CSC 590/599 ─ Graduate Project/Thesis (S25)
Project Progress Report

****

Project Title: Medical Text Classification using LLMs
Student Name: Medha Maisa

****

1. Description of Completed Work (Itemized)

Below is an itemized list detailing the work completed so far for the project titled *"Medical Text Classification using LLMs"*. All significant milestones and results are outlined below. Actual code implementations and outputs are included in Section 3 (Appendix) and referenced accordingly.

1.1 Dataset Acquisition and Initial Loading

• Downloaded the TCGA Pathology Reports dataset from Kaggle as the primary dataset for analysis.
• Loaded the dataset into a structured format using Python (Pandas), focusing on the following key fields:

- text: the main body of pathology report data.
- patient_filename: used as a unique identifier (not for classification purposes).

The dataset loading and parsing routines are documented in Appendix A.1.

1.2 Initial Data Exploration and Cleaning

Conducted comprehensive exploratory data analysis (EDA) on the text column:

- Analyzed text length distributions, null/missing values, and duplicates.
- Assessed the overall quality and volume of available data.

Performed data cleaning steps:

- Removed duplicates and filtered out records with extremely short texts (fewer than 50 characters).
- Applied text normalization, including lowercasing and removing special characters, ensuring uniformity across the dataset.

These steps are detailed in Appendix A.2.

1.3 Binary Label Generation from Text Content

• Generated binary classification labels for 'Cancer' and 'Non-Cancer' based on the presence of the term 'cancer' in the pathology reports:

- Assigned label 1 if the word "cancer" appeared in the text.
- Assigned label 0 otherwise.

Conducted a balance evaluation of the labels to confirm the distribution of classes.

1.4 Baseline Model: TF-IDF + Logistic Regression

• Built a baseline machine learning pipeline using TF-IDF vectorization for feature extraction and Logistic Regression for binary classification.
• To handle class imbalance, the class_weight="balanced" parameter was used in Logistic Regression, ensuring that the model gave more importance to the minority class during training.
• Performed 80/20 train-test split with stratification to maintain class balance.
• Evaluating performance using metrics such as accuracy, precision, recall, and F1-score:

The details of model evaluation and results are included in Appendix A.3.

1.5 Multi-Class Labeling Exploration (Preliminary Work)

• Explored the possibility of moving beyond binary classification to multi-class classification using cancer subtype references inferred from the text.
• Encountered class imbalance across potential subtypes (e.g., BRCA, COAD), which made it challenging to build meaningful classifiers for certain subtypes.
• Ongoing work is aimed at identifying the top-N most frequent and meaningful subcategories for further exploration.

1.6 Visualization and Reporting

• Created visualizations to support EDA and model interpretation:

- Class distribution (binary classification).
- Text length histograms to assess the variability of text size.
- Word clouds for cancer vs. non-cancer terms based on TF-IDF features.

All preprocessing steps and model results are documented in Jupyter Notebooks with markdown for clarity and reproducibility. Visualizations are included in Appendix A.4.

2. Remaining Work and Completion Plan

The following tasks are required to complete the project. These tasks are structured around key milestones to ensure timely and focused progress:

2.1 Dataset Refinement and Labeling

• Conduct further exploratory data analysis (EDA) to detect potential biases, inconsistencies, and outliers in the dataset.
• Apply oversampling or undersampling techniques (e.g., SMOTE or random undersampling) to balance the classes in the dataset. This will help to prevent the model from being biased toward the majority class and improve classification performance for the minority class.
• Validate and refine the binary classification labels to ensure consistency and accuracy based on the content of the pathology reports.

2.2 Rebuilding the Baseline Model

• Explore additional traditional machine learning algorithms such as Naive Bayes, SVM, and Random Forest for comparison.
• Train and evaluate each model using standard metrics like accuracy, F1-score, precision, and recall.
• Select the best-performing model as the final baseline based on comparative analysis.
• Document results and maintain reproducibility for future comparisons.

