Detecting Fraudulent Job Postings

1st Juhi Ramod 2nd Harshit Gadge 3rd Mohit Saluru 4th Anirudh Krishna
DS602,PCS2, *University of Maryland*, College Park, USA

***Abstract*—The proliferation of online job platforms has created significant opportunities for job seekers but has also introduced risks in the form of fraudulent job postings. These scams not only waste time and resources but can also lead to severe financial and personal consequences. Motivated by a personal experience with such a scam, this project aims to develop a robust solution for detecting fake job postings. Leveraging the "Real or Fake Job Posting Prediction" dataset from Kaggle, we explored various machine learning and deep learning models, including Logistic Regression with GloVe and BERT embeddings, XGBoost, Neural Networks, and Graph Neural Networks (GNNs). While BERT embeddings captured contextual richness in textual data, GNNs modeled relational structures among job postings, enhancing fraud detection capabilities. Given the dataset's class imbalance, recall was prioritized as the primary evaluation metric to minimize the risk of overlooking fraudulent listings. The project not only provides an effective tool to identify fake job postings but also contributes valuable insights into patterns of fraudulent behavior, enhancing trust and security in online job markets.**

***Index Terms*—BERT Embeddings, Graph Neural Networks (GNNs),Classifier Evaluation.**

I. Introduction

The rise of online job platforms has revolutionized the job market, providing unparalleled opportunities for job seekers worldwide. However, this growth has also led to a disturbing trend—fake job postings. These fraudulent listings not only waste time and effort but can also lead to financial and personal harm. One of our team members experienced this issue firsthand, becoming a victim of a fake job scam. This personal connection to the problem became a significant motivator for this project, driving us to develop a solution capable of effectively detecting fake job postings and helping others avoid similar pitfalls.

For our project, we leveraged the "Real or Fake Job Posting Prediction" dataset from Kaggle. This dataset contains a rich combination of textual and categorical features, such as job titles, company names, locations, and job descriptions. Its diversity makes it ideal for exploring advanced machine learning techniques to detect fraudulent job listings. Our objective was not only to build models that accurately predict the authenticity of job postings but also to uncover deeper insights into the patterns that characterize fraudulent listings.

To address this problem, we implemented a range of machine learning and deep learning models, including Logistic Regression using GloVe and BERT embeddings for feature representation, XGBoost as a robust ensemble learning technique, Neural Networks leveraging GloVe embeddings to capture semantic information, and Graph Neural Networks (GNNs) to model relationships between job postings by representing them as a graph structure based on feature similarities.

Given the class imbalance in the dataset, where fraudulent postings constituted a minority, we prioritized recall as the key evaluation metric. Detecting fake job postings with high recall is essential to minimize the risk to job seekers, as overlooking fraudulent listings can lead to serious consequences.

A. Motivation

We embarked on this project with a strong desire to learn and experiment with new machine learning techniques while addressing a real-world problem. The challenge of detecting fake job postings from the Kaggle dataset presented an opportunity to explore Graph Neural Networks (GNNs) and BERT embeddings, two advanced techniques we were eager to understand and apply. GNNs allowed us to model the relationships between job postings, capturing patterns based on features like job description and location, which we had not explored before. Similarly, BERT embeddings helped us apply state-of-the-art text analysis techniques to improve our model's understanding of job descriptions.

A major driving force behind this project was the personal experience of one of our team members, who fell victim to a fake job scam. This motivated us even further to develop a solution that could help identify fraudulent job postings and prevent others from experiencing the same. Overall, this project combined our interest in learning new techniques with a passion for solving a problem that affects many people.

B. Benefits of Model

This section discusses the advantages of analyzing the influence of priming questions, we can explore how subtle cues affect individuals' decision-making and moral judgments. This can have broad implications for fields such as psychology, marketing, and behavioral economics, where priming is commonly used to shape responses.

1) **Improved Fraud Detection Accuracy**: By utilizing GNNs, the model can analyze not just individual job postings but also their relationships to others in the dataset. This relational understanding allows the model to identify subtle patterns or anomalies that might indicate fraud.
2) **Effective Use of Relational Data**: Fraudulent job postings often share similarities (e.g., vague descriptions, suspicious locations). GNNs excel in leveraging these relationships, making them uniquely suited for this task compared to traditional machine learning models.
3) **Versatility:**: The methodology supports adaptation to other fraud detection scenarios, demonstrating flexibility in addressing similar challenges across industries.

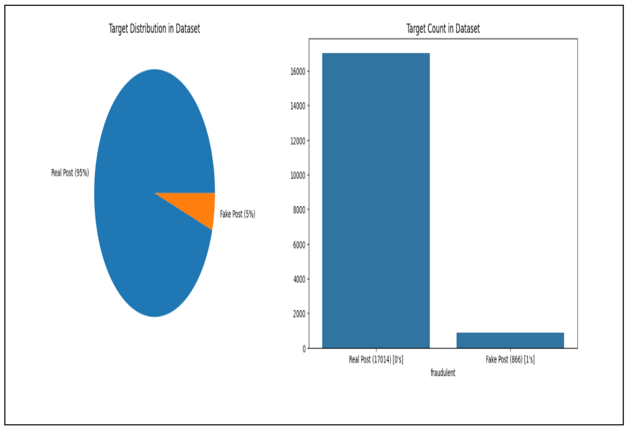


Fig. 1. Distribution

4) **Effective Use of Relational Data**: Fraudulent job postings often share similarities (e.g., vague descriptions, suspicious locations). GNNs excel in leveraging these relationships, making them uniquely suited for this task compared to traditional machine learning models.

II. Related Work

A. Fraud Detection in Job Postings:

Previous studies have utilized traditional machine learning models, such as Logistic Regression, Naive Bayes, and Random Forest, to classify fraudulent job postings. These approaches primarily focus on feature engineering from textual data, leveraging attributes like job descriptions and company profiles. However, they often fail to capture the relational context between postings, limiting their effectiveness.

B. Graph-Based Approaches:

Graph-based methods for fraud detection have gained traction in domains such as social networks and e-commerce. Models like Graph Neural Networks (GNNs) have proven effective in learning from both node features and graph structures, enabling the detection of anomalies through relational patterns. This approach inspired the use of GNNs in job fraud detection, where job postings and their interconnections provide valuable relational insights.

C. Natural Language Processing in Fraud Detection:

NLP techniques have been widely used for text analysis in fraud detection. For instance, Term Frequency-Inverse Document Frequency (TF-IDF) and word embeddings (e.g., Word2Vec, GloVe) have been employed to extract features from textual data like job descriptions and requirements. Advanced NLP models, such as BERT, have further enhanced feature representation, aiding in fraud classification tasks.

III. Method

A. Dataset

The dataset comprises 17,880 rows and 18 columns, with each row representing a unique job posting. It includes a range of features that provide detailed information about the postings, such as job titles, company profiles, locations, and job descriptions. The target column, labeled ”fraudulent,” serves as the classification indicator, specifying whether a job posting is legitimate or fraudulent. This comprehensive dataset offers a mix of categorical and textual attributes, making it well-suited for both exploratory and predictive analyses in fraud detection.

B. Data Cleaning: Issues and Handling techniques

In the process of cleaning the dataset for this project, I encountered several data quality issues that needed to be addressed to ensure the integrity and reliability of the analysis.

1) **Handling Missing Data** : Columns with a high percentage of missing values, such as ”salary range,” ”job title,” and ”job ID,” will be dropped due to their limited relevance to the prediction task. For other features with moderate missing data, imputation techniques will be applied to ensure the dataset remains comprehensive.
2) **Processing Text Data**: The ”description” column, which contains valuable details about job responsibilities and requirements, will be cleaned to remove HTML tags, whitespaces, and stop words. Natural Language Processing (NLP) techniques will be utilized to extract meaningful features, contributing significantly to model performance.

C. Exploratory Data Analysis

During the Exploratory Data Analysis (EDA) phase, I focused on examining the dataset to uncover key patterns, relationships, and anomalies.

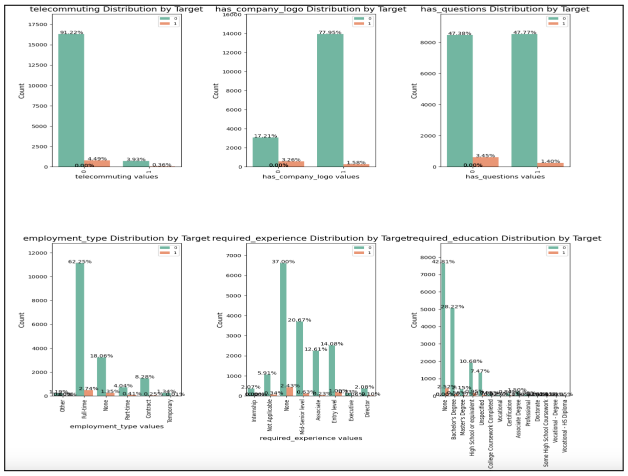


Fig. 2. EDA

1) **Data Overview** : Analyzed the dataset’s dimensions, feature types, and distributions to get a clear understanding of its content.
2) **Missing Data**: Identified and addressed missing values, especially in critical text fields like job descriptions and requirements.

3) **Imbalance Analysis**: Assessed the class distribution of the target variable (fraudulent) and observed a significant imbalance, which guided later resampling techniques.
4) **Feature Insights**: Performed univariate and bivariate analyses to identify trends in features such as telecommuting, has_company_logo, and has_questions, revealing their relationship with fraudulent postings.
5) **Text Analysis**: Explored text-based features using word clouds and frequency distributions to identify common terms and linguistic patterns associated with fraud.

D. Data Preprocessing

During preprocessing, I resolved inconsistencies and improved data quality to ensure the dataset was ready for modeling.

1) **Special Character Removal** : All non-alphanumeric characters, including punctuation marks (e.g., @, #, &) and numbers (e.g., phone numbers), were removed. This was important because such characters do not provide semantic meaning and could interfere with feature extraction.
2) **Stop Word Removal** : Common stop words like ”the,” ”is,” and ”and,” which do not contribute meaningfully to the context of the job posting, were removed. This was done using libraries like NLTK or spaCy, which provide a predefined list of stop words.
3) **Whitespace Normalization** : Excess whitespace, tabs, or line breaks were removed to avoid discrepancies in the tokenization process.
4) **Tokenization and Lemmatization** : The text was tokenized into individual words, and each word was lemmatized (e.g., converting ”running” to ”run”) to standardize the tokens. This helped reduce variability and ensured that related terms were treated as the same (e.g., ”jobs” and ”job” were reduced to a common root).

E. Feature Engineering

In the feature engineering phase, I focused on transforming the raw data into meaningful features for the machine learning models:

1) GloVe (Global Vectors for Word Representation): To capture more nuanced semantic meaning in job postings, GloVe word embeddings were applied. These embeddings represent words as dense vectors, where words with similar meanings are placed closer in the vector space. GloVe embeddings were pre-trained on a large corpus of text, providing a richer representation of job-related terms.

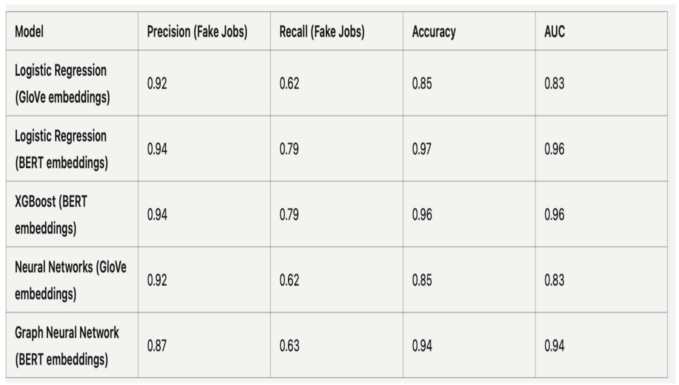
1) **Implementation** : Pre-trained GloVe embeddings (e.g., glove.6B.300d) were loaded, and each word in the job postings was mapped to its corresponding vector representation. If a word was not found in the GloVe vocabulary, it was skipped or initialized with a random vector.
2) **Result**: GloVe provided better semantic understanding, and the classifier performed better by identifying contextually similar words that appeared in fake job postings, such as ”guaranteed salary” or ”no experience required.” However, the model still struggled with complex patterns, such as subtle linguistic cues.

2) BERT (Bidirectional Encoder Representations from Transformers): BERT was employed to fine-tune a state-of-the-art deep learning model on the job posting dataset. Unlike Count Vectorization and GloVe, BERT generates contextualized word embeddings, meaning that the representation of a word depends on its surrounding words. This was particularly useful for identifying fraud in postings where the language is subtle or context-dependent.

