Csc:8980 Topics in Computer Science
Natural Language Processing
Project Report

TITLE : Design and implementation of a newspaper and magazine classification system - Analyzing Text Classification Performance using Machine Learning Models with BERT Embeddings

TEAM MEMBERS

Dinesh Kumar Kummara (002768558)
Sri Haneesha Davuluri (002804845)
Devaki Bolleneni (002778218)

ABSTRACT

This project focuses on the design and implementation of a newspaper and magazine classification system leveraging advanced machine learning (ML) techniques. We analyze text classification performance using BERT embeddings, coupled with five prominent ML models: Support Vector Machines (SVM), k-Nearest Neighbors (KNN), Random Forest, Gradient Boosting, and Logistic Regression. The dataset comprises a diverse range of news articles categorized into nine distinct categories. Our objective is to evaluate the efficiency of these ML models in accurately classifying news articles across these categories, thereby enhancing the automation and efficiency of content classification in media platforms.

INTRODUCTION

The rapid growth of digital content in newspapers and magazines has necessitated efficient text classification systems to organize and categorize diverse articles effectively. In this context, our project delves into the design and implementation of a sophisticated newspaper and magazine classification system, leveraging cutting-edge machine learning (ML) techniques. Our primary focus is on analyzing text classification performance using BERT embeddings, a state-of-the-art method known for its contextual understanding of language. The dataset utilized in our project comprises a wide array of news articles categorized into nine distinct categories: 'POLITICS', 'SPORTS', 'TECH', 'ENVIRONMENT', 'CRIME', 'BUSINESS', 'ENTERTAINMENT', 'COMEDY', and 'WELLNESS'. These categories represent the diverse spectrum of topics commonly found in newspapers and magazines, ranging from political affairs and sports events to technological innovations and environmental issues.

To assess the effectiveness of our classification system, we have employed a comprehensive analysis encompassing five prominent ML models: Support Vector Machines (SVM), k-Nearest Neighbors (KNN), Random Forest, Gradient Boosting, and Logistic Regression. Each of these models offers unique capabilities in handling text data and making accurate category predictions. By evaluating these models across the nine specified categories, we aim to identify the most suitable ML approach for newspaper and magazine classification tasks.

PROBLEM STATEMENT

The problem at hand revolves around the need for an effective and efficient newspaper and magazine classification system in the era of digital content proliferation. Traditional manual categorization methods are labor-intensive, time-consuming, and prone to errors, especially when dealing with large volumes of diverse articles. This project addresses the challenge of automating the classification process using advanced machine learning (ML) techniques, specifically focusing on text classification tasks in the context of newspapers and magazines.

With the proliferation of online news sources, readers face information overload, making it challenging to find relevant content quickly. A well-classified news platform enhances the user experience by providing tailored content recommendations based on user interests and preferences. Effective categorization helps news organizations and content aggregators organize their content repositories, making it easier to manage and retrieve information. The integration of machine learning models, particularly those leveraging BERT embeddings, offers a sophisticated approach to text classification, improving accuracy and efficiency. Accurate classification of news articles enables better decision-making processes in areas such as content recommendation systems, targeted advertising, and trend analysis.

OBJECTIVES AND SCOPE

The main objective of this project is to develop an automated newspaper and magazine classification system using BERT embeddings and machine learning algorithms like SVM, KNN, Random Forest, Logistic Regression and Gradient Boosting. We also evaluate the performance of SVM, KNN, Random Forest, Logistic Regression and Gradient Boosting for text classification tasks, focusing on accuracy, efficiency, and scalability.
BERT embeddings are utilized to enhance semantic understanding and improve classification accuracy across predefined categories or topics. The system's scalability, computational efficiency, and classification accuracy are assessed across diverse textual content types from newspapers and magazines.

The main focus is on classifying articles, reports, and editorials from newspapers and magazines into predefined categories such as politics, sports, entertainment, and technology. The machine Learning Models are implemented and compared with BERT embeddings for accurate text classification. Use standard metrics like accuracy, precision, recall, and F1-score to evaluate the system's performance in handling large volumes of textual data. Aim to create a scalable and efficient classification system applicable to real-world scenarios, ensuring usability and practicality for diverse content types in newspapers and magazines.

