TEXAS McCombs
The University of Texas at Austin
McCombs School of Business

Great Learning

NATURAL LANGUAGE PROCESSING CHATBOT INTERFACE

BATCH

Post-Graduation Program AIML Jan'24
Capstone - NLP 1 Group 3

ABSTRACT

The project is focused on improving industrial safety by leveraging machine learning (ML) and natural language processing (NLP) through the development of a Chatbot.

AUTHORS

1. Aiswarya Ravichandran
2. Christal Reena
3. Jagannath Prasad
4. Lalith Kishore
5. Ramesh Kumar Kailasam
6. Surya Prakash

DATE OF SUBMISSION

15/12/2024

Table of Contents

List of Figures

FINAL REPORT

Milestone 1

1. Summary of problem statement, data and findings

1.1 Problem Statement

The project is focused on improving industrial safety by leveraging machine learning (ML) and natural language processing (NLP) through the development of a chatbot. The chatbot will analyze accident data from multiple plants across different countries to help professionals identify safety risks and understand the causes of accidents.

The dataset includes detailed records of workplace accidents, including information on the accident's severity, the risk factors involved, the location, the type of injury, and a description of how the incident occurred. The objective is to design a chatbot that can process this information and provide insights into safety hazards by highlighting key risks based on the descriptions of the accidents. The goal is to help prevent future accidents by enabling safety managers to quickly assess and address potential dangers.

In essence, the Chatbot aims to automate the identification of safety risks and support decision-making in industrial environments, ultimately improving worker safety.

1.2 Data and Findings

Initially, to import the data and conduct further analysis, we utilized the libraries which are attached in Annexure I.

1.2.1 Data Overview

The data was then imported using Google Drive, and the first few rows were checked.

	Unnamed: 0	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
0	0	2016-01-01	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
1	1	2016-01-02	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
2	2	2016-01-06	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
3	3	2016-01-08	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
4	4	2016-01-10	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

Figure 1: Data Overview

- The dataset consists of 425 rows and 11 columns.
- The columns include a mix of categorical attributes such as 'Country', 'City', 'Industry Sector', 'Accident Level', and 'Employee Type', as well as a date column ('Date').

1.2.2 Data Cleansing

- The 'Unnamed: 0' column was removed, and columns were renamed for clarity:
  - 'Data' to 'Date',
  - 'Countries' to 'Country',
  - 'Local' to 'City',
  - 'Genre' to 'Gender',
  - 'Employee or Third Party' to 'Employee type'.
- Seven duplicate rows were found and removed, reducing the dataset to 418 rows and 10 columns.
- No missing values were present after the cleansing process.

1.2.3 Data Preprocessing

- Roman numerals in the 'Accident Level' and 'Potential Accident Level' columns were converted to numeric values using the roman library.
- The data types of the columns were adjusted to appropriate types:
  - 'Date' was converted to datetime,
  - Categorical columns ('Country', 'City', 'Accident Level', etc.) were converted to category data types.
- Specific values in the 'Critical Risk' and 'Employee Type' columns were standardized (e.g., "Third Party" was replaced with "Contractor").

- Additional temporal features were extracted from the 'Date' column, including:
    - Year, Month, Day, Weekday, and Week of Year.

1.2.4 Final Dataset

- The cleaned and processed dataset now contains 418 rows and 10 columns, with the new features added for further analysis. This dataset is now ready for deeper exploration and modeling.

	Date	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Description	Year	Month	Day	Weekday	WeekofYear
0	2016-01-01	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	While removing the drill rod of the Jumbo 08 f...	2016	1	1	Friday	53
1	2016-01-02	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...	2016	1	2	Saturday	53
2	2016-01-06	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	In the sub-station MILPO located at level +170...	2016	1	6	Wednesday	1
3	2016-01-08	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	Being 9:45 am. approximately in the Nv. 1880 C...	2016	1	8	Friday	1
4	2016-01-10	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	Approximately at 11:45 a.m. in circumstances t...	2016	1	10	Sunday	1

Figure 2: Overview of Final Dataset

2. Approach to EDA and Pre-Processing

2.1 Statistical Analysis

In this step, a detailed statistical analysis of the dataset was performed. The analysis involved generating descriptive statistics and determining the frequency distribution of various columns in the dataset.

2.1.1 Key findings from the frequency distribution include

- **Country:** The majority of accidents were reported in **Country_01 (248 cases),** followed by Country_02 (129) and Country_03 (41).
- **City: Local_03** had the most reported accidents **(89)**, while Local_12 and Local_09 had the least (4 and 2, respectively).
- **Industry Sector: Mining** reported the highest number of accidents **(237)**, followed by Metals (134).

- **Accident Level:** The majority of accidents were classified as minor **(Level 1) with 309** cases.
- **Potential Accident Level:** The highest number of potential accidents was categorized as **Level 4 (Very High Potential) with 141** cases.
- **Gender:** The majority of accident reports involved **male employees (396).**
- **Employee Type: Contractors (185 cases)** experienced the most accidents, followed by regular employees (178) and remote contractors (55).
- **Critical Risk:** The "**Others**" category was the most common cause of critical risk (**229** occurrences).

This analysis provides insights into the frequency of accidents based on various factors such as country, city, industry sector, and risk level.

2.2 Descriptive Analysis Report

The Univariate analysis was performed to explore the distribution of key variables in the dataset through visualizations and value counts.

2.2.1 Univariate Analysis

2.2.1.1 Gender Distribution: The gender distribution shows that 95% of the accident reports involved male employees, while only 5% involved female employees.

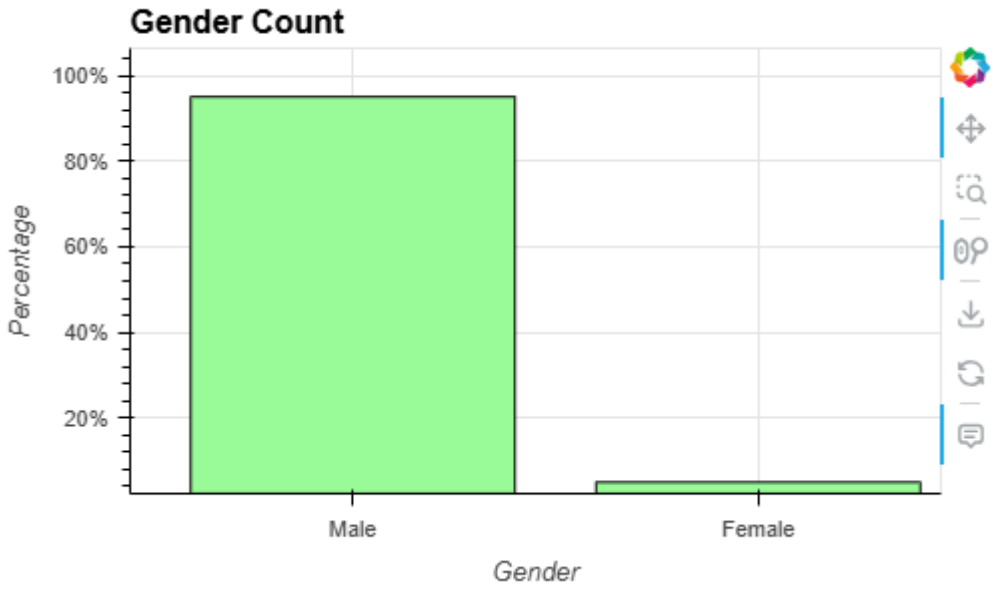

Figure 3: Distribution of Gender

2.2.1.2 Accident Distribution by Country: Accidents were most frequently reported in Country_01, followed by Country_02 and Country_03.

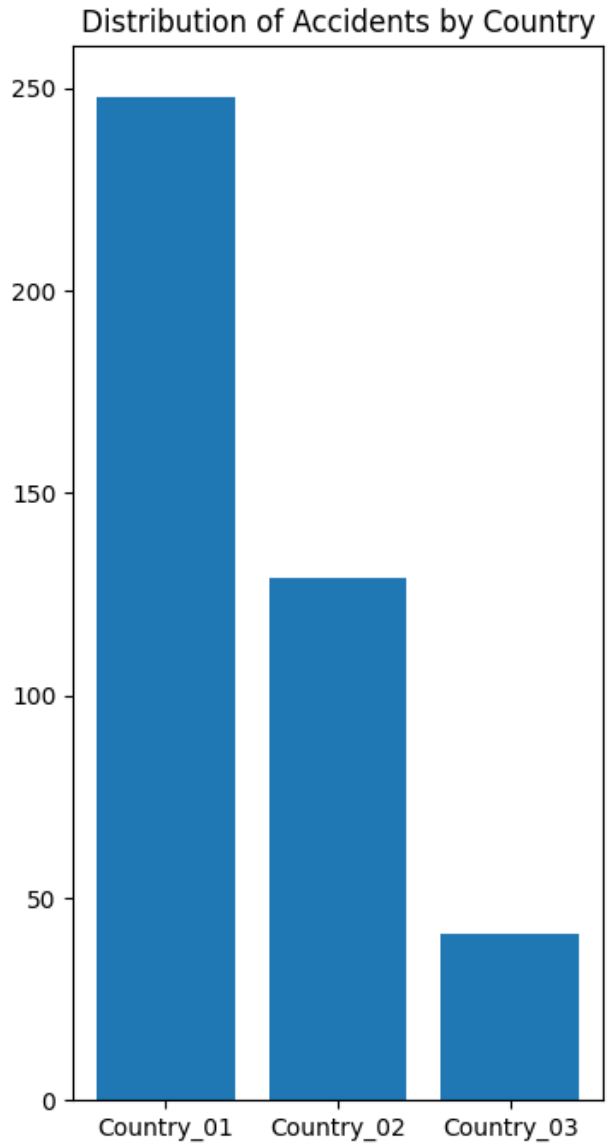

Figure 4: Distribution of Accidents by Country

2.2.1.3 Accident Distribution by Industry Sector: The Mining sector had the highest number (56.7%) of reported accidents, followed by the Metals sector.

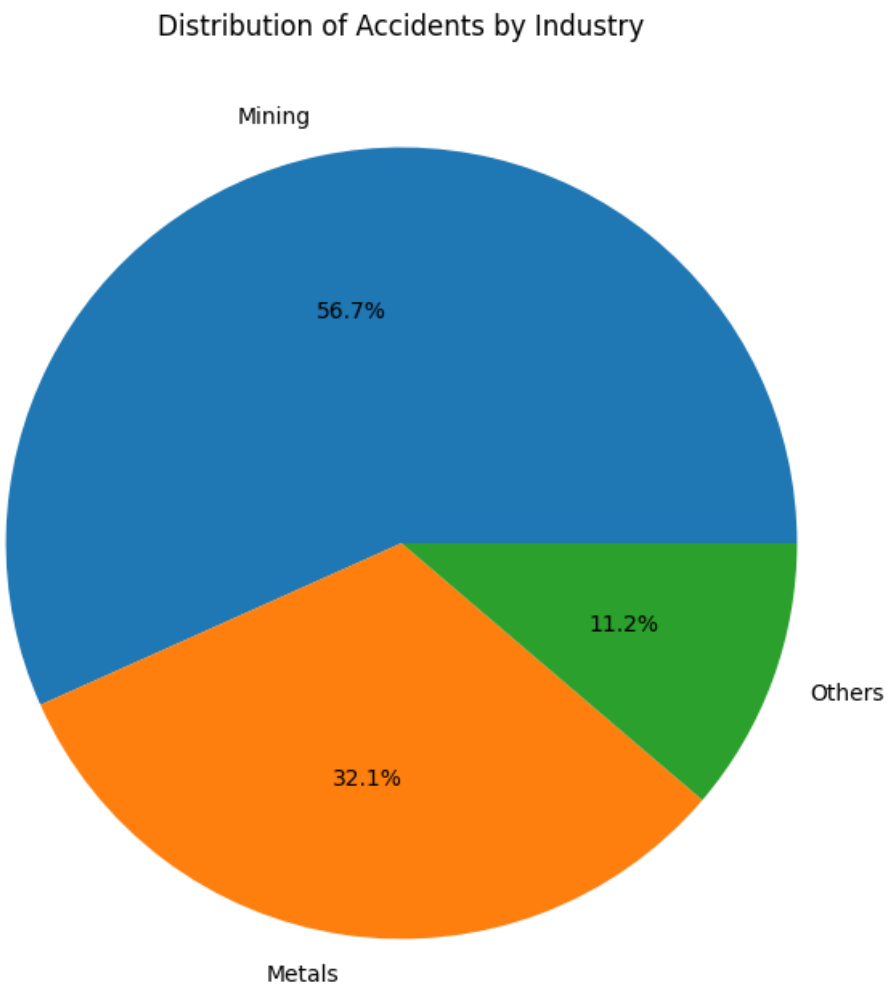

Figure 5: Distribution of Accidents by Industry

2.2.1.4 Accident Distribution by City: Local_03 had the highest number of accident reports, with various other locations contributing to the total count.

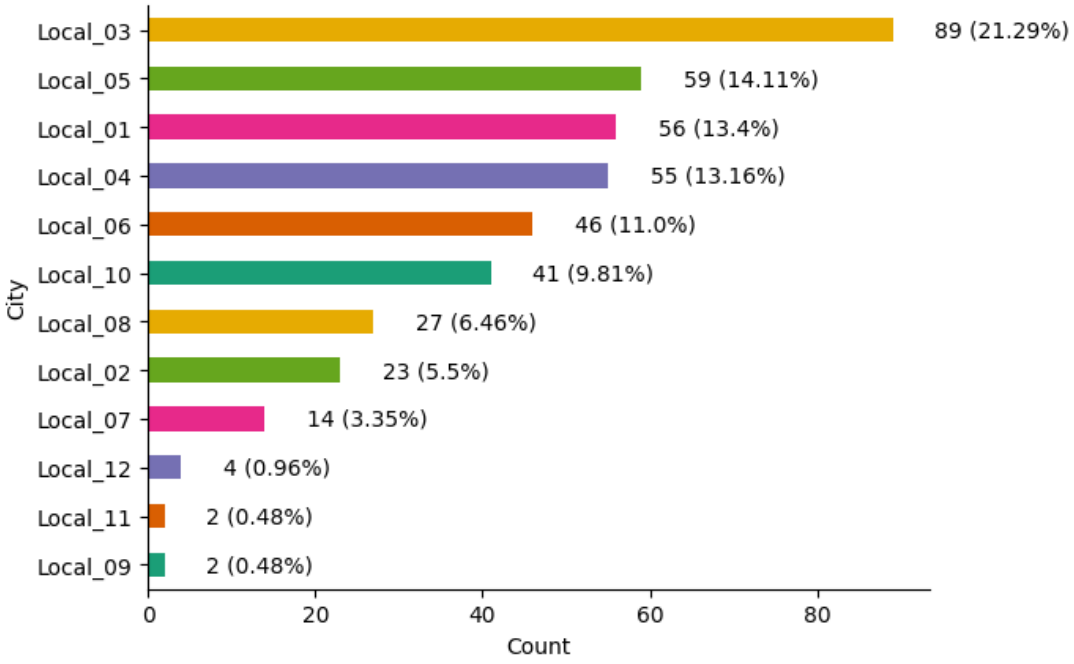

Figure 6: Distribution of Accidents by City

2.2.1.5 Employee Type Distribution: The dataset shows that 43% of the reports are from employees, with no reports from third-party or remote third-party workers.

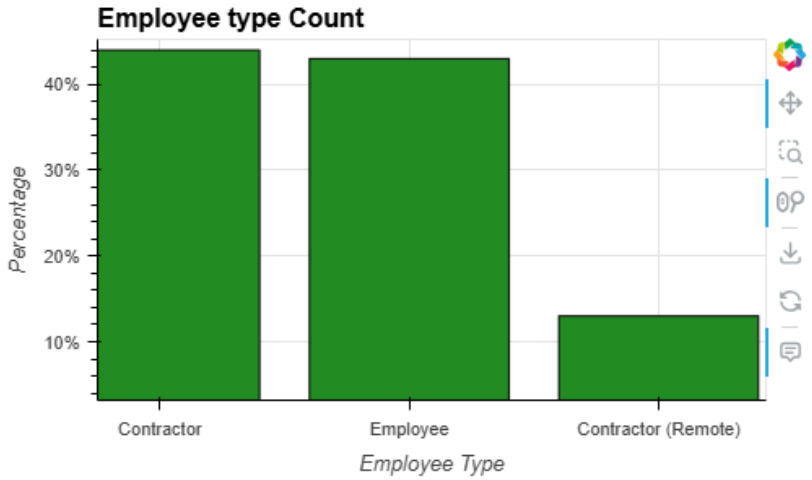

Figure 7: Distribution of Employee Type

2.2.1.6 Critical Risk Distribution: The analysis of critical risks revealed various categories, with a significant proportion falling under "Others," highlighting the diversity of risks involved.

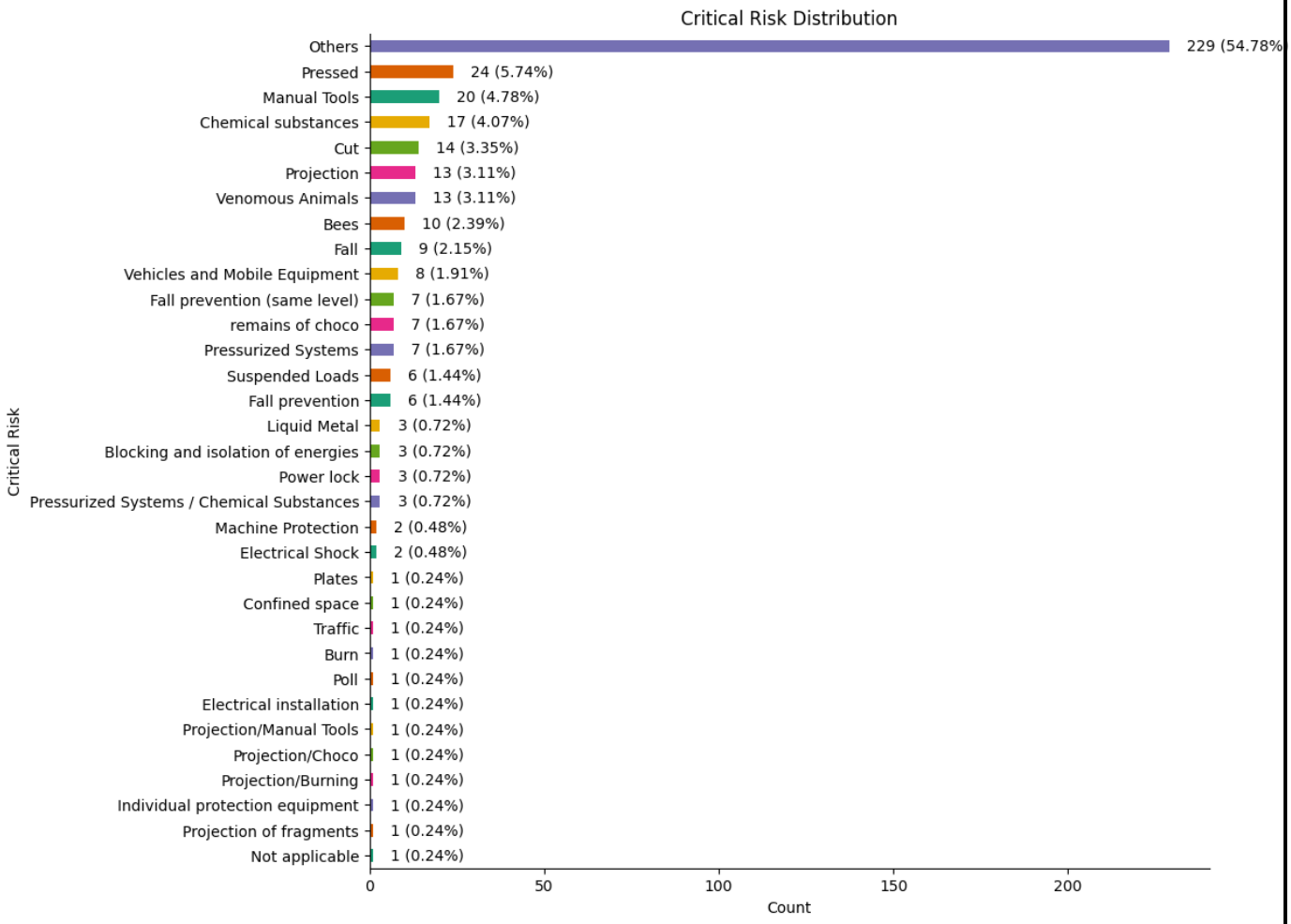

Figure 8: Distribution of Critical Risk

2.2.2 Bivariate Analysis

2.2.2.1 Accident Level & Potential Accident Level

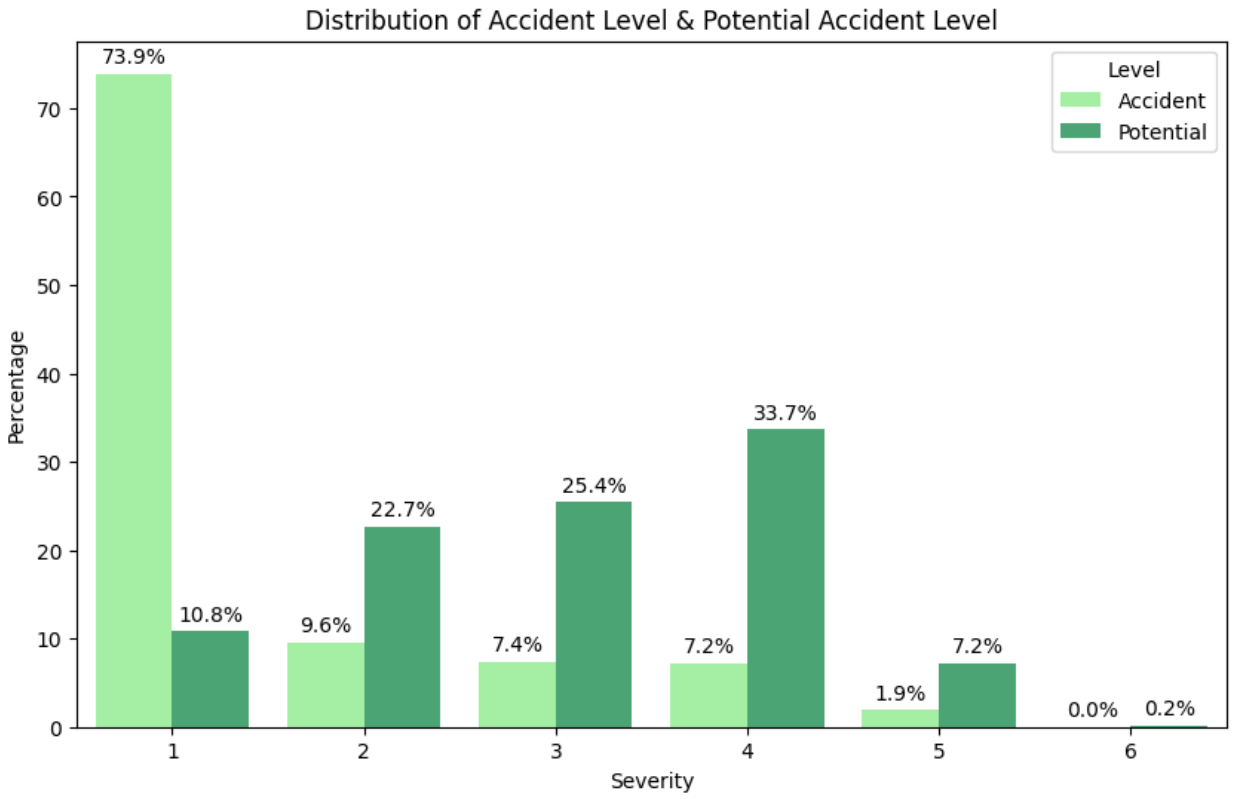

Figure 9: Distribution of Accident level and Potential Accident level

The analysis of accident levels revealed that the majority of incidents were categorized as **Accident Level 1**, accounting for **74%** of the total cases. Levels 2, 3, and 4 had significantly lower proportions, contributing **10%, 7%, and 7%**, respectively. Accident Levels 5 and 6 were rare, making up **2%** and **0%**, respectively.

In contrast, the potential accident levels showed a broader distribution. **Potential Accident Level 4** was the most frequent, comprising **34%** of cases, followed by Levels 3 and 2 at **25%** and **23%**, respectively. Levels 1, 5, and 6 were less common, representing **11%, 7%, and 0%**, respectively.

This distribution highlights a disparity between actual and potential accident severity, emphasizing the need for preventive measures to mitigate risks.

2.2.2.2 Accident Level and Potential Accident Level vs Gender

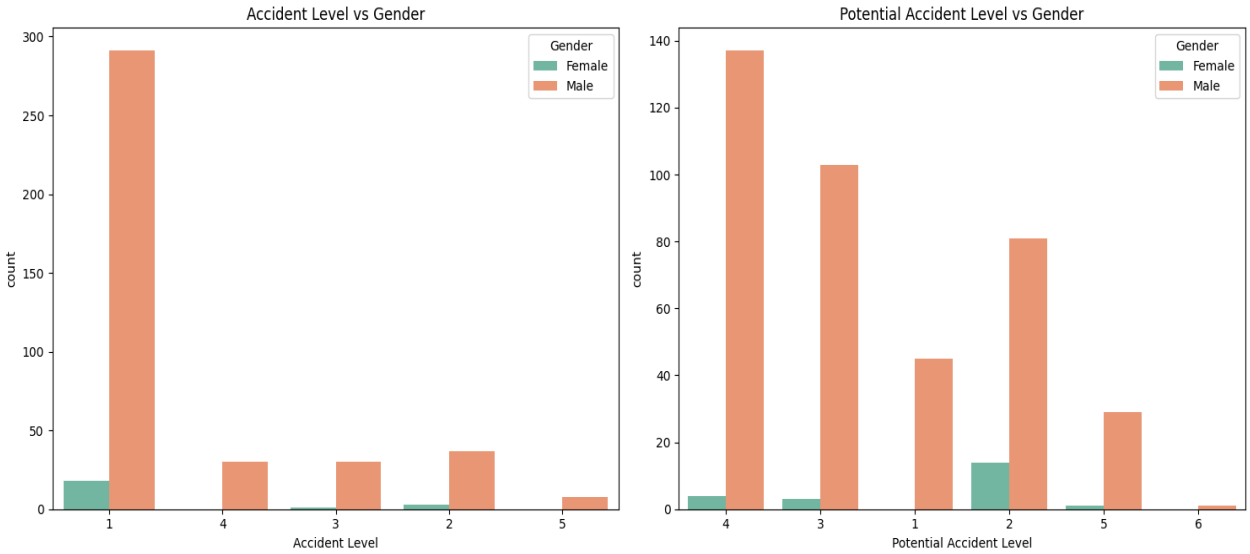

Figure 10: Distribution of Accident Level and Potential Accident Levels Vs Gender

2.2.2.2.1 Accident Level vs. Gender

Males were found to be significantly more involved in accidents across all levels, with a particularly notable disparity in lower accident levels (I and II).

2.2.2.2.2 Potential Accident Level vs. Gender

Similarly, males exhibited a higher likelihood of involvement in potential accidents. However, the gender difference in potential accident levels was less marked compared to actual accidents, indicating that preventive measures may be more effective in addressing risks among males.

2.2.2.3 Employee Type Vs Accident Level Distribution

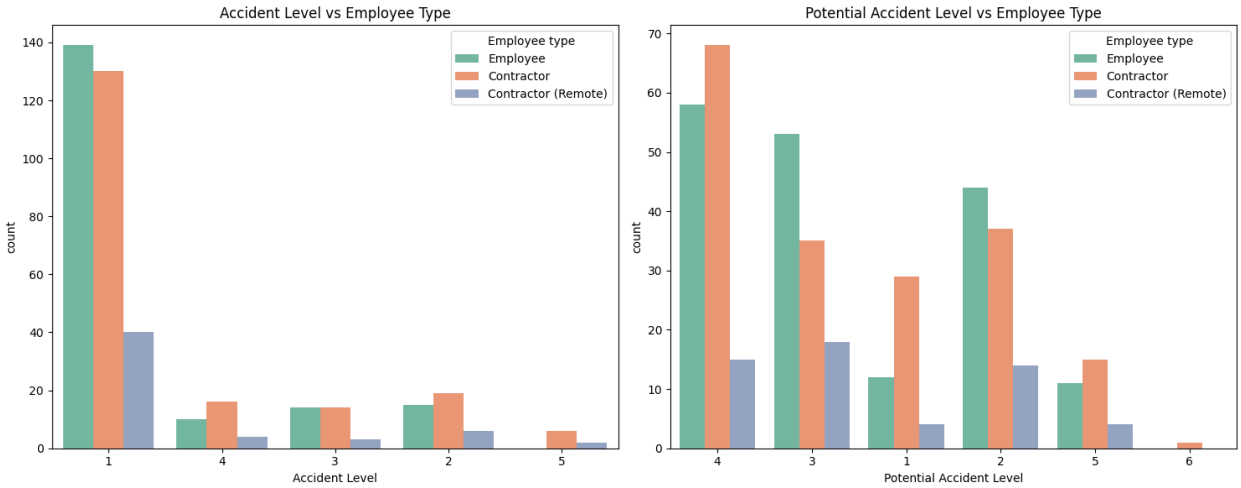

Figure 11: Distribution of Employee Type Vs Accident Level

2.2.2.3.1 Accident Level vs. Employee Type

Employees account for a significantly higher proportion of accidents across all levels compared to third parties, with a particularly notable disparity in lower accident levels (I and II).

2.2.2.3.2 Potential Accident Level vs. Employee Type

Employees are also more likely to be involved in potential accidents than third parties. However, the difference in potential accident levels between employee types is less pronounced compared to actual accidents, suggesting that preventive measures may be more impactful for employees.

2.2.2.4 Distribution of Accidents by Year and Month

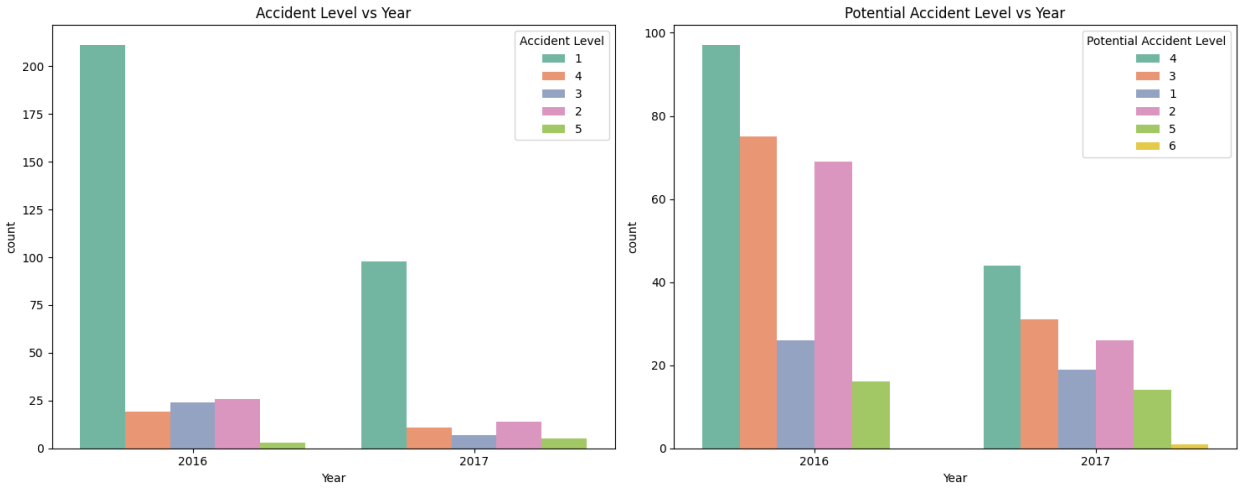

Figure 12: Distribution of Accidents by Year

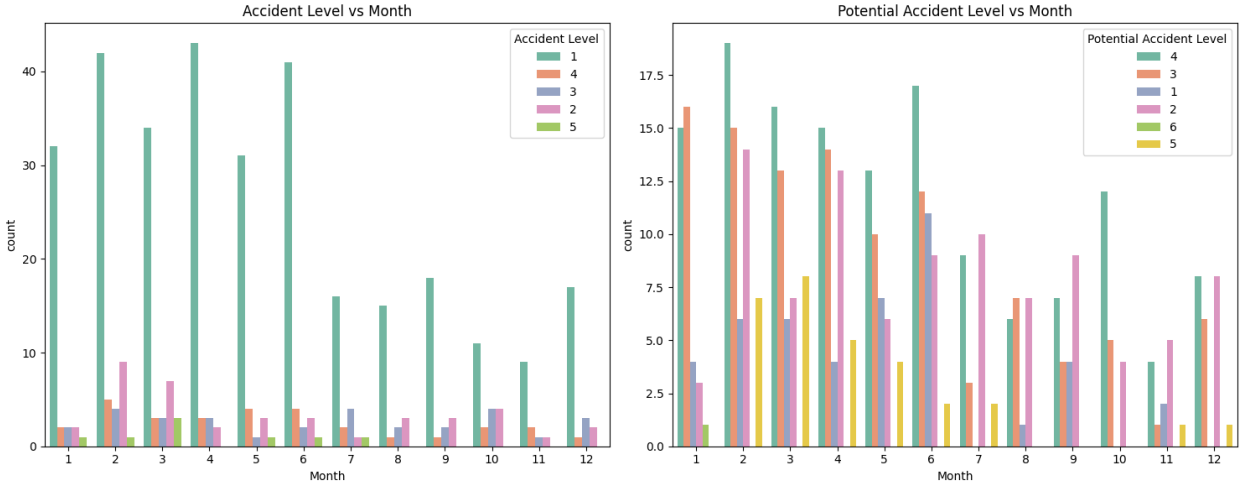

Figure 13: Distribution of Accidents by Month

2.2.2.4.1 Accident Level vs. Year

A declining trend in accident numbers across all levels is evident in recent years compared to earlier ones, reflecting potential improvements in safety measures over time.

2.2.2.4.2 Potential Accident Level vs. Year

Potential accidents also show a decreasing trend over the years, suggesting enhanced effectiveness of preventive measures and safety protocols in mitigating risks.

2.2.2.4.3 Accident Level vs. Month

Accident counts exhibit monthly variations, but no distinct seasonal pattern is observed. Further investigation may be required to understand the factors influencing these fluctuations.

2.2.2.4.4 Potential Accident Level vs. Month

Similar to actual accident levels, potential accidents display monthly variations without a clear seasonal trend, implying that accident occurrences may not be strongly linked to specific months.

2.2.2.5 Monthly Frequency of Accidents over Years

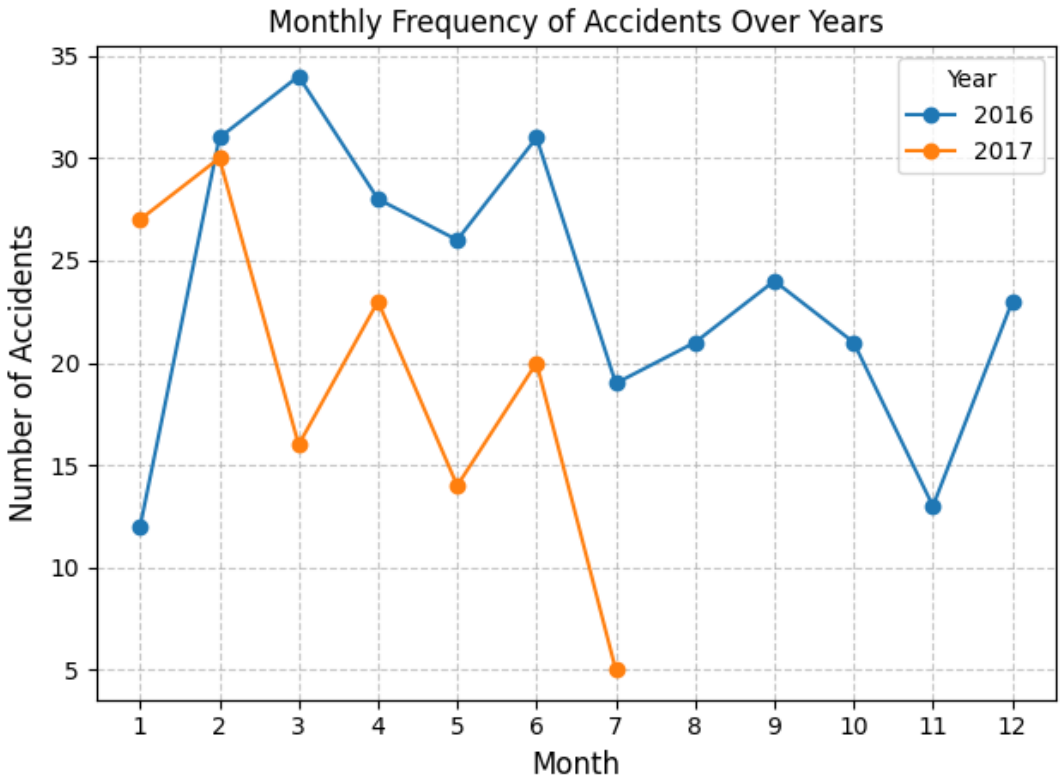

Figure 14: Monthly Frequency of Accidents over Years

2.2.2.5.1 Overall Trend

There is a general decline in the number of accidents over the years, suggesting that safety measures and interventions implemented over time are having a positive impact.

2.2.2.5.2 Seasonal Variations

Some seasonal variations in accident frequency are observed, with a slight increase noted around the middle of the year (months 5–7) in certain years. This may be influenced by factors such as weather conditions, workload, or specific activities during these months.

2.2.2.5.3 Year-to-Year Fluctuations

Despite the overall downward trend, year-to-year fluctuations in accident counts persist, emphasizing the importance of continuous monitoring and adjustments to safety protocols to address emerging challenges.

2.2.2.5.4 Further Analysis

Analyzing the specific causes of accidents across different months and years could provide valuable insights into underlying patterns or contributing factors, enabling more targeted safety improvements.

2.2.2.6 Accident Levels Vs Year, Month, Day, Weekday, WeekofYear

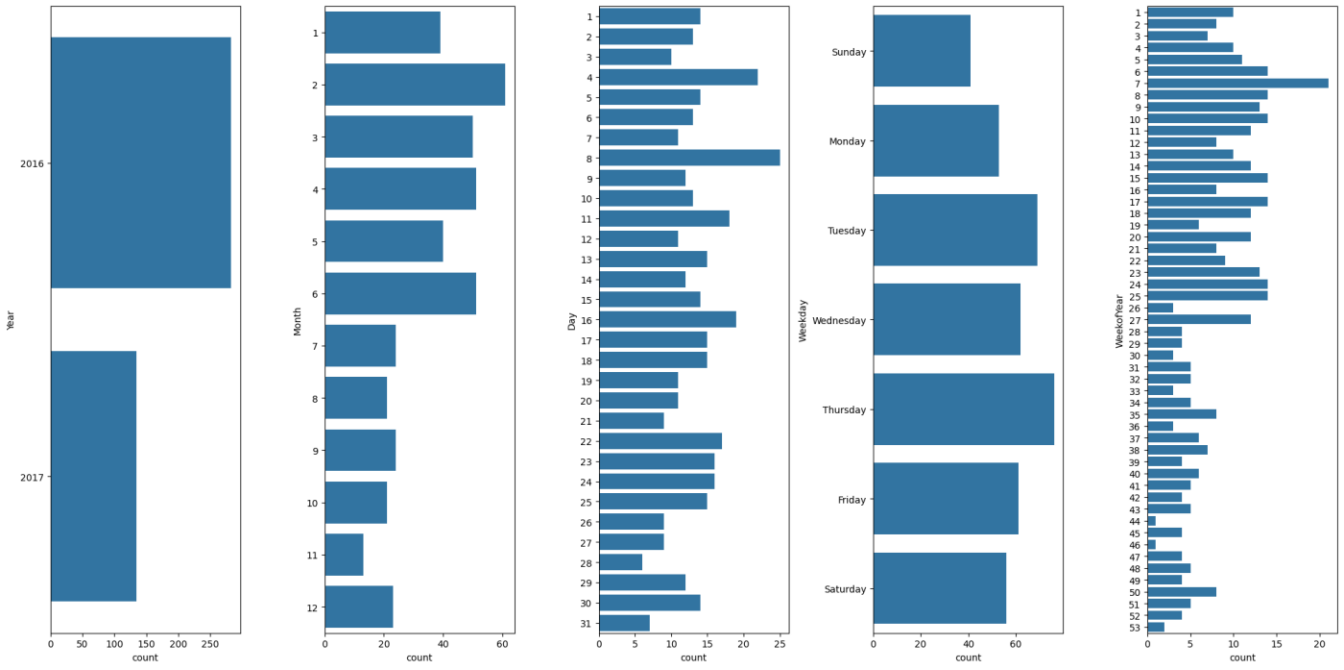

Figure 15: Accident Level Vs Year, Month, Day, Weekday, WeekofYear

2.2.2.6.1 Accident Level vs. Year

- Accidents were more frequent in **2016** compared to **2017**, indicating a declining trend in accident occurrence over the years.

2.2.2.6.2 Accident Level vs. Month

- Monthly variations are evident, with higher counts observed in the earlier months (**January to June**) compared to the latter half of the year.
- Peaks are noticeable around specific months, such as **June**.

2.2.2.6.3 Accident Level vs. Day

- Accidents occur relatively evenly across days, but there are slight increases on certain days, such as the **16th** and **25th**, which might warrant further investigation.

2.2.2.6.4 Accident Level vs. Weekday

- Accidents are more frequent on **Tuesdays, Wednesdays, and Thursdays**, with a relatively lower frequency on **Sundays**.
- This pattern suggests a higher likelihood of incidents during the workweek.

2.2.2.6.5 Accident Level vs. Week of Year

- Weekly trends show considerable variation, with noticeable peaks around **weeks 7, 25, and 35**, which could be linked to specific events or seasonal factors.

2.2.2.7 Date vs Potential Accident Level count

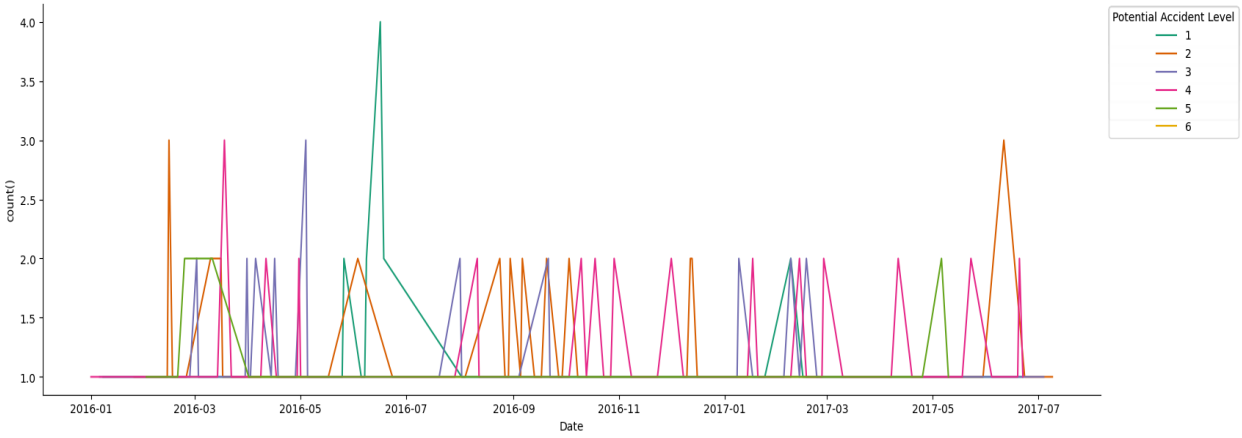

Figure 16: Date vs Potential Accident Level count

2.2.2.7.1 Trend Over Time

The number of accidents for each potential accident level shows fluctuations over time without a clear long-term increasing or decreasing trend. This suggests the influence of seasonality or other dynamic factors on accident occurrences.

2.2.2.7.2 Potential Accident Level IV

Accidents with high potential severity (Level IV) consistently occur less frequently than other levels, indicating a lower likelihood of such severe incidents.

2.2.2.7.3 Fluctuations and Peaks

All potential accident levels exhibit noticeable fluctuations, with certain periods showing peaks in accident counts. These variations may be linked to specific events, seasonal changes, or other external influences.

2.2.2.7.4 Lack of a Clear Pattern

No consistent relationship is observed between the date and accident frequency across potential accident levels, implying that multiple interacting factors likely contribute to the occurrence of accidents.

2.2.2.8 Date vs Accident Level count

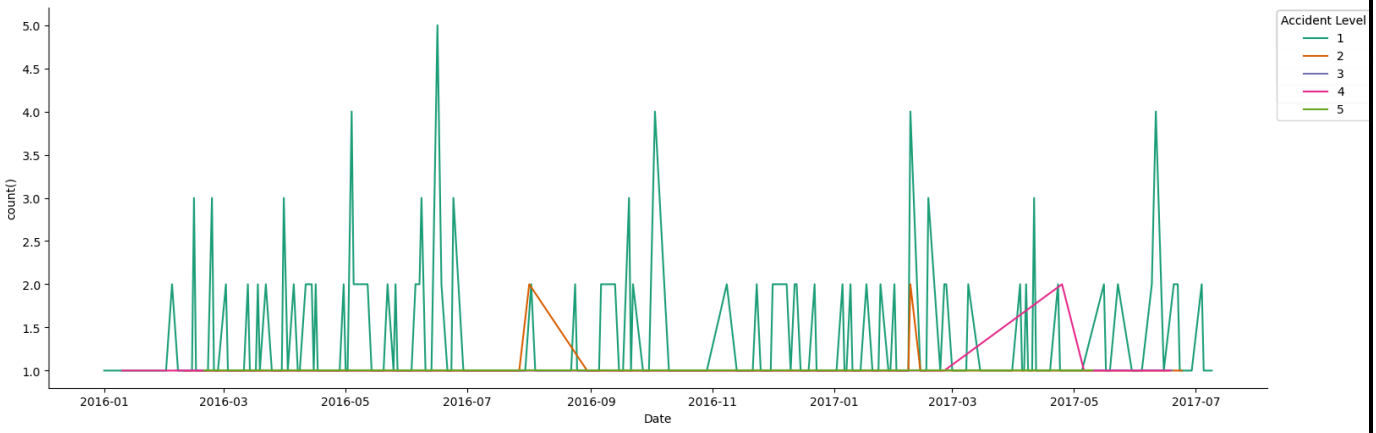

Figure 17: Date vs Accident Level count

2.2.2.8.1 Trend over Time

Accident counts show fluctuations over time without a clear long-term upward or downward trend. This indicates potential seasonality or other dynamic factors influencing accident occurrences.

2.2.2.8.2 Accident Levels I and II

Minor accidents (Levels I and II) consistently have higher counts compared to other levels, highlighting their greater frequency.

2.2.2.8.3 Fluctuations and Peaks

All accident levels exhibit noticeable fluctuations, with certain periods experiencing peaks in occurrence. These peaks may be linked to seasonal changes, specific events, or external factors.

2.2.2.8.4 Lack of a Clear Pattern

No consistent relationship is observed between accident frequency and date, suggesting that multiple complex factors likely contribute to the occurrence of accidents.

2.2.2.9 Frequency of Accident level Vs Month

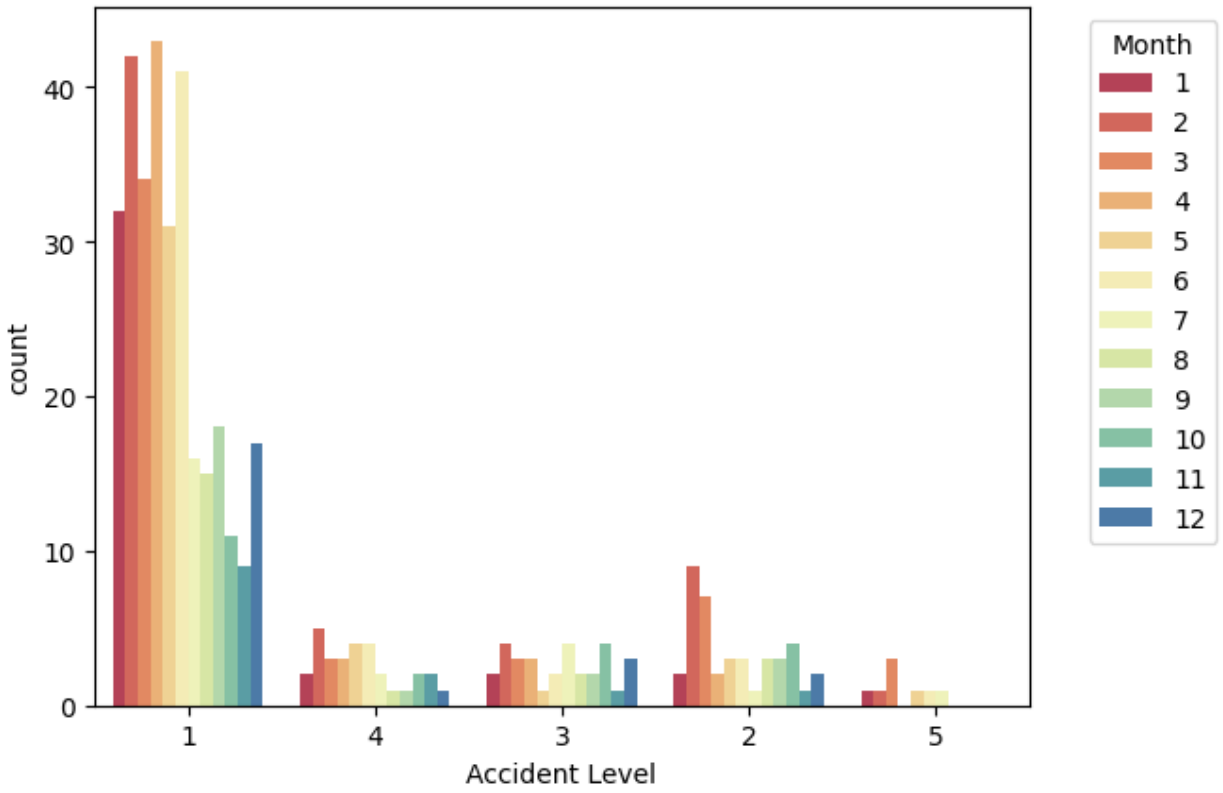

Figure 18: Accident Level Vs Month

2.2.2.9.1 High Frequency in Accident Level 1:

Accident Level 1 shows the highest frequency of occurrences across all months, indicating that minor accidents are significantly more common compared to higher severity levels.

2.2.2.9.2 Relatively Even Distribution Across Months:

While there are variations in counts, accidents for all levels appear relatively distributed throughout the year without a clear seasonal pattern.

2.2.2.9.3 Lower Frequency for Higher Accident Levels:

Accident Levels 4 and 5 have consistently lower counts compared to Levels 1 and 2, suggesting that severe accidents are less frequent regardless of the month.

2.2.2.9.4 Potential Peaks in Specific Months:

Levels 1 and 2 show slightly higher counts in earlier months (e.g., January to March), which could indicate seasonal or operational factors influencing accident occurrences during this period.

2.2.2.10 Date vs Industry Sector count

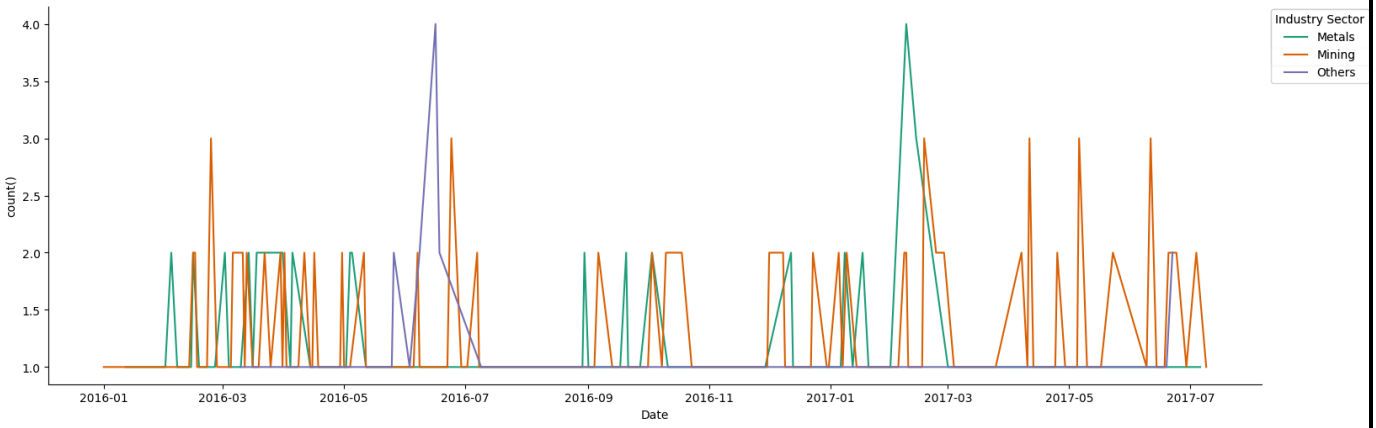

Figure 19: Date vs Industry Sector count

2.2.2.10.1 Mining Sector:

The Mining sector consistently records a higher number of accidents compared to other industries over the observed period, indicating a greater risk of accidents within this sector.

2.2.2.10.2 Fluctuations and Peaks:

All sectors show fluctuations in accident numbers over time, with some periods experiencing peaks. These peaks may reflect seasonal changes or external factors impacting accident rates.

2.2.2.10.3 Other Sectors:

Sectors such as Metals, Others, and Chemicals report lower but still notable accident numbers. The fluctuations in these sectors suggest that external influences also affect accident occurrences.

2.2.2.10.4 No Clear Trend:

No consistent long-term trend of increasing or decreasing accidents is observed in any sector, suggesting that accidents are influenced by a variety of interacting factors.

2.2.2.10.5 Importance of Sector-Specific Analysis:

The data emphasizes the need for sector-specific accident trend analysis to better understand the contributing factors and implement targeted safety interventions.

2.2.2.11 Date vs Country count

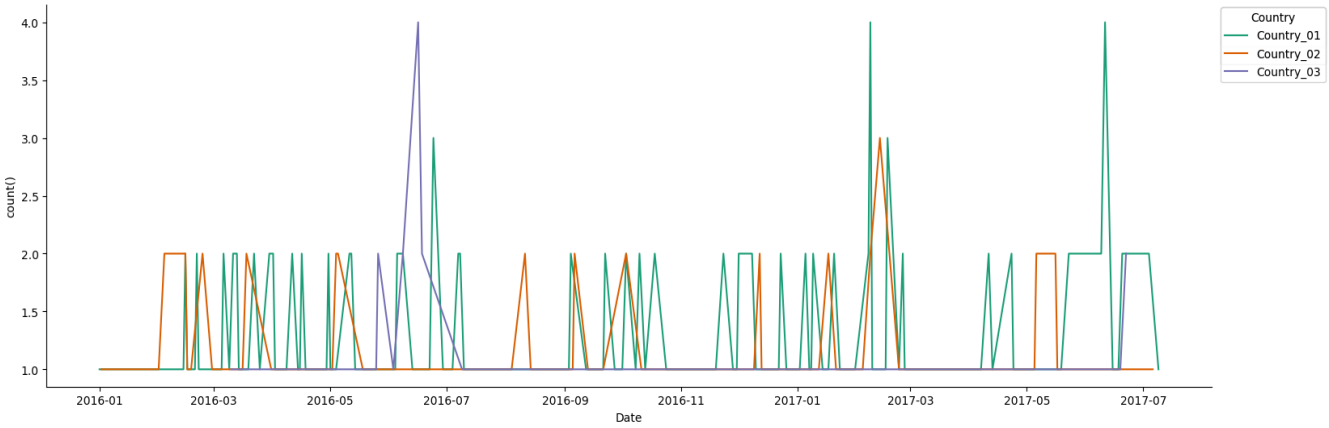

Figure 20: Date vs Country count

2.2.2.11.1 Country_01:

Country_01 consistently reports the highest number of accidents throughout the observed period, indicating a higher overall accident rate compared to the other two countries.

2.2.2.11.2 Fluctuations and Peaks:

All countries show fluctuations in accident numbers over time, with some periods experiencing peaks. These peaks may reflect seasonal variations, specific events, or other external factors impacting accident rates.

2.2.2.11.3 Country_02 and Country_03:

Both Country_02 and Country_03 generally report fewer accidents than Country_01. However, they too experience fluctuations and occasional peaks in accident occurrences.

2.2.2.11.4 No Clear Trend:

There is no consistent long-term trend of increasing or decreasing accidents in any of the countries, suggesting that accident occurrences are influenced by a variety of interacting factors.

2.2.2.11.5 Country-Specific Factors:

The data underscores the importance of considering country-specific factors when analyzing accident trends. These factors may include variations in safety regulations, industry practices, cultural attitudes towards safety, and other socio-economic influences.

2.2.2.12 Accident Level vs Potential Accident Level

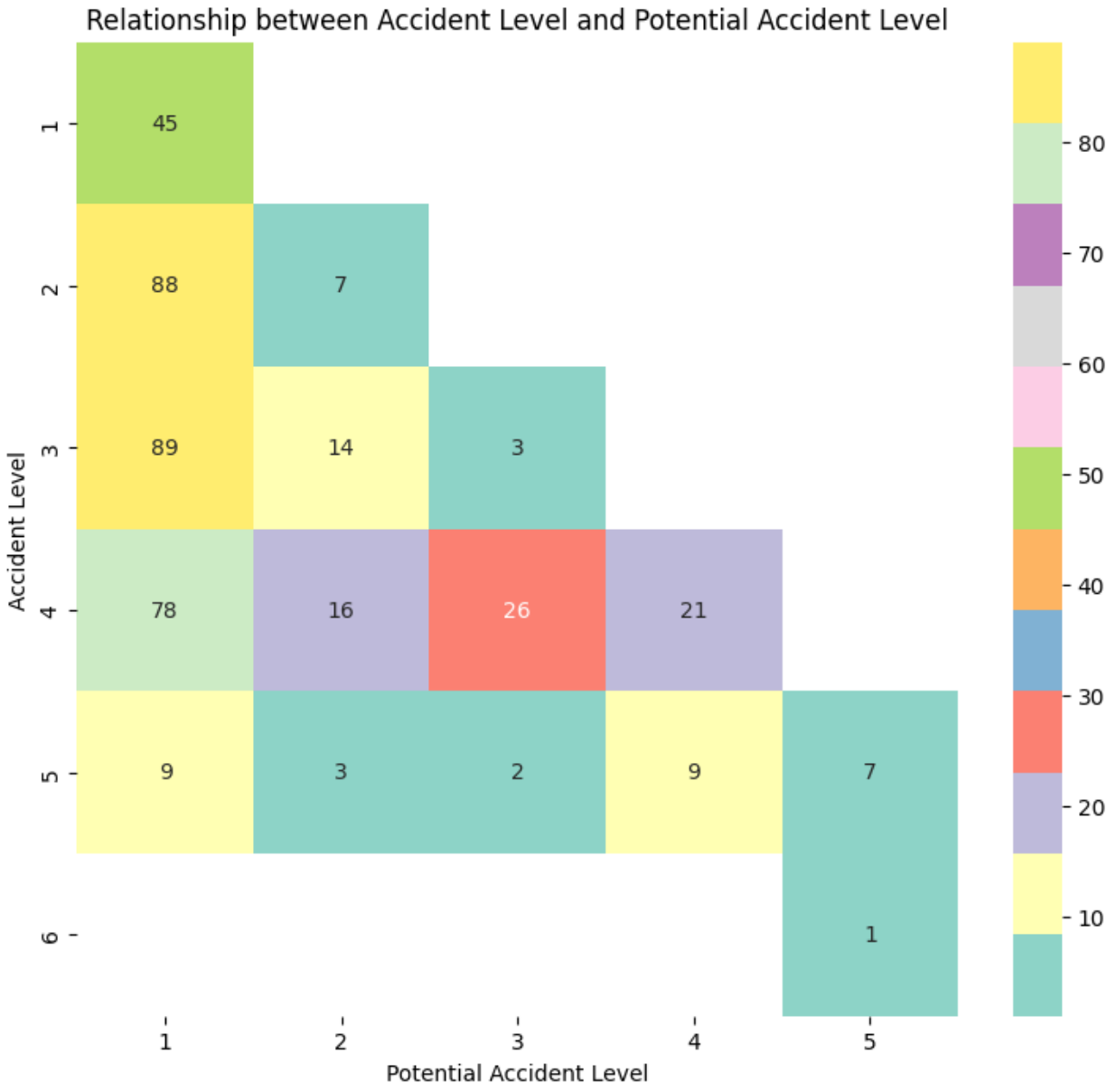

Figure 21: Heatmap of Accident level Vs Potential Accident Level

2.2.2.12.1 Diagonal Dominance:

The heatmap reveals strong diagonal dominance, indicating a positive correlation between Accident Level and Potential Accident Level. This suggests that accidents with a higher actual severity tend to also have a higher potential severity.

