heatmap(conf_matrix, xticklabels = class_label, yticklabels = class_label, annot = True, cmap="viridis" ,fmt ="g");
             ax.set_ylim([0,len(class_label)])
             plt.title(algorithm+" Confusion matrix")
             plt.ylabel('True class')
             plt.xlabel('Predicted class')
             plt.show()

In [26]: #now train SVM algorithm of TFIDF features
```

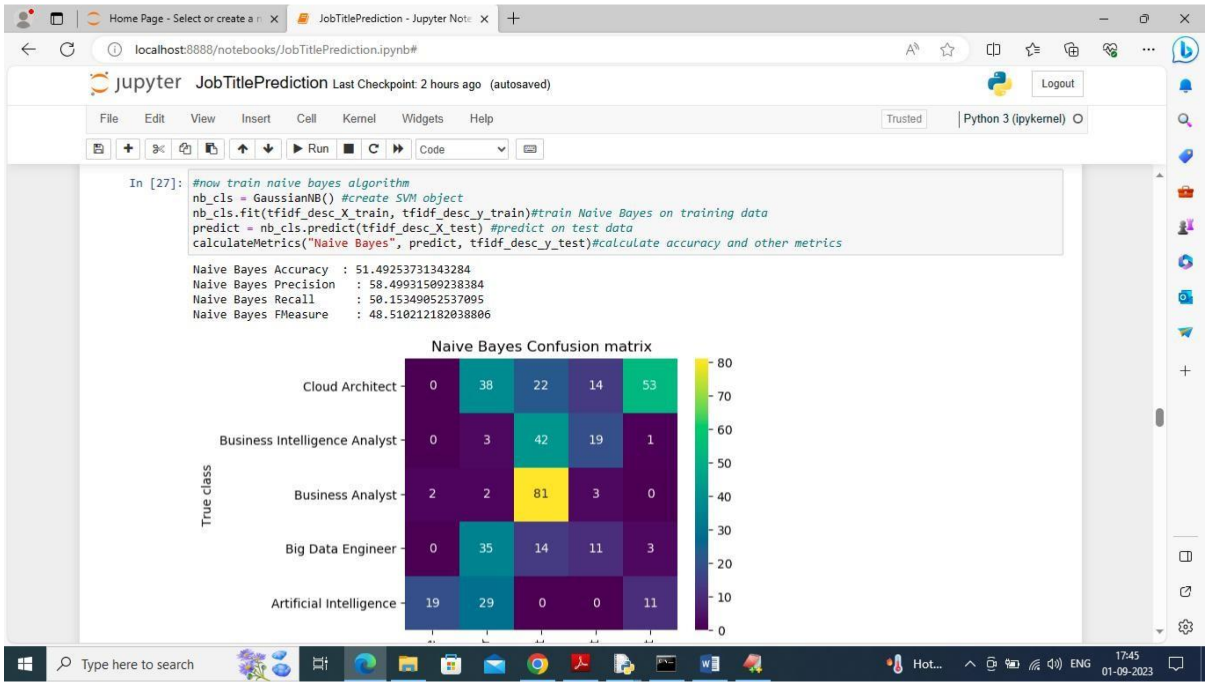
Figure 5.9: Function to calculate accuracy and other metrics.



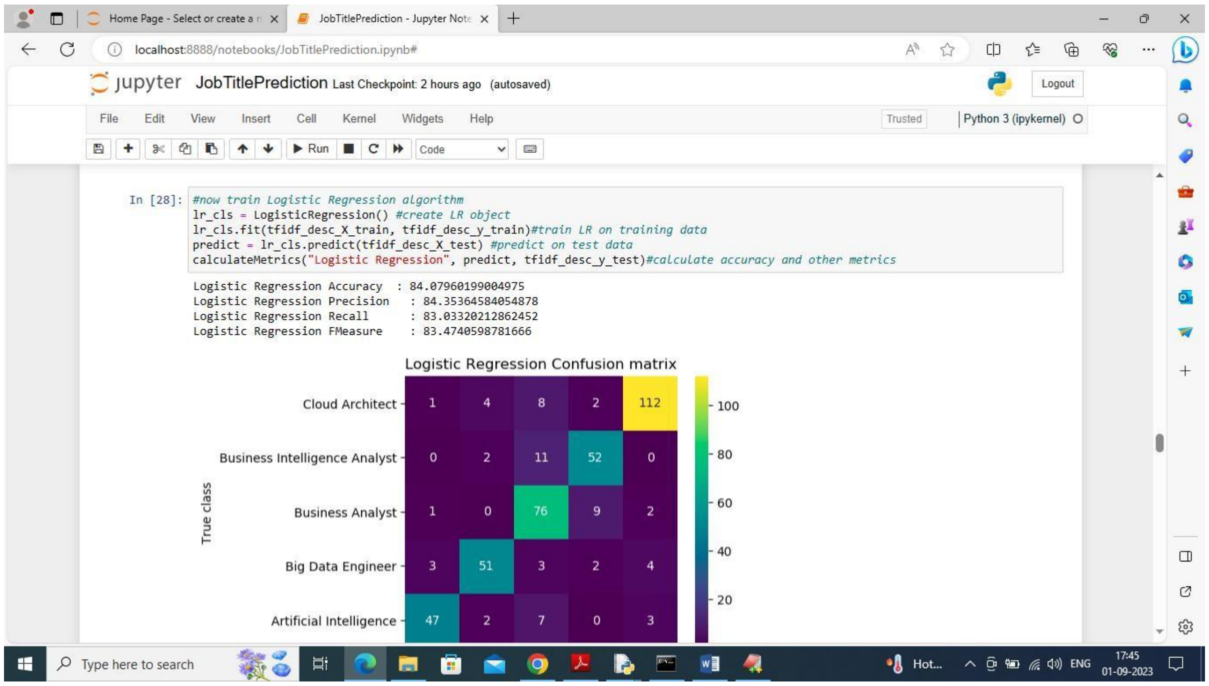
```
In [26]: #now train SVM algorithm of TFIDF features
         svm_cls = svm.SVC() #create SVM object
         svm_cls.fit(tfidf_desc_X_train, tfidf_desc_y_train)#train SVM on training data
         predict = svm_cls.predict(tfidf_desc_X_test) #predict on test data
         calculateMetrics("SVM", predict, tfidf_desc_y_test)#calculate accuracy and other metrics

