Software Engineering (AI - Stream)

NLP Final Project

Fake News Detection Report

Name	ID.No
1. Leul Wujira Berhe	UGR/8834/13
2. Mariam Yohannes Gustavo	UGR/1997/13
3. Melkishi Tesfaye Angassa	UGR/0078/13
4. Metsakal Zeleke Enyew	UGR/1027/13
5. Milka Fasika Gebre	UGR/7126/13

Submitted To: Dr. Fantahun Bogale
Date: Nov 21, 2024

Data Source

The data used in this preprocessing step originates from a dataset provided by Membere Hailu in the project "Amharic fake news detection on social media using pretrained language model", https://github.com/MenbereHailu/Amharic_Fake_News_Detection_On_Social_Media-_Using_Pretrained-_Language_Model/blob/main/Merged%20_Fakenews_Data.xlsx. No information on how the data was collected is shown in the data source. The dataset is a xlsx file with **8630 rows x 2 columns.** Each row is represented as the following**: Post|Label**

	Actual_Post	Label
0	መንግስት ከልምድ ተመሮ ምንም ጥቅምና አትኩሮት የሌለውን መግለጫ በመግለጫ...	Fake
1	ወይ ጉድ! ስለ ዜግነት በቅጡ ሳይረዱ ዜግነት ሰጪና ከልካይ የፖለቲካ መሪ...	Fake
2	አብሮነት በመከባበር ላይ የተመሰረተ መሆን አለበት። ኦሮሚያ ውስጥ የምትኖ...	Fake
3	እንኳን አሁን ወያኔም እያለ የህዝቦች ሁሉ ጠላት ነፍጠኛዉ ነው ብዬ ነበር...	Fake
4	እንደዚህ እንደዚያ የሚያረጉን "ኃይሎች" እያሉ መንግስት ነን ባዮች ችግሮ...	Fake
...	...	...
8625	በግብጽ ዋና ከተማ ካይሮ በሚገኝ አንድ ቅንጡ ሆቴል አንዲት ወጣት ሴትን ...	Real
8626	ዌስት ማቲውሰን ይባላሉ። እውቅ የአካባቢ ተንከባካቢ ነበሩ። በተለይም ከደ...	Real
8627	ሰርቫይቫል ኢንተርናሽናል የተባለ ተቋም በ1850ዎቹ ብሪታኒያ ሕንድን ስት...	Real
8628	ፓርቲዎቹ የመጀመሪያውን የምርጫ ክርክር በመገናኛ ብዙሃን በተላለፈ የቀጥታ...	Real
8629	በፍርድ ሂደቱ የመጀመሪያዎቹ ሦስት ቀናት ጉዳት የደረሰባቸው ሰዎች ምስክር...	Real

Preprocessing Steps

Importing Libraries:

The notebook imports necessary libraries such as os, sys, re, pandas, and a custom preprocessing module.

Setting Up Paths:

Paths for raw and processed data directories are defined.

Loading the Dataset:

The raw dataset is loaded from an Excel file into a pandas DataFrame.

Initial Data Profiling:

The raw data is compiled to identify URLs, English words, Amharic Geez numbers, special characters, emojis, extra spaces, HTML tags, elongated words, and leading/trailing spaces.

```
{'URLs': 1294,
 'English words/digits': 135414,
 'Amharic Geez numbers': 24,
 'Special characters/punctuation': 205193,
 'Emojis': 1686,
 'Extra spaces': 17345,
 'HTML tags': 110,
 'Elongated words': 11193,
 'Leading/trailing spaces': 150}
```

Custom Preprocessing Module
A custom preprocessing module is located in src/Preprocess that implements the cleaning and normalization of the data.

Data Cleaning

Removal of URLs: All URLs are removed from the text.
Removal of English Words/Digits: English words and digits are removed.
Removal of Amharic Geez Numbers: Amharic Geez numbers are removed.
Removal of Special Characters/Punctuation: Special characters and punctuation marks are removed.
Removal of Emojis: Emojis are removed from the text.
Removal of Extra Spaces: Extra spaces are removed.
Removal of HTML Tags: HTML tags are removed from the text.
Handling Elongated Words: Elongated words are normalized.
Removal of Leading/Trailing Spaces: Leading and trailing spaces are trimmed.

Data Normalization

A list of normalization rules used for preprocessing Amharic text. These rules standardize various forms of Amharic characters to a single, consistent form. Normalization has been performed in two ways: Normalization of Characters- different forms of the same character have been replaced with a standardized form. Normalization of composite characters- involves converting combinations of a base character followed by for example ፇ or ኣ into a single, standardized character.
This ensures that different representations of the same character are unified, which is crucial for accurate text processing and analysis.

Conversion of Labels

The labels are converted from 'Real'/'Fake' to binary values (1 for Real, 0 for Fake).