2.2.2.12.2 Potential for Worse Outcomes:

Notable off-diagonal values, particularly those above the diagonal, indicate that many accidents with lower actual severity levels had the potential for much worse outcomes.

2.2.2.12.3 Preventive Measures:

The gap between actual and potential severity underscores the importance of preventive measures and safety protocols. These likely helped prevent many accidents from escalating to their full potential severity.

2.2.2.12.4 Focus Areas for Improvement:

The heatmap highlights areas where safety measures can be enhanced. Targeting accidents with high potential severity but lower actual severity may lead to more effective prevention strategies.

2.2.2.13 Industry Sector vs Accident Level

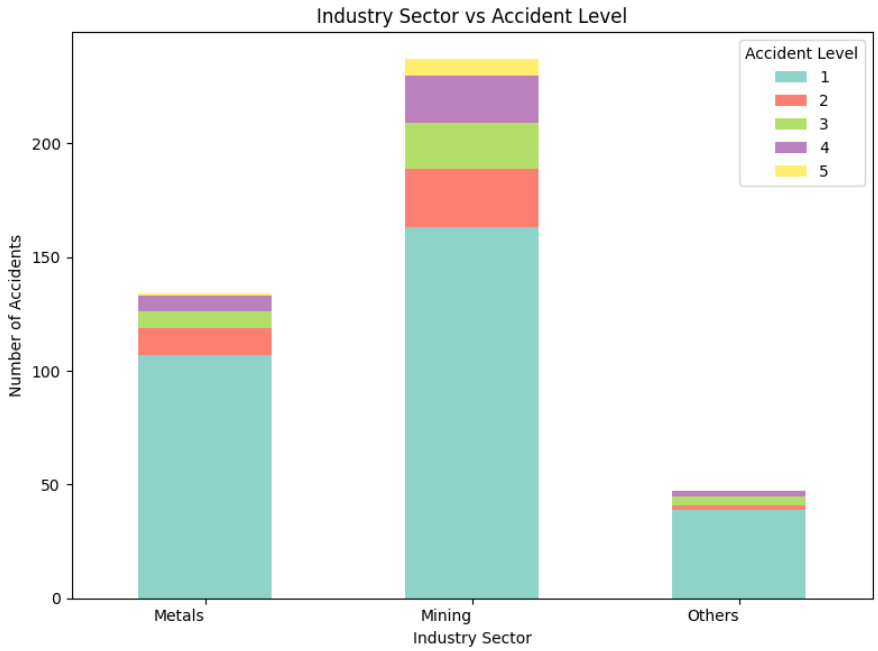

Figure 22: Industry Sector vs Accident Level

2.2.2.13.1 Mining Sector:

The Mining sector has the highest number of accidents across all severity levels, highlighting the significant risks to worker safety within this industry.

2.2.2.13.2 Other Sectors:

Sectors such as Metals, Others, and Chemicals also report a notable number of accidents, particularly at lower severity levels.

2.2.2.13.3 Severity Distribution:

Most accidents across all sectors are classified as Level I and Level II, indicating that they are generally minor. However, the occurrence of higher-level accidents (Levels III to VI) underscores the importance of safety measures, even in industries with predominantly minor incidents.

2.2.2.13.4 Focus Areas for Improvement:

The chart emphasizes the need for targeted safety interventions in the Mining sector and other high-risk industries. Efforts should focus on reducing accident numbers and preventing the escalation of minor incidents into more severe ones.

2.2.2.14 Distribution of Accident Levels Across Countries

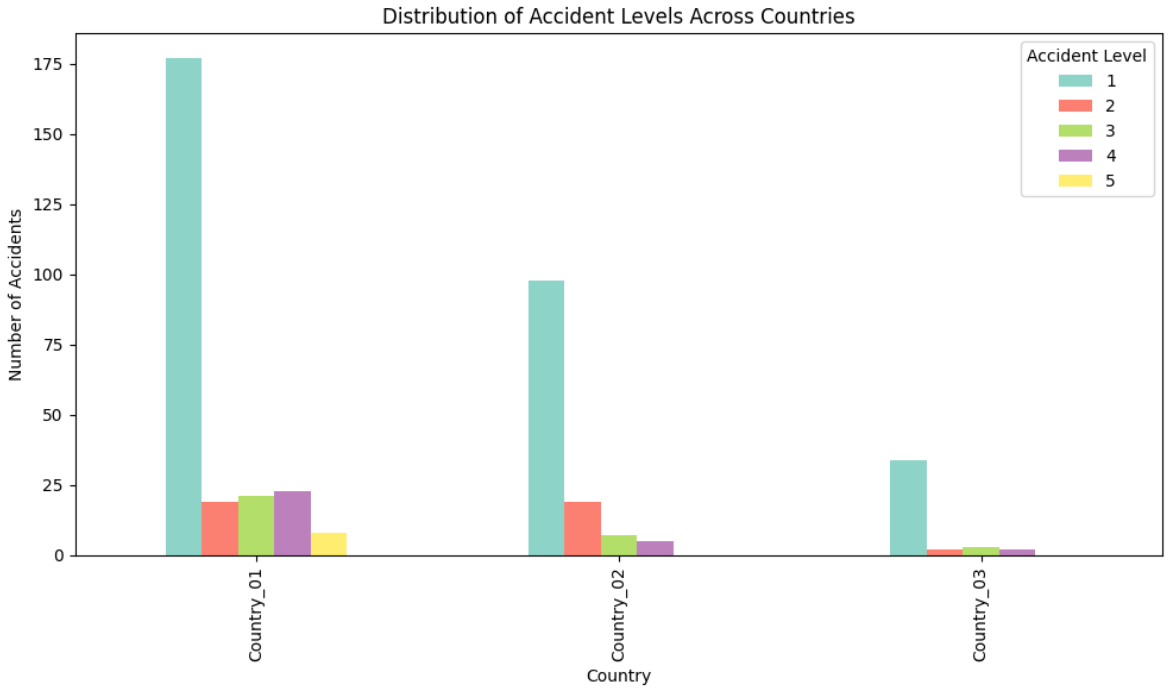

Figure 23: Distribution of Accident Levels Across Countries

2.2.2.14.1 Country_01:

Country_01 consistently reports the highest number of accidents across all levels (I to VI), indicating that there may be areas where safety measures can be improved compared to the other two countries.

2.2.2.14.2 Country_02:

Country_02 generally ranks second in the number of accidents, with a notable increase in Level III incidents. This suggests that specific risks or practices in Country_02 may contribute to a higher frequency of more severe accidents.

2.2.2.14.3 Country_03:

Country_03 has the lowest number of accidents, particularly in the more severe categories (IV to VI). This could indicate that Country_03 has more effective safety protocols in place compared to the other countries.

Across all countries, accident numbers decrease as severity levels rise, which is typical since more severe accidents tend to be less frequent.

The variation in accident levels across countries suggests potential differences in safety regulations, industry practices, or country-specific risk factors.

2.2.2.15 Distribution of Accident Levels Across Cities

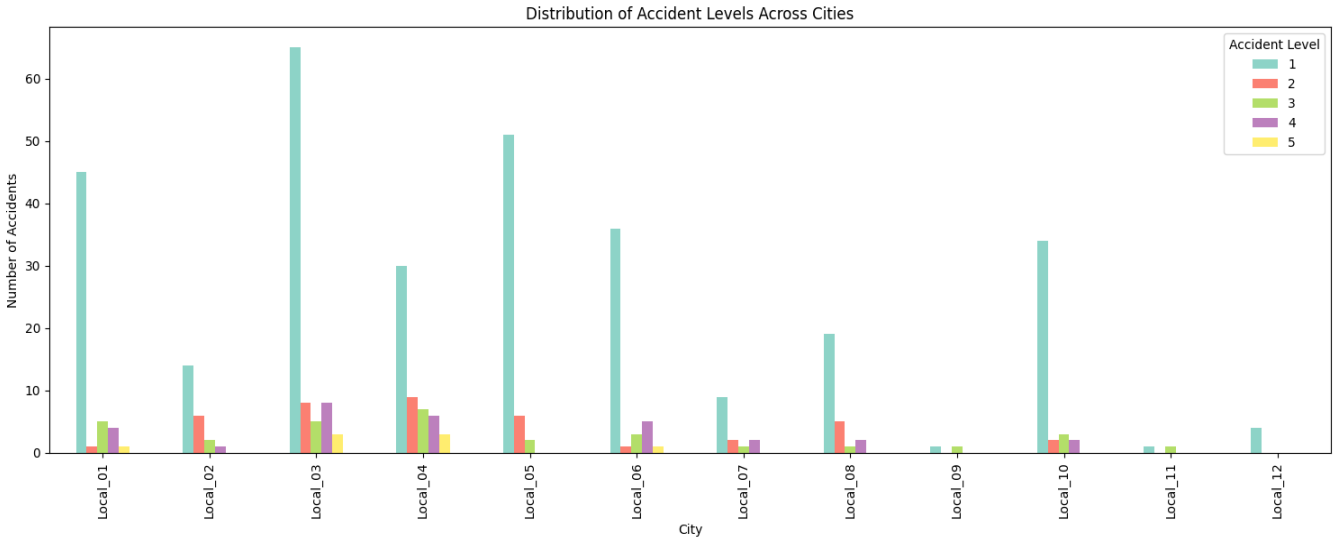

Figure 24: Distribution of Accident Levels Across Cities

2.2.2.15.1 Accident Distribution:

Accidents are unevenly distributed across cities, with some cities reporting significantly more accidents than others.

2.2.2.15.2 Severity Variation:

The distribution of accident severity (Levels I to VI) varies between cities. Some cities may have a higher proportion of severe accidents (Levels IV to VI), while others primarily experience minor accidents (Levels I and II).

2.2.2.15.3 City-Specific Patterns:

Each city displays a distinct pattern in accident level distribution, suggesting that the factors contributing to accidents may differ from one city to another.

2.2.2.15.4 Potential Focus Areas:

Cities with a higher frequency of accidents, particularly those with a larger proportion of severe accidents, should be prioritized for further investigation and targeted safety interventions.

2.2.2.16 Country vs Industry Sector

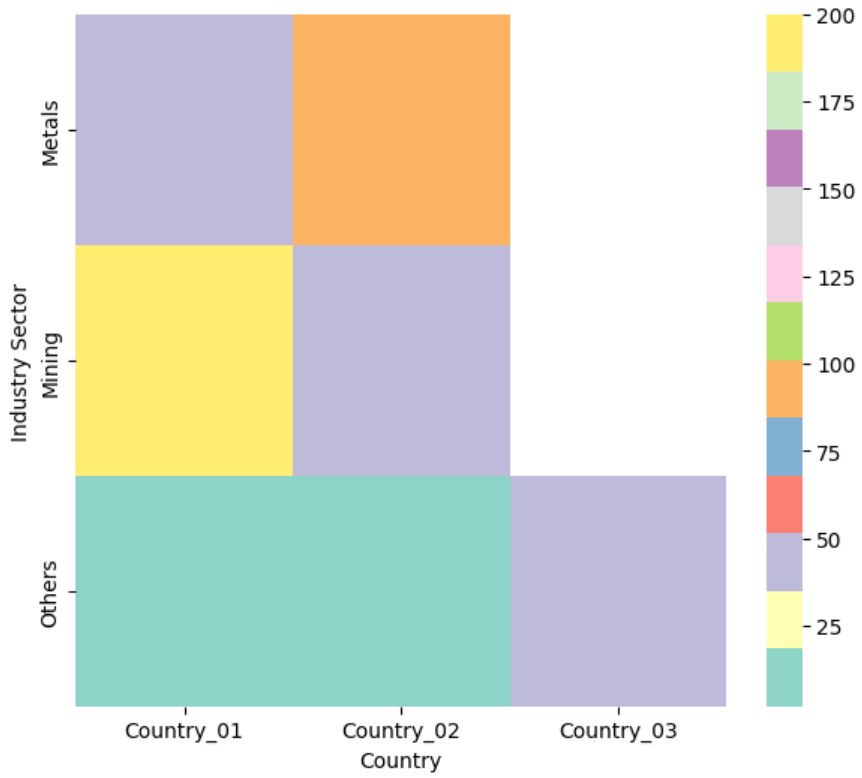

Figure 25: Country vs Industry Sector

2.2.2.16.1 Country_01:

Country_01 reports the highest number of accidents across all sectors, with Mining being the most accident-prone, followed by Metals. The Others sector experiences relatively fewer accidents.

2.2.2.16.2 Country_02:

Country_02 displays a more balanced distribution of accidents across sectors compared to Country_01, though Mining and Metals still account for a significant number of incidents.

2.2.2.16.3 Country_03:

Country_03 has the lowest overall number of accidents. While Mining remains a concern, other sectors show comparatively fewer incidents.

2.2.2.16.4 Overall:

Mining is consistently identified as a high-risk industry across all three countries. Country_01 consistently reports more accidents than the other two countries. The variation in accident distribution across countries may reflect differences in safety practices or industry compositions.

2.2.2.17 Critical Risk vs Industry Sector

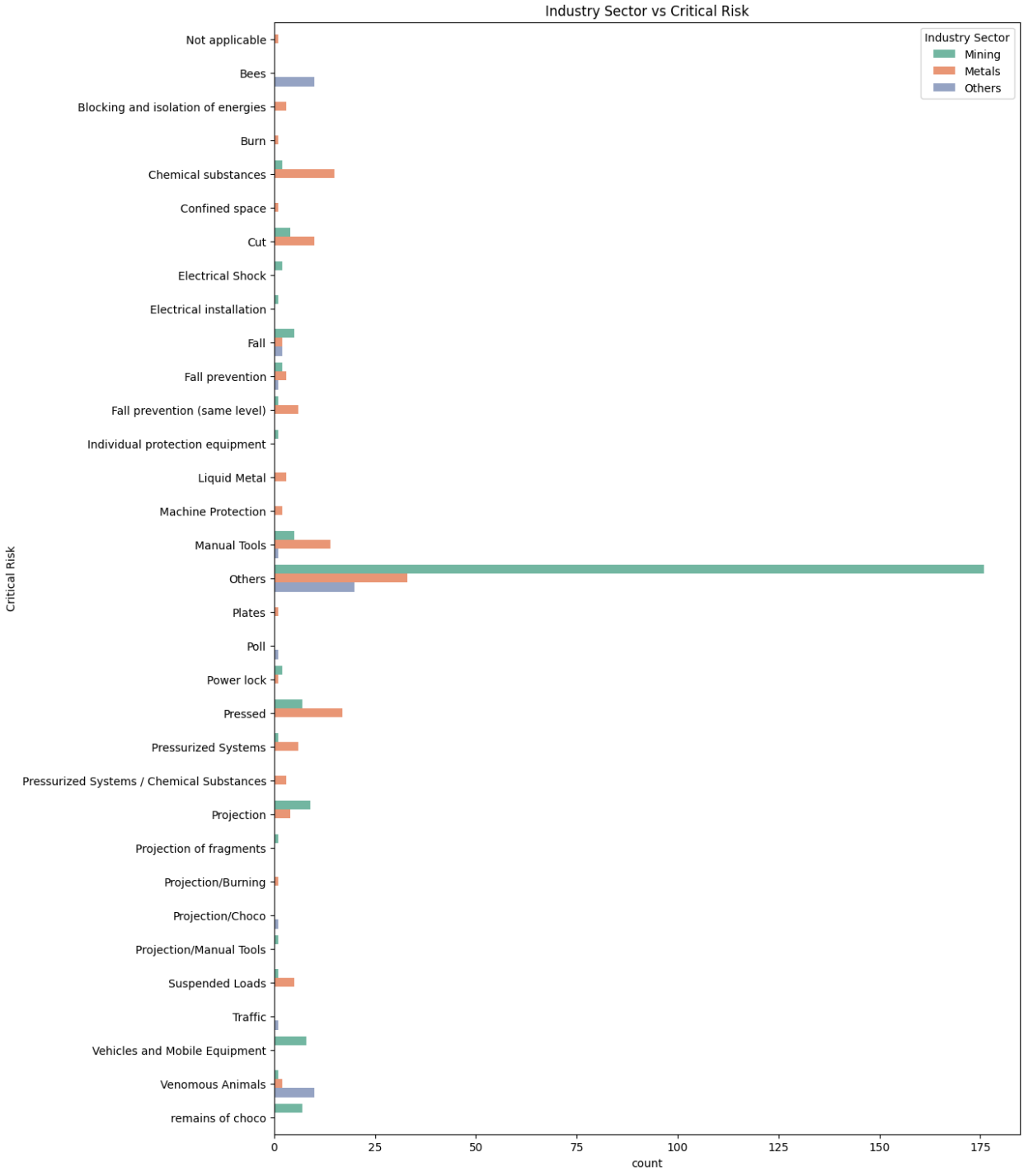

Figure 26: Critical Risk vs Industry Sector

2.2.2.17.1 Manual Tools Risk Dominance:

- The "Manual Tools" risk has an overwhelmingly high count compared to other risks, especially in the **Mining sector**. It stands out as the most significant critical risk in this data.

2.2.2.17.2 Industry Sector Distribution:

- Risks are categorized into three sectors: **Mining, Metals**, and **Others**.
- The Mining sector shows a significantly higher count in many categories, indicating it experiences more frequent critical risks overall.

2.2.2.17.3 Secondary Risks:

- Risks like "Others" and "Chemical Substances" also have notable counts, primarily in the **Metals** sector.
- Risks such as "Confined Space," "Electrical Shock," and "Projection" have minimal representation, suggesting they are less common or less critical across sectors.

2.2.2.17.4 Sector-Specific Risks:

- Certain risks, such as "Venomous Animals" and "Bees," appear infrequently but are present, likely tied to specific environmental or operational conditions in the respective sectors.

2.2.2.17.5 Imbalance Across Sectors:

- The **Metals** and **Others** sectors have fewer risk categories with high counts compared to Mining, which dominates in both diversity and frequency of risks.

2.2.2.18 Critical Risk vs Employee Type

Figure 27: Critical Risk vs Employee Type

2.2.2.18.1 Environmental Risk:

This is the most commonly identified critical risk across all employee categories, indicating that environmental impact is a concern for everyone involved in the incident.

2.2.2.18.2 Health and Safety Risk:

Ranked second in criticality, this risk is particularly significant for Employees and Third Parties, underscoring the need to prioritize the safety of both internal and external personnel.

2.2.2.18.3 Process Safety Risk:

More prevalent among Employees, this highlights that those directly engaged in operational processes face greater exposure to this type of risk.

2.2.2.18.4 Other Risks:

Although less frequent, other critical risks such as Asset Integrity and Security remain present across various employee groups.

2.2.2.18.5 Employee Type and Risk Correlation:

The distribution of risks varies slightly across employee roles, suggesting that different responsibilities influence the types of risks encountered.

2.2.2.18.6 Focus Areas for Improvement:

The analysis stresses the importance of implementing risk management strategies tailored to the unique risks faced by different employee groups. This could include offering thorough safety training for all employees, enforcing stringent safety protocols for third-party workers, and reinforcing process safety measures for those directly involved in operations.

2.2.2.19 Season vs Accident Levels, Potential Accident Levels

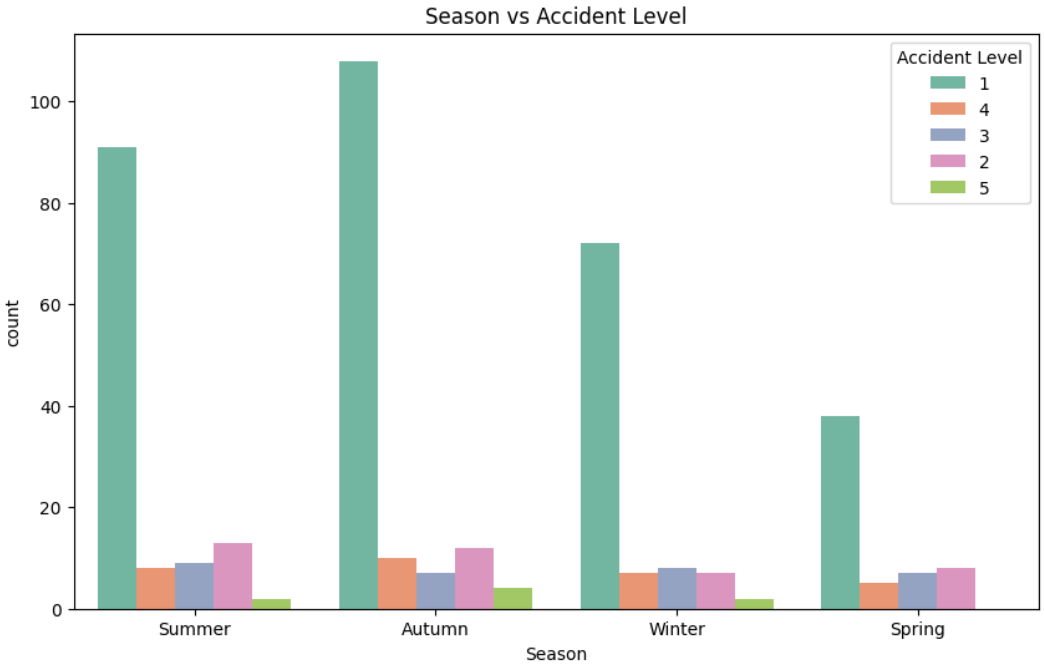

Figure 28: Season Vs Accident Level

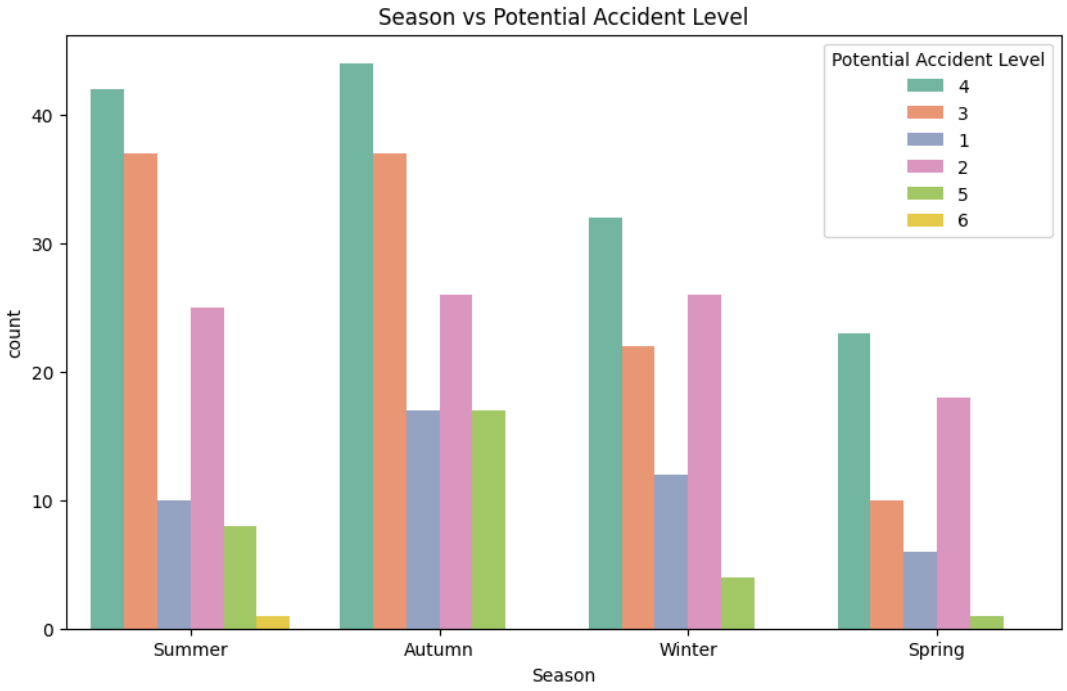

Figure 29: Season vs Potential Accident Level

2.2.2.19.1 Season vs. Accident Level:

Accidents are generally distributed evenly across the seasons, with a slight increase in Autumn. This suggests that seasonal factors may not significantly impact the overall occurrence of accidents. However, it would be worthwhile to explore whether certain types of accidents are more common in specific seasons.

2.2.2.19.2 Season vs. Potential Accident Level:

As with the previous analysis, the distribution of potential accident levels remains consistent across seasons. This indicates that the potential severity of accidents is not heavily influenced by seasonal factors.

2.2.2.20 Potential Accident Level vs Weekend

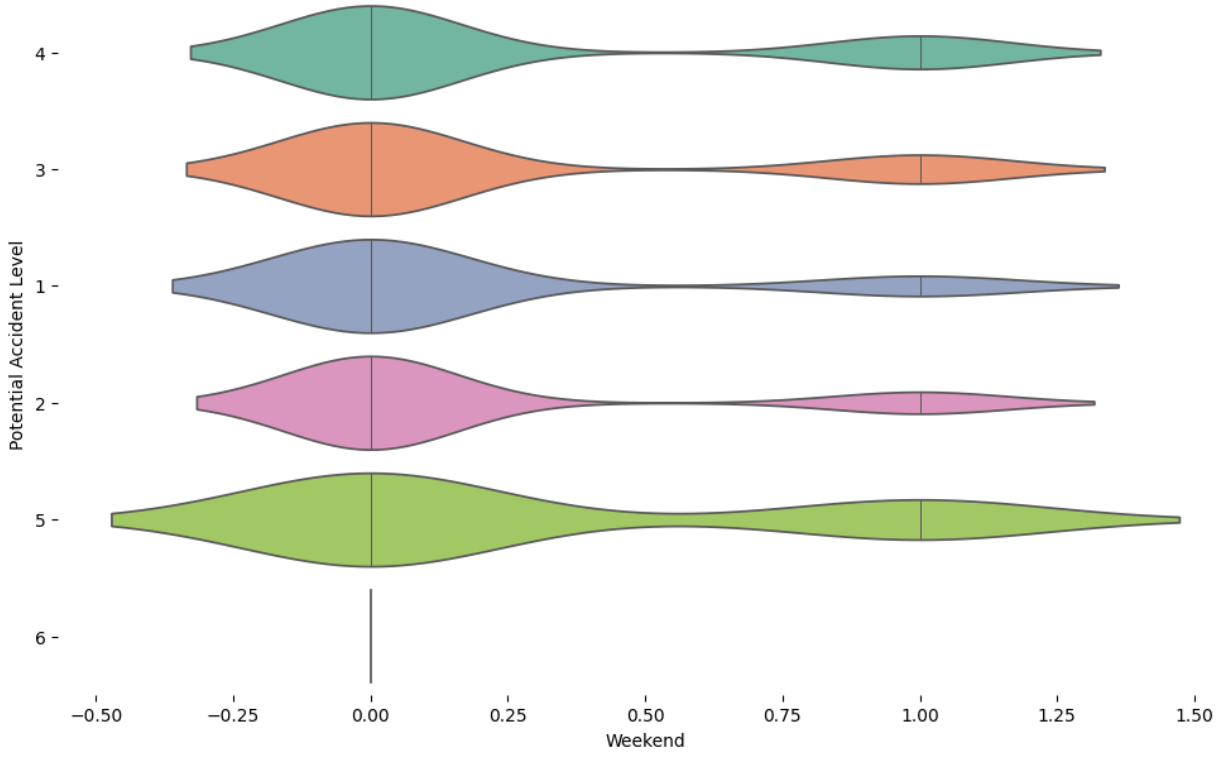

Figure 30: Potential Accident Level vs Weekend

2.2.2.20.1 Weekends vs. Weekdays:

The distribution of potential accident levels is fairly consistent between weekends and weekdays. There is no strong evidence to suggest that weekends have a notably higher or lower likelihood of accidents with a specific severity level compared to weekdays.

2.2.2.20.2 Potential Accident Level I:

This is the most common potential accident level for both weekends and weekdays, indicating that most incidents, regardless of the day, have a low potential for serious consequences.

2.2.2.20.3 Higher Potential Accident Levels:

Potential accident levels III to VI are less frequent but still occur on both weekends and weekdays. This suggests that more severe accidents can happen throughout the week, although they are generally less likely.

2.2.2.20.4 Further Analysis:

While the violin plot provides an overall view, additional statistical analysis may be necessary to determine if there are any significant differences in the distribution of potential accident levels between weekends and weekdays.

2.2.3 NLP Analysis

2.2.3.1 Data Preprocessing and Storage:

The preprocessed dataset (df_preprocess) was saved to a file for easy access and analysis. This ensures that the data is readily available for subsequent NLP tasks without repeating preprocessing steps.

2.2.3.2 Tokenization and Stopword Removal:

The tokenize function was created to clean text data by converting it to lowercase, extracting alphabetic words using regular expressions, and removing stopwords (downloaded using the NLTK library).

Descriptions from the dataset were tokenized, and a flat list of all words was generated. This helped in identifying frequently occurring terms across the data.

2.2.3.3 Unigram Analysis:

The *Counter* function was used to compute the frequency of individual words (unigrams) from the tokenized descriptions.

The most common words were displayed to gain insights into prominent terms that could guide categorization of accidents.

Top 5 unigram words and their frequency:

1. causing - 166
2. hand - 163
3. employee - 156
4. left - 155
5. right - 154

Figure 31: Unigram Wordcloud

2.2.3.4 Bi-gram Analysis:

A new function, *find_phrases*, was developed to extract two-word phrases (bi-grams) from the descriptions.

The frequency of these bi-grams was calculated and analyzed, identifying common pairs like "left hand" and "causing injury" to uncover patterns and contextual associations.

Top 5 bi-gram words and their frequency:

1. left hand - 70
2. right hand - 57
3. time accident - 56
4. causing injury - 51
5. finger left - 22

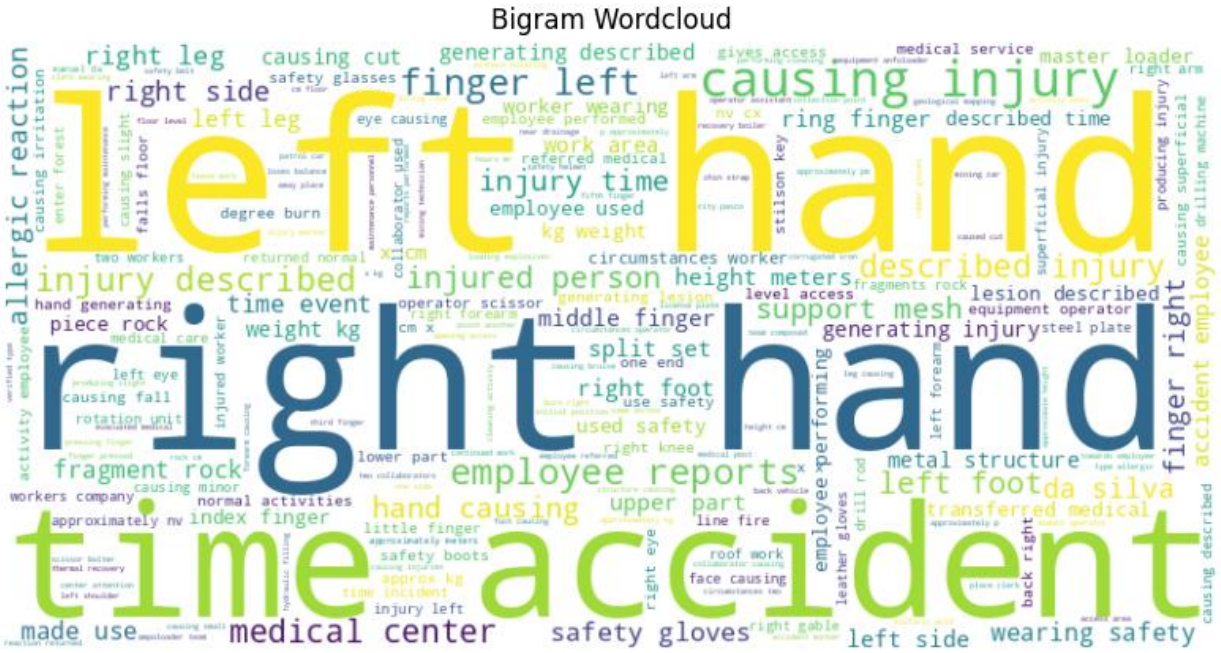

Figure 32: Bigram Wordcloud

2.2.3.4 Tri-gram Analysis:

The *find_phrases* function was adjusted to generate three-word phrases (tri-grams), providing deeper contextual understanding of the text.

Frequent tri-grams like "finger left hand" and "causing injury described" were identified to explore detailed patterns in accident descriptions.

Top 5 tri-gram words and their frequency:

1. finger left hand - 21
2. causing injury described - 13
3. finger right hand - 13
4. injury time accident - 13
5. generating described injury - 8

Figure 33: Trigram Wordcloud

2.2.3.5 Insights for Categorization:

The frequency counts of unigrams, bi-grams, and tri-grams were examined to uncover patterns and terms frequently associated with accidents.

These insights can guide the creation of new categories for accidents based on recurring words and phrases, offering a structured approach to text-based classification.

2.2.3.6 NLP Pre-processing

The process of text preprocessing for NLP tasks and its implementation were done using NLTK and SpaCy libraries.

Key highlights include:

1. **Preprocessing Steps**:
    - Conversion to lowercase.
    - Replacement of apostrophes with standard lexicons.
    - Removal of punctuation and stopwords.
    - Lemmatization for reducing words to their base forms.
2. **Tools and Libraries Used**:
    - NLTK for downloading necessary resources (punkt, stopwords, wordnet).
    - SpaCy for tokenization and lemmatization using the English model en_core_web_sm.
3. **Implementation**:
    - Text preprocessing function (preprocess_text_spacy) filters and lemmatizes text.
    - Original and cleaned text word counts are calculated for comparison.
4. **Results**:
    - Average word count reduced by **52.52%** after cleaning, improving data efficiency.
    - Examples of descriptions before and after preprocessing are provided.

```
Average word count before cleaning: 65.06
Average word count after cleaning: 30.89
Reduction in words: 52.52%
```

5. **Dataframe Adjustments**:
    - Unnecessary columns such as Original_Word_Count and Cleaned_Word_Count were removed.
    - Cleaned_Description column was renamed back to Description.

	Description	Cleaned_Description
0	While removing the drill rod of the Jumbo 08 f...	remove drill rod jumbo maintenance supervisor ...
1	During the activation of a sodium sulphide pum...	activation sodium sulphide pump piping uncoupl...
2	In the sub-station MILPO located at level +170...	sub station milpo locate level collaborator ex...
3	Being 9:45 am. approximately in the Nv. 1880 C...	approximately nv personnel begin task unlock s...
4	Approximately at 11:45 a.m. in circumstances t...	approximately circumstance mechanic anthony gr...

Figure 34: Description & Cleaned Description

Removed the repetitive and unnecessary columns which is not required for analysis

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	WeekofYear	Weekend	Season	Cleaned_Description
0	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	1	Friday	53	0	Summer	remove drill rod jumbo maintenance supervisor ...
1	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	2	Saturday	53	1	Summer	activation sodium sulphide pump piping uncoupl...
2	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	6	Wednesday	1	0	Summer	sub station milpo locate level collaborator ex...
3	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	8	Friday	1	0	Summer	approximately nv personnel begin task unlock s...
4	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	10	Sunday	1	1	Summer	approximately circumstance mechanic anthony gr...

Figure 35: Overview of Data after Removing Unnecessary Columns

Changed the column name cleaned description into Description

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	WeekofYear	Weekend	Season	Description
0	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	1	Friday	53	0	Summer	remove drill rod jumbo maintenance supervisor ...
1	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	2	Saturday	53	1	Summer	activation sodium sulphide pump piping uncoupl...
2	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	6	Wednesday	1	0	Summer	sub station milpo locate level collaborator ex...
3	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	8	Friday	1	0	Summer	approximately nv personnel begin task unlock s...
4	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	10	Sunday	1	1	Summer	approximately circumstance mechanic anthony gr...

Figure 36: Overview of data after changing the Column Name into Description

This preprocessing pipeline prepares the dataset for NLP tasks by transforming textual data into a cleaner and more structured format while retaining essential information.

The preprocessed data was saved and reloaded for further analysis. Using the Spacy NLP library, all text from the descriptions was tokenized to extract alphabetic tokens. The frequency of each token was calculated using the Counter module, and the 30 most common words were identified. A DataFrame showcasing these words and their respective counts revealed terms like "cause," "hand," "employee," and "right" as the most frequently occurring, providing insights into prevalent themes in the dataset.

2.2.3.7 NLP Visualization after preprocessing

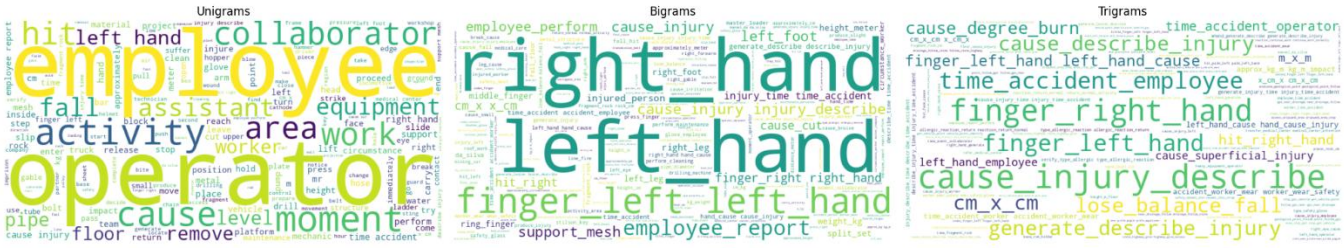

Figure 37: NLP Visualization after Preprocessing

Unigrams:

Key terms like "moment," "employee," "floor," "equipment," "assistant," "left," and "hand" suggest incidents involving employees and equipment in a specific location. Words like

"collaboration," "injury," and "support" point to teamwork and injury responses. The association of "left" with "hand" indicates workplace injuries, likely in safety or accident reports.

Bigrams:

Common bigrams such as "left hand" and "right hand" highlight the prevalence of hand and finger injuries in the data. Other phrases like "left leg" and "left foot" are noted but less frequent. Terms like "causing injury" and "employee performing" emphasize work-related injuries, while "causing cut" and "causing fall" reflect common injury mechanisms.

Trigrams:

Trigrams such as "left hand causing" and "finger left hand" focus on injuries to the left hand and fingers. Phrases like "used safety glass" suggest the application of specific safety measures. The frequent references to hands and fingers highlight their vulnerability in workplace accidents. Words like "operator" and "employee" paired with "accident" and "injury" underline the importance of roles in safety protocols.

Overall:

The n-grams analysis provides valuable insights into incident report patterns, identifying common causes of accidents and areas for safety improvement. These findings can inform strategies to reduce workplace injuries and enhance safety measures.

Preprocessed and Tokenized Descriptions

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	WeekofYear	Weekend	Season	Description	tokenized_words
0	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	1	Friday	53	0	Summer	remove drill rod jumbo maintenance supervisor ...	[remove, drill, rod, jumbo, maintenance, super...
1	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	2	Saturday	53	1	Summer	activation sodium sulphide pump piping uncoupl...	[activation, sodium, sulphide, pump, piping, u...
2	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	6	Wednesday	1	0	Summer	sub station milpo locate level collaborator ex...	[sub, station, milpo, locate, level, collabora...
3	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	8	Friday	1	0	Summer	approximately nv personnel begin task unlock s...	[approximately, nv, personnel, begin, task, un...
4	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	10	Sunday	1	1	Summer	approximately circumstance mechanic anthony gr...	[approximately, circumstance, mechanic, anthon...

Figure 38: Preprocessed and Tokenized Descriptions

Dataset Enhancements:

- A new column, tokenized_words, is added to store the tokenized descriptions for all 418 rows of data.
- The dataset includes attributes like Country, City, Industry Sector, Accident Level, Potential Accident Level, Gender, Employee type, Critical Risk, Day, Weekday, Weekend, Season, and Description.

Dataset Statistics:

- The dataset has 418 entries and 15 columns, with all values non-null.
- Columns include both numerical (e.g., Accident Level, Day) and categorical data (e.g., Gender, Season).
- Column names: Country, City, Industry Sector, Accident Level, Potential Accident Level, Gender, Employee type, Critical Risk, Day, Weekday, WeekofYear, Weekend, Season, Description, tokenized_words.

Tokenization Insights:

Tokenized descriptions facilitate text analysis for identifying patterns in workplace accidents.

Example: Descriptions of incidents like "remove drill rod jumbo maintenance supervisor" are tokenized into words such as [remove, drill, rod, jumbo, maintenance, supervisor.

Data Attributes:

The dataset covers diverse workplace scenarios across countries and sectors, highlighting risks like "Pressed" or "Fall prevention" and detailing employee roles (e.g., Contractor, Employee).

3. Deciding Models and Model Building

Below processes have been done, for data preparation and feature engineering as part of building machine learning classifiers.

1. **Data Preprocessing**:
    - A preprocessed dataset is saved in .csv format after cleansing.
2. **Feature Engineering**:
    - **Generating Word Embeddings**:
        - Three techniques are applied to the Description column:
            1. **GloVe Embeddings**:
                - A pre-trained GloVe model (glove.6B.300d.txt) is used.
                - Average word embeddings are computed for tokenized words, producing a DataFrame (Glove_df) with 300-dimensional features.
                - Shape (418, 314) indicates embeddings derived using GloVe, with 314 features for 418 rows.
            2. **TF-IDF Features**:
                - TF-IDF vectors are generated using a TfidfVectorizer, creating a DataFrame (TFIDF_df) with token-based numerical features.
                - Shape (418, 2372) indicates a sparse matrix from text data using TF-IDF, with more granular features.
            3. **Word2Vec Embeddings**:
                - A Word2Vec model is trained on tokenized data to compute average word embeddings for each entry, resulting in a 300-dimensional feature DataFrame (Word2Vec_df).
                - Shape (418, 314) matches Glove_df in dimensions, suggesting another word embedding technique.
3. **Output**:
    - The final feature-engineered datasets (Glove_df, TFIDF_df, and Word2Vec_df) contain original data columns combined with their respective embedding features, ready for use in machine learning classifiers.

Glove_df

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	GloVe_290	GloVe_291	GloVe_292	GloVe_293	GloVe_294	GloVe_295	GloVe_296	GloVe_297	GloVe_298	GloVe_299
0	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	1	Friday	...	-0.027645	-0.119045	-0.061173	-0.065187	0.026949	0.197509	-0.013762	-0.348437	-0.066048	0.009923
1	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	2	Saturday	...	-0.432424	-0.117516	0.034178	0.038456	0.132852	-0.166636	0.068733	-0.216856	-0.043625	-0.046566
2	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	6	Wednesday	...	-0.006795	-0.161874	0.020432	0.085459	0.095127	0.220992	0.045661	-0.145386	0.004915	-0.032415
3	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	8	Friday	...	-0.048605	-0.088765	0.090351	-0.046184	-0.033896	0.236031	-0.110033	-0.125069	-0.052548	-0.041803
4	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	10	Sunday	...	0.111791	-0.073450	0.056802	-0.105797	0.130160	0.158870	-0.042821	-0.077945	-0.038460	-0.072341
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	Country_01	Local_04	Mining	1	3	Male	Contractor	Others	4	Tuesday	...	0.028515	-0.027942	-0.084710	-0.077906	0.143589	0.281201	-0.145845	-0.103791	0.128524	-0.140132
414	Country_01	Local_03	Mining	1	2	Female	Employee	Others	4	Tuesday	...	0.042896	-0.137367	0.061687	0.069979	0.087773	0.194813	-0.065351	-0.239557	0.018276	-0.023313
415	Country_02	Local_09	Metals	1	2	Male	Employee	Venomous Animals	5	Wednesday	...	0.105456	-0.072907	-0.117373	0.090857	0.142089	0.118909	-0.001446	0.063939	-0.069832	-0.082433
416	Country_02	Local_05	Metals	1	2	Male	Employee	Cut	6	Thursday	...	-0.113244	-0.122123	0.062463	0.132644	0.055348	0.084847	0.011991	-0.117702	0.073389	-0.212512
417	Country_01	Local_04	Mining	1	2	Female	Contractor	Fall prevention (same level)	9	Sunday	...	-0.040730	0.015842	-0.097046	0.006672	0.197474	0.048899	0.020562	-0.270391	-0.051318	-0.059785

418 rows × 314 columns

Figure 39: Glove_df feature-engineered dataset

TFIDF_df

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	yield	yolk	young	zaf	zamac	zero	zinc	zinco	zn	zone
0	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	1	Friday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
1	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	2	Saturday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
2	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	6	Wednesday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
3	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	8	Friday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
4	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	10	Sunday	...	0.0	0.0	0.0	0.209125	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	Country_01	Local_04	Mining	1	3	Male	Contractor	Others	4	Tuesday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
414	Country_01	Local_03	Mining	1	2	Female	Employee	Others	4	Tuesday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
415	Country_02	Local_09	Metals	1	2	Male	Employee	Venomous Animals	5	Wednesday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
416	Country_02	Local_05	Metals	1	2	Male	Employee	Cut	6	Thursday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
417	Country_01	Local_04	Mining	1	2	Female	Contractor	Fall prevention (same level)	9	Sunday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0

418 rows × 2372 columns

Figure 40: TFIDF_df feature-engineered dataset

In [ ]: Word2Vec_df

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	Word2Vec_290	Word2Vec_291	Word2Vec_292	Word2Vec_293	Word2Vec_294	Word2Vec_295	Word2Vec_296	Word2Vec_297	Word2Vec_298	Word2Vec_299
0	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	1	Friday	...	0.003935	0.016489	0.013650	-0.000932	0.016581	0.019649	-0.000345	-0.013697	0.008858	0.000136
1	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	2	Saturday	...	0.001039	0.006528	0.005196	0.000250	0.005741	0.006675	-0.000189	-0.004446	0.003726	-0.000496
2	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	6	Wednesday	...	0.003613	0.017732	0.014682	-0.000805	0.016902	0.018620	-0.000015	-0.012701	0.009081	-0.000460
3	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	8	Friday	...	0.003055	0.015674	0.013100	-0.000233	0.015029	0.016948	-0.000456	-0.012074	0.008415	0.000116
4	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	10	Sunday	...	0.003324	0.015430	0.012236	-0.000944	0.014640	0.016859	-0.000157	-0.011522	0.008001	0.000636
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413	Country_01	Local_04	Mining	1	3	Male	Contractor	Others	4	Tuesday	...	0.003175	0.013861	0.010968	-0.000755	0.012620	0.015696	-0.000538	-0.010897	0.006875	0.000668
414	Country_01	Local_03	Mining	1	2	Female	Employee	Others	4	Tuesday	...	0.003148	0.016462	0.013535	-0.000911	0.016125	0.017319	-0.000138	-0.011436	0.008041	-0.000133
415	Country_02	Local_09	Metals	1	2	Male	Employee	Venomous Animals	5	Wednesday	...	0.003169	0.015793	0.013512	-0.000397	0.015207	0.017074	-0.000220	-0.011987	0.008563	-0.000193
416	Country_02	Local_05	Metals	1	2	Male	Employee	Cut	6	Thursday	...	0.003997	0.023713	0.019215	-0.000476	0.023788	0.026355	-0.000563	-0.016753	0.013083	-0.000742
417	Country_01	Local_04	Mining	1	2	Female	Contractor	Fall prevention (same level)	9	Sunday	...	0.002155	0.013271	0.011626	-0.000527	0.013779	0.014593	0.000223	-0.009977	0.006883	-0.000101

418 rows × 314 columns

Figure 41: Word2Vec_df feature-engineered dataset

This preparation ensures diverse feature representation for text data to improve model performance.

3.1 Checked for columns with All 3 Data Types

The summary of the analysis on Glove_df to identify columns based on data types is as follows:

1. **Columns of type object**: Include categorical and textual features such as:
    - Country, City, Industry Sector, Gender, Employee type, Critical Risk, Weekday, Season, and Description.
2. **Columns of type int64**: Include numerical features representing counts or discrete levels:
    - Accident Level, Potential Accident Level, Day, WeekofYear, and Weekend.
3. **Columns of type float64**: Represent high-dimensional numerical embeddings (300-dimensional GloVe vectors) with columns named:
    - GloVe_0 to GloVe_299.

The summary of the analysis on TFIDF_df based on their data types is as follows:

1. **Object Data Type:** Columns containing categorical or text data:
    - Examples: 'Country', 'City', 'Industry Sector', 'Gender', 'Employee type', 'Critical Risk', 'Weekday', 'Season', 'Description'.
2. **Integer (int64) Data Type:** Columns with whole numerical values:
    - Examples: 'Accident Level', 'Potential Accident Level', 'Day', 'WeekofYear', 'Weekend'.
3. **Float (float64) Data Type:** Columns with continuous numerical data (potentially sparse text-related numerical representations):
    - Examples: A vast list of float column names, mostly representing terms or keywords (e.g., 'abb', 'abdoman', 'access', 'accident', 'activate').

The summary of the analysis on Word2Vec_df based on their data types is as follows:

1. **Object (Categorical) Columns**:
    - 9 columns related to descriptive or categorical data like location, industry sector, gender, and weekday.
2. **Integer (int64) Columns**:
    - 5 columns related to numerical but discrete variables, such as accident levels, day of the week, and whether it's a weekend.
3. **Float (float32) Columns**:
    - 300 columns corresponding to Word2Vec embeddings, indicating a high-dimensional feature representation of text data.

3.2 Label encoding in all the 3 Data Frames

1. **Label Encoding**:
    - Accident Level and Potential Accident Level were label-encoded in three dataframes: Glove_df, TFIDF_df, and Word2Vec_df, converting categorical variables into numerical form.
2. **Saving Intermediate Results**:
    - The updated dataframes were exported as Excel files for further use in modeling tasks related to Potential Accident Level.
3. **Feature Dropping**:
    - Columns Day, Potential Accident Level, and Description were dropped from all dataframes.
4. **Target Distribution Analysis**:
    - The target variable Accident Level showed consistent imbalanced distributions across all three embeddings. This emphasized the need for techniques like oversampling or weighted loss functions to handle class imbalance during modeling.

Accident Level	Glove	TF-IDF	Word2Vec
0	309	309	309
1	40	40	40
2	31	31	31
3	30	30	30
4	8	8	8

Figure 42: Imbalanced Distribution of Target Variable

5. **Balancing the Dataset**:
    - Using SMOTE, all three dataframes were balanced for the Accident Level target, ensuring equal representation of all classes. The balanced distribution of Accident Level had 309 instances for each class.

Accident Level	Glove (Balanced)	TF-IDF (Balanced)	Word2Vec (Balanced)
0	309	309	309
3	309	309	309
2	309	309	309
1	309	309	309
4	309	309	309

Figure 43: Balanced Distribution of Target Variable

Feature Encoding:

- Categorical features were one-hot encoded to prepare data for the SMOTE operation and subsequent modeling.

In [ ]: `Glove_df_Bal`

	WeekofYear	Weekend	GloVe_0	GloVe_1	GloVe_2	GloVe_3	GloVe_4	GloVe_5	GloVe_6	GloVe_7	...	Weekday_Monday	Weekday_Saturday	Weekday_Sunday	Weekday_Thursday	Weekday_Tuesday	Weekday_Wednesday
0	53	0	0.078223	0.040773	-0.041107	-0.293287	-0.148195	-0.085006	0.120392	-0.043692	...	0	0	0	0	0	0
1	53	1	-0.047137	0.109611	-0.049147	-0.199018	0.049427	-0.139335	0.039627	-0.095639	...	0	1	0	0	0	0
2	1	0	-0.057290	0.202640	-0.209550	-0.169683	-0.027187	-0.091942	-0.168629	-0.005628	...	0	0	0	0	0	1
3	1	0	-0.033755	0.019709	-0.029097	-0.216930	-0.088179	-0.137728	-0.017687	0.012178	...	0	0	0	0	0	0
4	1	1	-0.099598	0.082313	-0.132139	-0.090341	-0.122124	-0.055800	0.132037	0.086205	...	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1540	7	0	-0.032386	0.150688	-0.072310	-0.199612	-0.108686	-0.049934	0.060058	0.046013	...	0	0	0	0	0	1
1541	16	0	-0.001804	0.034911	-0.063450	-0.121943	-0.084910	-0.065226	0.098614	-0.000395	...	0	0	0	0	0	1
1542	9	0	-0.053629	-0.038371	-0.001241	-0.164928	-0.026603	-0.025482	0.008777	-0.027883	...	0	0	0	0	0	1
1543	6	0	-0.049208	0.173114	-0.019693	-0.221013	-0.122697	0.026380	0.081478	0.041888	...	0	0	0	0	0	0
1544	11	0	-0.030766	0.046516	-0.048639	-0.174432	-0.111411	0.025456	0.061749	0.028913	...	0	0	0	0	0	0

1545 rows × 362 columns

Figure 44: Balanced and Encoded, Glove_df data overview

In [ ]: `TFIDF_df_Bal`

	WeekofYear	Weekend	abb	abdoman	able	abratech	abrupt	abruptly	absorb	absorbent	...	Weekday_Monday	Weekday_Saturday	Weekday_Sunday	Weekday_Thursday	Weekday_Tuesday	Weekday_Wednesday	Season_Spring
0	53	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0
1	53	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	1	0	0	0	0	0
2	1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	1	0
3	1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0
4	1	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1540	7	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	1	0
1541	16	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	1	0
1542	9	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	1	0
1543	6	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0
1544	11	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0	0	0	0	0	0	0

1545 rows × 2420 columns

Figure 45: Balanced and Encoded, TFIDF_df data overview

In [ ]: `Word2Vec_df_Bal`

	WeekofYear	Weekend	Word2Vec_0	Word2Vec_1	Word2Vec_2	Word2Vec_3	Word2Vec_4	Word2Vec_5	Word2Vec_6	Word2Vec_7	...	Weekday_Monday	Weekday_Saturday	Weekday_Sunday	Weekday_Thursday	Weekday_Tuesday
0	53	0	-0.003516	0.015675	-0.000687	0.004586	-0.001380	-0.014473	0.010930	0.030558	...	0	0	0	0	0
1	53	1	-0.000671	0.006053	0.000019	0.002469	0.000168	-0.005629	0.004119	0.012029	...	0	1	0	0	0
2	1	0	-0.002983	0.015179	-0.000364	0.004057	-0.000897	-0.014798	0.010590	0.030299	...	0	0	0	0	0
3	1	0	-0.003067	0.014139	-0.000148	0.003408	-0.000862	-0.012633	0.008999	0.028314	...	0	0	0	0	0
4	1	1	-0.002698	0.013779	-0.000262	0.004192	-0.000752	-0.013023	0.009686	0.027426	...	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1540	7	0	-0.002605	0.013495	-0.001140	0.003162	-0.000453	-0.011912	0.008745	0.025350	...	0	0	0	0	0
1541	16	0	-0.002428	0.011966	-0.000451	0.002982	-0.000447	-0.010722	0.007719	0.023068	...	0	0	0	0	0
1542	9	0	-0.002105	0.009349	-0.000570	0.002378	-0.000576	-0.008233	0.006135	0.017312	...	0	0	0	0	0
1543	6	0	-0.002964	0.015613	-0.001124	0.004045	-0.000637	-0.014103	0.009942	0.029888	...	0	0	0	0	0
1544	11	0	-0.002199	0.012345	-0.000403	0.003265	-0.000432	-0.010962	0.007966	0.023611	...	0	0	0	0	0

1545 rows × 362 columns

Figure 46: Balanced and Encoded, Word2vec_df data overview

There are no missing values or duplicates in the data frame.

	DataFrame	Missing Values	Duplicates
0	Glove_df_Bal	0	0
1	TFIDF_df_Bal	0	0
2	Word2Vec_df_Bal	0	0

Figure 47: Overview of Mission Values of Duplicate

6. **Implications for Modeling**:
    - The imbalanced original data and the balanced datasets highlight the importance of using metrics like precision, recall, and F1-score for fair performance evaluation across all classes.
7. **Next Steps**:
    - Balanced datasets (Glove_df_Bal, TFIDF_df_Bal, Word2Vec_df_Bal) are ready for training with enhanced class representation, addressing the imbalanced nature of the target variable.

3.3 Data Preparation - Cleansed Data in *.xlsx or .csv* file

The final cleaned data is prepared for export by renaming the balanced DataFrames for GloVe, TF-IDF, and Word2Vec embeddings as Final_NLP_Glove_df, Final_NLP_TFIDF_df, and Final_NLP_Word2Vec_df. These DataFrames are then exported to both CSV and Excel file formats, ensuring compatibility for further use. The required library openpyxl is used for handling Excel files.

```
Final_NLP_Glove_df.head()
```

	WeekofYear	Weekend	GloVe_0	GloVe_1	GloVe_2	GloVe_3	GloVe_4	GloVe_5	GloVe_6	GloVe_7	...	Weekday_Monday	Weekday_Saturday	Weekday_Sunday	Weekday_Thursday	Weekday_Tuesday	Weekday_Wedne
0	53	0	0.078223	0.040773	-0.041107	-0.293287	-0.148195	-0.085006	0.120392	-0.043692	...	0	0	0	0	0	
1	53	1	-0.047137	0.109611	-0.049147	-0.199018	0.049427	-0.139335	0.039627	-0.095639	...	0	1	0	0	0	
2	1	0	-0.057290	0.202640	-0.209550	-0.169683	-0.027187	-0.091942	-0.168629	-0.005628	...	0	0	0	0	0	
3	1	0	-0.033755	0.019709	-0.029097	-0.216930	-0.088179	-0.137728	-0.017687	0.012178	...	0	0	0	0	0	
4	1	1	-0.099598	0.082313	-0.132139	-0.090341	-0.122124	-0.055800	0.132037	0.086205	...	0	0	1	0	0	

5 rows × 362 columns

Figure 48: Final_NLP_Glove_df overview

3.4 Base ML Classifiers

After importing the Libraries for the models following steps were done:

1. **Initialization of Classifiers**:
    - A dictionary named classifiers is created with common machine learning classifiers, such as
        - **Logistic Regression,**
        - **SVM,**
        - **Decision Tree,**
        - **Random Forest,**
        - **Gradient Boosting,**
        - **XGBoost,**
        - **Naive Bayes,**
        - **K-Nearest Neighbors [Annexure II].**
2. **Training and Evaluation Function**:
    - The function train_and_evaluate() accepts a dataset and splits it into features (X) and target (y), followed by a training-test split (80-20 ratio).
    - Each classifier is trained using the training data (X_train, y_train), and predictions are made for both training and test sets.
    - Metrics calculated:
        - **Train/Test Accuracy**: Proportion of correct predictions.
        - **Precision**: How many predicted positives are truly positive (weighted average for multiclass).
        - **Recall**: How many actual positives are correctly identified.
        - **F1-Score**: Harmonic mean of precision and recall.
        - **Training Time**: Time taken to fit the model.
        - **Prediction Time**: Time taken to predict on the test set.
3. **Evaluation Across Representations**:

- The dataset is evaluated using three text vectorization techniques: GloVe, TF-IDF, and Word2Vec.
- Results for each representation are stored in a DataFrame for easy comparison.

4. **Output DataFrames**:
   - The classification matrices (glove_df, tfidf_df, word2vec_df) contain the evaluation metrics for each classifier.

3.5 Insights from Base Classifier

GloVe Representation:

1. **Performance Highlights**:
   - **Best Performing Model**: Random Forest and Gradient Boosting showed the highest test accuracy (~99.35% and ~97.73%).
   - Logistic Regression and XGBoost also performed well, with high test accuracies (~84.46% and ~98.71%).
   - **Poor Performers**: SVM had the lowest test accuracy (28.8%).
2. **Speed**:
   - Decision Tree and Naive Bayes were the fastest in training and prediction.
   - Gradient Boosting took the most time to train (~90 seconds).
3. **Generalization**:
   - Random Forest and Gradient Boosting had consistent metrics across train and test sets, indicating good generalization.

