**REAL/FAKE JOB POSTINGS PREDICTION**

Pie Infocomm Winter Internship (45 days)

Machine learning with Python

By

Parameshwari S

Amrita Vishwa Vidyapeetham

**INTRODUCTION**

Now-a-days, there are a lot of job scams because of unemployment. There are a lot of websites which connect recruiter to a suitable candidate. Sometimes, fake recruiters post a job posting on the job portal with a motive to get money. Many people are falling prey to these scammers using the desperation that is caused by an unprecedented incident. Most scammers do this to get personal information from the person they are scamming. Personal information can contain addresses, bank account details, social security numbers, etc. This problem occurs with many job portals. Later, people shift to a new job portal in search of real job but the fake recruiters join this portal as well. So, it is important to detect real and fake jobs.

**OBJECTIVE**

The main aim of this project is to classify job postings as real or fake to overcome the scam issues. This can be done using Natural Language Processing and Machine Learning model. The model gets trained on labelled dataset and finally it will take any relevant job posting and produce a final result determining whether the job is real or fake.

**BACKGROUND**

This dataset contains 18K job descriptions out of which about 800 are fake. The data consists of both textual information and meta-information about the jobs. The dataset can be used to create classification models which can learn the job descriptions which are fraudulent. Fake job postings are a very small fraction of this dataset. That is as excepted. We do not expect a lot of fake jobs postings. This project follows five stages. The five stages adopted for this project are –

1. Problem Definition

2. Data Collection

3. Data cleaning, exploring and pre-processing

4. Modelling/Training

5. Evaluating/Prediction

Attributes in the dataset –

1. **job\_id**→ Unique Job ID

2. **title** → Title of the job

3.**location** → Geographical location of the job

4. **department** → Corporate Department

5. **salary\_range** → Range of Expected Salary

6. **company\_profile** → Brief Description of the Company

7. **description** → Detailed description of the job

8. **requirements** → Enlisted requirements for the job opening

9. **benefits** → Enlisted offered benefits by the employer

10. **telecommunicating** → True for telecommuting positions

11. **has\_company\_logo** → True if company logo is present

12. **has\_questions** → True if screening questions are present

13. **employment\_type** → Full-type, Part-time, Contract, etc.

14. **required\_experience** → Executive, Entry level, Intern, etc.

15. **required\_education** → Doctorate, Master’s Degree, Bachelor, etc.

16. **industry** → Automotive, IT, Health care, Real estate, etc.

17. **function** → Consulting, Engineering, Research, Sales etc.

18. **fraudulent** → target — Classification attribute (1 for fraudulent, else 0)

**HARDWARE AND SOFTWARE REQUIREMENTS**

|  |  |
| --- | --- |
| Processor | Processor – i5 or above |
| RAM | 8GB |
| Operating System | Windows, MAC or Linux |
| Technology | Machine Learning Python |
| Coding Language | Python |
| IDE | Google Colaboratory |

**CODING**

**Importing necessary libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import re

import nltk

import spacy

nlp = spacy.load('en\_core\_web\_sm')

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('stopwords')

from nltk.tokenize import word\_tokenize

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import LinearSVC

from sklearn.metrics import f1\_score, accuracy\_score

from nltk.corpus import stopwords

stopwords = nlp.Defaults.stop\_words

nltk.download('wordnet')

from nltk.stem.wordnet import WordNetLemmatizer

lemma=WordNetLemmatizer()

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.text import one\_hot

from keras.preprocessing.sequence import pad\_sequences

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras import layers

import matplotlib.pyplot as plt

from tensorflow.keras.utils import to\_categorical

from keras.models import Sequential, Model

from keras.preprocessing import sequence

from tensorflow.keras.optimizers import RMSprop

from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding

import warnings

warnings.filterwarnings('ignore')

**Reading the dataset**

df = pd.read\_csv("/content/drive/MyDrive/Pie\_Infocomm/fake\_job\_postings.csv")

df.head(3)

df.shape

**Exploratory data analysis**

df.isnull().sum()

df.describe()

df = df.fillna("")

df.isnull().sum()

sns.set(style="darkgrid")

sns.countplot(x = "fraudulent" , data = df)

df['text'] = df['title'] + ' ' + df['location'] + ' ' + df['department'] + ' ' + df['company\_profile'] + ' ' + df['description'] + ' ' + df['requirements'] + ' ' + df['benefits'] + ' ' + df['employment\_type'] + ' ' + df['required\_education'] + ' ' + df['industry'] + ' ' + df['function']

df.head(3)

