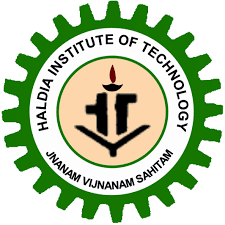
****

HALDIA INSTITUTE OF TECHNOLOGY

**PROJECT REPORT**

**Fake job post detection by applying deep learning approach**

Project Mentors :- Dr. Bidesh Chakraborty

& Mr. Rajesh Mukharjee

Name Roll No Univ. Roll

Ankit Gupta 18/CSE/128 10300118128

Ankit Kumar Singh 18/CSE/127 10300118127

Prince Kr. Singh 18/CSE/085 10300118085

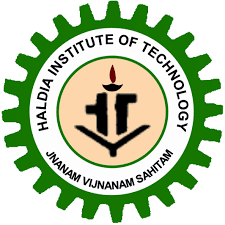
Divakar kumar 18/CSE/113 10300118113

**Group-24**

**Mentors Sign.**

**Fake job post detection by applying deep learning approach**

***Report submitted to Haldia Institute of Technology, Haldia for the award of the degree***



***Bachelor of Technology***

***Of***

***Computer Science & Engineering***

***By***

**Ankit Gupta ,**

**Ankit Kumar Singh ,**

**Divakar Kumar ,**

**Prince Kumar Singh**

**DEPARTMENT OFCOMPUTER SCIENCE & ENGINEERING**

**HALDIA INSTITUTE OF TECHNOLOGY, HALDIA**

**MAY2022**

**DECLARATION**

I/We certify that

a. The work contained in this report is original and has been done by me/us underthe guidance of my/our supervisor(s).

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I/We have followed the guidelines provided by the Institute in preparing the report.

d. I/We have conformed to the norms and guidelines given in the Ethical

Code of Conduct of the Institute.

e. Whenever I/we have used materials (data, theoretical analysis, figures, and text)from other sources, I / we have given due credit to them by citing them in the text of the report and giving their details in the references.

Signature of the Students

**CERTIFICATE**

This is to certify that the Dissertation Report entitled, “**Fake job post detection by applying deep learning or appropriate approach**” submitted by **Mr./Ms.** “**Ankit Gupta, Ankit Kumar Singh, Prince Kumar Singh, Divakar kumar**” to Haldia Institute of Technology, Haldia, India, is a record of bonafide Project work carried out by him/her under my/our supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology in Computer Science and

Engineering of the Institute.

Supervisor Supervisor

Date:

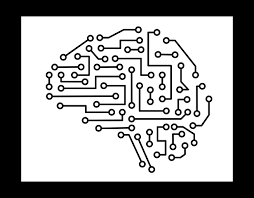
**ACKNOWLEDGEMENT**

It is a great pleasure for me to undertake this report. I feel highly overwhelmed doing the report on Studying simple Deep learning algorithms; their working principles; implementations and applications.

We would like to express my special thanks and gratitude to our project mentor “Dr. Bidesh Chakraborty & Mr. Rajesh Mukharjee”, who gave us the golden opportunity to do this wonderful project. We came to know about so many things we are really thankful to him.

Although, this report has been prepared with utmost care and deep routed interest yet We accept respondent and imperfection. **ABSTRACT**

This project proposes an application that uses a Deep learning-based classification technique to avoid fraudulent job listings on the Internet. Use different classifiers to check for fraudulent posts on the web and compare the results of these classifiers to identify the best model for detecting employment fraud. Helps identify fake classified ads from a huge number of positions. There are two main types of classifiers that can be used to detect fraudulent job listings. B. Single classifier and ensemble classifier. However, experimental results show that ensemble classifiers are the best classification for finding fraud on individual classifiers. Many people take advantage of the despair caused by unprecedented incidents to fall victim to these scammers. Most scammers do this to get personal information from the cheating person. Personal data includes addresses, bank details, social security numbers, and more. I am a student and have received some such fraudulent emails. Scammers offer users a very lucrative job opportunity and later demand money for it.

****

**Introduction**

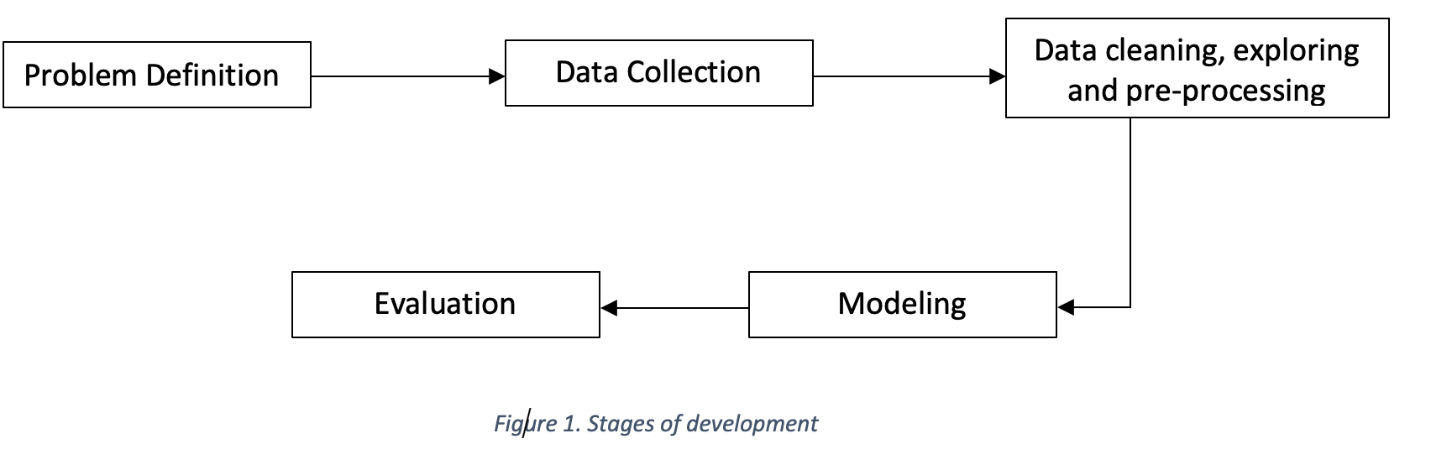
***Project Overview :***

Employment fraud is on the rise. According to CNBC, the number of fraud cases doubled in 2018 compared to 2017. Current market conditions have led to high unemployment. Due to financial stress and the effects of the coronavirus, job availability and unemployment for many have been significantly reduced. Such cases present fraudulent opportunities. Many people take advantage of the despair caused by unprecedented incidents to fall victim to these scammers. Most scammers do this to get personal information from the cheating person. Personal data includes addresses, bank details, social security numbers, and more. I am a student and have received some such fraudulent emails. Scammers offer users a very lucrative job opportunity and later demand money for it. Or they ask the job seeker to invest in a job promise. This is a dangerous problem that machine learning and natural language processing (NLP) technologies can address.

Malicious flag of fake news on social media User account, echo chamber effect. the Basic research for detecting fake news is based on Three perspectives on how to write fake news Fake news spreads how users relate to fake news. Features related to news content and social context Extracted and machine learning model Charged to detect fake news

This project uses Kaggle data. This data contains properties that define job listings. These job listings are classified as genuine or fake. Fake job listings are just a small part of this dataset. This is an exception. I don't expect much fake job listings. This project follows five phases. The five stages adopted for this project are –

1. Problem Definition (Project Overview, Project statement and Metrics)
2. Data Collection
3. Data cleaning, exploring and pre-processing
4. Modeling
5. Evaluating



**LITERATURE SURVEY**

According to several studies, Review spam detection, Email Spam detection, Fake news detection have drawn special attention in the domain of Online Fraud Detection. People often post their reviews online forum regarding the products they purchase. It may guide other purchaser while choosing their products. In this context, spammers can manipulate reviews for gaining profit and hence it is required to develop techniques that detects these spam reviews. This can be implemented by extracting features from the reviews by extracting features using Natural Language Processing (NLP). Next, machine learning techniques are applied on these features. Lexicon based approaches may be one alternative to machine learning techniques that uses dictionary or corpus to eliminate spam reviews[11].

B. Email Spam Detection

Unwanted bulk mails, belong to the category of spam emails, often arrive to user mailbox. This may lead to unavoidable storage crisis as well as bandwidth consumption. To eradicate this problem, Gmail, Yahoo mail and Outlook service providers incorporate spam filters using Neural Networks. While addressing the problem of email spam detection, content- based filtering, case-based filtering, heuristic based filtering, memory or instance-based filtering, adaptive spam filtering approaches are taken into consideration [7].