1) **Implementation** : The job descriptions and requirements were tokenized using BERT’s tokenizer. These tokenized inputs were then fed into a pre-trained BERT model, which was fine-tuned on the dataset to adjust the model’s weights based on the specific task of job fraud detection.
2) **Result**: BERT significantly outperformed the other methods, achieving higher accuracy and better generalization. The fine-tuned model was able to capture complex patterns, such as fraud in the way certain jobs were described (e.g., exaggerated salary promises, lack of company details, or unrealistic job qualifications).

F. Performance Evaluation Metrics

The comparison table highlights the performance of models.



Model	Precision (Fake Jobs)	Recall (Fake Jobs)	Accuracy	AUC
Logistic Regression (GloVe embeddings)	0.92	0.62	0.85	0.83
Logistic Regression (BERT embeddings)	0.94	0.79	0.97	0.96
XGBoost (BERT embeddings)	0.94	0.79	0.96	0.96
Neural Networks (GloVe embeddings)	0.92	0.62	0.85	0.83
Graph Neural Network (BERT embeddings)	0.87	0.63	0.94	0.94

Fig. 3. Metrics

IV. Proposed System

Fake Job Description will guide job-seekers to get only legitimate offers from companies. For tackling employment scam detection, several machine learning algorithms are proposed as countermeasures in this project. Supervised mechanism is used to exemplify the use of several classifiers for employment scam detection. In this project we propose logistic regression and ensemble classifier which outperforms the existing system.

V. Result and Discussion

The process began with data cleaning and extensive text preprocessing, including tokenization, stopword removal, and advanced vectorization techniques such as CountVectorizer, GloVe embeddings, and BERT embeddings. Various models were tested, including Logistic Regression, XGBoost, Neural Networks, and Graph Neural Networks (GNNs), all with the goal of maximizing recall, as detecting fraudulent job postings (the minority class) is the primary objective.

The performance of the models was evaluated, with particular focus on recall, the most important metric for this project. Logistic Regression and XGBoost with BERT embeddings achieved outstanding results in recall, both surpassing 0.79 for fake job detection. Specifically, Logistic Regression reached an AUC of 0.96 and a recall of 79 percent, while XGBoost delivered a similar recall at 79 percent, with an AUC of 0.96. These models demonstrated superior ability to identify fraudulent postings, significantly outperforming models using traditional vectorization methods like CountVectorizer (recall: 65percent) and GloVe embeddings with Neural Networks (recall: 62 percent).

Additionally, Graph Neural Networks (GNNs), using BERT features, showed promise with an AUC of 0.94, but their recall was lower at 63 percent. Although the GNN performed well in terms of AUC, its lower recall highlights the challenge of detecting minority class instances in graph-based approaches, which may be less effective in balancing the focus on fraudulent job postings.

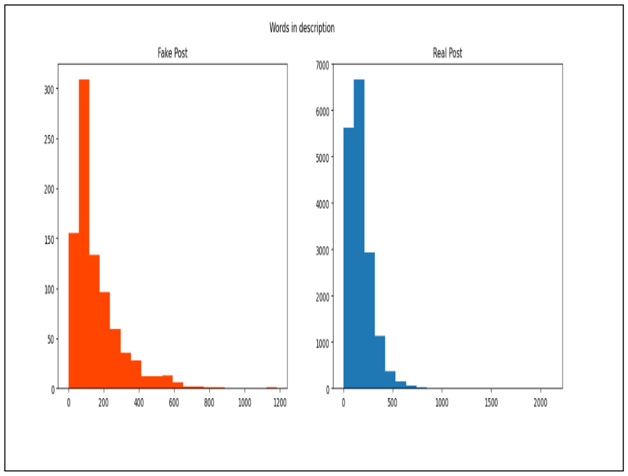


Fig. 4. Word Count

Visualizations such as the word and character counts for fake and real job postings reveal key differences. For example, fake job postings tend to have shorter descriptions, fewer words in the company profile, and fewer characters in the requirements and benefits sections. These findings indicate that fraudulent job postings may be less detailed compared to legitimate ones, which can be a potential indicator for model predictions.

A. Logistic Regression with GloVe Embeddings

Logistic Regression is a linear model used for binary classification problems. It calculates the probability that a given input belongs to a particular class using the logistic (sigmoid) function. It's simple, interpretable, and works well for text classification when combined with word embeddings like GloVe.

- GloVe embeddings (Global Vectors for Word Representation) capture semantic relationships between words by mapping them into a vector space. The embeddings help convert textual data into numerical form, preserving word similarities.
- After preprocessing (cleaning and tokenization), GloVe embeddings were used to transform the text into fixed-dimensional vectors, which were then passed as input to the Logistic Regression model.

Logistic Regression with GloVe embeddings is computationally efficient and easy to implement. It serves as a baseline model to compare against more complex techniques while leveraging the semantic relationships captured by GloVe.

B. Logistic Regression with BERT Embeddings

This model combines Logistic Regression with BERT embeddings. BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained transformer-based model that captures the contextual meaning of words in sentences. Unlike GloVe, BERT embeddings consider the context of a word within a sentence, making them more powerful for text analysis.

- BERT embeddings convert the textual features (e.g., job description, title) into meaningful numerical vectors that represent the text's semantics.
- Logistic Regression is then applied to classify these embeddings into real or fake job postings.

C. XGBoost with BERT Embeddings

XGBoost (Extreme Gradient Boosting) is an ensemble learning technique that builds multiple decision trees sequentially, where each tree learns from the errors of the previous one. It is highly effective for structured data and imbalanced datasets.

- Here, BERT embeddings are used to transform textual features into rich numerical representations.
- XGBoost then processes these embeddings to classify the job postings.

XGBoost is known for its superior performance in classification tasks, especially with imbalanced data like this project, where fake jobs are a minority. Its combination with BERT embeddings allows it to learn intricate relationships in the text data, leading to high recall and AUC scores.

D. Neural Networks with GloVe Embeddings

A Neural Network is a multi-layered model that learns hierarchical patterns in data. In this project:

- The input text (e.g., job description) is represented using GloVe embeddings.
- A Sequential Neural Network architecture was used, consisting of multiple layers:
  - **Dense Layers:** Fully connected layers that transform the embeddings.

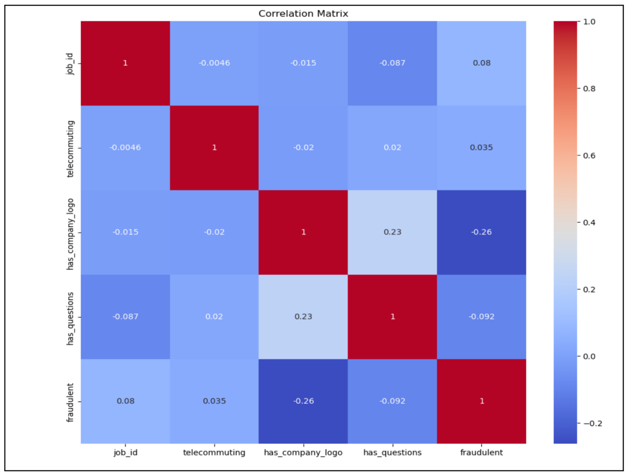


Fig. 5. Heatmap

- **Dropout Layers:** Prevent overfitting by randomly dropping connections during training.
- **Batch Normalization:** Normalizes layer inputs to stabilize and accelerate training.
- **Activation Functions:**
  - **ReLU:** Used for hidden layers to capture non-linear relationships.
  - **Sigmoid:** Used for the output layer to classify the job postings.

Neural Networks can model complex, non-linear patterns in the data. Using GloVe embeddings as input provides pre-trained word relationships, enhancing the model's understanding of job descriptions.

E. Graph Neural Network (GNN) with BERT Embeddings

Graph Neural Networks (GNNs) extend neural networks to graph-structured data. Unlike other models that process job postings independently, GNNs treat postings as nodes in a graph and model the relationships between them as edges.

- Nodes represent job postings, and edges are created based on feature similarities like job title, location, and description.
- BERT embeddings were used to represent each node (job posting) numerically.
- A GNN model (e.g., using GCNConv layers) learns both the features of individual nodes (job postings) and the structural relationships within the graph.

GNNs are particularly powerful when the data has relational dependencies. In this project, job postings are not isolated—they can share similarities based on descriptions, locations, or titles. GNNs enable the model to learn from both the job features and their interconnections, which traditional models cannot capture.

VI. Conclusion and Future Scope

In conclusion, the Logistic Regression and XGBoost models, utilizing BERT embeddings, emerged as the best-performing algorithms, with the highest recall, ensuring better detection of fake job postings. These models not only delivered high accuracy and AUC but also excelled at minimizing false negatives, making them highly effective for real-world fake job detection tasks. The GNN model, while strong in capturing complex relationships, demonstrated that traditional classifiers like Logistic Regression and XGBoost are more reliable when recall is prioritized. This project underscores the critical importance of recall in imbalanced classification tasks and highlights the effectiveness of combining advanced text representations with optimized machine learning models for better detection of fraudulent activities.

References

[1] D. Varmedja, M. Karanovic, S. Sladojevic, M. Arsenovic, and A. Anderla. 'Machine Learning Methods for Credit Card Fraud Detection,' Proceedings of the 18th International Symposium, 2019.

[2] Cardoso Durier da Silva, Ana Cristina Garcia, Fernando, and Rafael Vieira. 'Can Robots Detect Fake News? A Social Media Survey,' 2019.

[3] Stephanie Ludi, Aashir Amaar, Wajdi Aljedaani, Furqan Rustam, Saleem Ullah, and Vaibhav Rupapara. 'Using Machine Learning and Natural Language Processing to Detect Fake Job Postings,' Letters on Neural Processing M. Schoenberger, "Exploratory data analysis." 27 No. 5 (1979): 563-564 in IEEE Transactions on Acoustics, Speech, and Signal Processing.

[4] 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.

[5] Ibrahim M. Nasser, Amjad H. Alzaanin and Ashraf Yunis Maghari."Online Recruitment Fraud Detection using ANN"In 2019 Palestinian International Conference on Information and Communication Technology (PICICT).

[6] B. Alghamdi and F. Alharby. "An Intelligent Model for Online Recruitment Fraud Detection" in 2022 Journal of Information Security

Juhi Ramod - 121182883
Harshit Gadge- 121334770
Mohit Saluru- 121330970
Anirudh Krishna- 121093165

https://drive.google.com/file/d/1KFTBiVzmqlZNowL6YkMBNQIv
WyEa7ue-/view?usp=sharing

December 10, 2024

[47]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import time
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split, GridSearchCV,␣
  ↪StratifiedKFold,KFold, cross_val_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
import xgboost as xgb
from sklearn import preprocessing, model_selection, pipeline
from sklearn.metrics import f1_score, roc_auc_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import GlobalMaxPooling1D, Conv1D, MaxPooling1D,␣
  ↪Flatten, Bidirectional, SpatialDropout1D
from tensorflow.keras.preprocessing import sequence, text
from tensorflow.keras.callbacks import EarlyStopping
```

[49]:
```
df = pd.read_csv('fake_job_postings.csv')
df.head()
```

[49]:
```
   job_id                                      title            location  \
0       1                           Marketing Intern    US, NY, New York
1       2  Customer Service - Cloud Video Production      NZ, , Auckland
```

```
2       3    Commissioning Machinery Assistant (CMA)       US, IA, Wever
3       4          Account Executive - Washington DC  US, DC, Washington
4       5                        Bill Review Manager  US, FL, Fort Worth

  department salary_range                                    company_profile  \
0  Marketing          NaN  We're Food52, and we've created a groundbreaki…
1    Success          NaN  90 Seconds, the worlds Cloud Video Production …
2        NaN          NaN  Valor Services provides Workforce Solutions th…
3      Sales          NaN  Our passion for improving quality of life thro…
4        NaN          NaN  SpotSource Solutions LLC is a Global Human Cap…

                                         description  \
0  Food52, a fast-growing, James Beard Award-winn…
1  Organised - Focused - Vibrant - Awesome!Do you…
2  Our client, located in Houston, is actively se…
3  THE COMPANY: ESRI - Environmental Systems Rese…
4  JOB TITLE: Itemization Review ManagerLOCATION:…

                                        requirements  \
0  Experience with content management systems a m…
1  What we expect from you:Your key responsibilit…
2  Implement pre-commissioning and commissioning …
3  EDUCATION: Bachelor’s or Master’s in GIS, busi…
4  QUALIFICATIONS:RN license in the State of Texa…