**Text pre-processing**

def remove\_urls(text):

  text = text.lower()

  text = re.sub(r'https?://\S+|www\.\S+', ' ', text)

  return text

def remove\_special\_char(text):

  text = re.sub(r'[^a-zA-Z0-9\s]+', '', text)

  return text

def remove\_num(text):

  text = re.sub(r'\d+', ' ', text)

  return text

def remove\_single\_char(text):

  text = re.sub(r'\s[a-zA-Z]\s', ' ', text)

  return text

def remove\_extra\_space(text):

  text = re.sub(r'\s+', ' ', text)

  return text

def clean(text):

  tokens=word\_tokenize(text)

  tokens\_stop=[word for word in tokens if word not in stopwords]

  lemmatized\_words=[lemma.lemmatize(token) for token in tokens if len(token)>2]

  clean\_text=' '.join(lemmatized\_words)

  return clean\_text

df['text'] = df['text'].apply(remove\_urls)

df['text'] = df['text'].apply(remove\_special\_char)

df['text'] = df['text'].apply(remove\_num)

df['text'] = df['text'].apply(remove\_single\_char)

df['text'] = df['text'].apply(remove\_extra\_space)

df['text'] = df['text'].apply(clean)

df.head(3)

**Vectorization and Train-test-split**

cv = CountVectorizer(max\_features = 3000)

x\_bow = cv.fit\_transform(df['text']).toarray()

df1 = pd.DataFrame(data = x\_bow, columns = cv.get\_feature\_names())

df1.head()

X\_train\_bow, X\_val\_bow, y\_train\_bow, y\_val\_bow = train\_test\_split(df1.values, df['fraudulent'], test\_size=0.20, random\_state=0)

tfidf = TfidfVectorizer(max\_features = 3000)

x\_tfidf = tfidf.fit\_transform(df['text']).toarray()

df2 = pd.DataFrame(data = x\_tfidf, columns = tfidf.get\_feature\_names())

df2.head()

X\_train\_tfidf, X\_val\_tfidf, y\_train\_tfidf, y\_val\_tfidf = train\_test\_split(df2.values, df['fraudulent'], test\_size=0.20, random\_state=0)

**Evaluation**

1. **Logistic Regression**

lr = LogisticRegression()

lr.fit(X\_train\_bow,y\_train\_bow)

lr\_y\_pred\_bow = lr.predict(X\_val\_bow)

acc\_bow\_lr = accuracy\_score(lr\_y\_pred\_bow, y\_val\_bow)

print('Accuracy:', acc\_bow\_lr)

lr = LogisticRegression()

lr.fit(X\_train\_tfidf,y\_train\_tfidf)

lr\_y\_pred\_tfidf = lr.predict(X\_val\_tfidf)

acc\_tfidf\_lr = accuracy\_score(lr\_y\_pred\_tfidf, y\_val\_tfidf)

print('Accuracy:', acc\_tfidf\_lr)

1. **Support Vector Machine**

svc = LinearSVC()

svc.fit(X\_train\_bow,y\_train\_bow)

svc\_y\_pred\_bow = svc.predict(X\_val\_bow)

acc\_bow\_svm = accuracy\_score(svc\_y\_pred\_bow, y\_val\_bow)

print('Accuracy:', acc\_bow\_svm)

svc = LinearSVC()

svc.fit(X\_train\_tfidf,y\_train\_tfidf)

svc\_y\_pred\_tfidf = svc.predict(X\_val\_tfidf)

acc\_tfidf\_svm = accuracy\_score(svc\_y\_pred\_tfidf, y\_val\_tfidf)

print('Accuracy:', accuracy\_score(svc\_y\_pred\_tfidf, y\_val\_tfidf))

1. **Naïve-Bayes**

nb = MultinomialNB()

nb.fit(X\_train\_bow,y\_train\_bow)

nb\_y\_pred\_bow = nb.predict(X\_val\_bow)

acc\_bow\_nb = accuracy\_score(nb\_y\_pred\_bow, y\_val\_bow)

print('Accuracy:', acc\_bow\_nb)

nb = MultinomialNB()

nb.fit(X\_train\_tfidf,y\_train\_tfidf)

nb\_y\_pred\_tfidf = nb.predict(X\_val\_tfidf)

acc\_tfidf\_nb = accuracy\_score(nb\_y\_pred\_tfidf, y\_val\_tfidf)

print('Accuracy:', accuracy\_score(nb\_y\_pred\_tfidf, y\_val\_tfidf))

1. **CNN with Word Embedding**

le = LabelEncoder()

fraud = le.fit\_transform(df['fraudulent'])