C. Fake News Detection

Fake news in social media characterizes malicious user accounts, echo chamber effects. The fundamental study of fake news detection

relies on three perspectives- how fake news is written, how fake news spreads, how a user is related to fake news. Features related to news content and social context are extracted and a machine learning model are imposed to recognize fake news.

**Disadvantages:**

• In recent days, many companies prefer to post their vacancies online so that these can be accessed easily and timely by the job-seekers. However, this intention may be one type of scam by the fraud people because they offer employment to job-seekers in terms of taking money from them.

**Advantages:**

• Machine learning approach is applied which employs several classification algorithms

for recognizing fake posts. In this case, a classification tool isolates fake job posts from a

larger set of job advertisements and alerts the user.

**Analysis**

**Data Exploration :**

The data for this project is available at Kaggle - <https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction>. The dataset consists of 17,880 observations and 18 features.

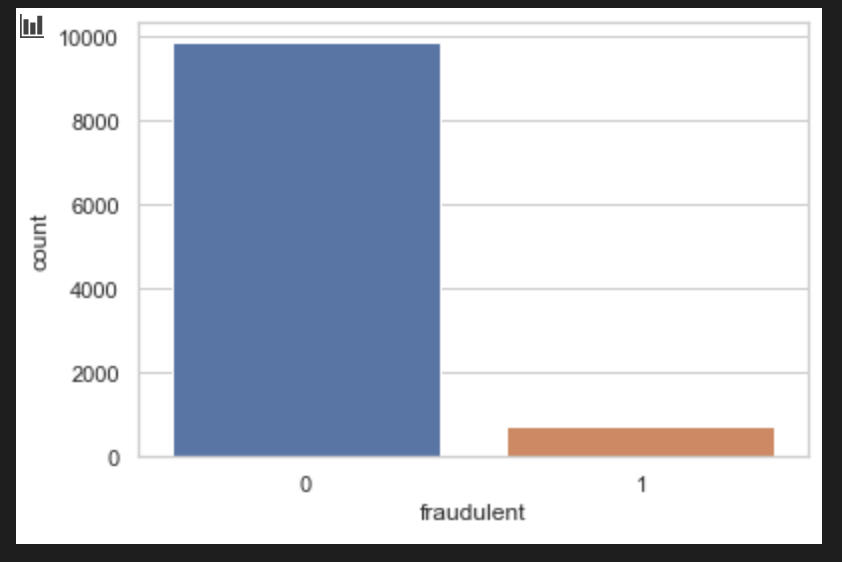
The data is combination of integer, binary and textual datatypes. A brief definition of the variables is given below:

|  |  |  |  |
| --- | --- | --- | --- |
| # | Variable | DataType | Description |
| 1 | job\_id | Int | Identification number given to each job posting |
| 2 | Title | Text | A name that describes the position or job |
| 3 | Location | Text | Information about where the job is located |
| 4 | Department | Text | Information about the department this job is offered by |
| 5 | Salary\_range | Text | Expected salary range |
| 6 | Company\_profile | Text | Information about the company |
| 7 | Description | Text | A brief description about the position offered |
| 8 | Requirements | Text | Pre-requisites to qualify for the job |
| 9 | Benefits | Text | Benefits provided by the job |
| 10 | Telecommuting | Boolean | Is work from home or remote work allowed |
| 11 | Has\_company\_logo | Boolean | Does the job posting have a company logo |
| 12 | Employment\_type | Text | 5 categories – Full-time, part-time, contract, temporary and other |
| 13 | Required\_experience | Text | Can be – Internship, Entry Level, Associate, Mid-senior level, Director, Executive or Not Applicable |
| 14 | Required\_education | Text | Can be – Bachelor’s degree, high school degree, unspecified, associate degree, master’s degree, certification, some college coursework, professional, some high school coursework, vocational |
| 15 | Industry | Text | The industry the job posting is relevant to |
| 16 | Function | Text | The umbrella term to determining a job’s functionality |
| 17 | Fraudulent | Boolean | The target variable  0: Real, 1: Fake |
| 18 | Has\_questions | Boolean | Does the job posting have any questions |

Since most of the datatypes are either Booleans or text a summary statistic is not needed here. The only integer is job\_id which is not relevant for this analysis. The dataset is further explored to identify null values.



Variables such as department and salary range have many missing values. These columns will be removed from subsequent analysis. Initial evaluation of the dataset revealed that these job listings were extracted from several countries and that the job listings were written in different languages. To simplify the process, this project uses data from US-based locations that make up almost 60% of the dataset. This was done to ensure that all data is in English for ease of interpretation. In addition, the location is divided into states and cities for further analysis. The final dataset contains 10593 observations and 20 characteristics. The records are very disproportionate, with 9868 (93% of jobs) being genuine and only 725 or 7% of fraudulent jobs. Count plots of the same can show parallax very clearly.



**Algorithms and Techniques**

Machine learning approach is applied which employs several classification algorithms for recognizing fake posts. In this case, a classification tool isolates fake job posts from a larger set of job advertisements and alerts the user. To address the problem of identifying scams on job posting, supervised learning algorithm as classification techniques are considered initially. A classifier maps input variable to target classes by considering training data. Classifiers addressed in the paper for identifying fake job posts from the others are described briefly. These classifiers based prediction may be broadly categorized into -Single Classifier based Prediction and Ensemble Classifiers based Prediction.

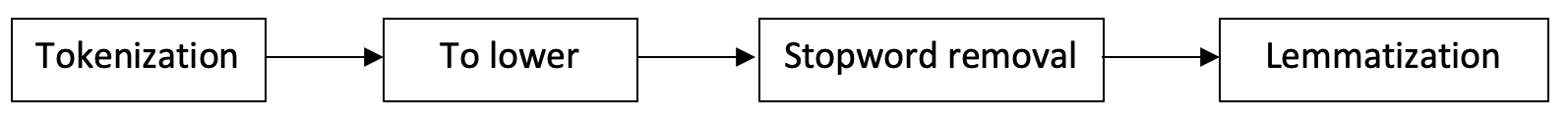
Deep Learning Approach : We can use CNN(convolutional neural network) The main strengths of CNNs are to provide an efficient dense network which performs the prediction or identification etc. efficiently. CNNs are the most popular topic in the pool of deep learning, which is indeed very vast, and this is usually because of the ConvNets. Immense datasets are applied to CNNs, it is even considered that larger the data, greater the accuracy will result, otherwise other operations such as transfer learning shall be applied to expand the data. The power of CNN is to detect distinct features from images all by itself, without any actual human intervention. The most popular dataset that CNN picks the features from are the Cats and Dogs dataset where each feature is picked automatically and the pictures are classified as dogs or cats.

**Methodology**

**Data Preprocessing:**

The following steps are taken for text processing:

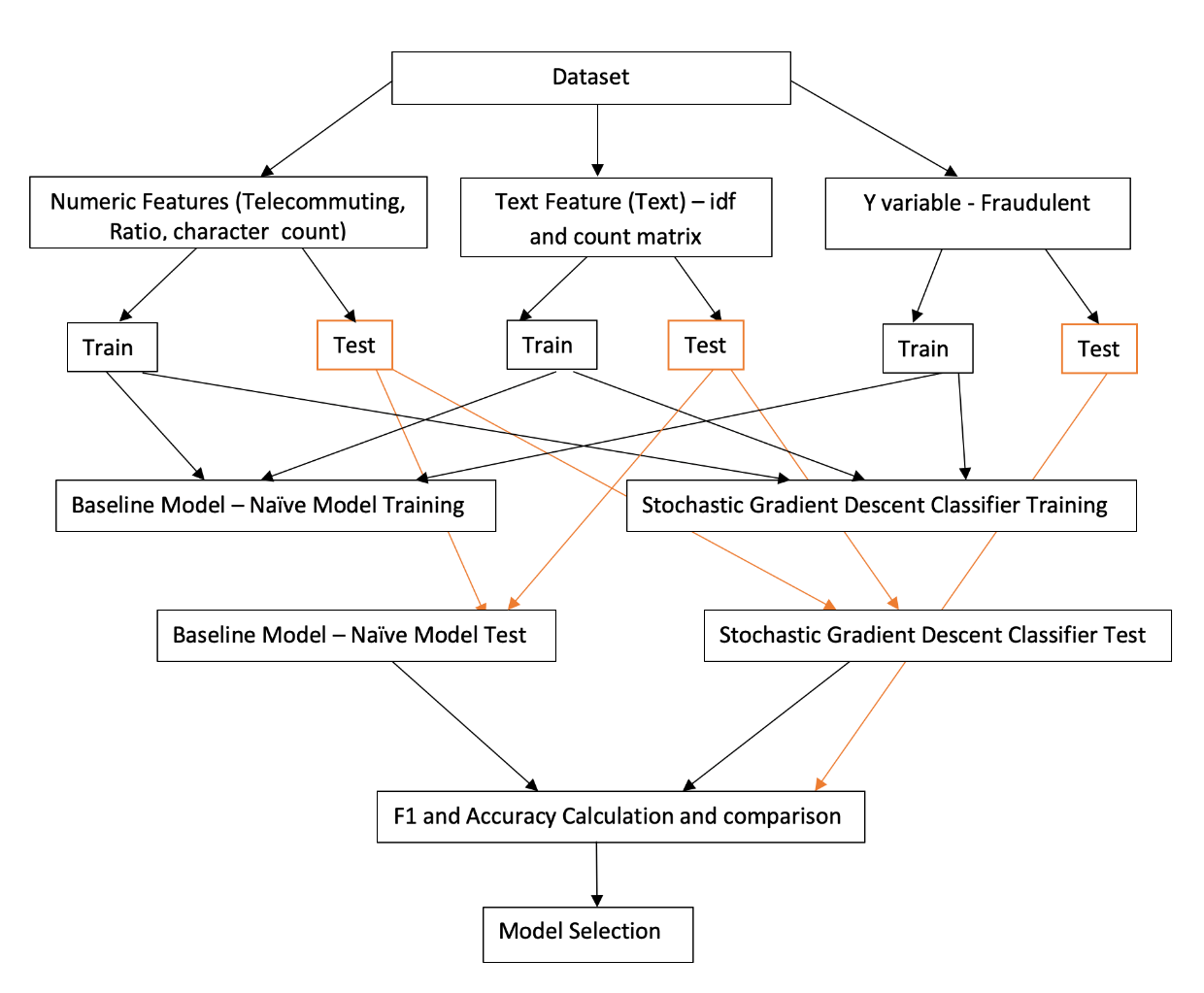
* Tokenization: The textual data is split into smaller units. In this case the data is split into words.
* To Lower: The split words are converted to lowercase
* Stopword removal: Stopwords are words that do not add much meaning to sentences. For example: the, a, an, he, have etc. These words are removed.
* Lemmatization: The process of lemmatization groups in which inflected forms of words are used together.



A diagrammatic representation of the implementation of this project is given below. The dataset is split into text, numeric and y-variable. The text dataset is converted into a term-frequency matrix for further analysis. Then using sci-kit learn, the datasets are split into test and train datasets. The baseline model Naïve bayes and another model SGD is trained on the using the train set which is 70% of the dataset. The final outcome of the models based on two test sets – numeric and text are combined such that if both models say that a particular data point is not fraudulent only then a job posting is fraudulent. This is done to reduce the bias of Machine Learning algorithms towards majority classes. The trained model is used on the test set to evaluate model performance. The Accuracy and F1-score of the two models – Naïve bayes and SGD are compared and the final model for our analysis is selected.

**Implementation**

A diagrammatic representation of the implementation of this project is given below. The dataset is split into text, numeric and y-variable. The text dataset is converted into a term-frequency matrix for further analysis. Then using sci-kit learn, the datasets are split into test and train datasets. The baseline model Naïve bayes and another model SGD is trained on the using the train set which is 70% of the dataset. The final outcome of the models based on two test sets – numeric and text are combined such that if both models say that a particular data point is not fraudulent only then a job posting is fraudulent. This is done to reduce the bias of Machine Learning algorithms towards majority classes. The trained model is used on the test set to evaluate model performance. The Accuracy and F1-score of the two models – Naïve bayes and SGD are compared and the final model for our analysis is selected.



**\*\*\*\*\*Cleaning and Analysis of the Data Set :\*\*\*\*\*\***

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

sns.set\_theme(style="whitegrid")

fake\_job\_postings = pd.read\_csv('data/fake\_job\_postings.csv')

fake\_job\_postings.describe()

fake\_job\_postings.info()

fake\_job\_postings.isnull().sum()

fake\_job\_postings.location = fake\_job\_postings.location.fillna('blank')

fake\_job\_postings\_US = fake\_job\_postings[fake\_job\_postings['location'].str.contains("US")]

loc\_split =[]

for loc in fake\_job\_postings\_US.location:

loc\_split.append(loc.split(','))

loc\_split = pd.DataFrame(loc\_split)

loc\_split = loc\_split[[1, 2]]

loc\_split = loc\_split.rename(columns={1: "state", 2:'city'})

len(fake\_job\_postings\_US)/len(fake\_job\_postings)

fake\_job\_postings\_US = fake\_job\_postings\_US.reset\_index()

fake\_job\_postings\_US = fake\_job\_postings\_US.join(loc\_split)

fake\_job\_postings\_US = fake\_job\_postings\_US[['job\_id', 'title', 'location', 'department', 'salary\_range',

'company\_profile', 'description', 'requirements', 'benefits',

'telecommuting', 'has\_company\_logo', 'has\_questions', 'employment\_type',

'required\_experience', 'required\_education', 'industry', 'function',

'fraudulent', 'state', 'city']]

fake\_job\_postings\_US = fake\_job\_postings\_US[fake\_job\_postings\_US['city'].notna()]

fake\_job\_postings\_US = fake\_job\_postings\_US[fake\_job\_postings\_US['state'].notna()]

fake\_job\_postings\_US.shape

fake\_job\_postings\_US['state\_city'] = fake\_job\_postings\_US['state'] + ", " + fake\_job\_postings\_US['city']

fake\_job\_postings\_US.isna().sum()

fake\_job\_postings\_US.city = fake\_job\_postings\_US.city.str.strip()

fake\_job\_postings\_US.state = fake\_job\_postings\_US.state.str.strip()

fake\_job\_postings\_US

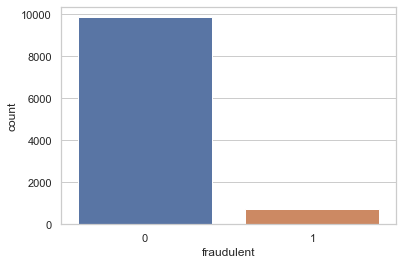
corr = fake\_job\_postings\_US.corr()

sns.heatmap(corr)

plt.show()

len(fake\_job\_postings\_US[fake\_job\_postings\_US.fraudulent == 0]), len(fake\_job\_postings\_US[fake\_job\_postings\_US.fraudulent == 1]),

sns.countplot(x='fraudulent', data=fake\_job\_postings\_US);



def sns\_countplot(feature):

sns.countplot(x=feature, data=fake\_job\_postings\_US, hue="fraudulent",

order=fake\_job\_postings\_US[feature].value\_counts().iloc[:10].index)

plt.xticks(rotation=90)

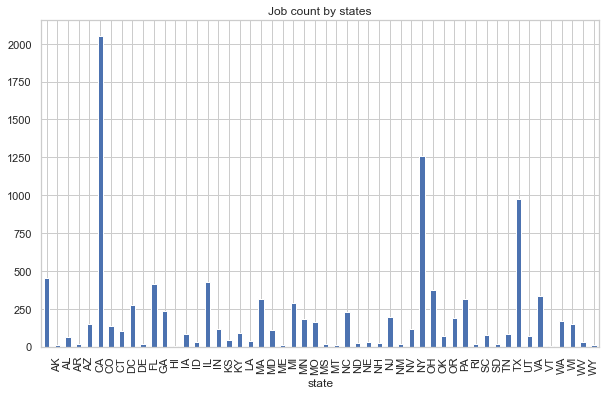
title = feature + ' fake job count'

plt.title('Location Fake Jobs')

plt.show()

plt.figure(figsize=(10,6))

fake\_job\_postings\_US.groupby('state').fraudulent.count().plot(kind='bar', title='Job count by states');



plt.figure(figsize=(10,6))

sns.countplot(x='state', data=fake\_job\_postings\_US, hue="fraudulent", order=fake\_job\_postings\_US['state'].value\_counts().iloc[:10].index)

plt.xticks(rotation=90)

plt.show()

sns.countplot(x='state\_city', data=fake\_job\_postings\_US, hue="fraudulent", order=fake\_job\_postings\_US['state\_city'].value\_counts().iloc[:10].index)

plt.xticks(rotation=90)

plt.show()

def sns\_countplot(feature):

sns.countplot(x=feature, data=fake\_job\_postings\_US, hue="fraudulent",

order=fake\_job\_postings\_US[feature].value\_counts().iloc[:10].index)

plt.xticks(rotation=90)

title = feature + ' fake job count'

plt.title(title)

plt.show()

location\_ratio = round(fake\_job\_postings\_US[fake\_job\_postings\_US.fraudulent == 1].groupby('state\_city').state\_city.count()/fake\_job\_postings\_US[fake\_job\_postings\_US.fraudulent == 0].groupby('state\_city').state\_city.count(), 2)

location\_ratio = pd.DataFrame({'state\_city':location\_ratio.index, 'ratio':location\_ratio.values})

fake\_job\_postings\_US = fake\_job\_postings\_US.merge(location\_ratio)

fake\_job\_postings\_US.ratio.fillna(0, inplace=True)

location\_ratio\_plot = location\_ratio[location\_ratio.ratio >= 1]

sns.barplot(data=location\_ratio\_plot.sort\_values(by='ratio'), x='state\_city', y='ratio')

plt.xticks(rotation=90)

plt.title('Fake to Real Job Ratio')

plt.show()

def missing\_count(feature, title='None'):

y\_axis = fake\_job\_postings\_US[fake\_job\_postings\_US[feature].isna()][['fraudulent', feature]]

y\_axis = y\_axis.fraudulent.value\_counts()

y\_axis.plot(kind='bar')

plt.ylabel('Count')

plt.xlabel('Category')