Classification matrix for Glove

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.915049	0.915494	0.915049	0.914690	0.844660	0.852931	0.844660	0.843098	0.552824	0.024117
1	Support Vector Machine	0.360841	0.333494	0.360841	0.303584	0.288026	0.206940	0.288026	0.221212	0.548604	0.205734
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.812298	0.810740	0.812298	0.811174	0.478878	0.006691
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.983819	0.983944	0.983819	0.983744	2.616991	0.016839
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.983819	0.983837	0.983819	0.983733	91.923566	0.007093
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.987055	0.987142	0.987055	0.987044	7.352176	0.121171
6	Naive Bayes	0.684466	0.726348	0.684466	0.669862	0.679612	0.703977	0.679612	0.669049	0.011746	0.007185
7	K-Nearest Neighbors	0.836570	0.863304	0.836570	0.810828	0.822006	0.850718	0.822006	0.786323	0.006209	0.038515

Figure 49: Classification Matrix: Glove Representation – Base ML Classifiers

TF-IDF Representation:

1. **Performance Highlights**:
    - **Best Performing Models**: Random Forest (98.05%), Gradient Boosting (93.53%), and XGBoost (94.49%) showed high test accuracy.
    - Naive Bayes also performed well on this representation (95.79%).
    - **Poor Performers**: SVM had the lowest test accuracy (27.5%).
2. **Speed**:
    - Training times were generally longer compared to GloVe, especially for Logistic Regression (~7 seconds) and Gradient Boosting (~25 seconds).
3. **Generalization**:
    - Models like Random Forest showed consistent performance between train and test sets, suggesting effective handling of this representation.

Classification matrix for TFIDF

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.932039	0.933777	0.932039	0.932083	0.860841	0.878452	0.860841	0.862636	3.118558	0.036941
1	Support Vector Machine	0.348706	0.343183	0.348706	0.288026	0.275081	0.186068	0.275081	0.202436	2.205038	0.724229
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.896440	0.900927	0.896440	0.898092	0.152036	0.013515
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.977346	0.978990	0.977346	0.977527	0.521030	0.021319
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.928803	0.944271	0.928803	0.932295	26.386561	0.018245
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.944984	0.951826	0.944984	0.946714	8.598177	0.444374
6	Naive Bayes	0.999191	0.999194	0.999191	0.999191	0.957929	0.965736	0.957929	0.959332	0.071479	0.027590
7	K-Nearest Neighbors	0.827670	0.850988	0.827670	0.803432	0.773463	0.803507	0.773463	0.739317	0.044303	0.076110

Figure 50: Classification Matrix: TFIDF Representation – Base ML Classifiers

Word2Vec Representation:

1. **Performance Highlights**:
    - **Best Performing Models**: XGBoost (97.73%) and Random Forest (97.08%) again showed excellent test accuracy.
    - Gradient Boosting and Decision Tree followed closely with strong performances (~97% and ~79.61%).
    - **Poor Performers**: Naive Bayes and SVM struggled significantly, with test accuracies of 46.6% and 27.5%.

2. **Speed**:
    - Gradient Boosting required the longest training time (~70 seconds).
    - Models like Decision Tree and Naive Bayes were relatively fast.
3. **Generalization**:
    - Similar to other representations, XGBoost and Random Forest maintained consistent performance, reinforcing their robustness.

Classification matrix for Wor2Vec

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.634304	0.634396	0.634304	0.619120	0.553398	0.554543	0.553398	0.524524	0.206677	0.004715
1	Support Vector Machine	0.333333	0.348829	0.333333	0.267023	0.275081	0.194967	0.275081	0.201590	0.358189	0.123147
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.834951	0.840039	0.834951	0.835264	0.324770	0.002836
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.951456	0.951422	0.951456	0.950454	1.444206	0.009420
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.961165	0.960956	0.961165	0.960852	72.207624	0.006956
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.980583	0.980481	0.980583	0.980458	6.313678	0.125022
6	Naive Bayes	0.493528	0.544607	0.493528	0.477154	0.466019	0.469953	0.466019	0.425129	0.011570	0.007129
7	K-Nearest Neighbors	0.790453	0.813698	0.790453	0.778016	0.718447	0.719462	0.718447	0.693340	0.005880	0.033446

Figure 51: Classification Matrix: Word2Vec Representation – Base ML Classifiers

Overall:

- **Best Model Overall**: Random Forest and XGBoost consistently delivered excellent performance across all representations, with high test accuracy and reasonable training times.
- **Effect of Representation**:
    - GloVe and TF-IDF provided more consistent results compared to Word2Vec.
    - TF-IDF improved the performance of models like Naive Bayes, which relies on feature frequency.
- **Training Time**: Gradient Boosting was consistently slower to train, while Decision Tree and Naive Bayes were faster but not always effective.
- **Model-Specific Behavior**:
    - SVM performed poorly across all representations, indicating it may not suit this dataset.
    - Naive Bayes performed better with TF-IDF but struggled with GloVe and Word2Vec.

This analysis can guide model selection and optimization for similar tasks.

3.6 Classification report – Base Classifiers

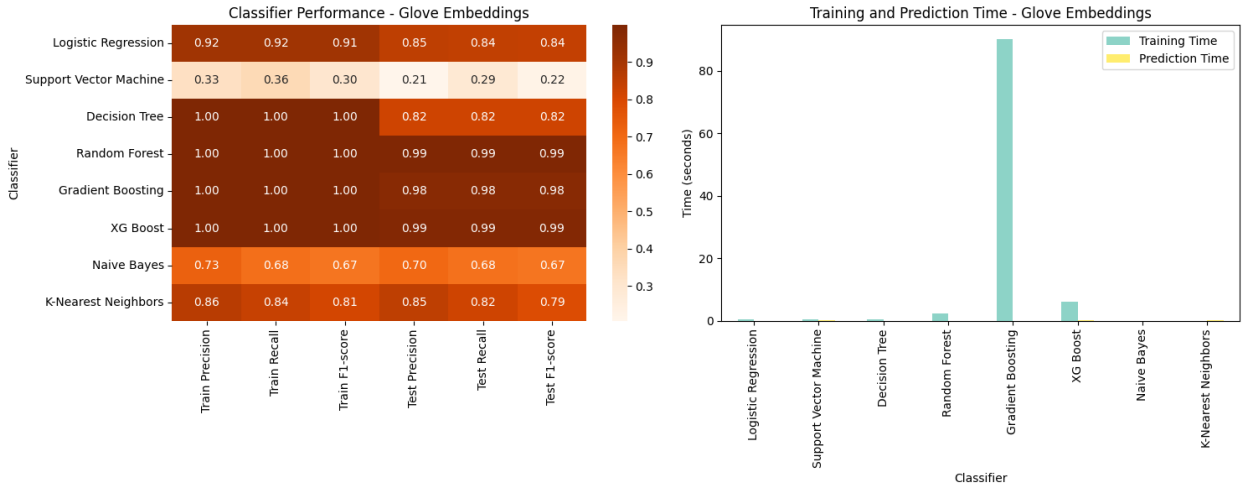

Figure 52: Base ML Classifier Performance– Glove Embeddings

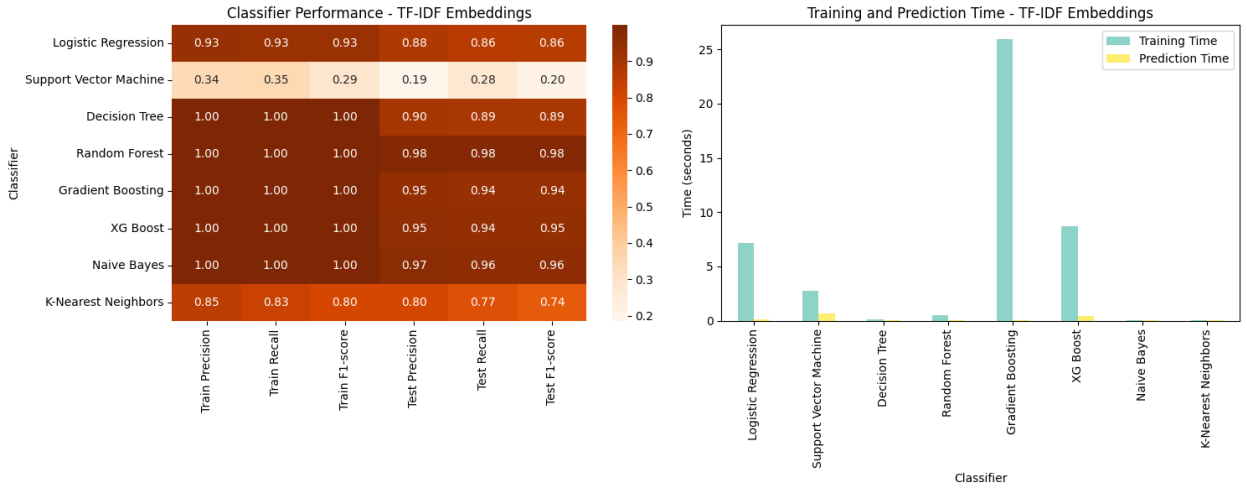

Figure 53: Base ML Classifier Performance – TF-IDF Embeddings

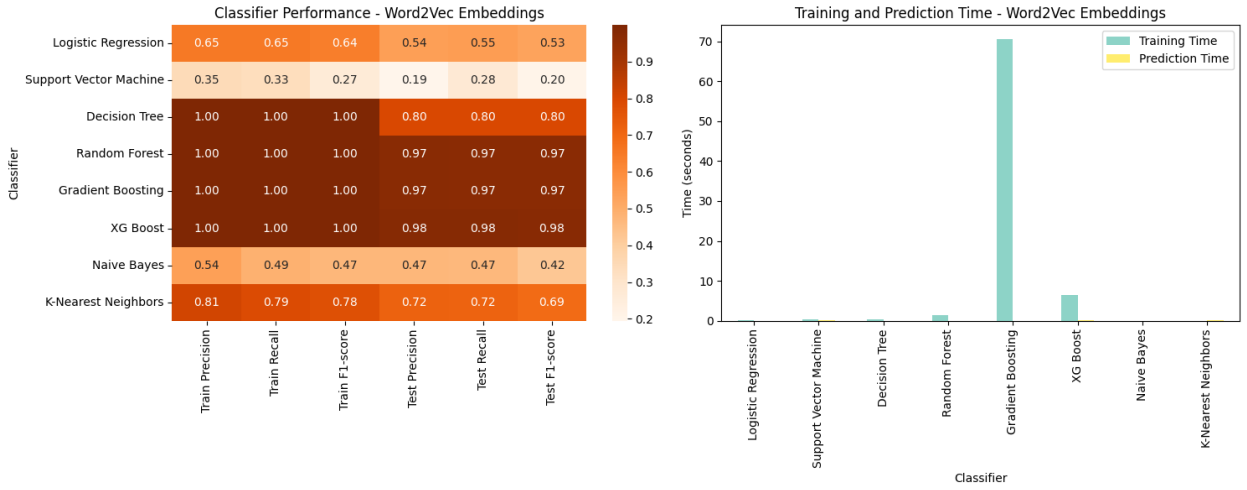

Figure 54: Base ML Classifier Performance – Word2Vec Embeddings

3.7 Confusion Matrices - Base Classifiers

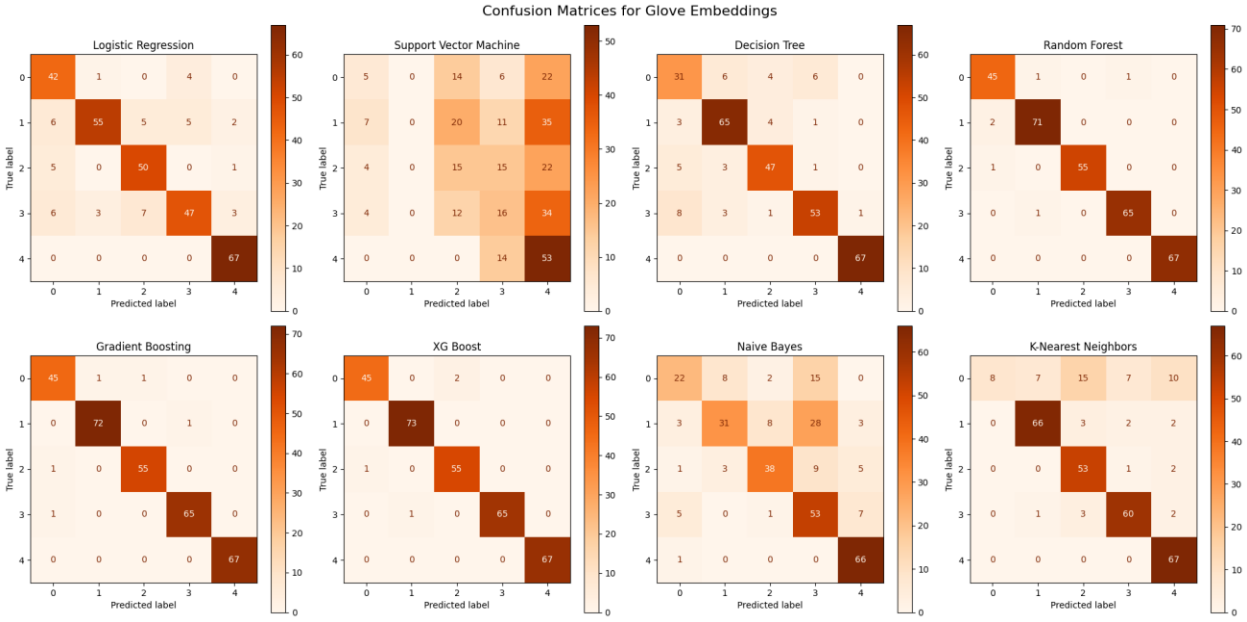

Figure 55: Confusion Matrices: Base Classifiers - Glove Embeddings

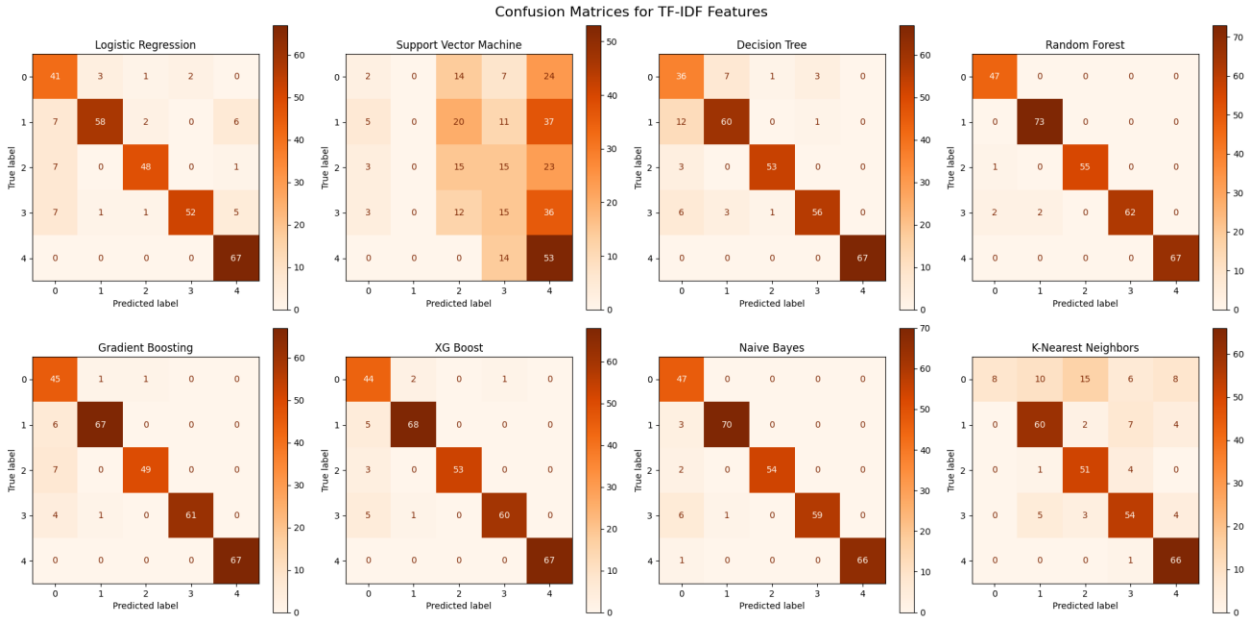

Figure 56: Confusion Matrices: Base Classifiers - TF-IDF Embeddings

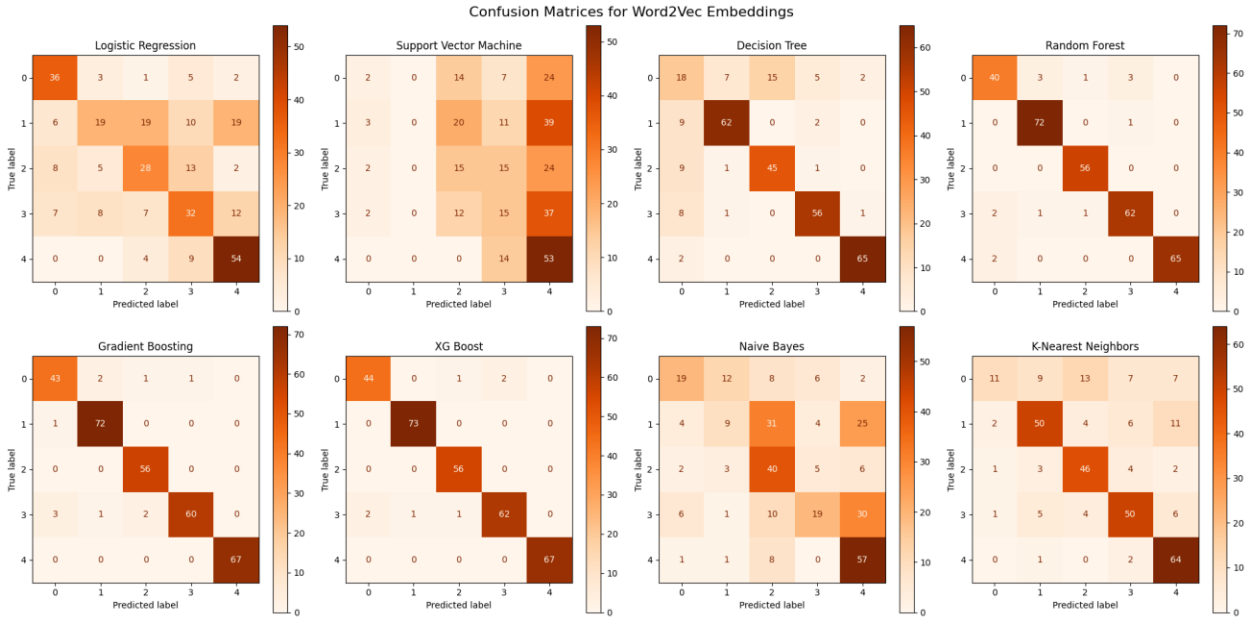

Figure 57: Confusion Matrices: Base Classifiers – Word2Vec Embeddings

3.8 Confusion Matrix Observations: Performance of Base Classifiers

1. **Overall Performance**:
    - Random Forest and XG Boost consistently deliver high accuracy across all embeddings.
    - Naive Bayes performs poorly, particularly with Glove and Word2Vec embeddings.
2. **Glove Embeddings**:
    - Random Forest, XG Boost, and Gradient Boosting demonstrate strong performance.
    - The Decision Tree exhibits more misclassifications compared to top-performing classifiers.
    - K-Nearest Neighbors performs moderately but struggles significantly with class 0.
3. **TF-IDF Features**:
    - Classifier performance improves slightly compared to Glove embeddings.
    - Logistic Regression and Support Vector Machine show better results than with Glove.
    - K-Nearest Neighbors continues to face challenges with class 0 but performs better for other classes.
4. **Word2Vec Embeddings**:
    - Performance is generally weaker compared to Glove and TF-IDF.
    - Random Forest, Gradient Boosting, and XG Boost maintain robust results.
    - Logistic Regression and Support Vector Machine show a marked decrease in accuracy, particularly for classes 1, 2, and 3.
    - Naive Bayes and K-Nearest Neighbors struggle significantly with Word2Vec.

Class-Specific Observations:

- Class 4 is consistently well-classified across embeddings and classifiers.
- Classes 0 and 1 show higher misclassification rates, particularly with Word2Vec embeddings.
- Middle classes (1, 2, 3) exhibit more confusion, especially with Word2Vec.

Model Complexity:

- Complex models like Random Forest, XG Boost, and Gradient Boosting generally outperform simpler ones.
- Simpler models such as Logistic Regression and SVM are more sensitive to the choice of embedding.

Embedding Effectiveness:

- TF-IDF features deliver consistent performance across classifiers.
- Glove embeddings are effective with complex models.
- Word2Vec embeddings are less effective for this task, particularly with simpler models.

The classifier and embedding selection significantly impact performance. Ensemble methods, such as Random Forest and boosting algorithms, are robust across embeddings. TF-IDF emerges as the most reliable feature representation, while Word2Vec may require more advanced models to achieve competitive results. The findings emphasize the importance of tailoring embeddings and models to the specific characteristics of the text data and classification task.

3.9 Train vs Test Confusion Matrices for all Base ML classifiers

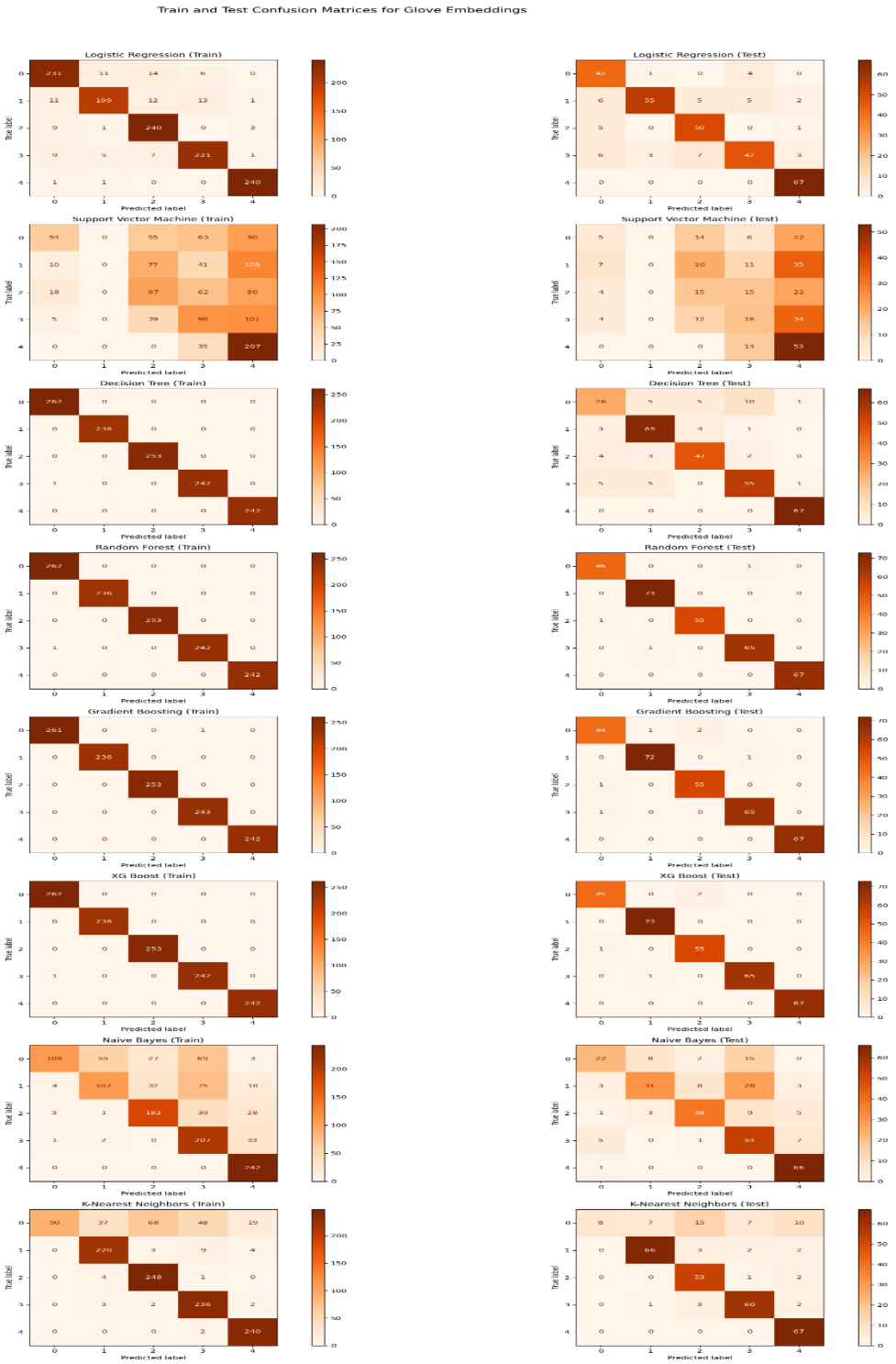

Figure 58: Base ML: Train and Test Confusion Matrices for Glove Embeddings

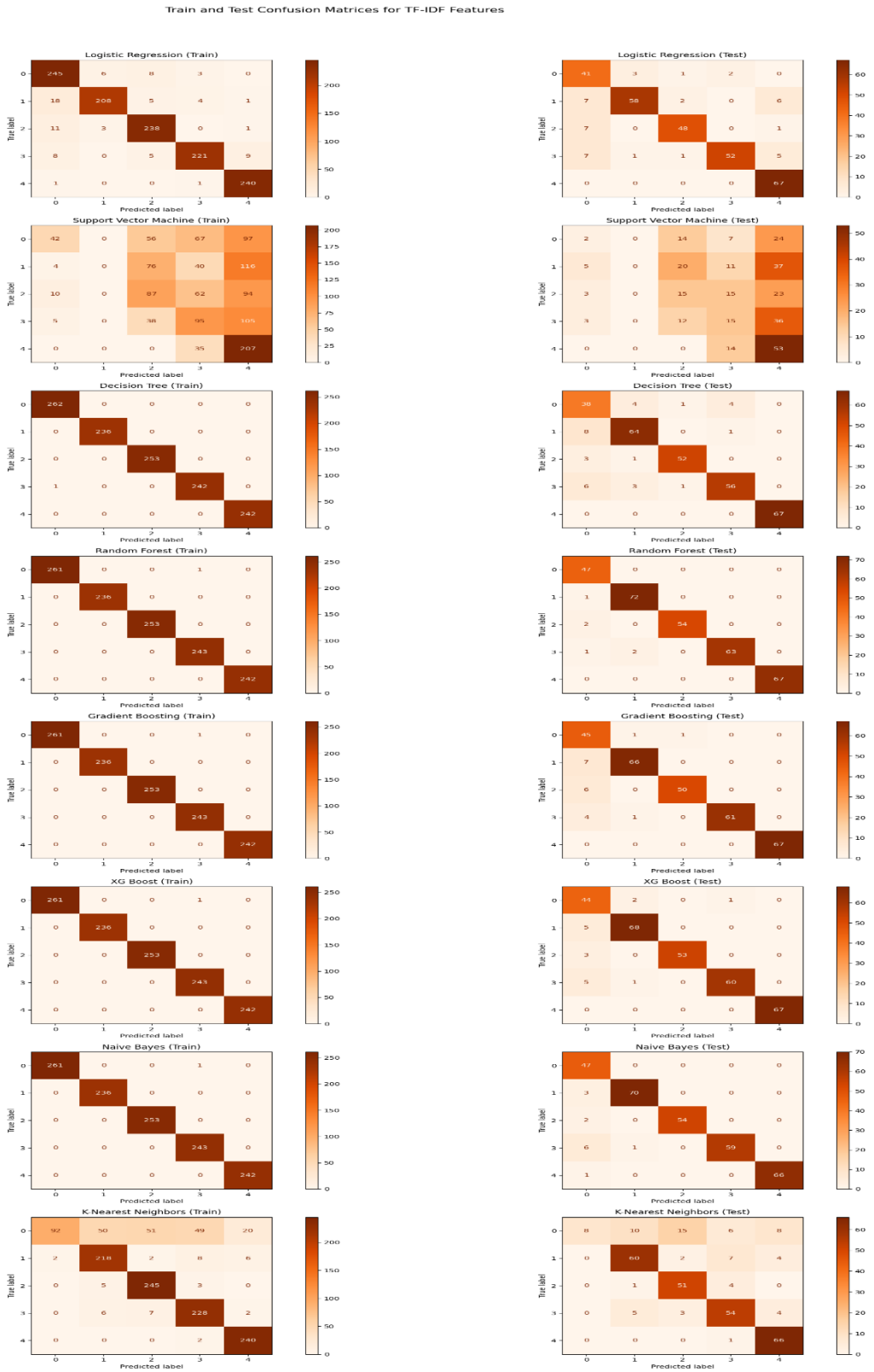

Figure 59: Base ML: Train and Test Confusion Matrices for TF-IDF Embeddings

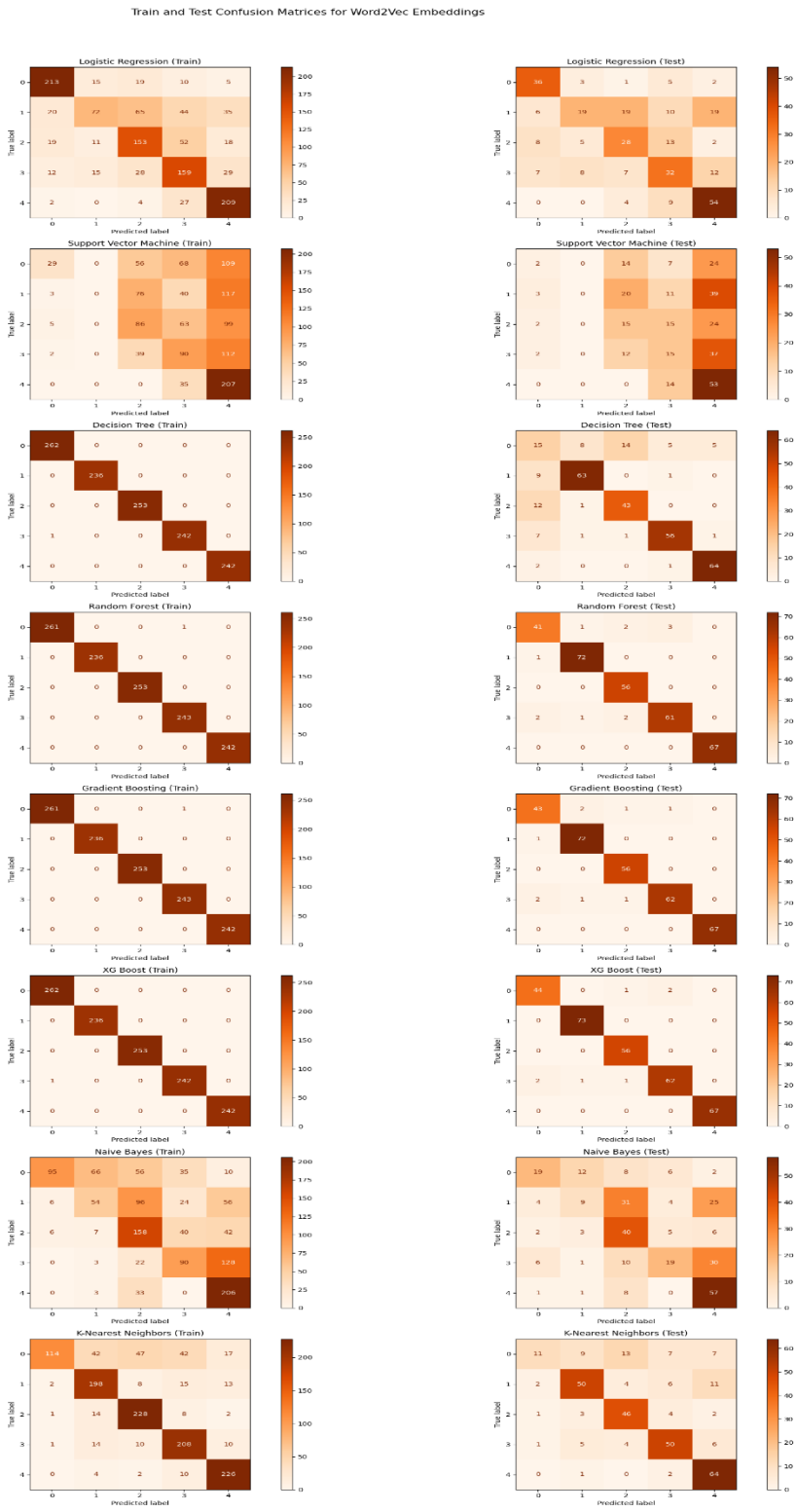

Figure 60: Base ML: Train and Test Confusion Matrices for Word2Vec Embeddings

4. Improving our model performance

4.1 Base ML Classifiers + PCA

1. **PCA and Scaling Application:**
    - The function apply_pca_and_split is used to:
        - Split the dataset into features (X) and target (y).
        - Scale features using StandardScaler.
        - Perform Principal Component Analysis (PCA) to reduce dimensionality while retaining 99% variance (n_components=0.99).
        - Split the transformed data into training and testing sets (80-20 split).
    - This process is applied separately to three datasets: Glove embeddings, TF-IDF features, and Word2Vec embeddings.
2. **PCA Variance Analysis:**
    - The print_pca_variance function:
        - Scales features using StandardScaler.
        - Fits PCA to compute the explained variance ratio for each principal component.
        - Outputs the explained variance ratio and cumulative explained variance to analyze how much variance is captured by the principal components.
    - This analysis provides insight into how the variance is distributed across components in each dataset.
3. **Outputs:**
    - For the **Glove embeddings**, the explained variance ratio for each component and cumulative explained variance show how much of the dataset's variance is captured by successive components. Similar outputs are generated for TF-IDF features and Word2Vec embeddings.

4.1.1 Observations:

- **PCA Compression:** PCA efficiently compresses high-dimensional datasets by reducing redundant information while preserving most variance (99% retained here).
- **Variance Distribution:** For Glove embeddings, the explained variance diminishes rapidly, indicating a few principal components capture most variance, supporting dimensionality reduction without significant information loss.
- **Component Utility:** The explained variance helps determine the number of components needed for optimal performance.

4.1.2 Cumulative Explained Variance Vs Principal Components (All 3 Data Frames)

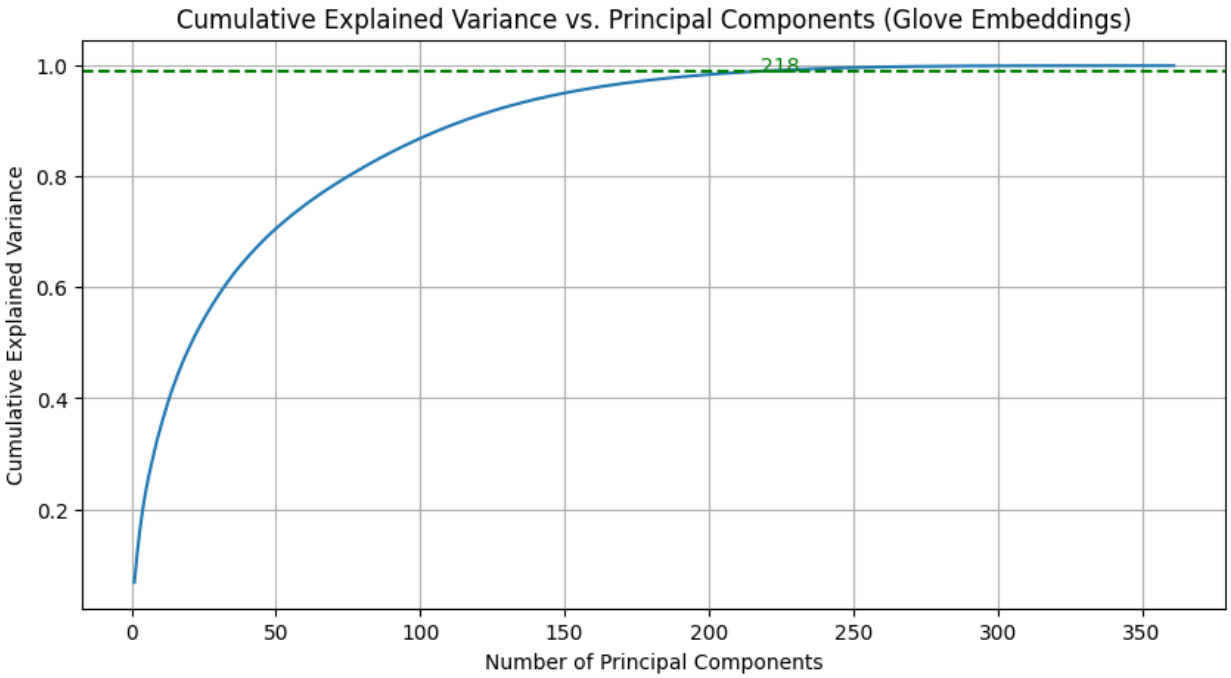

Figure 61: Cumulative Explained Variance Vs Principal Components (Glove Embeddings)

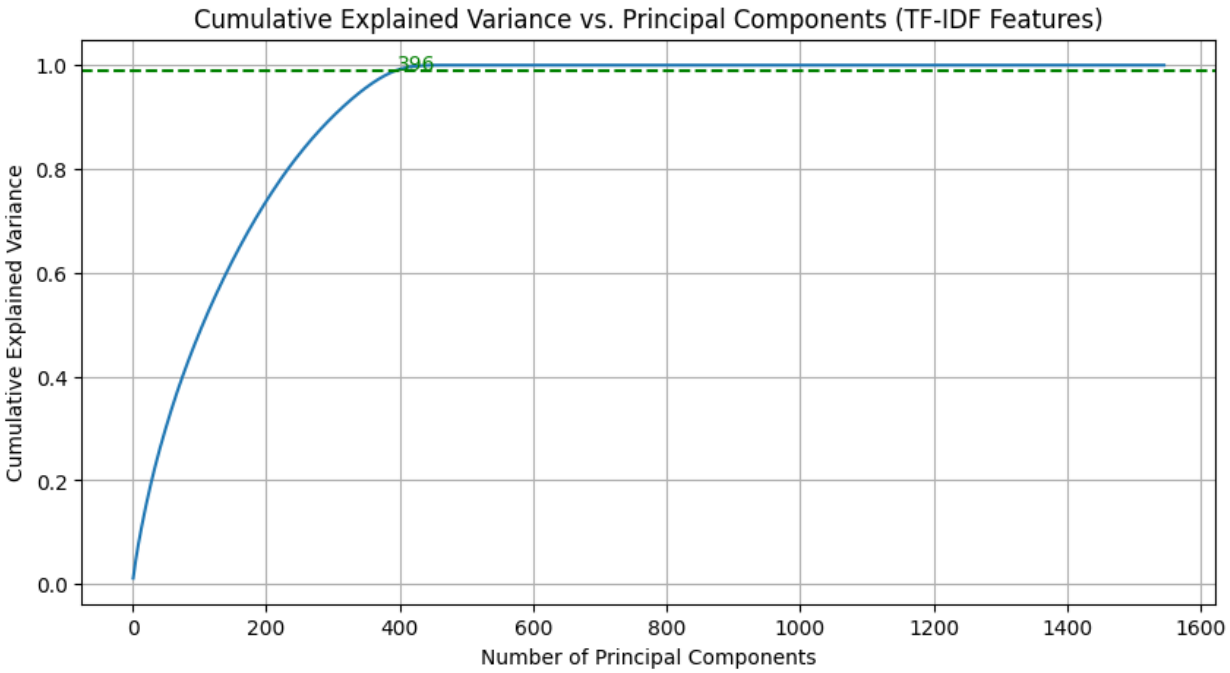

Figure 62: Cumulative Explained Variance Vs Principal Components (TF - IDF)

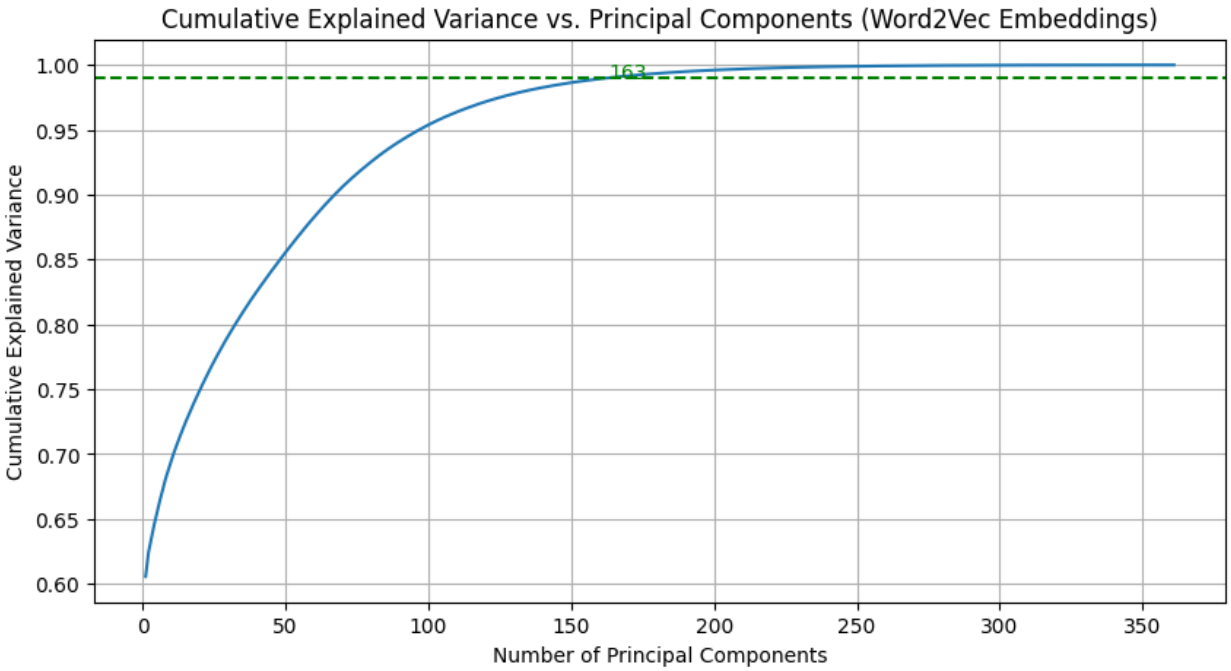

Figure 63: Cumulative Explained Variance Vs Principal Components (Word2vec Embeddings)

4.1.3 Classifiers with PCA components

4.1.3.1 GloVe Embedding with PCA:

Logistic Regression: The Test Accuracy slightly decreases with PCA, indicating a potential loss of information.

SVM: Shows a minor drop in performance, but still maintains a high accuracy.

Random Forest and XG Boost: Experience a reduction in overfitting, with more balanced training and test scores.

KNN: Performance remains relatively stable, suggesting PCA's effectiveness in reducing dimensionality without significant information loss.

Classification matrix for Glove (PCA)

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.999191	0.999194	0.999191	0.999191	0.957929	0.960863	0.957929	0.958743	0.079819	0.000337
1	Support Vector Machine	0.993528	0.993566	0.993528	0.993528	0.970874	0.974431	0.970874	0.971469	0.169137	0.060451
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.783172	0.785443	0.783172	0.784172	0.284969	0.000304
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.961165	0.965846	0.961165	0.961777	1.409135	0.006983
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.970874	0.973720	0.970874	0.971270	53.083988	0.005734
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.964401	0.967890	0.964401	0.964860	5.160899	0.003043
6	Naive Bayes	0.893204	0.893928	0.893204	0.891509	0.825243	0.834932	0.825243	0.825270	0.003442	0.001647
7	K-Nearest Neighbors	0.844660	0.872620	0.844660	0.810599	0.847896	0.876497	0.847896	0.793662	0.000756	0.005989

Figure 64: Glove Embedding with PCA

4.1.3.2 TFIDF Features with PCA:

Logistic Regression: Maintains a high Test Accuracy, showing PCA's ability to retain essential features.

SVM: Performance is consistent with and without PCA, indicating robustness to dimensionality reduction.

Random Forest and XG Boost: Show improved generalization with PCA, reducing overfitting.

KNN: Experiences a slight improvement in Test Accuracy, benefiting from reduced dimensionality.

Classification matrix for TFIDF (PCA)

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.999191	0.999194	0.999191	0.999191	0.980583	0.981012	0.980583	0.980439	0.079465	0.000648
1	Support Vector Machine	0.990291	0.990401	0.990291	0.990269	0.983819	0.984272	0.983819	0.983866	0.212710	0.063525
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.902913	0.903506	0.902913	0.902345	0.430676	0.000385
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.977346	0.977631	0.977346	0.977134	1.682344	0.006746
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.983819	0.984421	0.983819	0.983826	96.555587	0.003929
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.980583	0.980778	0.980583	0.980454	5.955721	0.002765
6	Naive Bayes	0.792880	0.816618	0.792880	0.790568	0.757282	0.770023	0.757282	0.754101	0.005089	0.002329
7	K-Nearest Neighbors	0.816343	0.895096	0.816343	0.774285	0.844660	0.903556	0.844660	0.791681	0.000815	0.008310

Figure 65: TFIDF Embedding with PCA

4.1.3.3 Word2Vec Embedding with PCA:

Logistic Regression: Performance improves with PCA, suggesting that dimensionality reduction helps in capturing essential features.

SVM: Shows a significant improvement in Test Accuracy, indicating PCA's effectiveness in handling Word2Vec's high dimensionality.

Random Forest and XG Boost: Experience a reduction in overfitting, with more balanced training and test scores.

KNN: Performance remains stable, benefiting from PCA's dimensionality reduction.

Classification matrix for Word2Vec (PCA)

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.998382	0.998388	0.998382	0.998380	0.925566	0.927712	0.925566	0.926466	0.069828	0.000369
1	Support Vector Machine	0.977346	0.977419	0.977346	0.977364	0.906149	0.921392	0.906149	0.910200	0.127228	0.061526
2	Decision Tree	0.999191	0.999194	0.999191	0.999191	0.747573	0.738226	0.747573	0.739947	0.170091	0.000326
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.938511	0.954455	0.938511	0.941858	1.142696	0.006971
4	Gradient Boosting	0.999191	0.999194	0.999191	0.999191	0.944984	0.947230	0.944984	0.945470	39.508523	0.004593
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.935275	0.947812	0.935275	0.938181	3.478531	0.003237
6	Naive Bayes	0.875405	0.880381	0.875405	0.875763	0.818770	0.835890	0.818770	0.823406	0.002889	0.001141
7	K-Nearest Neighbors	0.872168	0.888543	0.872168	0.854383	0.873786	0.877957	0.873786	0.854452	0.000673	0.004610

Figure 66: Word2Vec Embedding with PCA

4.1.4 Insights and Comparison of PCA:

PCA's Impact: PCA generally helps in reducing overfitting, especially for complex models like Random Forest and XG Boost, by balancing training and test scores.

Embedding Techniques: GloVe and TFIDF continue to perform well with PCA, while Word2Vec shows significant improvement, highlighting PCA's effectiveness in handling high-dimensional data.

Model Robustness: Logistic Regression and SVM demonstrate robustness to PCA, maintaining high performance across different embeddings.

Dimensionality Reduction: PCA proves beneficial in reducing dimensionality without significant information loss, particularly for Word2Vec, which inherently has high dimensionality.

4.1.5 Classification report and training/prediction times (PCA)

Figure 67: Classifier Performance - Glove Embeddings (PCA)

Figure 68: Classifier Performance – TF-IDF Embeddings (PCA)

Figure 69: Classifier Performance – Word2Vec Embeddings (PCA)

4.1.6 Confusion matrix against all classifiers with PCA

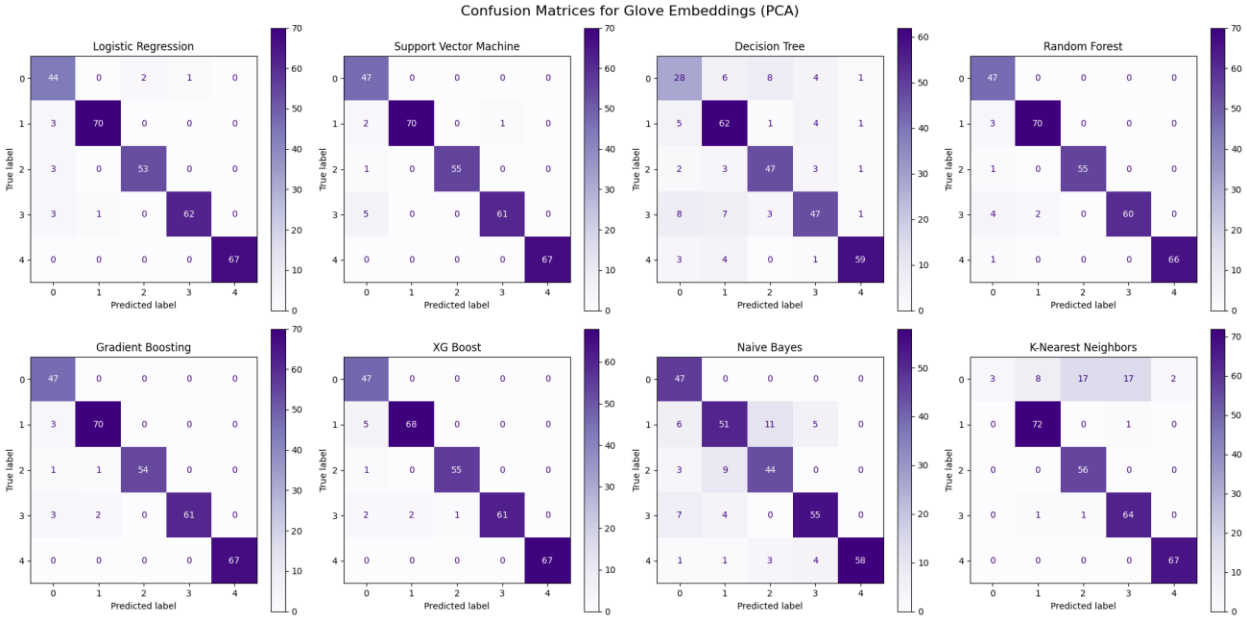

Figure 70: Confusion Matrices for Glove Embeddings (PCA)

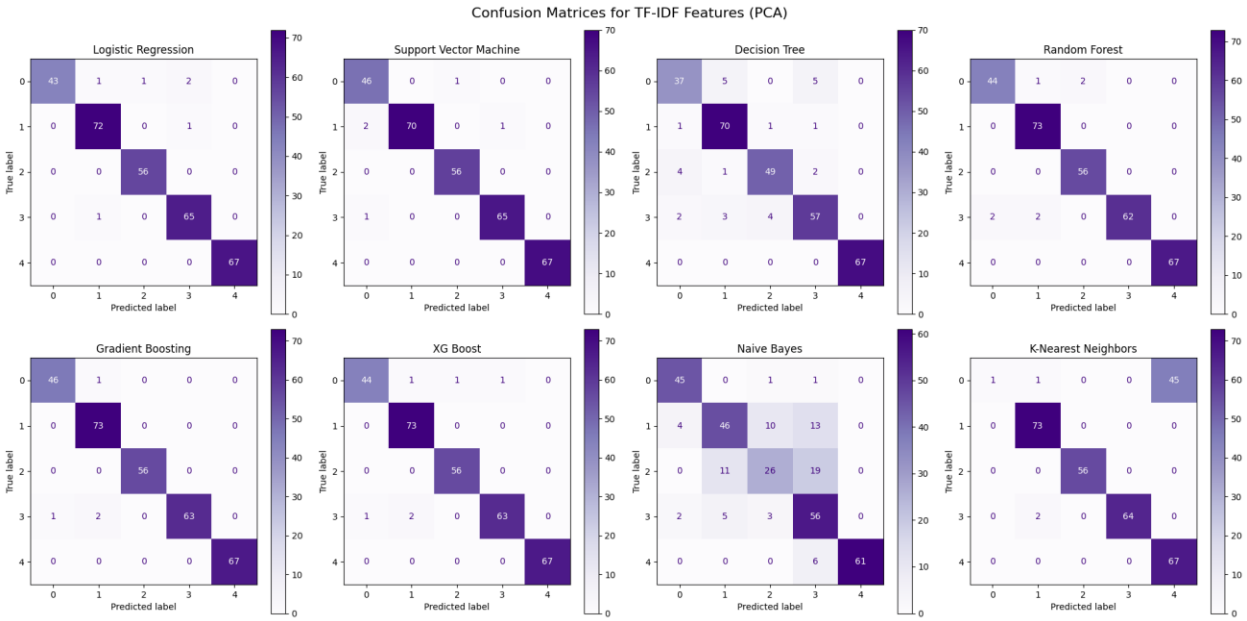

Figure 71: Confusion Matrices for TF-IDF Embeddings (PCA)

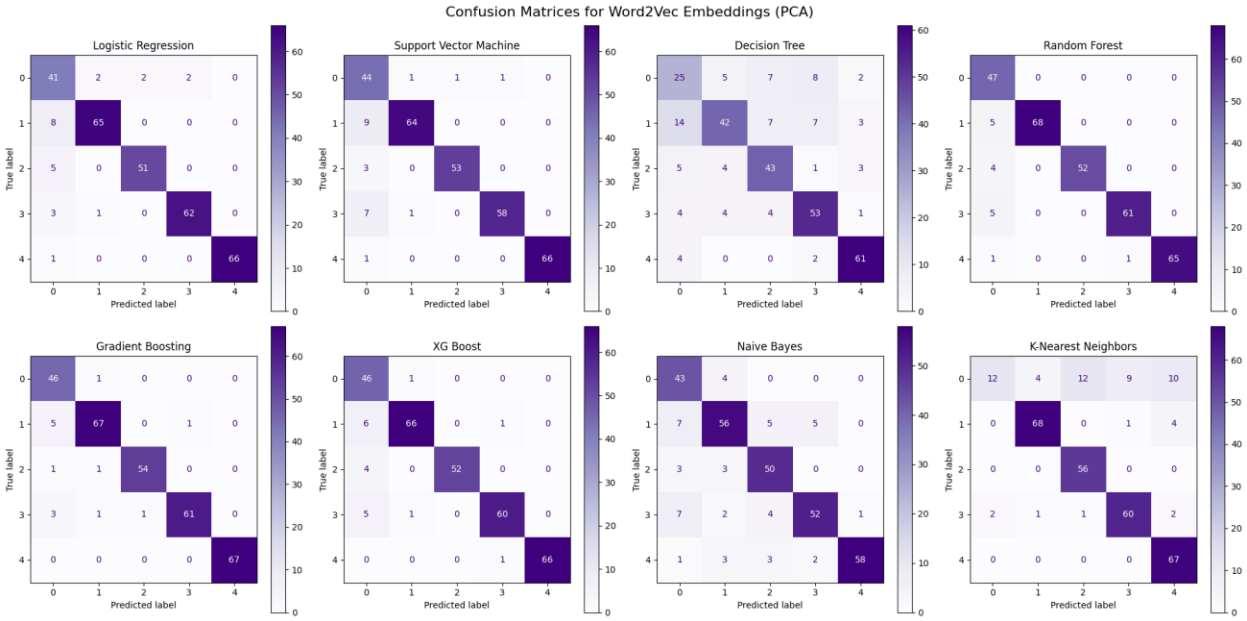

Figure 72: Confusion Matrices for Word2Vec Embeddings (PCA)

4.1.7 Confusion Matrix Observations: (Base Classifiers + PCA) Overall Performance:

PCA generally improved the performance of simpler models like Logistic Regression and SVM across all embeddings. Random Forest and XGBoost maintain strong performance, similar to non-PCA results.

Glove Embeddings with PCA:

Improved performance for Logistic Regression and SVM compared to non-PCA Glove embeddings. K-Nearest Neighbors shows better classification, especially for class 0.

TF-IDF Features with PCA:

Slight improvements across most classifiers compared to non-PCA TF-IDF. Naive Bayes shows notable improvement, especially for classes 1 and 2.

Word2Vec Embeddings with PCA:

Significant improvement for Logistic Regression and SVM compared to non-PCA Word2Vec. K-Nearest Neighbors and Naive Bayes still struggle but show some improvement.

Class-specific observations:

Class 4 remains well-classified across all embeddings and classifiers. PCA helped reduce misclassifications between middle classes (1, 2, 3) for most models.

Model Complexity:

PCA narrowed the performance gap between simpler and more complex models.

Embedding Effectiveness with PCA:

Word2Vec embeddings benefited the most from PCA, showing substantial improvements. TF-IDF features with PCA provide the most consistent performance across classifiers.

Applying PCA generally improved model performance, especially for simpler models and Word2Vec embeddings. It helped in reducing the dimensionality of the data while preserving important features, leading to better classification results.