2.3 Building the LLM-Based Classifier

• Investigate and evaluate various approaches for applying Large Language Models (LLMs) to medical text classification:

- Explore architectures such as BERT, BioBERT, and GPT variants.
- Prepare the dataset for compatibility with LLMs, ensuring proper formatting for prompt-based classification.
- Finalize the LLM approach after evaluating different model architectures and implementation methods.

2.4 Few-Shot Learning Experimentation

• Investigate the use of few-shot learning techniques with LLMs.
• Experiment with different prompt designs and input formats to optimize performance.
• Run experiments and compare results with both baseline models and fine-tuned LLM models to assess the effectiveness of few-shot learning.

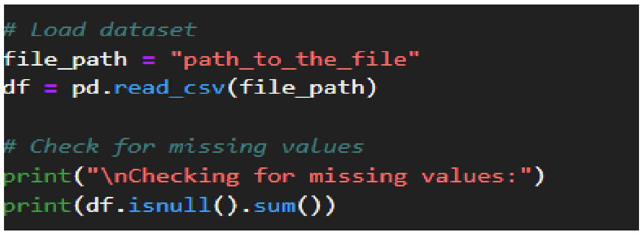
2.5 Analysis and Visualization

• Create comparative performance charts and tables for all models evaluated (both baseline and LLM-based).
• Use visual tools such as confusion matrices, ROC curves, and bar graphs to illustrate the models' effectiveness and behavior.
• Identify insights from misclassifications or edge cases, if any, to improve model performance.

3. Appendix – Source Code

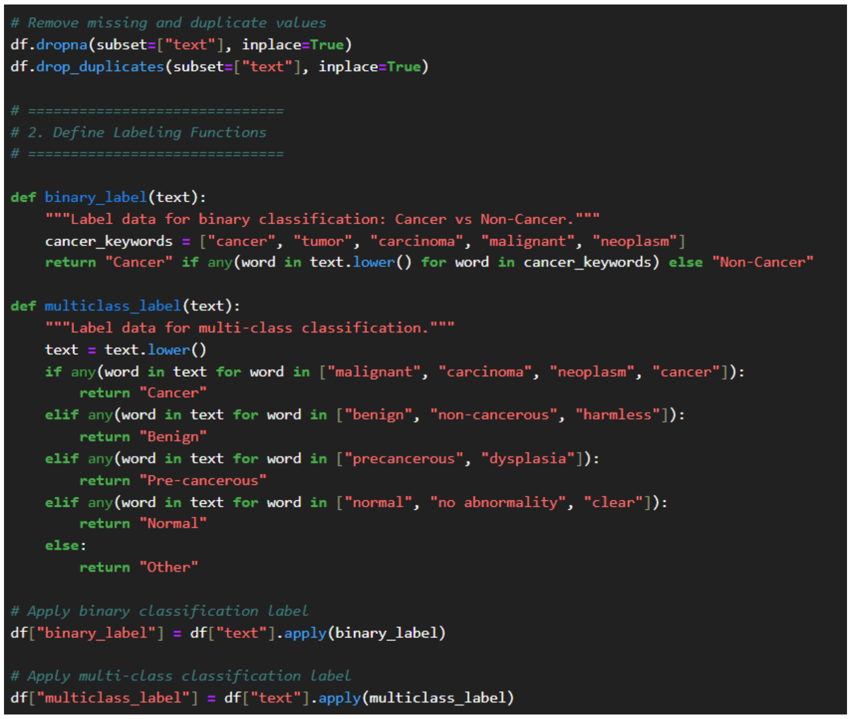
The full source code for data preprocessing, model training, evaluation, and visualizations is included in this section. The code is well-organized and properly documented to ensure reproducibility. Specific code snippets and references are provided below:

Appendix A.1: Dataset loading and parsing code.