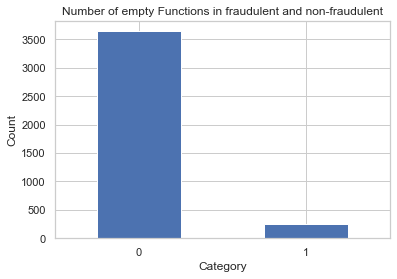
title = "Number of empty " + title + " in fraudulent and non-fraudulent"

plt.title(title)

plt.xticks(rotation=0)

plt.show()

return 0



telecommuting\_list = []

has\_company\_logo\_list = []

for idx, tel, logo in zip(range(len(fake\_job\_postings\_US)), fake\_job\_postings\_US.telecommuting, fake\_job\_postings\_US.has\_company\_logo):

if fake\_job\_postings.fraudulent[idx] == 1:

telecommuting\_list.append(tel)

has\_company\_logo\_list.append(logo)

else:

pass

telecommuting\_logo\_df = pd.DataFrame({'telecommuting':telecommuting\_list, 'has\_company\_logo':has\_company\_logo\_list})

fake\_count = 0

for fraud, tel, logo in zip(fake\_job\_postings\_US.fraudulent, fake\_job\_postings\_US.telecommuting, fake\_job\_postings\_US.has\_company\_logo):

if (tel == 0 and logo == 0):

if (fraud == 1):

fake\_count +=1

else:

pass

else:

pass

print(fake\_count)

fake\_count = 0

for fraud, tel, logo, ques in zip(fake\_job\_postings\_US.fraudulent, fake\_job\_postings\_US.telecommuting, fake\_job\_postings\_US.has\_company\_logo, fake\_job\_postings\_US.has\_questions):

if (tel == 0):# and logo == 0 and ques == 0):

if (fraud == 1):

fake\_count +=1

else:

pass

else:

pass

print(fake\_count)

fake\_count/len(fake\_job\_postings\_US[fake\_job\_postings\_US.fraudulent == 1]) \* 100

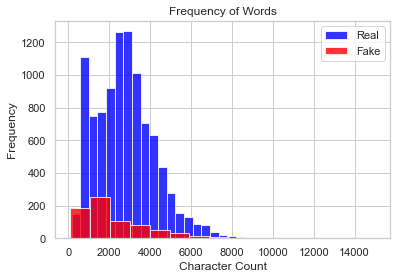
fake\_job\_postings\_US.fillna(" ",inplace = True)

fake\_job\_postings\_US['text'] = fake\_job\_postings\_US['title'] + ' ' + fake\_job\_postings\_US['location'] + ' ' + fake\_job\_postings\_US['company\_profile'] + ' ' + \

fake\_job\_postings\_US['description'] + ' ' + fake\_job\_postings\_US['requirements'] + ' ' + fake\_job\_postings\_US['benefits'] + ' ' + \

fake\_job\_postings\_US['required\_experience'] + ' ' + fake\_job\_postings\_US['required\_education'] + ' ' + fake\_job\_postings\_US['industry'] + ' ' + fake\_job\_postings\_US['function']

fake\_job\_postings\_US.drop(['job\_id', 'department', 'salary\_range', 'title','location','department','company\_profile','description','requirements','benefits','employment\_type','required\_experience','required\_education','industry','function', 'city', 'state\_city', 'has\_company\_logo', 'has\_questions', 'state'], axis = 1, inplace = True)



**\*\*\*\*\*\*\*\*\*\*\*\*Model Training Using Deep Learning\*\*\*\*\*\*\*\*\*\*\***

import torch

import re

from sklearn.feature\_extraction.text import TfidfVectorizer, TfidfTransformer, CountVectorizer

import numpy as np

import pandas as pd

from copy import deepcopy

SMALL\_TEXT\_VECTOR\_SIZE = 10

MEDIUM\_TEXT\_VECTOR\_SIZE = 25

LARGE\_TEXT\_VECTOR\_SIZE = 50

class JobPosting:

def \_\_init\_\_(self, title="", location="", department="", salary\_range="", company\_profile="",

description="", requirement="", benefit="", telecommuting="", has\_company\_logo="",

has\_question="", employment\_type="", required\_experience="",

required\_education="", industry="", function="", fraudulent="job\_pridiction"):

self.title = title if isinstance(title, str) else ""

self.location = location if isinstance(location, str) else ""

self.department = department if isinstance(department, str) else ""

self.salary\_range = [salary\_range if isinstance(salary\_range, str) else ""]

self.company\_profile = company\_profile if isinstance(company\_profile, str) else ""

self.description = description if isinstance(description, str) else ""

self.requirement = requirement if isinstance(requirement, str) else ""

self.benefit = benefit if isinstance(benefit, str) else ""

self.telecommuting = telecommuting if isinstance(telecommuting, str) else ""

self.has\_company\_logo = has\_company\_logo if str(isinstance(has\_company\_logo, int)) else ""

self.has\_question = has\_question if isinstance(has\_question, str) else ""

self.employment\_type = employment\_type if isinstance(employment\_type, str) else ""

self.required\_experience = required\_experience if isinstance(required\_experience, str) else ""

self.required\_education = required\_education if isinstance(required\_education, str) else ""

self.industry = industry if isinstance(industry, str) else ""

self.function = function if isinstance(function, str) else ""

self.fraudulent = fraudulent if isinstance(fraudulent, str) else ""

def get\_data\_list(self):

return [\*self.title, \*self.location, \*self.department, \*self.salary\_range, \*self.company\_profile,

\*self.description, \*self.requirement, \*self.benefit, self.telecommuting, self.has\_company\_logo,

self.has\_company\_logo, self.has\_question, \*self.employment\_type, \*self.required\_experience,

\*self.required\_education, \*self.industry, \*self.function]

def get\_target\_list(self):

if int(self.fraudulent) == 0:

return [1.0, 0.0]

else:

return [0.0, 1.0]

def get\_target(self):

return int(self.fraudulent)

def get\_data\_tensor(self):

return torch.tensor(self.get\_data\_list())

class JobPostingsDataset(torch.utils.data.Dataset):

def \_\_init\_\_(self, job\_postings\_list=[]):

self.job\_postings\_list = job\_postings\_list

self.vectorized\_job\_postings\_dict = {}

def \_\_len\_\_(self):

return len(self.job\_postings\_list)

def prepare\_all\_text\_vectorizers(self):

self.title\_vectorizer = TfidfVectorizer(max\_features=MEDIUM\_TEXT\_VECTOR\_SIZE)

all\_titles\_list = [self.preprocess\_text(job\_posting.title) for job\_posting in self.job\_postings\_list]

self.title\_vectorizer.fit\_transform(all\_titles\_list)

self.location\_vectorizer = TfidfVectorizer(max\_features=MEDIUM\_TEXT\_VECTOR\_SIZE)

all\_locations\_list = [self.preprocess\_text(job\_posting.location) for job\_posting in self.job\_postings\_list]

self.location\_vectorizer.fit\_transform(all\_locations\_list)

self.department\_vectorizer = TfidfVectorizer(max\_features=SMALL\_TEXT\_VECTOR\_SIZE)

all\_departments\_list = [self.preprocess\_text(job\_posting.department) for job\_posting in self.job\_postings\_list]

self.department\_vectorizer.fit\_transform(all\_departments\_list)

self.company\_profile\_vectorizer = TfidfVectorizer(max\_features=LARGE\_TEXT\_VECTOR\_SIZE)

all\_company\_profiles\_list = [self.preprocess\_text(job\_posting.company\_profile) for job\_posting in self.job\_postings\_list]