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df['text'], fraud, test\_size = 0.20, random\_state = 100)

tokenizer = Tokenizer(num\_words = 3000)

tokenizer.fit\_on\_texts(x\_train)

xcnn\_train = tokenizer.texts\_to\_sequences(x\_train)

xcnn\_test = tokenizer.texts\_to\_sequences(x\_test)

vocab\_size = len(tokenizer.word\_index) + 1

ycnn\_train = to\_categorical(y\_train)

ycnn\_test = to\_categorical(y\_test)

max\_len = 200

xcnn\_train = pad\_sequences(xcnn\_train, padding = 'post', maxlen = max\_len)

xcnn\_test = pad\_sequences(xcnn\_test, padding = 'post', maxlen = max\_len)

print(xcnn\_train[0, :])

embedding\_dim = 200

model = Sequential()

model.add(layers.Embedding(vocab\_size, embedding\_dim, input\_length = max\_len))

model.add(layers.Conv1D(128, 5, activation = 'relu'))

model.add(layers.GlobalMaxPool1D())

model.add(layers.Dense(512, activation = 'relu'))

model.add(layers.Dense(2, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

model.summary()

model.fit(xcnn\_train, ycnn\_train, epochs = 5, validation\_data = (xcnn\_test, ycnn\_test), batch\_size = 100)

loss\_word\_emb, accuracy\_word\_emb = model.evaluate(xcnn\_train, ycnn\_train, verbose = False)

print("Training accuracy - ", accuracy\_word\_emb)

print("Training loss - ", loss\_word\_emb)

1. **LSTM**

x=df.text

y=df.fraudulent

le=LabelEncoder()

y=le.fit\_transform(y)

y=y.reshape(-1,1)

x\_train, x\_test, y\_train, y\_test=train\_test\_split(x,y,test\_size=0.15)

max\_words = 3000

max\_len = 200

tok = Tokenizer(num\_words = max\_words)

tok.fit\_on\_texts(x\_train)

sequences = tok.texts\_to\_sequences(x\_train)

sequences\_matrix = sequence.pad\_sequences(sequences, maxlen = max\_len)

def RNN():

    inputs = Input(name='inputs',shape=[max\_len])

    layer = Embedding(max\_words,50,input\_length=max\_len)(inputs)

    layer = LSTM(64)(layer)

    layer = Dense(256,name='FC1')(layer)

    layer = Activation('relu')(layer)

    layer = Dropout(0.5)(layer)

    layer = Dense(1,name='out\_layer')(layer)

    layer = Activation('sigmoid')(layer)

    model = Model(inputs=inputs,outputs=layer)

    return model

model = RNN()

model.summary()

model.compile(loss = 'binary\_crossentropy', optimizer = RMSprop(), metrics = ['accuracy'])

model.fit(sequences\_matrix, y\_train, batch\_size = 128, epochs = 5, validation\_split = 0.2)

test\_sequences = tok.texts\_to\_sequences(x\_test)

test\_sequences\_matrix = sequence.pad\_sequences(test\_sequences,maxlen=max\_len)

loss, acc\_lstm = model.evaluate(test\_sequences\_matrix, y\_test)

print("Accuracy - ", acc\_lstm \*100)

**Comparison of Accuracies**

data\_dict = {'BoW\_LR':acc\_bow\_lr, 'Tf-idf\_LR':acc\_tfidf\_lr, 'BoW\_SVM':acc\_bow\_svm, 'Tf-idf\_SVM':acc\_tfidf\_svm, 'BoW\_NB':acc\_bow\_nb, 'Tf-idf\_NB':acc\_tfidf\_nb, 'Word-embedding - CNN model':accuracy\_word\_emb, 'LSTM':acc\_lstm}

x = list(data\_dict.keys())

y = list(data\_dict.values())

fig = plt.figure(figsize = (20, 7))

plt.bar(x, y,width = 0.3)

plt.xlabel("Models")

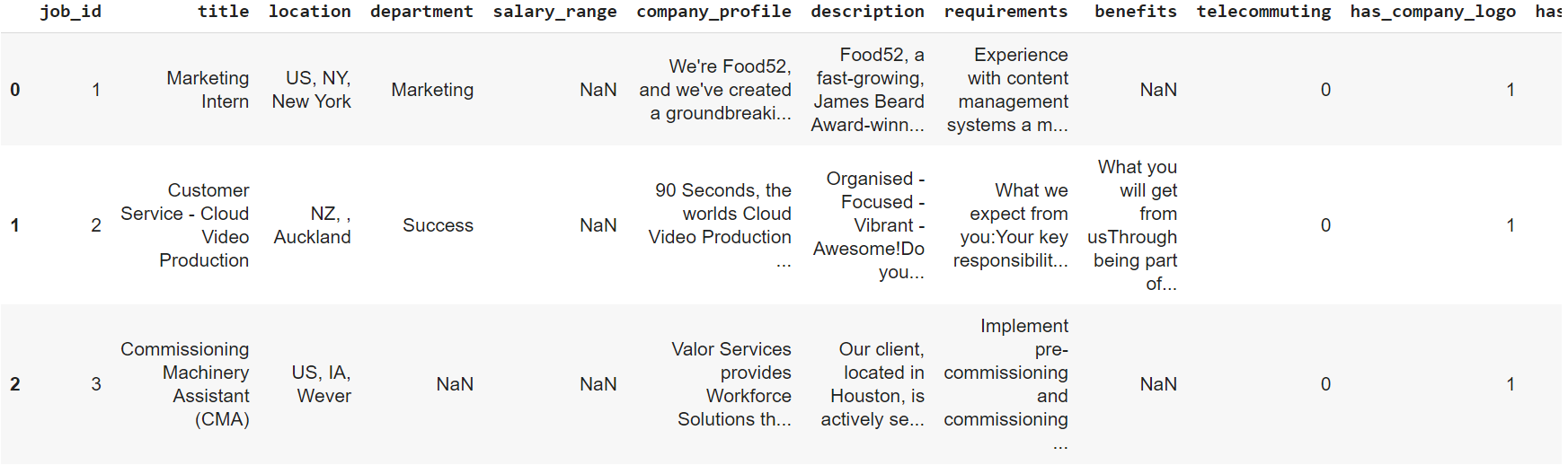
plt.ylabel("Accuracy score")