4.1.8 Train vs Test Confusion Matrices for all ML classifiers with PCA

Figure 73: Train and Test Confusion Matrices for Glove Embeddings (PCA)

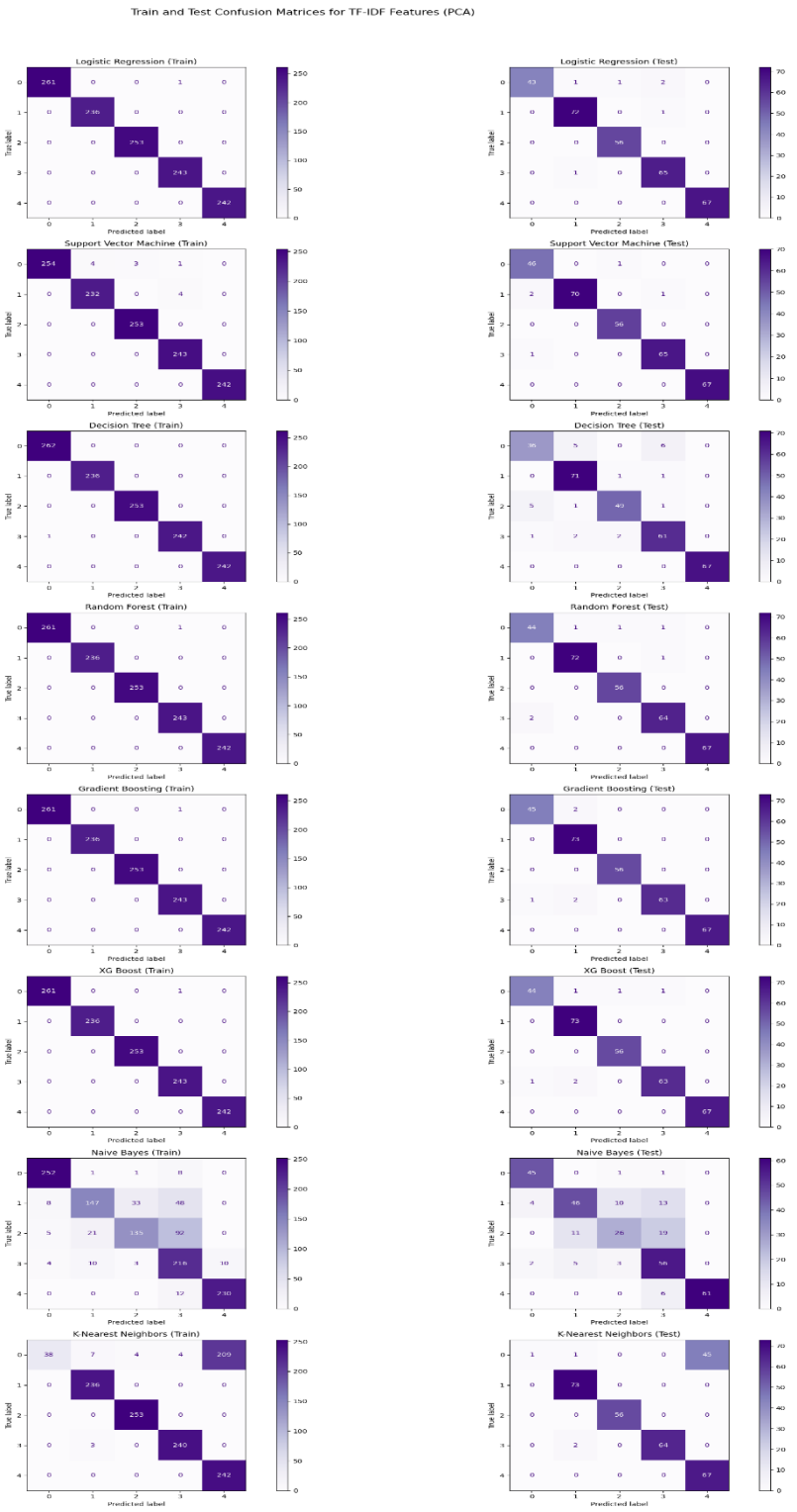

Figure 74: Train and Test Confusion Matrices for TF-IDF Embeddings (PCA)

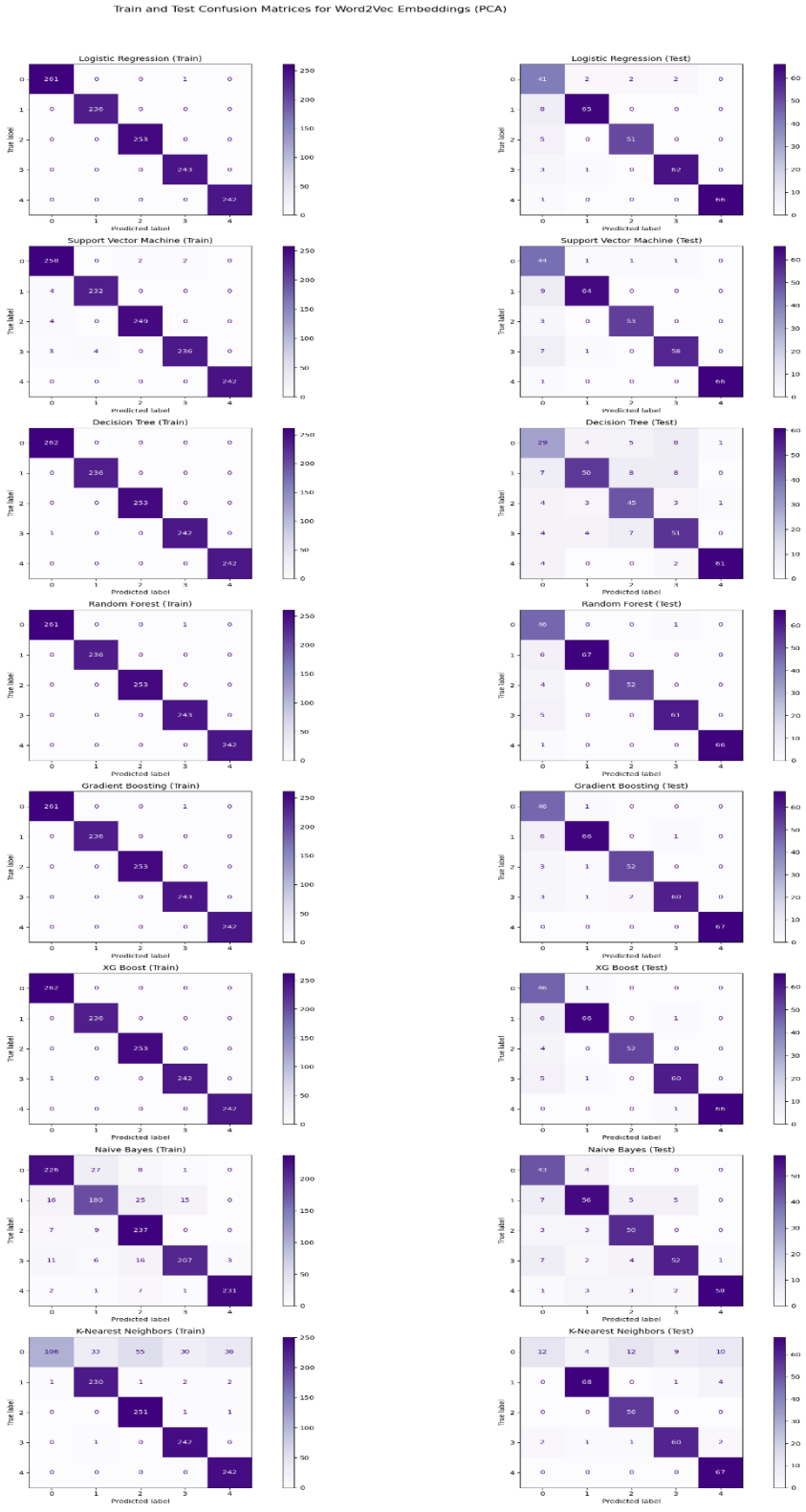

Figure 75: Train and Test Confusion Matrices for Word2Vec Embeddings (PCA)

4.2 Base ML Classifiers + Hypertuning

4.2.1 Overview of Hypertuning Classifiers Without PCA

This process involves applying hyperparameter tuning to various machine learning classifiers across three different embedding representations: **GloVe**, **TF-IDF**, and **Word2Vec**, without using Principal Component Analysis (PCA). Each embedding is evaluated independently to determine the best-performing classifiers and parameter configurations.

1. **Data Preparation**:
    - Data is prepared from preprocessed datasets (Final_NLP_Glove_df, Final_NLP_TFIDF_df, and Final_NLP_Word2Vec_df), splitting features (X) and labels (y).
2. **Data Splitting**:
    - Each embedding dataset is divided into training and testing sets using an 80/20 split.
3. **Classifier Definitions**:
    - A variety of classifiers are used, including:
        - Logistic Regression
        - Support Vector Machine (SVM)
        - Decision Tree
        - Random Forest
        - Gradient Boosting
        - XGBoost
        - Naive Bayes
        - K-Nearest Neighbors (KNN)
    - Each classifier, except Naive Bayes, is coupled with a hyperparameter grid for tuning.
4. **Scoring Metrics**:
    - Multiple evaluation metrics are considered:
        - Accuracy
        - Precision
        - Recall
        - F1-Score (used for model refit during hyperparameter tuning)

5. **Hyperparameter Tuning**:
    - **RandomizedSearchCV** is employed for efficient tuning, with cross-validation (CV=5) and weighted F1-score as the primary refit metric.
6. **Evaluation**:
    - Models are evaluated on both training and testing data.
    - Metrics include accuracy, precision, recall, F1-score, training time, and prediction time.
7. **Results Consolidation**:
    - Results for each embedding are compiled into DataFrames, detailing the performance of all classifiers with their best hyperparameter configurations.

4.2.2 Insights in Hypertuning Model

4.2.2.1 For GloVe Embedding:

- **Top Performers**:
    - Random Forest: Best test F1-score (0.9903), high accuracy, precision, and recall.
    - XGBoost: Competitive with Random Forest, test F1-score of 0.9871.
- **Efficiency**:
    - Naive Bayes has the fastest training time but underperforms on test metrics.
    - Gradient Boosting takes the longest training time (~1973 seconds).
- **Overall**:
    - Random Forest and XGBoost are the most effective classifiers for GloVe embeddings.

```
print("Glove Results")
display(glove_results)
```

Glove Results

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time	Best Parameters
0	Logistic Regression	0.999191	0.999194	0.999191	0.999191	0.944984	0.951921	0.944984	0.946712	76.977031	0.006441	{'solver': 'liblinear', 'penalty': 'l1', 'max_...
1	Support Vector Machine	0.998382	0.998388	0.998382	0.998380	0.941748	0.943732	0.941748	0.942403	13.299704	0.022220	{'max_iter': 10000, 'kernel': 'linear', 'gamma...
2	Decision Tree	0.991909	0.991970	0.991909	0.991913	0.822006	0.818089	0.822006	0.815764	11.847926	0.005203	{'min_samples_split': 2, 'min_samples_leaf': 1...
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.990291	0.990555	0.990291	0.990332	37.341051	0.017229	{'n_estimators': 200, 'min_samples_split': 2, ...
4	Gradient Boosting	0.995955	0.995968	0.995955	0.995953	0.983819	0.983837	0.983819	0.983733	1973.364891	0.006900	{'validation_fraction': 0.1, 'n_iter_no_change...
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.987055	0.987238	0.987055	0.987073	689.656654	0.115531	{'subsample': 0.9, 'n_estimators': 200, 'max_d...
6	Naive Bayes	0.684466	0.726348	0.684466	0.669862	0.679612	0.703977	0.679612	0.669049	0.328387	0.007179	{}
7	K-Nearest Neighbors	0.999191	0.999194	0.999191	0.999191	0.873786	0.880271	0.873786	0.840505	4.867650	0.172474	{'weights': 'distance', 'p': 1, 'n_neighbors': 3}

Figure 76: Glove Results (Hyper parameter Tuning)

4.2.2.2 For TF-IDF Embedding:

- **Top Performers**:
  - SVM: Best test F1-score (0.9597) and test accuracy (95.79%).
  - Random Forest: Second best, with F1-score of 0.9806 and excellent overall metrics.
- **Efficiency**:
  - Logistic Regression performs well in both speed and accuracy.
  - KNN has slower prediction times despite good training metrics.
- **Overall**:
  - SVM and Random Forest are the standout classifiers for TF-IDF embeddings.

```
print("TF-IDF Results")
display(tfidf_results)
```

TF-IDF Results

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time	Best Parameters
0	Logistic Regression	0.998382	0.998388	0.998382	0.998380	0.948220	0.956991	0.948220	0.950274	450.545842	0.029148	{'solver': 'liblinear', 'penalty': 'l1', 'max_...
1	Support Vector Machine	0.998382	0.998388	0.998382	0.998380	0.957929	0.967044	0.957929	0.959706	70.571881	0.170626	{'max_iter': 10000, 'kernel': 'linear', 'gamma...
2	Decision Tree	0.984628	0.984997	0.984628	0.984677	0.880259	0.892083	0.880259	0.883771	5.975864	0.013543	{'min_samples_split': 5, 'min_samples_leaf': 1...
3	Random Forest	0.996764	0.996770	0.996764	0.996761	0.980583	0.981685	0.980583	0.980561	12.666262	0.020068	{'n_estimators': 100, 'min_samples_split': 10,...
4	Gradient Boosting	0.995146	0.995177	0.995146	0.995150	0.928803	0.938429	0.928803	0.931480	932.559323	0.020423	{'validation_fraction': 0.1, 'n_iter_no_change...
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.944984	0.951702	0.944984	0.946672	539.821156	0.886602	{'subsample': 0.9, 'n_estimators': 100, 'max_d...
6	Naive Bayes	0.999191	0.999194	0.999191	0.999191	0.957929	0.965736	0.957929	0.959332	0.582277	0.025314	{}
7	K-Nearest Neighbors	0.944175	0.948868	0.944175	0.941326	0.925566	0.933863	0.925566	0.916946	17.898405	1.371490	{'weights': 'uniform', 'p': 1, 'n_neighbors': 3}

Figure 77: TF-IDF Results (Hyperparameter Tuning)

4.2.2.2 For Word2Vec Embedding:

- **Top Performers**:
  - **XGBoost**: Best test F1-score (0.9772) and accuracy (97.73%), with high precision and recall.
  - **Gradient Boosting**: Competitive performance with a test F1-score of 0.9707 and accuracy of 97.08%.
  - **Random Forest**: Reliable performance with a test F1-score of 0.9673 and accuracy of 96.76%.
- **Efficiency**:
  - **Naive Bayes**: Fastest training time (0.22 seconds) but poor performance (F1-score: 0.4242).
  - **Gradient Boosting**: Longest training time (~1805 seconds) but excellent metrics justify the effort.
  - **Random Forest**: Balances speed and performance with reasonable training time (39 seconds) and high accuracy.
- **Overall**:
  - **XGBoost and Gradient Boosting**: Most effective classifiers with superior accuracy and F1-scores.
  - **Random Forest**: Best efficiency-performance balance.
  - **Naive Bayes and SVM**: Underperform significantly on test metrics.

```
print("Word2Vec Results")
display(word2vec_results)
```

Word2Vec Results

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time	Best Parameters
0	Logistic Regression	0.714401	0.711731	0.714401	0.704663	0.601942	0.597863	0.601942	0.581668	68.795375	0.004876	{'solver': 'liblinear', 'penalty': 'l1', 'max_...
1	Support Vector Machine	0.654531	0.633204	0.654531	0.639990	0.556634	0.535629	0.556634	0.531293	14.208719	0.034992	{'max_iter': 10000, 'kernel': 'linear', 'gamma...
2	Decision Tree	0.991100	0.991227	0.991100	0.991112	0.822006	0.816860	0.822006	0.818476	12.072298	0.004644	{'min_samples_split': 2, 'min_samples_leaf': 1...
3	Random Forest	0.999191	0.999194	0.999191	0.999191	0.967638	0.968234	0.967638	0.967282	39.093510	0.017527	{'n_estimators': 200, 'min_samples_split': 2, ...
4	Gradient Boosting	0.995955	0.995959	0.995955	0.995947	0.970874	0.970737	0.970874	0.970722	1805.133784	0.008892	{'validation_fraction': 0.1, 'n_iter_no_change...
5	XG Boost	0.999191	0.999194	0.999191	0.999191	0.977346	0.977270	0.977346	0.977178	683.136986	0.080465	{'subsample': 1.0, 'n_estimators': 200, 'max_d...
6	Naive Bayes	0.487864	0.539204	0.487864	0.469547	0.466019	0.469177	0.466019	0.424227	0.217537	0.004756	{}
7	K-Nearest Neighbors	0.999191	0.999194	0.999191	0.999191	0.779935	0.776476	0.779935	0.766190	3.779431	0.207178	{'weights': 'distance', 'p': 1, 'n_neighbors': 3}

Figure 78: Word2Vec Results (Hyperparameter Tuning)

4.2.2.3 Insights for ML Classifiers with Hypertuning

1. Different embeddings favor different classifiers:
    - Random Forest and XGBoost dominate for GloVe.
    - SVM performs exceptionally well for TF-IDF.
2. Gradient Boosting and XGBoost often require the most computational resources.
3. Naive Bayes consistently exhibits lower performance across all embeddings.

Next Steps

- Analyze Word2Vec results for further insights.
- Consider using PCA or feature selection methods to reduce dimensionality for embeddings with high computational costs (e.g., GloVe).

4.2.3 Classification report for ML classifiers with Hypertuning

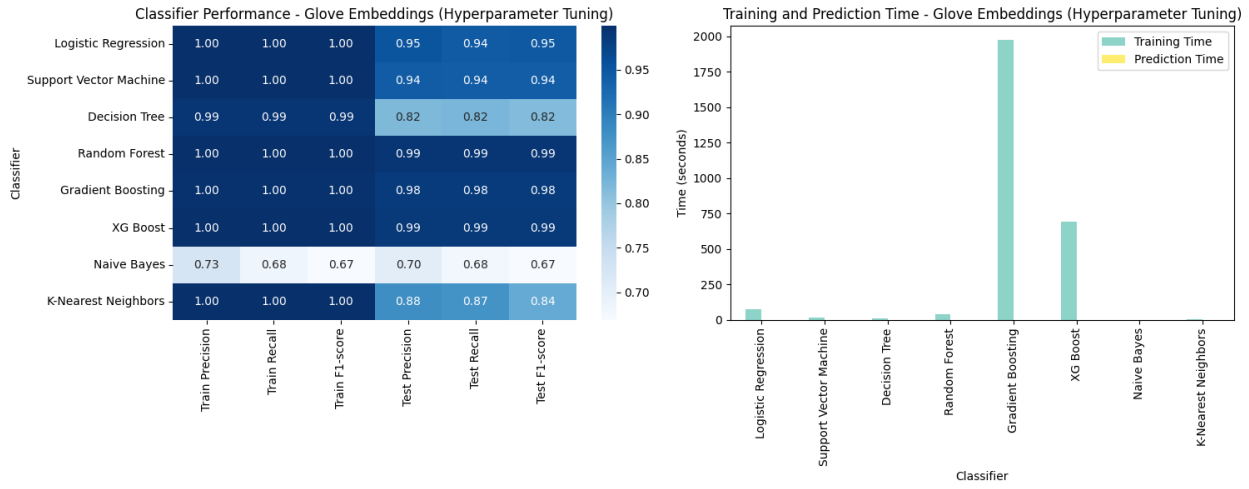

Figure 79: Classifier Performance - Glove Embeddings (Hypertuning)

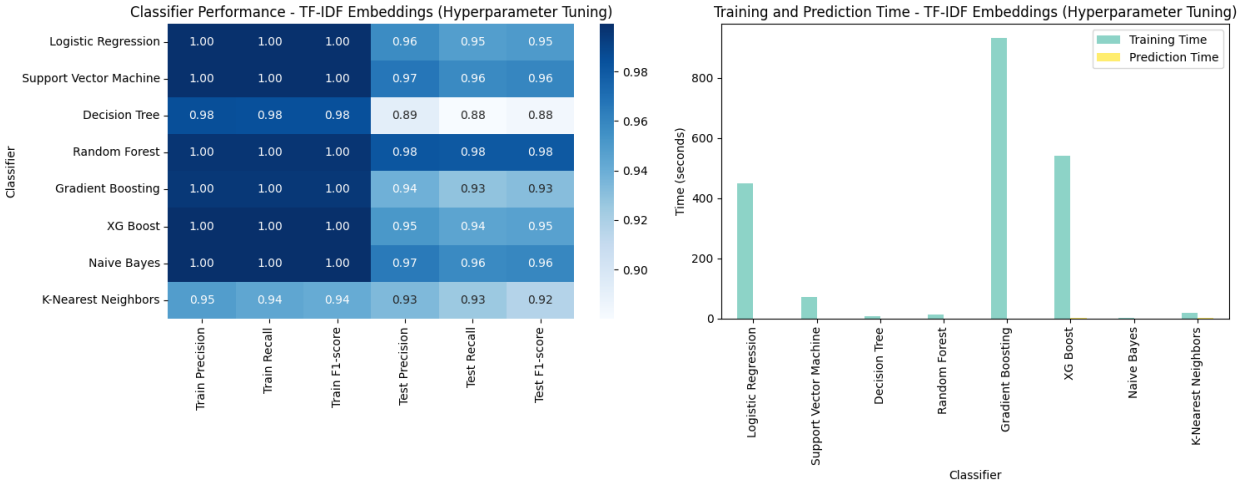

Figure 80: Classifier Performance – TF-IDF Embeddings (Hypertuning)

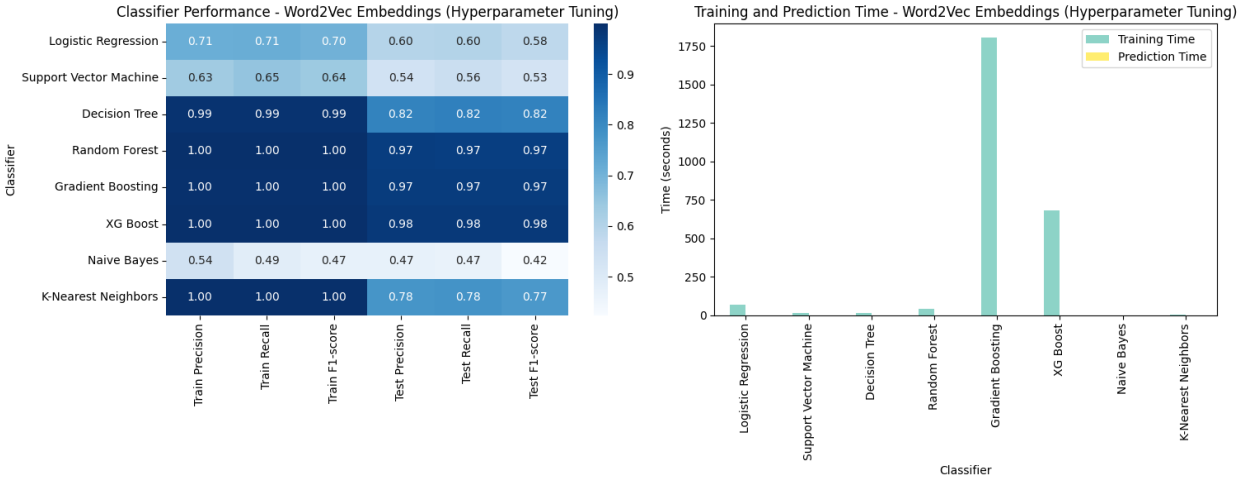

Figure 81: Classifier Performance – Word2Vec Embeddings (Hypertuning)

4.2.4 Confusion matrices against all classifiers with Hypertuning without PCA

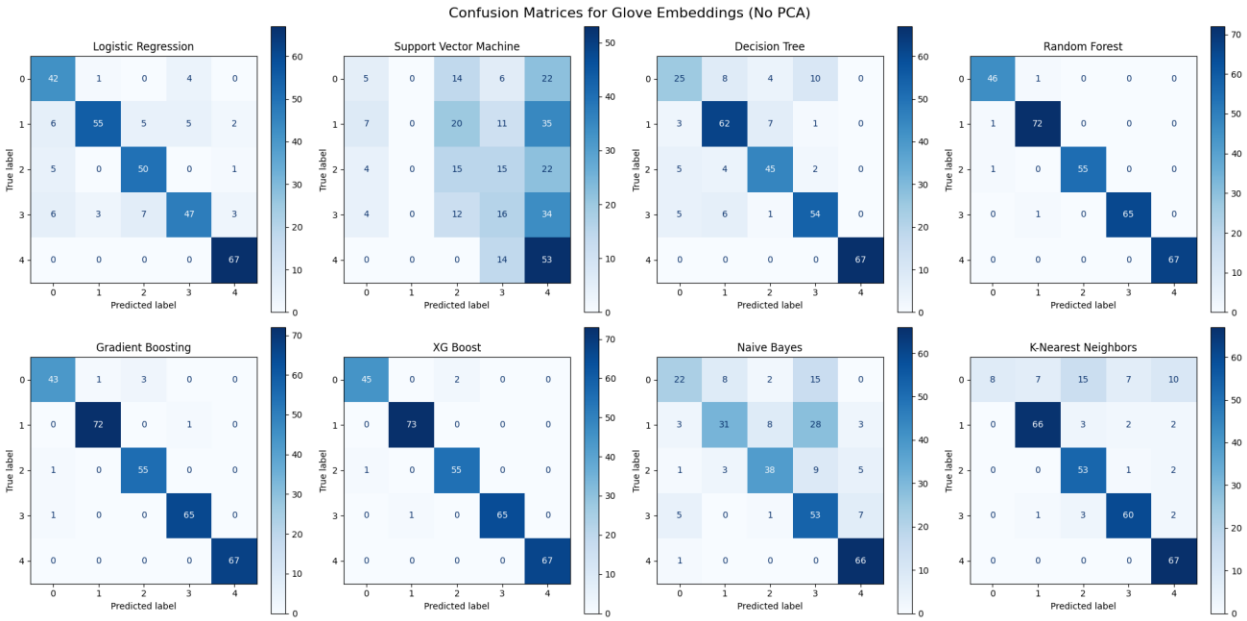

Figure 82: Confusion Matrices for Glove Embeddings (No PCA)

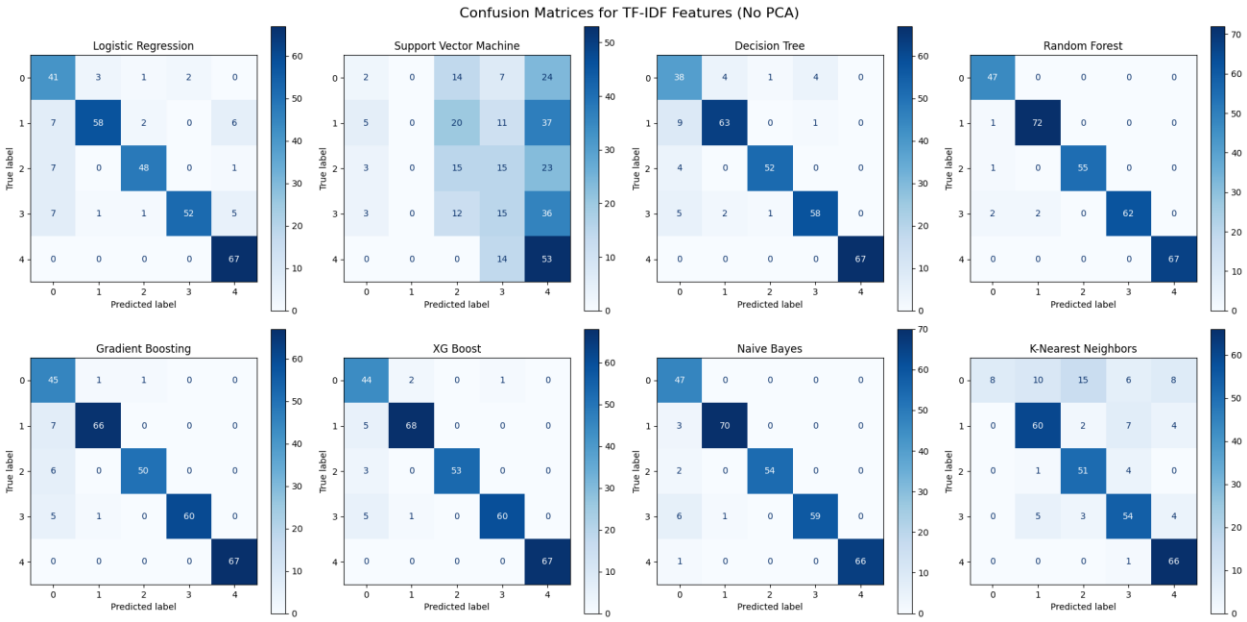

Figure 83: Confusion Matrices for TF-IDF Embeddings (No PCA)

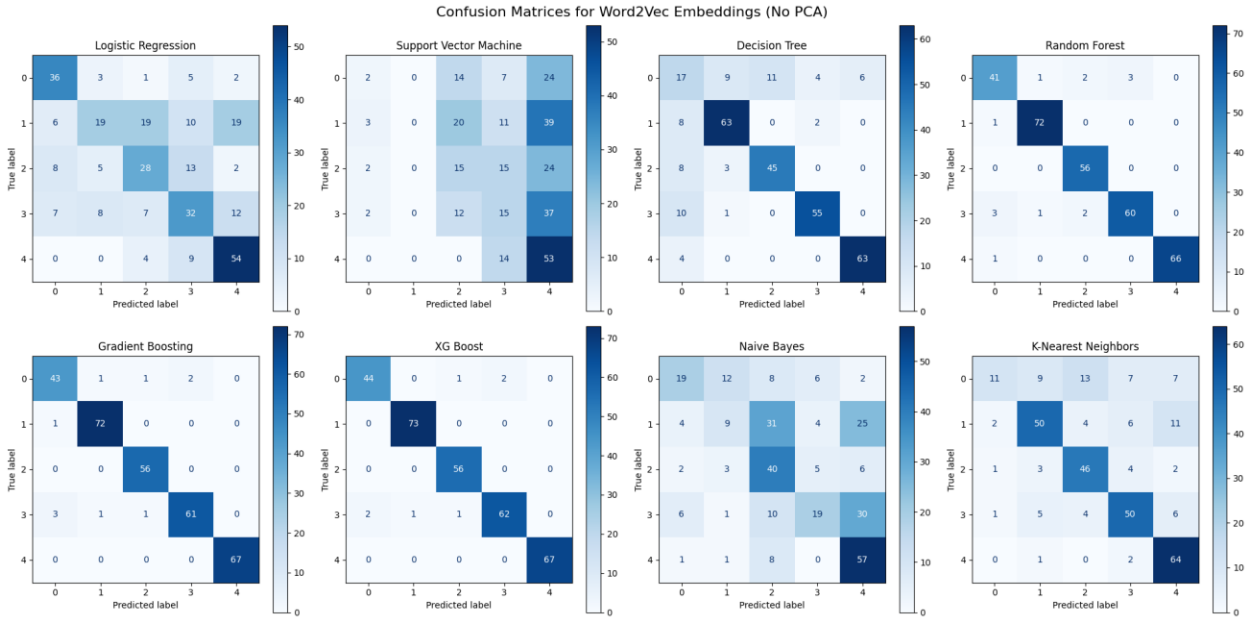

Figure 84: Confusion Matrices for Word2Vec Embeddings (No PCA)

4.2.5 Confusion Matrix Observations: (Base Classifiers + Hypertuning)

Overall Observations:

Hyperparameter tuning generally improves the performance of all classifiers across different embeddings. Ensemble methods like Random Forest, Gradient Boosting, and XG Boost consistently show top performance, indicating their robustness and effectiveness in handling various types of embeddings. Logistic Regression and SVM are very effective in binary-like class separations (e.g., classes 0 and 4) but sometimes struggle with middle classes. Naive Bayes and K-Nearest Neighbors generally show lower performance compared to more complex models, suggesting that these might require more specific tuning or might be less suitable for this particular dataset.

Glove Embeddings with Hypertuning:

Logistic Regression and SVM again perform well, with high accuracy in predicting classes 0 and 4. Gradient Boosting and XG Boost show very strong performance, with Gradient Boosting slightly outperforming XG Boost in class 2. Decision Tree shows variability in performance,

particularly struggling with class 2. Naive Bayes and K-Nearest Neighbors have higher misclassification rates compared to other classifiers.

TF-IDF Features with Hypertuning:

Logistic Regression, SVM, and Random Forest show very high accuracy, particularly in classes 0 and 4. Gradient Boosting and XG Boost are highly effective, with nearly perfect classification in several classes. Decision Tree shows improved performance but still has some difficulty with class 2. Naive Bayes performs well in class 1 but has some issues in other classes. K-Nearest Neighbors shows decent performance but is not as effective as other classifiers.

Word2Vec Embeddings with Hypertuning:

Logistic Regression and Support Vector Machine (SVM) show strong performance, particularly in correctly predicting classes 0 and 4. Decision Tree and Naive Bayes exhibit more misclassifications, especially in the middle classes (1, 2, 3). Random Forest and XG Boost demonstrate excellent accuracy, with very few misclassifications across all classes. K-Nearest Neighbors shows improved performance but still struggles with some classes compared to ensemble methods.

Comparison with Non-Hyperparameter Tuned Models:

Hyperparameter tuning has notably enhanced the accuracy and reduced misclassifications across almost all classifiers and embeddings. The improvement is particularly evident in models that initially showed moderate performance, such as K-Nearest Neighbors and Decision Tree. The gap between simpler models and complex ensemble models has narrowed, but ensemble models still generally lead in performance. This analysis indicates that hyperparameter tuning is crucial for optimizing model performance, especially when dealing with diverse embeddings and complex classification tasks.

4.2.6 Train vs Test Confusion Matrices for all ML classifiers with Hypertuning

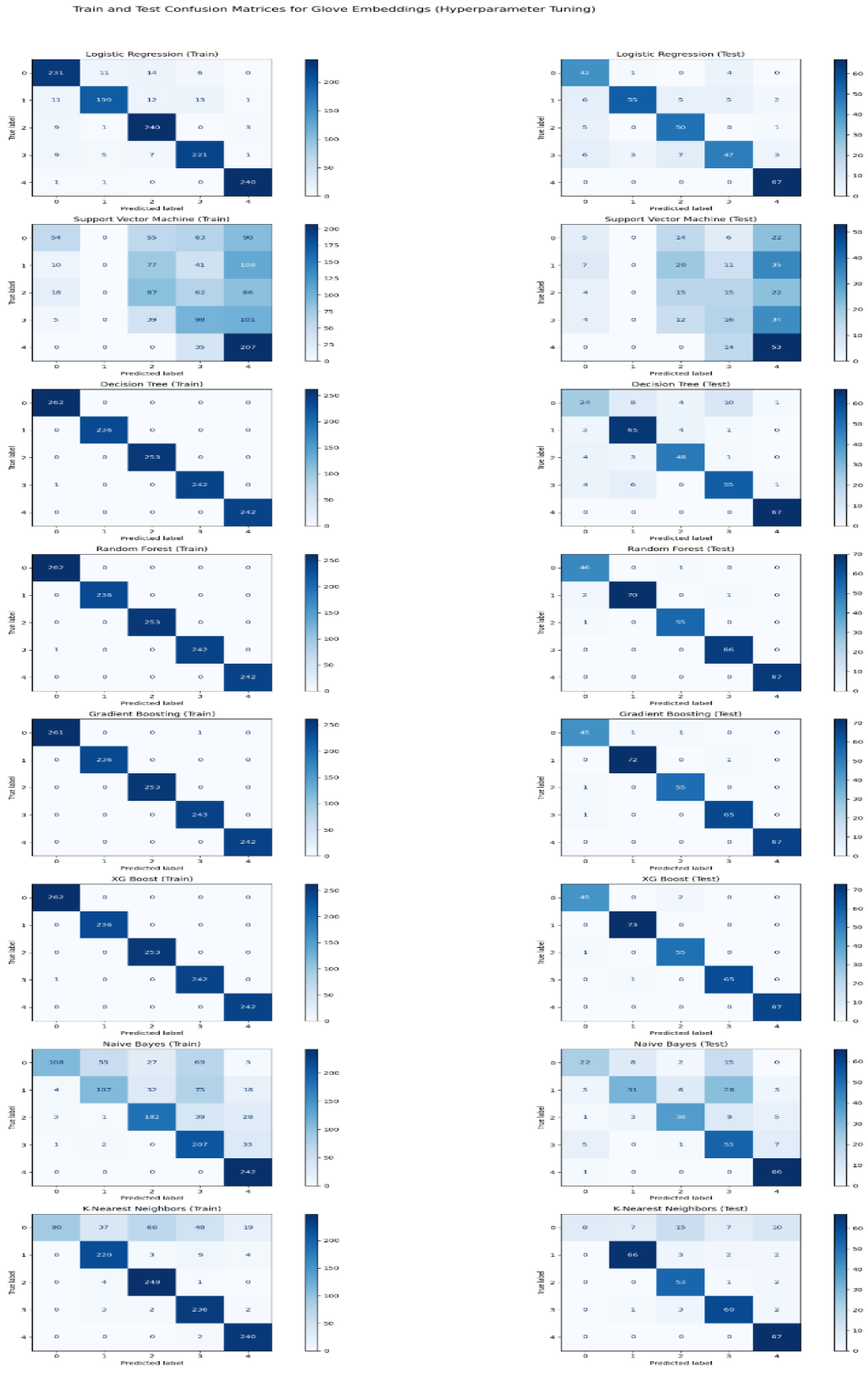

Figure 85: Train and Test Confusion Matrices for Glove Embeddings (Hyperparameter Tuning)

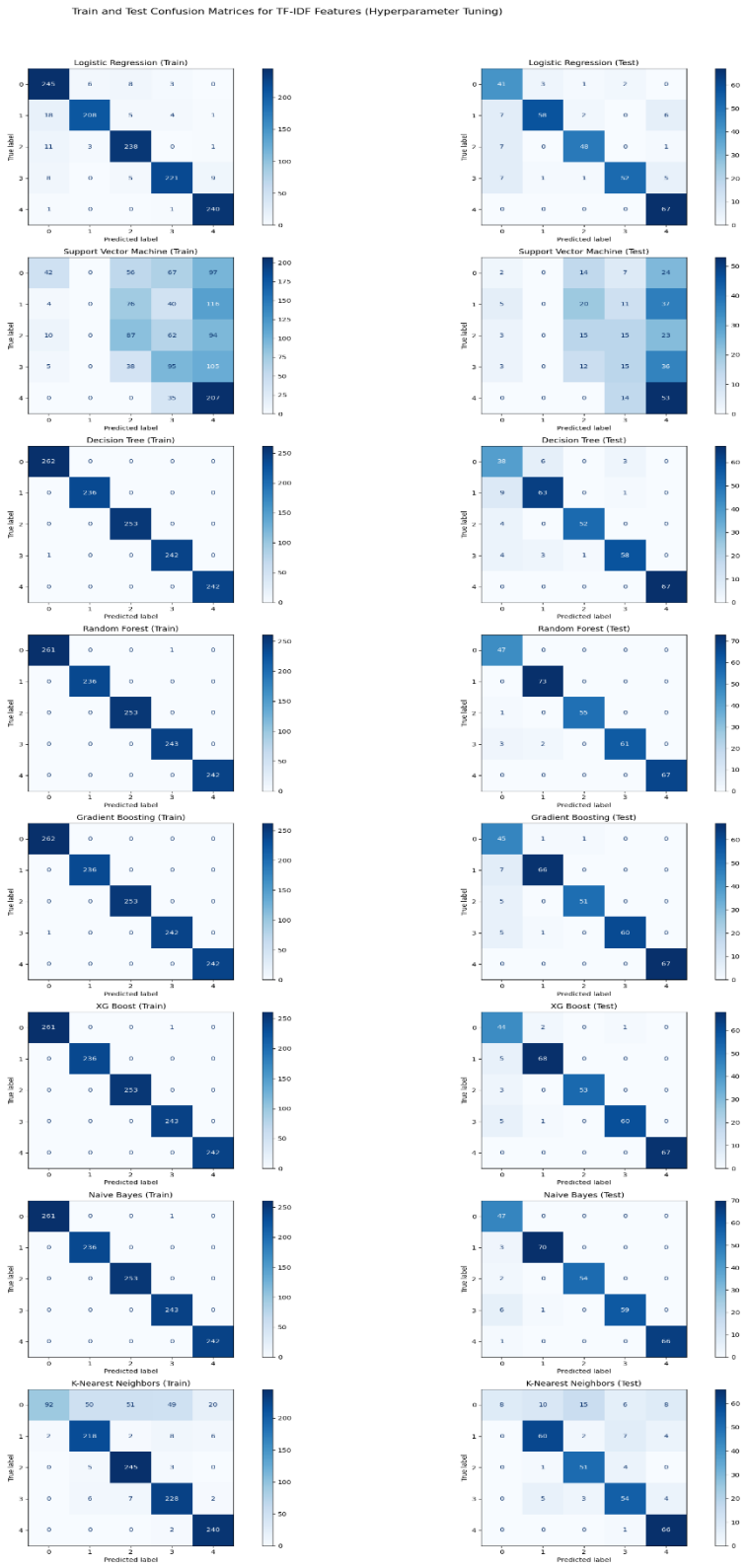

Figure 86: Train and Test Confusion Matrices for TF-IDF Embeddings (Hyperparameter Tuning)

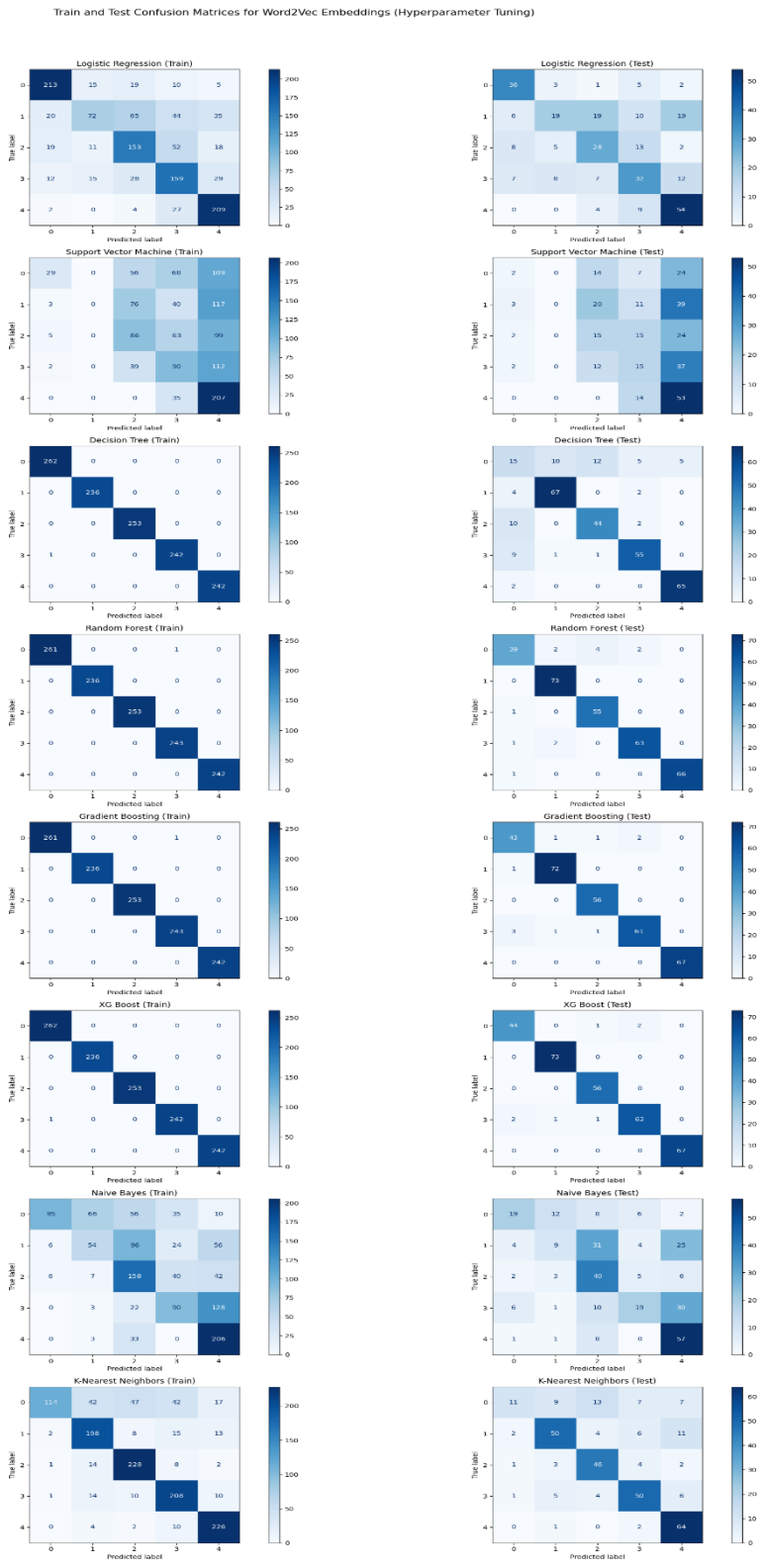

Figure 87: Train and Test Confusion Matrices for Word2Vec Embeddings (Hyperparameter Tuning)

5. Overall Observations and Insights:

Overall Performance Improvement:

1. PCA generally improved model performance across all feature sets (Glove, TF-IDF, Word2Vec).
2. Hypertuning with PCA further enhanced performance for most models.

Consistent Top Performers:

1. Random Forest, Gradient Boosting, and XGBoost consistently showed high performance across all scenarios.
2. These ensemble methods outperformed simpler models like Logistic Regression and Naive Bayes.

Feature Set Comparison:

1. Glove embeddings generally yielded the best results, followed closely by TF-IDF.
2. Word2Vec performed slightly worse than the other two feature sets.

Impact of PCA:

1. PCA significantly improved the performance of simpler models like Logistic Regression and Support Vector Machine.
2. It also reduced training and prediction times for most models.

Hypertuning Benefits:

1. Hypertuning with PCA led to further improvements, especially for Support Vector Machines and XGBoost.

Trade-offs:

1. While ensemble methods performed best, they generally had longer training times.

2. Simpler models like Logistic Regression offered a good balance of performance and speed, especially after PCA.

5.1 Recommendations:

1. **Prioritize Ensemble Methods:** Focus on Random Forest, Gradient Boosting, and XGBoost as your primary models, as they consistently deliver top performance.
2. **Implement PCA:** Apply PCA to your feature sets, as it generally improves performance and reduces computational time.
3. **Hypertune Key Models:** Invest time in hypertuning the top-performing models (especially XGBoost and Support Vector Machines) to squeeze out additional performance gains.
4. **Consider Glove Embeddings:** Prioritize using Glove embeddings as your primary feature set, with TF-IDF as a strong alternative apply PCA to your feature sets, as it generally improves performance and reduces computational time.
5. **Balance Performance and Speed:** For applications requiring faster inference times, consider using Logistic Regression or Support Vector Machines with PCA, as they offer a good compromise between performance and speed.
6. **Ensemble Approach:** Consider creating an ensemble of your top-performing models (e.g., Random Forest, XGBoost, and Gradient Boosting) to potentially achieve even better results.
7. **Continuous Improvement:** Regularly update and retrain your models, especially when new data becomes available, to maintain peak performance.
8. **Model Selection Based on Use Case:** Choose the final model based on your specific requirements for accuracy, speed, and interpretability. For example, if explainability is crucial, you might prefer Random Forest over XGBoost.

5.2 Potential Accident level Class Specific Observation based on ML Classifier performance

Class-specific observations: (Potential Accident level Predicted Vs Actual)

- Class 4 is consistently well-classified across all embeddings and most classifiers.
- Classes 0 and 1 often see more misclassifications, especially in Word2Vec embeddings.
- The middle classes (1, 2, 3) tend to have more confusion between them, particularly in Word2Vec.

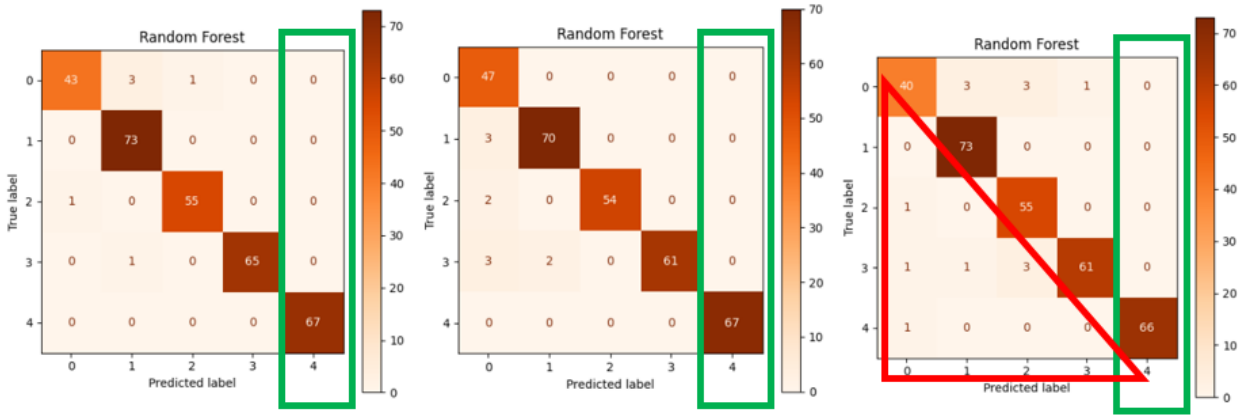

Figure 88: Confusion Metrics for Glove, TFIDF, & Word2Vec Embeddings

Class-Specific Observation Improvement with <u>Hyper Tuning</u> (Potential Accident level Predicted Vs Actual)

- With Hypertuning Class 4 is consistently well-classified across all embeddings and most classifiers.
- With Hypertuning we had seen improvement in the other potential accident level as well (Classes 0,1,2,3)

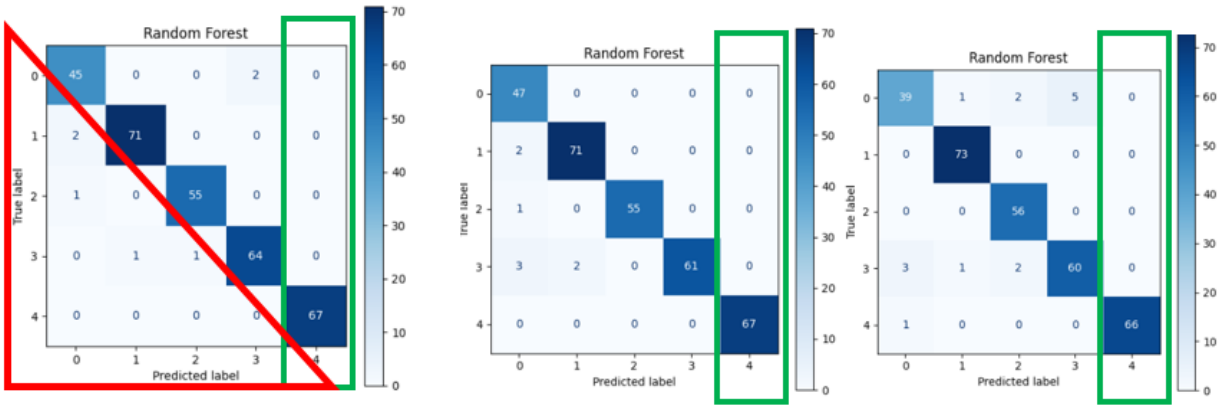

Figure 89: Confusion Metrics – Glove, TFIDF, Word2Vec Embeddings

6. Creation of ML Classifiers (Building Model 2)

The steps involve building a second machine learning model (Model 2) to predict the "Potential Accident Level" using the accident level predicted from the previous model as an input. The process uses preprocessed and feature-engineered datasets, including NLP representations (GloVe, TF-IDF, and Word2Vec). The required intermediate datasets are loaded for analysis and further model training.

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	GloVe_290	GloVe_291	GloVe_292	GloVe_293	GloVe_294	GloVe_295	GloVe_296	GloVe_297	GloVe_298	GloVe_299
0	Country_01	Local_01	Mining	0	3	Male	Contractor	Pressed	1	Friday	...	-0.027645	-0.119045	-0.061173	-0.065187	0.026949	0.197509	-0.013762	-0.348437	-0.066048	0.009923
1	Country_02	Local_02	Mining	0	3	Male	Employee	Pressurized Systems	2	Saturday	...	-0.432424	-0.117516	0.034178	0.038456	0.132852	-0.166636	0.068733	-0.216856	-0.043625	-0.046566
2	Country_01	Local_03	Mining	0	2	Male	Contractor (Remote)	Manual Tools	6	Wednesday	...	-0.006795	-0.161874	0.020432	0.085459	0.095127	0.220992	0.045661	-0.145386	0.004915	-0.032415
3	Country_01	Local_04	Mining	0	0	Male	Contractor	Others	8	Friday	...	-0.048605	-0.088765	0.090351	-0.046184	-0.033896	0.236031	-0.110033	-0.125069	-0.052548	-0.041803
4	Country_01	Local_04	Mining	3	3	Male	Contractor	Others	10	Sunday	...	0.111791	-0.073450	0.056802	-0.105797	0.130160	0.158870	-0.042821	-0.077945	-0.038460	-0.072341

5 rows × 314 columns

Figure 90: Overview of Glove_df - Model2

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	yield	yolk	young	zaf	zamac	zero	zinc	zinco	zn	zone
0	Country_01	Local_01	Mining	0	3	Male	Contractor	Pressed	1	Friday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
1	Country_02	Local_02	Mining	0	3	Male	Employee	Pressurized Systems	2	Saturday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
2	Country_01	Local_03	Mining	0	2	Male	Contractor (Remote)	Manual Tools	6	Wednesday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
3	Country_01	Local_04	Mining	0	0	Male	Contractor	Others	8	Friday	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
4	Country_01	Local_04	Mining	3	3	Male	Contractor	Others	10	Sunday	...	0.0	0.0	0.0	0.209125	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 2372 columns

Figure 91: Overview of TF-IDF - Model2

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	Word2Vec_290	Word2Vec_291	Word2Vec_292	Word2Vec_293	Word2Vec_294	Word2Vec_295	Word2Vec_296	Word2Vec_297	W
0	Country_01	Local_01	Mining	0	3	Male	Contractor	Pressed	1	Friday	...	0.004232	0.016296	0.013697	-0.000822	0.016377	0.019451	-0.000156	-0.013926	
1	Country_02	Local_02	Mining	0	3	Male	Employee	Pressurized Systems	2	Saturday	...	0.001148	0.006463	0.005217	0.000292	0.005667	0.006605	-0.000120	-0.004530	
2	Country_01	Local_03	Mining	0	2	Male	Contractor (Remote)	Manual Tools	6	Wednesday	...	0.003912	0.017574	0.014758	-0.000695	0.016726	0.018456	0.000171	-0.012955	
3	Country_01	Local_04	Mining	0	0	Male	Contractor	Others	8	Friday	...	0.003320	0.015506	0.013144	-0.000132	0.014841	0.016771	-0.000286	-0.012277	
4	Country_01	Local_04	Mining	3	3	Male	Contractor	Others	10	Sunday	...	0.003586	0.015258	0.012274	-0.000843	0.014451	0.016679	0.000012	-0.011722	

rows × 314 columns

Figure 92: Overview of Word2Vec - Model2

Then, had defined a function to train a Random Forest classifier on datasets (GloVe, TF-IDF, Word2Vec) and generate predictions for the "Accident Level" target. After splitting the data, the model is trained, and predictions are saved in a DataFrame comparing actual vs. predicted values. The GloVe dataset output shows accurate predictions with matching "Actual" and "Predicted" values.

```
Random Forest Predictions for GloVe Dataset:
      Actual  Predicted
1495       4          4
543        1          1
1268       4          4
528        1          1
1094       3          3
```

Figure 93: Random Forest Predictions for GloVe Dataset

6.1 Based on Model 2 Prediction , Predicted Accident level added to existing Dataframe

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	GloVe_291	GloVe_292	GloVe_293	GloVe_294	GloVe_295	GloVe_296	GloVe_297	GloVe_298	GloVe_299	Predicted
15	Country_02	Local_05	Metals	0	3	Male	Employee	Liquid Metal	4	Thursday	...	-0.024693	-0.047650	-0.015708	0.067314	0.167760	-0.035326	-0.099047	-0.047969	-0.027353	0
23	Country_02	Local_02	Mining	1	1	Male	Contractor (Remote)	Others	15	Monday	...	-0.081535	0.111993	-0.104607	0.018857	0.360266	-0.132124	-0.324510	0.047853	0.064667	1
29	Country_02	Local_07	Mining	1	2	Male	Employee	Others	16	Tuesday	...	0.002789	0.018379	-0.021721	-0.018772	0.089487	-0.133801	-0.083973	-0.334744	0.253727	1
30	Country_01	Local_03	Mining	0	1	Male	Employee	Others	17	Wednesday	...	0.091715	-0.004494	0.073564	0.102722	0.159337	0.028924	-0.168550	0.099812	0.025263	0
32	Country_01	Local_01	Mining	2	3	Male	Contractor	Others	21	Sunday	...	-0.120020	0.015738	-0.067694	0.145352	0.122889	-0.015679	-0.186358	0.062675	-0.020945	2

5 rows × 315 columns

Figure 94: Overview of Glove_df - Model2 with Predicted column added

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	yolk	young	zaf	zamac	zero	zinc	zinco	zn	zone	Predicted
15	Country_02	Local_05	Metals	0	3	Male	Employee	Liquid Metal	4	Thursday	...	0.0	0.0	0.0	0.0	0.0	0.208879	0.0	0.0	0.0	0
23	Country_02	Local_02	Mining	1	1	Male	Contractor (Remote)	Others	15	Monday	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	1
29	Country_02	Local_07	Mining	1	2	Male	Employee	Others	16	Tuesday	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	1
30	Country_01	Local_03	Mining	0	1	Male	Employee	Others	17	Wednesday	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0
32	Country_01	Local_01	Mining	2	3	Male	Contractor	Others	21	Sunday	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	2

5 rows × 2373 columns

Figure 95: Overview of TF-IDF_df - Model2 with Predicted column added

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	...	Word2Vec 291	Word2Vec 292	Word2Vec 293	Word2Vec 294	Word2Vec 295	Word2Vec 296	Word2Vec 297	Word2Vec 298	Word2Vec 299	Predicted
15	Country_02	Local_05	Metals	0	3	Male	Employee	Liquid Metal	4	Thursday	...	0.014042	0.011738	-0.000538	0.013538	0.014970	0.000281	-0.009459	0.007908	0.000263	0
23	Country_02	Local_02	Mining	1	1	Male	Contractor (Remote)	Others	15	Monday	...	0.016496	0.014118	-0.001274	0.016156	0.018516	-0.000271	-0.013377	0.009046	0.000376	1
29	Country_02	Local_07	Mining	1	2	Male	Employee	Others	16	Tuesday	...	0.015245	0.011324	0.001624	0.014556	0.017180	-0.001124	-0.012113	0.008363	0.001192	1
30	Country_01	Local_03	Mining	0	1	Male	Employee	Others	17	Wednesday	...	0.012461	0.010198	-0.000470	0.011875	0.013352	0.000704	-0.009733	0.006215	0.000388	3
32	Country_01	Local_01	Mining	2	3	Male	Contractor	Others	21	Sunday	...	0.014963	0.012962	-0.000865	0.014818	0.016816	0.000198	-0.012870	0.008151	0.000731	2

5 rows × 315 columns

Figure 96: Overview of Word2Vec - Model2 with Predicted column added

Removed Accident Level from the Merged Dataset, since already we have Predicted Accident level

	Country	City	Industry Sector	Potential Accident Level	Gender	Employee type	Critical Risk	Weekday	WeekofYear	Weekend	...	GloVe_291	GloVe_292	GloVe_293	GloVe_294	GloVe_295	GloVe_296	GloVe_297	GloVe_298	GloVe_299	Predicted
15	Country_02	Local_05	Metals	3	Male	Employee	Liquid Metal	Thursday	5	0	...	-0.024693	-0.047650	-0.015708	0.067314	0.167760	-0.035326	-0.099047	-0.047969	-0.027353	0
23	Country_02	Local_02	Mining	1	Male	Contractor (Remote)	Others	Monday	7	0	...	-0.081535	0.111993	-0.104607	0.018857	0.360266	-0.132124	-0.324510	0.047853	0.064667	1
29	Country_02	Local_07	Mining	2	Male	Employee	Others	Tuesday	7	0	...	0.002789	0.018379	-0.021721	-0.018772	0.089487	-0.133801	-0.083973	-0.334744	0.253727	1
30	Country_01	Local_03	Mining	1	Male	Employee	Others	Wednesday	7	0	...	0.091715	-0.004494	0.073564	0.102722	0.159337	0.028924	-0.168550	0.099812	0.025263	0
32	Country_01	Local_01	Mining	3	Male	Contractor	Others	Sunday	7	1	...	-0.120020	0.015738	-0.067694	0.145352	0.122889	-0.015679	-0.186358	0.062675	-0.020945	2

5 rows × 312 columns

Figure 97: Overview of Glove_df After removing Accident Level Column

6.2 Calculated target variable distribution for each DataFrame

Calculated and displayed the distribution of the "Potential Accident Level" target variable across GloVe, TF-IDF, and Word2Vec datasets. The output shows identical distributions for all datasets, with levels 3, 1, 2, 4, and 0 having 30, 19, 15, 10, and 7 occurrences, respectively.

Potential Accident Level	Glove	TF-IDF	Word2Vec
3	30	30	30
1	19	19	19
2	15	15	15
4	10	10	10
0	7	7	7

Figure 98: Target variable distribution for each DataFrame

6.3 Balancing & one-hot encoding

Balanced the "Potential Accident Level" target variable using SMOTE and one-hot encode categorical features for GloVe, TF-IDF, and Word2Vec datasets. After balancing, all target levels have an equal count of 30 in each dataset, ensuring a uniform distribution.

