MODEL DOCUMENTATION

Afaan Oromoo News Classifier

BY: ABDULMUNIM JUNDURAHMAN JEMAL

Table of Contents

1. Project Overview

The Afaan Oromoo News Classification System is a solo, personal project aimed at developing a machine learning model to categorize news articles written in Afaan Oromoo, the language spoken by the Oromo people in Ethiopia. The primary goal of this project is to create a tool that efficiently organizes and classifies Afaan Oromoo news content, facilitating better content management and retrieval.

1.1 Motivation

The motivation behind this project is to address the lack of automated tools for organizing and categorizing Afaan Oromoo news articles. With the increasing volume of online news content in Afaan Oromoo, manually categorizing and managing these articles becomes challenging. The project seeks to leverage machine learning techniques to automate this process, making it more efficient and scalable.

1.2 Objectives

- *Text Classification:* Develop a text classification model capable of categorizing Afaan Oromoo news articles into predefined topics or categories.
- *Data Collection*: Collect a diverse dataset of Afaan Oromoo news articles from various online sources to ensure a representative sample for training.
- *Data Preprocessing*: Implement a robust data preprocessing pipeline that includes tokenization, normalization, removal of stopwords, and specific stemming for Afaan Oromoo.
- *Model Training:* Train a machine learning model using a suitable architecture for text classification, considering the nuances of the Afaan Oromoo language.
- *Model Evaluation*: Evaluate the model's performance using appropriate metrics, ensuring its effectiveness in accurately categorizing news articles.
- *Deployment*: Create an API for users to interact with the trained model and obtain predictions for new Afaan Oromoo text.
- *Documentation*: Document the entire project, including data sources, preprocessing steps, model architecture, and deployment instructions.

1.3 Tools and Technologies

- *Programming Language*: Python
- *Machine Learning Framework*: TensorFlow and Keras
- *Web Framework:* Flask (for API deployment)
- *Data Scraping:* Beautiful Soup and Scrapy
- *Version Control:* Git
- *Documentation:* Sphinx

1.4 Project Timeline

- *Data Collection*:
- *Data Preprocessing*:
- *Model Development and Training*:
- *Model Evaluation and Fine-Tuning*:
- *Deployment*:
- *Documentation and Testing*:

1.5 Expected Challenges

- *Limited Labeled Data*: Finding a sufficient amount of labeled Afaan Oromoo news articles for training may be challenging.
- *Language Specifics*: Handling the unique linguistic characteristics of Afaan Oromoo during preprocessing and modeling requires careful consideration.
- *Deployment Complexity*: Deploying a machine learning model as an API introduces challenges related to server setup and maintenance.

1.6 Project Deliverables

- *Trained Model:* A machine learning model capable of accurately classifying Afaan Oromoo news articles.
- *Web Interface or API:* A deployed interface or API for users to interact with the model.
- *Documentation:* Comprehensive documentation covering data sources, preprocessing steps, model architecture, and deployment instructions.

1.7 Success Criteria

The project will be considered successful if the deployed model achieves a satisfactory level of accuracy in categorizing Afaan Oromoo news articles and provides a application development interface (API) for interaction.

1.8 Future Enhancements

Potential future enhancements include expanding the model to handle additional languages, improving the model's accuracy through continuous training, and incorporating user feedback for refining categorization.

1.9 Conclusion

The Afaan Oromoo News Classification System aims to contribute to the development of language-specific tools for content organization and management, catering to the linguistic and cultural needs of the Afaan Oromoo-speaking community.

2. Model Overview

Model Name: AfaanOromoNewsClassifier

Model Type: Text Classification Model using Natural Language Processing (NLP)

2.1 Model Architecture

The Afaan Oromoo News Classification System utilizes a deep neural network architecture for effective text classification. The model is designed to categorize Afaan Oromoo news articles into predefined topics or categories. The following sections provide an overview of the key components and layers within the architecture.

2.1.1 Key Components

1. Embedding Layer

The Embedding Layer is responsible for converting input Afaan Oromoo text into dense numerical vectors. It captures the semantic relationships between words, allowing the model to understand the contextual meaning of each term.

2. Global Average Pooling 1D Layer

The Global Average Pooling 1D Layer reduces the spatial dimensions of the embedded representation. By averaging across the sequence, it extracts essential features from the input, providing a more concise representation for further processing.

3. Dense Layers

The architecture includes multiple Dense Layers, which are fully connected neural network layers. These layers learn hierarchical representations of the input text, enabling the model to discern complex patterns and relationships within the Afaan Oromoo news articles. Rectified Linear Unit (ReLU) activation functions introduce non-linearity to the network.