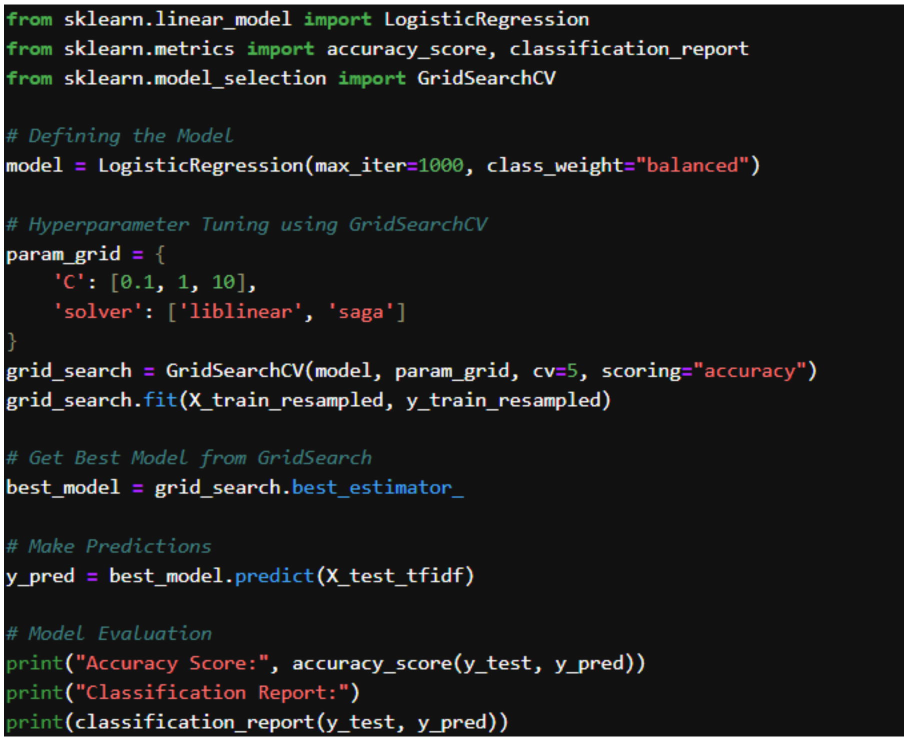


```
# Load dataset
file_path = "path_to_the_file"
df = pd.read_csv(file_path)

# Check for missing values
print("\nChecking for missing values:")
print(df.isnull().sum())
```

Appendix A.2: Data cleaning and normalization code.


```python
# Remove missing and duplicate values
df.dropna(subset=["text"], inplace=True)
df.drop_duplicates(subset=["text"], inplace=True)

# ==============================
# 2. Define Labeling Functions
# ==============================

def binary_label(text):
    """Label data for binary classification: Cancer vs Non-Cancer."""
    cancer_keywords = ["cancer", "tumor", "carcinoma", "malignant", "neoplasm"]
    return "Cancer" if any(word in text.lower() for word in cancer_keywords) else "Non-Cancer"

def multiclass_label(text):
    """Label data for multi-class classification."""
    text = text.lower()
    if any(word in text for word in ["malignant", "carcinoma", "neoplasm", "cancer"]):
        return "Cancer"
    elif any(word in text for word in ["benign", "non-cancerous", "harmless"]):
        return "Benign"
    elif any(word in text for word in ["precancerous", "dysplasia"]):
        return "Pre-cancerous"
    elif any(word in text for word in ["normal", "no abnormality", "clear"]):
        return "Normal"
    else:
        return "Other"

# Apply binary classification label
df["binary_label"] = df["text"].apply(binary_label)

# Apply multi-class classification label
df["multiclass_label"] = df["text"].apply(multiclass_label)
```

Appendix A.3: Code for training the baseline model (TF-IDF + Logistic Regression) and evaluating its performance.


```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV

# Defining the Model
model = LogisticRegression(max_iter=1000, class_weight="balanced")

# Hyperparameter Tuning using GridSearchCV
param_grid = {
    'C': [0.1, 1, 10],
    'solver': ['liblinear', 'saga']
}
grid_search = GridSearchCV(model, param_grid, cv=5, scoring="accuracy")
grid_search.fit(X_train_resampled, y_train_resampled)

# Get Best Model from GridSearch
best_model = grid_search.best_estimator_

# Make Predictions
y_pred = best_model.predict(X_test_tfidf)

# Model Evaluation
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Appendix A.4: Visualizations and figures used for EDA and model interpretation.