                                            benefits  telecommuting  \
0                                                NaN              0
1  What you will get from usThrough being part of…              0
2                                                NaN              0
3  Our culture is anything but corporate-we have …              0
4                              Full Benefits Offered              0

   has_company_logo  has_questions employment_type required_experience  \
0                 1              0           Other          Internship
1                 1              0       Full-time      Not Applicable
2                 1              0             NaN                 NaN
3                 1              0       Full-time    Mid-Senior level
4                 1              1       Full-time    Mid-Senior level

  required_education                   industry              function  \
0                NaN                        NaN             Marketing
1                NaN  Marketing and Advertising      Customer Service
2                NaN                        NaN                   NaN
3  Bachelor's Degree          Computer Software                 Sales
4  Bachelor's Degree     Hospital & Health Care  Health Care Provider

   fraudulent
```

```
0           0
1           0
2           0
3           0
4           0
```

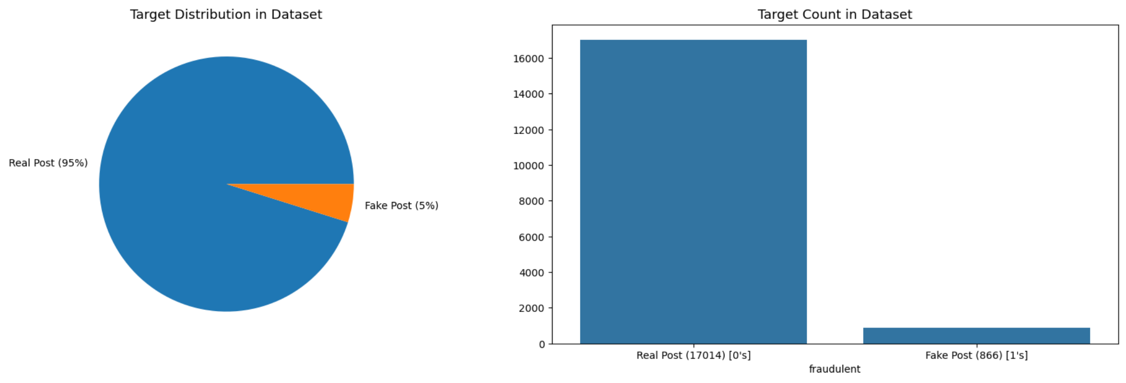
[51]:
```
df.isnull().sum()
```

[51]:
```
job_id                     0
title                      0
location                 346
department             11547
salary_range           15012
company_profile         3308
description                1
requirements            2696
benefits                7212
telecommuting              0
has_company_logo           0
has_questions              0
employment_type         3471
required_experience     7050
required_education      8105
industry                4903
function                6455
fraudulent                 0
dtype: int64
```

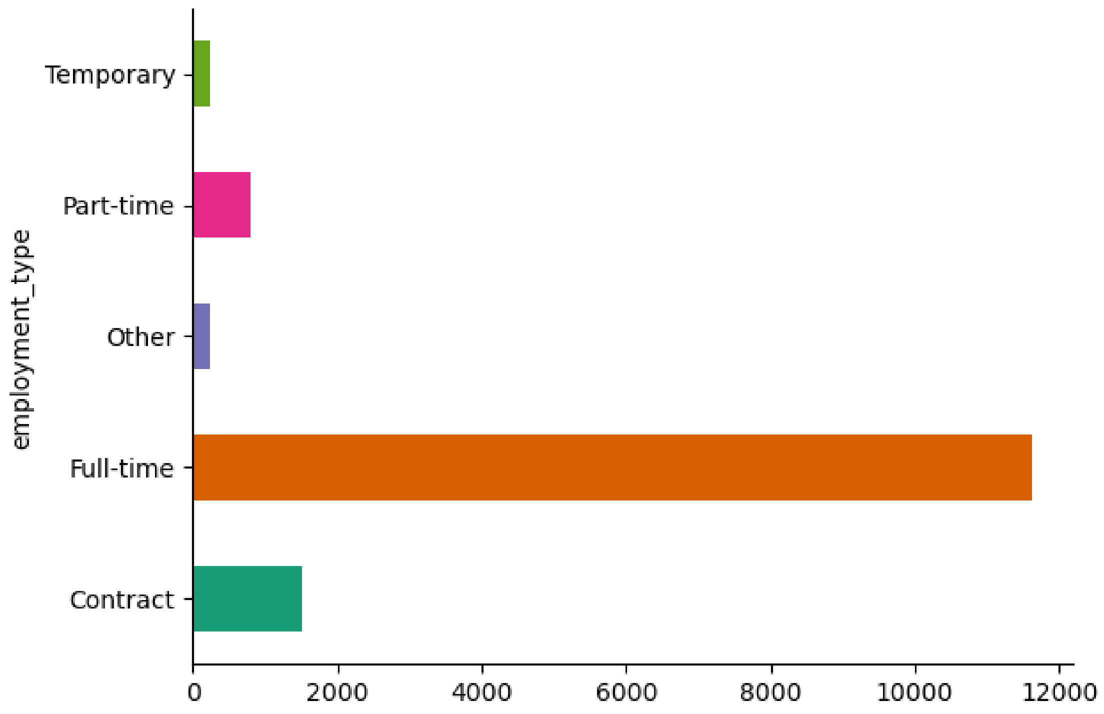
[53]:
```
text_df = df[["title", "company_profile", "description", "requirements",␣
 ↪"benefits","fraudulent"]]
text_df = text_df.fillna(' ')
cat_df = df[["telecommuting", "has_company_logo", "has_questions",␣
 ↪"employment_type", "required_experience", "required_education", "industry",␣
 ↪"function","fraudulent"]]
cat_df = cat_df.fillna("None")
```

[59]:
```
fig, axes = plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
plt.tight_layout()

# Pie chart
df["fraudulent"].value_counts().plot(kind='pie', ax=axes[0], labels=['Real Post␣
 ↪(95%)', 'Fake Post (5%)'])

# Bar chart
temp = df["fraudulent"].value_counts()
sns.barplot(x=temp.index, y=temp.values, ax=axes[1])
```

```
# Setting labels
axes[0].set_ylabel(' ')
axes[1].set_ylabel(' ')
axes[1].set_xticks([0, 1])  # Explicitly set ticks
axes[1].set_xticklabels(["Real Post (17014) [0's]", "Fake Post (866) [1's]"])

# Titles
axes[0].set_title('Target Distribution in Dataset', fontsize=13)
axes[1].set_title('Target Count in Dataset', fontsize=13)

plt.show()
```

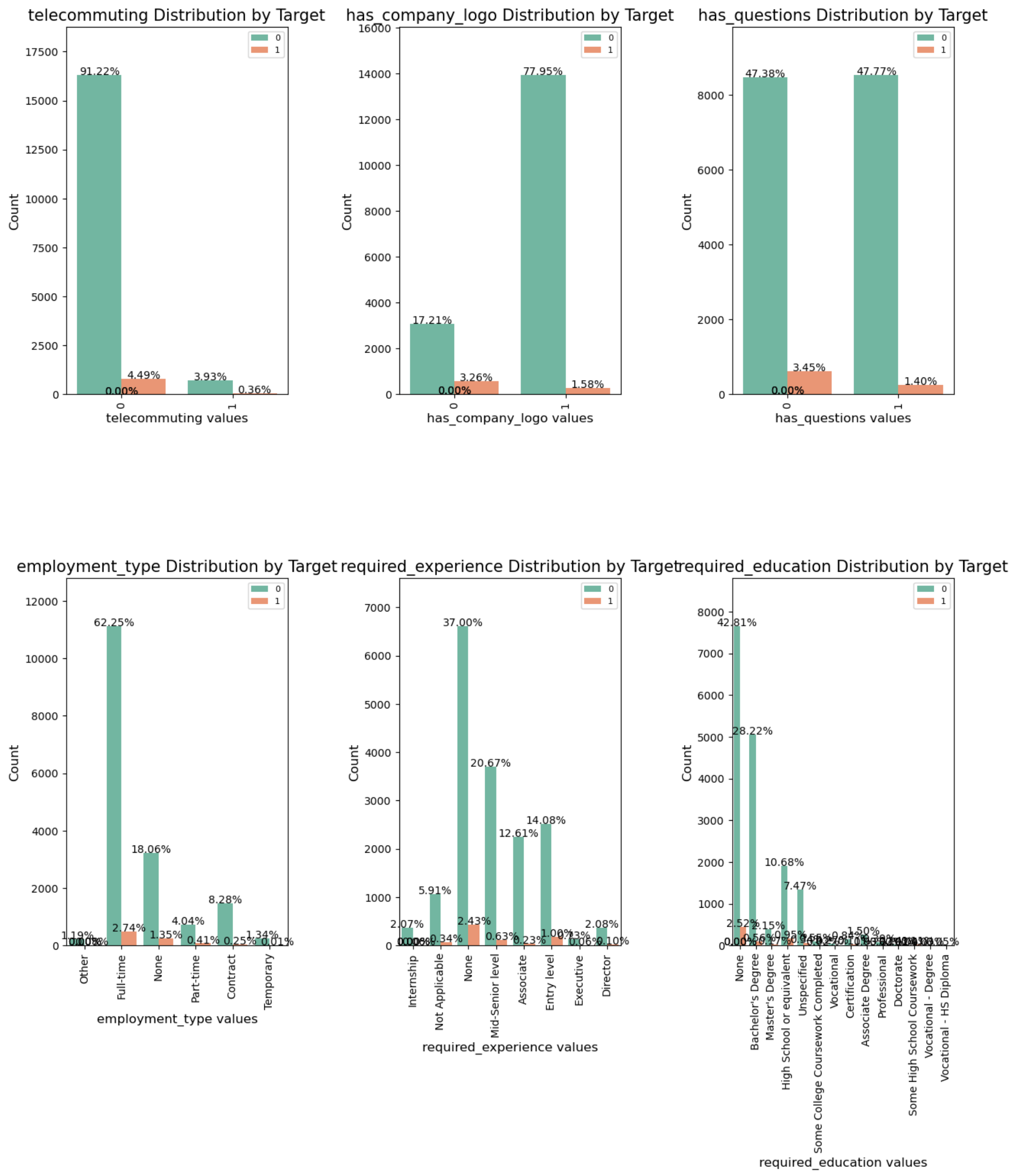


[95]:
```
df.groupby('employment_type').size().plot(kind='barh', color=sns.palettes.
 ↪mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)
```

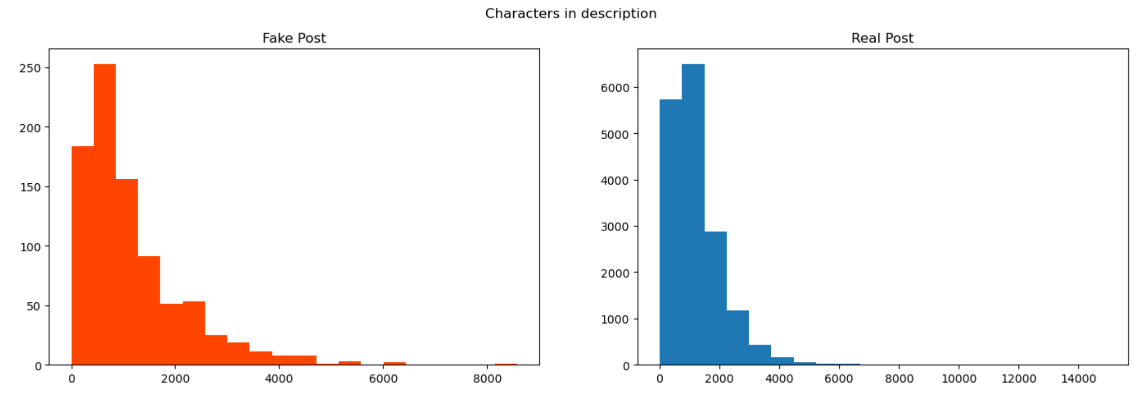


[63]:
```python
cat_cols = ["telecommuting", "has_company_logo", "has_questions",␣
↪"employment_type", "required_experience", "required_education"]
# visualizing categorical variables by target
import matplotlib.gridspec as gridspec  # to do the grid of plots
grid = gridspec.GridSpec(3, 3, wspace=0.5, hspace=0.5)  # The grid of charts
plt.figure(figsize=(15, 25))  # size of figure

# Loop to create subplots for each column
for n, col in enumerate(cat_df[cat_cols]):
    ax = plt.subplot(grid[n])  # feeding the figure of grid
    sns.countplot(x=col, data=cat_df, hue='fraudulent', palette='Set2')
    ax.set_ylabel('Count', fontsize=12)  # y-axis label
    ax.set_title(f'{col} Distribution by Target', fontsize=15)  # title label
    ax.set_xlabel(f'{col} values', fontsize=12)  # x-axis label