         SVM Accuracy  : 83.08457711442786
         SVM Precision   : 82.88712677885434
         SVM Recall      : 81.52097456201287
         SVM FMeasure    : 82.02835560218466
```

Figure 5.10: Training SVM on TFIDF features and it got 83% accuracy

In above screen training SVM on TFIDF features and it got 83% accuracy and can see other metrics also and in confusion matrix graph x-axis represents True Job Title and Y-axis represents Predicted Job Title and all different colour boxes in diagnol represents correct prediction count and remaining blue boxes contains incorrect prediction count

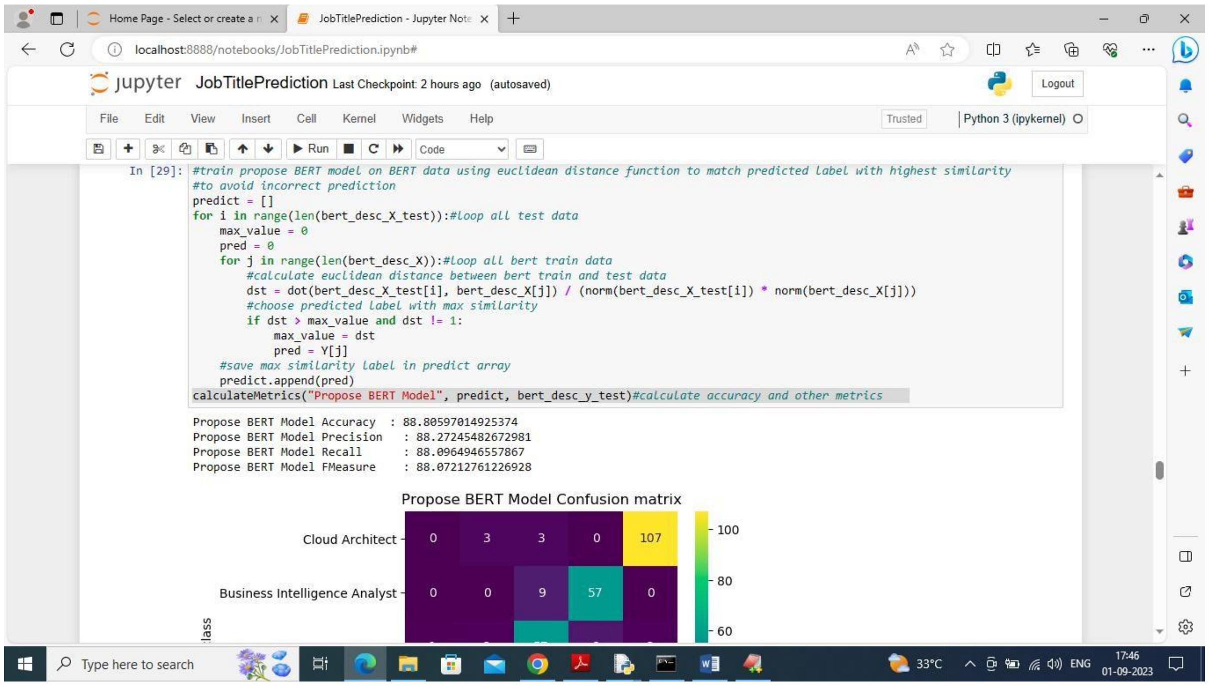


```
In [27]: #now train naive bayes algorithm
         nb_cls = GaussianNB() #create SVM object
         nb_cls.fit(tfidf_desc_X_train, tfidf_desc_y_train)#train Naive Bayes on training data
         predict = nb_cls.predict(tfidf_desc_X_test) #predict on test data
         calculateMetrics("Naive Bayes", predict, tfidf_desc_y_test)#calculate accuracy and other metrics

         Naive Bayes Accuracy  : 51.49253731343284
         Naive Bayes Precision   : 58.49931509238384
         Naive Bayes Recall      : 50.15349052537095
         Naive Bayes FMeasure    : 48.510212182038806
```



```
In [28]: #now train Logistic Regression algorithm
         lr_cls = LogisticRegression() #create LR object
         lr_cls.fit(tfidf_desc_X_train, tfidf_desc_y_train)#train LR on training data
         predict = lr_cls.predict(tfidf_desc_X_test) #predict on test data
         calculateMetrics("Logistic Regression", predict, tfidf_desc_y_test)#calculate accuracy and other metrics