APPROACH

Data Collection and Preprocessing:

- The dataset was obtained from Kaggle, consisting of a collection of news articles across various categories relevant to newspapers and magazines. We have considered 500 records for each of the nine categories. The dataset's size and composition provided a diverse and representative sample for analysis.
- Preprocessing steps were applied to the dataset, including text cleaning, lowercasing, punctuation removal, tokenization, stop words removal, and lemmatization. These steps were crucial in preparing the text data for modeling by ensuring consistency and removing noise.

Category Selection and Sampling:

- Specific categories, including 'POLITICS', 'SPORTS', 'TECH', 'ENVIRONMENT', 'CRIME', 'BUSINESS', 'ENTERTAINMENT', 'COMEDY', and 'WELLNESS', were selected for analysis based on their significance and prevalence in news media. The sampling strategy aimed to maintain a balanced representation of articles across these categories while mitigating class imbalance issues through careful selection and sampling techniques.

Feature Engineering with BERT Embeddings:

- BERT (Bidirectional Encoder Representations from Transformers) was employed to generate contextualized word embeddings, enhancing the model's understanding of text semantics and relationships.
- BERT embeddings were integrated into the classification pipeline using tokenization, padding, truncation, and batch processing techniques to efficiently process and utilize the contextualized embeddings for classification tasks.

Model Selection and Training:

In this project, several machine learning models have been used for text classification. These models include:

- Support Vector Machine (SVM): SVM is a powerful and versatile machine learning model that can perform linear or nonlinear classification, regression, and even outlier detection. It's particularly well-suited for high-dimensional data, which is the case with BERT embeddings. The linear kernel is used here for its simplicity and interpretability.

- K-Nearest Neighbors (KNN): KNN is a simple and intuitive algorithm that can be used for both classification and regression tasks. It's a non-parametric method, meaning it doesn't make any assumptions about the underlying data distribution. It's used here to compare its performance with other models.

- Random Forest: Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) of the individual trees. It's used here because it handles high dimensional data well and is robust to overfitting.

- Gradient Boosting Machine (GBM): GBM is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

- Logistic Regression: Despite its name, logistic regression is used for classification problems, not regression problems. It is most commonly used when the dependent variable is binary. It's used here to compare its performance with other models.

The choice of these models is based on their suitability for the task of text classification. Each model has its own strengths and weaknesses, and by using a variety of models, we can compare their performance and choose the one that works best for our specific task. The models have been helpful in this project by providing a way to classify news articles into different categories. The models have been trained on a dataset of news articles, and their performance has been evaluated on a separate test set. The model with the highest accuracy on the test set can be considered the best model for this task.

Evaluation Metrics and Performance Analysis:

- Evaluation metrics such as accuracy, precision, recall, F1 score, and confusion matrices were utilized to assess the models' performance in classifying news articles.
- The experimental results from model training and testing were analyzed, highlighting the comparative performance of each ML model across different categories and providing insights into their strengths, limitations, and implications for newspaper and magazine classification tasks.

Optimization:

- Optimization strategies were implemented to enhance model efficiency, scalability, and accuracy, focusing on fine-tuning hyperparameters and optimizing training processes.
- GridSearchCV are hyperparameter optimization techniques used to systematically search through a predefined grid of hyperparameters to find the best combination that maximizes model performance.
- GridSearchCV automates the process of hyperparameter tuning by exhaustively searching through a specified grid of hyperparameters, evaluating each combination using cross-validation. This helps in finding the optimal set of hyperparameters that yield the best model performance without manual intervention.
  By applying GridSearchCV, we can identify the most suitable hyperparameters for the SVM and KNN classifiers, while gridsearch is used for leading to improved model accuracy and generalization. Fine-tuning these models through hyperparameter optimization enhances their performance on the test data, potentially resulting in better classification results and higher accuracy.

Code Implementation and Tools Used:

- The classification system was implemented using Python programming language, along with libraries such as pandas, scikit-learn, transformers, and TensorFlow for data processing, modeling, and evaluation. Key code segments related to data preprocessing, model training, and evaluation were implemented and optimized.