self.company\_profile\_vectorizer.fit\_transform(all\_company\_profiles\_list)

self.description\_vectorizer = TfidfVectorizer(max\_features=LARGE\_TEXT\_VECTOR\_SIZE)

all\_descriptions\_list = [self.preprocess\_text(job\_posting.description) for job\_posting in self.job\_postings\_list]

self.description\_vectorizer.fit\_transform(all\_descriptions\_list)

self.requirement\_vectorizer = TfidfVectorizer(max\_features=LARGE\_TEXT\_VECTOR\_SIZE)

all\_requirements\_list = [self.preprocess\_text(job\_posting.requirement) for job\_posting in self.job\_postings\_list]

self.requirement\_vectorizer.fit\_transform(all\_requirements\_list)

self.benefit\_vectorizer = TfidfVectorizer(max\_features=LARGE\_TEXT\_VECTOR\_SIZE)

all\_benefits\_list = [self.preprocess\_text(job\_posting.benefit) for job\_posting in self.job\_postings\_list]

self.benefit\_vectorizer.fit\_transform(all\_benefits\_list)

self.employment\_type\_vectorizer = TfidfVectorizer(max\_features=SMALL\_TEXT\_VECTOR\_SIZE)

all\_employment\_types\_list = [self.preprocess\_text(job\_posting.employment\_type) for job\_posting in self.job\_postings\_list]

self.employment\_type\_vectorizer.fit\_transform(all\_employment\_types\_list)

self.required\_experience\_vectorizer = TfidfVectorizer(max\_features=SMALL\_TEXT\_VECTOR\_SIZE)

all\_required\_experiences\_list = [self.preprocess\_text(job\_posting.required\_experience) for job\_posting in self.job\_postings\_list]

self.required\_experience\_vectorizer.fit\_transform(all\_required\_experiences\_list)

self.required\_education\_vectorizer = TfidfVectorizer(max\_features=SMALL\_TEXT\_VECTOR\_SIZE)

all\_required\_educations\_list = [self.preprocess\_text(job\_posting.required\_education) for job\_posting in self.job\_postings\_list]

self.required\_education\_vectorizer.fit\_transform(all\_required\_educations\_list)

self.industry\_vectorizer = TfidfVectorizer(max\_features=MEDIUM\_TEXT\_VECTOR\_SIZE)

all\_industries\_list = [self.preprocess\_text(job\_posting.industry) for job\_posting in self.job\_postings\_list]

self.industry\_vectorizer.fit\_transform(all\_industries\_list)

self.function\_vectorizer = TfidfVectorizer(max\_features=MEDIUM\_TEXT\_VECTOR\_SIZE)

all\_functions\_list = [self.preprocess\_text(job\_posting.function) for job\_posting in self.job\_postings\_list]

self.function\_vectorizer.fit\_transform(all\_functions\_list)

@staticmethod

def preprocess\_text(text):

text = re.sub("<[^>]\*>", "", text)

symbols = re.findall("(?::|;|=)(?:-)?(?:\)|\(|D|P)", text)

text = (re.sub("[\W]+", " ", text.lower()) + " ".join(symbols).replace("-", ""))

return text

def \_\_getitem\_\_(self, index):

if index in self.vectorized\_job\_postings\_dict:

return self.vectorized\_job\_postings\_dict[index]

else:

vectorized\_job\_posting = deepcopy(self.job\_postings\_list[index])

vectorized\_job\_posting.title = self.title\_vectorizer.transform([vectorized\_job\_posting.title]).toarray()[0]

vectorized\_job\_posting.location = self.location\_vectorizer.transform([vectorized\_job\_posting.location]).toarray()[0]

vectorized\_job\_posting.department = self.department\_vectorizer.transform([vectorized\_job\_posting.department]).toarray()[0]

vectorized\_job\_posting.company\_profile = self.company\_profile\_vectorizer.transform([vectorized\_job\_posting.company\_profile]).toarray()[0]

vectorized\_job\_posting.description = self.description\_vectorizer.transform([vectorized\_job\_posting.description]).toarray()[0]

vectorized\_job\_posting.requirement = self.requirement\_vectorizer.transform([vectorized\_job\_posting.requirement]).toarray()[0]

vectorized\_job\_posting.benefit = self.benefit\_vectorizer.transform([vectorized\_job\_posting.benefit]).toarray()[0]

vectorized\_job\_posting.employment\_type = self.employment\_type\_vectorizer.transform([vectorized\_job\_posting.employment\_type]).toarray()[0]

vectorized\_job\_posting.required\_experience = self.required\_experience\_vectorizer.transform([vectorized\_job\_posting.required\_experience]).toarray()[0]

vectorized\_job\_posting.required\_education = self.required\_education\_vectorizer.transform([vectorized\_job\_posting.required\_education]).toarray()[0]

vectorized\_job\_posting.industry = self.industry\_vectorizer.transform([vectorized\_job\_posting.industry]).toarray()[0]

vectorized\_job\_posting.function = self.function\_vectorizer.transform([vectorized\_job\_posting.function]).toarray()[0]

if len(list(vectorized\_job\_posting.salary\_range[0].split("-"))) == 2:

try:

vectorized\_job\_posting.salary\_range = tuple(map(int, vectorized\_job\_posting.salary\_range[0].split("-")))

except:

vectorized\_job\_posting.salary\_range = (0, 0)

else:

vectorized\_job\_posting.salary\_range = (0, 0)

vectorized\_job\_posting.telecommuting = int(vectorized\_job\_posting.telecommuting)

vectorized\_job\_posting.has\_company\_logo = int(vectorized\_job\_posting.has\_company\_logo)

vectorized\_job\_posting.has\_question = int(vectorized\_job\_posting.has\_question)

vectorized\_job\_posting.fraudulent = int(vectorized\_job\_posting.fraudulent)

self.vectorized\_job\_postings\_dict[index] = vectorized\_job\_posting

return self.vectorized\_job\_postings\_dict[index]

import torch

from torch import nn, optim

from torch.nn import functional as F

NETWORK\_INPUT\_SIZE = 342

NETWORK\_OUTPUT\_SIZE = 2

class Network(nn.Module):

def \_\_init\_\_(self, input\_size=NETWORK\_INPUT\_SIZE, output\_size=NETWORK\_OUTPUT\_SIZE):

super(Network, self).\_\_init\_\_()

self.fc1 = nn.Linear(input\_size, 256)

self.fc2 = nn.Linear(256, 128)

self.fc3 = nn.Linear(128, 64)

self.fc4 = nn.Linear(64, 32)

self.fc5 = nn.Linear(32, 16)

self.fc6 = nn.Linear(16, 8)

self.fc7 = nn.Linear(8, 4)

self.fc8 = nn.Linear(4, output\_size)

def forward(self, x):

x = self.fc1(x)

x = F.relu(x)

x = self.fc2(x)

x = F.relu(x)

x = self.fc3(x)

x = F.relu(x)

x = self.fc4(x)

x = F.relu(x)

x = self.fc5(x)

x = F.relu(x)

x = self.fc6(x)

x = F.relu(x)

x = self.fc7(x)

x = F.relu(x)

x = self.fc8(x)

return x

def normal\_init(m, mean, std):

if isinstance(m, nn.Linear):

m.weight.data.normal\_(mean, std)

m.bias.data.zero\_()

import torch

from torch import nn, optim

import pandas as pd

import random

from pycm import ConfusionMatrix

from skorch import NeuralNetClassifier

from sklearn.model\_selection import cross\_val\_predict, cross\_val\_score

CSV\_FILENAME = "dataset.csv"

CSV\_FILE\_DELIMITER = ","