         Logistic Regression Accuracy  : 84.07960199004975
         Logistic Regression Precision   : 84.35364584054878
         Logistic Regression Recall      : 83.03320212862452
         Logistic Regression FMeasure    : 83.4740598781666
```

Figure 5.11: **Logistic Regression got 84% accuracy**

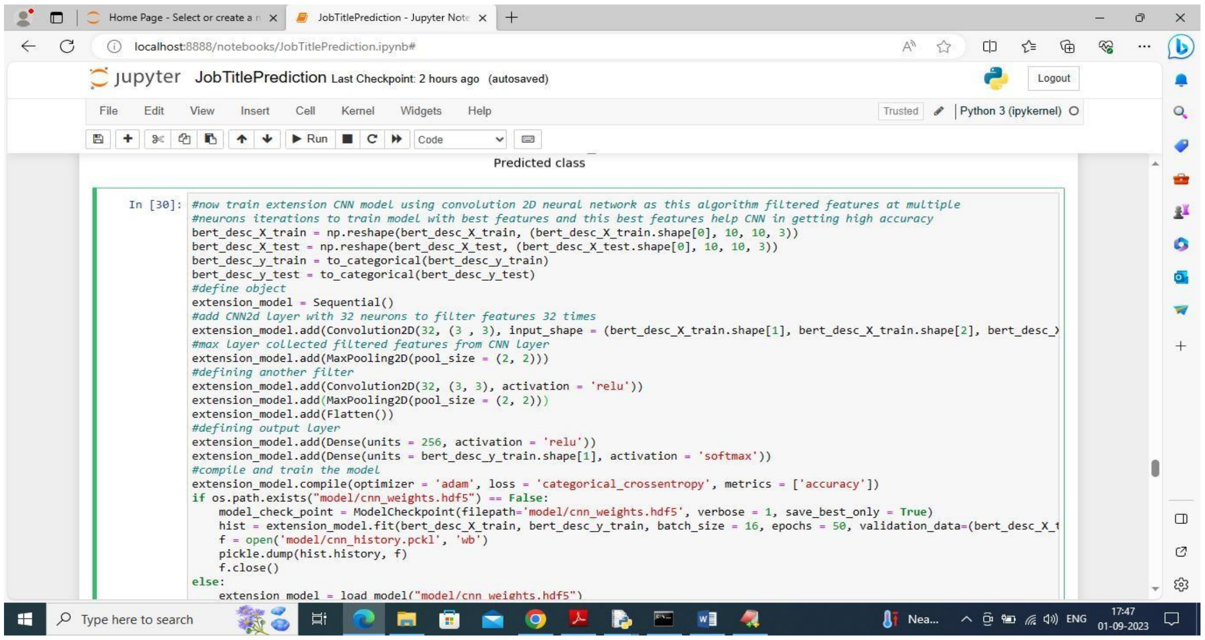


```
In [29]: #train propose BERT model on BERT data using euclidean distance function to match predicted label with highest similarity
         #to avoid incorrect prediction
         predict = []
         for i in range(len(bert_desc_X_test)):#loop all test data
             max_value = 0
             pred = 0
             for j in range(len(bert_desc_X)):#loop all bert train data
                 #calculate euclidean distance between bert train and test data
                 dst = dot(bert_desc_X_test[i], bert_desc_X[j]) / (norm(bert_desc_X_test[i]) * norm(bert_desc_X[j]))
                 #choose predicted label with max similarity
                 if dst > max_value and dst != 1:
                     max_value = dst
                     pred = Y[j]
             #save max similarity label in predict array
             predict.append(pred)
         calculateMetrics("Propose BERT Model", predict, bert_desc_y_test)#calculate accuracy and other metrics

Propose BERT Model Accuracy  : 88.80597014925374
Propose BERT Model Precision   : 88.27245482672981
Propose BERT Model Recall      : 88.0964946557867
Propose BERT Model FMeasure    : 88.07212761226928
```

Propose BERT Model Confusion matrix
Cloud Architect
0
3
3
0
107
Business Intelligence Analyst
0
0
9
57
0
100
80
60

Figure 5.12: Propose BERT model with max similarity measure

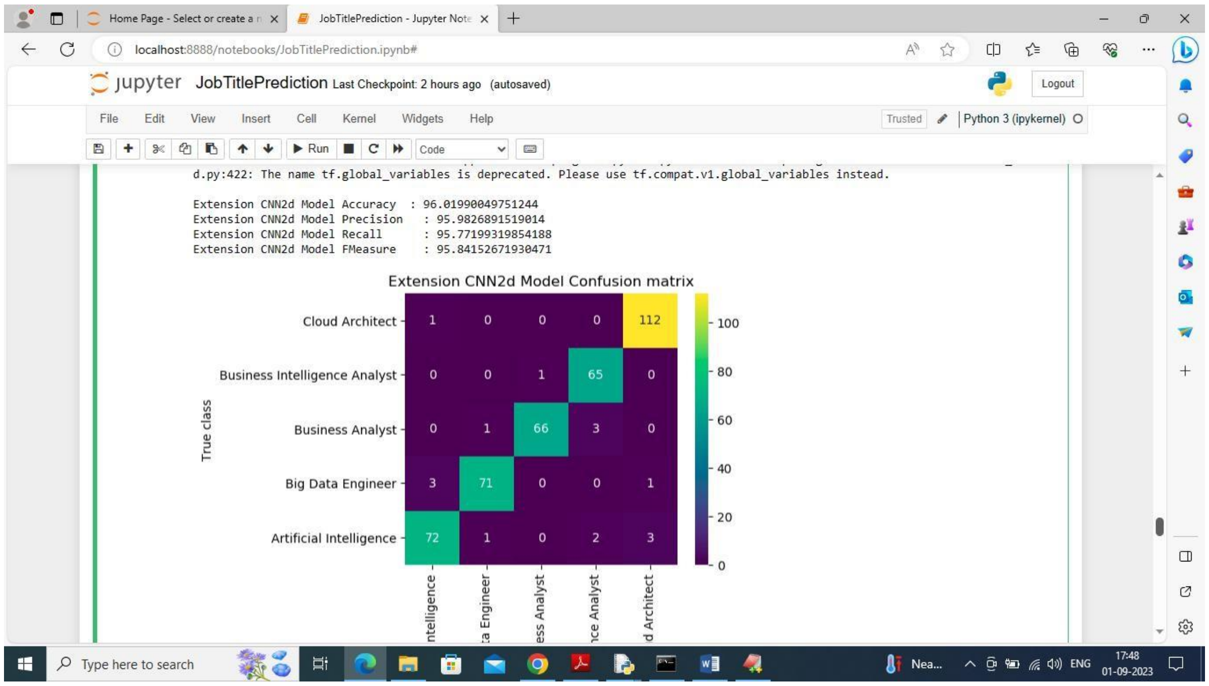
In above screen propose BERT model with max similarity measure got 88% accuracy which is higher than existing algorithms and can see other metrics also