    # Adjust tick label font sizes
    ax.tick_params(axis='x', labelsize=10, rotation=90)
    ax.tick_params(axis='y', labelsize=10)
    plt.legend(fontsize=8)

    # Adding percentages above bars
    total = len(cat_df)
    sizes = []  # Get highest values in y
    for p in ax.patches:  # loop through all objects
```

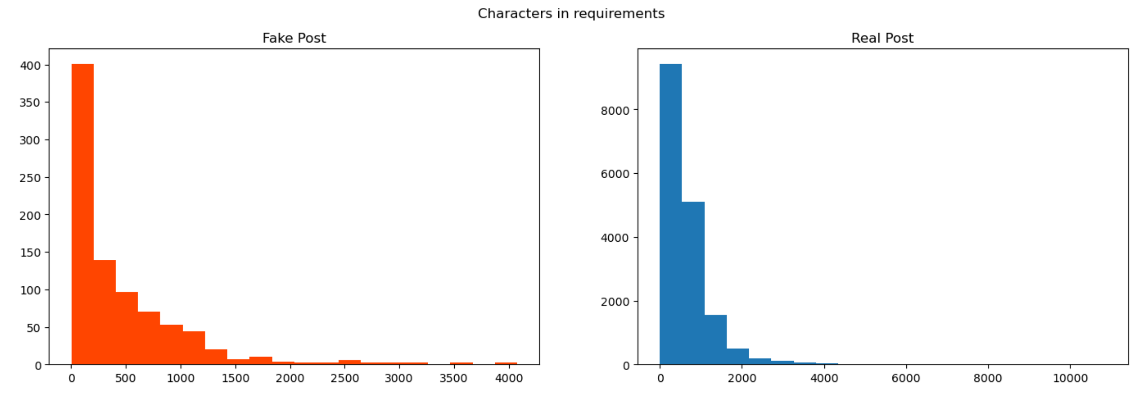
```
        height = p.get_height()
        sizes.append(height)
        ax.text(p.get_x() + p.get_width() / 2.,
                height + 3,
                '{:1.2f}%'.format(height / total * 100),
                ha="center", fontsize=10)
    ax.set_ylim(0, max(sizes) * 1.15)  # set y limit based on highest heights

plt.show()
```



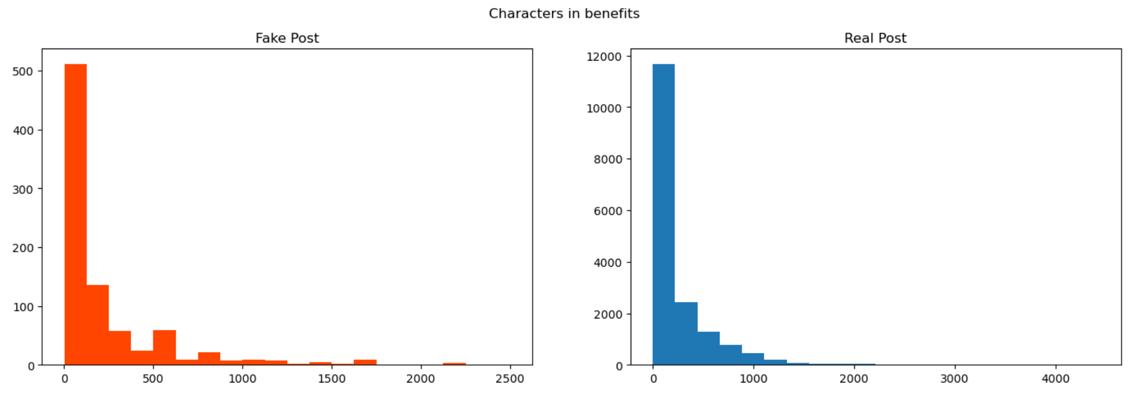
[65]:
```
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
length=text_df[text_df["fraudulent"]==1]['description'].str.len()
ax1.hist(length,bins = 20,color='orangered')
ax1.set_title('Fake Post')
length=text_df[text_df["fraudulent"]==0]['description'].str.len()
ax2.hist(length, bins = 20)
ax2.set_title('Real Post')
fig.suptitle('Characters in description')
plt.show()
```



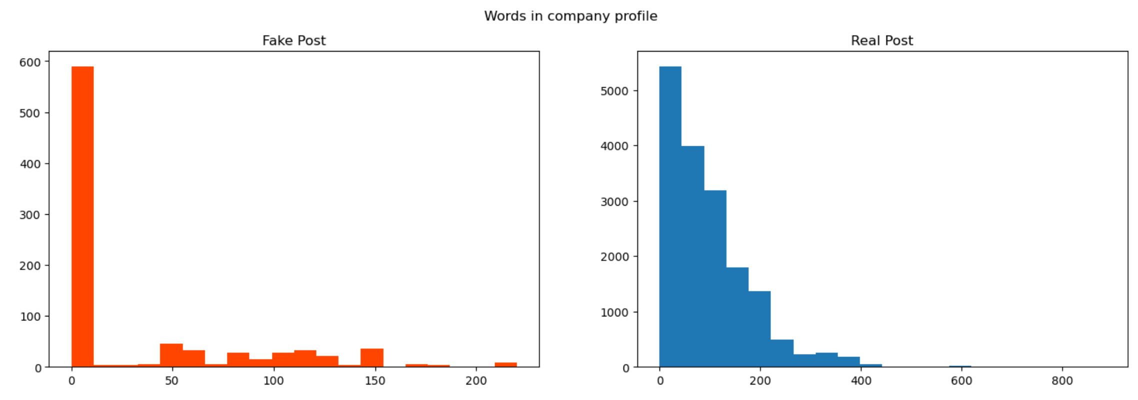
[67]:
```
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
length=text_df[text_df["fraudulent"]==1]['requirements'].str.len()
ax1.hist(length,bins = 20,color='orangered')
ax1.set_title('Fake Post')
length=text_df[text_df["fraudulent"]==0]['requirements'].str.len()
ax2.hist(length,bins = 20)
ax2.set_title('Real Post')
fig.suptitle('Characters in requirements')
plt.show()
```



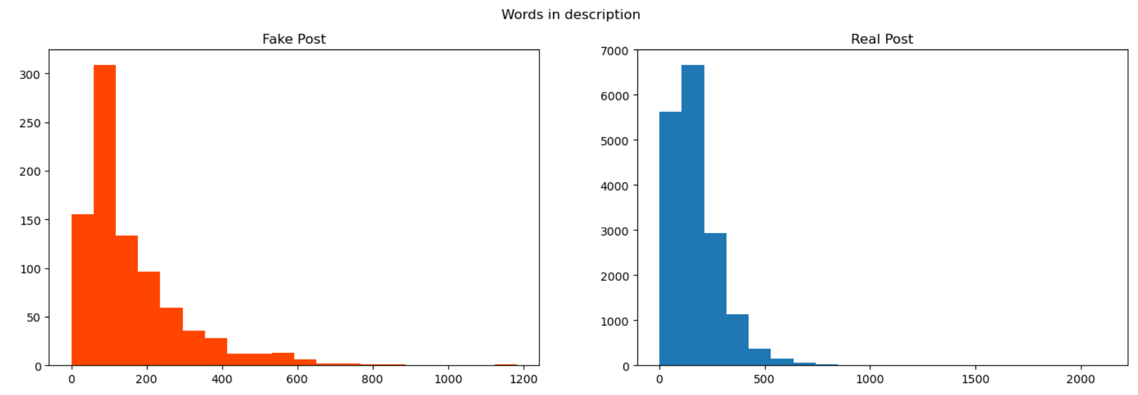
[69]:
```
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
length=text_df[text_df["fraudulent"]==1]['benefits'].str.len()
ax1.hist(length,bins = 20,color='orangered')
ax1.set_title('Fake Post')
length=text_df[text_df["fraudulent"]==0]['benefits'].str.len()
ax2.hist(length,bins = 20)
ax2.set_title('Real Post')
fig.suptitle('Characters in benefits')
plt.show()
```



[71]:
```
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
num=text_df[text_df["fraudulent"]==1]['company_profile'].str.split().map(lambda␣
  ↪x: len(x))
ax1.hist(num,bins = 20,color='orangered')
ax1.set_title('Fake Post')
num=text_df[text_df["fraudulent"]==0]['company_profile'].str.split().map(lambda␣
  ↪x: len(x))
ax2.hist(num, bins = 20)
ax2.set_title('Real Post')
fig.suptitle('Words in company profile')
plt.show()
```

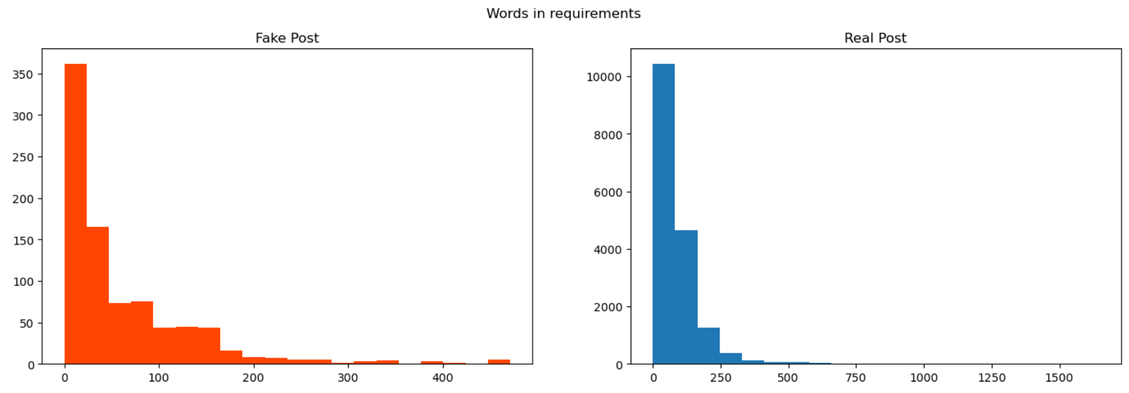


[73]:
```
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
num=text_df[text_df["fraudulent"]==1]['description'].str.split().map(lambda x:␣
  ↪len(x))
ax1.hist(num,bins = 20,color='orangered')
ax1.set_title('Fake Post')
num=text_df[text_df["fraudulent"]==0]['description'].str.split().map(lambda x:␣
  ↪len(x))
ax2.hist(num, bins = 20)
ax2.set_title('Real Post')
fig.suptitle('Words in description')
plt.show()
```

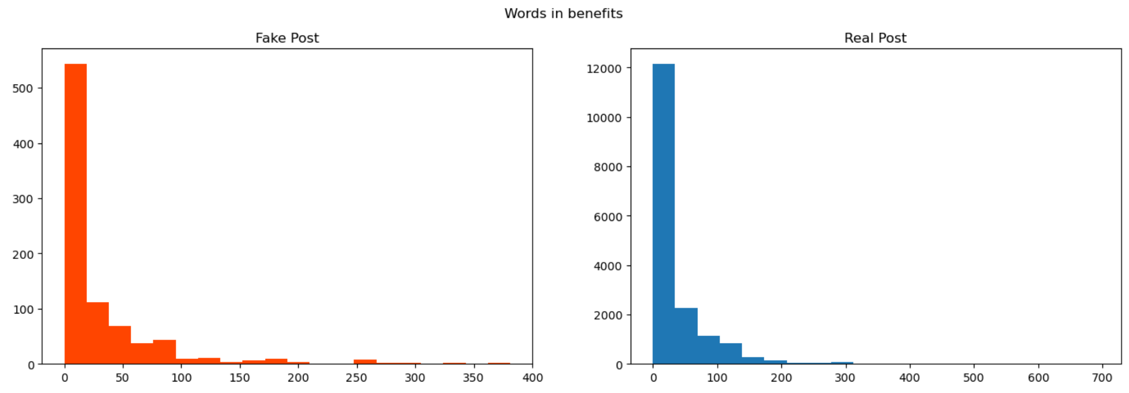


[75]:
```
fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
num=text_df[text_df["fraudulent"]==1]['requirements'].str.split().map(lambda x:␣
  ↪len(x))
ax1.hist(num,bins = 20,color='orangered')
ax1.set_title('Fake Post')
num=text_df[text_df["fraudulent"]==0]['requirements'].str.split().map(lambda x:␣
  ↪len(x))
```

```
ax2.hist(num,bins = 20)
ax2.set_title('Real Post')
fig.suptitle('Words in requirements')
plt.show()
```