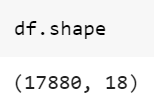
plt.title("Comparison of Accuracy")

plt.show()

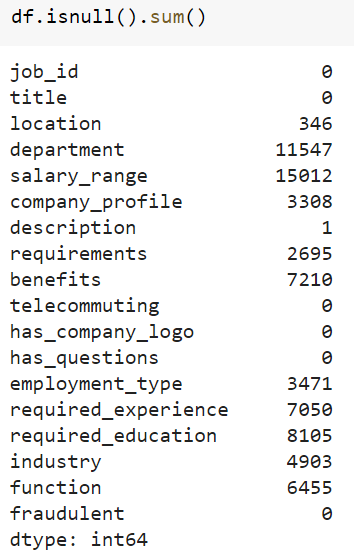
**OUTPUT SCREENSHOT**

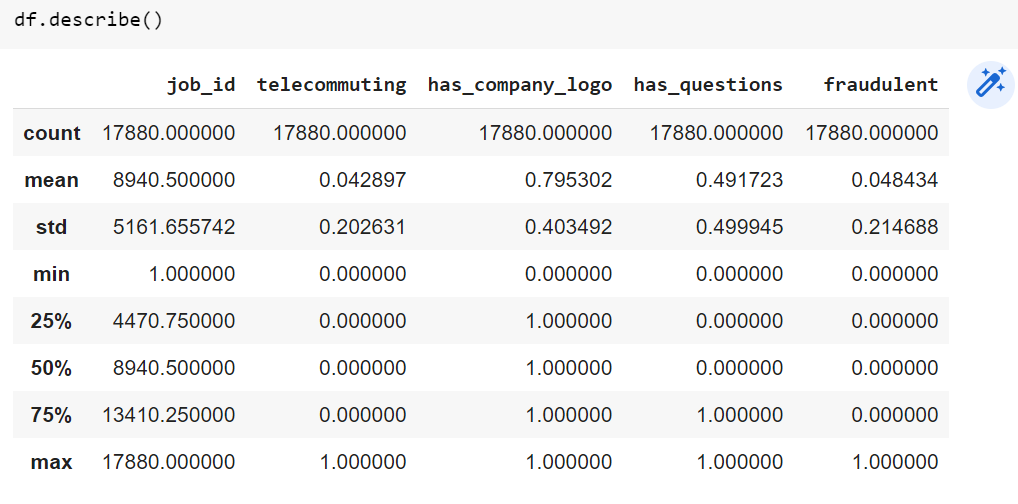
**Reading the dataset**

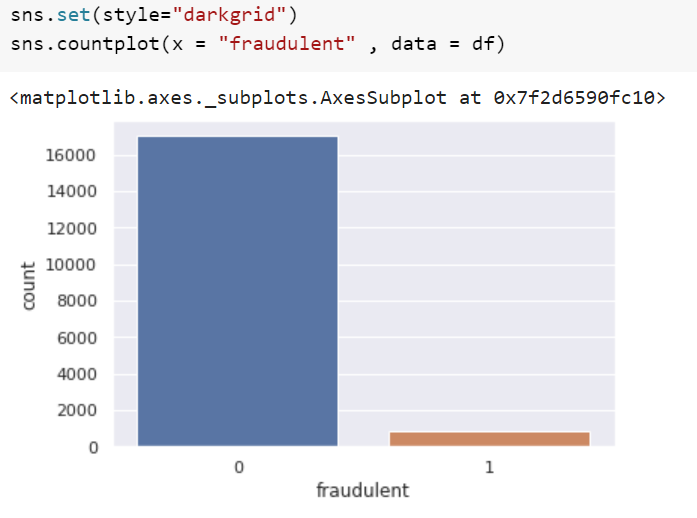
****

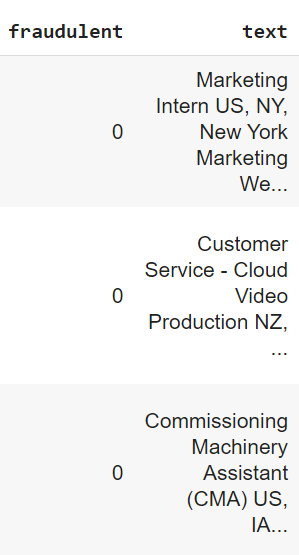
****

**Exploratory data analysis**

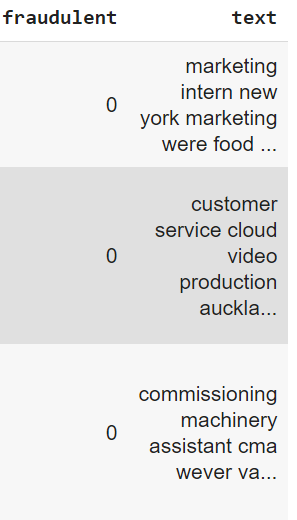
****

****

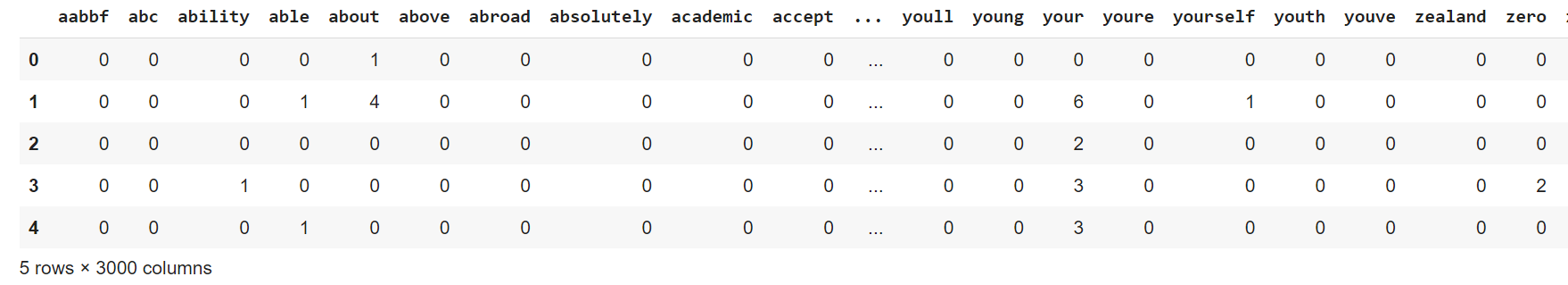
****

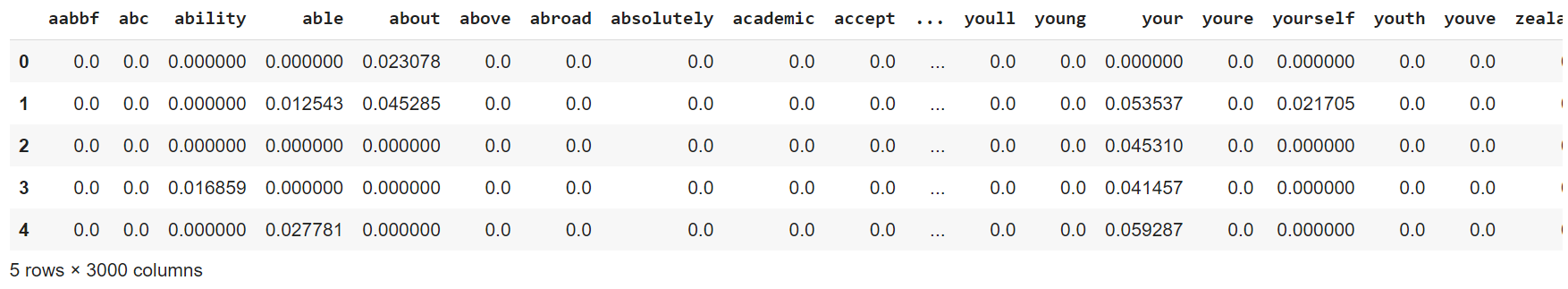
****

**Text pre-processing**

****

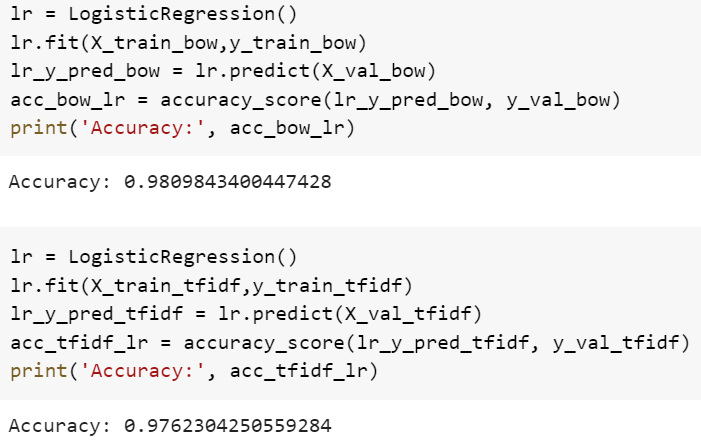