```
In [30]: #now train extension CNN model using convolution 2D neural network as this algorithm filtered features at multiple
         #neurons iterations to train model with best features and this best features help CNN in getting high accuracy
         bert_desc_X_train = np.reshape(bert_desc_X_train, (bert_desc_X_train.shape[0], 10, 10, 3))
         bert_desc_X_test = np.reshape(bert_desc_X_test, (bert_desc_X_test.shape[0], 10, 10, 3))
         bert_desc_y_train = to_categorical(bert_desc_y_train)
         bert_desc_y_test = to_categorical(bert_desc_y_test)
         #define object
         extension_model = Sequential()
         #add CNN2d layer with 32 neurons to filter features 32 times
         extension_model.add(Convolution2D(32, (3 , 3), input_shape = (bert_desc_X_train.shape[1], bert_desc_X_train.shape[2], bert_desc_X
         #max layer collected filtered features from CNN layer
         extension_model.add(MaxPooling2D(pool_size = (2, 2)))
         #defining another filter
         extension_model.add(Convolution2D(32, (3, 3), activation = 'relu'))
         extension_model.add(MaxPooling2D(pool_size = (2, 2)))
         extension_model.add(Flatten())
         #defining output layer
         extension_model.add(Dense(units = 256, activation = 'relu'))
         extension_model.add(Dense(units = bert_desc_y_train.shape[1], activation = 'softmax'))
         #compile and train the model
         extension_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
         if os.path.exists("model/cnn_weights.hdf5") == False:
             model_check_point = ModelCheckpoint(filepath='model/cnn_weights.hdf5', verbose = 1, save_best_only = True)
             hist = extension_model.fit(bert_desc_X_train, bert_desc_y_train, batch_size = 16, epochs = 50, validation_data=(bert_desc_X_t
             f = open('model/cnn_history.pckl', 'wb')
             pickle.dump(hist.history, f)
             f.close()
         else:
             extension_model = load_model("model/cnn_weights.hdf5")
```

Figure 5.13: Train extension CNN2D algorithm.

In above screen train extension CNN2D algorithm and after executing above block will get below output