4. Output Layer

The Output Layer employs the softmax activation function to produce probability distributions over the predefined categories. Each category is assigned a probability, and the category with the highest probability is chosen as the predicted label for the input article.

2.1.2 Tokenization and Padding

Before inputting text into the neural network, Afaan Oromoo text undergoes tokenization. This process converts words into numerical tokens, enabling the model to process and understand the input. Padding is applied to ensure consistent input lengths, as neural networks require fixed-size inputs.

2.2 Model Training Approach

2.2.1 Data Collection and Preprocessing

1. Data Collection

A diverse dataset of Afaan Oromoo news articles is collected from various online sources. The dataset covers a wide range of topics, ensuring the model's exposure to diverse language patterns.

2. Data Preprocessing

The collected data undergoes a comprehensive preprocessing pipeline. This includes tokenization, normalization, removal of stop words, and language-specific stemming. The preprocessing steps aim to enhance the quality of the input data and improve the model's ability to generalize.

2.2.2 Model Training

1. Loss Function and Optimizer

The model is trained using the categorical cross-entropy loss function, which measures the dissimilarity between predicted and actual category distributions. The Adam optimizer is employed for parameter updates, facilitating efficient convergence during training.

2. Training Process

The training process involves iterative updates to the model's parameters based on backpropagation. During each iteration, the model learns to minimize the loss, improving its ability to accurately categorize Afaan Oromoo news articles.

3. Validation Set

A portion of the dataset is reserved as a validation set to monitor the model's performance during training. The validation set helps detect potential overfitting and guides hyperparameter tuning decisions.

The combination of a robust model architecture, effective tokenization, and a meticulous training approach contributes to the Afaan Oromoo News Classification System's ability to accurately categorize Afaan Oromoo news articles.

2.3 Model Evaluation

The evaluation of the Afaan Oromoo News Classification System is crucial for assessing its performance, generalization capabilities, and reliability. This section outlines the key metrics and methodologies used to evaluate the model's effectiveness in categorizing Afaan Oromoo news articles.

2.3.1 Evaluation Metrics

Accuracy

Accuracy measures the percentage of correctly classified articles out of the total number of articles. It provides a general overview of the model's overall performance.

2.3.2 Methodology

1. Test Set

The model is evaluated on a separate test set that was not used during training or validation. This ensures an unbiased assessment of its performance on new, unseen data.

2. Prediction and Ground Truth

The model generates predictions for each article in the test set, and these predictions are compared to the ground truth labels to compute evaluation metrics.

2.4 Model Usage

The Afaan Oromoo News Classification System is designed to categorize Afaan Oromoo news articles into predefined topics or classes. This section provides a guide on how to use the trained model for making predictions on new text data.

Prerequisites

Before using the model, ensure that you have the necessary dependencies and files:

1. Trained Model: The saved model file (e.g., `model_20231116_accuracy_0.9263.h5`) obtained from the training process.

2. Tokenizer: The saved tokenizer file (e.g., `tokenizer.joblib`) used for tokenizing input text.

3. Label Encoder: The saved label encoder file (e.g., `label_encoder.joblib`) for mapping class labels.

Steps for Model Usage

1. Load Dependencies

In your Python script or environment, import the required libraries and modules:

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
import joblib
```

2. Load the Model, Tokenizer, and Label Encoder

```
main_path = '/models/trained_models/'
model_filename = 'model_20231116_accuracy_0.9263.h5'
tokenizer_filename = 'tokenizer.joblib'
label_encoder_filename = 'label_encoder.joblib'

model_path = main_path + model_filename
tokenizer_path = main_path + tokenizer_filename
label_encoder_path = main_path + label_encoder_filename

def load_model_and_dependencies(model_path=model_path, tokenizer_path=tokenizer_path,
label_encoder_path=label_encoder_path):
    # Load the model
    loaded_model = load_model(model_path)
    # Load the tokenizer
    with open(tokenizer_path, 'rb') as tokenizer_file:
        loaded_tokenizer = joblib.load(tokenizer_file)
    # Load the label encoder
    loaded_label_encoder = joblib.load(label_encoder_path)
    return loaded_model, loaded_tokenizer, loaded_label_encoder
model, tokenizer, label_encoder = load_model_and_dependencies()
```

3. Preprocess Input Text

Use the preprocessor to preprocess the input text before making predictions:

```
from preprocessing.preprocessing_pipeline import PreprocessingPipeline
def preprocess_input_text(text):
    preprocessor = PreprocessingPipeline()
    preprocessed_text = preprocessor.preprocess(text)
    return preprocessed_text
```

4. Make Predictions