**Vectorization and Train-test-split**

****

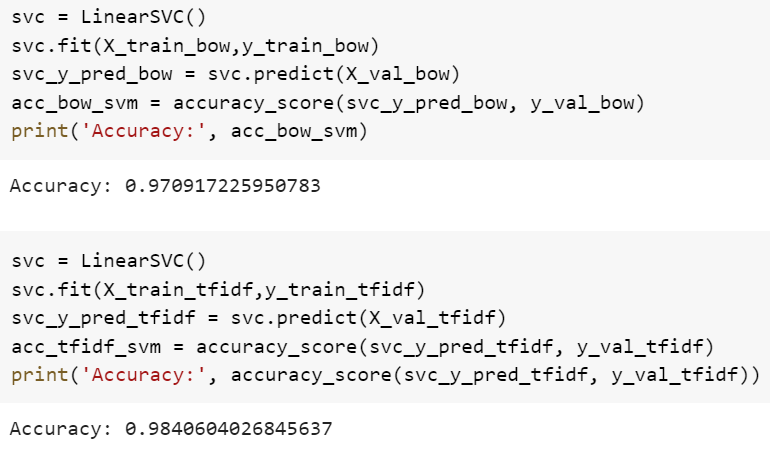
****

**Evaluation**

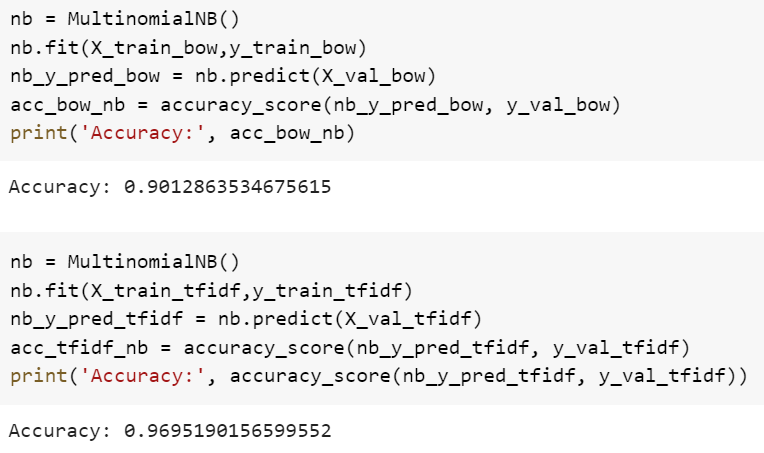
1. **Logistic Regression**



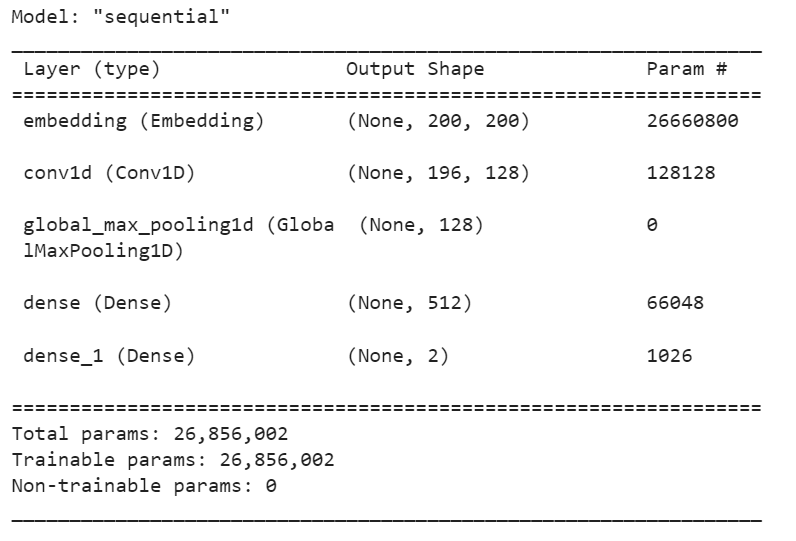
1. **Support Vector Machine**

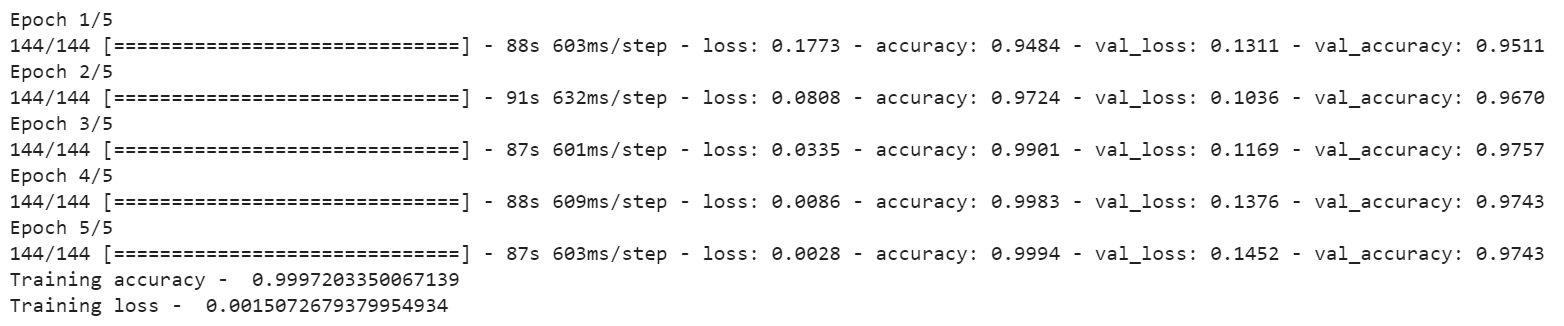
****

1. **Naïve-Bayes**

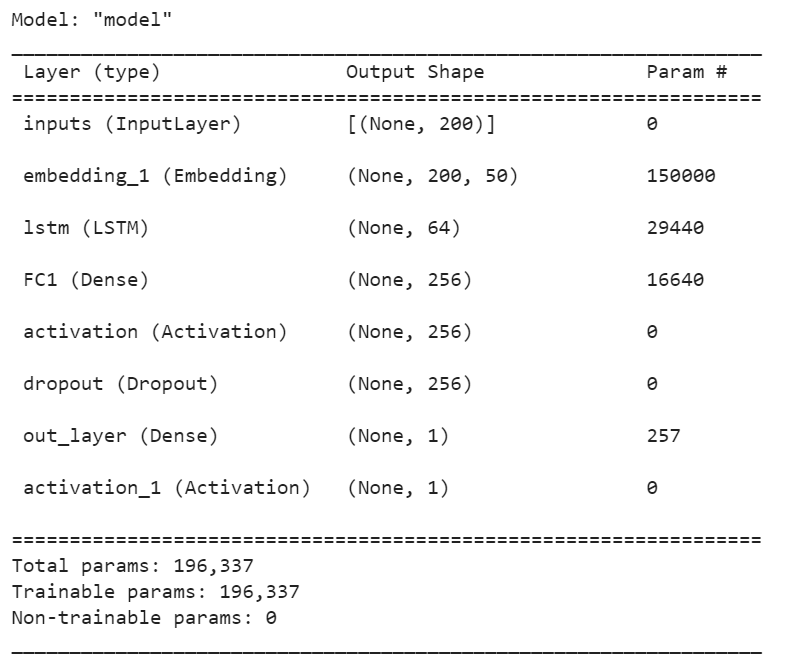
****

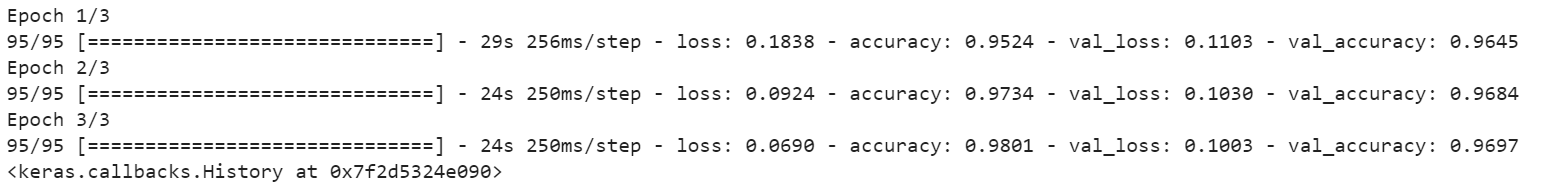
1. **CNN with Word Embedding**

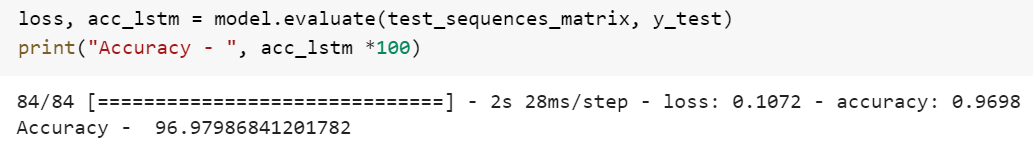