```
d.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Extension CNN2d Model Accuracy  : 96.01990049751244
Extension CNN2d Model Precision   : 95.9826891519014
Extension CNN2d Model Recall      : 95.77199319854188
Extension CNN2d Model FMeasure    : 95.84152671930471
```

True class	ntelligence	a Engineer	ess Analyst	nce Analyst	d Architect
Cloud Architect	1	0	0	0	112
Business Intelligence Analyst	0	0	1	65	0
Business Analyst	0	1	66	3	0
Big Data Engineer	3	71	0	0	1
Artificial Intelligence	72	1	0	2	3

Figure 5.14: Extension CNN2D model got 96% accuracy

In above screen extension CNN2D model got 96% accuracy and can see other metrics also

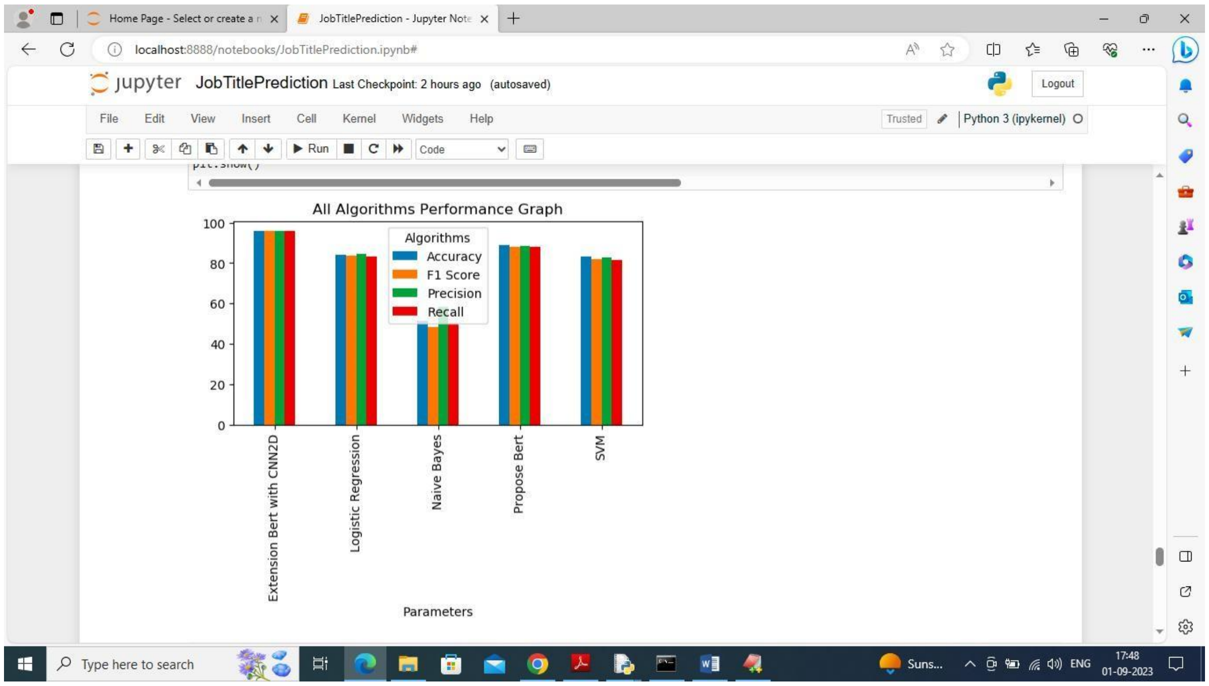
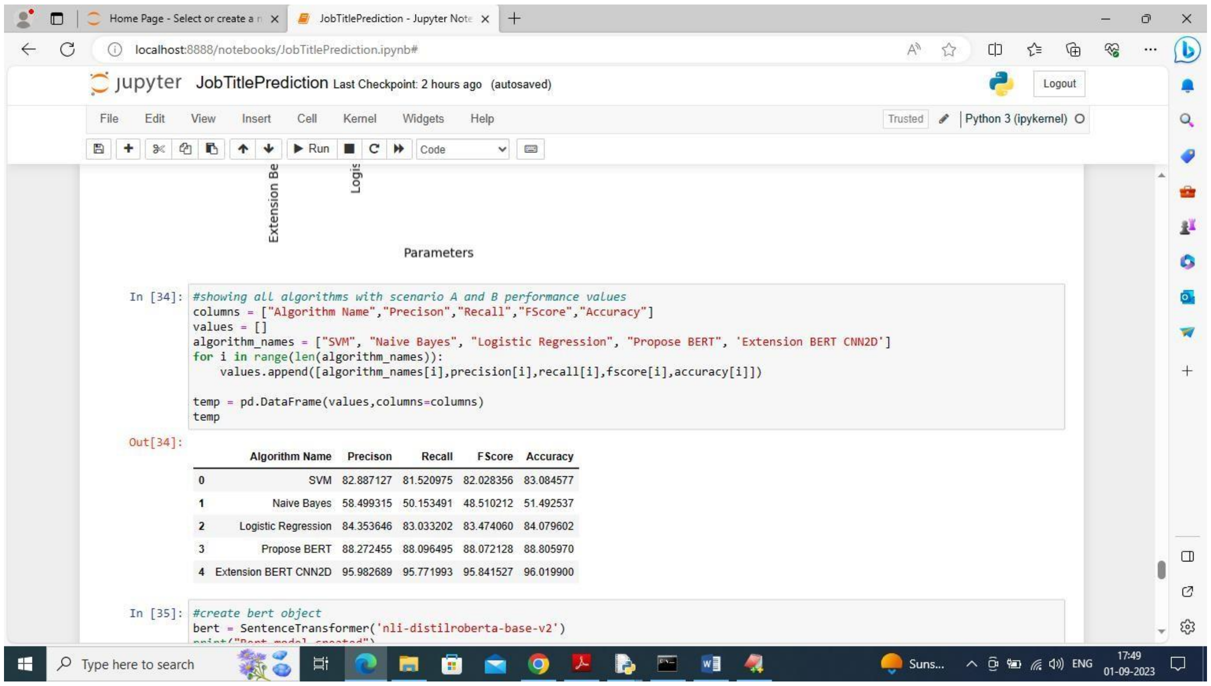


Figure 5.15: Displaying all algorithm performance where x-axis represents algorithm names.



```
In [34]: #showing all algorithms with scenario A and B performance values
         columns = ["Algorithm Name","Precison","Recall","FScore","Accuracy"]
         values = []
         algorithm_names = ["SVM", "Naive Bayes", "Logistic Regression", "Propose BERT", 'Extension BERT CNN2D']
         for i in range(len(algorithm_names)):
             values.append([algorithm_names[i],precision[i],recall[i],fscore[i],accuracy[i]])

         temp = pd.DataFrame(values,columns=columns)
         temp
```

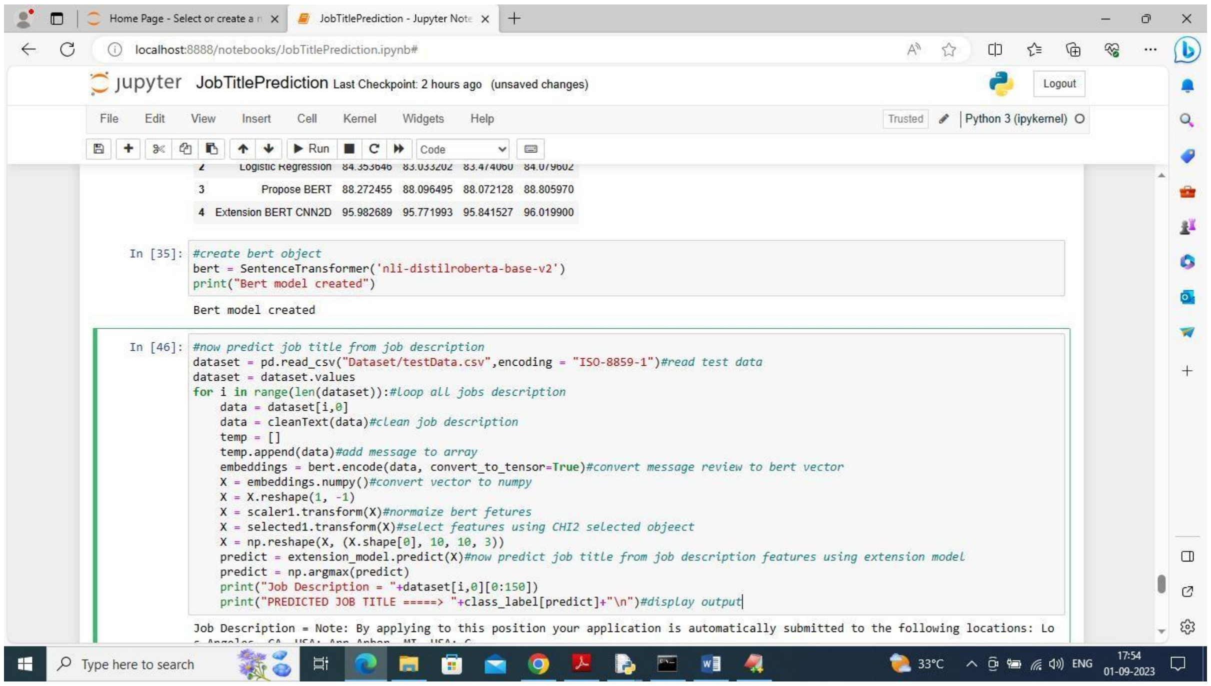
Out[34]:

	Algorithm Name	Precison	Recall	FScore	Accuracy
0	SVM	82.887127	81.520975	82.028356	83.084577
1	Naive Bayes	58.499315	50.153491	48.510212	51.492537
2	Logistic Regression	84.353646	83.033202	83.474060	84.079602
3	Propose BERT	88.272455	88.096495	88.072128	88.805970
4	Extension BERT CNN2D	95.982689	95.771993	95.841527	96.019900