```
def predict_category(text, loaded_model, loaded_tokenizer, loaded_label_encoder):
    # Preprocess the text
    preprocessed_text = preprocess_input_text(text)

    # Tokenize and pad the sequence
    sequence = loaded_tokenizer.texts_to_sequences([preprocessed_text])
    max_length = loaded_model.input_shape[1]
    padded_sequence = pad_sequences(sequence, maxlen=max_length, padding='post')

    # Make predictions
    predictions = loaded_model.predict(padded_sequence)

    # Decode the predictions using the loaded label encoder
    predicted_category_index = predictions.argmax()
    predicted_category = loaded_label_encoder.inverse_transform([[predicted_category_index]])[0]

    return predicted_category
```

5. Example Usage

```
if __name__ == '__main__':
    # Example input text
    input_text = “””Your sample Afaan Oromoo news article goes here.”””
    # Make a prediction
    predicted_category = predict_category(input_text, model, tokenizer, label_encoder)
    print("Predicted Category:", predicted_category)
```

Adjust the `input_text` variable with your own Afaan Oromoo news article text to see the model's prediction.

2.5 Model Deployment

Model deployment is a crucial step in bringing a machine learning model into production, allowing it to make real-time predictions on new data. In this section, we discuss the deployment process for the Afaan Oromoo News Classification System.

Deployment Steps

1. Hosting the Flask API

The Flask API acts as the interface between the model and external systems. Follow these steps to deploy the API:

- *File Structure:*
  - Create a directory for the API (e.g., api).
  - Place the Flask application (app.py), model loader (model_loader.py), and any other necessary files in the directory.
- *Dependencies:*
  - Ensure all required dependencies are installed. You can use a virtual environment to manage dependencies.
- *Web Server:*
  - Choose a production-ready web server such as Gunicorn or uWSGI to host the Flask application.

2. Configuration

Adjust the Flask application configuration for production:

- *Debug Mode:*
  - Set the Flask app to run in production mode by setting debug=False in the app.py file.
- *Security:*
  - Implement security measures, such as using HTTPS, to protect data transmission.

3. Scalability

Consider scalability to handle increased traffic and concurrent requests:

- *Load Balancing:*
  - Implement load balancing to distribute incoming requests across multiple server instances.
- *Containerization:*
  - Explore containerization solutions like Docker to simplify deployment and scalability.

4. Monitoring

Implement monitoring and logging to track the API's performance and identify potential issues:

- *Logging:*
  - Integrate logging to capture errors, warnings, and other relevant information.
- *Metrics:*
  - Use monitoring tools to collect performance metrics and detect anomalies.

2.6 Model Maintenance

Continuous model maintenance is essential for ensuring optimal performance and adapting to changing data. Here are key maintenance activities:

Regular Updates

Periodically retrain the model with new data to improve accuracy and relevance.

Data Monitoring

Monitor incoming data for shifts or changes that may impact model performance.

Feedback Loop

Establish a feedback loop to collect user feedback and address any issues promptly.

Dependency Updates

Regularly update dependencies, including libraries and frameworks, to benefit from the latest features and security patches.

3. Conclusion

The Afaan Oromoo News Classification System demonstrates the application of machine learning techniques for text classification in the context of Afaan Oromoo news articles. The project encompasses data preprocessing, model training, deployment, and maintenance.

As the system moves into production, continuous monitoring and updates will be crucial for maintaining its effectiveness over time. The Flask API serves as a robust interface for making predictions, and regular model retraining ensures that the system stays relevant in a dynamic environment.

By following best practices in model deployment and maintenance, this project lays the foundation for a scalable and reliable Afaan Oromoo news classification solution. The journey from initial data exploration to a fully deployed and maintained model reflects the interdisciplinary nature of machine learning projects and their real-world impact.

References

Dataset:

Fana Broadcasting Corporation Afaan Oromoo [Link]

Stemming Algorithm:

Tesfaye, Debela. "Designing a Stemmer for Afaan Oromo Text: A Hybrid Approach." [Link]

Natural Language Processing Basics:

Manning, Christopher D., and Schütze, Hinrich. "Foundations of Statistical Natural Language Processing."

Model Layers and Architecture:

Goldberg, Yoav. "A Primer on Neural Network Models for Natural Language Processing."

Model Evaluation Techniques:

Brownlee, Jason. "How to Choose a Metric for Imbalanced Classification." Machine Learning Mastery.

Cross-Validation:

Varma, Rohit, and Simon, Vinayaka. "A Beginner's Guide to Cross-validation." Analytics Vidhya.