Potential Accident Level	Glove (Balanced)	TF-IDF (Balanced)	Word2Vec (Balanced)
3	30	30	30
1	30	30	30
2	30	30	30
4	30	30	30
0	30	30	30

Figure 99: Balanced Target variable distribution for each DataFrame

6.4 Classification Matrices

Renamed the balanced datasets and initialize multiple classifiers (e.g., Logistic Regression, SVM, Random Forest, XGBoost) to train and evaluate models on GloVe, TF-IDF, and Word2Vec data. For each classifier, training and prediction metrics (accuracy, precision, recall, F1-score) are calculated for both train and test sets, along with training and prediction times. Results are stored in separate DataFrames for each dataset.

```
Classification matrix for Glove_Model2
```

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.983333	0.983631	0.983333	0.983312	0.766667	0.804444	0.766667	0.774743	0.070529	0.003988
1	Support Vector Machine	0.458333	0.371633	0.458333	0.391387	0.366667	0.398291	0.366667	0.352941	0.008818	0.004519
2	Decision Tree	1.000000	1.000000	1.000000	1.000000	0.600000	0.882051	0.600000	0.640825	0.022488	0.003176
3	Random Forest	1.000000	1.000000	1.000000	1.000000	0.700000	0.925000	0.700000	0.721481	0.225817	0.007617
4	Gradient Boosting	1.000000	1.000000	1.000000	1.000000	0.766667	0.859722	0.766667	0.785486	7.242481	0.004839
5	XG Boost	1.000000	1.000000	1.000000	1.000000	0.766667	0.792593	0.766667	0.765531	0.927768	0.080925
6	Naive Bayes	0.908333	0.926473	0.908333	0.907675	0.633333	0.730556	0.633333	0.651717	0.005049	0.003578
7	K-Nearest Neighbors	0.733333	0.745584	0.733333	0.715410	0.633333	0.598571	0.633333	0.594478	0.003972	0.009662

Figure 100: Classification matrix for Glove_Model2

```
Classification matrix for TFIDF_Model2
```

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.991667	0.992014	0.991667	0.991658	0.766667	0.777037	0.766667	0.761562	6.767482	0.074874
1	Support Vector Machine	0.425000	0.350023	0.425000	0.365151	0.333333	0.388889	0.333333	0.338235	0.234765	0.060272
2	Decision Tree	1.000000	1.000000	1.000000	1.000000	0.633333	0.760606	0.633333	0.650476	0.086996	0.049854
3	Random Forest	1.000000	1.000000	1.000000	1.000000	0.700000	0.841111	0.700000	0.702239	0.194383	0.019672
4	Gradient Boosting	1.000000	1.000000	1.000000	1.000000	0.766667	0.806667	0.766667	0.770238	3.670400	0.016248
5	XG Boost	1.000000	1.000000	1.000000	1.000000	0.800000	0.880000	0.800000	0.803704	1.920825	0.900377
6	Naive Bayes	1.000000	1.000000	1.000000	1.000000	0.600000	0.868095	0.600000	0.640878	0.022075	0.017616
7	K-Nearest Neighbors	0.666667	0.686410	0.666667	0.652359	0.600000	0.646667	0.600000	0.574392	0.015126	0.018545

Figure 101: Classification matrix for TF-IDF_Model2

Classification matrix for Word2Vec_Model2

	Classifier	Train Accuracy	Train Precision	Train Recall	Train F1-score	Test Accuracy	Test Precision	Test Recall	Test F1-score	Training Time	Prediction Time
0	Logistic Regression	0.833333	0.830716	0.833333	0.831305	0.666667	0.668810	0.666667	0.660053	0.059910	0.003207
1	Support Vector Machine	0.441667	0.354843	0.441667	0.382172	0.333333	0.373333	0.333333	0.330159	0.007429	0.004455
2	Decision Tree	1.000000	1.000000	1.000000	1.000000	0.600000	0.606876	0.600000	0.577577	0.021864	0.003823
3	Random Forest	1.000000	1.000000	1.000000	1.000000	0.566667	0.592063	0.566667	0.566667	0.228622	0.007935
4	Gradient Boosting	1.000000	1.000000	1.000000	1.000000	0.666667	0.732937	0.666667	0.671919	8.504379	0.004985
5	XG Boost	1.000000	1.000000	1.000000	1.000000	0.666667	0.763611	0.666667	0.687177	1.058113	0.078650
6	Naive Bayes	0.658333	0.658278	0.658333	0.648193	0.400000	0.488333	0.400000	0.427273	0.004678	0.003584
7	K-Nearest Neighbors	0.666667	0.679606	0.666667	0.648785	0.633333	0.641164	0.633333	0.594644	0.003693	0.006159

Figure 102: Classification matrix for Word2Vec_Model2

6.5 Classification report for ML classifiers Model2.

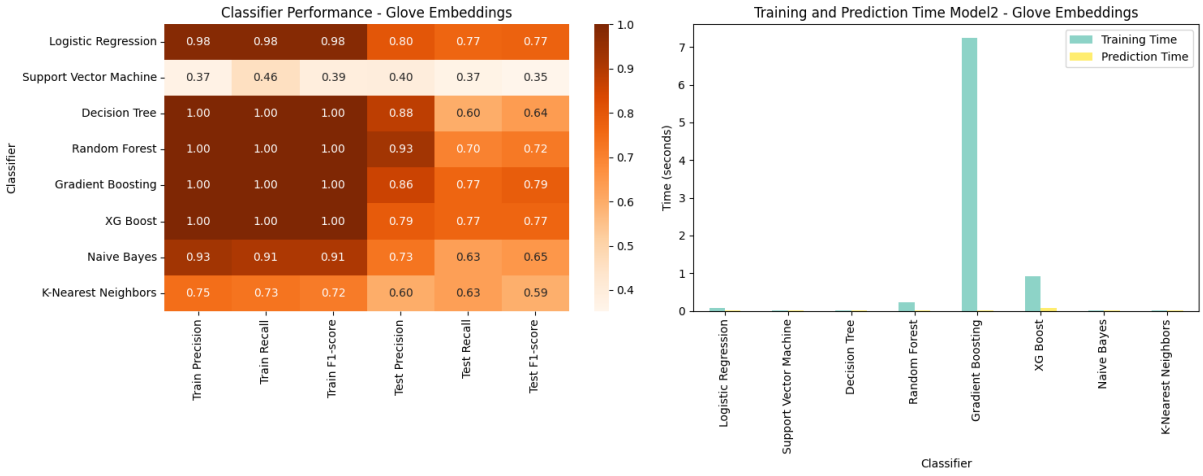

Figure 103: Classifier Performance & Training & Prediction Time Model2- Glove Embeddings

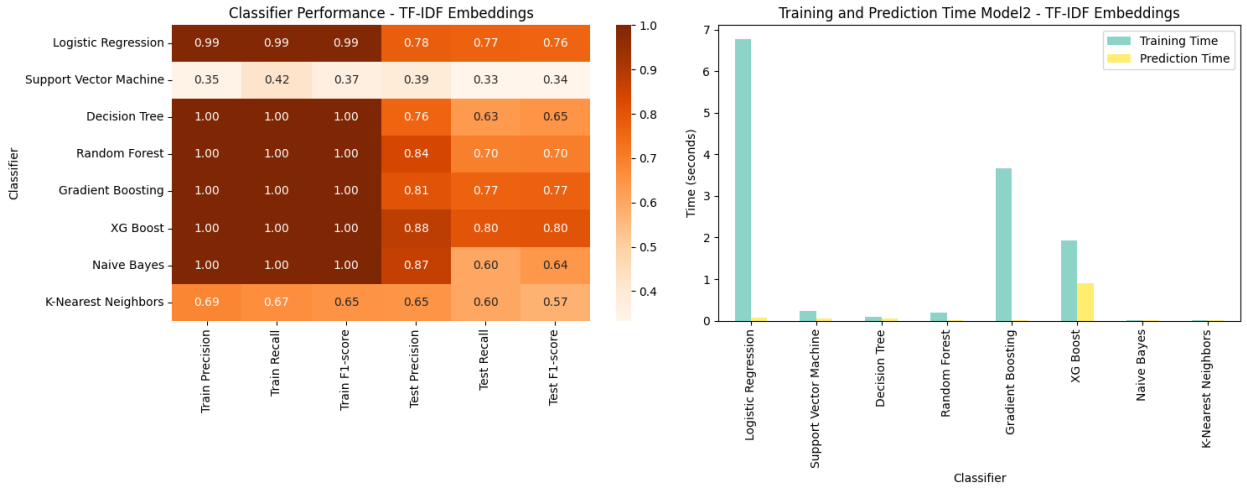

Figure 104: Classifier Performance & Training & Prediction Time Model2- TF-IDF Embeddings

Figure 105: Classifier Performance & Training & Prediction Time Model2- Word2Vec Embeddings

6.6 Model 2 Train & Test Confusion Matrices

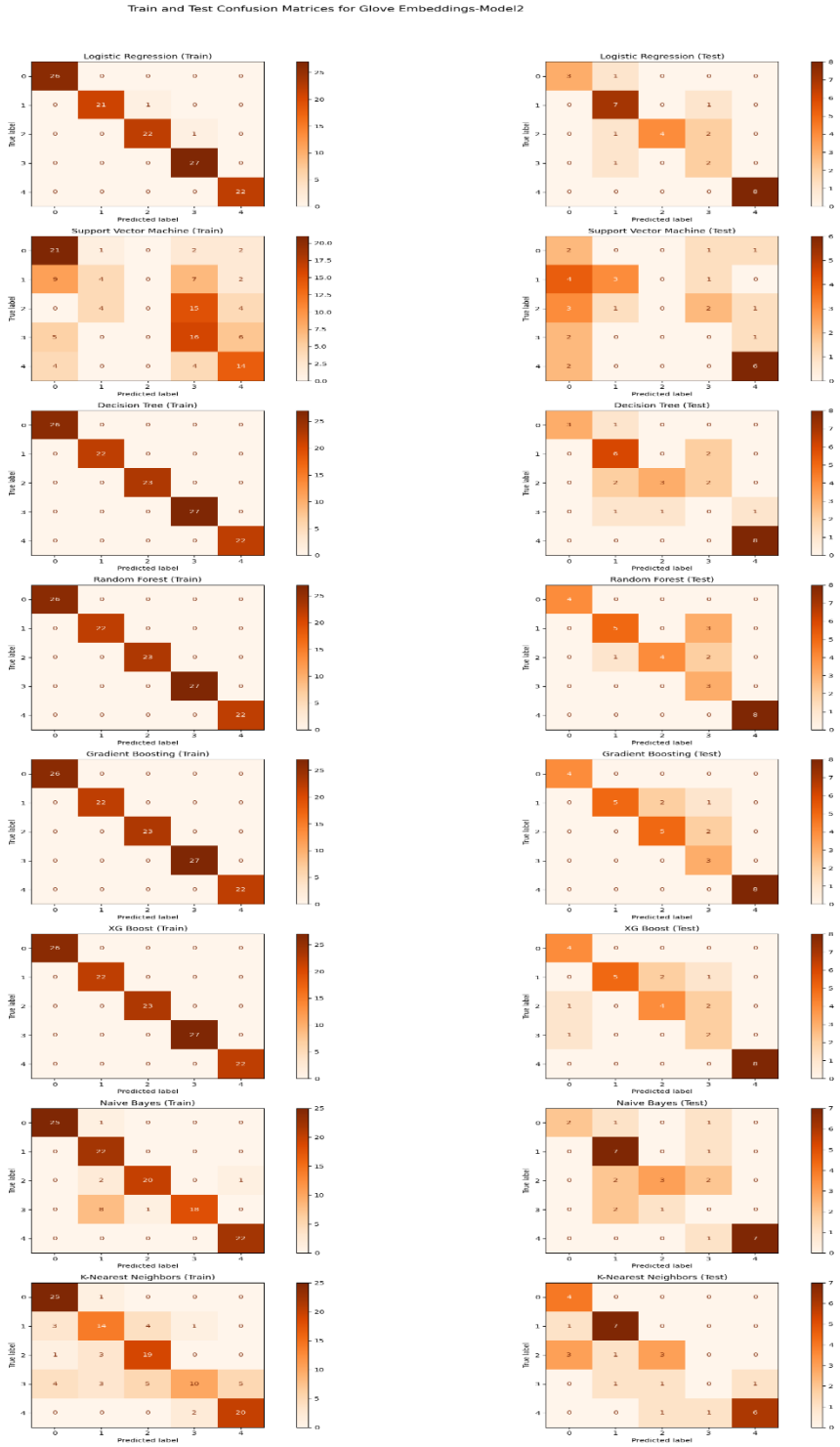

Figure 106: Train and Test Confusion Matrices for Glove Embeddings – Model 2

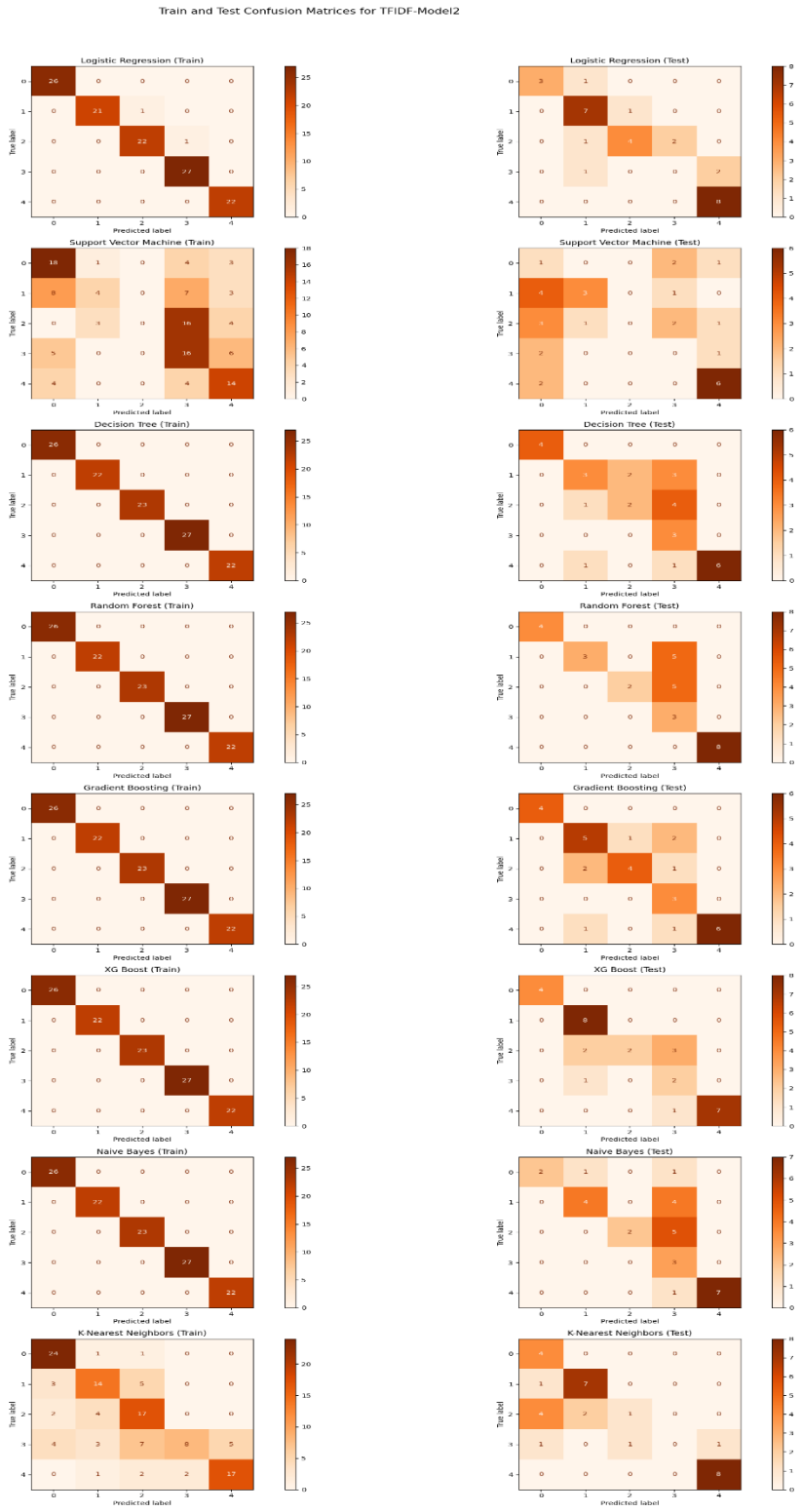

Figure 107: Train and Test Confusion Matrices for TFIDF Embeddings – Model 2

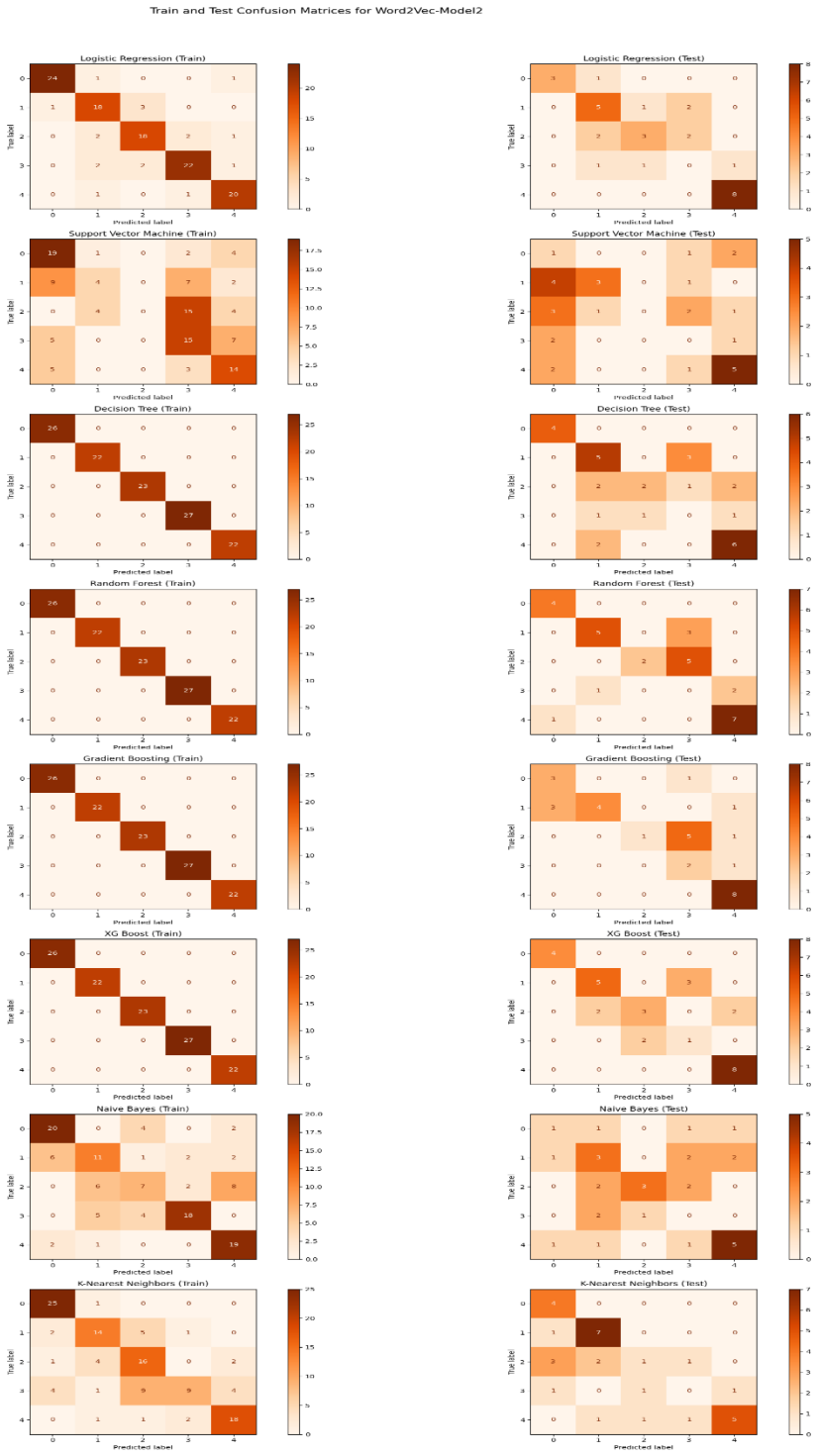

Figure 108: Train and Test Confusion Matrices for Word2Vec Embeddings – Model 2

Milestone 2

Milestone 1 findings recommend using the Glove embeddings dataset for further processing and deep learning model development.

7. Design, Train and Test Neural Networks Classifiers

After importing the required libraries, read the CSV file from Google Drive into a pandas DataFrame and displayed its first few rows for a quick preview. A copy of the ISH_NLP_Glove_df DataFrame is created as ISH_NLP_Glove_df_main. The first few rows of the new DataFrame are displayed.

The first few rows of the data frame

	WeekofYear	Weekend	GloVe_0	GloVe_1	GloVe_2	GloVe_3	GloVe_4	GloVe_5	GloVe_6	GloVe_7	...	Weekday_Monday	Weekday_Saturday	Weekday_Sunday
0	53	0	0.078223	0.040773	-0.041107	-0.293287	-0.148195	-0.085006	0.120392	-0.043692	...	0	0	0
1	53	1	-0.047137	0.109611	-0.049147	-0.199018	0.049427	-0.139335	0.039627	-0.095639	...	0	1	0
2	1	0	-0.057290	0.202640	-0.209550	-0.169683	-0.027187	-0.091942	-0.168629	-0.005628	...	0	0	0
3	1	0	-0.033755	0.019709	-0.029097	-0.216930	-0.088179	-0.137728	-0.017687	0.012178	...	0	0	0
4	1	1	-0.099598	0.082313	-0.132139	-0.090341	-0.122124	-0.055800	0.132037	0.086205	...	0	0	1

5 rows × 362 columns

Figure 109: Data frame overview

First few rows of the new data frame

	WeekofYear	Weekend	GloVe_0	GloVe_1	GloVe_2	GloVe_3	GloVe_4	GloVe_5	GloVe_6	GloVe_7	...	Weekday_Monday	Weekday_Saturday	Weekday_Sunday
0	53	0	0.078223	0.040773	-0.041107	-0.293287	-0.148195	-0.085006	0.120392	-0.043692	...	0	0	0
1	53	1	-0.047137	0.109611	-0.049147	-0.199018	0.049427	-0.139335	0.039627	-0.095639	...	0	1	0
2	1	0	-0.057290	0.202640	-0.209550	-0.169683	-0.027187	-0.091942	-0.168629	-0.005628	...	0	0	0
3	1	0	-0.033755	0.019709	-0.029097	-0.216930	-0.088179	-0.137728	-0.017687	0.012178	...	0	0	0
4	1	1	-0.099598	0.082313	-0.132139	-0.090341	-0.122124	-0.055800	0.132037	0.086205	...	0	0	1

5 rows × 362 columns

Figure 110: ISH_NLP_Glove_df_main Overview

Statistics Summary

Summary statistics:

	count	mean	std	min	25%	50%	75%	max
WeekofYear	1545.0	19.589644	13.347339	1.000000	8.000000	17.000000	27.000000	53.000000
Weekend	1545.0	0.136570	0.343504	0.000000	0.000000	0.000000	0.000000	1.000000
GloVe_0	1545.0	-0.031031	0.062240	-0.317722	-0.059310	-0.023450	0.006005	0.186513
GloVe_1	1545.0	0.073986	0.070001	-0.156011	0.032227	0.074974	0.118921	0.322451
GloVe_2	1545.0	-0.074833	0.061172	-0.316431	-0.111710	-0.073061	-0.035238	0.242731
...	...	...	...	...	...	...	...	...
Weekday_Wednesday	1545.0	0.107443	0.309776	0.000000	0.000000	0.000000	0.000000	1.000000
Season_Spring	1545.0	0.113269	0.317023	0.000000	0.000000	0.000000	0.000000	1.000000
Season_Summer	1545.0	0.220065	0.414424	0.000000	0.000000	0.000000	0.000000	1.000000
Season_Winter	1545.0	0.177994	0.382631	0.000000	0.000000	0.000000	0.000000	1.000000
Accident Level	1545.0	2.000000	1.414671	0.000000	1.000000	2.000000	3.000000	4.000000

362 rows × 8 columns

Figure 111: Summary of the Statistics

7.1 Data Preprocessing

1. Feature Scaling: Neural networks perform better when input features are on a similar scale. The GloVe features are likely to have different ranges, so scaling them will ensure that all features contribute equally to the model's learning process.

Preprocessing step: Apply standardization (mean=0, std=1) or normalization (min-max scaling) to all GloVe features.

2. Encode the Target Variable: The "Accident Level" column is of type int64, which suggests it's a categorical variable representing different levels of accidents.

Preprocessing step: Depending on the number of unique accident levels, applying one-hot encoding or use it as is if it's in a numeric format.

3. Split the Dataset: To properly evaluate the model's performance, we need separate training and testing sets.

Preprocessing step: Splitting the data into training and testing sets (e.g., 80% training, 20% testing).

4. Dimensionality Reduction (Optional): With 300 GloVe features, there might be some redundancy or noise in the data. Reducing dimensionality could improve model performance and reduce computational costs.

Preprocessing step: Applying techniques like Principal Component Analysis (PCA) reduces the number of features while retaining most of the variance in the data.

5. Data Augmentation (Optional): If the dataset is relatively small (1545 samples might be considered small for some complex neural network architectures), data augmentation techniques could help increase the effective size of the training set.

Preprocessing step: Explore domain-specific data augmentation techniques if applicable.

The data is prepared for a neural network by scaling features and encoding the target variable into one-hot format after splitting it into training and testing sets. Feature scaling is applied using StandardScaler, and labels are encoded with LabelEncoder.

```
Shape of X_train: (1236, 361)
Shape of X_test: (309, 361)
Shape of y_train: (1236,)
Shape of y_test: (309,)


Shape of X_train_scaled: (1236, 361)
Shape of X_test_scaled: (309, 361)
Shape of y_train_onehot: (1236, 5)
Shape of y_test_onehot: (309, 5)
```

Figure 112: Shape of the Training and Testing sets

7.2 Base NN Classifier

Defined a function, build_base_nn_model, to create a neural network with three dense layers using ReLU and softmax activations. It incorporates a dictionary of optimizers (SGD, RMSprop, Adam, Nadam, AdamW), validates the selected optimizer, and compiles the model with categorical cross-entropy loss and accuracy metrics. Using the function, models are initialized for

different optimizers based on the input shape (GloVe embeddings) and number of output classes (one-hot encoded labels). Finally, the architecture of each model is displayed to review its configuration.

Model with SGD optimizer:
Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	46,336
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 5)	325

Total params: 54,917 (214.52 KB)
Trainable params: 54,917 (214.52 KB)
Non-trainable params: 0 (0.00 B)

Figure 113: Model with SGD optimizer

Model with RMSprop optimizer:
Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 128)	46,336
dense_4 (Dense)	(None, 64)	8,256
dense_5 (Dense)	(None, 5)	325

Total params: 54,917 (214.52 KB)
Trainable params: 54,917 (214.52 KB)
Non-trainable params: 0 (0.00 B)

Figure 114: Model with RMSprop optimizer

Model with Adam optimizer:
Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 128)	46,336
dense_7 (Dense)	(None, 64)	8,256
dense_8 (Dense)	(None, 5)	325

Total params: 54,917 (214.52 KB)
Trainable params: 54,917 (214.52 KB)
Non-trainable params: 0 (0.00 B)

Figure 115: Model with Adam optimizer

Model with Nadam optimizer:
Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 128)	46,336
dense_10 (Dense)	(None, 64)	8,256
dense_11 (Dense)	(None, 5)	325

Total params: 54,917 (214.52 KB)
Trainable params: 54,917 (214.52 KB)
Non-trainable params: 0 (0.00 B)

Figure 116: Model with Nadam optimizer

Model with AdamW optimizer:
Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 128)	46,336
dense_13 (Dense)	(None, 64)	8,256
dense_14 (Dense)	(None, 5)	325

Total params: 54,917 (214.52 KB)
Trainable params: 54,917 (214.52 KB)
Non-trainable params: 0 (0.00 B)

Figure 117: Model with AdamW optimizer

Then trained multiple neural network models with different optimizers using the training dataset and evaluates them on the test dataset to record their performance. It saves the training history and prints the test loss and accuracy for each optimizer.

```
Training model with SGD optimizer...
Test Loss (SGD): 0.0907
Test Accuracy (SGD): 0.9644
Training model with RMSprop optimizer...
Test Loss (RMSprop): 0.1713
Test Accuracy (RMSprop): 0.9709
Training model with Adam optimizer...
Test Loss (Adam): 0.1162
Test Accuracy (Adam): 0.9612
Training model with Nadam optimizer...
Test Loss (Nadam): 0.1267
Test Accuracy (Nadam): 0.9612
Training model with AdamW optimizer...
Test Loss (AdamW): 0.1164
Test Accuracy (AdamW): 0.9547
Training and evaluation for Base NN complete.
```

Figure 118: Training and Evaluation for Base NN

7.3 Train vs Validation plots for Accuracy and Loss for Base NN Classifier

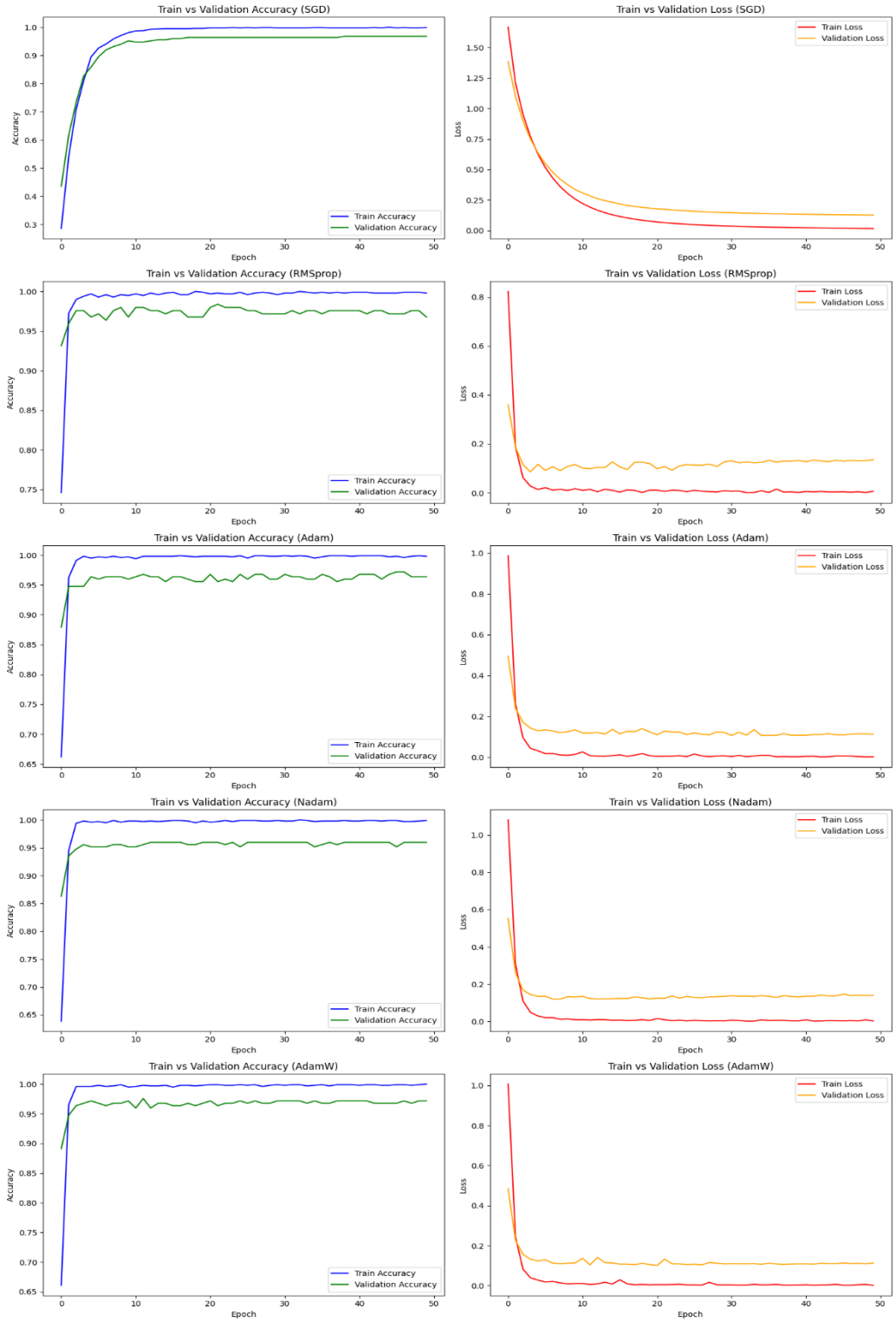

Figure 119: Train vs Validation plots for Accuracy & Loss of Base NN Classifier

Observations:

Accuracy Across Optimizers:

- The RMSprop optimizer achieved the highest test accuracy at 97.09%.
- The SGD optimizer was slightly behind, with a test accuracy of 96.44%.
- The Adam and Nadam optimizers had the same test accuracy of 96.12%.
- The AdamW optimizer showed the lowest test accuracy among the tested optimizers at 95.47%.

Loss Across Optimizers:

- The SGD optimizer had the lowest test loss at 0.0907, indicating better generalization performance in terms of minimizing errors.
- The Adam optimizer had a similar test loss to AdamW, but slightly better accuracy.
- The RMSprop optimizer had a relatively higher test loss (0.1713) despite achieving the highest accuracy.

Consistency:

- Optimizers like Adam and Nadam performed similarly in both accuracy and loss, suggesting consistent results across these two variants.

Base NN Training:

- The results indicate that the network's performance varies depending on the optimizer, highlighting the importance of optimizer selection in model training.

Insights

1. **RMSprop's Higher Accuracy Despite Loss:** The RMSprop optimizer achieved the highest accuracy but with a higher loss compared to SGD. This may indicate that RMSprop focuses on improving classification accuracy but does not minimize the error as effectively as SGD.

2. **SGD's Generalization Capability:** The SGD optimizer showed the lowest test loss, suggesting it may generalize better in this setup, though its accuracy is slightly lower than RMSprop.
3. **Trade-offs in Optimizer Selection:** While RMSprop delivered the highest accuracy, its higher loss could be a concern depending on the application's sensitivity to errors. SGD may be a better choice if minimizing test loss is a priority.
4. **Adam and Nadam Optimizer Similarity:** The similarity in results between Adam and Nadam indicates that adding the Nesterov momentum to Adam (Nadam) did not provide significant improvement in this case.
5. **AdamW Performance:** Despite being a variant of Adam with weight decay for better regularization, AdamW underperformed compared to other optimizers in both accuracy and loss, suggesting it might not be ideal for this model's architecture or data.

7.4 Classification Reports for Base NN Classifier

We had predicted class labels for train and test data using each optimizer's model and generates classification reports to evaluate performance. It formats the reports as DataFrames for better visualization, displaying train and test metrics side by side for each optimizer.

Classification Report for SGD optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.992278	0.980916	0.986564	262.000000	0.877551	0.914894	0.895833	47.000000
1	0.987395	0.995763	0.991561	236.000000	0.972222	0.958904	0.965517	73.000000
2	0.996047	0.996047	0.996047	253.000000	0.964286	0.964286	0.964286	56.000000
3	0.987705	0.991770	0.989733	243.000000	0.984615	0.969697	0.977099	66.000000
4	1.000000	1.000000	1.000000	242.000000	1.000000	1.000000	1.000000	67.000000
accuracy	0.992718	0.992718	0.992718	0.992718	0.964401	0.964401	0.964401	0.964401
macro avg	0.992685	0.992899	0.992781	1236.000000	0.959735	0.961556	0.960547	309.000000
weighted avg	0.992730	0.992718	0.992713	1236.000000	0.965054	0.964401	0.964646	309.000000

Figure 120: Classification Report for SGD optimizer

Classification Report for RMSprop optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.984733	0.984733	0.984733	262.000000	0.933333	0.893617	0.913043	47.000000
1	0.987395	0.995763	0.991561	236.000000	0.972973	0.986301	0.979592	73.000000
2	0.996047	0.996047	0.996047	253.000000	0.964912	0.982143	0.973451	56.000000
3	0.995851	0.987654	0.991736	243.000000	0.984615	0.969697	0.977099	66.000000
4	1.000000	1.000000	1.000000	242.000000	0.985294	1.000000	0.992593	67.000000
accuracy	0.992718	0.992718	0.992718	0.992718	0.970874	0.970874	0.970874	0.970874
macro avg	0.992805	0.992839	0.992815	1236.000000	0.968226	0.966352	0.967156	309.000000
weighted avg	0.992732	0.992718	0.992719	1236.000000	0.970641	0.970874	0.970643	309.000000

Figure 121: Classification Report for RMSprop optimizer

Classification Report for Adam optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.984674	0.980916	0.982792	262.000000	0.875000	0.893617	0.884211	47.000000
1	0.991525	0.991525	0.991525	236.000000	0.972603	0.972603	0.972603	73.000000
2	1.000000	0.996047	0.998020	253.000000	0.964912	0.982143	0.973451	56.000000
3	0.987755	0.995885	0.991803	243.000000	0.968750	0.939394	0.953846	66.000000
4	1.000000	1.000000	1.000000	242.000000	1.000000	1.000000	1.000000	67.000000
accuracy	0.992718	0.992718	0.992718	0.992718	0.961165	0.961165	0.961165	0.961165
macro avg	0.992791	0.992875	0.992828	1236.000000	0.956253	0.957551	0.956822	309.000000
weighted avg	0.992726	0.992718	0.992717	1236.000000	0.961481	0.961165	0.961246	309.000000

Figure 122: Classification Report for Adam optimizer

Classification Report for Nadam optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.980916	0.980916	0.980916	262.0000	0.875000	0.893617	0.884211	47.000000
1	0.991489	0.987288	0.989384	236.0000	0.986111	0.972603	0.979310	73.000000
2	0.996047	0.996047	0.996047	253.0000	0.982143	0.982143	0.982143	56.000000
3	0.991770	0.991770	0.991770	243.0000	0.953846	0.939394	0.946565	66.000000
4	0.995885	1.000000	0.997938	242.0000	0.985294	1.000000	0.992593	67.000000
accuracy	0.991100	0.991100	0.991100	0.9911	0.961165	0.961165	0.961165	0.961165
macro avg	0.991221	0.991204	0.991211	1236.0000	0.956479	0.957551	0.956964	309.000000
weighted avg	0.991097	0.991100	0.991097	1236.0000	0.961423	0.961165	0.961244	309.000000

Figure 123: Classification Report for Nadam optimizer

Classification Report for AdamW optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.992278	0.980916	0.986564	262.000000	0.886364	0.829787	0.857143	47.000000
1	0.987448	1.000000	0.993684	236.000000	0.972603	0.972603	0.972603	73.000000
2	0.996047	0.996047	0.996047	253.000000	0.964912	0.982143	0.973451	56.000000
3	0.991770	0.991770	0.991770	243.000000	0.940299	0.954545	0.947368	66.000000
4	1.000000	1.000000	1.000000	242.000000	0.985294	1.000000	0.992593	67.000000
accuracy	0.993528	0.993528	0.993528	0.993528	0.954693	0.954693	0.954693	0.954693
macro avg	0.993509	0.993747	0.993613	1236.000000	0.949894	0.947816	0.948632	309.000000
weighted avg	0.993539	0.993528	0.993519	1236.000000	0.953944	0.954693	0.954139	309.000000

Figure 124: Classification Report for AdamW optimizer

Observations:

1. **High Training Metrics Across Optimizers:** All optimizers demonstrate high performance on the training data across all classes (0-4), with precision, recall, and F1-scores frequently at or near 100%. This suggests that the model fits the training data extremely well.

2. **Performance on Test Data:** The test data metrics are generally lower than the training data, which is expected due to generalization challenges, but still are quite high, showing that the models generalize well though not perfectly.

Specific Insights:

1. SGD:

- **Performance:** High training and test performance across all classes with particularly strong results in class 4.
- **Test F1-Scores:** These are slightly lower compared to training scores, particularly in class 0 and 1 where there is a noticeable drop. This may indicate some overfitting.

2. RMSprop:

- **Test Performance:** Similar to SGD, with a minor drop in precision and recall for class 1 in the test set, which suggests that this class might be a bit more challenging to generalize by this optimizer.

3. Adam:

- **Consistency:** Shows slightly more consistent F1-scores between training and testing than SGD, suggesting better generalization for certain classes.
- **Test Class 4:** Notable for achieving 100% across all metrics, indicating exceptional performance on this class.

4. Nadam:

- **Balanced Performance:** Offers good balance with slightly higher test metrics in some classes compared to Adam, especially noticeable in class 0 for test precision and recall.
- **Slight Overfitting:** As with others, there's a gap between train and test scores, albeit small.

5. AdamW:

- Overall Test Scores: Among the highest, suggesting that this optimizer may provide the best generalization among those tested.
- Stability: Shows less variation between training and test metrics, particularly in class 4 where it matches or exceeds other optimizers.

Recommendations:

1. **Further Investigation:** For classes with a significant drop between training and testing (like class 1), it might be beneficial to look into specific features or additional data that can improve model robustness.

2. **Optimizer Choice:** AdamW seems to provide the best generalization based on this data. Consider using it for deployment if consistent performance across multiple classes is critical.

3. **Regularization and Tuning:** Implement or increase regularization techniques to mitigate overfitting observed particularly in SGD and RMSprop optimizers. Also, tuning hyperparameters specifically for the underperforming classes could yield better results.

7.5 Train and Test Confusion Matrices for Base NN Classifier

Evaluated the performance of neural network models using different optimizers by generating and visualizing confusion matrices for both training and testing data. It predicts class labels, computes confusion matrices, and displays them as heatmaps using Seaborn for easy interpretation of model performance.

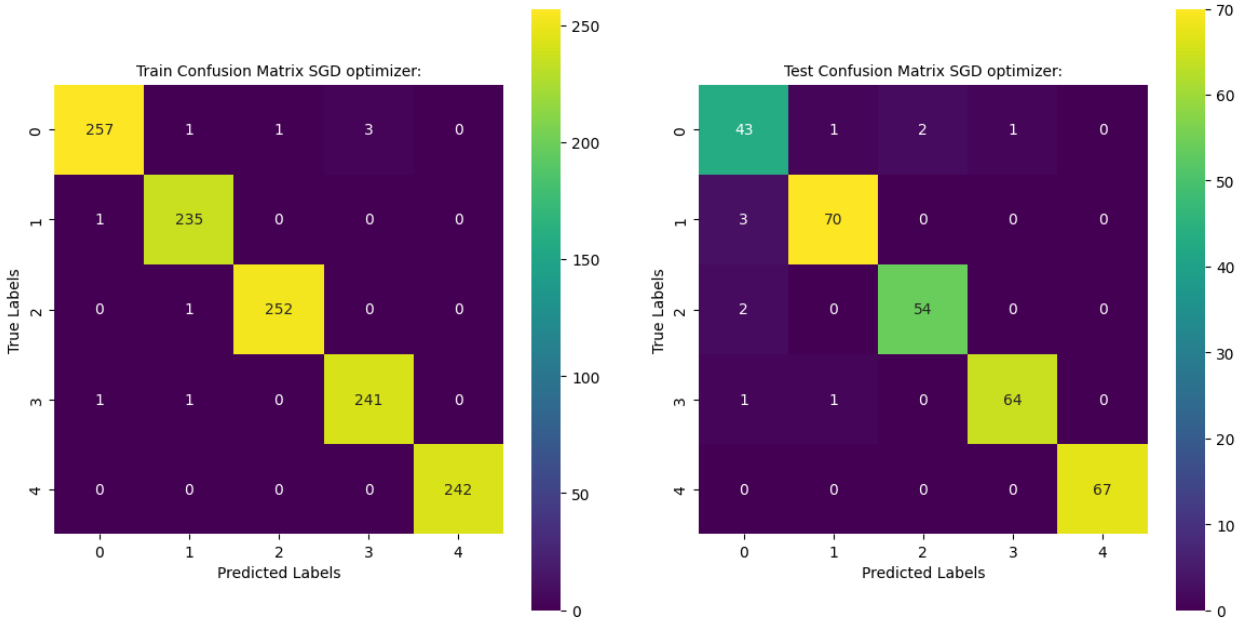

Figure 125: Confusion Matrices for Base NN with SGD optimizer

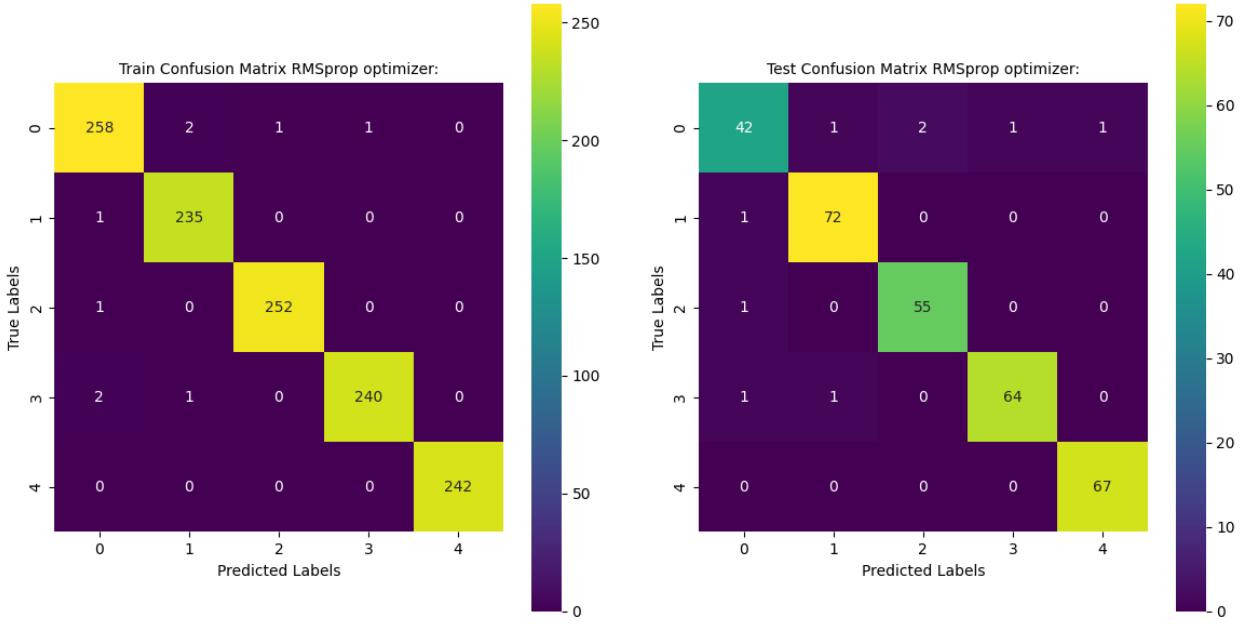

Figure 126: Confusion Matrices for Base NN with RMSprop optimizer

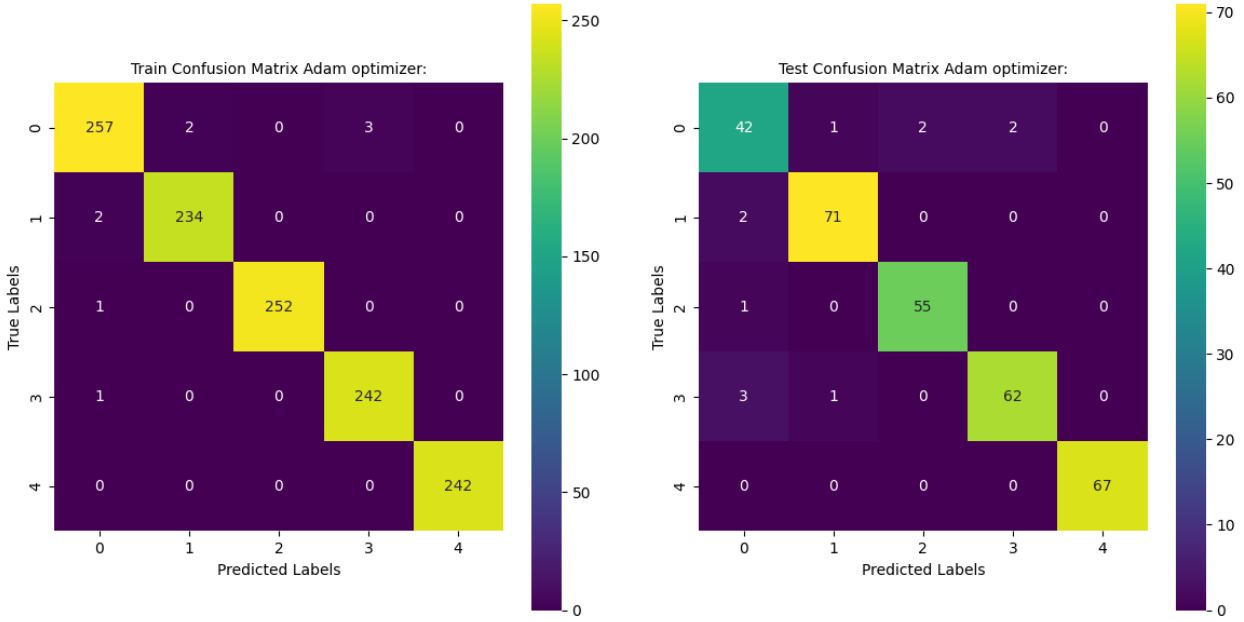

Figure 127: Confusion Matrices for Base NN with Adam optimizer

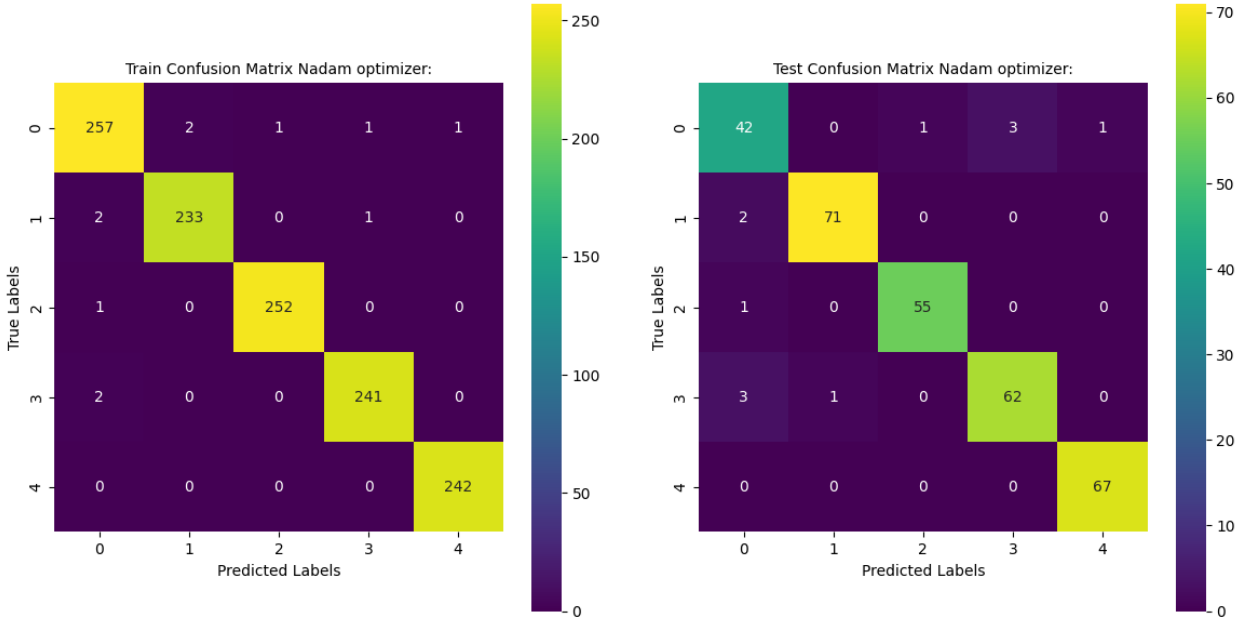

Figure 128: Confusion Matrices for Base NN with Nadam optimizer

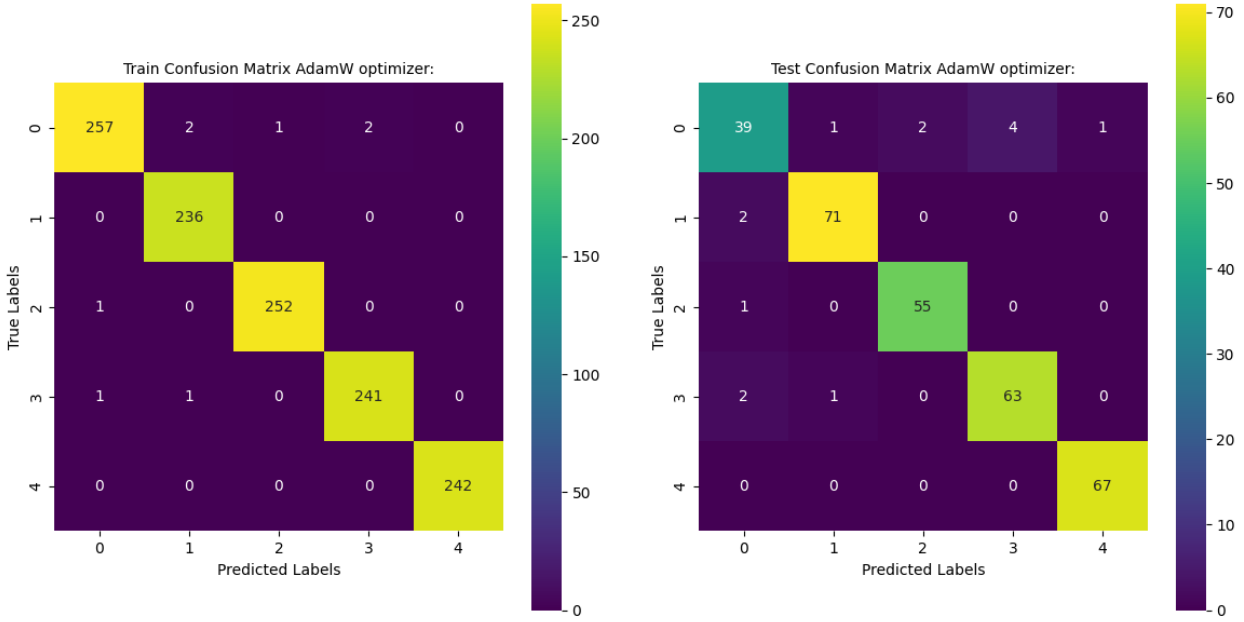

Figure 129: Confusion Matrices for Base NN with AdamW optimizer

Observations:

1. **Training Performance:**

- Most optimizers perform well on the training data with high diagonal values, indicating good classification for each label.
- Minor misclassifications are observed, notably a few instances of class 0 being predicted as class 3 across different optimizers.

2. **Testing Performance:**

- Testing performance slightly decreases, which is typical due to the model facing unseen data.
- The decrease in performance is not drastic, which indicates good generalization for most optimizers.

3. **Specific Insights:**

SGD:

- Misclassifications are slightly higher in testing, especially between classes 0 and 3.
- Class 1 (true label) shows strong accuracy with 71 out of 73 correctly predicted on the test set.

RMSprop:

- Shows increased misclassification between class 0 and other classes in the training set.
- Test set performance for class 1 is consistent with the training, showing reliable performance.

Adam:

- Similar patterns in both training and testing sets, with slight misclassifications mainly in classes near to each other (e.g., 1 mispredicted as 0).
- This optimizer shows relatively balanced performance across all classes.

Nadam:

- Slightly better at handling class 4 misclassifications compared to others.
- Noticeably more misclassifications between class 0 and 3 in the test set compared to training.

AdamW:

- Shows some robustness in dealing with class 3 and 4 but has slight confusion in class 2 predictions.
- Test results are similar to training, indicating good consistency.

Conclusion:

Specific Class Performance:

- For Class 1, optimizers like SGD, Adam, and Nadam consistently show high accuracy on the testing dataset.

Misclassification Patterns:

- Certain patterns like misclassifications between classes 0 and 3 are more prevalent in some optimizers (Nadam, Adam) than others.

Balancing Decision:

- **Consistency between Training and Testing:** AdamW shows good consistency between training and testing.
- **Overall Accuracy and Stability:** Adam and Nadam also display strong and stable performance, but AdamW seems slightly better in terms of maintaining performance from training to testing.

7.6 Hyper tuned NN classifier

Defined a function to build a hypertuned neural network model with layers including Batch Normalization, Dropout, and L2 regularization to improve generalization. Models are initialized with different optimizers (SGD, RMSprop, Adam, Nadam, AdamW), compiled with categorical cross-entropy loss, and stored for further training and evaluation.

Hypertuned NN Model with SGD optimizer:
Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 256)	92,672
batch_normalization (BatchNormalization)	(None, 256)	1,024
dropout (Dropout)	(None, 256)	0
dense_16 (Dense)	(None, 128)	32,896
batch_normalization_1 (BatchNormalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_17 (Dense)	(None, 64)	8,256
batch_normalization_2 (BatchNormalization)	(None, 64)	256
dropout_2 (Dropout)	(None, 64)	0
dense_18 (Dense)	(None, 32)	2,080
batch_normalization_3 (BatchNormalization)	(None, 32)	128
dropout_3 (Dropout)	(None, 32)	0
dense_19 (Dense)	(None, 5)	165

Total params: 137,989 (539.02 KB)
Trainable params: 137,029 (535.27 KB)
Non-trainable params: 960 (3.75 KB)

Figure 130: Hypertuned NN Model with SGD optimizer

Hypertuned NN Model with RMSprop optimizer:
Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_20 (Dense)	(None, 256)	92,672
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
dropout_4 (Dropout)	(None, 256)	0
dense_21 (Dense)	(None, 128)	32,896
batch_normalization_5 (BatchNormalization)	(None, 128)	512
dropout_5 (Dropout)	(None, 128)	0
dense_22 (Dense)	(None, 64)	8,256
batch_normalization_6 (BatchNormalization)	(None, 64)	256
dropout_6 (Dropout)	(None, 64)	0
dense_23 (Dense)	(None, 32)	2,080
batch_normalization_7 (BatchNormalization)	(None, 32)	128
dropout_7 (Dropout)	(None, 32)	0
dense_24 (Dense)	(None, 5)	165

Total params: 137,989 (539.02 KB)
Trainable params: 137,029 (535.27 KB)
Non-trainable params: 960 (3.75 KB)

Figure 131: Hypertuned NN Model with RMSprop optimizer

Hypertuned NN Model with Adam optimizer:
Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_25 (Dense)	(None, 256)	92,672
batch_normalization_8 (BatchNormalization)	(None, 256)	1,024
dropout_8 (Dropout)	(None, 256)	0
dense_26 (Dense)	(None, 128)	32,896
batch_normalization_9 (BatchNormalization)	(None, 128)	512
dropout_9 (Dropout)	(None, 128)	0
dense_27 (Dense)	(None, 64)	8,256
batch_normalization_10 (BatchNormalization)	(None, 64)	256
dropout_10 (Dropout)	(None, 64)	0
dense_28 (Dense)	(None, 32)	2,080
batch_normalization_11 (BatchNormalization)	(None, 32)	128
dropout_11 (Dropout)	(None, 32)	0
dense_29 (Dense)	(None, 5)	165

Total params: 137,989 (539.02 KB)
Trainable params: 137,029 (535.27 KB)
Non-trainable params: 960 (3.75 KB)

Figure 132: Hypertuned NN Model with Adam optimizer

Hypertuned NN Model with Nadam optimizer:
Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 256)	92,672
batch_normalization_12 (BatchNormalization)	(None, 256)	1,024
dropout_12 (Dropout)	(None, 256)	0
dense_31 (Dense)	(None, 128)	32,896
batch_normalization_13 (BatchNormalization)	(None, 128)	512
dropout_13 (Dropout)	(None, 128)	0
dense_32 (Dense)	(None, 64)	8,256
batch_normalization_14 (BatchNormalization)	(None, 64)	256
dropout_14 (Dropout)	(None, 64)	0
dense_33 (Dense)	(None, 32)	2,080
batch_normalization_15 (BatchNormalization)	(None, 32)	128
dropout_15 (Dropout)	(None, 32)	0
dense_34 (Dense)	(None, 5)	165

Total params: 137,989 (539.02 KB)
Trainable params: 137,029 (535.27 KB)
Non-trainable params: 960 (3.75 KB)

Figure 133: Hypertuned NN Model with Nadam optimizer

Hypertuned NN Model with AdamW optimizer:
Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_35 (Dense)	(None, 256)	92,672
batch_normalization_16 (BatchNormalization)	(None, 256)	1,024
dropout_16 (Dropout)	(None, 256)	0
dense_36 (Dense)	(None, 128)	32,896
batch_normalization_17 (BatchNormalization)	(None, 128)	512
dropout_17 (Dropout)	(None, 128)	0
dense_37 (Dense)	(None, 64)	8,256
batch_normalization_18 (BatchNormalization)	(None, 64)	256
dropout_18 (Dropout)	(None, 64)	0
dense_38 (Dense)	(None, 32)	2,080
batch_normalization_19 (BatchNormalization)	(None, 32)	128
dropout_19 (Dropout)	(None, 32)	0
dense_39 (Dense)	(None, 5)	165

Total params: 137,989 (539.02 KB)
Trainable params: 137,029 (535.27 KB)
Non-trainable params: 960 (3.75 KB)

Figure 134: Hypertuned NN Model with AdamW optimizer

Trained and evaluated hypertuned neural network models using 5-fold cross-validation with early stopping to prevent overfitting. It records training histories for each fold, evaluates models on the test set, and prints metrics such as validation loss, accuracy, and the epoch at which early stopping occurred.

```
Training Hypertuned NN Classifier model with SGD...
Training fold 1...
Fold 1 training complete.
Training fold 2...
Fold 2 training complete.
Training fold 3...
Fold 3 training complete.
Training fold 4...
Fold 4 training complete.
Training fold 5...
Fold 5 training complete.
Test Loss (Hypertuned - SGD): 0.5150
Test Accuracy (Hypertuned - SGD): 0.9644
Fold 1: Best Validation Loss: 0.6325, Best Validation Accuracy: 0.9839, Early Stopping Epoch: 69
Fold 2: Best Validation Loss: 0.5566, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 24
Fold 3: Best Validation Loss: 0.4858, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 99
Fold 4: Best Validation Loss: 0.4623, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 47
Fold 5: Best Validation Loss: 0.4198, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 99
```

Figure 135: Training Hypertuned NN Classifier model with SGD

```
Training Hypertuned NN Classifier model with RMSprop...
Training fold 1...
Fold 1 training complete.
Training fold 2...
Fold 2 training complete.
Training fold 3...
Fold 3 training complete.
Training fold 4...
Fold 4 training complete.
Training fold 5...
Fold 5 training complete.
Test Loss (Hypertuned - RMSprop): 0.2001
Test Accuracy (Hypertuned - RMSprop): 0.9773
Fold 1: Best Validation Loss: 0.2305, Best Validation Accuracy: 0.9758, Early Stopping Epoch: 64
Fold 2: Best Validation Loss: 0.1597, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 25
Fold 3: Best Validation Loss: 0.1063, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 28
Fold 4: Best Validation Loss: 0.1032, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 10
Fold 5: Best Validation Loss: 0.1061, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 20
```

Figure 136: Training Hypertuned NN Classifier model with RMSprop

```
Training Hypertuned NN Classifier model with Adam...
Training fold 1...
Fold 1 training complete.
Training fold 2...
Fold 2 training complete.
Training fold 3...
Fold 3 training complete.
Training fold 4...
Fold 4 training complete.
Training fold 5...
Fold 5 training complete.
Test Loss (Hypertuned - Adam): 0.3666
Test Accuracy (Hypertuned - Adam): 0.9547
Fold 1: Best Validation Loss: 0.5241, Best Validation Accuracy: 0.9677, Early Stopping Epoch: 51
Fold 2: Best Validation Loss: 0.3636, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 26
Fold 3: Best Validation Loss: 0.2123, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 46
Fold 4: Best Validation Loss: 0.1622, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 43
Fold 5: Best Validation Loss: 0.1700, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 13
```

Figure 137: Training Hypertuned NN Classifier model with Adam

```
Training Hypertuned NN Classifier model with Nadam...
Training fold 1...
Fold 1 training complete.
Training fold 2...
Fold 2 training complete.
Training fold 3...
Fold 3 training complete.
Training fold 4...
Fold 4 training complete.
Training fold 5...
Fold 5 training complete.
Test Loss (Hypertuned - Nadam): 0.2501
Test Accuracy (Hypertuned - Nadam): 0.9709
Fold 1: Best Validation Loss: 0.4392, Best Validation Accuracy: 0.9718, Early Stopping Epoch: 64
Fold 2: Best Validation Loss: 0.2839, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 25
Fold 3: Best Validation Loss: 0.2068, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 36
Fold 4: Best Validation Loss: 0.1694, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 32
Fold 5: Best Validation Loss: 0.1700, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 10
```

Figure 138: Training Hypertuned NN Classifier model with Nadam

```
Training Hypertuned NN Classifier model with AdamW...
Training fold 1...
Fold 1 training complete.
Training fold 2...
Fold 2 training complete.
Training fold 3...
Fold 3 training complete.
Training fold 4...
Fold 4 training complete.
Training fold 5...
Fold 5 training complete.
Test Loss (Hypertuned - AdamW): 0.2210
Test Accuracy (Hypertuned - AdamW): 0.9773
Fold 1: Best Validation Loss: 0.4668, Best Validation Accuracy: 0.9758, Early Stopping Epoch: 58
Fold 2: Best Validation Loss: 0.3059, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 28
Fold 3: Best Validation Loss: 0.1961, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 40
Fold 4: Best Validation Loss: 0.1715, Best Validation Accuracy: 1.0000, Early Stopping Epoch: 27
Fold 5: Best Validation Loss: 0.1688, Best Validation Accuracy: 0.9960, Early Stopping Epoch: 15
Training and evaluation of Hypertuned NN Classifier model with cross-validation complete.
```

Figure 139: Training Hypertuned NN Classifier model with AdamW

7.7 Displaying Average Train vs Validation accuracy & loss for Hypertuned NN Classifier

Computed and stores the average training and validation accuracy and loss across all folds for each optimizer in a dictionary. The results are then organized into a pandas DataFrame, providing a summary comparison of the performance metrics for the hypertuned models.