CHALLENGES

We have initially worked on extracting text from newspaper pdfs and then cluster them accordingly. However, extracting articles (text) from newspaper PDFs had presented several challenges due to the diverse layout, formatting, and content structure. Below are the key challenges faced by us in the process:

- Newspaper PDFs contained articles laid out in different formats, including multi-column layouts, headlines, captions, and sidebars. Extracting text especially when dealing with columns or irregular text arrangements was quite challenging to handle. With reference to [2] blog, we have tried to use the bounding boxes to extract data. However, finding the coordinates of these boxes was quite challenging.

- Newspaper articles include various formatting elements such as bold text, italics, different font sizes and underlining. Extracting text accurately with these formatting attributes required text recognition algorithms and some libraries like pdfplumber. Although they were a great help, it still failed to extract the whole content and missed a few sections of the newspaper and was not entirely accurate.

- Moreover, newspaper PDFs contained a mix of text, images, advertisements, and other graphical elements. Separating article text from non-textual content and identifying relevant sections was quite a challenge.

- In multi-column layouts, text flowed across columns, wrapped around images and graphics. We failed to maintain the correct column alignment and text flow during extraction hampering the readability and comprehension of the text.

- Newspaper PDFs included headers, footers, page numbers, publication dates, special characters, symbols and other metadata. These elements were successfully identified and removed.

- As we failed to mitigate the multi-column layouts and wrapped text around images, we were unsuccessful in extracting the accurate text from pdfs. Although we were able to text properly, sectioning them accurately was another key challenge we failed to resolve. Instead we used kaggle dataset to use bert model to generate word embeddings and build machine learning models to classify the news articles.

CODE SNIPPETS

The below Figure 1 code snippet plays a vital role in the project by preparing the text data for machine learning model training. It starts by standardizing category names and encoding them into numerical format for model compatibility. The defined function ensures text uniformity by converting to lowercase, removing punctuation, and cleaning up spaces. By applying this function to 'headline' and 'short_description' columns and combining them into a single 'text' column, the data is streamlined for further processing. The advanced text processing step involves removing stopwords and lemmatizing words to focus on meaningful content. This

comprehensive text preprocessing pipeline enhances the quality of the input data, making it more suitable for training models like SVM, KNN, Random Forest, Gradient Boosting, and Logistic Regression for accurate text classification in the project.

```python
# Preprocess the data
# Standardizing category names for consistency
df['category'] = df['category'].replace({"THE WORLDPOST": "WORLDPOST", "GREEN": "ENVIRONMENT"})
# Encoding category labels into numerical format for model training
encoder = LabelEncoder()
df['categoryEncoded'] = encoder.fit_transform(df['category'])

# Function to clean and preprocess text
def clean_text(text):
    text = text.lower()  # Convert text to lowercase for uniformity
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation to reduce noise in text
    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces with a single space for clean formatting
    return text

# Applying the cleaning function to headlines and descriptions
df['headline'] = df['headline'].apply(clean_text)
df['short_description'] = df['short_description'].apply(clean_text)
# Combining headline and short description into a single text column
df['text'] = df['headline'] + " " + df['short_description']

# Advanced text processing with stopwords removal and lemmatization
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def advanced_clean_text(text):
    # Remove stopwords to focus on meaningful words and lemmatize to reduce words to their base form
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]
    lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_words]
    return ' '.join(lemmatized_words)

# Applying advanced text processing to the combined text
df['text'] = df['text'].apply(advanced_clean_text)
```

Figure 1: Data Preprocessing Code snippet

```python
# Selecting specific categories to focus our analysis on a subset of data
selected_categories = ['POLITICS', 'SPORTS', 'TECH', 'ENVIRONMENT', 'CRIME', 'BUSINESS', 'ENTERTAINMENT', 'COMED

# Filtering the DataFrame to only include rows from the selected categories
df_selected = df[df['category'].isin(selected_categories)]

# Sampling up to 500 records from each category to balance the dataset and ensure model generalization
df_sampled = df_selected.groupby('category').apply(lambda x: x.sample(n=min(len(x), 500), random_state=42)).rese
```