Saving the Processed Data:
The cleaned data is saved to a CSV file.

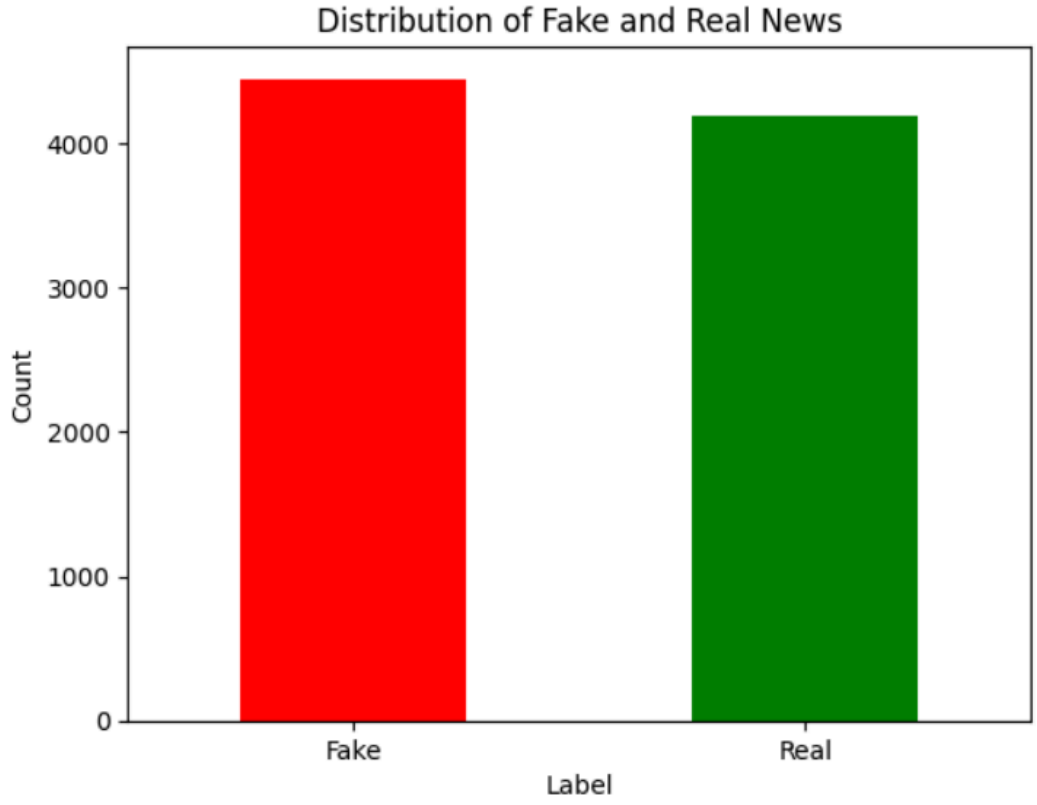
Post-Processing Data Profiling

The cleaned data is profiled to ensure all unwanted elements have been removed.

```
{'URLs': 0,
 'English words/digits': 0,
 'Amharic Geez numbers': 0,
 'Special characters/punctuation': 0,
 'Emojis': 0,
 'Extra spaces': 0,
 'HTML tags': 0,
 'Elongated words': 0,
 'Leading/trailing spaces': 0}
```

Data Visualization

The distribution of fake and real news is visualized using a bar plot. Showing that the fake-to-real ratio is balanced.



The preprocessing steps ensure the dataset is cleaned and prepared for further analysis and model training.

Model Training

1. Splitting the dataset

The dataset was divided using the train_test_split method from sklearn into training and testing subsets with a splitting ratio of 80% for training and 20% for testing, ensuring a robust evaluation setup. The random state was fixed to 42 to ensure reproducibility.

2. Feature Vectorization

Textual data was transformed into numerical features using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization generating XV_train - Transformed training data and XV_test - Transformed testing data.

3. Classical Model Training and Evaluation

1) **Model**: The models were trained on the TF-IDF transformed training data (XV_train) and labels (y_train) and predictions were made on the test data (XV_test).
2) **Evaluation**: Accuracy and a classification report (precision, recall, F1-score) were calculated.
    a. **Logistic Regression:** is a statistical **supervised machine learning algorithm** used for **classification tasks** where the goal is to predict the probability that an instance belongs to a given class or not. We used a LogisticRegression library from sklearn.linear_model.

Results:

```
lr_prediction = LR.predict(XV_test)
print("Accuracy: ", LR.score(XV_test, y_test))
print("Classification Report: " , classification_report(y_test, lr_prediction))
```

```
Accuracy:  0.9351476548928779
Classification Report:                precision    recall  f1-score   support

           0       0.93      0.94      0.93       855
           1       0.94      0.93      0.94       872

    accuracy                           0.94      1727
   macro avg       0.94      0.94      0.94      1727
weighted avg       0.94      0.94      0.94      1727
```

    b. **Decision Tree:** is a tree-like structured **supervised machine learning algorithm** used for both **regression** and **classification tasks** where it is commonly used in machine learning to model and predict outcomes based on input data. We used a DecisionTreeClassifier library from sklearn.treee.