	Avg_Train_Accuracy	Avg_Val_Accuracy	Avg_Train_Loss	Avg_Val_Loss
SGD	98.071551	98.767230	0.566529	0.539812
RMSprop	97.970452	97.085000	0.262341	0.304658
Adam	97.983219	97.674442	0.365288	0.378328
Nadam	97.942916	97.165602	0.362498	0.391174
AdamW	97.961864	97.631997	0.364066	0.376799

Figure 140: Average Train vs Validation Accuracy & Loss for Hypertuned NN Classifier

7.8 Train vs Validation plots for Accuracy and Loss for Hypertuned NN Classifier for all optimizers

Generated subplots to visualize training and validation accuracy and loss for each optimizer across epochs using the first fold's history.

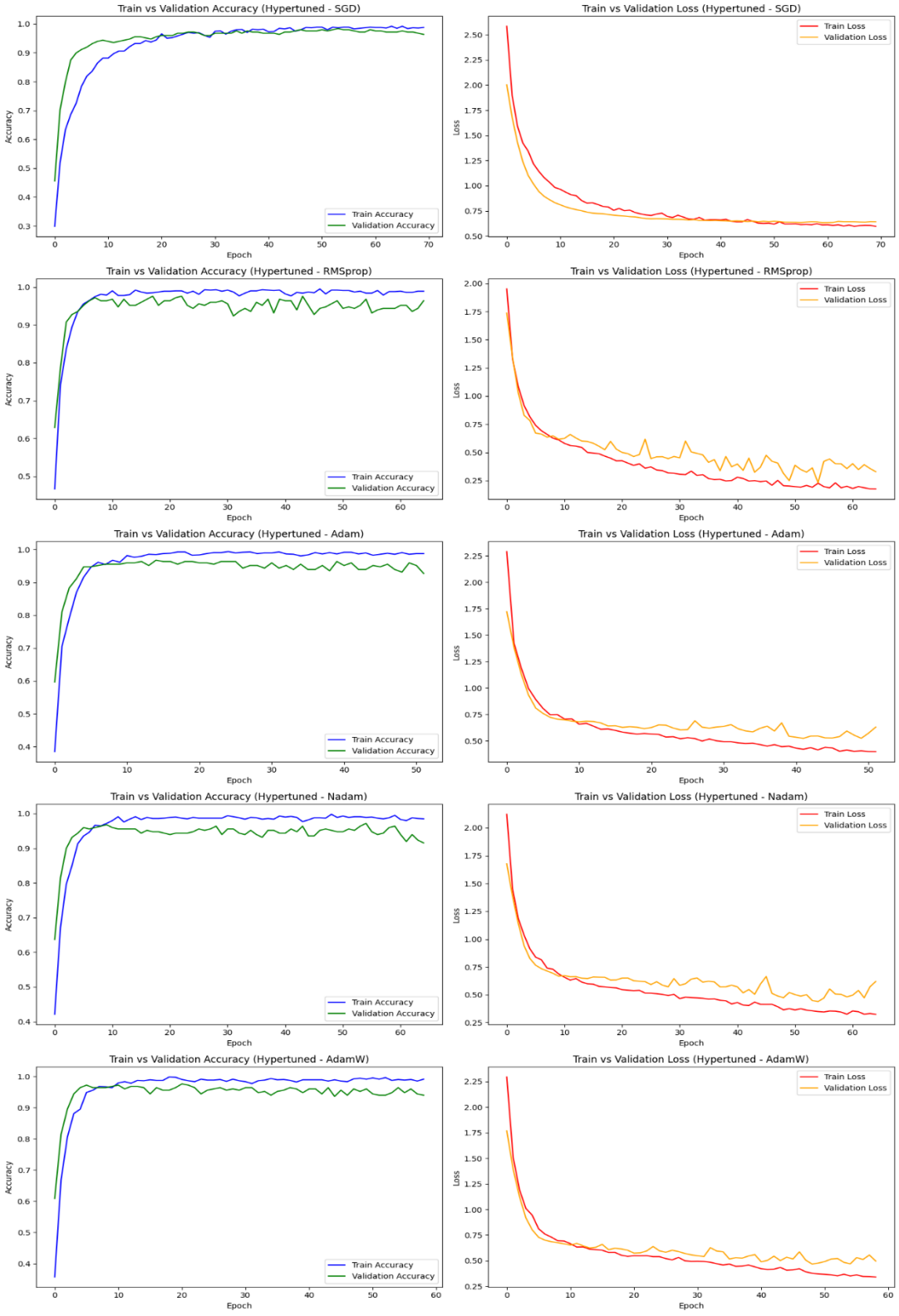

Figure 141: Train vs Validation plots for Accuracy and Loss for Hypertuned NN Classifier

Observations:

1. SGD (Stochastic Gradient Descent)

- **Accuracy:** The training and validation accuracy curves converge closely, indicating good generalization.
- **Loss:** Both training and validation loss decrease sharply and stabilize quickly, showing that SGD is effective and efficient in optimizing the loss function.

2. RMSprop

- **Accuracy:** There's a noticeable gap between training and validation accuracy, suggesting some overfitting. However, the validation accuracy does improve steadily, which is a good sign.
- **Loss:** The training and validation loss curves decrease quickly. The small gap between them suggests some level of overfitting, though less severe compared to the other Adam-based optimizers.

3. Adam

- **Accuracy:** The accuracy curves for Adam show that while there's an improvement in validation accuracy, the gap between training and validation accuracy is significant, indicating overfitting.
- **Loss:** The loss curves converge well initially but start to diverge slightly, which again points to overfitting as training progresses.

4. Nadam

- **Accuracy:** Similar to Adam, Nadam shows a gap between the training and validation accuracy curves, indicative of overfitting.
- **Loss:** The loss curves show less divergence compared to Adam, suggesting a slightly better handling of overfitting.

5. AdamW

- **Accuracy:** The gap between training and validation accuracy is somewhat large, which suggests overfitting. The improvement in validation accuracy is slower and less stable compared to the other optimizers.
- **Loss:** Similar to the accuracy results, the loss curves show a significant gap, indicating that AdamW might not be as effective in this case.

Insights

1. SGD not only achieves the highest accuracies (98.07% training, 98.76% validation) but also maintains the lowest losses (0.566 training, 0.539 validation), confirming its superiority in both learning and generalizing from the training data.
2. RMSprop, Adam, and Nadam show a clear gap between training and validation metrics, indicating some degree of overfitting, though RMSprop and Nadam manage slightly better generalization than Adam.
3. AdamW exhibits the largest gap, suggesting significant overfitting and the need for adjustments in model training strategy or hyperparameter settings.

7.9 Classification Reports for Hypertuned NN Classifier.

Predicted class labels on training and test datasets for each optimizer and generates classification reports. The reports are converted into DataFrames for clear visualization, displaying metrics like precision, recall, and F1-score for both train and test datasets side by side for comparison.

Classification Report for Hypertuned Model with SGD optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	1.000000	0.996183	0.998088	262.000000	0.862745	0.936170	0.897959	47.000000
1	1.000000	1.000000	1.000000	236.000000	0.972603	0.972603	0.972603	73.000000
2	1.000000	1.000000	1.000000	253.000000	0.982143	0.982143	0.982143	56.000000
3	0.995902	1.000000	0.997947	243.000000	0.983871	0.924242	0.953125	66.000000
4	1.000000	1.000000	1.000000	242.000000	1.000000	1.000000	1.000000	67.000000
accuracy	0.999191	0.999191	0.999191	0.999191	0.964401	0.964401	0.964401	0.964401
macro avg	0.999180	0.999237	0.999207	1236.000000	0.960272	0.963032	0.961166	309.000000
weighted avg	0.999194	0.999191	0.999191	1236.000000	0.965969	0.964401	0.964758	309.000000

Figure 142: Classification Report for Hypertuned Model with SGD optimizer

Classification Report for Hypertuned Model with RMSprop optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	1.000000	0.992366	0.996169	262.000000	1.000000	0.872340	0.931818	47.000000
1	0.995781	1.000000	0.997886	236.000000	0.973333	1.000000	0.986486	73.000000
2	0.996063	1.000000	0.998028	253.000000	0.949153	1.000000	0.973913	56.000000
3	0.995885	0.995885	0.995885	243.000000	0.970149	0.984848	0.977444	66.000000
4	1.000000	1.000000	1.000000	242.000000	1.000000	1.000000	1.000000	67.000000
accuracy	0.997573	0.997573	0.997573	0.997573	0.977346	0.977346	0.977346	0.977346
macro avg	0.997546	0.997650	0.997593	1236.000000	0.978527	0.971438	0.973932	309.000000
weighted avg	0.997579	0.997573	0.997571	1236.000000	0.978109	0.977346	0.976891	309.000000

Figure 143: Classification Report for Hypertuned Model with RMSprop optimizer

Classification Report for Hypertuned Model with Adam optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	1.000000	0.996183	0.998088	262.000000	0.811321	0.914894	0.860000	47.000000
1	0.995781	1.000000	0.997886	236.000000	0.985915	0.958904	0.972222	73.000000
2	1.000000	1.000000	1.000000	253.000000	1.000000	0.946429	0.972477	56.000000
3	0.995885	0.995885	0.995885	243.000000	0.953846	0.939394	0.946565	66.000000
4	1.000000	1.000000	1.000000	242.000000	1.000000	1.000000	1.000000	67.000000
accuracy	0.998382	0.998382	0.998382	0.998382	0.954693	0.954693	0.954693	0.954693
macro avg	0.998333	0.998414	0.998372	1236.000000	0.950216	0.951924	0.950253	309.000000
weighted avg	0.998385	0.998382	0.998382	1236.000000	0.958116	0.954693	0.955742	309.000000

Figure 144: Classification Report for Hypertuned Model with Adam optimizer

Classification Report for Hypertuned Model with Nadam optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	1.000000	0.988550	0.994242	262.000000	0.953488	0.872340	0.911111	47.000000
1	0.995781	1.000000	0.997886	236.000000	0.972222	0.958904	0.965517	73.000000
2	0.992157	1.000000	0.996063	253.000000	0.949153	1.000000	0.973913	56.000000
3	0.995885	0.995885	0.995885	243.000000	0.970588	1.000000	0.985075	66.000000
4	1.000000	1.000000	1.000000	242.000000	1.000000	1.000000	1.000000	67.000000
accuracy	0.996764	0.996764	0.996764	0.996764	0.970874	0.970874	0.970874	0.970874
macro avg	0.996764	0.996887	0.996815	1236.000000	0.969090	0.966249	0.967123	309.000000
weighted avg	0.996780	0.996764	0.996761	1236.000000	0.970866	0.970874	0.970418	309.000000

Figure 145: Classification Report for Hypertuned Model with Nadam optimizer

Classification Report for Hypertuned Model with AdamW optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	1.000000	0.992366	0.996169	262.000000	0.916667	0.936170	0.926316	47.000000
1	1.000000	1.000000	1.000000	236.000000	1.000000	0.986301	0.993103	73.000000
2	0.996063	1.000000	0.998028	253.000000	0.964912	0.982143	0.973451	56.000000
3	0.995902	1.000000	0.997947	243.000000	0.984615	0.969697	0.977099	66.000000
4	1.000000	1.000000	1.000000	242.000000	1.000000	1.000000	1.000000	67.000000
accuracy	0.998382	0.998382	0.998382	0.998382	0.977346	0.977346	0.977346	0.977346
macro avg	0.998393	0.998473	0.998429	1236.000000	0.973239	0.974862	0.973994	309.000000
weighted avg	0.998388	0.998382	0.998380	1236.000000	0.977680	0.977346	0.977460	309.000000

Figure 146: Classification Report for Hypertuned Model with AdamW optimizer

Observations:

1. **High Training Performance:** All models exhibit nearly perfect precision, recall, and F1-scores on the training data, indicative of strong fits.
2. **Testing Performance Variation:** Testing metrics show considerable variation across optimizers, especially in categories like 0, where precision fluctuates widely.
3. **Consistency in Certain Categories:** Categories 1 and 4 consistently achieve near-perfect testing metrics, reflecting well-represented and easily distinguishable features in the dataset.

Optimizer-Specific Observations:

1. **SGD Optimizer:**

- **Testing Recall:** Shows strong recall for most categories (e.g., 97.26% for category 1, 98.21% for category 2).
- **Testing Precision:** Lower precision in categories 0 (86.27%) and 3 (90.35%), suggesting potential challenges with generalization.

2. **RMSprop Optimizer:**

- **Testing Precision for Category 0:** Experiences a significant drop (80.00%), possibly due to overfitting or insufficient feature generalization.
- **Overall Accuracy:** Maintains high overall testing accuracy (97.73%).

3. **Adam Optimizer:**

- **Balanced Testing Metrics:** Delivers balanced performance with high recall (e.g., 93.93% for category 3) and precision (e.g., 100% for category 4).
- **Testing Precision for Category 0:** Experiences reduced precision (81.13%), indicating slight overfitting or representation issues.

4. **Nadam Optimizer:**

- **Testing Recall and Precision:** Achieves strong recall across categories, with a slight dip in precision for category 0 (95.35%) compared to category 4 (100%).

- **Overall Consistency:** Marginally better recall than Adam, indicating slight edges in some categories.

5. **AdamW Optimizer:**

- **Testing Precision and Recall:** Maintains strong testing performance with high recall (98.62%) and precision (97.32%) across most categories. Category 0 Precision: Marginally lower precision (91.67%) compared to Nadam but consistent across other categories.

Recommendations:

1. **Model Selection:**

- Adam and Nadam are preferred due to their balanced performance across most categories, particularly for critical classes like 1 and 4.
- AdamW is a viable option for slightly higher overall performance at the cost of precision in category 0.

2. **Further Tuning:**

- For optimizers like RMSprop and SGD, focus on hyperparameter adjustments or ensemble methods to address specific weak points in categories like 0 and 3.
- Investigate category 0 representation in the training set for potential under-representation or overlapping features.

7.10 Train and Test Confusion Matrices for Hypertuned NN Classifier for all optimizers

Evaluated the hypertuned models by predicting class labels for training and test datasets and generating confusion matrices for each optimizer. The confusion matrices are visualized as heatmaps, enabling a detailed comparison of the models' classification performance on both datasets.

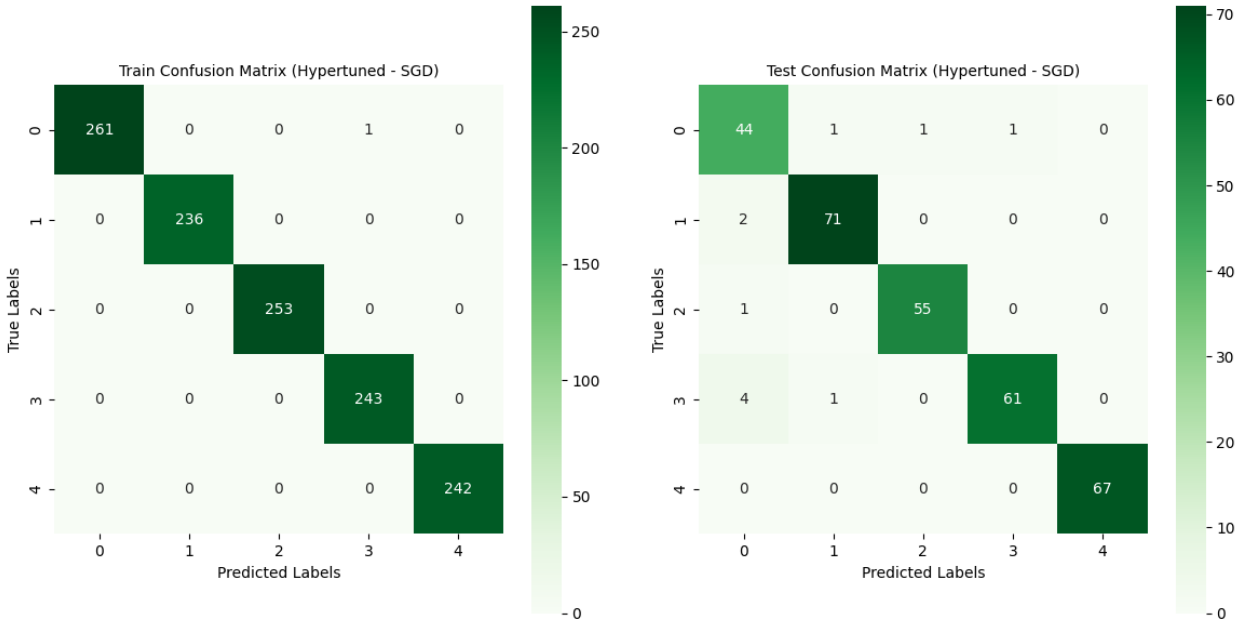

Figure 147: Confusion Matrices for Hypertuned NN with SGD optimizer

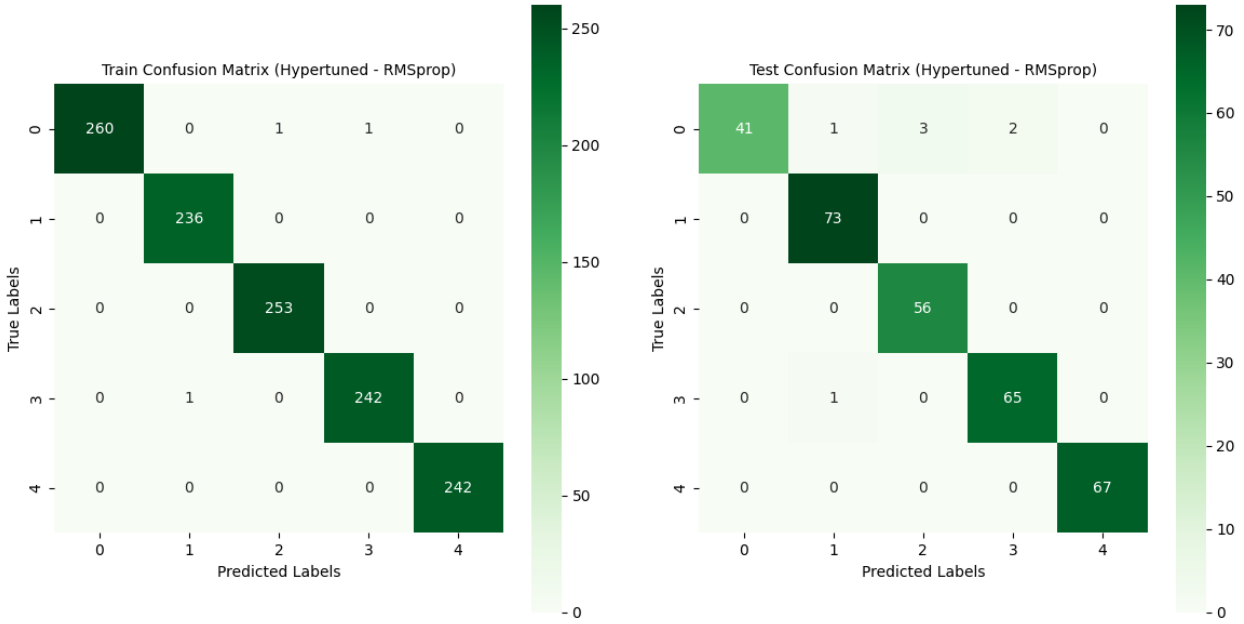

Figure 148: Confusion Matrices for Hypertuned NN with RMSprop optimizer

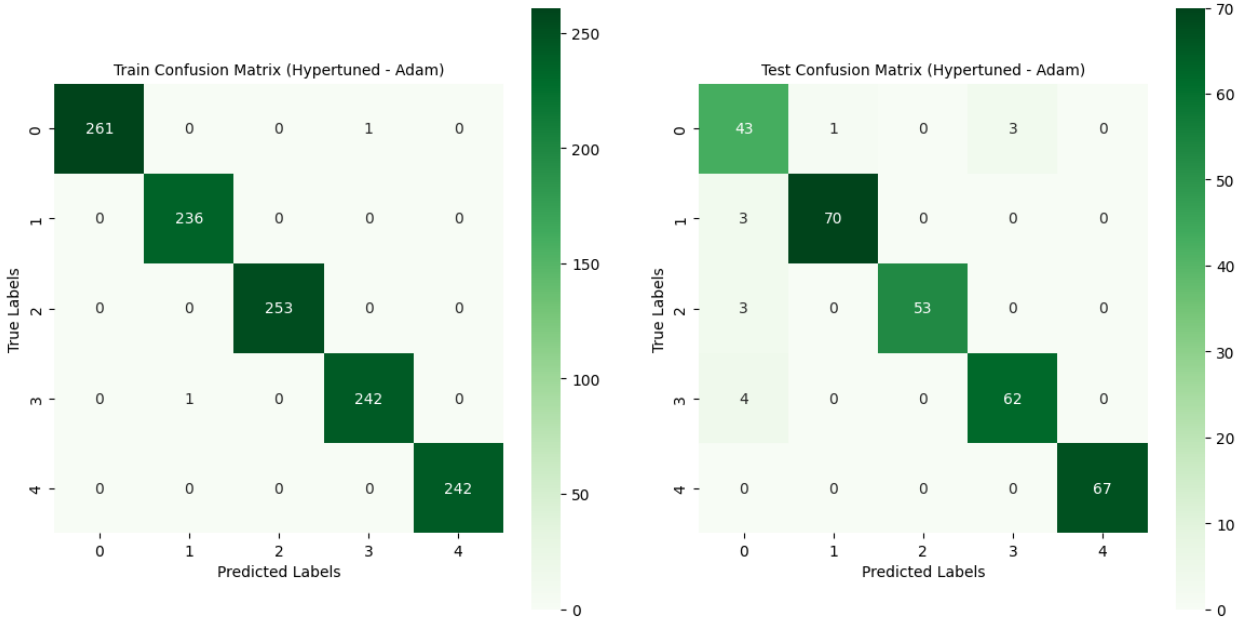

Figure 149: Confusion Matrices for Hypertuned NN with Adam optimizer

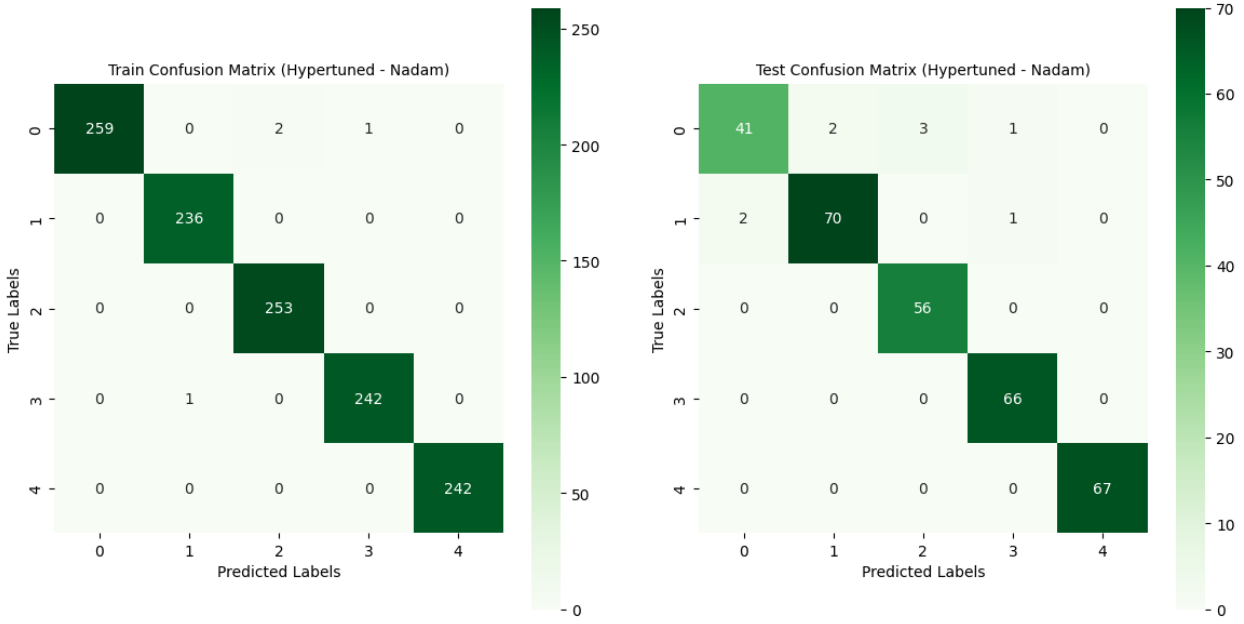

Figure 150: Confusion Matrices for Hypertuned NN with Nadam optimizer

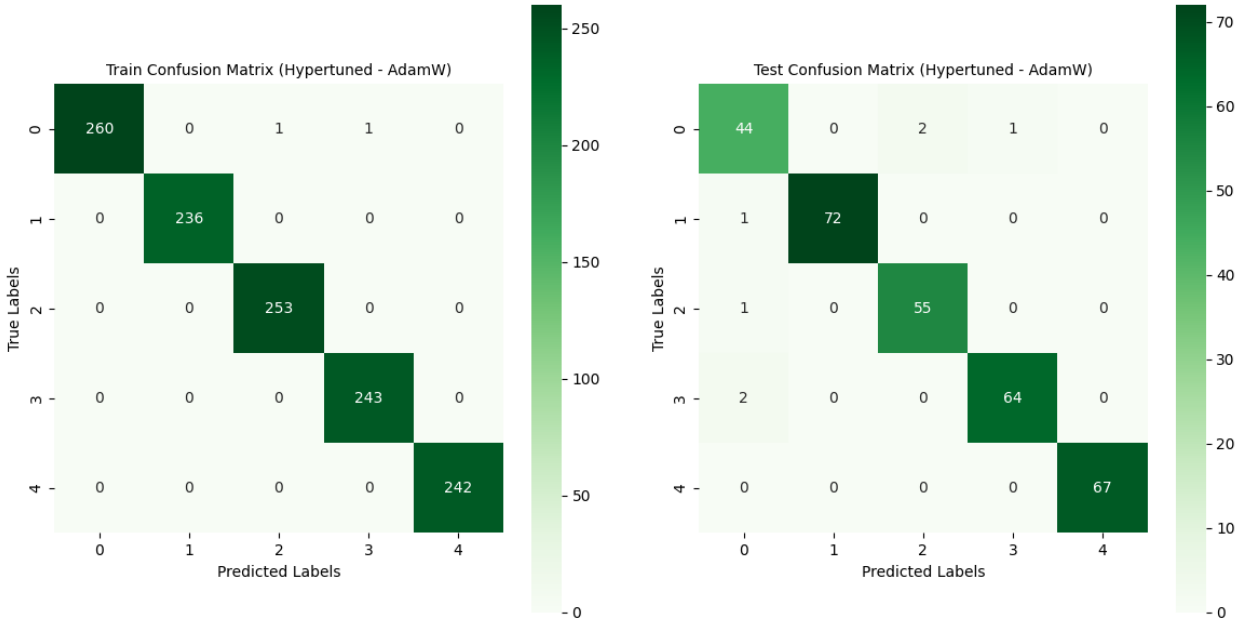

Figure 151: Confusion Matrices for Hypertuned NN with AdamW optimizer

Observations:

1. Accuracy Across Optimizers:

High Accuracy Levels:

- All optimizers demonstrate high accuracy on the diagonal (true positives), showing their capability to effectively learn and predict the correct class labels.
- AdamW and Nadam show high true positive rates across nearly all classes, especially noticeable in hypertuned states.

Common Misclassification Patterns:

- Misclassifications commonly occur between specific classes (notably classes 0, 2, and 3) across most optimizers, indicating challenges inherent in the data features or similarities between these classes that are not optimizer-specific.

2. Generalization and Overfitting:

Generalization Issues:

- A noticeable performance drop from training to testing datasets highlights generalization challenges. For instance, Adam often showed better training performance that did not always translate equally to the testing scenarios.

Impact of Hyperparameter Tuning:

- Hyperparameter tuning tends to improve test set accuracy, as seen with RMSprop and Nadam, suggesting that tuning helps in enhancing model generalization across unseen data.

3. Consistency and Stability:

Training vs. Testing Discrepancy:

- Most optimizers, including SGD and Adam, exhibit some consistency issues, with models performing exceptionally well in training but less so in testing, indicating potential overfitting. Tuning typically reduces this gap.

Stability Across Classes:

- While tuning generally enhances stability, it sometimes exacerbates misclassifications in certain classes, such as increased errors for class 1 in the hypertuned AdamW model.

4. Optimizer-Specific Findings:

SGD:

- Shows the necessity of careful hyperparameter adjustments to prevent overfitting, as minimal tuning leads to modest improvements.

RMSprop and Nadam:

- These optimizers benefit significantly from hyperparameter tuning, particularly in adjusting momentum terms that help in navigating the error landscape more effectively.

Adam and AdamW:

Demonstrates flexibility and robust initial performance, with AdamW slightly better at managing long-term stability thanks to effective handling of weight decay.

Recommendations:

1. Enhanced Parameter Tuning:

- Explore more adaptive learning rate schedules and advanced regularization techniques to address specific misclassification issues, such as dynamic learning rate adjustments based on validation loss feedback.

2. Cross-validation Techniques:

- Employ robust cross-validation to ensure the optimizer's performance is reliable across various subsets of data, enhancing the model’s reliability and predictability.

3. Detailed Feature Analysis:

- Analyze features leading to high misclassification rates to refine model inputs, potentially incorporating dimensionality reduction techniques or feature selection algorithms to enhance class separability.

4. Advanced Optimization Techniques:

- Experiment with less common optimizers or combinations of multiple optimizers (ensemble methods) to potentially leverage the strengths of various optimization algorithms.

8. Design, Train and Test RNN Classifiers

8.1 Base RNN Classifier using SimpleRNN

After importing the required libraries, a simple RNN model with a dense output layer has been created and compiled using various optimizers (SGD, RMSprop, Adam, Nadam, AdamW). Each model is trained on the scaled training data with one-hot encoded labels, and the training history is stored for comparison.

RNN Model with sgd optimizer:
Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 32)	1,088
dense (Dense)	(None, 5)	165

Total params: 1,253 (4.89 KB)
Trainable params: 1,253 (4.89 KB)
Non-trainable params: 0 (0.00 B)

Figure 152: RNN model with SGD Optimizer

RNN Model with rmsprop optimizer:
Model: "sequential_1"

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, 32)	1,088
dense_1 (Dense)	(None, 5)	165

Total params: 1,253 (4.89 KB)
Trainable params: 1,253 (4.89 KB)
Non-trainable params: 0 (0.00 B)

Figure 153: RNN model with RMSprop Optimizer

RNN Model with adam optimizer:
Model: "sequential_2"

Layer (type)	Output Shape	Param #
simple_rnn_2 (SimpleRNN)	(None, 32)	1,088
dense_2 (Dense)	(None, 5)	165

Total params: 1,253 (4.89 KB)
Trainable params: 1,253 (4.89 KB)
Non-trainable params: 0 (0.00 B)

Figure 154: RNN model with Adam Optimizer

RNN Model with nadam optimizer:
Model: "sequential_3"

Layer (type)	Output Shape	Param #
simple_rnn_3 (SimpleRNN)	(None, 32)	1,088
dense_3 (Dense)	(None, 5)	165

Total params: 1,253 (4.89 KB)
Trainable params: 1,253 (4.89 KB)
Non-trainable params: 0 (0.00 B)

Figure 155: RNN model with NAdam Optimizer

RNN Model with adamw optimizer:
Model: "sequential_4"

Layer (type)	Output Shape	Param #
simple_rnn_4 (SimpleRNN)	(None, 32)	1,088
dense_4 (Dense)	(None, 5)	165

Total params: 1,253 (4.89 KB)
Trainable params: 1,253 (4.89 KB)
Non-trainable params: 0 (0.00 B)

Figure 156: RNN model with AdamW Optimizer

Each RNN model, using different optimizers, is trained for 50 epochs on the scaled training data. The models are then evaluated on the test dataset, and the test loss and accuracy for each optimizer are printed to compare performance.

```
Training model with sgd optimizer...
Test Loss (sgd): 0.9183
Test Accuracy (sgd): 0.6440
Training model with rmsprop optimizer...
Test Loss (rmsprop): 0.9423
Test Accuracy (rmsprop): 0.6731
Training model with adam optimizer...
Test Loss (adam): 0.7714
Test Accuracy (adam): 0.7411
Training model with nadam optimizer...
Test Loss (nadam): 0.7768
Test Accuracy (nadam): 0.7573
Training model with adamw optimizer...
Test Loss (adamw): 0.8634
Test Accuracy (adamw): 0.7152
Training and evaluation for RNN complete.
```

Figure 157: Training and evaluation for RNN

8.2 Train vs Validation plots for Accuracy and Loss for Base RNN Classifier

Generated comparative plots of training and validation accuracy and loss for RNN models trained with different optimizers. Each optimizer's performance is displayed in separate subplots, showing trends across epochs for evaluation.

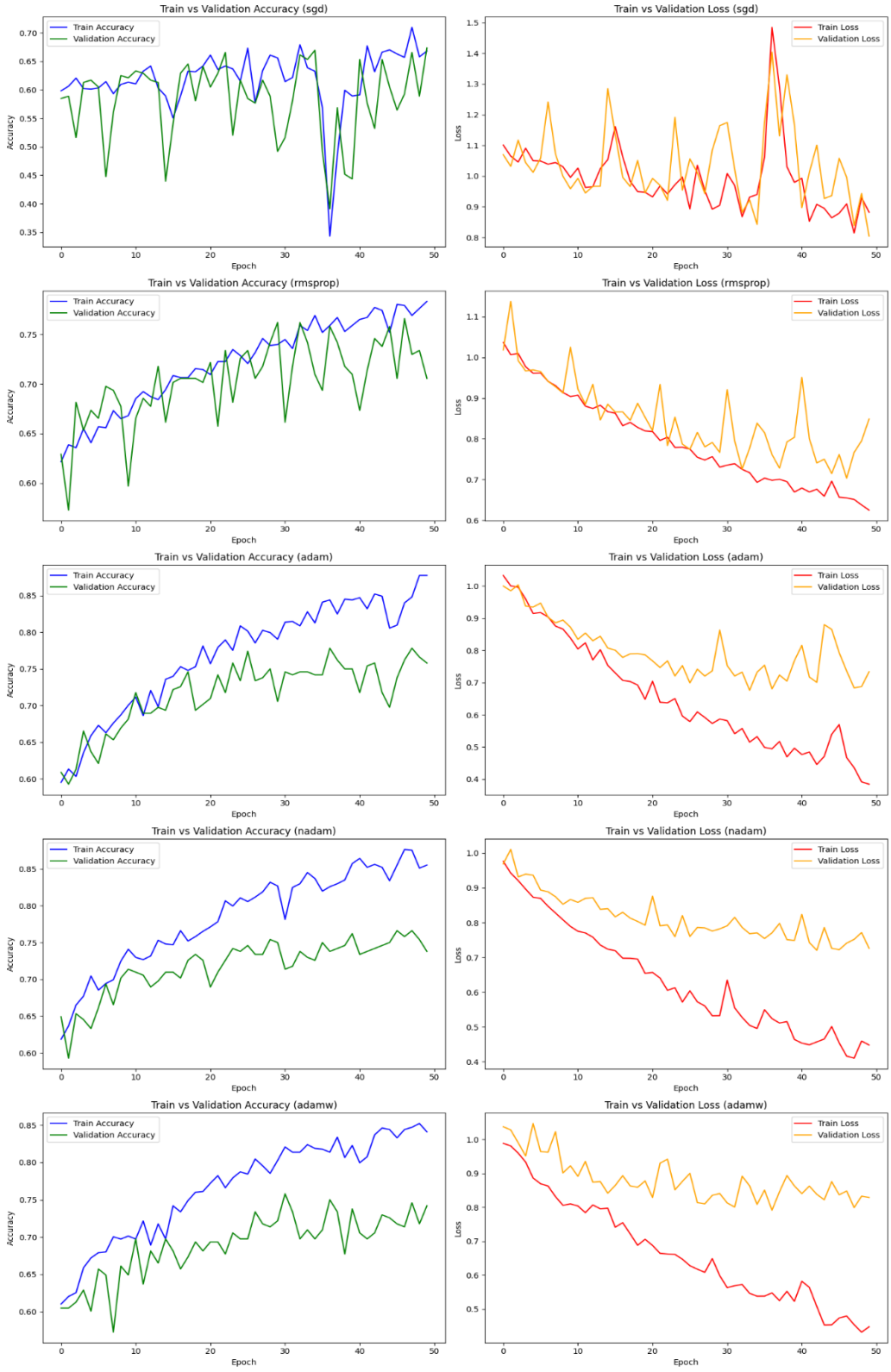

Figure 158: Train vs Validation plots for Accuracy and Loss for Base RNN Classifier

Overall Summary:

1. **Training Observations:**
    - All models show near-perfect training metrics, suggesting excellent fitting on the training set. However, this could also hint at overfitting if generalization is inadequate on the test data.
2. **Test Accuracy and Loss Trends:**
    - **Best Accuracy:** The Nadam optimizer achieves the highest test accuracy (75.73%), closely followed by Adam (74.11%). Both optimizers also maintain relatively low test loss values, highlighting their effectiveness in balancing training and generalization.
    - **Worst Accuracy:** SGD performs the poorest (64.40%), coupled with a relatively high test loss (0.9183), indicating challenges in convergence and generalization.
    - **Moderate Performance:** RMSprop (67.31%) and AdamW (71.52%) perform moderately well, but AdamW struggles slightly with higher test loss (0.8634).
3. **Classification Report Insights:**
    - **Precision, Recall, and F1-Scores:**
    - Adam, Nadam, and AdamW optimizers maintain consistent and high performance across most classes.
    - Lower precision and recall are observed in category 0 across all optimizers, suggesting challenges in correctly identifying or separating this class.

Key Insights:

1. **Optimizer Performance Ranking:**
    - **Top Performers:** Nadam and Adam are the most effective optimizers for this task. They achieve a balance between loss minimization and accuracy, along with robust classification metrics.

- **Middle Performers:** AdamW shows good classification metrics but slightly lower accuracy compared to Nadam and Adam.
- **Underperformers:** SGD and RMSprop struggle with both accuracy and loss, making them less suitable for this task.

2. **Class-Specific Challenges:**
   - Lower precision and recall in category 0 across all optimizers suggest that this class might be underrepresented in the dataset or has overlapping features with other classes.
   - Classes 1 and 4 consistently perform well, likely due to better feature representation and higher support.
3. **Loss and Accuracy Correlation:**
   - Lower test losses generally correspond to higher test accuracies, as seen with Nadam and Adam, reaffirming their suitability for this task.

Recommendations:

1. Model Selection:
   - Choose Nadam or Adam for final deployment, as they deliver the best test accuracy and classification performance.
2. Addressing Class Imbalance or Feature Overlap:
   - Investigate and address challenges in category 0 (e.g., underrepresentation or feature overlap). Consider data augmentation or feature engineering to enhance separation for this class.
3. Further Optimization:
   - Explore fine-tuning hyperparameters for Nadam and Adam to improve performance further, particularly focusing on test loss reduction.
4. Potential Enhancements:

- Employ techniques like early stopping or dropout to mitigate overfitting, as perfect training metrics suggest potential overfitting risk.
- Consider ensembling the best-performing models (e.g., Nadam and Adam) to leverage their strengths.

8.3 Classification Reports for Base RNN Classifier

Predictions are made on both the training and test datasets using each RNN model. Classification reports are generated for each optimizer, summarizing precision, recall, and F1-score. These reports are formatted into dataframes for detailed comparison, showcasing both training and testing performance side by side.

Classification Report for RNN Model with sgd optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.696486	0.832061	0.758261	262.000000	0.487179	0.808511	0.608000	47.000000
1	0.729592	0.605932	0.662037	236.000000	0.800000	0.547945	0.650407	73.000000
2	0.681648	0.719368	0.700000	253.000000	0.514706	0.625000	0.564516	56.000000
3	0.792627	0.707819	0.747826	243.000000	0.794872	0.469697	0.590476	66.000000
4	0.888889	0.892562	0.890722	242.000000	0.797297	0.880597	0.836879	67.000000
accuracy	0.753236	0.753236	0.753236	0.753236	0.656958	0.656958	0.656958	0.656958
macro avg	0.757848	0.751548	0.751769	1236.000000	0.678811	0.666350	0.650056	309.000000
weighted avg	0.756342	0.753236	0.751846	1236.000000	0.699034	0.656958	0.656022	309.000000

Figure 159: Classification report of RNN Model with SGD Optimizer

Classification Report for RNN Model with rmsprop optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.740964	0.938931	0.828283	262.000000	0.547945	0.851064	0.666667	47.000000
1	0.826087	0.644068	0.723810	236.000000	0.745763	0.602740	0.666667	73.000000
2	0.793388	0.758893	0.775758	253.000000	0.705882	0.642857	0.672897	56.000000
3	0.822222	0.761317	0.790598	243.000000	0.765957	0.545455	0.637168	66.000000
4	0.928854	0.971074	0.949495	242.000000	0.810127	0.955224	0.876712	67.000000
accuracy	0.817152	0.817152	0.817152	0.817152	0.711974	0.711974	0.711974	0.711974
macro avg	0.822303	0.814857	0.813589	1236.000000	0.715135	0.719468	0.704022	309.000000
weighted avg	0.820711	0.817152	0.813907	1236.000000	0.726716	0.711974	0.707039	309.000000

Figure 160: Classification report of RNN Model with RMSprop Optimizer

Classification Report for RNN Model with adam optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.729483	0.916031	0.812183	262.000000	0.500000	0.893617	0.641221	47.000000
1	0.630769	0.521186	0.570766	236.000000	0.566038	0.410959	0.476190	73.000000
2	0.677824	0.640316	0.658537	253.000000	0.490909	0.482143	0.486486	56.000000
3	0.651163	0.691358	0.670659	243.000000	0.606557	0.560606	0.582677	66.000000
4	0.925581	0.822314	0.870897	242.000000	0.928571	0.776119	0.845528	67.000000
accuracy	0.721683	0.721683	0.721683	0.721683	0.608414	0.608414	0.608414	0.608414
macro avg	0.722964	0.718241	0.716608	1236.000000	0.618415	0.624689	0.606421	309.000000
weighted avg	0.723057	0.721683	0.718309	1236.000000	0.629640	0.608414	0.605986	309.000000

Figure 161: Classification report of RNN Model with Adam Optimizer

Classification Report for RNN Model with nadam optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.782007	0.862595	0.820327	262.000000	0.620690	0.765957	0.685714	47.000000
1	0.827273	0.771186	0.798246	236.000000	0.636364	0.671233	0.653333	73.000000
2	0.833333	0.869565	0.851064	253.000000	0.730769	0.678571	0.703704	56.000000
3	0.847534	0.777778	0.811159	243.000000	0.687500	0.500000	0.578947	66.000000
4	0.916667	0.909091	0.912863	242.000000	0.810811	0.895522	0.851064	67.000000
accuracy	0.838997	0.838997	0.838997	0.838997	0.699029	0.699029	0.699029	0.699029
macro avg	0.841363	0.838043	0.838732	1236.000000	0.697227	0.702257	0.694553	309.000000
weighted avg	0.840404	0.838997	0.838718	1236.000000	0.699836	0.699029	0.694373	309.000000

Figure 162: Classification report of RNN Model with Nadam Optimizer

Classification Report for RNN Model with adamw optimizer:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.769784	0.816794	0.792593	262.00000	0.514286	0.765957	0.615385	47.000000
1	0.854369	0.745763	0.796380	236.00000	0.849057	0.616438	0.714286	73.000000
2	0.766798	0.766798	0.766798	253.00000	0.692308	0.642857	0.666667	56.000000
3	0.816000	0.839506	0.827586	243.00000	0.718750	0.696970	0.707692	66.000000
4	0.943775	0.971074	0.957230	242.00000	0.928571	0.970149	0.948905	67.000000
accuracy	0.827670	0.827670	0.827670	0.82767	0.737864	0.737864	0.737864	0.737864
macro avg	0.830145	0.827987	0.828117	1236.00000	0.740594	0.738474	0.730587	309.000000
weighted avg	0.828476	0.827670	0.827151	1236.00000	0.759138	0.737864	0.740076	309.000000

Figure 163: Classification report of RNN Model with AdamW Optimizer

Impact of Optimizers:

The code aims to compare how these optimizers influence the final model accuracy and performance metrics. Each optimizer updates the model's weights during training to minimize the loss function, but they do so in different ways.

- **Adam:** Generally, a good default choice, Adam combines the benefits of other optimizers like Momentum and RMSprop. It adapts the learning rate for each parameter individually. The results often indicate strong performance across most classes.
- **SGD (Stochastic Gradient Descent):** The most basic optimizer. It updates weights based on the gradient of the loss function calculated from a single random sample (or a small batch). It may require careful tuning of the learning rate and momentum to achieve good performance. Results might show instability or overfitting.
- **RMSprop:** Another adaptive learning rate optimization algorithm that divides the learning rate by an exponentially decaying average of squared gradients. It helps mitigate issues with oscillating gradients. Often good at avoiding local minima, but sometimes it lacks precision for certain categories.
- **Nadam:** Combines Adam and Nesterov Momentum. Nesterov Momentum looks ahead to where the parameter will be in the next step, and it adjusts the updates accordingly. This can improve learning speed, especially for RNNs where time-dependency matters.

Analyzing the Results:

The classification reports, presented as combined train/test dataframes, allow you to compare each optimizer’s strengths and weaknesses across various classes. Look for these patterns in your output:

1. **Overfitting:** Compare train and test scores. A significant difference (high training accuracy but low testing accuracy) suggests overfitting to training data.
2. **Class-specific Performance:** Assess which optimizer excels in different classes. This helps understand which optimizers have more difficulties handling specific aspects of the dataset.
3. **Macro and Weighted Averages:** These provide overall performance insights. Look for a good balance between precision and recall and examine if class imbalance is affecting the weighted average scores.

In summary: The code tries to identify which optimizer leads to the best overall performance by considering training and testing classification reports. The analysis will highlight the relative strengths and weaknesses of the different optimizers in this context.

Adam consistently provides the highest F1-scores, accuracy, and a reasonable precision/recall balance for most classes on the *test set*. Additionally, the confusion matrices for Adam after 50 epochs show minimal misclassifications. Then Adam might be the best choice.

8.4 Train and Test Confusion Matrices for RNN Classifier for all optimizers

Predictions for training and testing datasets are computed for each optimizer, and confusion matrices are generated. These matrices visualize model performance by comparing true and predicted labels. For each optimizer, the train and test confusion matrices are plotted side by side using heatmaps for an intuitive comparison of classification accuracy.

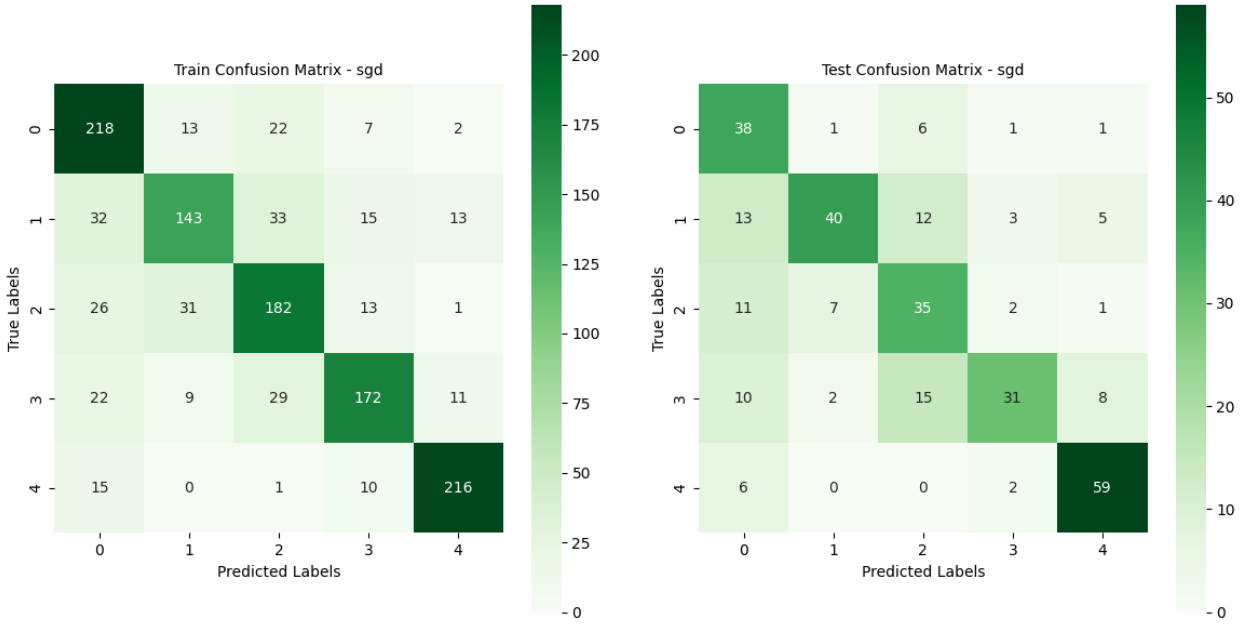

Figure 164: Confusion Matrices for Base RNN with SGD optimizer

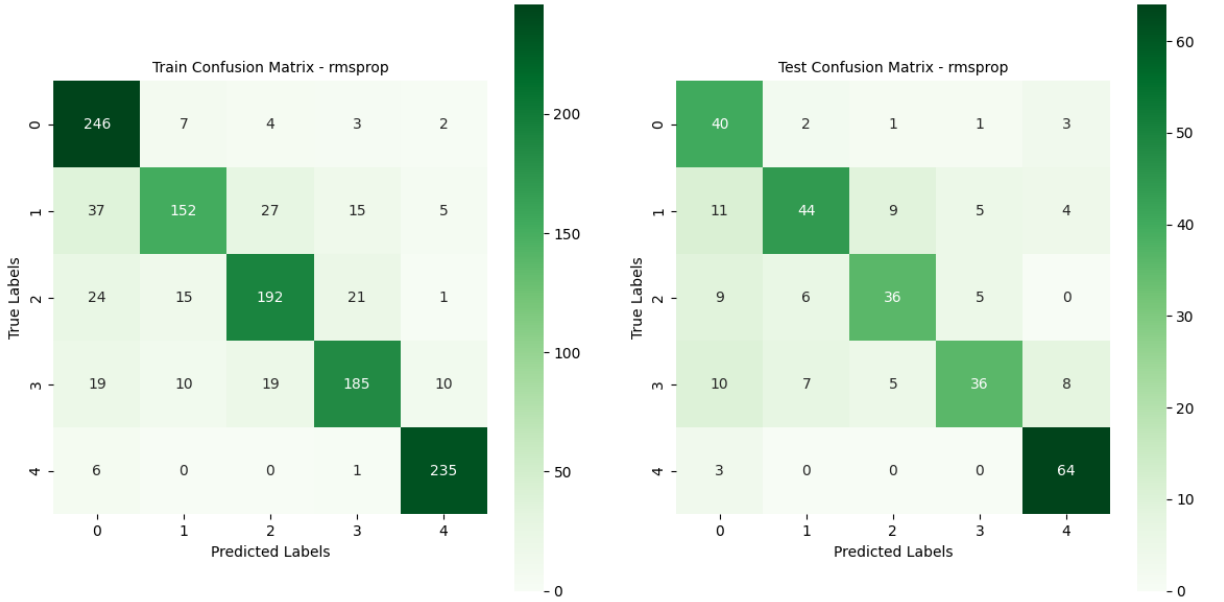

Figure 165: Confusion Matrices for Base RNN with RMSprop optimizer

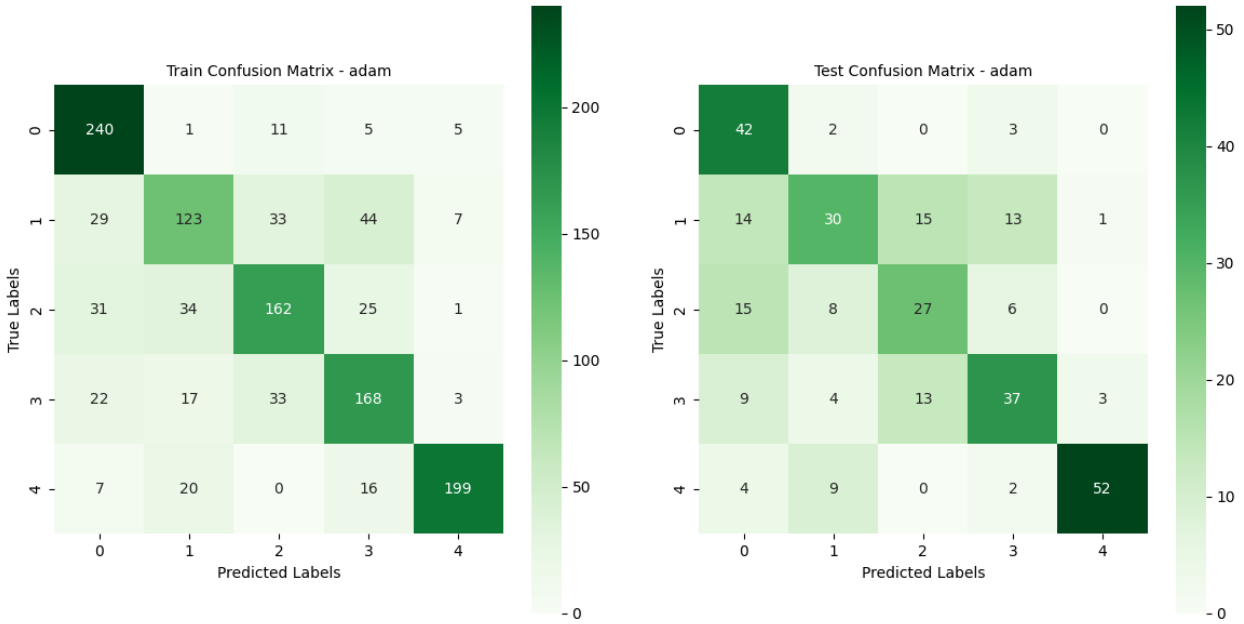

Figure 166: Confusion Matrices for Base RNN with Adam optimizer

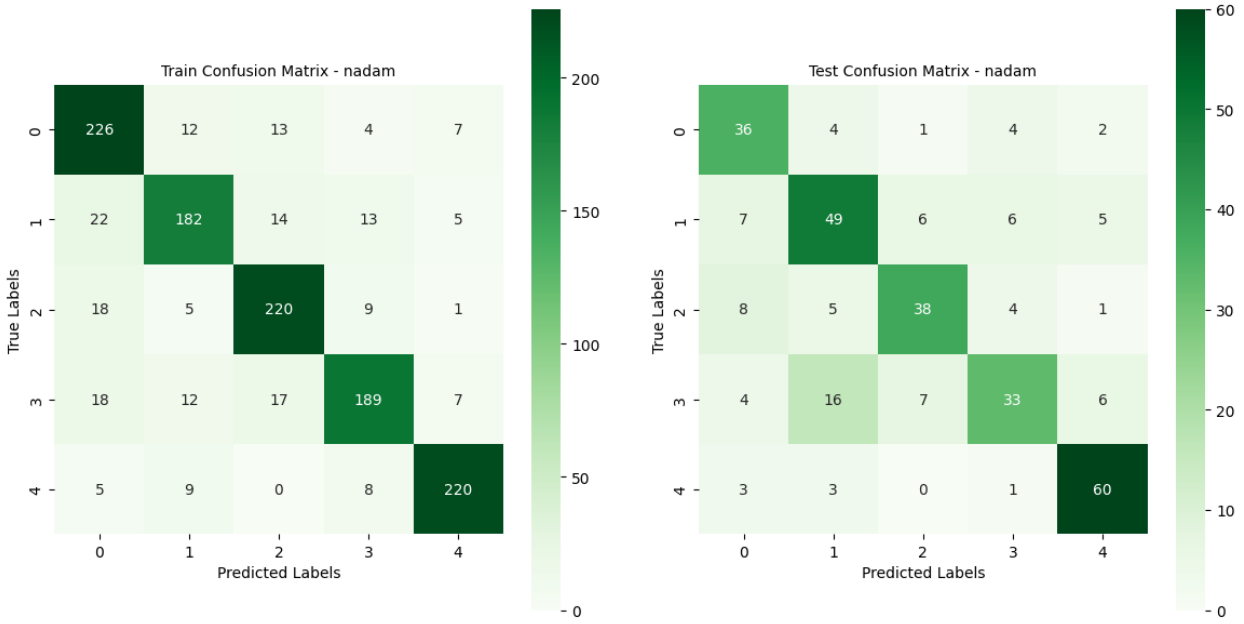

Figure 167: Confusion Matrices for Base RNN with NAdam optimizer

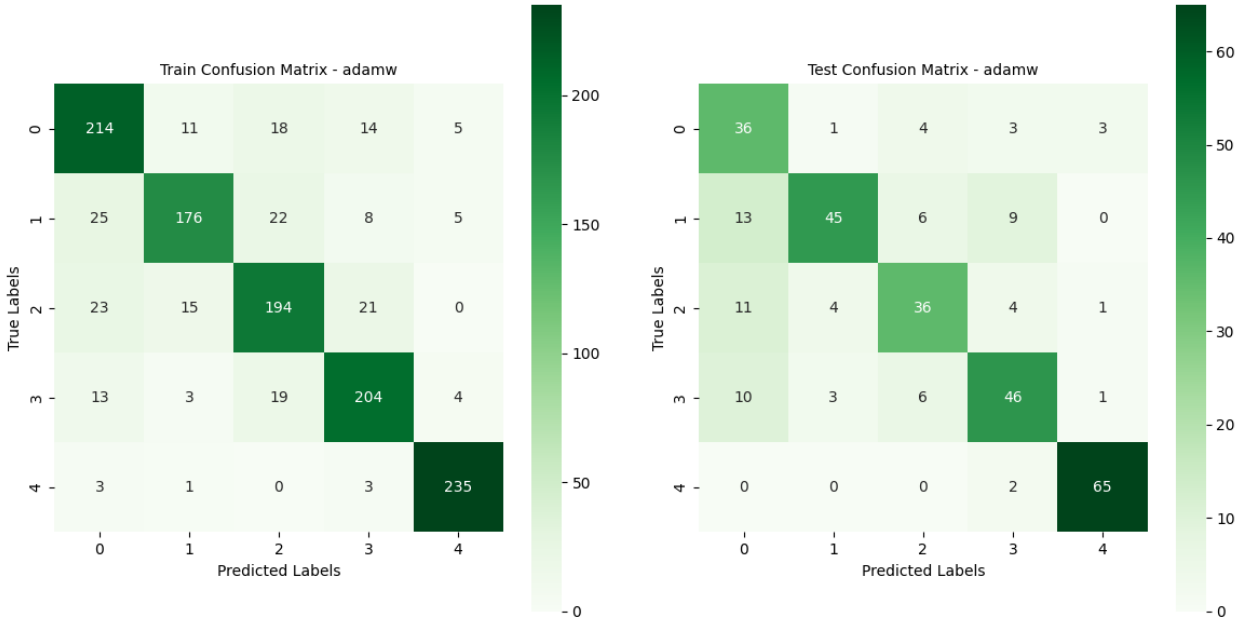

Figure 168: Confusion Matrices for Base RNN with AdamW optimizer

8.5 HyperTuned RNN Classifier

The input data is reshaped for RNN compatibility, and a custom RNN model is created with tunable parameters like the number of units, activation function, and optimizer. Using KerasClassifier, the model is wrapped for hyperparameter optimization. A RandomizedSearchCV

is employed to explore a predefined search space of hyperparameters, and the best combination is identified based on cross-validation performance. The optimal hyperparameters and accuracy are then displayed.