Figure 2: Data Selection Code snippet

The above Figure 2 code snippet is focusing on specific categories for analysis. By selecting categories like 'POLITICS', 'SPORTS', 'TECH', and others, the data is filtered to include only rows corresponding to these categories. Subsequently, the dataset is balanced by sampling up to 500 records from each selected category to ensure model generalization and prevent bias towards

overrepresented categories. This process helps create a more representative and diverse dataset for training machine learning models, improving their ability to make accurate predictions across various categories in the project.

```
from sklearn.model_selection import train_test_split

# Splitting the dataset into training and testing sets to evaluate the model's performance on unseen data
X_train, X_test, y_train, y_test = train_test_split(
    df_sampled['text'],
    df_sampled['categoryEncoded'],
    test_size=0.1,  # 10% of the data is used for testing
    random_state=2020,  # Ensures reproducibility of results
    stratify=df_sampled['categoryEncoded']  # Keeps the same proportion of categories in both sets
)
```

Figure 3: Dataset Preparation Code snippet

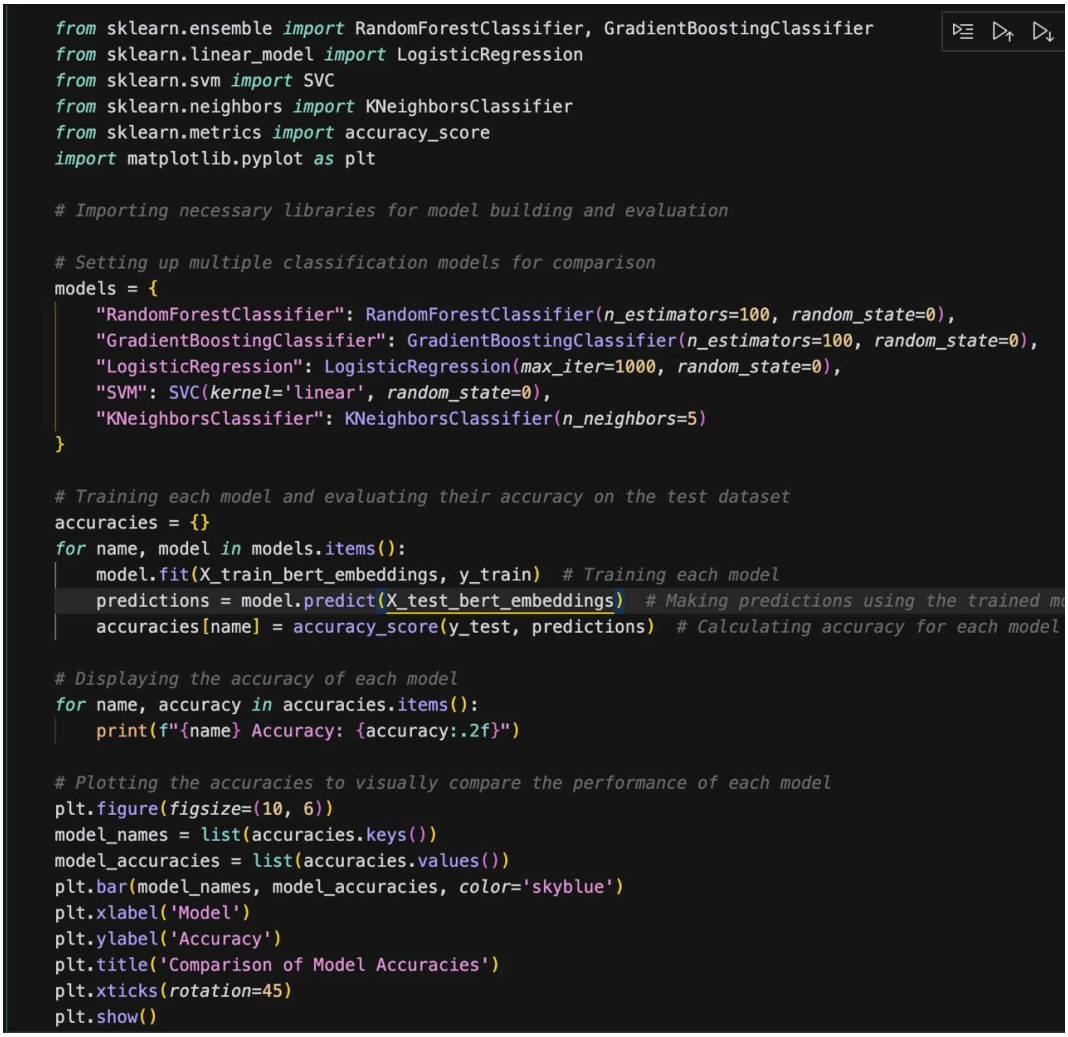
The above Figure 3 code snippet is preparing the dataset for model training and evaluation in the project. It utilizes the function from sklearn.model_selection to split the data into training and testing sets. The 'text' column from the preprocessed dataset is assigned as the feature data (X) while the 'categoryEncoded' column, representing the numerical category labels, is designated as the target data (y). By specifying a of 0.1, 10% of the data is reserved for testing, allowing the model to be evaluated on unseen data. Setting to 2020 ensures reproducibility of results across runs, while using 'categoryEncoded' maintains the same proportion of categories in both the training and testing sets, essential for preserving data balance and representative sampling. This systematic dataset splitting process establishes a solid foundation for training machine learning models and assessing their performance on unseen data, contributing to the reliability and generalization of the models in the project.

```
from transformers import BertTokenizer, TFBertModel
import tensorflow as tf

# Check for GPU availability to speed up computation
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        # Use only the first GPU and allow memory growth to avoid pre-allocating all memory
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        tf.config.experimental.set_memory_growth(gpus[0], True)
    except RuntimeError as e:
        # Handle potential errors during GPU setup
        print(e)

# Initialize BERT tokenizer and model for text processing and embedding generaation
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')
model = TFBertModel.from_pretrained('bert-large-uncased')
```