```
In [35]: #create bert object
         bert = SentenceTransformer('nli-distilroberta-base-v2')
```

Figure 5.16: Displaying all algorithms

In above screen displaying all algorithms performance in tabular format

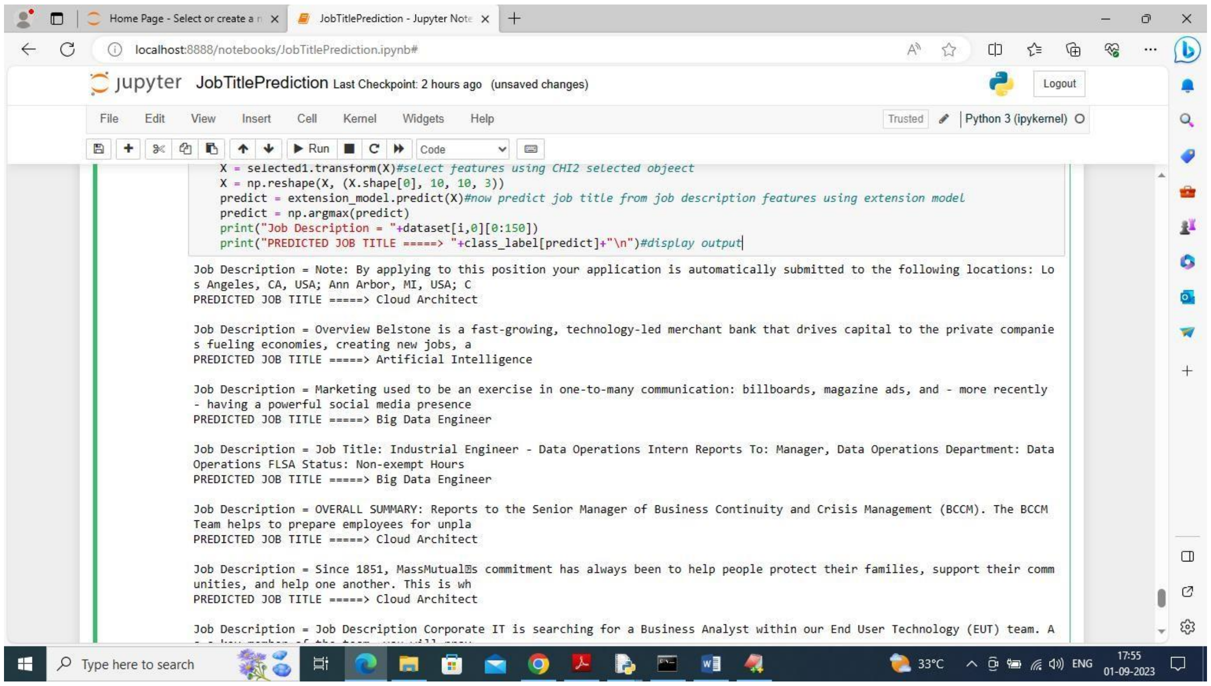


2	Logistic Regression	84.353646	83.033202	83.474060	84.079602
3	Propose BERT	88.272455	88.096495	88.072128	88.805970
4	Extension BERT CNN2D	95.982689	95.771993	95.841527	96.019900

```
In [35]: #create bert object
         bert = SentenceTransformer('nli-distilroberta-base-v2')
         print("Bert model created")

         Bert model created

In [46]: #now predict job title from job description
         dataset = pd.read_csv("Dataset/testData.csv",encoding = "ISO-8859-1")#read test data
         dataset = dataset.values
         for i in range(len(dataset)):#loop all jobs description
             data = dataset[i,0]
             data = cleanText(data)#clean job description
             temp = []
             temp.append(data)#add message to array
             embeddings = bert.encode(data, convert_to_tensor=True)#convert message review to bert vector
             X = embeddings.numpy()#convert vector to numpy
             X = X.reshape(1, -1)
             X = scaler1.transform(X)#normaize bert fetures
             X = selected1.transform(X)#select features using CHI2 selected objeect
             X = np.reshape(X, (X.shape[0], 10, 10, 3))
             predict = extension_model.predict(X)#now predict job title from job description features using extension model
             predict = np.argmax(predict)
             print("Job Description = "+dataset[i,0][0:150])
             print("PREDICTED JOB TITLE =====> "+class_label[predict]+"\n")#display output