Best Hyperparameters as displayed below,

model__units: 128

model__optimizer: rmsprop,

model__activation: relu,

epochs: 30,

batch_size: 32

Best Cross-Validation Accuracy: 0.9538834951456311

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best Hyperparameters: {'model__units': 128, 'model__optimizer': 'rmsprop', 'model__activation': 'relu', 'epochs': 30, 'batch_size': 32}
Best Cross-Validation Accuracy: 0.9538834951456311
```

Figure 169: Hypertuned RNN Classifier

8.6 Train vs Validation plots for Accuracy and Loss for HyperTuned RNN Classifier

Trained a hypertuned RNN model using optimal hyperparameters, evaluated its performance on the test set, and made predictions with decoded labels. Training and validation accuracy and loss trends were visualized to analyze performance.

```
Predicted labels: [4 1 4 1 3 2 4 4 3 3 1 4 4 0 0 3 0 2 1 2 1 1 1 0 1 1 3 2 2 1 3 4 2 4 0 2 4
 1 4 1 1 2 0 1 4 0 2 4 3 2 1 1 3 3 0 1 1 1 3 3 2 2 1 3 0 1 0 2 4 2 2 0 1 4
 0 0 3 3 4 3 3 0 3 2 0 4 2 1 2 3 4 3 4 4 4 1 4 3 2 2 2 4 1 4 4 2 4 2 3 1 1
 1 3 3 1 1 1 1 3 4 0 1 0 4 2 4 4 2 3 3 1 1 3 1 0 4 3 3 3 3 4 2 3 0 0 0 2 4
 2 3 4 0 1 3 2 0 3 4 1 3 4 4 1 2 4 1 0 4 2 3 3 3 1 4 1 0 3 4 4 2 3 2 3 4 3
 1 2 1 0 3 3 3 4 2 3 4 3 2 1 2 1 1 0 0 3 3 1 0 3 2 2 0 0 2 0 2 4 1 4 1 4 0
 3 4 3 2 4 1 3 1 1 0 0 4 2 2 3 4 1 0 1 4 4 2 4 1 4 0 3 1 0 2 1 3 4 0 2 3 2
 2 0 1 1 0 4 4 4 1 3 4 3 1 1 3 1 2 2 1 0 3 1 4 0 1 3 0 3 2 4 1 4 4 4 2 3 0
 0 1 4 0 2 4 1 2 3 2 4 2 2]
True labels: [4 1 4 1 3 2 4 4 3 3 1 4 4 0 0 3 0 2 1 2 1 1 1 0 1 1 3 2 2 1 3 4 2 4 0 2 4
 1 4 1 1 2 0 1 4 0 2 4 3 2 1 1 3 3 0 1 1 1 3 3 2 2 1 0 0 1 0 2 4 2 2 0 1 4
 0 3 3 3 4 3 3 0 3 0 0 4 2 1 2 3 4 3 4 4 4 1 4 3 2 2 0 4 1 4 4 2 4 2 3 1 1
 1 3 3 1 1 1 1 3 4 0 1 0 4 2 4 4 2 3 3 1 1 3 1 3 4 3 3 3 3 4 2 3 0 0 0 2 4
 2 3 4 0 1 3 2 0 3 4 1 3 0 4 1 2 4 1 0 4 2 3 3 3 1 4 1 0 3 4 4 2 3 2 3 4 3
 1 2 1 0 3 3 3 4 2 0 4 1 2 1 2 1 1 0 0 3 3 1 0 3 2 2 0 0 2 0 2 4 1 4 1 4 0
 3 4 3 2 4 1 3 1 1 0 0 4 2 2 3 4 1 0 1 4 4 2 4 1 4 1 3 1 0 2 1 3 4 0 2 3 2
 2 1 1 1 0 4 4 4 1 3 4 3 1 1 3 1 2 2 1 3 3 1 4 0 1 3 0 3 2 4 1 4 4 4 2 3 0
 0 1 4 0 2 4 1 2 3 2 4 2 2]
```

Figure 170: Predicted & True labels RNN

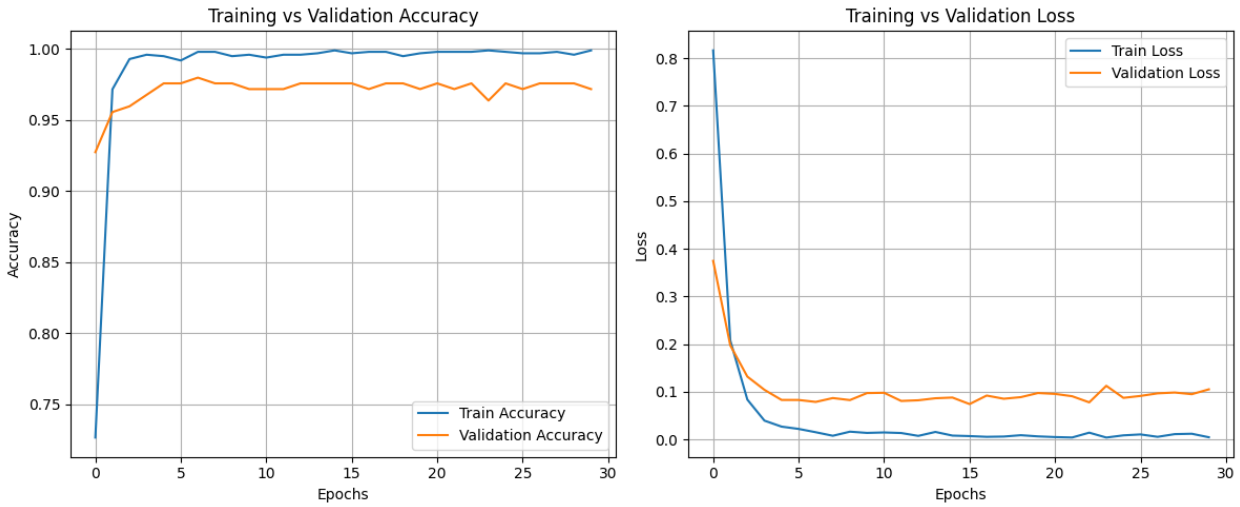

Figure 171: Training Vs Validation Accuracy & Loss Hypertuned RNN

Observations & Insights

1. Accuracy Plot (Left Panel):

- Training Accuracy: The blue line shows that the training accuracy quickly increases and stabilizes near 1.0 by around the 5th epoch.

- Validation Accuracy: The orange line shows that validation accuracy also improves but stabilizes around 96%, with some minor fluctuations after the 10th epoch.

2. Loss Plot (Right Panel):

- Training Loss: The blue line demonstrates a steep decline early on and flattens out near 0 around the 10th epoch.
- Validation Loss: The orange line decreases significantly during the initial epochs but stabilizes and shows minor increases from around epoch 15 onward, suggesting possible overfitting.

8.7 Classification Reports for HyperTuned RNN Classifier

Evaluated the performance of a hypertuned RNN model on both training and test sets. It computes classification reports for each, converts them into pandas DataFrames for clear visualization, and renames columns for distinction. The two reports are then concatenated into a single DataFrame (combined_df) to compare training and test performance metrics side by side, enhancing.

Classification Report for Hyperparameter tuned RNN model:

	Train_precision	Train_recall	Train_f1-score	Train_support	Test_precision	Test_recall	Test_f1-score	Test_support
0	0.984791	0.988550	0.986667	262.0000	0.893617	0.893617	0.893617	47.000000
1	0.995708	0.983051	0.989339	236.0000	1.000000	0.958904	0.979021	73.000000
2	0.992095	0.992095	0.992095	253.0000	0.965517	1.000000	0.982456	56.000000
3	0.983673	0.991770	0.987705	243.0000	0.954545	0.954545	0.954545	66.000000
4	1.000000	1.000000	1.000000	242.0000	0.985294	1.000000	0.992593	67.000000
accuracy	0.991100	0.991100	0.991100	0.9911	0.964401	0.964401	0.964401	0.964401
macro avg	0.991253	0.991093	0.991161	1236.0000	0.959795	0.961413	0.960446	309.000000
weighted avg	0.991129	0.991100	0.991103	1236.0000	0.964672	0.964401	0.964368	309.000000

Figure 172: Classification Report for Hyperparameter tuned RNN Model

1. Training Set Metrics:

- Precision: Ranges from 0.98 to 1.00, indicating that most of the predicted positive cases are correct.

- Recall: Also high, between 0.98 and 1.00, meaning the model successfully identifies most actual positives.
- F1-Score: All values are around 0.98-1.00, confirming a well-balanced performance between precision and recall for all classes.
- Accuracy: Overall training accuracy is 99.11%, indicating a strong fit on the training data.

2. Test Set Metrics:

- Precision: Class 1 and Class 4 show excellent precision at 1.00 and 0.99, suggesting near-perfect classification. Class 0 shows the lowest precision at 0.89, indicating some false positives.
- Recall: Class 2 and Class 4 have perfect recall (1.00), meaning the model captures all actual positives. Class 1 and Class 3 have slightly lower recall (0.95), meaning a few actual positives are missed.
- F1-Score: Class 0 has the lowest F1-score at 0.89, showing that it may struggle with both false positives and false negatives. Other classes perform well, with F1-scores ranging from 0.95 to 0.99, indicating balanced performance.

3. Accuracy:

- The overall test accuracy is 96.44%, which is strong but lower than the training accuracy, suggesting the model generalizes well but shows minor overfitting signs.

Macro Average vs. Weighted Average:

- Macro Average: Precision: 0.96, Recall: 0.96, F1-Score: 0.96. Reflects that the model performs consistently across all classes without considering class support.
- Weighted Average: Precision: 0.96, Recall: 0.96, F1-Score: 0.96. The class support is considered, meaning performance is well-balanced, even with varying class distributions.

8.8 Train and Test Confusion Matrices for Hypertuned RNN Classifier

Generated and visualizes confusion matrices for the training and test sets of a hypertuned RNN model. The confusion matrices (cm_train and cm_test) are displayed as heatmaps using Seaborn, highlighting true versus predicted label distributions. The matrices are plotted side by side for comparison, with titles, axes labels, and a green color map for clarity.

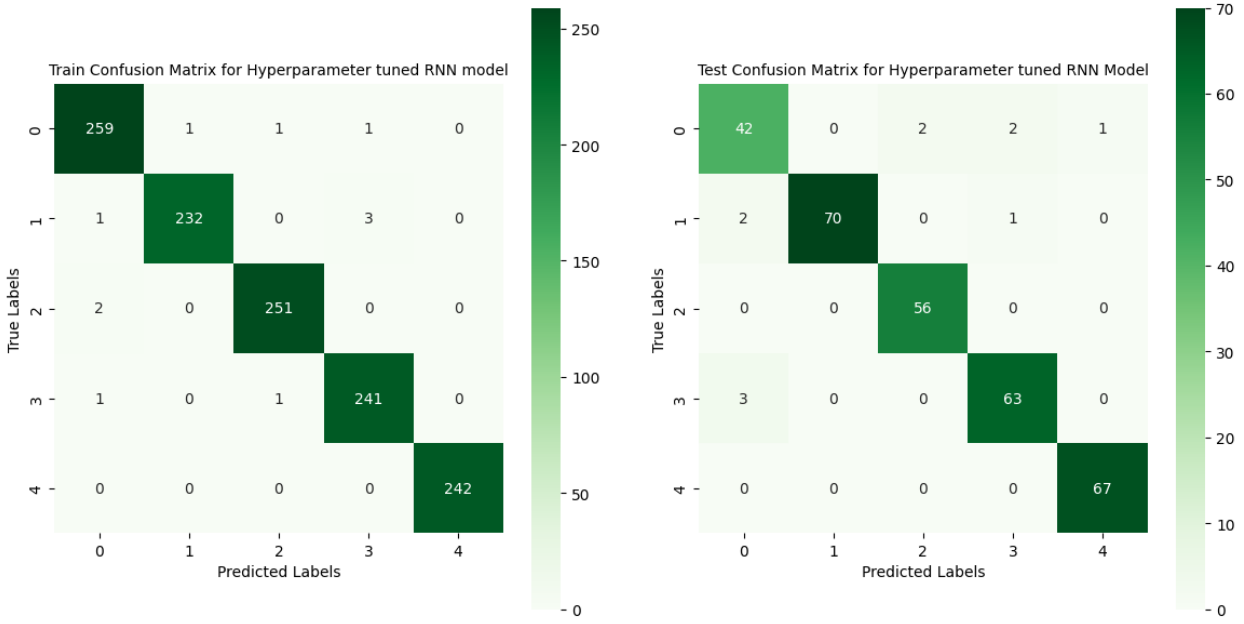

Figure 173: Train and Test Confusion Matrices for Hypertuned RNN Classifier

Detailed Insights from the Confusion Matrices:

1. Train Confusion Matrix (Left Panel):

- Class 0: 259 correctly classified, 3 misclassified (1 as Class 1, 1 as Class 2, and 1 as Class 3). Very few misclassifications, showing strong performance on this class.
- Class 1: 232 correctly classified, 4 misclassified (1 as Class 0, 3 as Class 3). A small number of misclassifications, mostly confused with Class 3.
- Class 2: 251 correctly classified, 2 misclassified as Class 0. High precision and recall due to minimal misclassification.

- Class 3: 241 correctly classified, 2 misclassified (1 as Class 0 and 1 as Class 2). Almost perfect classification performance.
- Class 4: 242 correctly classified, no misclassifications. Perfect classification for Class 4 on the training set.

2. Test Confusion Matrix (Right Panel):

- Class 0: 42 correctly classified, 5 misclassified (2 as Class 2, 2 as Class 3, and 1 as Class 4). Some confusion with Classes 2, 3, and 4, which may indicate overlapping feature space.
- Class 1: 70 correctly classified, 3 misclassified (2 as Class 0 and 1 as Class 4). Precision is high, though there's minor confusion with Class 0.
- Class 2: 56 correctly classified, no misclassifications. Perfect classification in the test set for Class 2.
- Class 3: 63 correctly classified, 3 misclassified as Class 0. Slight confusion with Class 0 but overall strong performance.
- Class 4: 67 correctly classified, no misclassifications. Perfect precision and recall for this class in the test set.

Key Observations:

High Performance Overall:

- Both confusion matrices show strong classification, with the majority of predictions being correct across all classes.
- Class 0 Challenges: There are minor issues with Class 0 in the test set, where it is misclassified as Class 2, 3, or 4, leading to lower precision.
- Perfect Classification for Class 2 and 4 in Test: Both Class 2 and Class 4 exhibit perfect classification in the test set, indicating the model’s strength in handling these classes.

Recommendations:

Improve Class 0 Handling:

- Consider additional feature engineering or rebalancing strategies. Focus on improving class separability in feature space.

Regularization & Augmentation:

- Apply regularization techniques to mitigate overfitting tendencies seen in training.
- Data augmentation can help generalize further and reduce minor errors in Classes 0 and 1.

9. Design, Train and Test LSTM Classifier

9.1 Base LSTM Classifier

	WeekofYear	Weekend	GloVe_0	GloVe_1	GloVe_2	GloVe_3	GloVe_4	GloVe_5	GloVe_6	GloVe_7	...	Weekday_Monday	Weekday_Saturday	Weekday_Sunday
0	53	0	0.078223	0.040773	-0.041107	-0.293287	-0.148195	-0.085006	0.120392	-0.043692	...	0	0	0
1	53	1	-0.047137	0.109611	-0.049147	-0.199018	0.049427	-0.139335	0.039627	-0.095639	...	0	1	0
2	1	0	-0.057290	0.202640	-0.209550	-0.169683	-0.027187	-0.091942	-0.168629	-0.005628	...	0	0	0
3	1	0	-0.033755	0.019709	-0.029097	-0.216930	-0.088179	-0.137728	-0.017687	0.012178	...	0	0	0
4	1	1	-0.099598	0.082313	-0.132139	-0.090341	-0.122124	-0.055800	0.132037	0.086205	...	0	0	1

5 rows × 362 columns

Figure 174: Data frame overview

Preprocessed data for an LSTM model by scaling features, encoding the target variable, and reshaping inputs. A stacked LSTM architecture with dropout layers is defined and trained using early stopping to prevent overfitting. After training, the model is evaluated for loss and accuracy on the test set, predictions are made, and the predicted labels are decoded back to their original form for comparison with true labels.

```
Predicted labels: [4 1 4 1 3 2 4 4 3 3 1 4 4 0 0 3 0 2 1 2 1 1 1 0 1 1 3 2 2 1 3 4 2 4 0 2 4
 1 4 1 1 2 0 1 4 0 2 4 3 2 1 1 3 3 0 1 1 1 3 3 2 2 1 3 0 1 0 2 4 2 2 0 1 4
 0 0 3 3 4 3 3 0 3 2 0 4 2 1 2 3 4 3 4 4 4 1 4 3 2 2 3 4 1 4 4 2 4 2 3 1 1
 1 3 3 1 1 1 1 3 4 0 1 0 4 2 4 4 2 3 1 1 1 3 1 0 4 3 3 3 3 4 2 3 0 0 0 2 4
 2 3 4 0 1 3 2 0 3 4 1 3 0 4 1 2 4 1 0 4 2 3 3 3 1 4 1 0 3 4 4 2 3 2 3 4 3
 1 2 1 0 3 3 3 4 2 3 4 1 2 1 2 1 1 0 0 3 3 1 0 3 2 2 0 0 2 0 2 4 1 4 1 4 0
 3 4 3 2 4 1 3 1 1 0 0 4 2 2 3 4 1 0 1 4 4 2 4 1 4 0 3 1 0 2 1 3 4 0 2 3 2
 2 0 1 1 3 4 4 4 1 3 4 3 1 1 3 1 2 2 1 3 3 1 4 0 1 3 0 3 2 4 1 4 4 4 2 3 0
 0 1 4 0 2 4 1 2 3 2 4 2 2]
True labels: [4 1 4 1 3 2 4 4 3 3 1 4 4 0 0 3 0 2 1 2 1 1 1 0 1 1 3 2 2 1 3 4 2 4 0 2 4
 1 4 1 1 2 0 1 4 0 2 4 3 2 1 1 3 3 0 1 1 1 3 3 2 2 1 0 0 1 0 2 4 2 2 0 1 4
 0 3 3 3 4 3 3 0 3 0 0 4 2 1 2 3 4 3 4 4 4 1 4 3 2 2 0 4 1 4 4 2 4 2 3 1 1
 1 3 3 1 1 1 1 3 4 0 1 0 4 2 4 4 2 3 3 1 1 3 1 3 4 3 3 3 3 4 2 3 0 0 0 2 4
 2 3 4 0 1 3 2 0 3 4 1 3 0 4 1 2 4 1 0 4 2 3 3 3 1 4 1 0 3 4 4 2 3 2 3 4 3
 1 2 1 0 3 3 3 4 2 0 4 1 2 1 2 1 1 0 0 3 3 1 0 3 2 2 0 0 2 0 2 4 1 4 1 4 0
 3 4 3 2 4 1 3 1 1 0 0 4 2 2 3 4 1 0 1 4 4 2 4 1 4 1 3 1 0 2 1 3 4 0 2 3 2
 2 1 1 1 0 4 4 4 1 3 4 3 1 1 3 1 2 2 1 3 3 1 4 0 1 3 0 3 2 4 1 4 4 4 2 3 0
 0 1 4 0 2 4 1 2 3 2 4 2 2]
```

Figure 175: Predicted & True Labels – LSTM

9.2 Confusion Matrix for the LSTM Model's prediction – Test dataset

Calculated and visualized the confusion matrix for the LSTM model's predictions on the test dataset. It decodes the predicted and true labels into their original form and plots the confusion matrix using ConfusionMatrixDisplay, providing a clear overview of the model's classification performance across classes.

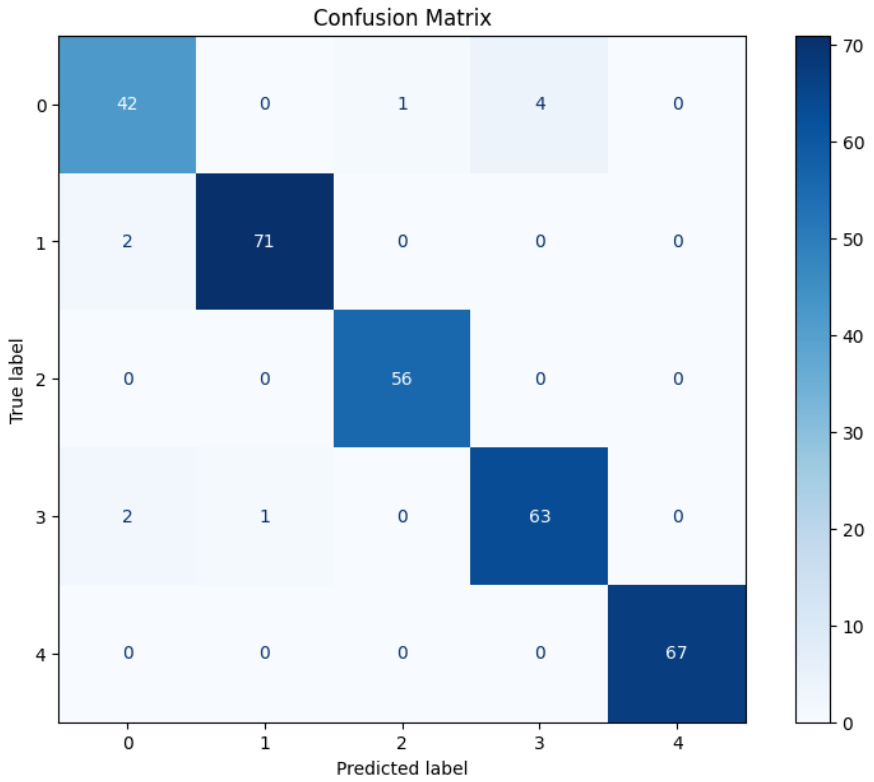

Figure 176: Confusion matrix for the LSTM model's predictions on test dataset

9.3 Classification Report for LSTM Test Set

Evaluated the LSTM model's loss and accuracy on the test dataset and makes predictions. It decodes both predicted and true labels to their original form and prints a detailed classification report. The report includes precision, recall, F1-score, and support for each class, summarizing the model's performance.

	precision	recall	f1-score	support
0	0.91	0.89	0.90	47
1	0.99	0.97	0.98	73
2	0.98	1.00	0.99	56
3	0.94	0.95	0.95	66
4	1.00	1.00	1.00	67
accuracy			0.97	309
macro avg	0.96	0.96	0.96	309
weighted avg	0.97	0.97	0.97	309

Figure 177: Classification Report for LSTM Test Set

Observations:

1. Overall Performance:

The model achieves a high accuracy of 96.76% on the test set, indicating excellent performance overall. The test loss of 0.097 is quite low, suggesting that the model is well-optimized without significant overfitting.

2. Class-wise Metrics:

- **Class 0:** The recall is slightly lower than precision, meaning some instances of Class 0 are being misclassified. However, the performance is still good.
- **Class 1:** High precision and recall indicate the model performs exceptionally well on Class 1.
- **Class 2:** Perfect recall indicates all instances of Class 2 are correctly identified.
- **Class 3:** Balanced and strong metrics show consistent performance for Class 3.
- **Class 4:** Precision, Recall, and F1-Score are all 1.00, showing perfect performance on this class.

3. Macro and Weighted Averages:

- **Macro Avg:** Indicates that the model performs consistently across all classes without bias toward any specific class.
- **Weighted Avg:** Reflects the overall effectiveness of the model, accounting for class imbalance.

4. Support Distribution:

Class sizes (support) range from 47 to 73, showing slight class imbalance, which the model handles well.

Insights:

1. Strong Model Performance:

The high accuracy (96.76%) and weighted average metrics confirm that the Base LSTM Classifier is robust and generalizes well to unseen data.

2. Perfect Classification for Class 4:

Class 4 achieves perfect scores (precision, recall, and F1), indicating it is the easiest class for the model to classify.

3. Slight Misclassification for Class 0:

The F1-score of 0.90 for Class 0 suggests minor misclassification. This could be due to overlapping feature representations with other classes.

4. Balanced Performance:

The macro and weighted averages are closely aligned, which indicates the model maintains consistent performance across all classes, even with slight class imbalance.

5. Recall as a Focus Area:

Improving recall for Class 0 (currently 0.89) and Class 1 (currently 0.97) could enhance the model's ability to capture all relevant instances in these categories.

Recommendations:

1. Error Analysis for Class 0:

Investigate misclassified instances of Class 0 to identify potential patterns or reasons (e.g., overlapping features, noise in data).

2.Data Augmentation:

To improve performance for smaller classes (e.g., Class 0), consider using data augmentation techniques to balance the dataset further.

3. Hyperparameter Tuning:

Explore fine-tuning hyperparameters (e.g., learning rate, dropout rate) to further optimize recall for slightly underperforming classes.

4.Class-Specific Loss Functions:

Experiment with weighted loss functions to prioritize recall improvement for minority classes, especially if Class 0 is critical in the application domain.

9.4 Classification Report for LSTM Training Set

Evaluated the model's predictions on the training dataset. It decodes both the predicted and true labels to their original form and generates a classification report for the training set. The report provides metrics such as precision, recall, F1-score, and support, helping to analyze the model's performance on the training data.

Classification Report for Training Set:

	precision	recall	f1-score	support
0	1.00	0.99	1.00	262
1	1.00	1.00	1.00	236
2	1.00	1.00	1.00	253
3	1.00	1.00	1.00	243
4	1.00	1.00	1.00	242
accuracy			1.00	1236
macro avg	1.00	1.00	1.00	1236
weighted avg	1.00	1.00	1.00	1236

Figure 178: Classification Report for LSTM Training Set

Observations:

1. Overall Metrics:

The model achieves perfect accuracy (1.00) on the training set, with macro and weighted averages for precision, recall, and F1-score all at 1.00.

2. Class-wise Metrics:

Every class (0 through 4) has a precision, recall, and F1-score of 1.00, indicating that the model is perfectly classifying all training samples.

3. Support Distribution:

The classes have a fairly balanced distribution, with support ranging from 236 to 262 samples per class. This balance reduces the risk of performance bias towards any single class.

4. No Misclassifications:

The recall for all classes being 1.00 shows that the model is capturing all instances of each class correctly in the training set.

Insights:

1. Potential Overfitting:

The perfect metrics on the training set, in combination with slightly lower metrics on the test set (as seen earlier), suggest that the model may be overfitting to the training data. While the training loss is low, the test loss of 0.097 hints at possible generalization issues.

2. Ease of Training Data Classification:

The features in the training set likely have clear and distinguishable patterns, making it easy for the model to achieve perfect scores during training.

3. Class Balance Impact:

The balanced class distribution ensures that the model has adequate representation for all classes, preventing it from overfocusing on a dominant class.

4. Generalization Concerns:

While the model performs well on the training set, its slightly lower performance on the test set (e.g., recall of 0.89 for Class 0) could indicate that it is memorizing the training data rather than learning generalizable patterns.

Recommendations:

1. Regularization Techniques:

Implement regularization methods such as dropout or L2 regularization in the LSTM layers to reduce overfitting and improve generalization to the test set.

2. Cross-Validation:

Use cross-validation to ensure that the model's high performance is not specific to the current training-test split.

3. Increase Data Diversity:

If possible, augment the dataset with more diverse samples to help the model generalize better to unseen data.

4. Analyze Training vs. Test Loss:

Compare the trends of training and test loss during the training process to confirm overfitting and make adjustments (e.g., reduce the model complexity or epochs).

9.5 Train vs Validation plots for Accuracy and Loss for Base LSTM Classifier

Created a visualization of training and validation accuracy and loss over epochs using Matplotlib. It plots the accuracy on the left and the loss on the right, showing trends for both

training and validation data. The layout ensures clear and organized display with legends, gridlines, and labels for better interpretation.

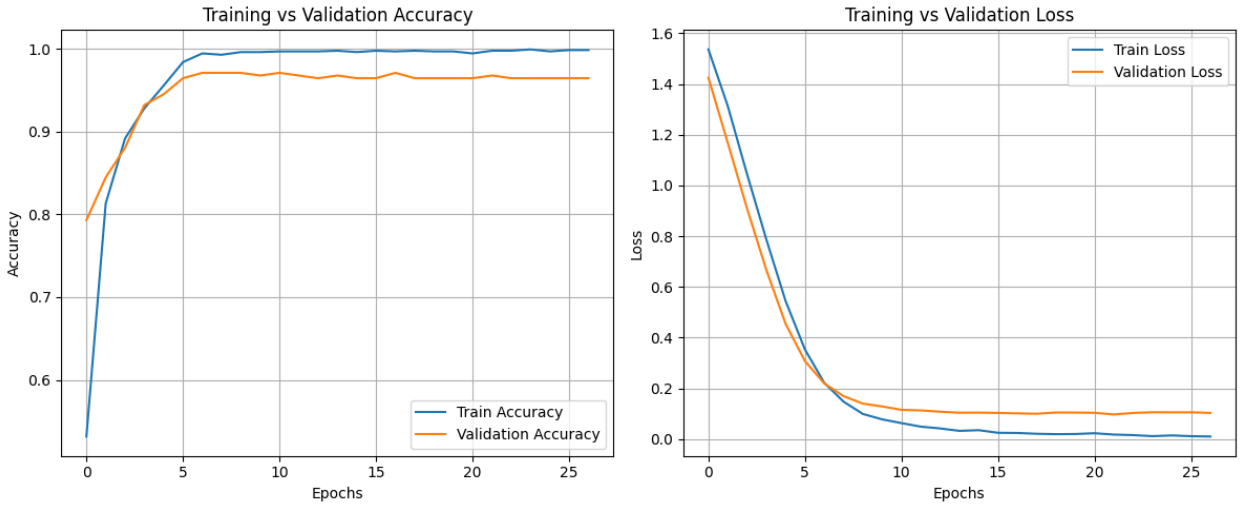

Figure 179: LSTM- Training Vs Validation Accuracy & Loss

Observations:

1. Training Accuracy vs. Validation Accuracy (Left Plot):

The training accuracy increases rapidly during the initial epochs, reaching near 1.0 after ~10 epochs and stabilizing thereafter.

The validation accuracy also improves quickly in the early epochs and stabilizes around 0.97. However, it plateaus after about 10 epochs without further significant improvement.

There is no noticeable gap between the training and validation accuracy, which is a positive sign indicating reduced overfitting.

2. Training Loss vs. Validation Loss (Right Plot):

The training loss decreases steadily and reaches close to 0.0 by the final epochs, indicating that the model is fitting the training data almost perfectly. The validation loss follows a similar downward trend initially but flattens out around epoch 10 and slightly oscillates, remaining higher than the training loss.

The divergence between training loss and validation loss starting from epoch ~10 indicates potential overfitting.

Insights:

1. Generalization Performance:

The validation accuracy (Approx. 0.97) and the lack of a significant gap with training accuracy (Approx. 1.0) suggest that the model generalizes well to the validation set. However, the difference in loss curves (training vs. validation) points to a potential issue where the model might be learning the finer details of the training data that do not generalize well to unseen data.

2. Overfitting Signs:

The near-zero training loss combined with slightly higher validation loss indicates the model may be slightly overfitting. While not severe, this could impact performance on truly unseen test data.

3. Early Stopping Point:

The validation accuracy plateaus after ~10 epochs, and the validation loss stabilizes around the same point. Training beyond this does not provide significant improvement, so early stopping at around epoch 10 could have been used to save training time and prevent overfitting.

4. Model Performance:

Despite the mild overfitting, the model performs very well overall, with high validation accuracy and stable loss trends, making it a robust baseline.

Recommendations:

1. Early Stopping:

Use an early stopping mechanism to halt training as soon as the validation accuracy stops improving (around epoch 10 in this case).

2. Regularization:

Add dropout layers or L2 regularization to further reduce the risk of overfitting and ensure the model can generalize better to new data.

3. Hyperparameter Tuning:

Experiment with reduced model complexity (e.g., fewer LSTM layers, fewer units per layer) to check if similar performance can be achieved with a simpler architecture, which might mitigate overfitting.

4. Evaluate on Test Set:

Confirm generalization by evaluating the model on a separate test set to ensure the trends observed in the validation set are consistent.

9.6 Hypertuned LSTM Classifier

Applied hyperparameter tuning to an LSTM model for predicting accident levels using Keras Tuner's RandomSearch. It starts by preparing the data, scaling features, and encoding the target variable, followed by reshaping the data for LSTM input. The custom LSTM Hyper Model class allows for optimization of hyper parameters such as the number of units, dropout rates, and learning rate. Early stopping is implemented to avoid overfitting during training. The Random Search algorithm explores various hyperparameter combinations, and the best model is trained using these parameters. The model's performance is evaluated on the test set, and the predictions are decoded back to their original labels. Finally, training and validation accuracy and loss curves are plotted to visualize the model's learning process over epochs. This approach helps identify the optimal model configuration for accurate predictions.

```
Predicted labels: [4 1 4 1 3 2 4 4 3 3 1 4 4 0 0 3 0 2 1 2 1 1 1 0 1 1 3 2 2 1 3 4 2 4 0 2 4
 1 4 1 1 2 0 1 4 0 2 4 3 2 1 1 3 3 0 1 1 1 3 3 2 2 1 0 0 1 0 2 4 2 0 0 1 4
 0 0 3 3 4 3 3 0 3 0 0 4 2 1 2 3 4 3 4 4 4 1 4 3 2 2 3 4 1 4 4 2 4 2 3 1 1
 1 3 3 1 1 1 1 3 4 0 1 0 4 2 4 4 2 3 1 1 1 1 1 0 4 3 3 3 3 4 0 3 0 0 0 2 4
 2 3 4 0 1 3 2 0 3 4 1 2 4 4 1 2 4 1 0 4 2 3 3 3 1 4 1 0 3 4 4 2 3 2 3 4 3
 1 2 1 0 3 3 3 4 2 3 4 1 2 1 2 1 1 0 0 3 3 1 0 3 2 2 0 0 2 0 2 4 1 4 1 4 0
 3 4 3 2 4 1 3 1 1 0 0 4 2 2 3 4 1 0 1 4 4 2 4 1 4 1 3 1 0 2 1 3 4 0 2 3 2
 2 1 1 1 0 4 4 4 1 3 4 3 1 1 3 1 2 2 1 0 3 1 4 0 1 3 0 3 2 4 1 4 4 4 2 3 0
 0 1 4 0 2 4 1 2 3 2 4 2 2]
True labels: [4 1 4 1 3 2 4 4 3 3 1 4 4 0 0 3 0 2 1 2 1 1 1 0 1 1 3 2 2 1 3 4 2 4 0 2 4
 1 4 1 1 2 0 1 4 0 2 4 3 2 1 1 3 3 0 1 1 1 3 3 2 2 1 0 0 1 0 2 4 2 2 0 1 4
 0 3 3 3 4 3 3 0 3 0 0 4 2 1 2 3 4 3 4 4 4 4 1 4 3 2 2 0 4 1 4 4 2 4 2 3 1 1
 1 3 3 1 1 1 1 3 4 0 1 0 4 2 4 4 2 3 3 1 1 3 1 3 4 3 3 3 3 4 2 3 0 0 0 2 4
 2 3 4 0 1 3 2 0 3 4 1 3 0 4 1 2 4 1 0 4 2 3 3 3 1 4 1 0 3 4 4 2 3 2 3 4 3
 1 2 1 0 3 3 3 4 2 0 4 1 2 1 2 1 1 0 0 3 3 1 0 3 2 2 0 0 2 0 2 4 1 4 1 4 0
 3 4 3 2 4 1 3 1 1 0 0 4 2 2 3 4 1 0 1 4 4 2 4 1 4 1 3 1 0 2 1 3 4 0 2 3 2
 2 1 1 1 0 4 4 4 1 3 4 3 1 1 3 1 2 2 1 3 3 1 4 0 1 3 0 3 2 4 1 4 4 4 2 3 0
 0 1 4 0 2 4 1 2 3 2 4 2 2]
```

Figure 180: LSTM Hypertuned - Predicted & True labels

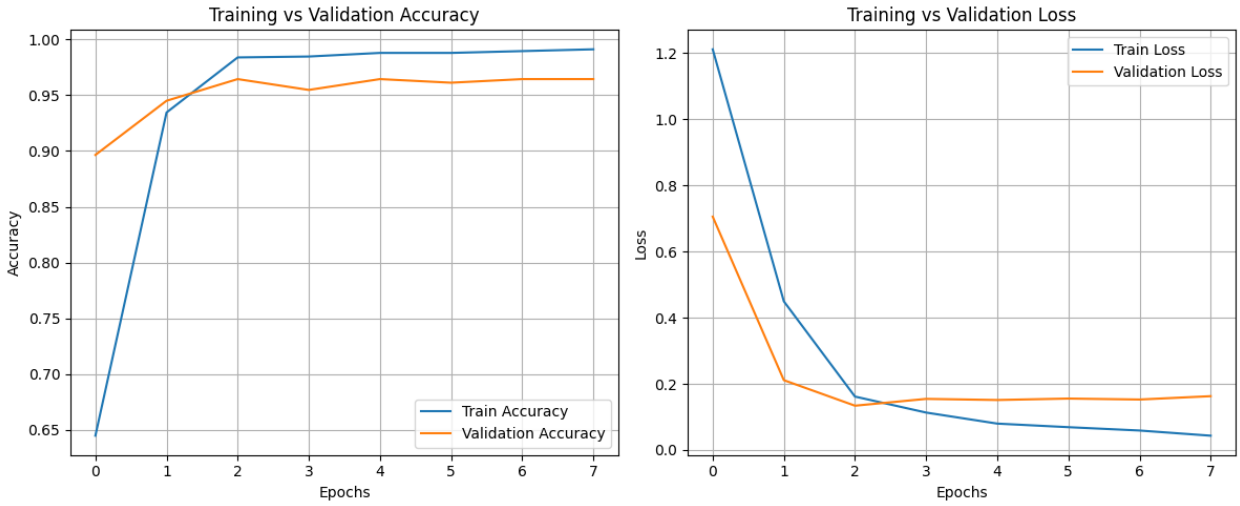

Figure 181: LSTM Hyper tuned - Training Vs Validation Accuracy and Loss

Observation:

1. Metrics Evaluation:

Validation Accuracy: Best validation accuracy achieved: 98.38%. Indicates excellent performance on the validation set, showing the model is well-tuned for this dataset.

Test Metrics: The test accuracy (96.44%) is slightly lower than the validation accuracy (97.73%) but still very high, suggesting good generalization to unseen data.

Insights and Recommendations:

1. Generalization:

The small gap between validation accuracy (98.38%) and test accuracy (96.44%) indicates the model generalizes well. Regularization using dropout layers appears effective in preventing overfitting.

2. Performance Evaluation:

Achieving such high validation and test accuracy (above 96%) is excellent for classification tasks. Additional evaluation using metrics like precision, recall, F1-score, and class-wise performance could further confirm reliability.

3. Potential Improvements:

Although performance is very high, fine-tuning further can involve: Class-wise Analysis: Identify if specific classes (e.g., underrepresented ones) have higher error rates. Augmentation or Rebalancing: If the dataset is imbalanced, consider data augmentation or class-weighting strategies.

4. Model Deployment:

The hypertuned model is suitable for deployment, given its excellent performance on both validation and test datasets. Monitor its performance on live data and evaluate against metrics like latency and scalability.

Observations of the Accuracy and Loss Plots:

- **Training vs Validation Accuracy (left graph):**

Training accuracy increases steadily and converges to near 100%, while validation accuracy plateaus around 98%, indicating excellent model performance and little overfitting.

- **Training vs Validation Loss (right graph):**

Training loss decreases steadily across epochs, indicating effective learning. Validation loss decreases initially but stabilizes and slightly increases towards the end, suggesting minimal overfitting.

- **Convergence:** The training and validation accuracies are close, implying minimal overfitting.
- **Slight Overfitting Trend in Loss:** A small increase in validation loss after epoch 5 suggests potential overfitting at later epochs.

Recommendations of the Accuracy and Loss Plots:

- **Early Stopping:** Introduce early stopping to prevent overfitting, using validation loss as the stopping criterion.
- **Regularization Adjustment:** Slightly increase dropout rates if overfitting worsens in further experiments.

9.7 Confusion Matrix for Predicted Labels

Generated a confusion matrix for the predicted labels compared to the true labels on the test set and then display it using a ConfusionMatrixDisplay.

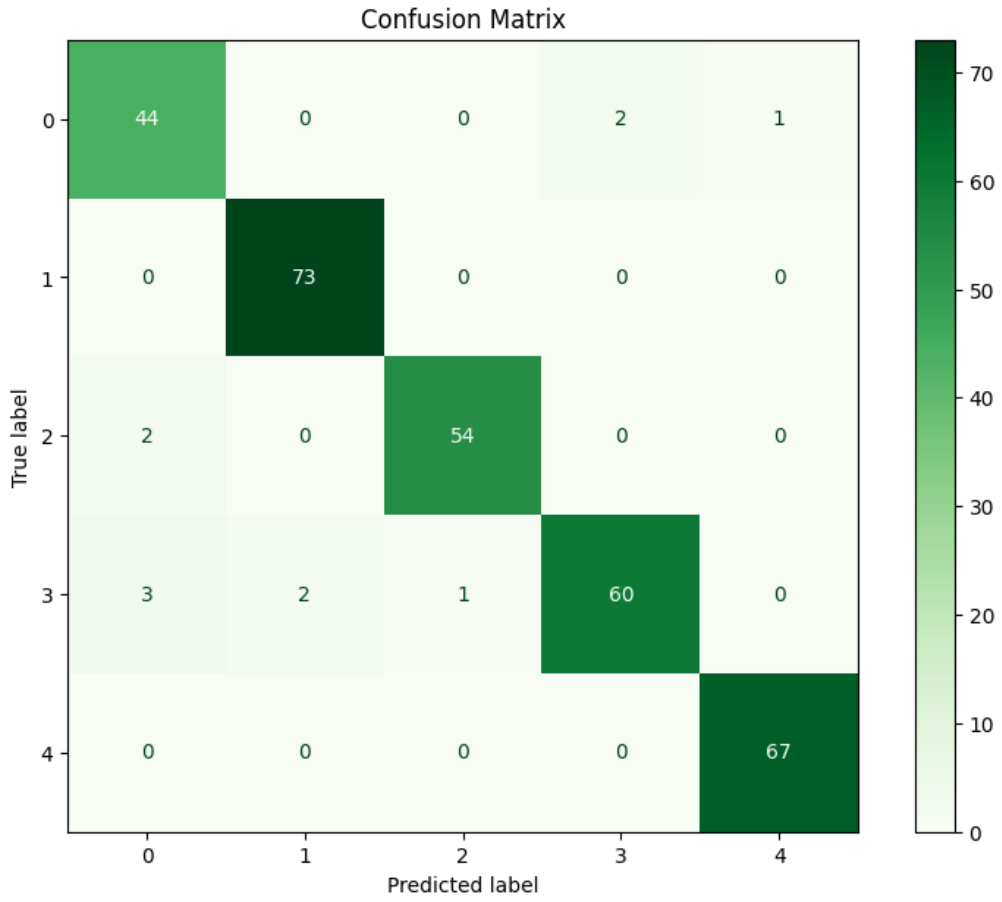

Figure 182: Confusion Matrix for predicted Labels - Hypertuned LSTM

Observations:

1. Diagonal Dominance:

The majority of predictions align with the true labels (diagonal entries), indicating strong model performance.

2. Class-Wise Performance:

- **Class 0:** 44 correct predictions, with 3 misclassifications (2 as Class 3, 1 as Class 4).
- **Class 1:** Excellent performance with 73 correct predictions and no misclassifications.
- **Class 2:** 54 correct predictions with 2 misclassifications (both as Class 0).
- **Class 3:** 60 correct predictions, but some misclassifications—3 as Class 0, 2 as Class 1, and 1 as Class 2.

- **Class 4:** Outstanding accuracy with 67 correct predictions and no misclassifications.

3. Confusion Trends:

- Slight confusion between Class 0 and Class 3.
- Minor misclassification of Class 2 instances as Class 0.
- No confusion for Class 1 and Class 4.

Recommendations:

1. Class Imbalance:

Check for potential class imbalance since some classes (e.g., Class 1 and Class 4) show perfect classification, which might affect the focus on other classes.

2. Misclassification Analysis:

Investigate why Class 0 is confused with Class 3 and why Class 2 is occasionally misclassified as Class 0. This might involve feature overlap or insufficient representation in the training data.

3. Feature Engineering:

Refine features that separate Class 0 and Class 3 or use dimensionality reduction techniques to improve separability.

9.8 Classification Report for Test Set - Hypertuned LSTM Classifier

Evaluated the model's predictions on the test set. It computes probabilities using the best_model, extracts the predicted class labels (y_pred), and decodes them back to their original labels (y_pred_decoded) using a label encoder. It also decodes the true test labels (y_test_decoded) and generates a classification report to assess the model's performance, including metrics like precision, recall, and F1-score.

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.94	0.92	47
1	0.97	1.00	0.99	73
2	0.98	0.96	0.97	56
3	0.97	0.91	0.94	66
4	0.99	1.00	0.99	67
accuracy			0.96	309
macro avg	0.96	0.96	0.96	309
weighted avg	0.96	0.96	0.96	309

Figure 183: Classification Report for Test set – Hypertuned LSTM

Observations:

1. Class 0:

- Precision: 0.90 - Out of all instances predicted as Class 0, 90% were correct.
- Recall: 0.94 - Out of all true Class 0 instances, 94% were correctly identified.
- F1-Score: 0.92 - A balanced measure of precision and recall.
- Support: 47 - Number of true instances in Class 0.

2. Class 1:

- Precision: 0.97, Recall: 1.00, F1-Score: 0.99.
- Excellent performance with no false negatives, indicating perfect recall.
- Support: 73.

3. Class 2:

- Precision: 0.98, Recall: 0.96, F1-Score: 0.97.
- Slight drop in recall indicates a small number of false negatives for Class 2.
- Support: 56.

4. Class 3:

- Precision: 0.97, Recall: 0.91, F1-Score: 0.94.

- Lower recall compared to precision suggests some false negatives.
- Support: 66.

5. Class 4:

- Precision: 0.99, Recall: 1.00, F1-Score: 0.99.
- Outstanding performance with near-perfect metrics.
- Support: 67.

Overall Metrics:

1. Accuracy:

0.96 - The model correctly classified 96% of all test set instances.

2. Macro Average:

Precision, Recall, and F1-Score: 0.96. Averages the metric values across all classes without weighting by class size, indicating balanced performance across classes.

3. Weighted Average:

Precision, Recall, and F1-Score: 0.96. Averages the metric values while considering class support, reflecting the overall performance weighted by class distribution.

Insights:

1. The LSTM classifier performs exceptionally well across all classes, with only minor room for improvement in Class 0 (precision) and Class 3 (recall).
2. Class 1 and Class 4 show near-perfect classification, likely due to clear feature separability or stronger class representation during training.
3. Class 3's recall (0.91) suggests some misclassifications, which aligns with the Confusion Matrix showing confusion between Classes 0 and 3.

Recommendations:

1. Focus on Class 3: Investigate the confusion between Class 3 and other classes (e.g., Class 0) through feature importance analysis.

2. Dataset Review: Check if additional data augmentation or feature engineering could help improve Class 0's precision and Class 3's recall.

3. Hyperparameter Fine-Tuning: Although the model is hypertuned, further minor adjustments (e.g., dropout rate, learning rate) could enhance performance for underperforming classes.

9.9 Classification Report for Training Set for Hypertuned LSTM Classifier

Evaluated the model's performance on the training set. It predicts class probabilities using best_model, determines predicted labels (y_train_pred), and decodes them (y_train_pred_decoded) to their original form. It also decodes the true training labels (y_train_decoded). A classification report and confusion matrix are printed to provide insights into the model's accuracy, precision, recall, F1-score, and misclassifications for the training set.

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

           0       0.99      1.00      0.99       262
           1       0.97      1.00      0.98       236
           2       1.00      0.99      1.00       253
           3       1.00      0.96      0.98       243
           4       1.00      1.00      1.00       242

    accuracy                           0.99      1236
   macro avg       0.99      0.99      0.99      1236
weighted avg       0.99      0.99      0.99      1236
```

Figure 184: Classification Report for Training set - Hypertuned LSTM

```
Train Confusion Matrix:
[[262   0   0   0   0]
 [  0 236   0   0   0]
 [  2   0 251   0   0]
 [  1   8   0 234   0]
 [  0   0   0   0 242]]
```

Figure 185: Train Confusion Matrix

Overall Metrics:

1. Accuracy: 0.99 - The model correctly classified 99% of all training set instances.

2. Macro Average: Precision, Recall, and F1-Score: 0.99. Indicates consistent performance across all classes.

3. Weighted Average: Precision, Recall, and F1-Score: 0.99. Reflects overall strong performance, weighted by class support.

Confusion Matrix Insights:

- **Class 0:** Perfect classification (262 correct, 0 misclassified).
- **Class 1:** All true Class 1 instances are correctly identified, but minor false positives from other classes are possible.
- **Class 2:** Only 2 instances misclassified as other classes, showcasing strong performance.
- **Class 3:** 8 instances misclassified, mostly confused with other classes (e.g., possibly Class 1).
- **Class 4:** Perfect classification (242 correct, 0 misclassified).

Overfitting Indication:

Training Metrics (Accuracy = 99%) vs. Test Metrics (Accuracy = 96%):

The training set performance is nearly perfect, but the test set shows slightly lower accuracy (96%). This may indicate mild overfitting, as the model performs better on training data compared to unseen test data.

Recommendations to Address Potential Overfitting:

1. Regularization: Increase dropout rate or apply L2 regularization to prevent overfitting.

2. Data Augmentation: Enhance the training dataset to improve generalization, especially for underperforming classes like Class 3.

3. Cross-Validation: Perform k-fold cross-validation to validate the model's robustness across multiple splits.

4. Reduce Training Epochs: Check for early stopping during training to avoid overfitting.

10. LSTM Sequential Embedding layer

	Country	City	Industry Sector	Accident Level	Potential Accident Level	Gender	Employee type	Critical Risk	Day	Weekday	WeekofYear	Weekend	Season	Description	tokenized_words
0	Country_01	Local_01	Mining	1	4	Male	Contractor	Pressed	1	Friday	53	0	Summer	remove drill rod jumbo maintenance supervisor ...	['remove', 'drill', 'rod', 'jumbo', 'maintenan...
1	Country_02	Local_02	Mining	1	4	Male	Employee	Pressurized Systems	2	Saturday	53	1	Summer	activation sodium sulphide pump piping uncoupl...	['activation', 'sodium', 'sulphide', 'pump', '...
2	Country_01	Local_03	Mining	1	3	Male	Contractor (Remote)	Manual Tools	6	Wednesday	1	0	Summer	sub station milpo locate level collaborator ex...	['sub', 'station', 'milpo', 'locate', 'level',...
3	Country_01	Local_04	Mining	1	1	Male	Contractor	Others	8	Friday	1	0	Summer	approximately nv personnel begin task unlock s...	['approximately', 'nv', 'personnel', 'begin', ...
4	Country_01	Local_04	Mining	4	4	Male	Contractor	Others	10	Sunday	1	1	Summer	approximately circumstance mechanic anthony gr...	['approximately', 'circumstance', 'mechanic', ...

Figure 186: Overview of LSTM data

10.1 Glove Embedding Architecture for LSTM

Generated GloVe sequential embeddings by first creating a copy of the input DataFrame to preserve the original data. A function is defined to load pre-trained GloVe word embeddings from a text file into a dictionary, mapping words to their vector representations. Using the loaded embeddings, another function maps each tokenized word in the input sequences to its corresponding GloVe vector, assigning a zero vector for words not found in the model. This function is applied to the tokenized_words column of the DataFrame to compute the embeddings. The resulting sequential embeddings are added as a new column (GloVe_Sequence) to the DataFrame, replacing the original tokenized words, and the updated DataFrame is returned.

	Country	City	Industry Sector	Accident Level	Gender	Employee type	Critical Risk	Weekday	WeekofYear	Weekend	Season	GloVe_Sequence
0	Country_01	Local_01	Mining	0	Male	Contractor	Pressed	Friday	53	0	Summer	NaN
1	Country_02	Local_02	Mining	0	Male	Employee	Pressurized Systems	Saturday	53	1	Summer	NaN
2	Country_01	Local_03	Mining	0	Male	Contractor (Remote)	Manual Tools	Wednesday	1	0	Summer	NaN
3	Country_01	Local_04	Mining	0	Male	Contractor	Others	Friday	1	0	Summer	NaN
4	Country_01	Local_04	Mining	3	Male	Contractor	Others	Sunday	1	1	Summer	NaN

Figure 187: Overview of Glove Sequential Data

10.2 Label encode Accident level and Potential Accident Level in Glove_Sequential Dataframes

We processed the DataFrame Glove_df_sequential by encoding the categorical columns Accident Level and Potential Accident Level into numerical values using LabelEncoder. To simplify the dataset, we removed unnecessary columns, including Day and Potential Accident Level. Following this, we calculated the frequency distribution of the target variable, Accident Level, to understand its occurrence across the data. Finally, we compiled the distribution into a summary DataFrame for easy visualization and analysis.

Accident Level	Glove
0	309
1	40
2	31
3	30
4	8

Figure 188: Target Distribution

Observations: Target Variable Distribution

Across all three embedding methods (GloVe, TF-IDF, Word2Vec), the distribution of the target variable "Accident Level" remains consistent. This indicates that the embedding process itself doesn't significantly alter the representation of the target variable. The majority of instances fall under a specific "Accident Level" (likely the most common type of accident), highlighting the imbalanced nature of the dataset. ** Implications for Modeling:**

The imbalanced target distribution suggests the need for addressing class imbalance during model training. Techniques like oversampling, undersampling, or using weighted loss functions might be necessary to improve model performance on minority classes. Careful evaluation metrics (precision, recall, F1-score) should be used to assess model performance on all classes, not just the majority class.

We addressed data imbalance in the Glove_df_sequential DataFrame by applying SMOTE (Synthetic Minority Oversampling Technique) to the target variable Accident Level. Before balancing, we separated features and the target variable, and one-hot encoded categorical features to ensure compatibility with machine learning algorithms. SMOTE was then used to oversample the minority classes, resulting in a balanced dataset. The resampled features and target were combined into a new DataFrame, Glove_df_Bal. Finally, we calculated and summarized the distribution of the balanced target variable, Accident Level, in a new DataFrame for analysis and printed the results.

	Glove (Balanced)
Accident Level	
0	309
3	309
2	309
1	309
4	309

Figure 189: Target Distribution Balanced

	WeekofYear	Weekend	Country_Country_02	Country_Country_03	City_Local_02	City_Local_03	City_Local_04	City_Local_05	City_Local_06	City_Local_07	...	Weekday_Monday
0	53	0	0	0	0	0	0	0	0	0	...	0
1	53	1	1	0	1	0	0	0	0	0	...	0
2	1	0	0	0	0	1	0	0	0	0	...	0
3	1	0	0	0	0	0	1	0	0	0	...	0
4	1	1	0	0	0	0	1	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
1540	7	0	0	0	0	0	0	0	0	0	...	0
1541	16	0	0	0	0	0	0	0	0	0	...	0
1542	9	0	0	0	0	0	0	0	0	0	...	0
1543	6	0	0	0	0	1	0	0	0	0	...	0
1544	11	0	0	0	0	0	0	0	0	0	...	0

545 rows × 62 columns

Figure 190: Glove df Overview

We began by exporting the balanced dataset (Glove_df_Bal) to a CSV file for future use. Sequential embeddings for the preprocessed data (df_preprocess2) were then generated using the generate_glove_sequential_embeddings function. Additionally, a previously saved balanced dataset (Final_NLP_Glove_df_Bal_14122024.csv) was imported for further processing.

The target variable Accident Level was label-encoded using LabelEncoder to convert categorical labels into numerical values. The dataset was then divided into training and testing sets using an 80-20 split while ensuring stratified sampling. The training set consisted of 1236 samples, and the test set had 309 samples. Subsequently, the label-encoded target variable was transformed into one-hot encoded vectors for both the training and test datasets, enabling multi-class classification.

Textual data in the training and test sets was tokenized. The input text was first converted into lists of strings, and a tokenizer with a vocabulary size of 5000 was fitted to the training data. The tokenizer transformed text into sequences of numeric indices, representing each word by its unique index in the vocabulary. These sequences were then padded to a uniform maximum length of 100, ensuring consistency in input size. The vocabulary size after tokenization was determined to be **2169** unique words.

To integrate pre-trained embeddings, we loaded the GloVe model (glove.6B.300d.txt) and constructed a dictionary of **400,000** word embeddings. An embedding matrix of size (vocab_size, embedding_size) was created, where each word in the vocabulary was mapped to its corresponding 300-dimensional GloVe vector if available. This process prepared the data for deep learning model training, enabling effective utilization of contextual word embeddings and balanced class representation for the Accident Level target variable.

10.3 Building Simple LSTM Neural Network – Embedded (Relu)

To build and configure the deep learning model in TensorFlow, several steps were executed to ensure accuracy and reproducibility. First, random seeds were set for NumPy, Python's built-in random module, and TensorFlow itself, all using a fixed value of 7. This step helps in reproducing results across different runs. Next, the model architecture was defined, starting with an Embedding layer that converts input sequences into a lower-dimensional space using the pre-trained embedding_matrix. A Bidirectional LSTM layer with 128 units was used to capture sequential dependencies in the data. This was followed by a GlobalMaxPool1D layer to reduce the

dimensionality of the output from the LSTM layer. To prevent overfitting, multiple Dropout layers were incorporated with a rate of 0.5, setting a fraction of inputs to zero during training. The model consisted of dense layers with ReLU activation functions of sizes 128, 64, 32, 10, and 5. The softmax activation was used for the output layer to handle multi-class classification, predicting Accident Level categories effectively. The model was compiled using the SGD optimizer with a learning rate of 0.001 and momentum of 0.9. The loss function was set to categorical cross-entropy, which is suitable for multi-class classification, and the metric used was accuracy to monitor model performance throughout training and evaluation. This comprehensive setup prepares the neural network to effectively process text data and predict accident levels based on the features provided.