Figure 4:BERT Tokenizer Code snippet

The above Figure 4 code snippet is significant for setting up the BERT tokenizer and model, leveraging the Hugging Face Transformers library for natural language processing tasks.

Initially, it checks for the availability of GPUs using TensorFlow to accelerate computation. If a GPU is detected, it configures the environment to utilize the first GPU and dynamically allocate memory to prevent pre-allocating all resources, optimizing performance. Subsequently, it initializes the BERT tokenizer and model for text processing and embedding generation. The BertTokenizer is loaded from the 'bert-large-uncased' pre-trained model, enabling tokenization of text data. Similarly, the TFBertModel is instantiated from the same pre-trained model, facilitating the use of BERT for text embedding generation and downstream tasks like text classification. This setup with BERT tokenizer and model integration is essential for leveraging pre-trained language models in the project for advanced text processing and feature extraction, enhancing the model's understanding of textual data and improving classification performance.



```python
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Importing necessary libraries for model building and evaluation

# Setting up multiple classification models for comparison
models = {
    "RandomForestClassifier": RandomForestClassifier(n_estimators=100, random_state=0),
    "GradientBoostingClassifier": GradientBoostingClassifier(n_estimators=100, random_state=0),
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=0),
    "SVM": SVC(kernel='linear', random_state=0),
    "KNeighborsClassifier": KNeighborsClassifier(n_neighbors=5)
}

# Training each model and evaluating their accuracy on the test dataset
accuracies = {}
for name, model in models.items():
    model.fit(X_train_bert_embeddings, y_train)  # Training each model
    predictions = model.predict(X_test_bert_embeddings)  # Making predictions using the trained m
    accuracies[name] = accuracy_score(y_test, predictions)  # Calculating accuracy for each model

# Displaying the accuracy of each model
for name, accuracy in accuracies.items():
    print(f"{name} Accuracy: {accuracy:.2f}")

# Plotting the accuracies to visually compare the performance of each model
plt.figure(figsize=(10, 6))
model_names = list(accuracies.keys())
model_accuracies = list(accuracies.values())
plt.bar(model_names, model_accuracies, color='skyblue')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.xticks(rotation=45)
plt.show()
```

Figure 5: ML Models Code snippet

The above Figure 5 code snippet is pivotal for model building, evaluation, and comparison in the project. It begins by importing necessary libraries for model construction and assessment,

including RandomForestClassifier, GradientBoostingClassifier, LogisticRegression, SVC, KNeighborsClassifier, and accuracy_score from scikit-learn, as well as matplotlib for visualization. Multiple classification models are set up for comparison, each initialized with specific parameters like the number of estimators or neighbors and random state for reproducibility. The models are then trained on the BERT embeddings in the training set and evaluated for accuracy on the test set. The accuracies of each model are calculated and stored in a dictionary for comparison. Finally, a bar plot is generated to visually compare the performance of each model, displaying their accuracies to provide insights into their effectiveness in classifying text data. This comprehensive approach to model training, evaluation, and visualization aids in selecting the best-performing classifier for the text classification task in the project.