         Job Description = Note: By applying to this position your application is automatically submitted to the following locations: Lo
```

Figure 5.17: Reading JOB description from TEST data

In above screen reading JOB description from TEST data and then predicting JOB TITLE and below is the output


Home Page - Select or create a n
JobTitlePrediction - Jupyter Note
localhost:8888/notebooks/JobTitlePrediction.ipynb#
jupyter JobTitlePrediction Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
Type here to search
33°C
ENG
17:55
01-09-2023

```
    X = selected1.transform(X)#select features using CHI2 selected objeect
    X = np.reshape(X, (X.shape[0], 10, 10, 3))
    predict = extension_model.predict(X)#now predict job title from job description features using extension model
    predict = np.argmax(predict)
    print("Job Description = "+dataset[i,0][0:150])
    print("PREDICTED JOB TITLE =====> "+class_label[predict]+"\n")#display output
```

```
Job Description = Note: By applying to this position your application is automatically submitted to the following locations: Lo
s Angeles, CA, USA; Ann Arbor, MI, USA; C
PREDICTED JOB TITLE =====> Cloud Architect

Job Description = Overview Belstone is a fast-growing, technology-led merchant bank that drives capital to the private companie
s fueling economies, creating new jobs, a
PREDICTED JOB TITLE =====> Artificial Intelligence

Job Description = Marketing used to be an exercise in one-to-many communication: billboards, magazine ads, and - more recently
- having a powerful social media presence
PREDICTED JOB TITLE =====> Big Data Engineer

Job Description = Job Title: Industrial Engineer - Data Operations Intern Reports To: Manager, Data Operations Department: Data
Operations FLSA Status: Non-exempt Hours
PREDICTED JOB TITLE =====> Big Data Engineer

Job Description = OVERALL SUMMARY: Reports to the Senior Manager of Business Continuity and Crisis Management (BCCM). The BCCM
Team helps to prepare employees for unpla
PREDICTED JOB TITLE =====> Cloud Architect

Job Description = Since 1851, MassMutual s commitment has always been to help people protect their families, support their comm
unities, and help one another. This is wh
PREDICTED JOB TITLE =====> Cloud Architect