EPOCHS\_COUNT = 200

LEARNING\_RATE = 0.0001

BATCH\_SIZE = 32

KFOLD\_PARTITIONS\_COUNT = 5

FAKE\_TO\_REAL\_RATIO = 1.0/20.0

TRAINING\_PARALLEL\_JOBS\_COUNT = -1 # -1 means maximum

def prepare\_dataset():

global job\_postings

global all\_job\_posting\_data

global all\_job\_posting\_targets

random.shuffle(job\_postings)

job\_postings = JobPostingsDataset(job\_postings)

job\_postings.prepare\_all\_text\_vectorizers()

all\_job\_posting\_data = torch.tensor([job\_posting.get\_data\_list() for job\_posting in job\_postings]).float().to(device)

all\_job\_posting\_targets = torch.tensor([job\_posting.get\_target\_list() for job\_posting in job\_postings]).float().to(device)

def read\_job\_postings\_data\_from\_csv(csv\_filename=CSV\_FILENAME, csv\_file\_delimiter=CSV\_FILE\_DELIMITER):

global job\_postings

job\_postings = []

csv\_contents = pd.read\_csv(csv\_filename, delimiter=csv\_file\_delimiter, dtype=str)

rows = len(csv\_contents)

cols = len(csv\_contents.iloc[0])

for i in range(rows):

data = csv\_contents.iloc[i]

job\_posting = JobPosting(data["title"], data["location"], data["department"], data["salary\_range"], data["company\_profile"], data["description"], data["requirements"],

data["benefits"], data["telecommuting"], data["has\_company\_logo"], data["has\_questions"], data["employment\_type"], data["required\_experience"],

data["required\_education"], data["industry"], data["function"], data["fraudulent"])

job\_postings.append(job\_posting)

def initialize\_network():

global device

global model

global optimizer

global criterion

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

model = Network().to(device)

criterion = torch.nn.BCEWithLogitsLoss

optimizer = torch.optim.Adam

for m in model.\_modules:

normal\_init(model.\_modules[m], 0, 1)

def print\_confusion\_matrix(actual\_labels, predicted\_labels):

cm = ConfusionMatrix(actual\_labels, predicted\_labels)

print(cm)

if \_\_name\_\_ == "\_\_main\_\_":

print("Initializing network")

initialize\_network()

print("Reading CSV file")

read\_job\_postings\_data\_from\_csv()

print("Preparing dataset")

prepare\_dataset()

print("Creating classifier")

classifier = NeuralNetClassifier(Network, max\_epochs=EPOCHS\_COUNT, lr=LEARNING\_RATE,

train\_split=None, criterion=criterion, optimizer=optimizer,

batch\_size=BATCH\_SIZE, criterion\_\_pos\_weight=torch.tensor([FAKE\_TO\_REAL\_RATIO, 1.0]))

print("Training")

predictions = cross\_val\_predict(classifier, all\_job\_posting\_data, all\_job\_posting\_targets,

cv=KFOLD\_PARTITIONS\_COUNT, verbose=1, n\_jobs=TRAINING\_PARALLEL\_JOBS\_COUNT)

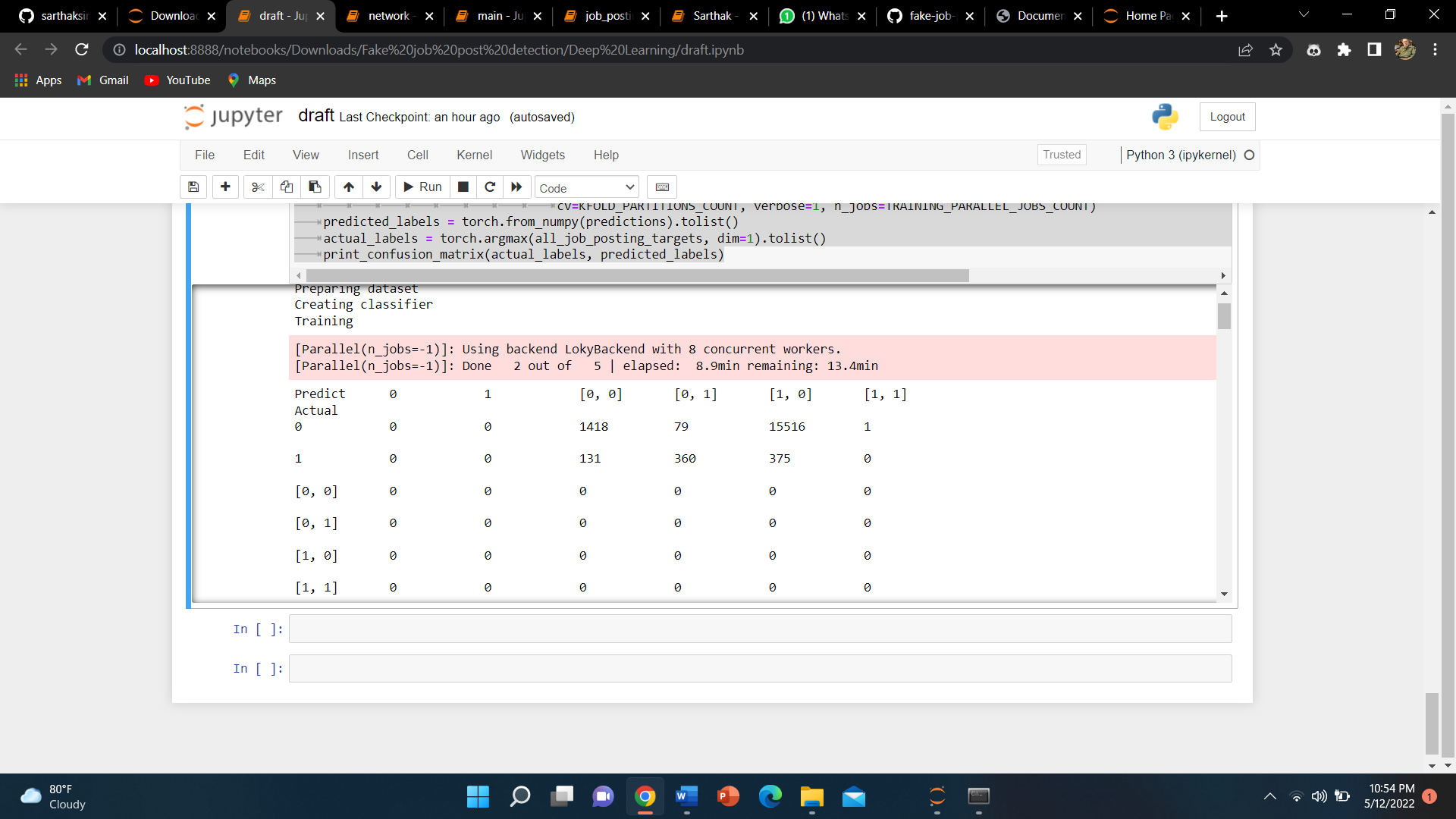
predicted\_labels = torch.from\_numpy(predictions).tolist()

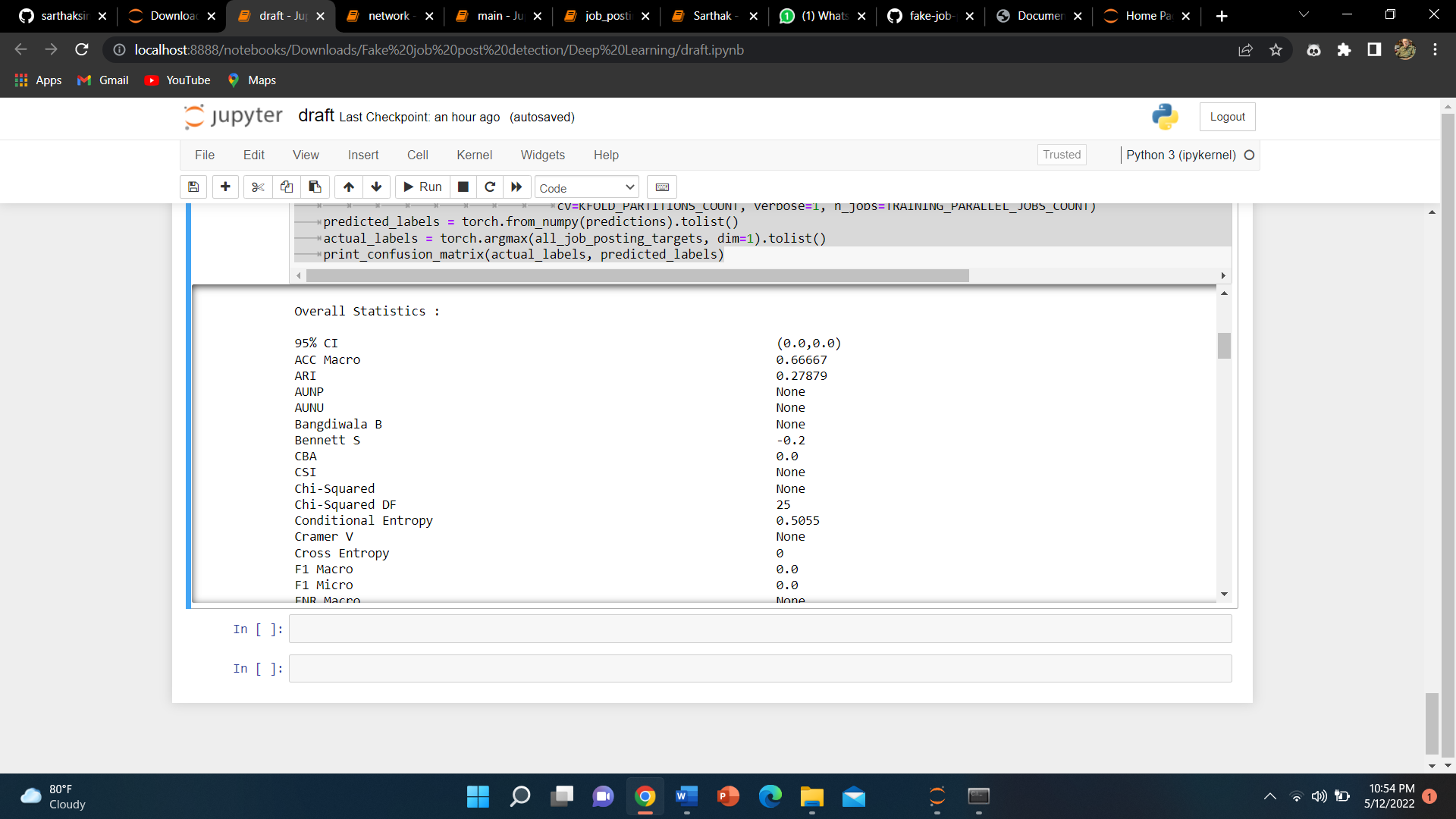
actual\_labels = torch.argmax(all\_job\_posting\_targets, dim=1).tolist()