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Collecting predictions from different models to compare their performance
model_predictions = {
    "RandomForest": rf_predictions,
    "GradientBoosting": gbm_predictions,
    "LogisticRegression": lr_predictions,
    "SVM": svm_predictions,
    "KNN": knn_predictions
}

# Setting up a grid of plots for visual comparison
fig, axes = plt.subplots(3, 2, figsize=(12, 18))
fig.delaxes(axes[2,1])  # Remove the unused subplot space

# Visualizing the performance of each model using confusion matrices
for ax, (name, predictions) in zip(axes.flatten(), model_predictions.items()):
    conf_mat = confusion_matrix(y_test, predictions)
    # Highlighting the top 5 performing labels to focus on more frequent or important classifications
    top_labels = conf_mat.diagonal().argsort()[-5:][::-1]
    filtered_conf_mat = conf_mat[top_labels, :][:, top_labels]
    sns.heatmap(filtered_conf_mat, annot=True, fmt='d', ax=ax, cmap="Blues")
    ax.set_title(f'{name} Confusion Matrix')
    ax.set_xlabel('Predicted Labels')
    ax.set_ylabel('True Labels')
    # Enhancing readability of the plot labels
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
    ax.set_yticklabels(ax.get_yticklabels(), rotation=0)

plt.tight_layout()
plt.show()
```

Figure 6: Visualization Code snippet

The above Figure 6 code snippet is essential for visualizing and comparing the performance of different classification models through confusion matrices. Initially, predictions from various models, including RandomForest, GradientBoosting, LogisticRegression, SVM, and KNN, are

collected for comparison. A grid of plots is set up using matplotlib to display the confusion matrices for each model, allowing for a side-by-side visual assessment of their classification results. Each subplot represents a model, with the confusion matrix heatmap generated using seaborn to highlight the performance across different classes. The top 5 performing labels are emphasized to focus on the most relevant classifications, enhancing the interpretability of the results. By presenting the confusion matrices in a structured and informative manner, this visualization aids in understanding the strengths and weaknesses of each model in classifying text data, facilitating informed decision-making on model selection and optimization in the project.

EXPERIMENTAL RESULTS

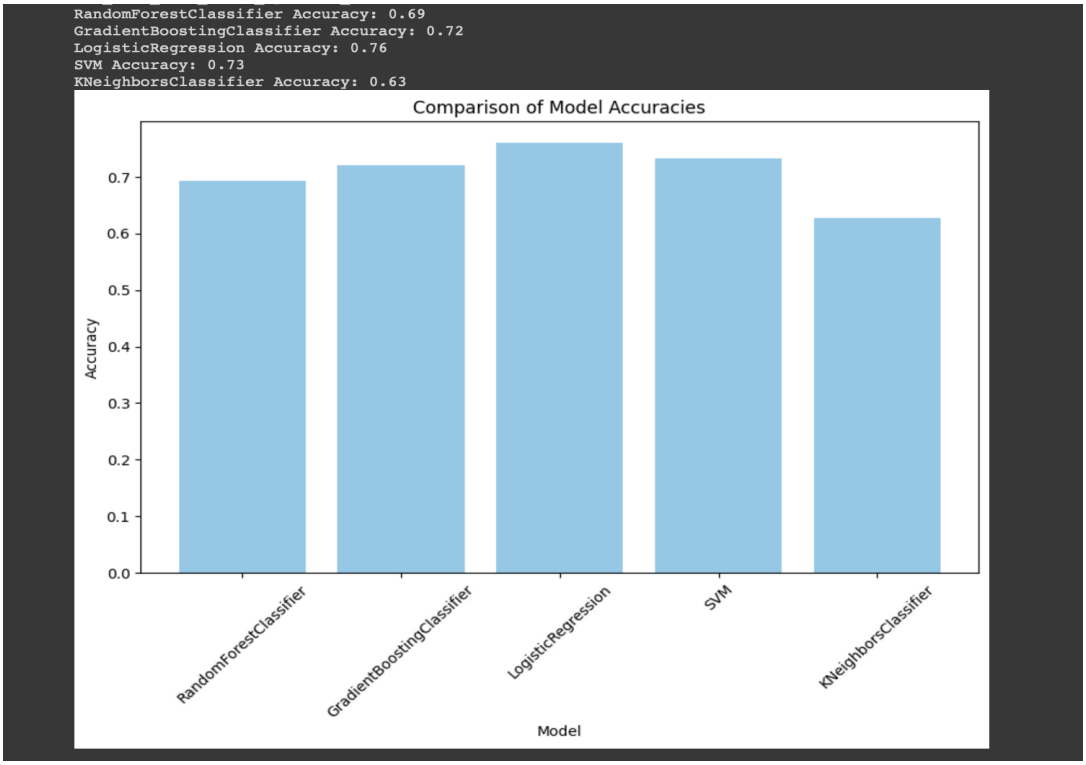