****

****

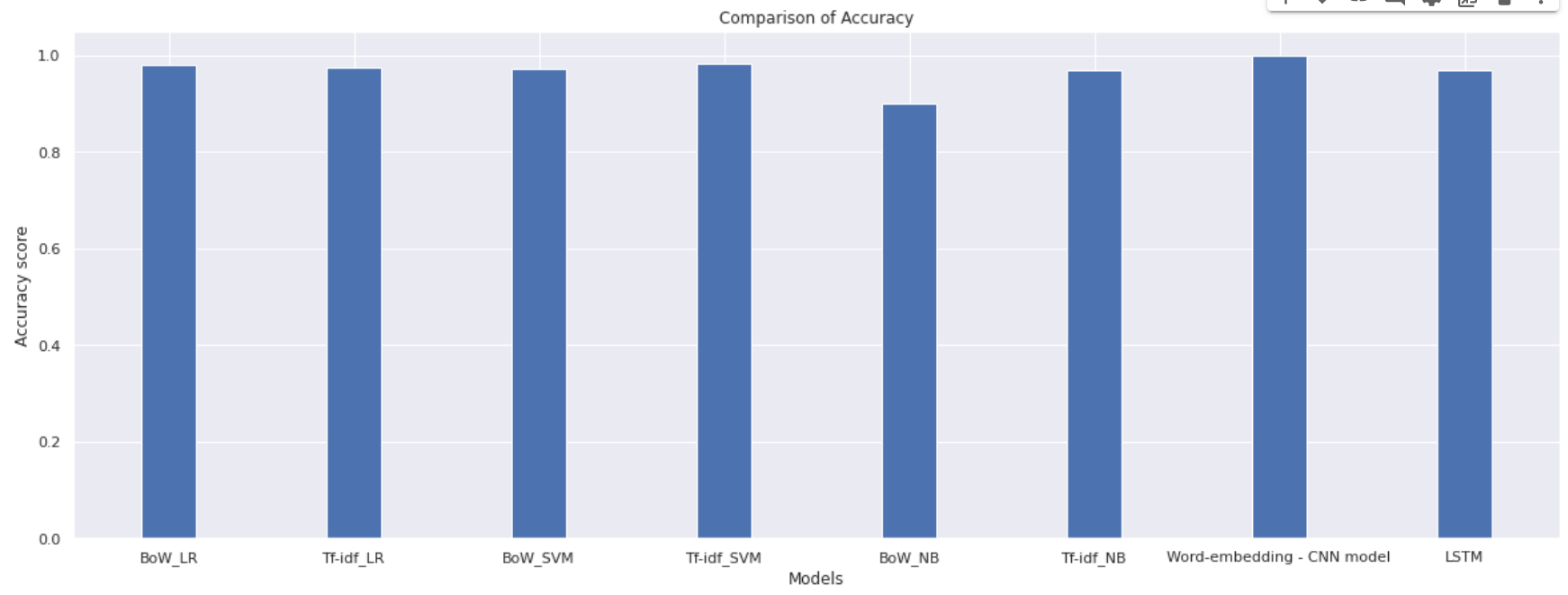
1. **LSTM**

****

****

****

**Comparison of Accuracies**

****

**FUTURE SCOPE**

The dataset contains lot of text descriptions; hence we use the most powerful tool for text analysis that is, Natural Language Processing (NLP) for pre-processing which is a major step after which it can be passed onto any Machine Learning model for training and evaluation. Data has been evaluated with 5 models out of which 3 are Machine Learning models and 2 are Deep Learning model. Now, given a job posting we can predict if it is real or fake.

**CONCLUSION**

In this Real/Fake Job posting Prediction project, I have presented a detailed approach to tackle this problem. Also, the project aims on predicting new postings based on the trained model. Thus, we take the help of Python with Machine/Deep learning as well as NLP for completing this project.

**REFERENCES AND BIBLIOGRAPHY**

* [Real / Fake Job Posting Prediction | Kaggle](https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction)
* Shawni Dutta and Samir Kumar Bandyopadhyay. 2020. Fake job recruitment detection using machine learning approach. International Journal of Engineering Trends and Technology, 68(4):48–53.
* [Anshupriya2694/Fake-Job-Posting-Prediction: This is a classifier that uses NLP to determine if a job posting is real or fake (github.com)](https://github.com/Anshupriya2694/Fake-Job-Posting-Prediction)
* [FelixLuciano/Fake-JobPosting-Prediction: Real and fake job postings prediction (github.com)](https://github.com/FelixLuciano/Fake-JobPosting-Prediction)