```
[77]: fig,(ax1,ax2)= plt.subplots(ncols=2, figsize=(17, 5), dpi=100)
num=text_df[text_df["fraudulent"]==1]['benefits'].str.split().map(lambda x:␣
 ↪len(x))
ax1.hist(num,bins = 20,color='orangered')
ax1.set_title('Fake Post')
num=text_df[text_df["fraudulent"]==0]['benefits'].str.split().map(lambda x:␣
 ↪len(x))
ax2.hist(num, bins = 20)
ax2.set_title('Real Post')
fig.suptitle('Words in benefits')
plt.show()
```



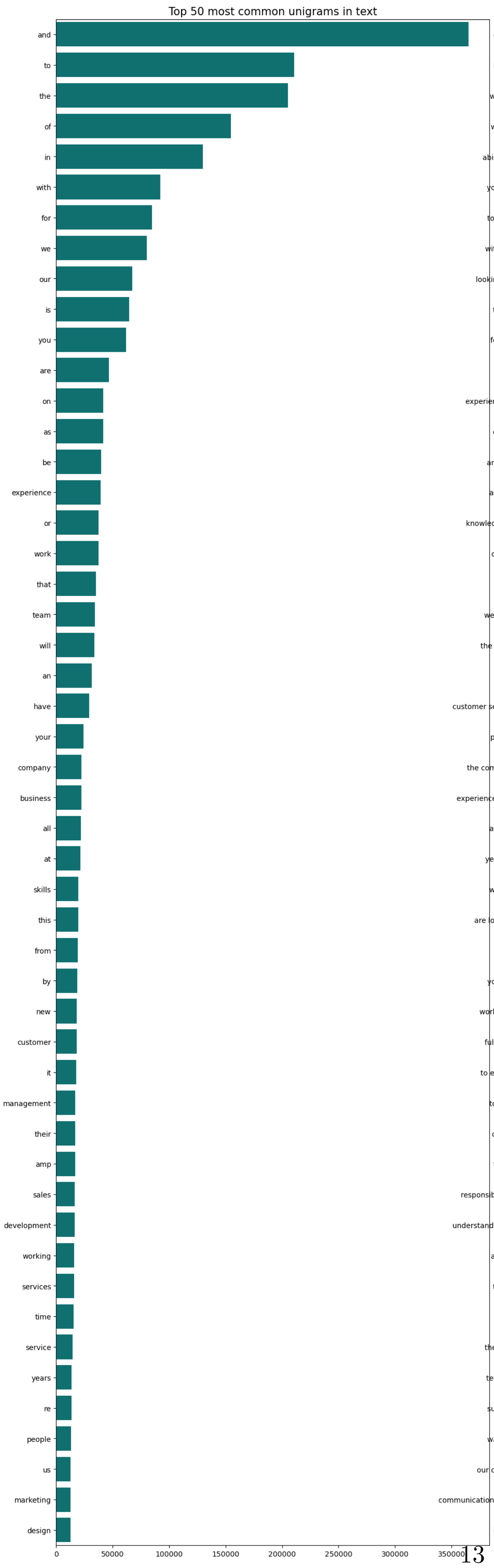
```
[79]: """Concate the text data for preprocessing and modeling"""
```

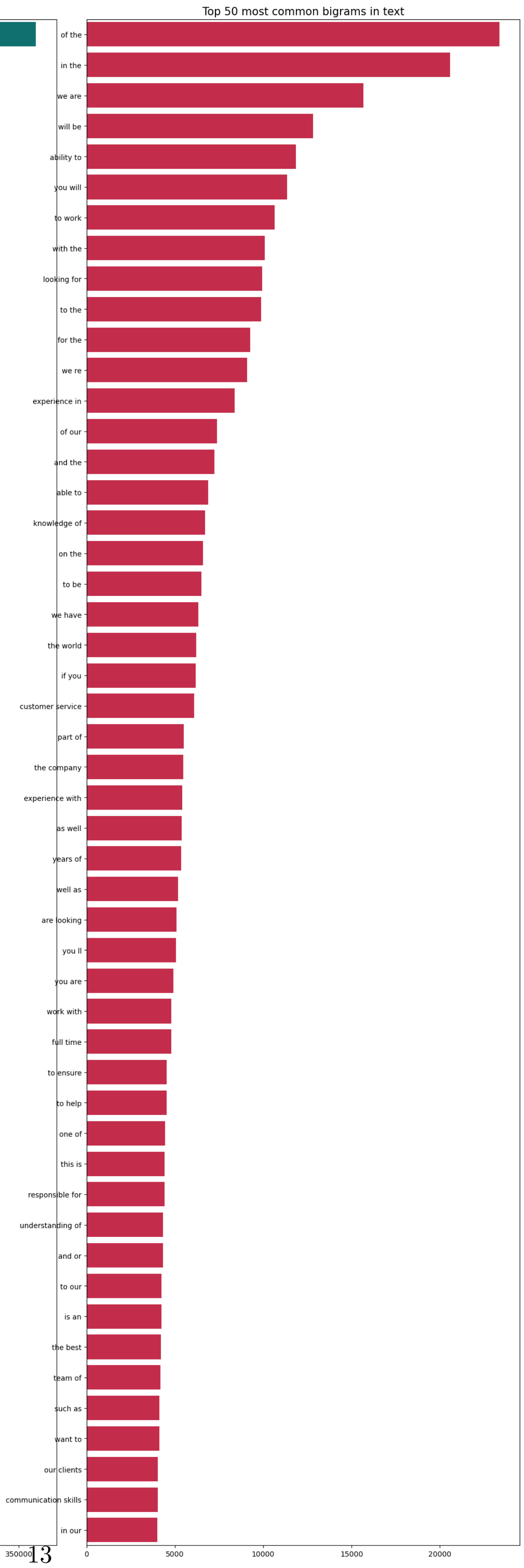
```
text = text_df[text_df.columns[0:-1]].apply(lambda x: ','.join(x.dropna().
 ↪astype(str)),axis=1)
target = df['fraudulent']

print(len(text))
print(len(target))
```

```
17880
17880
```

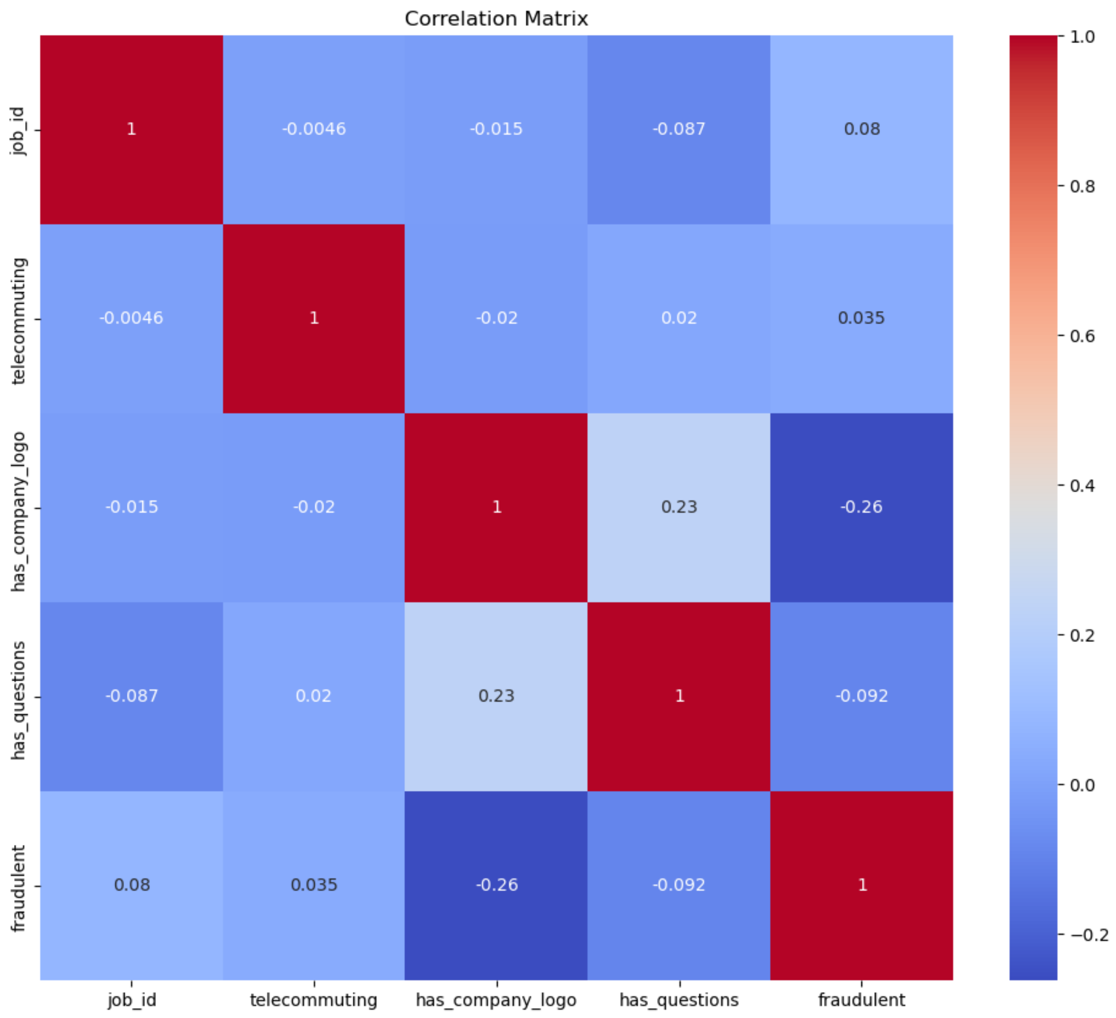
[81]:
```
def get_top_tweet_unigrams(corpus, n=None):
    vec = CountVectorizer(ngram_range=(1, 1)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.
 ↪items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

def get_top_tweet_bigrams(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2, 2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.
 ↪items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

fig, axes = plt.subplots(ncols=2, figsize=(18, 30), dpi=100)
plt.tight_layout()

top_unigrams=get_top_tweet_unigrams(text)[:50]
x,y=map(list,zip(*top_unigrams))
sns.barplot(x=y,y=x, ax=axes[0], color='teal')


top_bigrams=get_top_tweet_bigrams(text)[:50]
x,y=map(list,zip(*top_bigrams))
sns.barplot(x=y,y=x, ax=axes[1], color='crimson')


axes[0].set_ylabel(' ')
axes[1].set_ylabel(' ')

axes[0].set_title('Top 50 most common unigrams in text', fontsize=15)
axes[1].set_title('Top 50 most common bigrams in text', fontsize=15)
```

```
plt.show()
```


Top 50 most common unigrams in text
and
to
the
of
in
with
for
we
our
is
you
are
on
as
be
experience
or
work
that
team
will
an
have
your
company
business
all
at
skills
this
from
by
new
customer
it
management
their
amp
sales
development
working
services
time
service
years
re
people
us
marketing
design
0
50000
100000
150000
200000
250000
300000
350000


Top 50 most common bigrams in text
of the
in the
we are
will be
ability to
you will
to work
with the
looking for
to the
for the
we re
experience in
of our
and the
able to
knowledge of
on the
to be
we have
the world
if you
customer service
part of
the company
experience with
as well
years of
well as
are looking
you ll
you are
work with
full time
to ensure
to help
one of
this is
responsible for
understanding of
and or
to our
is an
the best
team of
such as
want to
our clients
communication skills
in our
0
5000
10000
15000
20000

[97]:
```python
numerical_cols = df.select_dtypes(include=np.number).columns
plt.figure(figsize=(12, 10))
sns.heatmap(df[numerical_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



[85]:
```python
import re
import string

def clean_text(text):
    '''Make text lowercase, remove text in square brackets, remove links,
    remove punctuation, and remove words containing numbers.'''
    text = text.lower()
    text = re.sub(r'\[.*?\]', '', text)  # Raw string to avoid warning
    text = re.sub(r'https?://\S+|www\.\S+', '', text)  # Raw string
    text = re.sub(r'<.*?>+', '', text)  # Raw string
    text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub(r'\n', '', text)  # Raw string
    text = re.sub(r'\w*\d\w*', '', text)  # Raw string
    return text

# Applying the cleaning function
text = text.apply(lambda x: clean_text(x))
text.head(3)
```

[85]:
```
0    marketing internwere  and weve created a groun…
1    customer service  cloud video  seconds the wor…
2    commissioning machinery assistant cmavalor ser…
dtype: object
```

[87]:
```
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')

# appling tokenizer5
text = text.apply(lambda x: tokenizer.tokenize(x))
text.head(3)
```

[87]:
```
0    [marketing, internwere, and, weve, created, a,…
1    [customer, service, cloud, video, seconds, the…
2    [commissioning, machinery, assistant, cmavalor…
dtype: object
```