Figure 7: Model Accuracy Comparison

Figure 7 displays a bar chart titled "Comparison of Model Accuracies," which compares the accuracy of five different machine learning models: RandomForestClassifier, GradientBoostingClassifier, LogisticRegression, SVM (Support Vector Machine), and KNeighborsClassifier. Each bar represents the accuracy of one model, measured on a scale from 0 to 1 (0% to 100%).

From the chart, it is evident that all models perform relatively similarly, with accuracies hovering around the 0.6 to 0.7 range. The RandomForestClassifier and GradientBoostingClassifier show

slightly higher accuracies compared to the other models, indicating they might be more effective for this particular dataset. The LogisticRegression and SVM models also display competitive performance, while the KNeighborsClassifier has a slightly lower accuracy compared to the others. This visual comparison helps in quickly assessing which models might be more suitable for further tuning or deployment based on their accuracy metrics.

The below Figure 8 image displays a series of confusion matrices for five different machine learning models: RandomForest, GradientBoosting, LogisticRegression, SVM (Support Vector Machine), and KNN (K-Nearest Neighbors). Each matrix is a visual representation of the performance of a model in terms of its predictions across different top 5 categories (labeled from 0 to 4).

Analysis of Each Model's Confusion Matrix:

1. RandomForest Confusion Matrix:
   - Shows relatively balanced performance across most categories.
   - Notable for relatively fewer misclassifications between categories 0 and 4.

2. GradientBoosting Confusion Matrix:
   - Similar to RandomForest, shows good performance but with slightly better accuracy in predicting category 1.
   - Shows some confusion between categories 0 and 4, similar to RandomForest.

3. LogisticRegression Confusion Matrix:
   - Exhibits strong performance in categories 1 and 3.
   - Shows some misclassifications between categories 0 and 4, indicating possible similarities or overlaps in features between these categories.

4. SVM Confusion Matrix:
   - Performs well across all categories, particularly in categories 1 and 3.
   - Similar to LogisticRegression, it shows confusion between categories 0 and 4.

5. KNN Confusion Matrix:
   - Shows a different pattern of misclassification, particularly struggling with category 0 where it frequently misclassified instances as belonging to categories 2 and 4.