```
Binary Label Distribution
binary_label
Cancer        9327
Non-Cancer     178
Name: count, dtype: int64
```

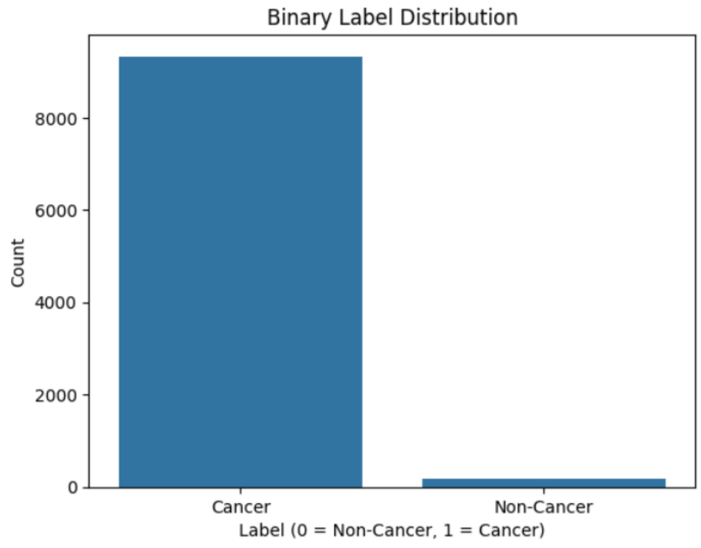
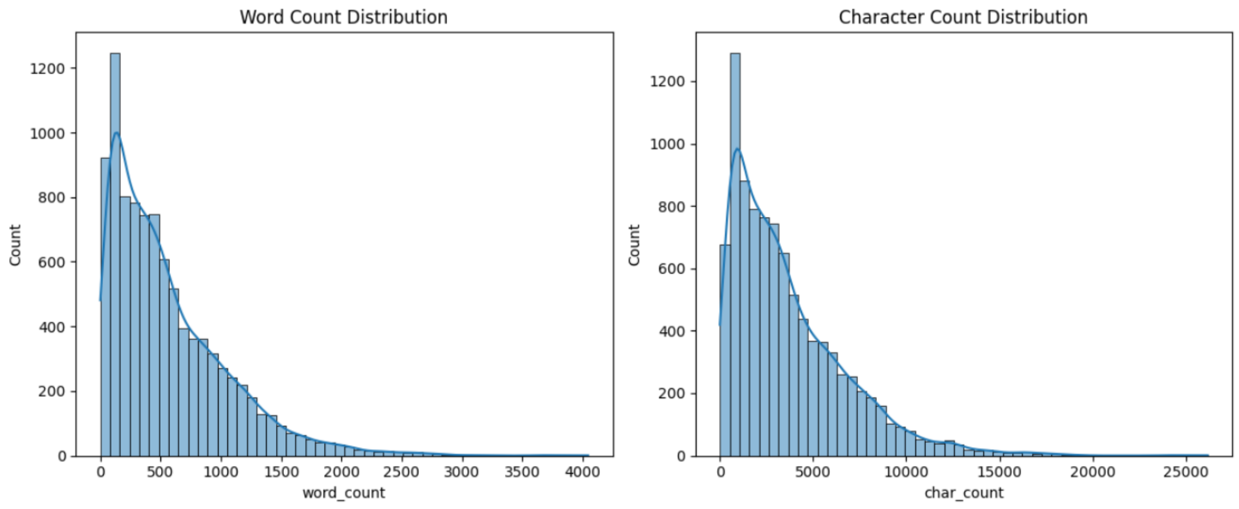


Figure 1Binary class Distribution



```
Average word count: 560.9814834297738
Average char count: 3671.8004208311413
```

Figure 2 Text length histograms