Job Description = Job Description Corporate IT is searching for a Business Analyst within our End User Technology (EUT) team. A
```

Figure 5.18: First line we can see JOB Description

In above screen in first line we can see JOB Description and then after ==➔ arrow symbol can see predicted JOB title as Big data Engineer or Cloud Architect

6. VALIDATION

Implementation and Testing

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

Implementation

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifies as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

System Testing

Testing has become an System integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to user the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

Module Testing

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

Test Case ID	Test case Name	TestCase Desc.	Step	Expected	Actual	Test Case Status	Test Case Priority
01	Calculate Metrics	Verify The calculate Metrics done or not	If The calculate Metrics may not done	We cannot do the further operations	We Can do further operations	High	High
02	Train LR	Verify the Train LR done or not	If the Train LR May Not be done	We cannot do the further operations	We Can do further operations	High	High
03	Train SVM	Verify The Train SVM done or not	If The Train SVM may not done	We cannot do the further operations	We Can do further operations	High	High
04	Train NB	Verify The Train NB done or not	If The Train NB may not done	We cannot do the further operations	We Can do further operations	High	High
05	Train CNN	Verify Train CNN done or not	If The Train CNN may not done	We cannot do the further operations	We Can do further operations	High	High
06	Train BERT	Verify Train BERT done or not	If The Train BERT may not done	We cannot do the further operations	We Can do further operations	High	High
07	graph	Verify The graph done or not	If The graph may not done	We cannot.	We Can do further operations	High	High

Integration Testing

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

Acceptance Testing

When that user fined no major problems with its accuracy the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

7. CONCLUSION & FUTURE ASPECTS

In this paper, we present a two-stage job tile identification methodology based on semi-supervised and unsupervised machine learning algorithms with minimal labeling. In particular, for each job ad based on similarity measures, we find the most appropriate occupation using a standard occupational classifier. During the conducted experiments and after pre-processing the collected job ads, we tested several word and document representation methods such as TFIDF, neural language models that rely on distributional semantics (Word2Vec, FastText), and deep contextualized word representation (BERT). They were all subjected to several weighting strategies in order to reduce the impact of irrelevant words, especially in the description. Then, we tested various balance factors to identify the degree of contribution of both the title and the description to the process. According to the experiment results, classifying the job ads by sector improved the accuracy of our methodology by 14% since the similarity measures between the job ad and the occupations will be applied only within the predicted sector instead of using all the occupations from the referential. For document representation, we found that results using W2V outperformed BERT since there is a difference in vocabulary between the training dataset and job vacancies. However, in the case where the sector is not specified, we found that BERT provides the most accurate results. When it comes to weighting strategies, results show that uniform and frequency word weighting work best for short text (job ad titles, occupation titles), as these are not sensitive to word weighting, while the TFIDF weighting strategy for long text (job ad descriptions, occupation descriptions) significantly improves performance. In addition, we found that document embedding using only the top N selective words from the description using weighting scores gives the most accurate results among all the configurations we tested since we add relevant context to the title. Finally, experiments also verify the effectiveness of using both the title and the description in the matching process. They also verify that we should not give them equal weights because the title is more relevant since it contains more dense words related to the job. These findings helped us improve the accuracy of our comparable

to those obtained by the classification approach. Specifically, we obtained an overall accuracy of 76.5%, which can sometimes exceed 85% depending on the sector, such as the health sector and hotel & tourism sector. Furthermore, these findings can also be applied to improve the accuracy of the classifier when considering the task of job title identification as a classification problem. Finally, this methodology can be replicated in other languages using other occupation classifiers with minimal interaction to normalize the job ads and get insights from them. The proposed technique has been tested in a real-life setting framed within the project called ‘‘Data science for improved education and employment in Morocco’’ supported by USAID which aims at analyzing the job market needs and extracting skills from them [4]. It can also be applied in the process of defining training courses by universities based on job market needs. At the same time, youth and job seekers looking for employment can benefit from the results of studies using this methodology to analyze the labor market. In the future, we intend to add a step of job enrichment with skills terms based on the occupation description so that the job ad and occupation description are as similar as possible because recruiters do not follow a specific format when writing job advertisements. We also intend to do more cleaning of the list of top N words generated by weighting strategies to keep only relevant words. Furthermore, we plan to train our own Word2Vec model on sentences related to jobs in French, which may increase the accuracy of our methodology.

8. BIBLIOGRAPHY

8.1 REFERENCES

- F. Javed, Q. Luo, M. McNair, F. Jacob, M. Zhao, and T. S. Kang, ‘‘Carotene: A job title classification system for the online recruitment domain,’’ in Proc. IEEE 1st Int. Conf. Big Data Comput. Service Appl., Mar. 2015, pp. 286–293.
- M. S. Pera, R. Qumsiyeh, and Y.-K. Ng, ‘‘Web-based closed-domain data extraction on online advertisements,’’ Inf. Syst., vol. 38, no. 2, pp. 183–197, Apr. 2013.
- R. Kessler, N. Béchet, M. Roche, J.-M. Torres-Moreno, and M. El-Bèze, ‘‘A hybrid approach to managing job offers and candidates,’’ Inf. Process. Manage., vol. 48, no. 6, pp. 1124–1135, Nov. 2012.
- I. Rahhal, K. Carley, K. Ismail, and N. Sbihi, ‘‘Education path: Student orientation based on the job market needs,’’ in Proc. IEEE Global Eng. Educ. Conf. (EDUCON), Mar. 2022, pp. 1365–1373.
- S. Mittal, S. Gupta, K. Sagar, A. Shamma, I. Sahni, and N. Thakur, ‘‘A performance comparisons of machine learning classification techniques for job titles using job descriptions,’’ SSRN Electron. J., 2020. Accessed: Feb. 22, 2023. [Online]. Available: https://www.ssrn.com/abstract=3589962, doi: 10.2139/ssrn.3589962.
- R. Boselli, M. Cesarini, F. Mercorio, and M. Mezzanzanica, ‘‘Using machine learning for labour market intelligence,’’ in Machine Learning and Knowledge Discovery in Databases (Lecture Notes in Computer Science), Y. Altun, K. Das, T. Mielikäinen, D. Malerba, J. Stefanowski, J. Read, M. Zitnik, M. Ceci, and S. Dzeroski, Eds. Cham, Switzerland: Springer, 2017, pp. 330–342.
- T. Van Huynh, K. Van Nguyen, N. L.-T. Nguyen, and A. G.-T. Nguyen, ‘‘Job prediction: From deep neural network models to applications,’’ in Proc. RIVF Int. Conf. Comput. Commun. Technol. (RIVF), Oct. 2020, pp. 1–6.

- F. Amato, R. Boselli, M. Cesarini, F. Mercorio, M. Mezzanzanica, V. Moscato, F. Persia, and A. Picariello, ''Challenge: Processing web texts for classifying job offers,'' in Proc. IEEE 9th Int. Conf. Semantic Comput. (IEEE ICSC), Feb. 2015, pp. 460–463.
- H. T. Tran, H. H. P. Vo, and S. T. Luu, ''Predicting job titles from job descriptions with multi-label text classification,'' in Proc. 8th NAFOSTED Conf. Inf. Comput. Sci. (NICS), Dec. 2021, pp. 513–518.
- R. Boselli, M. Cesarini, F. Mercorio, and M. Mezzanzanica, ''Classifying online job advertisements through machine learning,'' Future Gener. Comput. Syst., vol. 86, pp. 319–328, Sep. 2018.
- M. Vinel, I. Ryazanov, D. Botov, and I. Nikolaev, ''Experimental comparison of unslupervised approaches in the task of separating specializations within professions in job vacancies,'' in Proc. Conf. Artif. Intell. Natural Lang., Cham, Switzerland: Springer, 2019, pp. 99–112.
- E. Malherbe, M. Cataldi, and A. Ballatore, ''Bringing order to the job market: Efficient job offer categorization in E-recruitment,'' in Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr., Aug. 2015, pp. 1101–1104.
- F. Saberi-Movahed, M. Rostami, K. Berahmand, S. Karami, P. Tiwari, M. Oussalah, and S. S. Band, ''Dual regularized unsupervised feature selection based on matrix factorization and minimum redundancy with application in gene selection,'' Knowl.-Based Syst., vol. 256, Nov. 2022, Art. no. 109884.
- I. Khaouja, I. Rahhal, M. Elouali, G. Mezzour, I. Kassou, and K. M. Carley, ''Analyzing the needs of the offshore sector in Morocco by mining job ads,'' in Proc. IEEE Global Eng. Educ. Conf. (EDUCON), Apr. 2018, pp. 1380–1388.
- R. Bekkerman and M. Gavish, ''High-precision phrase-based document classification on a modern scale,'' in Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2011, pp. 231–239.

8.2 GITHUB LINK

https://github.com/SURAJGUDELLY/Two-Stage-Job-Title-Identification-System-for-Online-Job-Advertisments