[89]:
```
stop_words = stopwords.words('english')
def remove_stopwords(text):
    """
    Removing stopwords belonging to english language

    """
    words = [w for w in text if w not in stop_words]
    return words


text = text.apply(lambda x : remove_stopwords(x))
```

[91]:
```
%%time
def combine_text(list_of_text):
    combined_text = ' '.join(list_of_text)
    return combined_text

text = text.apply(lambda x : combine_text(x))
text.head(3)
```

```
CPU times: user 92 ms, sys: 5.3 ms, total: 97.3 ms
Wall time: 96.3 ms
```

[91]:
```
0    marketing internwere weve created groundbreaki…
1    customer service cloud video seconds worlds cl…
2    commissioning machinery assistant cmavalor ser…
dtype: object
```

[93]:
```python
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
auc_buf = []
cnt = 0
predictions = 0
# enumerate the splits and summarize the distributions
for train_ix, test_ix in kfold.split(text, target):
    print('Fold {}'.format(cnt + 1))
    train_X, test_X = text[train_ix], text[test_ix]
    train_y, test_y = target[train_ix], target[test_ix]

    # Appling Count Vectorizer
    count_vectorizer = CountVectorizer()
    train_X_vec = count_vectorizer.fit_transform(train_X)
    test_X_vec = count_vectorizer.transform(test_X)

    lr = LogisticRegression(C=0.1, solver='lbfgs', max_iter=1000, verbose=0,␣
  ↪n_jobs=-1)
    lr.fit(train_X_vec, train_y)
    preds = lr.predict(test_X_vec)

    auc = roc_auc_score(test_y, preds)
    print('{} AUC: {}'.format(cnt, auc))
    auc_buf.append(auc)
    cnt += 1

print('AUC mean score = {:.6f}'.format(np.mean(auc_buf)))
print('AUC std score = {:.6f}'.format(np.std(auc_buf)))
```

```
Fold 1
0 AUC: 0.8457923049876087
Fold 2
1 AUC: 0.8517195809885532
Fold 3
2 AUC: 0.8283043353450458
Fold 4
3 AUC: 0.8720977240415205
Fold 5
4 AUC: 0.8611871313020738
AUC mean score = 0.851820
AUC std score = 0.014753
```

```
[99]: # spliting tthe data for glove
X_train, X_test, y_train, y_test = train_test_split(text, target, test_size=0.
 ↪2, random_state=4, stratify=target)
```

```
[103]: import numpy as np

# Path to the GloVe embeddings file
glove_file_path = '/Users/mohitsalur/Documents/word_embeddings/glove.6B/glove.
 ↪6B.200d.txt'

# Load the GloVe embeddings into a dictionary
embeddings_index = {}
try:
    with open(glove_file_path, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            vectors = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = vectors
    print('Found %s word vectors.' % len(embeddings_index))
except FileNotFoundError:
    print(f"Error: File not found at '{glove_file_path}'. Please verify the␣
 ↪path.")
```

```
Found 400000 word vectors.
```

```
[105]: """ Function Creates a normalized vector for the whole sentence"""
def sent2vec(s):
    words = str(s).lower()
    words = word_tokenize(words)
    words = [w for w in words if not w in stop_words]
    words = [w for w in words if w.isalpha()]
    M = []
    for w in words:
        try:
            M.append(embeddings_index[w])
        except:
            continue
    M = np.array(M)
    v = M.sum(axis=0)
    if type(v) != np.ndarray:
        return np.zeros(200)
    return v / np.sqrt((v ** 2).sum())
```

```
[107]: # create glove features
xtrain_glove = np.array([sent2vec(x) for x in tqdm(X_train)])
xtest_glove = np.array([sent2vec(x) for x in tqdm(X_test)])
```

```
100%|                    | 14304/14304 [00:08<00:00, 1715.70it/s]
100%|                     | 3576/3576 [00:02<00:00, 1783.83it/s]
```

[109]:
```
"""scale the data before any neural net"""
scl = preprocessing.StandardScaler()
xtrain_glove_scl = scl.fit_transform(xtrain_glove)
xtest_glove_scl = scl.transform(xtest_glove)
```

[113]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization,␣
 ↪Activation, Input

# Define the model
model = Sequential()

# Add Input layer (explicit shape declaration)
model.add(Input(shape=(200,)))

model.add(Dense(200, activation='relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

# Summary of the model to verify
model.summary()
```

```
Model: "sequential_1"


 Layer (type)                      Output Shape                 Param #

 dense_4 (Dense)                   (None, 200)                   40,200

 dropout_3 (Dropout)               (None, 200)                        0
 batch_normalization_3            (None, 200)                      800
 (BatchNormalization)

 dense_5 (Dense)                  (None, 100)                   20,100

 dropout_4 (Dropout)              (None, 100)                        0

 batch_normalization_4            (None, 100)                      400
 (BatchNormalization)

 dense_6 (Dense)                  (None, 100)                   10,100

 dropout_5 (Dropout)              (None, 100)                        0

 batch_normalization_5            (None, 100)                      400
 (BatchNormalization)

 dense_7 (Dense)                  (None, 1)                        101


Total params: 72,101 (281.64 KB)

Trainable params: 71,301 (278.52 KB)

Non-trainable params: 800 (3.12 KB)
```

[115]:
```python
model.fit(xtrain_glove_scl, y=y_train, batch_size=64,
          epochs=10, verbose=1,
          validation_data=(xtest_glove_scl, y_test))
```

```
Epoch 1/10
224/224              1s 1ms/step -
accuracy: 0.7189 - loss: 0.5724 - val_accuracy: 0.9645 - val_loss: 0.1341
Epoch 2/10
224/224              0s 994us/step -
accuracy: 0.9590 - loss: 0.1459 - val_accuracy: 0.9720 - val_loss: 0.1016
Epoch 3/10
224/224              0s 954us/step -
accuracy: 0.9606 - loss: 0.1229 - val_accuracy: 0.9709 - val_loss: 0.0996
Epoch 4/10
224/224              0s 950us/step -
accuracy: 0.9693 - loss: 0.0975 - val_accuracy: 0.9746 - val_loss: 0.0888
Epoch 5/10
224/224              0s 945us/step -
```

```
accuracy: 0.9726 - loss: 0.0873 - val_accuracy: 0.9757 - val_loss: 0.0897
Epoch 6/10
224/224              0s 957us/step -
accuracy: 0.9752 - loss: 0.0779 - val_accuracy: 0.9771 - val_loss: 0.0845
Epoch 7/10
224/224              0s 981us/step -
accuracy: 0.9773 - loss: 0.0669 - val_accuracy: 0.9776 - val_loss: 0.0819
Epoch 8/10
224/224              0s 1ms/step -
accuracy: 0.9805 - loss: 0.0587 - val_accuracy: 0.9771 - val_loss: 0.0890
Epoch 9/10
224/224              0s 1ms/step -
accuracy: 0.9844 - loss: 0.0483 - val_accuracy: 0.9779 - val_loss: 0.0859
Epoch 10/10
224/224              0s 940us/step -
accuracy: 0.9824 - loss: 0.0508 - val_accuracy: 0.9782 - val_loss: 0.0950
```

```
[115]: <keras.src.callbacks.history.History at 0x3b6b86000>
```

```
[117]: predictions = model.predict(xtest_glove_scl)
       predictions = np.round(predictions).astype(int)
       print('2 layer sequential neural net on GloVe Feature')
       print ("AUC score :", np.round(roc_auc_score(y_test, predictions),5))
```

```
112/112              0s 598us/step
2 layer sequential neural net on GloVe Feature
AUC score : 0.83766
```

```
[159]: #Implementing BERT
```

```
[119]: new_text = text_df[text_df.columns[0:-1]].apply(lambda x: ','.join(x.dropna().
        ↪astype(str)),axis=1)
       target = df['fraudulent']
```

```
[123]: import re
       import string

       def clean_text(text):
           '''Make text lowercase, remove text in square brackets, remove links,␣
        ↪remove punctuation,
           and remove words containing numbers.'''
           text = text.lower()
           text = re.sub(r'\[.*?\]', '', text)  # Raw string: avoids escape issues
           text = re.sub(r'https?://\S+|www\.\S+', '', text)
           text = re.sub(r'<.*?>+', '', text)
           text = re.sub(r'[%s]' % re.escape(string.punctuation), '', text)
           text = re.sub(r'\n', '', text)
```

```
    text = re.sub(r'\w*\d\w*', '', text)
    return text

# Applying the cleaning function
new_text = new_text.apply(lambda x: clean_text(x))
new_text.head(3)
```

[123]:
```
0    marketing internwere  and weve created a groun…
1    customer service  cloud video  seconds the wor…
2    commissioning machinery assistant cmavalor ser…
dtype: object
```

[125]:
```
# Apply the cleaning function to the entire dataset
new_text_cleaned = new_text.apply(lambda x: clean_text(x))

# Assuming you want to apply it to the entire 'target' column too:
target_cleaned = target

# Check the value counts for the entire target
target_cleaned.value_counts()
```

[125]:
```
fraudulent
0    17014
1      866
Name: count, dtype: int64
```

[127]:
```
# importing the tools
import torch
import transformers as ppb
import warnings
warnings.filterwarnings('ignore')
```

[137]:
```
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.
  ↪DistilBertTokenizer, 'distilbert-base-uncased')

# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

[141]:
```
%%time
# Tokenization
tokenized = new_text_cleaned.apply((lambda x: tokenizer.encode(x, max_length =␣
  ↪60, add_special_tokens=True)))
tokenized.shape
```

```
Truncation was not explicitly activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitly truncate examples to max
```

```
length. Defaulting to 'longest_first' truncation strategy. If you encode pairs
of sequences (GLUE-style) with the tokenizer you can select this strategy more
precisely by providing a specific strategy to `truncation`.
```

```
CPU times: user 45.7 s, sys: 113 ms, total: 45.8 s
Wall time: 45.8 s
```

[141]:
```
(17880,)
```

[143]:
```python
# Padding ==> convert 1D array to 2D array
max_len = 0
for i in tokenized.values:
    if len(i) > max_len:
        max_len = len(i)

padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
np.array(padded).shape
```

[143]:
```
(17880, 60)
```

[145]:
```python
# Masking ==>  ignore (mask) the padding we've added
attention_mask = np.where(padded != 0, 1, 0)
attention_mask.shape
```

[145]:
```
(17880, 60)
```

[147]:
```python
# Deep Learning
input_ids = torch.tensor(padded)
attention_mask = torch.tensor(attention_mask)

with torch.no_grad():
    last_hidden_states = model(input_ids, attention_mask=attention_mask)

last_hidden_states[0].shape
```

[147]:
```
torch.Size([17880, 60, 768])
```

[151]:
```python
features = last_hidden_states[0][:,0,:].numpy()
labels = target_cleaned
```

[165]:
```python
print(features)
```

```
[[ 0.0600361  -0.0864278   0.05234081 … -0.37005758  0.40280047
  -0.13225777]
 [-0.0575145  -0.13685808  0.17555796 … -0.23379458  0.2694164
   0.19146906]
 [-0.02826902 -0.05046919  0.05097549 … -0.21316147  0.19898905
   0.19785255]
 …
 [ 0.02593137 -0.00855615 -0.10332425 … -0.18978877  0.38207355
   0.32542232]
 [ 0.21182308 -0.04256425  0.1196022  … -0.37073663  0.3569724
  -0.05651754]
 [ 0.1979817  -0.09458296 -0.0272286  … -0.41856277  0.30310032
   0.2993164 ]]
```

[167]:
```
print(labels)
```

```
0        0
1        0
2        0
3        0
4        0
        ..
17875    0
17876    0
17877    0
17878    0
17879    0
Name: fraudulent, Length: 17880, dtype: int64
```

[153]:
```
train_features, test_features, train_labels, test_labels =␣
 ↪train_test_split(features, labels)
```

[155]:
```
# train default para
lr_clf = LogisticRegression()
lr_clf.fit(train_features, train_labels)
```