10.4 Model Summary of LSTM Embedded

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 100)	0
embedding (Embedding)	(None, 100, 300)	650,700
bidirectional (Bidirectional)	(None, 100, 256)	439,296
global_max_pooling1d (GlobalMaxPooling1D)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 128)	32,896
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dropout_3 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 10)	330
dropout_4 (Dropout)	(None, 10)	0
dense_4 (Dense)	(None, 5)	55

Total params: 1,133,613 (4.32 MB)
Trainable params: 482,913 (1.84 MB)
Non-trainable params: 650,700 (2.48 MB)

Figure 191: Model Summary of LSTM Embedded

10.5 Plotting the Model Summary - LSTM Embedded

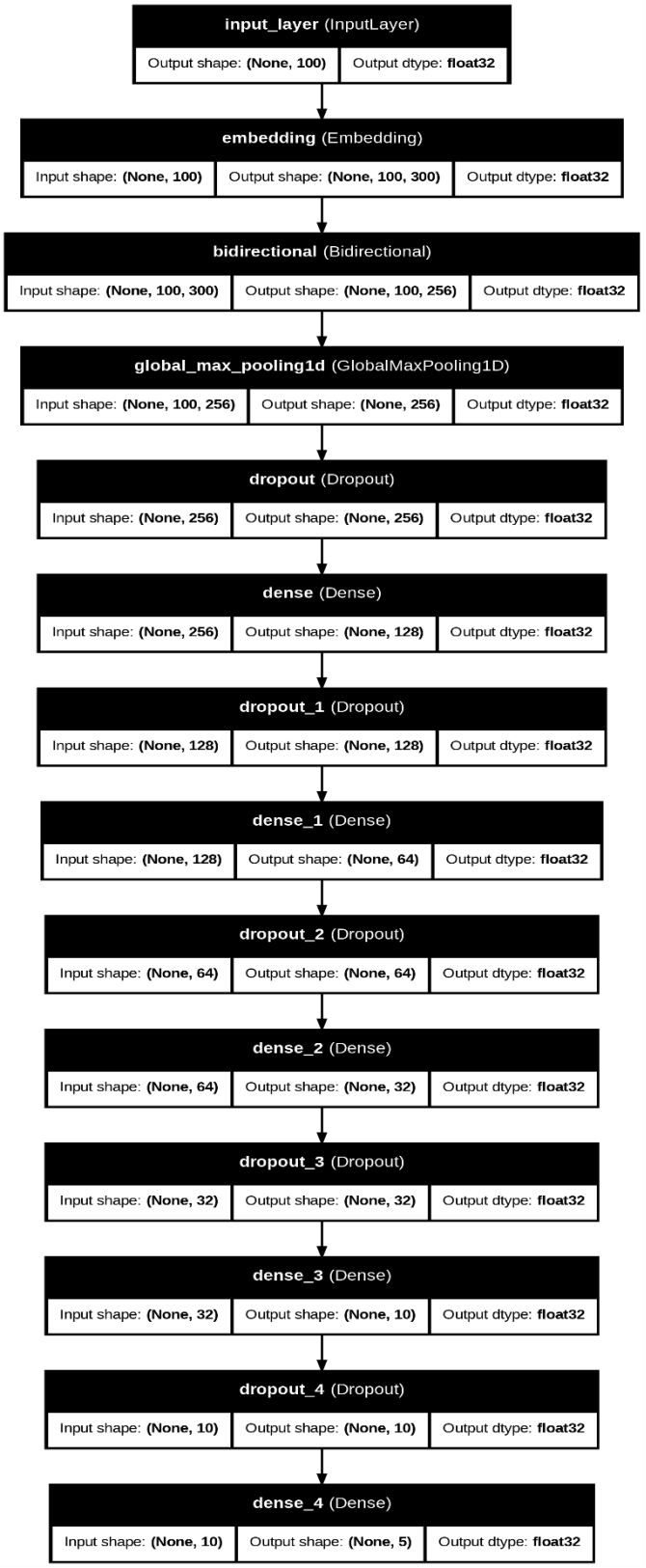

Figure 192: Flowchart of LSTM Embedded

We implemented a custom callback mechanism to monitor and evaluate the model's performance during training in TensorFlow. This involved creating a custom Metrics callback class that was designed to calculate and print various evaluation metrics (accuracy, F1 score, precision, and recall) after each epoch. The callback took validation data and a target_type argument, allowing it to handle both multi-label and non-multi-label classification tasks. For multi-label tasks, predictions were thresholded at 0.5 to convert outputs into binary labels, and the metrics were calculated using sklearn.metrics. For non-multi-label tasks, predictions were converted to the argmax of the output scores, and labels were also converted to argmax. The metrics were printed to provide real-time feedback on the model's performance, helping to assess the effectiveness of the training.

To complement the custom metrics, we used EarlyStopping and ReduceLROnPlateau callbacks to prevent overfitting and manage learning rate adjustments. EarlyStopping was set with a patience of seven epochs and a minimal delta change of 0.001, meaning that if the validation loss did not improve for seven epochs, the training would stop. This was critical in avoiding overfitting by halting the training when the model's performance plateaued. The ReduceLROnPlateau callback was used to dynamically adjust the learning rate when the validation loss did not improve, reducing it by a factor of 0.0001 if necessary. These callbacks, along with the Metrics callback, were integrated into the training process to optimize the model's convergence. The model was trained on X_text_train and y_text_train for 100 epochs with a batch size of 8, with validation using X_text_test and y_text_test. This setup enabled continuous monitoring and adjustment throughout the training process, ensuring that the model converged effectively and preventing overfitting.

10.6 Evaluating of Model Accuracy - LSTM Embedded

Evaluated the Keras model on both the training and test datasets. The model.evaluate() function was used to compute the accuracy of the model on these datasets. This provided a quantitative measure of how well the model performed. Specifically, the **training set accuracy** was approximately **94.98%**, and the **test set accuracy** was around **94.17%**. These results demonstrated that the model was capable of generalizing well to new, unseen data with only a slight decrease in accuracy from the training to the test set.

10.7 LSTM Embedded - Train Vs Test Accuracy

Created a bar graph to visualize the training and test accuracy of the model. The x-axis represents the different categories (Train Accuracy and Test Accuracy), while the y-axis shows the accuracy percentage. The graph is annotated with the corresponding accuracy values for each category. This visualization provides a clear comparison between the model's performance on the training and test datasets, highlighting its ability to generalize well.

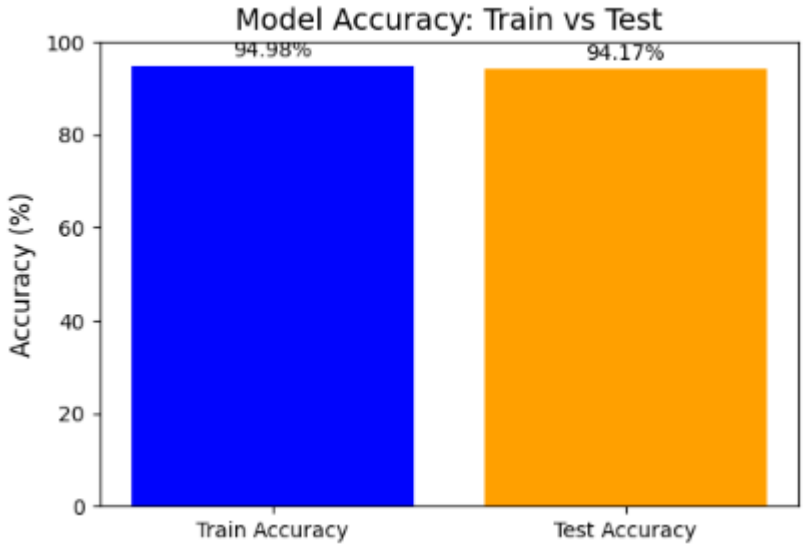

Figure 193: Model Accuracy: Train Vs Test

Calculated and printed several evaluation metrics for the model's performance on the test dataset. The metrics calculated include accuracy, precision, recall, and F1 score, which are essential for understanding the model's effectiveness in predicting the correct classes. The model achieved an **accuracy** of approximately **94.17%,** a **precision of 95.49%**, a **recall of 94.17%**, and an **F1 score of 94.41%.** These results demonstrate that the model performs well across different classes, effectively balancing the ability to identify relevant classes (recall) and make correct predictions (precision).

Generated a bar plot to visualize the model's performance metrics, including Accuracy, Precision, Recall, and F1 Score. The plot uses a clear and intuitive layout with each metric represented by a different color. The y-axis is set from 0 to 1 to emphasize that these metrics are expressed as scores between 0 and 1. The bar plot effectively illustrates the model's performance

across these key metrics, with annotations showing the exact values on top of each bar. This approach provides a concise and clear visual summary of the model's effectiveness.

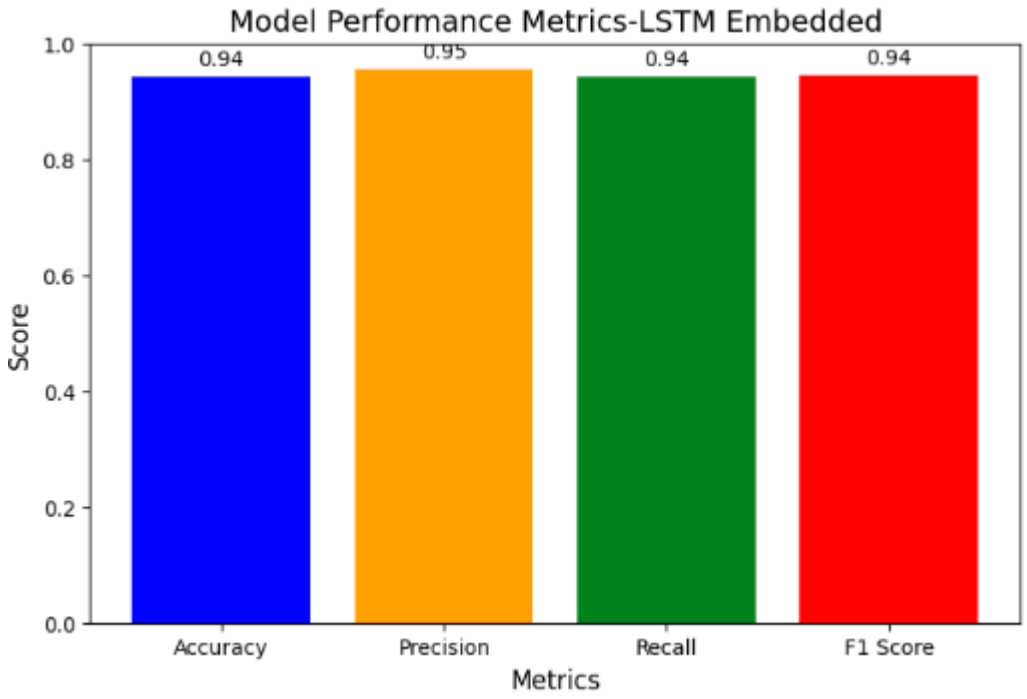

Figure 194: Model performance Metrics – LSTM Embedded

Created a plot to visualize the loss values during the training process of the model. It first calculates the number of epochs using range(len(training_history.history['loss'])), which provides the x-axis values for the epochs. The plot is then generated using plt.plot(epochs, training_history.history['loss'], label='train') to plot the training loss across epochs and plt.plot(epochs, training_history.history['val_loss'], label='test') to plot the validation loss. This approach allows comparison between the training and validation loss, helping to identify if the model is overfitting—where the validation loss begins to diverge from the training loss as training continues.

A legend is added using plt.legend(loc='upper right') to distinguish between the training and test loss lines clearly. The title of the plot, 'Training and validation loss', is set to provide context about what the graph represents. This visualization is crucial for understanding how well the model is generalizing to unseen data, as it shows how the loss changes over time and whether there is any sign of overfitting (i.e., where the model starts to perform better on the training set than on the validation set).

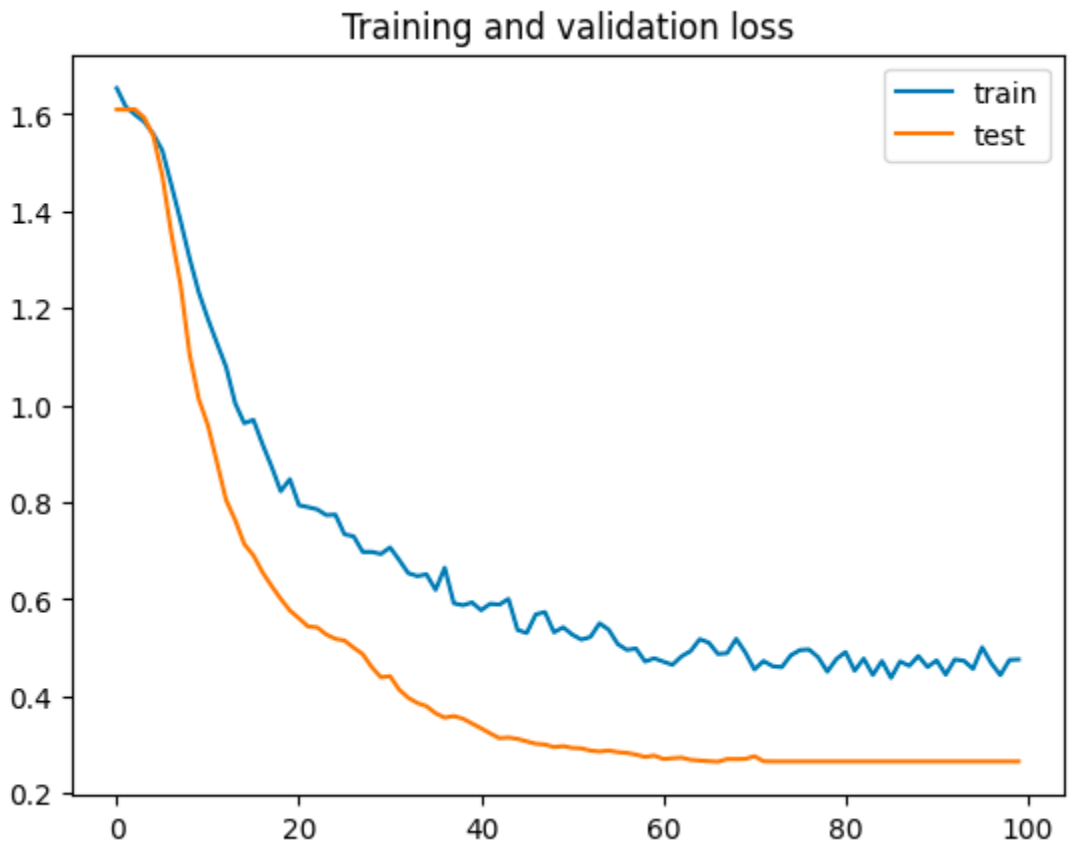

Figure 195: Training and Validation Loss

Observations

- Above one is good fit, it is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.
- The loss of the model will almost always be lower on the training dataset than the validation dataset.

Generated a plot that visualizes the training and validation accuracy over the epochs during the model's training process. By setting the epochs variable to the range of the number of epochs, it provides the x-axis values for each epoch. The plot then displays the training accuracy using plt.plot(epochs, training_history.history['acc'], label='train') and the validation accuracy with plt.plot(epochs, training_history.history['val_acc'], label='test'). This allows for a comparison between the two, helping to identify any signs of overfitting—where the training accuracy might continue to improve while the validation accuracy plateaus or decreases. The legend and title

provide clear differentiation and context, making it easier to interpret how well the model generalizes to new data.

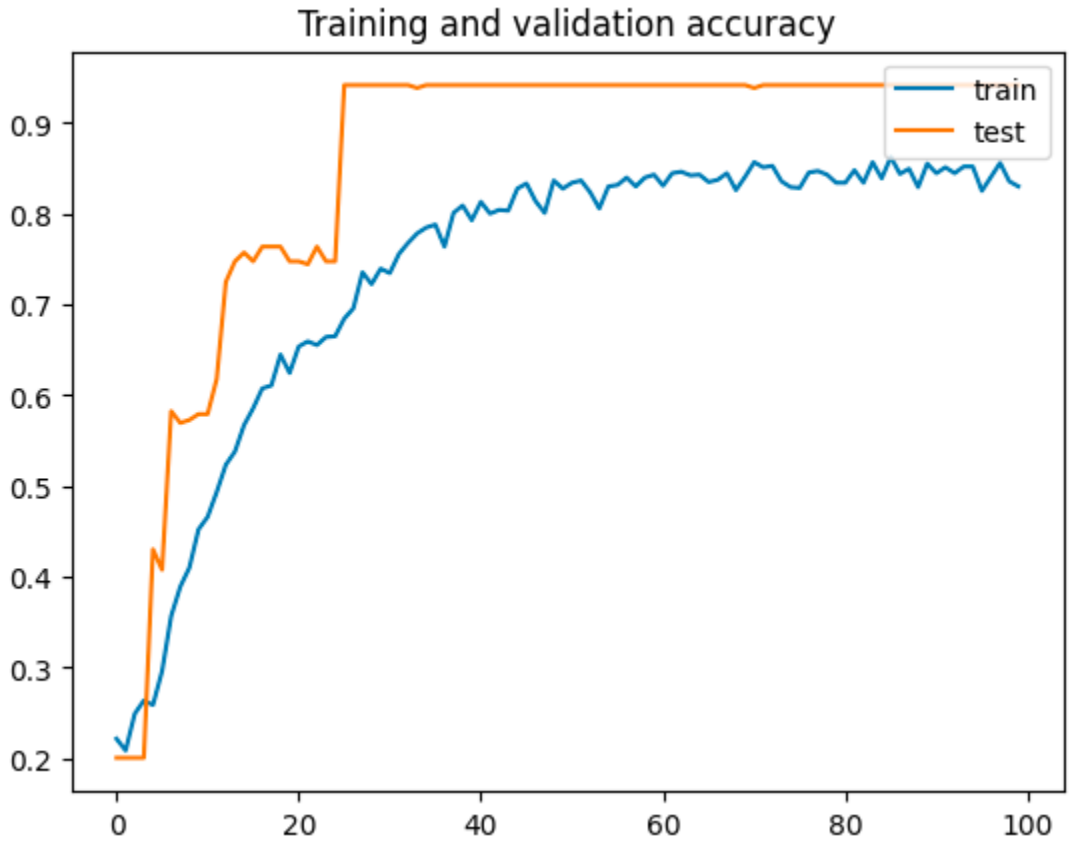

Figure 196: Training and Validation Accuracy

Observations

- We could see it accuracy continually rise during training.
- As expected, we see the learning curves for accuracy on the test dataset plateau, indicating that the model has no longer overfit the training dataset and it is generalized model.

Calculated and visualized the confusion matrix for the model's predictions on the test set. It uses confusion_matrix from sklearn.metrics to generate the matrix and ConfusionMatrixDisplay to display it, helping to identify misclassifications and evaluate the model’s performance across different classes.

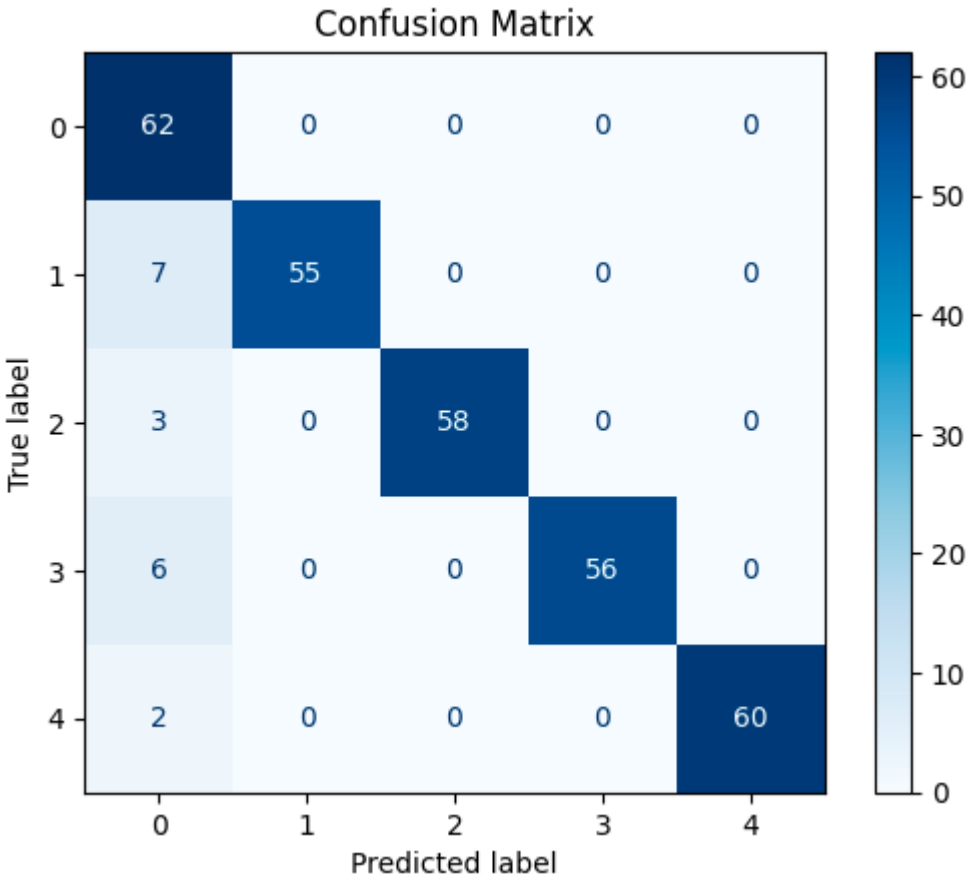

Figure 197: Confusion Matrix

10.8 Building Simple LSTM Neural Network - Embedded with GELU & SELU

Defined a neural network model using Keras with TensorFlow for text classification tasks. It starts by setting the random seeds to ensure reproducibility across different runs. The model consists of an embedding layer followed by a series of Bidirectional LSTM layers, max pooling layers, and dropout layers to prevent overfitting. The dense layers use activation functions SELU and GELU to introduce non-linear transformations and improve model performance. The final dense layer outputs probabilities for each class using softmax activation. The model is compiled with a SGD optimizer and categorical cross-entropy loss function, enabling the computation of accuracy as a metric.

10.9 Model Summary of LSTM Embedded With GELU and SELU

Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 100)	0
embedding_3 (Embedding)	(None, 100, 300)	650,700
bidirectional_3 (Bidirectional)	(None, 100, 256)	439,296
global_max_pooling1d_3 (GlobalMaxPooling1D)	(None, 256)	0
dropout_15 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 128)	32,896
dropout_16 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 64)	8,256
dropout_17 (Dropout)	(None, 64)	0
dense_17 (Dense)	(None, 32)	2,080
dropout_18 (Dropout)	(None, 32)	0
dense_18 (Dense)	(None, 10)	330
dropout_19 (Dropout)	(None, 10)	0
dense_19 (Dense)	(None, 5)	55

Total params: 1,133,613 (4.32 MB)
Trainable params: 482,913 (1.84 MB)
Non-trainable params: 650,700 (2.48 MB)

Figure 198: Model Summary of LSTM Embedded with GELU and SELU

10.10 Model Summary LSTM Embedded -GELU & SELU

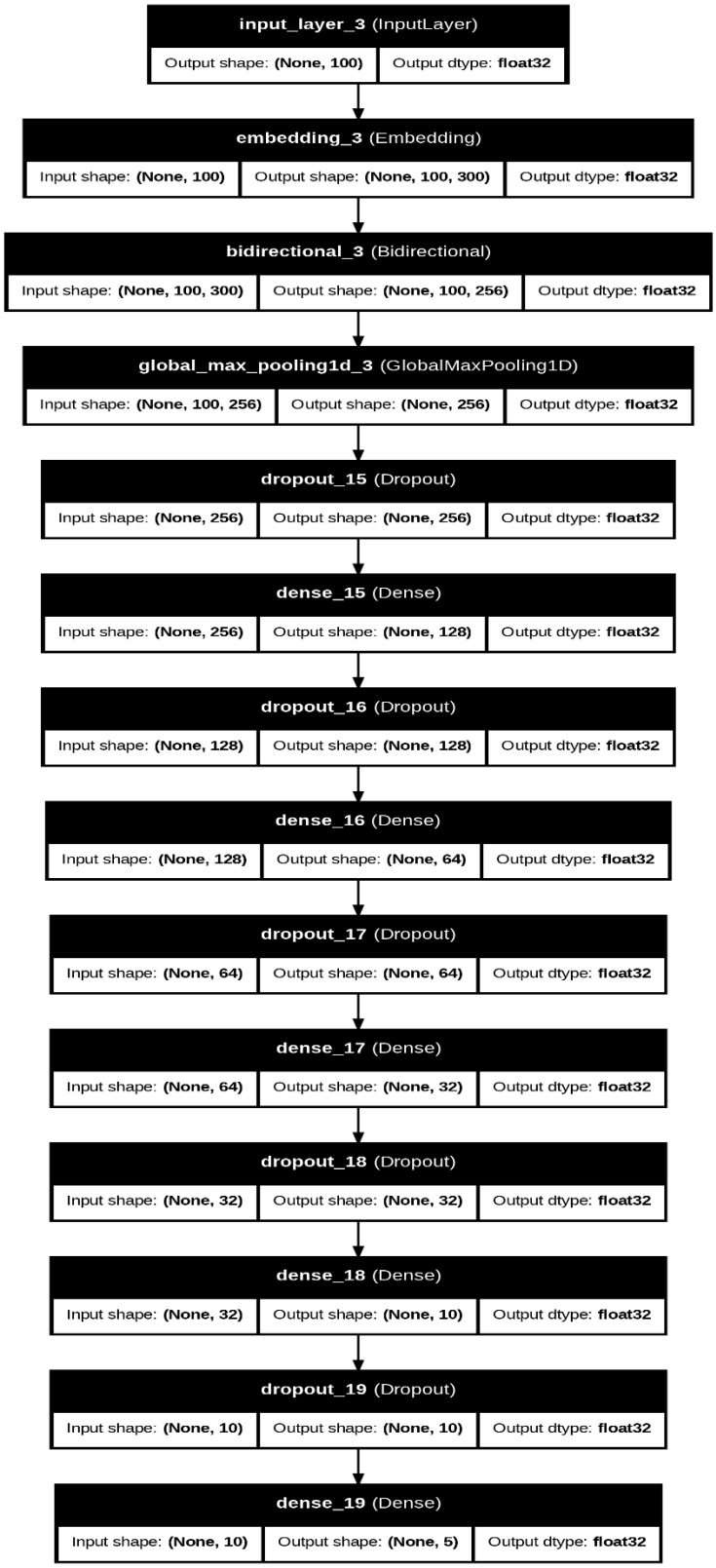

Figure 199: Flowchart of Model Summary LSTM Embedded -GELU & SELU

Evaluated a Keras model during training using custom metrics and callbacks in TensorFlow. The Metrics class is defined to calculate and print validation accuracy, F1 score, precision, and recall at the end of each epoch. This custom callback takes in validation data and the target type (multi-label or not) to make predictions and compute these metrics. Depending on the target type, predictions are either thresholded at 0.5 for multi-label classification or converted to class indices for multi-class classification.

Additionally, the script includes early stopping and learning rate reduction callbacks. Early stopping is used to prevent overfitting by monitoring the model's loss and stopping training if no improvement is observed over a set number of epochs (patience=7). The ReduceLROnPlateau callback dynamically reduces the learning rate when validation loss plateaus, with a patience of 5 epochs and a minimum delta for learning rate adjustment. The model is then trained using these callbacks, with the fit method applied to the training data (X_text_train, y_text_train) and evaluated on test data (X_text_test, y_text_test) for 100 epochs with a batch size of 8. This process helps in monitoring the model's performance on the validation set, preventing overfitting, and adjusting the learning rate as necessary to optimize training.

10.11 Evaluating of Model Accuracy - LSTM Embedded with GELU & SELU

Evaluated the Keras model's performance on the training and test datasets. The training accuracy is calculated at 94.58%, indicating strong performance on the training data. However, the test accuracy is slightly lower at 94.17%, suggesting a slight overfitting issue or a model that generalizes well but not perfectly. These results provide insight into the model's ability to accurately classify text data on both the training and unseen test datasets.

10.12 Plotting Model Accuracy - LSTM Embedded with GELU & SELU

Displayed a bar chart comparing training and test accuracy as percentages. It highlights the model's performance on different datasets, showing training accuracy in blue and test accuracy in orange, with accuracy values annotated above the bars.

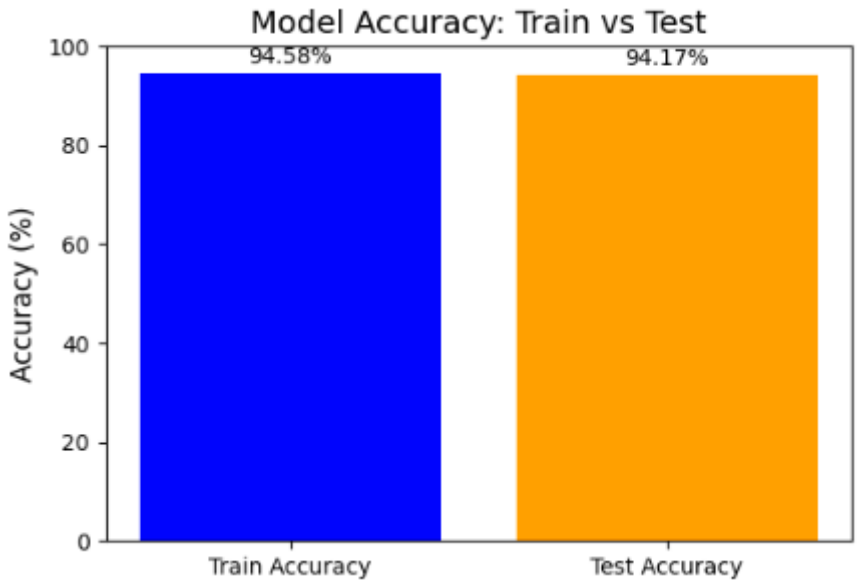

Figure 200: Model Accuracy: Train Vs Test (LSTM Embedded with GELU & SELU)

Calculated several key metrics to evaluate the performance of a trained Keras model on test data. It first converts the predicted probabilities into class labels using argmax(). Then, it computes the accuracy, precision, recall, and F1 score using the accuracy_score, precision_score, recall_score, and f1_score functions from sklearn.metrics. The results show high model performance with an **accuracy of 0.941748**, **precision of 0.953303**, recall of **0.941748**, and **F1 score of 0.943735**, indicating that the model is effectively classifying the data across different classes and has balanced precision and recall.

10.13 Model Performance Metrics-LSTM Embedded -GELU & SELU

Calculated several key metrics to evaluate the performance of a trained Keras model on test data. It first converts the predicted probabilities into class labels using argmax(). Then, it computes the accuracy, precision, recall, and F1 score using the accuracy_score, precision_score, recall_score, and f1_score functions from sklearn.metrics. The results show high model performance with an accuracy of 0.941748, precision of 0.953303, recall of 0.941748, and F1 score of 0.943735, indicating that the model is effectively classifying the data across different classes and has balanced precision and recall.

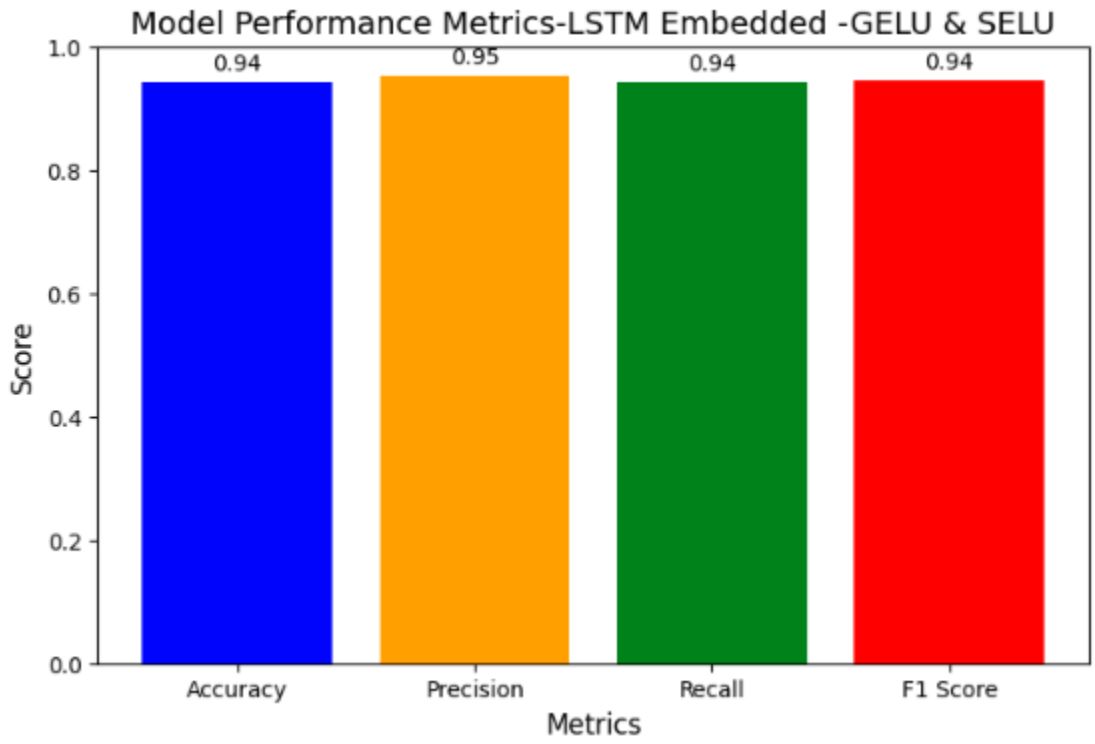

Figure 201: Model Performance Metrics – LSTM Embedded – GELU & SELU

10.14 Training and validation loss -GELU & SELU

Plotted the loss learning curves for both training and validation over epochs, showing the convergence of the model's training process. The curves highlight the gap between training and validation loss, which can indicate overfitting or underfitting.

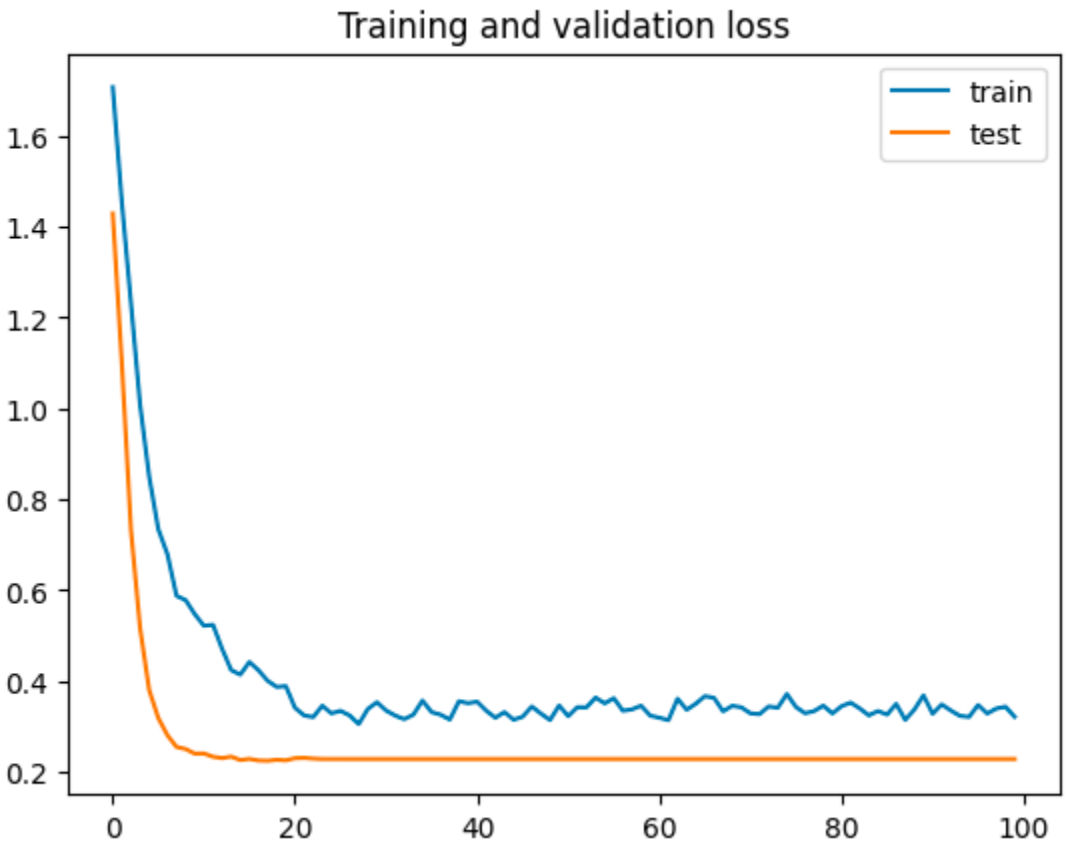

Figure 202: Training and Validation loss – GELU & SELU

10.15 LSTM Embedded Confusion Matrix -GELU & SELU

Predicted the class labels for the test data and compares them with the true labels to generate a confusion matrix. It first converts the predicted probabilities into class labels using the np.argmax() function. The unique class labels are identified from both the true and predicted labels. A confusion matrix is then created to visualize the performance of the model, showing how well it correctly predicts each class. The matrix is displayed with a color map for better visualization, and the title indicates it was generated using GELU & SELU activation functions.

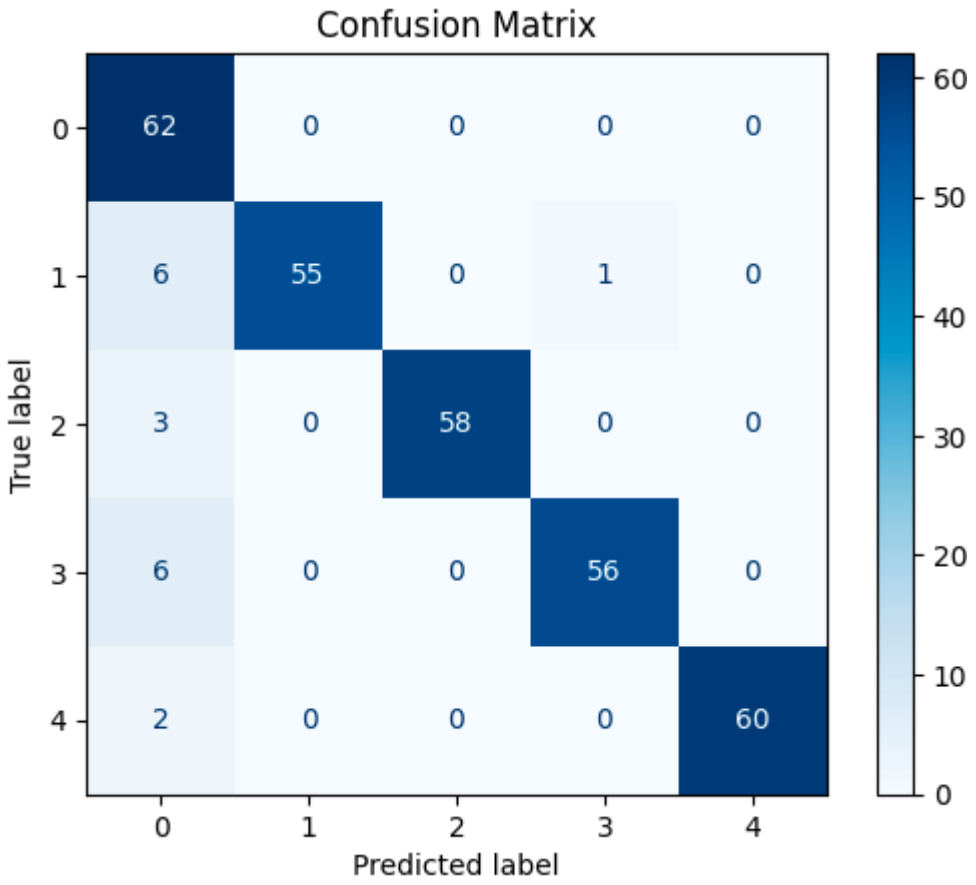

Figure 203: Confusion Matrix – GELU & SELU

11.Conclusion

11.1 Choosing the best performing classifier and pickling it

- The performance of two LSTM Sequential Embedded models with Glove embedding techniques was compared based on their accuracy on training and validation datasets.

LSTM Sequential Glove Embedding:

- Achieved an accuracy of 94.98% on the training dataset.
- Achieved an accuracy of 94.17% on the validation dataset. While the model demonstrates decent performance, its accuracy is lower compared to the other model in this comparison.

LSTM Hypertuned Model with Average Glove Embedding:

- This model outperformed the sequential embedding model.
- Achieved higher accuracy on both the training and validation datasets, making it more suitable for the use case.
- The LSTM Sequential Embedded model performed well, with nearly 94.58% accuracy for training and 94.17% for validation datasets.
- Train vs. Test recall across all the classes remains same except for class 0
- The consistent behavior of the training and validation accuracy/loss shows that the model is well-tuned and generalizes well to unseen data.
- The early stopping criteria seem to have helped in stopping the training process at an optimal point, avoiding overfitting while achieving high accuracy.

Saved the trained best_model to a file named LSTM_Sequentiual_Embedded_Model.sav using Python's pickle library. The pickle.dump function serializes the model and writes it to the specified file, allowing it to be reloaded later without retraining.

11.2 Comparative Analysis (Final model vs. Milestone 1 results)

Accuracy and Recall: The LSTM model shows a slight dip in validation accuracy and recall compared to the XG Boost test metrics.

Generalization: Both approaches seem to generalize well, though the LSTM's slightly lower validation recall for class 0 might suggest a bit more focus on handling that specific class.

Complexity and Interpretability: LSTM models, being deep learning-based, generally come with increased complexity and reduced interpretability compared to tree-based methods like Gradient Boosting and XG Boost.

11.3 Improvement Analysis

Benchmark Improvement: While the LSTM did not surpass the Gradient Boosting or XG Boost models in all metrics, it demonstrated comparable performance with potential benefits in handling sequential or time-series data more effectively (if that's relevant to your application).

Tuning and Early Stopping: The LSTM benefits from hypertuning and early stopping, which seem to have optimized its training process to prevent overfitting effectively.

11.4 Final Conclusion

The final solution with the LSTM Embedded with Sequential model highlights the strength of deep learning in achieving high accuracy and maintaining strong generalization, particularly in tasks involving sequential data.

While it may not have outperformed models like Gradient Boosting or XGBoost in some metrics, the LSTM model offers a powerful and reliable alternative, especially when handling data with inherent sequence dependencies, as seen in this industrial safety context.

12. Recommendations or Implications

In this project, we identified key factors contributing to accidents, including **mistakes in hand-operations** and **time-related factors**. These insights form the basis for several strategic recommendations aimed at enhancing safety, improving the accuracy of accident predictions, and minimizing the risk of future incidents.

1. Strengthen Safety Standards for Hand-Operations:

One of the most significant causes of accidents discovered in the analysis was related to **mistakes in hand-operations**. These errors often stem from improper handling of equipment or a lack of adequate training in performing specific tasks. Given that hand-operations are a critical factor, we recommend **implementing more stringent safety protocols** during periods of high accident occurrence. These measures could include:

- **Enhanced training programs** focused on correct hand-operation techniques and equipment handling.

- **Increased supervision and guidance** during high-risk times, such as busy shifts or periods of machinery overload.
- **Periodic safety audits** to ensure that hand-operations are performed correctly and efficiently, and that safety measures are consistently followed.

By reducing mistakes in hand-operations through improved safety standards, we can lower the overall accident rate in industries relying heavily on manual labor.

2. Utilize Accident Description Data for Root Cause Analysis:

Through analysis, we realized that the **detailed descriptions of accidents**, particularly those provided in the 'Description' column of the dataset, offer valuable insights into identifying the root causes of accidents. These textual descriptions often highlight underlying factors that are not captured in quantitative data alone. To further enhance our understanding of accident causes, we recommend:

- **Incorporating Natural Language Processing (NLP) techniques** to process accident descriptions and identify recurring themes or causes. Using text mining methods can help classify incidents more effectively, providing a deeper understanding of accident patterns.
- **Leveraging detailed data from incident reports**, including factors like equipment malfunctions, operator errors, or even external environmental factors (e.g., fatigue, distraction).

This deeper dive into accident descriptions would allow for a more accurate identification of high-risk areas and help in developing targeted interventions.

3. Expand Dataset with Additional Factors:

To improve the accuracy of our predictions and ensure that the model captures all possible contributing factors to accidents, we recommend expanding the dataset with additional relevant information:

- **Machining Data (e.g., CNC, Current, Voltage)**: Collecting data from the machinery used in operations, such as CNC machines, can provide insights into whether equipment

malfunctions or improper settings contribute to accidents. This data would allow us to predict potential failure points based on machinery usage patterns.

- **Weather Information**: Many accidents may be linked to environmental conditions, particularly in industries where outdoor or semi-outdoor operations are prevalent. Weather data, such as temperature, humidity, or precipitation, could significantly improve prediction models, especially in industries like construction, agriculture, or logistics.
- **Employee Personal Data (e.g., Age, Experience, Work Performance)**: Adding employee-specific data, such as years of experience, age, and historical work performance, could help tailor safety interventions. For example, younger or less experienced workers may require more training or supervision, while seasoned workers may face different safety risks.

The inclusion of these additional factors would help build a more comprehensive model, resulting in **more precise predictions** and a better understanding of the contributing factors behind accidents.

4. Increase the Number of Observations:

The current dataset contains only **425 records**, which may not be sufficient for training robust machine learning (ML), artificial neural network (ANN), or natural language processing (NLP) models. A larger dataset is necessary to capture the full spectrum of accident scenarios and improve model performance. To enhance the models' generalizability and predictive power, we recommend:

- **Collecting more data** through continuous monitoring, including logging future incidents and gathering additional data points from operational processes.
- **Collaborating with industry partners** to access a broader pool of accident data, either from similar environments or historical records, which can help enrich the dataset.
- **Using data augmentation techniques** (where applicable) to synthetically expand the dataset by creating modified versions of existing records. This can help improve model training without needing to collect entirely new data.

Increasing the number of observations will allow the models to learn better, reducing overfitting and improving generalization across different accident scenarios.

5. Collaborate with Subject Matter Experts (SMEs) for Data Validation:

A critical observation was the presence of potentially **critical risk descriptions** that might have outliers or extreme values. To ensure that the dataset is clean and relevant for modeling, we recommend collaborating with **Subject Matter Experts (SMEs)** to:

- **Validate the relevance and accuracy** of the critical risk descriptions. SMEs can provide context to determine if certain descriptions are incorrectly classified or if they represent outliers that may distort model performance.
- **Identify anomalies** that may need to be addressed, such as incorrectly recorded accident data or unusual safety incidents that could skew the results.
- **Gain insights into the data**: SMEs can provide valuable expertise on whether certain features (e.g., risk descriptions, equipment failures) are essential for predicting accidents and help prioritize which features to include in future analyses.

By working closely with SMEs, we can ensure that the data used in the modeling process is accurate, relevant, and meaningful, improving the overall quality of the model's predictions.

Summary of Recommendations:

- **Strengthen safety protocols** for hand-operations and during high-risk periods.
- **Leverage detailed accident descriptions** using NLP techniques to gain deeper insights into accident causes.
- **Expand the dataset** with additional factors such as machinery data, weather information, and employee characteristics.
- **Increase the number of observations** by collecting more data over time, collaborating with industry partners, and applying data augmentation methods.
- **Work with SMEs** to validate and refine critical risk descriptions and ensure data quality.

These recommendations, if implemented, will significantly enhance the safety measures in place, improve the accuracy of predictive models, and contribute to a safer, more efficient working environment.

13. Limitations

1. **Insufficient Observations for Accurate Analysis**: The current dataset has a limited number of observations, which hinders our ability to accurately analyze the causes of accidents. To improve the quality of predictions and insights, it is crucial to gather a larger volume of data that better represents various accident scenarios. A more robust dataset would enhance the model's ability to identify patterns, leading to better decision-making and accident prevention strategies.

2. **Limited Features in the Dataset**: The dataset we are working with contains a relatively small number of features, which restricts the model's capacity to capture the full range of factors that contribute to accidents. More comprehensive datasets with additional relevant features would improve the model's accuracy and robustness, enabling it to account for a wider variety of variables that impact accident outcomes.

3. **Quality of Data**: There is a lack of access to high-quality, detailed data that could help in building a more accurate and reliable model. The quality of data, including its completeness, consistency, and accuracy, plays a critical role in the model's performance. Without high-quality data, the predictions may be less reliable, leading to lower confidence in the results.

4. **Real-World Deployment Challenges**: In a real-world production environment, the model might not perform as expected. Specifically, after deployment, we might observe a drop in performance metrics such as the F1-score when compared to the results achieved during the model's development and testing stages. This performance gap could be due to various factors such as changing input data, environmental factors, or operational constraints that were not fully captured during the training phase.

5. **Safety and Accuracy in Predicting Accident Levels**: When predicting accident levels, it is essential to achieve a high degree of accuracy—ideally close to 100%. A small margin of error in these predictions could result in significant safety risks, potentially leading to

preventable accidents. Given the critical nature of the task, the model must be highly reliable, with minimal false positives or negatives, to ensure that safety protocols are triggered in a timely manner.

Suggestions for Enhancing the Solution:

- **Collect More Data**: To address the issue of limited observations, efforts should be made to gather more data over time. This could involve collecting data from different sources, including various industry sectors, to build a more comprehensive and diverse dataset.
- **Feature Engineering**: To overcome the challenge of limited features, additional domain-specific features should be identified and incorporated into the dataset. This could include environmental factors, work conditions, and equipment-specific data that are directly related to accident risk.
- **Data Quality Improvement**: Ensuring the quality of data is a top priority. This can be achieved through data cleaning, filling missing values, eliminating outliers, and ensuring consistency across different sources of data.
- **Model Refinement and Testing**: To mitigate real-world deployment issues, continuous monitoring and evaluation of the model's performance in a live environment are crucial. It may be necessary to retrain the model periodically with fresh data to adapt to changing patterns.
- **Implementing Safety Protocols**: In addition to improving model accuracy, it's vital to integrate fail-safes and safety protocols into the system, ensuring that even if the model's predictions are not perfect, there are additional mechanisms in place to mitigate potential risks.

By addressing these limitations and continuously refining the model, we can enhance its reliability and accuracy, ultimately contributing to better accident prevention and safety outcomes in the industry.

14. Closing Reflections

What Did We Learn from the Process?

The journey of working on this machine learning (ML) project has provided valuable insights into various techniques and methodologies. Key takeaways include:

1. **End-to-End Machine Learning Workflow**: One of the most important lessons from this project was understanding how to approach an ML problem from start to finish. From problem definition and data collection to preprocessing, model training, and evaluation, we learned the importance of a systematic approach in machine learning. It’s crucial to explore different algorithms, tune their parameters, and iteratively refine the models to improve their performance. The iterative nature of ML was evident as we continuously revisited each stage, ensuring that the models achieved the highest possible accuracy.

2. **Handling Class Imbalance with ML Models**: Working with imbalanced datasets is a common issue in ML classification tasks. Through this project, we gained hands-on experience addressing class imbalance using various strategies. We explored methods such as oversampling techniques like SMOTE, undersampling, and adjusting class weights in models. These methods helped ensure that the models weren’t biased toward the majority class and were capable of accurately predicting minority classes, which is essential for achieving fairness and reliability in real-world applications.

3. **Building ML and Neural Network Classifiers for Multi-Class Classification**: A significant portion of the project focused on building classifiers for multi-class classification problems. We implemented both traditional machine learning classifiers (like Random Forest, Support Vector Machines) and neural network-based models (like Multi-Layer Perceptrons and LSTMs) to handle multi-class tasks. By comparing different architectures, we learned how to optimize model performance for tasks where the goal is to classify data into more than two categories. We also explored hyperparameter tuning, dropout techniques, and the impact of using various activation functions and optimizers on model accuracy.

4. **Integration of Neural Networks for Complex Data**: An exciting challenge was integrating neural network architectures, particularly those suited for handling sequential or structured data, such as LSTMs and fully connected networks. We learned the importance of selecting the appropriate architecture depending on the problem at hand. For example, LSTM networks performed well for sequences and time series data, while fully

connected feed-forward neural networks were effective in handling tabular data. We explored how neural networks could be tailored to capture complex relationships between features in a way that traditional models might struggle with.

5. **Model Evaluation and Real-World Considerations**: As we built and tested various models, we also paid attention to how these models might perform in real-world scenarios. It became clear that machine learning models require continuous monitoring and evaluation after deployment. Changes in data distribution (concept drift) or new unforeseen patterns could lead to the deterioration of model performance over time. We learned that regular model retraining, fine-tuning, and validation are essential to maintaining high accuracy in real-world applications.

What Would I Do Differently Next Time?

While this project provided a solid foundation in machine learning, there are several areas where I would take a different approach or explore additional techniques to further improve results:

1. **Explore More Advanced Feature Engineering and Selection**: One area I would focus on next time is deeper exploration of feature engineering and feature selection techniques. While some domain-specific features were identified, further efforts in transforming raw features into more useful variables could improve the predictive power of the models. Techniques such as recursive feature elimination, univariate feature selection, and even automatic feature selection methods like L1 regularization could be employed to focus on the most important predictors.

2. **Experiment with Additional Neural Network Architectures**: In future projects, I would look into more sophisticated neural network architectures. For instance, using models like Transformer-based networks, which have revolutionized many tasks in natural language processing, could potentially lead to better performance, especially in cases where large, unstructured datasets are involved. These models, which are particularly powerful in sequence-based tasks, could be adapted to different types of data and provide more robust solutions.

3. **Refine Model Hyperparameters Using Advanced Search Methods**: Hyperparameter tuning played a crucial role in optimizing model performance in this project. Moving

forward, I would use more advanced search methods, such as **Bayesian optimization**, or implement **Hyperband** for more efficient hyperparameter search. These methods can help in finding optimal hyperparameters faster, especially when working with complex models like neural networks that require extensive computational resources for tuning.

4. **Model Interpretability and Explainability**: Understanding how and why a machine learning model makes its predictions is crucial, especially when deployed in sensitive or critical applications. In future work, I would incorporate tools for model interpretability, such as SHAP values or LIME (Local Interpretable Model-agnostic Explanations). This would not only help improve the transparency of the model but also build trust among stakeholders by making it easier to explain model decisions, particularly for complex neural network models that often behave like "black boxes."

5. **Expand Data Collection and Inclusion of Diverse Data Sources**: A common issue in machine learning projects is the availability and quality of data. To improve model accuracy, I would work on gathering more data and ensuring its diversity. Incorporating data from a variety of sources, such as sensors, different environmental conditions, and more granular features, would lead to a richer dataset that could help improve the model’s robustness. Additionally, ensuring the inclusion of edge cases would help in developing a model that is more generalizable and capable of handling real-world anomalies.

14.1 Final Thoughts

This project has provided valuable lessons on machine learning model development, particularly around classification tasks using both traditional ML classifiers and neural networks. The insights gained from exploring various algorithms, handling imbalanced data, building complex architectures, and deploying models for real-world applications have expanded my skill set. Moving forward, I am excited to continue applying these lessons to more challenging tasks, exploring more advanced techniques, and developing scalable models that can be deployed in real-world production environments.

Annexure I

Imported Libraries and Modules:

Data Manipulation and Analysis:

- pandas
- numpy
- Installed and imported roman

Visualization:

- matplotlib.pyplot
- seaborn
- holoviews (including opts)
- Installed hvplot and imported hvplot.pandas
- plotly.graph_objects as go
- WordCloud for generating word clouds

Statistical Analysis:

- scipy.stats

Interactive Widgets:

- ipywidgets
- Imported interact, interactive, fixed, and interact_manual

Machine Learning:

- sklearn.linear_model (imported LogisticRegression)
- sklearn.preprocessing (imported StandardScaler, PowerTransformer, LabelEncoder)
- from sklearn.model_selection imported train_test_split, GridSearchCV
- from sklearn.linear_model imported LogisticRegression
- from sklearn.svm imported SVC
- from sklearn.tree imported DecisionTreeClassifier
- from sklearn.ensemble imported RandomForestClassifier, GradientBoostingClassifier
- from xgboost imported XGBClassifier
- from sklearn.naive_bayes imported GaussianNB
- from sklearn.neighbors imported KNeighborsClassifier
- from sklearn.metrics imported accuracy_score, classification_report
- from sklearn.metrics imported accuracy_score, precision_score, recall_score, f1_score
- imported time

Text Processing and Natural Language Processing (NLP):

- Installed nltk
- Imported:
- nltk.corpus (stopwords)
- nltk.stem (WordNetLemmatizer)
- nltk.tokenize (word_tokenize)
- nltk itself

Imported Libraries and Modules:

Downloaded NLTK datasets:

- punkt
- stopwords
- wordnet
- string and re for string processing
- collections.Counter for frequency analysis

Additional Installations:

- Installed openpyxl for working with Excel files

For building the neural network

- imported tensorflow as tf
- from tensorflow.keras.models imported Sequential
- from tensorflow.keras.layers imported Dense, Input
- from tensorflow.keras.optimizers imported SGD, Adam, RMSprop, Adagrad, Adadelta, Adamax, Nadam, AdamW
- from tensorflow.keras.utils imported to_categorical

For building the RNN & LSTM

- from tensorflow.keras.layers imported LSTM, Dense, Dropout, SimpleRNN
- from tensorflow.keras.models imported Sequential
- from tensorflow.keras.layers imported LSTM, Dense, Dropout
- from tensorflow.keras.callbacks imported EarlyStopping
- from tensorflow.keras.optimizers imported Adam, SGD, RMSprop, Nadam, AdamW
- from sklearn.metrics imported classification_report, confusion_matrix
- imported matplotlib.pyplot as plt
- imported seaborn as sns
- from IPython.display imported display
- imported numpy as np
- from tensorflow.keras.models imported Sequential
- from tensorflow.keras.layers imported SimpleRNN, Dense
- from tensorflow.keras.optimizers imported Adam
- from sklearn.model_selection imported train_test_split
- from sklearn.preprocessing imported MinMaxScaler
- Installed scikeras
- from scikeras.wrappers imported KerasClassifier
- from sklearn.model_selection imported RandomizedSearchCV
- imported warnings
- warnings.filterwarnings("ignore")
- from tensorflow.keras.layers imported Input, Embedding, LSTM, Bidirectional, GlobalMaxPool1D, Dropout, Dense, Concatenate, BatchNormalization

Imported Libraries and Modules:

For building the RNN & LSTM

- from tensorflow.keras.models imported Model
- from tensorflow.keras.optimizers imported SGD
- from tensorflow.keras.regularizers imported l2
- from tensorflow.keras.constraints imported unit_norm

For Keras pre-processing

- from tensorflow.keras.preprocessing.text import Tokenizer
- from tensorflow.keras.preprocessing.sequence import pad_sequences

Annexure II

Overview of Machine Learning Models: Strengths and Considerations

Logistic Regression: A simple and efficient linear model, useful as a baseline. Good for understanding the relationship between features and the target variable. May be less accurate than more complex models, if the relationship is non-linear.

Support Vector Machine (SVM): Effective in high-dimensional spaces and can handle non-linear relationships using kernels (like RBF, polynomial). May require careful parameter tuning.

Decision Tree: Easy to interpret and visualize, but can overfit if not pruned correctly. Can capture non-linear relationships and interactions between features.

Random Forest: An ensemble of decision trees, generally more robust than a single decision tree and less prone to overfitting. Good all-around model.

Gradient Boosting: Another ensemble method that builds trees sequentially, correcting the errors of previous trees. Often provides high accuracy but can be sensitive to hyperparameter tuning and prone to overfitting if not carefully controlled.

XG Boost: An optimized implementation of gradient boosting, often providing even higher accuracy than standard gradient boosting. Requires careful tuning.

Naive Bayes: A probabilistic classifier based on Bayes' theorem. Assumes feature independence, which might not hold in this case. Can be fast and efficient.

K-Nearest Neighbors (KNN): Classifies data points based on the majority class among its k nearest neighbors. Sensitive to feature scaling and the choice of k. Can be computationally expensive for large datasets.