Results:

```
]: ds_prediction = DT.predict(XV_test)
print("Accuracy: ", DT.score(XV_test, y_test))
print("Classification Report: " , classification_report(y_test, ds_prediction))
```

```
Accuracy:  0.8760856977417487
Classification Report:               precision    recall  f1-score   support

           0       0.88      0.87      0.87       855
           1       0.88      0.88      0.88       872

    accuracy                           0.88      1727
   macro avg       0.88      0.88      0.88      1727
weighted avg       0.88      0.88      0.88      1727
```

c. **Random Forest:** is **supervised machine learning nonlinear classification and regression** algorithm for classifying a group of datasets in categories or classes.We used a RandomForestClassifier library from sklearn.ensemble.

Results:

```
: rf_prediction = RF.predict(XV_test)
print("Accuracy: ", RF.score(XV_test, y_test))
print("Classification Report: " , classification_report(y_test, rf_prediction))
```

```
Accuracy:  0.9148812970469021
Classification Report:               precision    recall  f1-score   support

           0       0.91      0.92      0.91       855
           1       0.92      0.91      0.92       872

    accuracy                           0.91      1727
   macro avg       0.91      0.91      0.91      1727
weighted avg       0.91      0.91      0.91      1727
```

d. **Gradient Boosting:** is a popular boosting algorithm in machine learning used for classification and regression tasks. We used a GradientBoostingClassifier library from sklearn.ensemble.

Results:

```
: gb_prediction = GB.predict(XV_test)
print("Accuracy: ", GB.score(XV_test, y_test))
print("Classification Report: " , classification_report(y_test, gb_prediction))
```

```
Accuracy:  0.8922987840185292
Classification Report:               precision    recall  f1-score   support

           0       0.86      0.93      0.90       855
           1       0.93      0.85      0.89       872

    accuracy                           0.89      1727
   macro avg       0.89      0.89      0.89      1727
weighted avg       0.90      0.89      0.89      1727
```

```
def output_lable(n): if n == 0: return "Fake News" elif n == 1: return "Not A Fake News"
```

4. Using fine-tuned pretrained XLM-RoBERTa Model

We used a different splitting strategy here we have splitted the dataset into training, test, and validation set.

```
# Assume df is your DataFrame with "Post" (text) and "Label" (target)
X = df["Post"]
y = df["Label"]

# Step 1: Split dataset into train, validation, and test sets
train_texts, temp_texts, train_labels, temp_labels = train_test_split(
    df["Post"], df["Label"], test_size=0.4, random_state=42, stratify=df["Label"]
)
val_texts, test_texts, val_labels, test_labels = train_test_split(
    temp_texts, temp_labels, test_size=0.5, random_state=42, stratify=temp_labels
)

# Step 2: Convert splits to Hugging Face Dataset format
train_data = Dataset.from_dict({"text": train_texts, "label": train_labels})
val_data = Dataset.from_dict({"text": val_texts, "label": val_labels})
test_data = Dataset.from_dict({"text": test_texts, "label": test_labels})

dataset = DatasetDict({
    "train": train_data,
    "validation": val_data,
    "test": test_data
})
```

We have tokenized our dataset. After tokenization we started training our model.

Training the model

We used the following training parameters:

- **Learning Rate:** A small learning rate of 2e-5 is chosen for fine-tuning.
- **Batch Sizes:** Both training and evaluation batch sizes are set to 16.
- **Number of Epochs:** The model is trained for 3 epochs.
- **Weight Decay:** Regularization is achieved using a weight decay of 0.01.
- **Output Directory:** Results are saved in `./results`.
- **Evaluation Strategy:** Evaluation is performed at the end of each epoch.
- **Save Strategy:** Model checkpoints are saved at the end of each epoch.
- **Logging:** Logs are saved in `./logs` with a logging frequency of every 10 steps.
- **Best Model:** The model with the highest F1 score during evaluation is loaded.
- **Metrics Reporting:** The F1 score is used to evaluate model performance.
- **Datasets:** Tokenized training and validation datasets.
- **Model:** The pre-trained `xlm-roberta-base` model from `AutoModelForSequenceClassification` library.

[972/972 32:13, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.097900	0.236477	0.949015	0.929260	0.975253	0.951701
2	0.145900	0.322366	0.928737	0.884538	0.991001	0.934748
3	0.076800	0.231286	0.956547	0.939525	0.978628	0.958678

```
]: TrainOutput(global_step=972, training_loss=0.09535779348678058, metrics={'train_runtime': 1935.4568, 'train
   _samples_per_second': 8.026, 'train_steps_per_second': 0.502, 'total_flos': 4087167133962240.0, 'train_los
   s': 0.09535779348678058, 'epoch': 3.0})
```

Testing the model

We have used accuracy, precision, recall, and F-score to test our model using the test dataset.