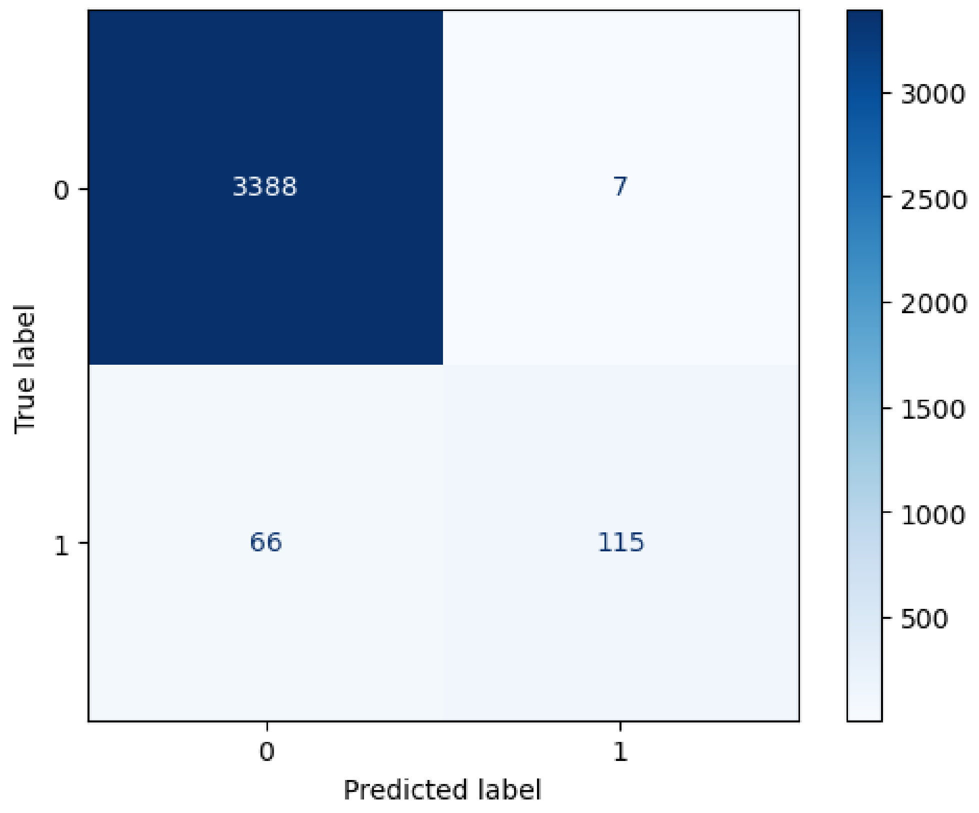
[155]:
```
LogisticRegression()
```

[157]:
```
predictions = lr_clf.predict(test_features)
predictions = np.round(predictions).astype(int)
print ("AUC score :", np.round(roc_auc_score(test_labels, predictions),5))
```

```
AUC score : 0.79251
```

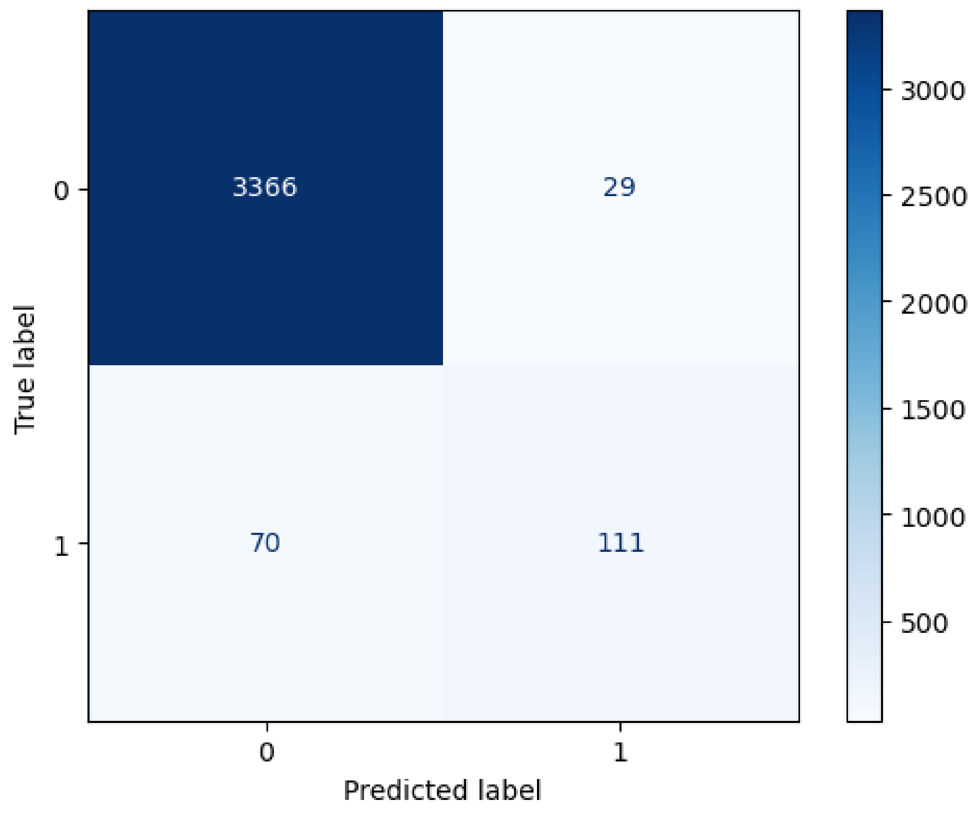
[193]:
```
import xgboost as xgb
from sklearn.metrics import accuracy_score, roc_auc_score,␣
 ↪precision_recall_fscore_support, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import torch
import matplotlib.pyplot as plt

# Assuming `features` is the extracted BERT features (last_hidden_states[0][:,␣
 ↪0, :].numpy())
# Assuming `labels` are your target labels (e.g., target_cleaned)
```

```python
# Step 1: Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.2, random_state=42)

# Step 2: Create DMatrix for XGBoost
train_dmatrix = xgb.DMatrix(X_train, label=y_train)
test_dmatrix = xgb.DMatrix(X_test, label=y_test)

# Step 3: Define XGBoost parameters
params = {
    'objective': 'binary:logistic',  # Binary classification
    'eval_metric': 'auc',  # For binary classification, we use AUC
    'max_depth': 6,  # You can tune this parameter
    'learning_rate': 0.1,  # You can tune this parameter
    'n_estimators': 100  # Number of boosting rounds
}

# Step 4: Train the XGBoost model
xgb_model = xgb.train(params, train_dmatrix, num_boost_round=100)

# Step 5: Make predictions on the test set
y_pred_prob = xgb_model.predict(test_dmatrix)  # Probabilities for binary␣
 ↪classification

# Step 6: Convert probabilities to binary labels (0 or 1)
y_pred = (y_pred_prob > 0.5).astype(int)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

# Step 8: Calculate precision and recall for both 0 and 1 labels
precision, recall, _, _ = precision_recall_fscore_support(y_test, y_pred,␣
 ↪labels=[0, 1])

# Step 9: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
plt.show()

# Print metrics
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'AUC Score: {roc_auc:.5f}')
print(f'Precision (label 0): {precision[0]:.5f}')
print(f'Recall (label 0): {recall[0]:.5f}')
print(f'Precision (label 1): {precision[1]:.5f}')
print(f'Recall (label 1): {recall[1]:.5f}')
```



```
Accuracy: 97.96%
AUC Score: 0.97073
Precision (label 0): 0.98089
Recall (label 0): 0.99794
Precision (label 1): 0.94262
Recall (label 1): 0.63536
```

[195]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score,␣
↪precision_recall_fscore_support, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Assuming `features` is the extracted BERT features (last_hidden_states[0][:,␣
↪0, :].numpy())
# Assuming `labels` are your target labels (e.g., target_cleaned)
# Step 1: Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
↪test_size=0.2, random_state=42)

# Step 2: Train a Logistic Regression model
lr_clf = LogisticRegression(max_iter=1000)  # Set max_iter to a higher value if␣
↪needed
lr_clf.fit(X_train, y_train)

# Step 3: Make predictions on the test set
y_pred = lr_clf.predict(X_test)
y_pred_prob = lr_clf.predict_proba(X_test)[:, 1]  # Probabilities for class 1

# Step 4: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

# Step 5: Calculate precision and recall for both 0 and 1 labels
precision, recall, _, _ = precision_recall_fscore_support(y_test, y_pred,␣
↪labels=[0, 1])

# Step 6: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
plt.show()

# Print metrics
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'AUC Score: {roc_auc:.5f}')
print(f'Precision (label 0): {precision[0]:.5f}')
print(f'Recall (label 0): {recall[0]:.5f}')
print(f'Precision (label 1): {precision[1]:.5f}')
print(f'Recall (label 1): {recall[1]:.5f}')
```



```
Accuracy: 97.23%
AUC Score: 0.95602
Precision (label 0): 0.97963
Recall (label 0): 0.99146
Precision (label 1): 0.79286
Recall (label 1): 0.61326
```

[197]:
```
import torch
import torch_geometric.nn as pyg_nn
import torch.nn.functional as F
from torch_geometric.data import Data
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Assuming the node features are BERT embeddings and are already in the␣
 ↪`features` variable
# target is the binary labels for your classification task

# Step 1: Prepare your data for PyTorch Geometric
# Assuming node_features is of shape (num_nodes, feature_dim)
# edge_index is the adjacency matrix, derived earlier (shape: [2, num_edges])

# Convert node features and labels to PyTorch tensors
node_features = torch.tensor(features, dtype=torch.float32)
target = torch.tensor(target.values, dtype=torch.long)  # Binary labels (0 or 1)

# Assuming edge_index is prepared (from cosine similarity as shown earlier)
edge_index = torch.tensor(edge_index, dtype=torch.long)

# Step 2: Define the GNN model
class GNN(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GNN, self).__init__()
        self.conv1 = pyg_nn.GCNConv(input_dim, hidden_dim)  # First GCN layer
        self.conv2 = pyg_nn.GCNConv(hidden_dim, output_dim)  # Second GCN layer

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.relu(self.conv1(x, edge_index))  # Apply first GCN layer + ReLU␣
 ↪activation
        x = self.conv2(x, edge_index)  # Apply second GCN layer
        return x

# Step 3: Instantiate the model
input_dim = node_features.shape[1]  # Number of features per node (BERT␣
 ↪embeddings)
hidden_dim = 128  # Hidden layer size
output_dim = 2  # Output dimension for binary classification (fraudulent or not)

model = GNN(input_dim, hidden_dim, output_dim)

# Step 4: Prepare the data for PyTorch Geometric
data = Data(x=node_features, edge_index=edge_index, y=target)

# Step 5: Define the optimizer and loss function
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.CrossEntropyLoss()

# Step 6: Training the model
model.train()
num_epochs = 100  # Number of epochs

for epoch in range(num_epochs):
    optimizer.zero_grad()

    # Forward pass
    out = model(data)
    # Compute loss
    loss = criterion(out, target)

    # Backward pass
    loss.backward()

    # Optimize the model parameters
    optimizer.step()

    if epoch % 10 == 0:  # Print loss every 10 epochs
        print(f'Epoch {epoch}, Loss: {loss.item()}')

# Step 7: Evaluate the model
model.eval()
with torch.no_grad():
    out = model(data)

    # Get the predicted labels by taking the argmax of the output logits
    _, predicted = torch.max(out, dim=1)

    # Calculate AUC score using the predicted probabilities for class 1
    y_pred_prob = F.softmax(out, dim=1)[:, 1]  # Probabilities for class 1
    auc_score = roc_auc_score(target.numpy(), y_pred_prob.numpy())
    print(f'AUC score: {auc_score:.5f}')

    # Confusion Matrix and Classification Report
    print("Confusion Matrix:")
    print(confusion_matrix(target.numpy(), predicted.numpy()))
    print("\nClassification Report:")
    print(classification_report(target.numpy(), predicted.numpy()))
```