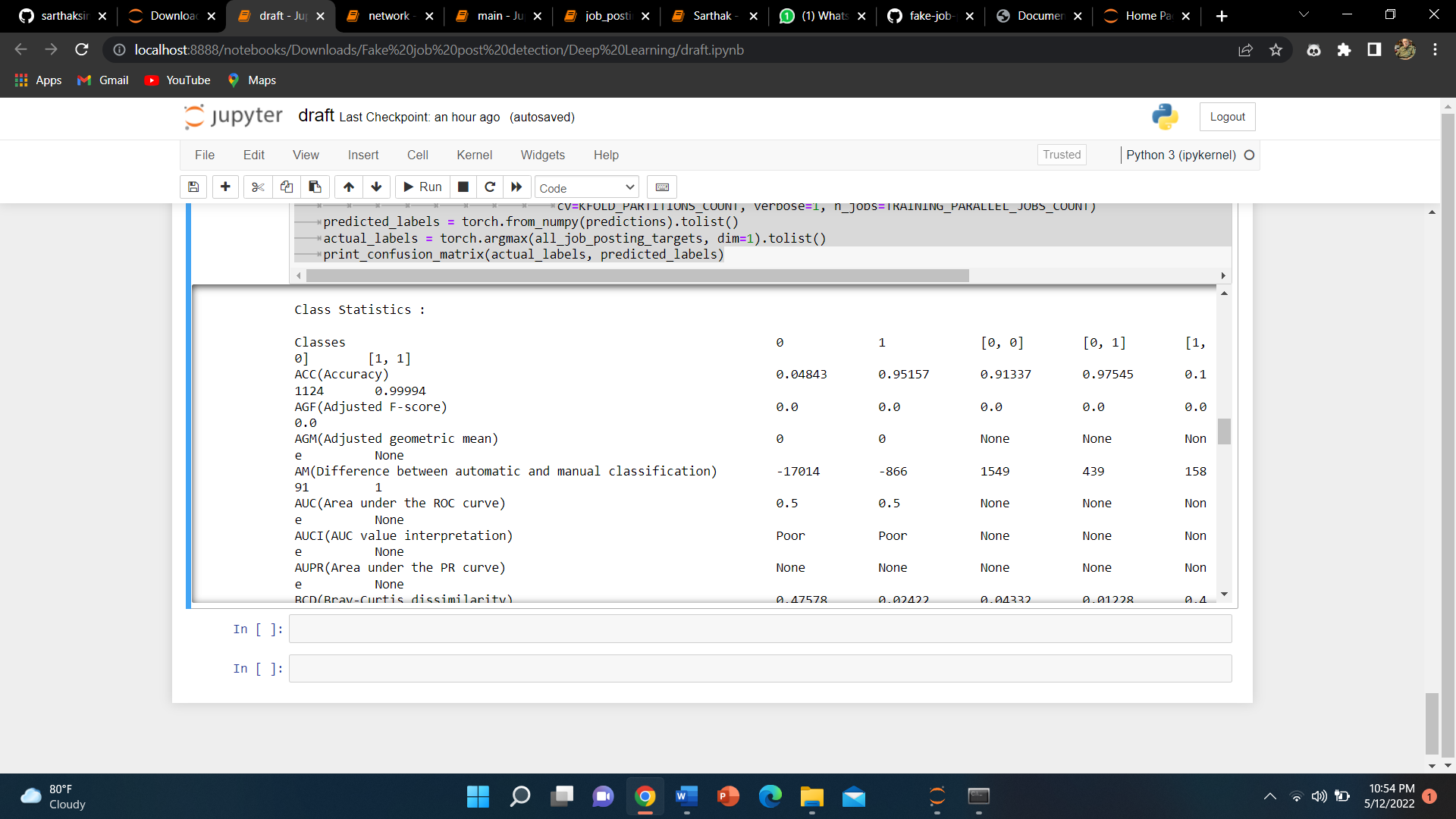
print\_confusion\_matrix(actual\_labels, predicted\_labels)

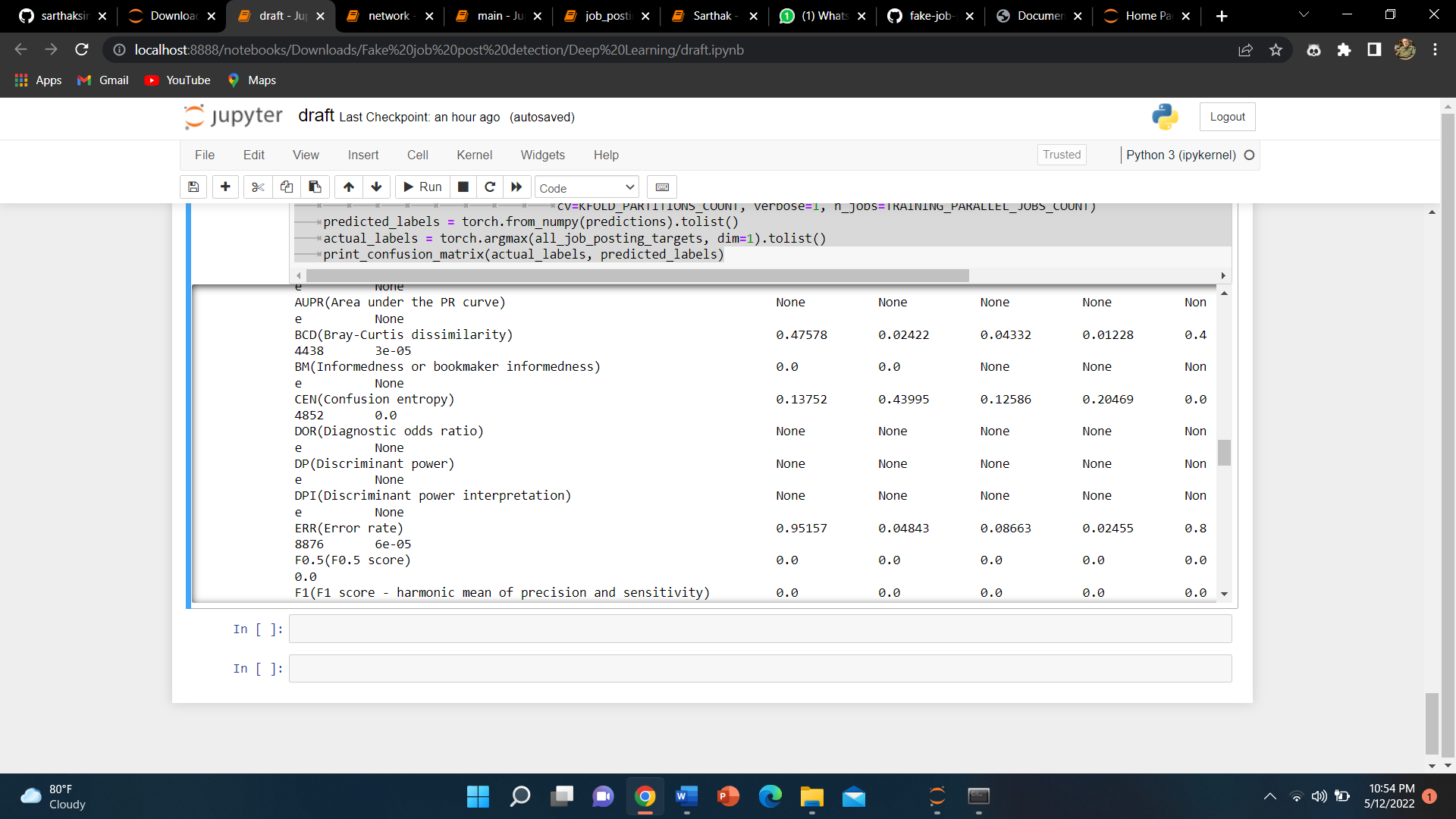
**Link of the GirHub Repository :** [**https://github.com/ankitcs66/Fake-job-post-detection**](https://github.com/ankitcs66/Fake-job-post-detection)