   - Performs well in category 1 but shows significant confusion in categories 3 and 4.

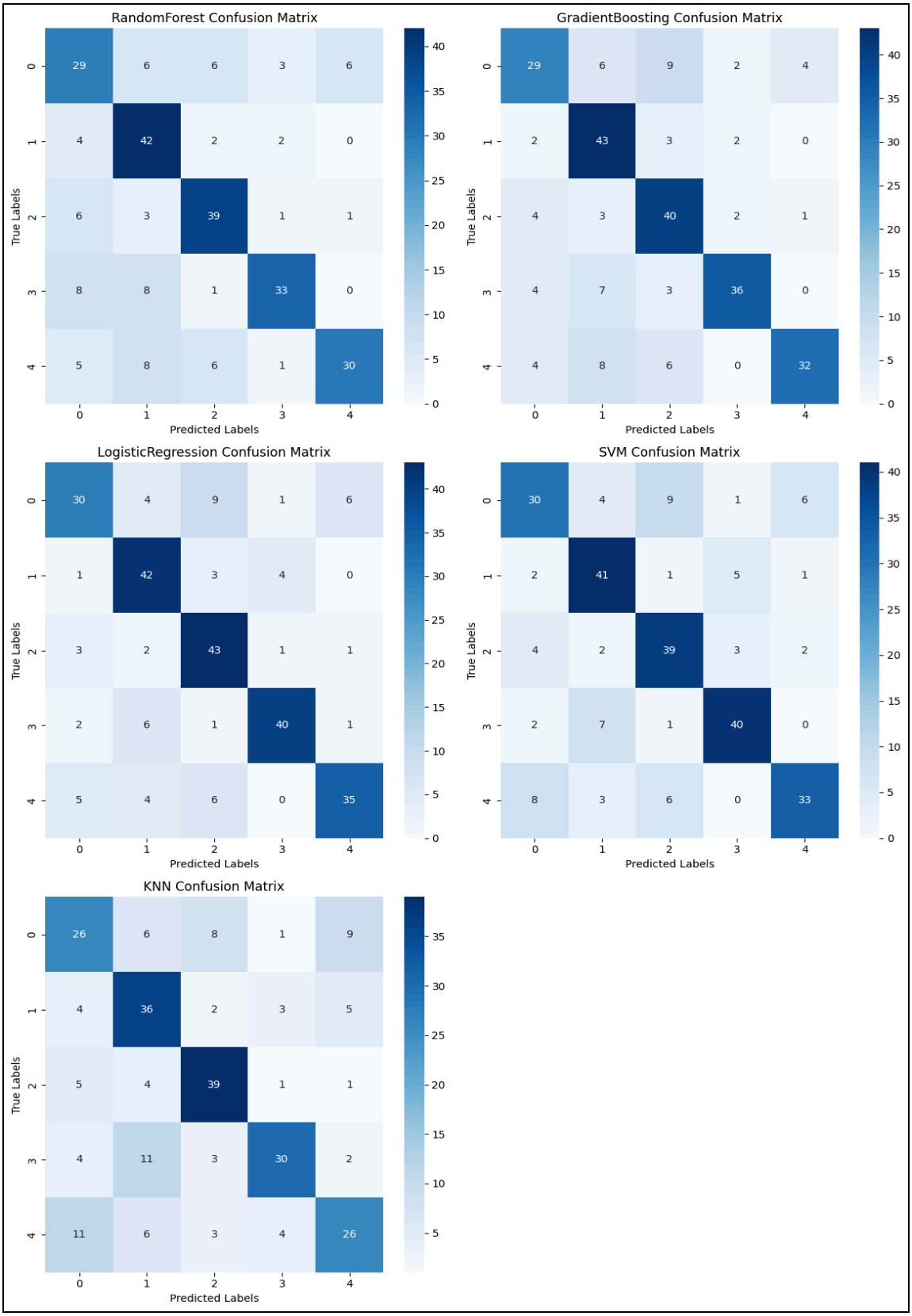


Figure 8: Confusion Matrix

Key Observations:
- All models show some degree of confusion between categories 0 and 4, suggesting that these categories might have overlapping or similar features that are hard to distinguish.
- Categories 1 and 3 are generally predicted with higher accuracy across models, indicating distinct features that are easier to classify.
- The intensity of the blue color in each cell of the matrices corresponds to the number of predictions, with darker shades indicating higher numbers. This visual cue helps in quickly identifying which categories are most accurately predicted and which ones are prone to errors.

These confusion matrices are crucial for understanding not just the overall accuracy of each model, but also how well each model performs on individual categories, which can guide further improvements in model training and feature engineering.

FUTURE SCOPE

In the future, this project could be expanded by incorporating advanced techniques such as hyperparameter tuning, ensemble methods, and deep learning architectures to further enhance the model's performance and generalization capabilities. By exploring ensemble learning and deep learning models like LSTM, CNN, or Transformers, the project can capture intricate patterns in text data for improved classification accuracy. Developing a key strategy to extract accurate data from newspapers with the advanced technologies would bring great benefits to the media and research industry. Furthermore, developing a deployment strategy, such as creating a user-friendly interface or API for real-time text classification, would enable the seamless integration of the model into practical applications, ensuring its usability and impact in real-world scenarios. These future enhancements aim to elevate the project's capabilities in text classification and advance its effectiveness in handling diverse text analysis tasks.

REFERENCES

1. Devlin, J., et al. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

2. https://www.analyticsvidhya.com/blog/2021/06/data-extraction-from-unstructured-pdfs/

3. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

4. TensorFlow Documentation. GPU support. (https://www.tensorflow.org/guide/gpu)