```
Epoch 0, Loss: 0.3196116089820862
Epoch 10, Loss: 0.16227371990680695
Epoch 20, Loss: 0.1812790036201477
Epoch 30, Loss: 0.13694632053375244
Epoch 40, Loss: 0.1352633237838745
Epoch 50, Loss: 0.1257287859916687
Epoch 60, Loss: 0.11935460567474365
Epoch 70, Loss: 0.11397257447242737
Epoch 80, Loss: 0.10881032794713974
Epoch 90, Loss: 0.1040743961930275
AUC score: 0.94071
Confusion Matrix:
[[16926    88]
 [  496   370]]
```

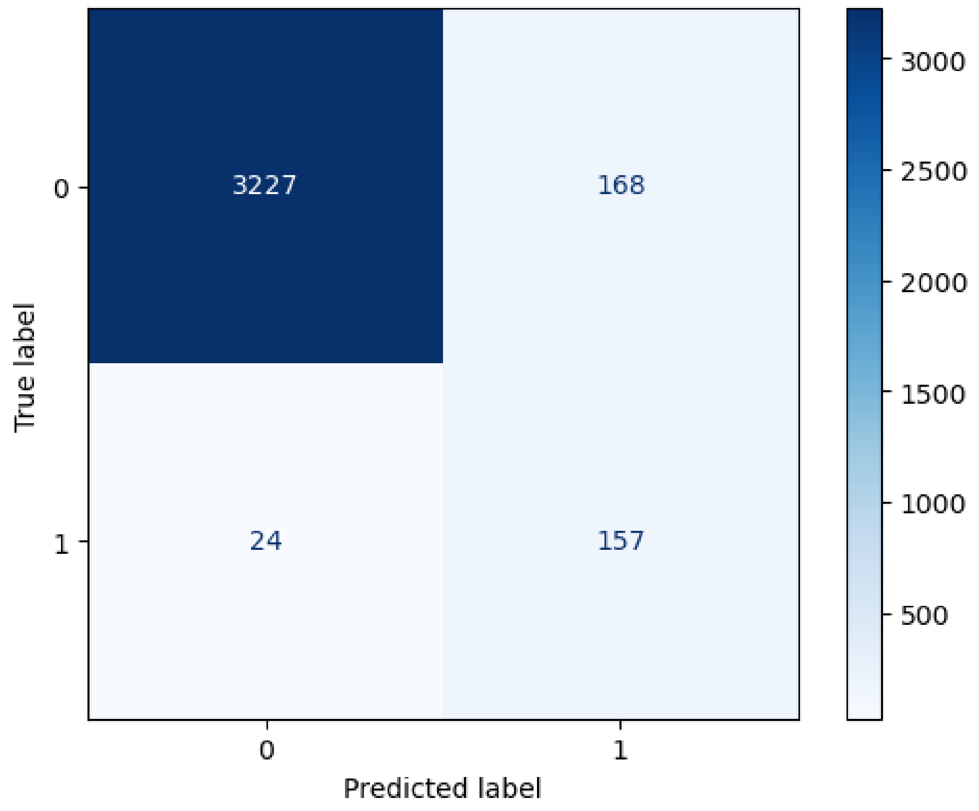
```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     17014
           1       0.81      0.43      0.56       866

    accuracy                           0.97     17880
   macro avg       0.89      0.71      0.77     17880
weighted avg       0.96      0.97      0.96     17880
```

[217]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score,␣
 ↪precision_recall_fscore_support, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE  # For resampling the minority class
import matplotlib.pyplot as plt

# Assuming `features` is the extracted BERT features (last_hidden_states[0][:,␣
 ↪0, :].numpy())
# Assuming `labels` are your target labels (e.g., target_cleaned)

# Step 1: Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.2, random_state=42)

# Step 2: Resample the training data using SMOTE (oversample the minority class)
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Step 3: Train a Logistic Regression model with class weights (balanced) or␣
 ↪without resampling
lr_clf = LogisticRegression(max_iter=1000, class_weight='balanced')  # You can␣
 ↪also use class_weight='balanced' if not using SMOTE
lr_clf.fit(X_train_resampled, y_train_resampled)  # Train on the resampled data

# Step 4: Make predictions on the test set
y_pred = lr_clf.predict(X_test)
y_pred_prob = lr_clf.predict_proba(X_test)[:, 1]  # Probabilities for class 1

# Step 5: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

# Step 6: Calculate precision and recall for both 0 and 1 labels
precision, recall, _, _ = precision_recall_fscore_support(y_test, y_pred,␣
 ↪labels=[0, 1])
```

```
# Step 7: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
plt.show()

# Print metrics
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'AUC Score: {roc_auc:.5f}')
print(f'Precision (label 0): {precision[0]:.5f}')
print(f'Recall (label 0): {recall[0]:.5f}')
print(f'Precision (label 1): {precision[1]:.5f}')
print(f'Recall (label 1): {recall[1]:.5f}')
```

```
python(17377) MallocStackLogging: can't turn off malloc stack logging because it
was not enabled.
```



```
Accuracy: 94.63%
```

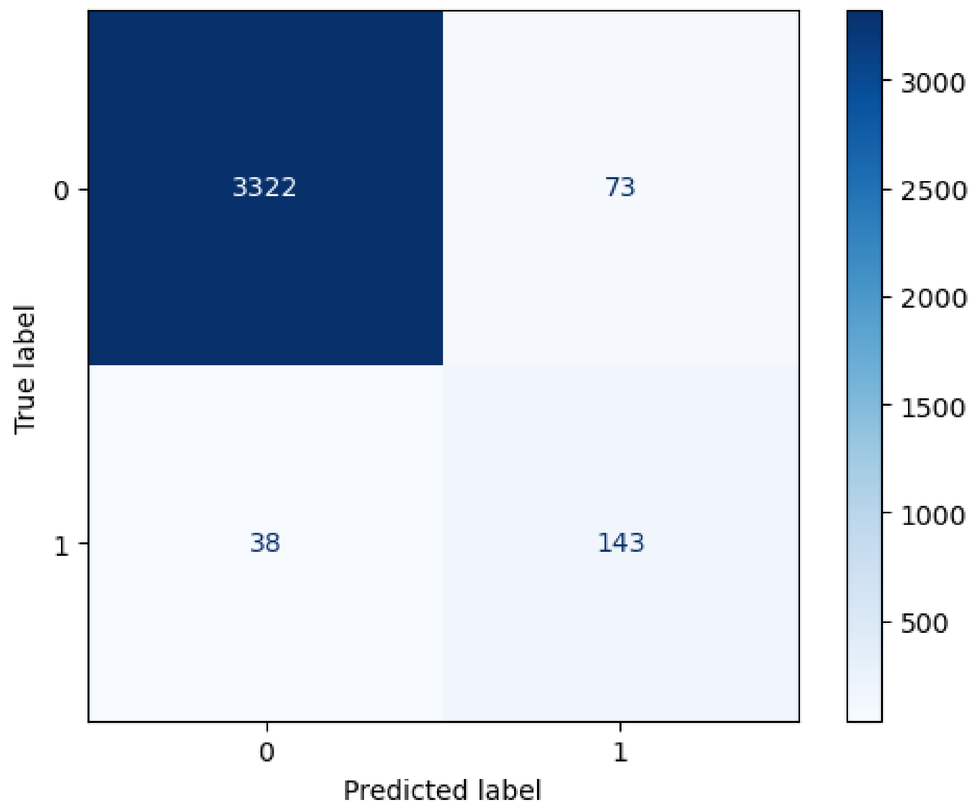
```
AUC Score: 0.95883
Precision (label 0): 0.99262
Recall (label 0): 0.95052
Precision (label 1): 0.48308
Recall (label 1): 0.86740
```

[219]:
```
unset MallocStackLogging
```

```
  Cell In[219], line 1
    unset MallocStackLogging
          ^
SyntaxError: invalid syntax
```

[221]:
```
import xgboost as xgb
from sklearn.metrics import accuracy_score, roc_auc_score,␣
 ↪precision_recall_fscore_support, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE  # For resampling the minority class
import matplotlib.pyplot as plt

# Assuming `features` is the extracted BERT features (last_hidden_states[0][:,␣
 ↪0, :].numpy())
# Assuming `labels` are your target labels (e.g., target_cleaned)

# Step 1: Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.2, random_state=42)

# Step 2: Resample the training data using SMOTE (oversample the minority class)
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Step 3: Train an XGBoost model with class weights or using scale_pos_weight␣
 ↪for imbalanced classes
# scale_pos_weight is usually set to the ratio of negative class to positive␣
 ↪class in imbalanced datasets
neg_pos_ratio = (y_train == 0).sum() / (y_train == 1).sum()  # Calculate ratio␣
 ↪of negative to positive class
xgb_clf = xgb.XGBClassifier(
    max_depth=6,
    n_estimators=100,
    scale_pos_weight=neg_pos_ratio,  # Adjust class imbalance
    use_label_encoder=False,
    eval_metric='logloss'
)
```

```python
# Train the XGBoost model on the resampled data
xgb_clf.fit(X_train_resampled, y_train_resampled)

# Step 4: Make predictions on the test set
y_pred = xgb_clf.predict(X_test)
y_pred_prob = xgb_clf.predict_proba(X_test)[:, 1]  # Probabilities for class 1

# Step 5: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

# Step 6: Calculate precision and recall for both 0 and 1 labels
precision, recall, _, _ = precision_recall_fscore_support(y_test, y_pred,␣
 ↪labels=[0, 1])

# Step 7: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
plt.show()

# Print metrics
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'AUC Score: {roc_auc:.5f}')
print(f'Precision (label 0): {precision[0]:.5f}')
print(f'Recall (label 0): {recall[0]:.5f}')
print(f'Precision (label 1): {precision[1]:.5f}')
print(f'Recall (label 1): {recall[1]:.5f}')
```



```
Accuracy: 96.90%
AUC Score: 0.96162
Precision (label 0): 0.98869
Recall (label 0): 0.97850
Precision (label 1): 0.66204
Recall (label 1): 0.79006
```

[2]:
```
import torch
import torch_geometric.nn as pyg_nn
import torch.nn.functional as F
from torch_geometric.data import Data
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Assuming the node features are BERT embeddings and are already in the␣
↪`features` variable
# target is the binary labels for your classification task

# Step 1: Prepare your data for PyTorch Geometric
# Assuming node_features is of shape (num_nodes, feature_dim)
# edge_index is the adjacency matrix, derived earlier (shape: [2, num_edges])

# Print the target variable to check its type and content
print(f"Type of target before conversion: {type(target)}")
print(f"Content of target: {target}")

# Convert node features to PyTorch tensor
node_features = torch.tensor(features, dtype=torch.float32)

# If target is already a tensor, we don't need to convert it
if isinstance(target, torch.Tensor):
    target = target.long()  # Ensure it's the correct type (long) for␣
 ↪classification
elif isinstance(target, (pd.Series, np.ndarray)):
    target = torch.tensor(target, dtype=torch.long)  # Binary labels (0 or 1)
elif hasattr(target, 'values'):  # Handle if it's a pandas DataFrame
    target = torch.tensor(target.values, dtype=torch.long)
else:
    raise TypeError("Target is not of a valid type.")

# Assuming edge_index is prepared (from cosine similarity as shown earlier)
edge_index = torch.tensor(edge_index, dtype=torch.long)

# Step 2: Define the GNN model
class GNN(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GNN, self).__init__()
        self.conv1 = pyg_nn.GCNConv(input_dim, hidden_dim)  # First GCN layer
        self.conv2 = pyg_nn.GCNConv(hidden_dim, output_dim)  # Second GCN layer

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.relu(self.conv1(x, edge_index))  # Apply first GCN layer + ReLU␣
 ↪activation
        x = self.conv2(x, edge_index)  # Apply second GCN layer
        return x

# Step 3: Instantiate the model
input_dim = node_features.shape[1]  # Number of features per node (BERT␣
 ↪embeddings)
hidden_dim = 128  # Hidden layer size
output_dim = 2  # Output dimension for binary classification (fraudulent or not)

model = GNN(input_dim, hidden_dim, output_dim)

# Step 4: Prepare the data for PyTorch Geometric
data = Data(x=node_features, edge_index=edge_index, y=target)
```

```python
# Step 5: Define the optimizer and loss function
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.CrossEntropyLoss()

# Step 6: Training the model
model.train()
num_epochs = 100  # Number of epochs

for epoch in range(num_epochs):
    optimizer.zero_grad()

    # Forward pass
    out = model(data)

    # Compute loss
    loss = criterion(out, target)

    # Backward pass
    loss.backward()

    # Optimize the model parameters
    optimizer.step()

    if epoch % 10 == 0:  # Print loss every 10 epochs
        print(f'Epoch {epoch}, Loss: {loss.item()}')

# Step 7: Evaluate the model
model.eval()
with torch.no_grad():
    out = model(data)

    # Get the predicted labels by taking the argmax of the output logits
    _, predicted = torch.max(out, dim=1)

    # Calculate AUC score using the predicted probabilities for class 1
    y_pred_prob = F.softmax(out, dim=1)[:, 1]  # Probabilities for class 1
    auc_score = roc_auc_score(target.numpy(), y_pred_prob.numpy())
    print(f'AUC score: {auc_score:.5f}')

    # Confusion Matrix and Classification Report
    print("Confusion Matrix:")
    print(confusion_matrix(target.numpy(), predicted.numpy()))
    print("\nClassification Report:")
    print(classification_report(target.numpy(), predicted.numpy()))
```

```
---------------------------------------------------------------------------
```

```
NameError                                 Traceback (most recent call last)
Cell In[2], line 17
      7 import numpy as np
      9 # Assuming the node features are BERT embeddings and are already in the
  ↪`features` variable
     10 # target is the binary labels for your classification task
     11 
    (…)
     15 
     16 # Print the target variable to check its type and content
---> 17 print(f"Type of target before conversion: {type(target)}")
     18 print(f"Content of target: {target}")
     20 # Convert node features to PyTorch tensor

NameError: name 'target' is not defined
```

```
[ ]:
```