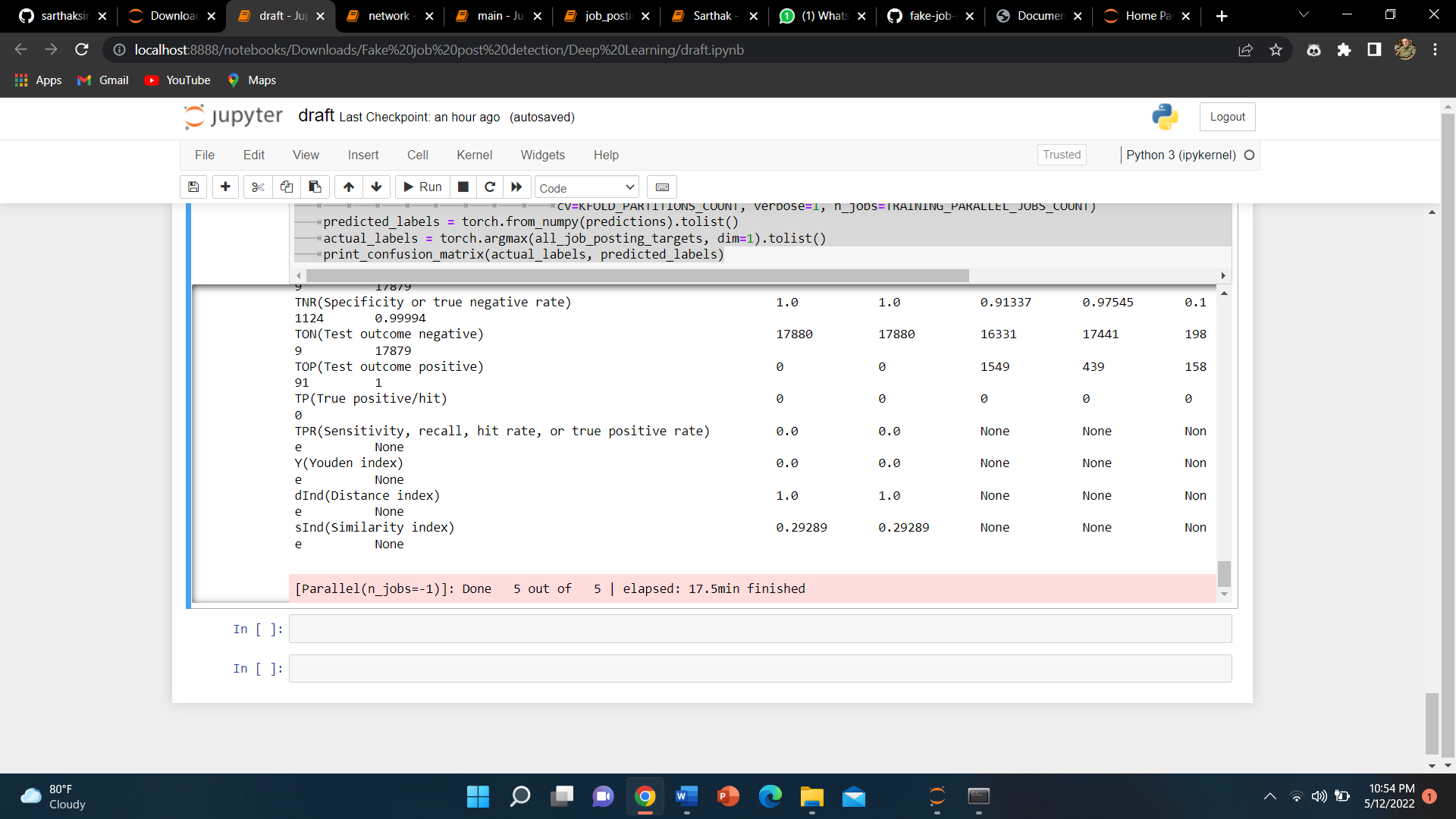
**Results**











**Improvement**

The dataset that is used in this project is very unbalanced. Most jobs are real, and few are fraudulent. Due to this, real jobs are being identified quite well. Certain techniques like SMOTE can be used to generate synthetic minority class samples. A balanced dataset should be able to generate better results.

Conclusion

Detecting employment fraud helps job seekers get only legitimate offers. get To combat the detection of employment fraud, this paper proposes several machine learning and deep learning algorithms as countermeasures. The monitored mechanism is used to demonstrate the use of multiple classifiers to detect employment fraud. Experimental results show that random forest classifiers are superior to peer classification tools. Detecting employment fraud helps job seekers get only the legitimate offers from companies. To combat the detection of employment fraud, several machine learning algorithms are proposed in this paper as countermeasures. The monitored mechanism is used to demonstrate the use of multiple classifiers to detect employment fraud. Experimental results show that random forest classifiers are superior to peer classification tools.

To combat employment fraud, this paper proposes several machine learning algorithms as countermeasures. The monitored mechanism is used to demonstrate the use of multiple classifiers to detect employment fraud. Experimental results show that random forest classifiers are superior to peer classification tools. Detecting employment fraud helps job seekers get only legitimate offers from companies. To combat the detection of employment fraud, this paper proposes several machine learning algorithms as countermeasures. The monitored mechanism is used to demonstrate the use of multiple classifiers to detect employment fraud. Experimental results show that random forest classifiers are superior to peer classification tools. The proposed approach achieved 98.27% accuracy. This is much more accurate than the existing method.

**REFERENCES**

Bishop, C. M. (2006) [Pattern Recognition and Machine Learning](http://research.microsoft.com/en-us/um/people/cmbishop/prml/). Chapter 5: Neural Networks.  
Schmidhuber, J. (2015). [Deep Learning in Neural Networks: An Overview](http://www.sciencedirect.com/science/article/pii/S0893608014002135). Neural Networks 61: 85-117.  
Bengio, Y., LeCun, Y., Hinton, G. (2015). [Deep Learning](http://www.nature.com/nature/journal/v521/n7553/full/nature14539.html). Nature 521: 436-44.  
Goodfellow, I., Bengio, Y. and Courville, A. (2016) [Deep Learning](https://github.com/HFTrader/DeepLearningBook). MIT Press.

**B. Alghamdi and F. Alharby, ―An Intelligent Model for Online Recruitment Fraud Detection,” J. Inf. Secur., vol. 10, no. 03, pp. 155–176, 2019, doi: 10.4236/jis.2019.103009.**

**[2] I. Rish, ―An Empirical Study of the Naïve Bayes Classifier An empirical study of the naive Bayes classifier,‖ no. January 2001, pp. 41–46, 2014.**

**[3] D. E. Walters, ―Bayes’s Theorem and the Analysis of Binomial Random Variables,‖ Biometrical J., vol. 30, no. 7, pp. 817–825, 1988, doi: 10.1002/bimj.4710300710.**

**[4] F. Murtagh, ―Multilayer perceptrons for classification and regression,‖ Neurocomputing, vol. 2, no. 5–6, pp. 183–197, 1991, doi: 10.1016/0925-2312(91)90023-5.**

**[5] P. Cunningham and S. J. Delany, ―K - Nearest Neighbour Classifiers,‖ Mult. Classif. Syst., no. May, pp. 1–17, 2007, doi: 10.1016/S0031-3203(00)00099-6.**

**[6] H. Sharma and S. Kumar, ―A Survey on Decision Tree Algorithms of Classification in Data Mining,‖ Int. J. Sci. Res., vol. 5, no. 4, pp. 2094–2097, 2016, doi: 10.21275/v5i4.nov162954.**

**[7] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, “Machine learning for email spam filtering: review, approaches and open research problems,‖ Heliyon, vol. 5, no. 6, 2019, doi: 10.1016/j.heliyon.2019.e01802.**

**[8] L. Breiman,**

**―ST4\_Method\_Random\_Forest,‖ Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1017/CBO9781107415324.004.**

**[9] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, ―Bagging classifiers for fighting poisoning attacks in adversarial classification tasks,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6713 LNCS, pp. 350–359, 2011, doi: 10.1007/978-3-642-21557-5\_37.**