[108/108 00:50]

```
Test Results: {'eval_loss': 0.2797089219093323, 'eval_accuracy': 0.9496236247828604, 'eval_precision': 0.92
13011542497377, 'eval_recall': 0.9865168539325843, 'eval_f1': 0.9527943570265871, 'eval_runtime': 51.1718,
'eval_samples_per_second': 33.749, 'eval_steps_per_second': 2.111, 'epoch': 3.0}
```

```
|: ('./amharic_fake_news_model/tokenizer_config.json',
    './amharic_fake_news_model/special_tokens_map.json',
    './amharic_fake_news_model/sentencepiece.bpe.model',
    './amharic_fake_news_model/added_tokens.json',
    './amharic_fake_news_model/tokenizer.json')
```

Model Testing

After training, the model is saved and reused for predictions:

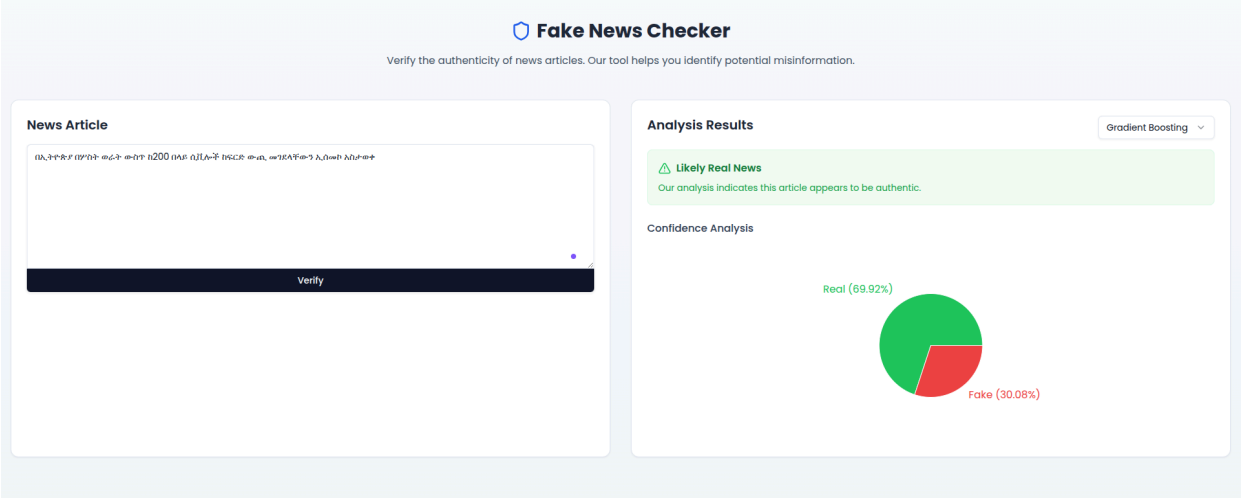
1. **Loading the Model and Tokenizer:**
   - The fine-tuned model and tokenizer are loaded from the directory `./amharic_fake_news_model`.
2. **Prediction Function (`predict_news`):**
   - The input text is tokenized with padding, truncation, and conversion to tensors suitable for the model.
   - The model is put in evaluation mode (`model.eval()`), ensuring no gradients are computed during inference.
   - The logits from the model's output are used to predict the class (`torch.argmax`).
   - The predicted class (0 for "Fake" and 1 for "Real") is mapped to a label for interpretation.
3. **Example Prediction:** A sample news text, `"ኢትና ላነት መደጸች"`, is passed to the `predict_news` function, and the output is printed.

```python
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

model_path = "./amharic_fake_news_model"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSequenceClassification.from_pretrained(model_path)

def predict_news(news_text):
    inputs = tokenizer(
        news_text,
        padding="max_length",
        truncation=True,
        max_length=512,
        return_tensors="pt"
    )

    model.eval()

    # Perform the prediction
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
        predicted_class = torch.argmax(logits, dim=1).item()

    # Map prediction to label
    label_map = {0: "Fake", 1: "Real"}
    return label_map[predicted_class]


sample_news = "ኢትዮ ላነት መደጸቸ"
prediction = predict_news(sample_news)
print(f"The news is predicted to be: {prediction}")
```

```
The news is predicted to be: Fake
```

Model Deployment

We have built an app endpoint to utilize our fake news detection. The application can be found using this link https://fake-news-detector-snowy.vercel.app/



The above is a sample example using Gradient Booting on a real new from
https://amharic.voanews.com/a/ehrc-report-civilians-killing/7949338.html