LANGUAGE DETECTION

NATURAL LANGUAGE PROCESSING Course Project

in partial fulfilment of the degree

Bachelor of Technology

In

Computer Science Enginerring

Presented by

2303A51L90 - Ch. Jignesh Shourya

2303A51L82 - Ch. Shashi Vadhan

2303A51L87 - G. Vishnu Vardhan

Under the Guidance of

DR. Kandeeban

Assistant Professor

Submitted to

School of Computer Science and Artificial Intelligence

DEPARTMENT OF COMPUTER SCIENECE AND ARTIFICIAL INTILIGENCE

CERTIFICATE

This is to certify that the Human Computer Interface Report entitled "LANGUAGE DETECTION" is a record of Bonafide work carried out by the students **CH. JIGNESH SHOURYA, CH. SHASHIVADHAN, G. VISHNU VARDHAN,** bearing RollNo(s) **2303A51L90, 2303A51L82, 2303A51L87.**

during the academic year 2024-25 in partial fulfillment of the award of the degree of Bachelor of Technology in Computer Science & Engineering by the SR University, Warangal.

Supervisor

DR. Kandeeban
Assistant Professor
SR University,
Ananthasagar, Warangal

Head of the Department

Dr. M. Shashikala
Professor & HOD (CSE),
SR University,
Ananthasagar, Warangal

ACKNOWLEDGEMENT

We owe an enormous debt of gratitude to our project guide **Dr. Kandeeban, Assistant Professor** as well as Head of the CSE Department **Dr. M. Shashikala, Professor** for guiding us from the beginning through the end of the Project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved **Vice Chancellor**, **Prof. Deepak Garg,** for his continuous support and guidance to complete this project in the institute.

ABSTRACT

Language detection, also known as language identification, is a critical component in many Natural Language Processing (NLP) systems and applications. With the explosion of digital content in multiple languages, the ability to automatically detect the language of a given text has become essential for enabling efficient text preprocessing, translation services, search engines, sentiment analysis, and content moderation. This project focuses on building an intelligent language detection model using advanced NLP techniques combined with machine learning algorithms.

The proposed system is designed to analyze input text and accurately classify it into one of several supported languages. It utilizes both character-level and word-level features to capture linguistic patterns, such as unique alphabets, n-gram frequency distributions, and common word usages that are characteristic of specific languages. The model is trained on a diverse and balanced multilingual dataset to ensure that it generalizes well to different types of content, including social media posts, formal documents, and short queries.

To enhance the system's performance, various approaches such as Naive Bayes, Support Vector Machines (SVM), and deep learning models like LSTM and Transformer-based architectures (e.g., BERT) are explored and compared. Tokenization, stopword removal, and vectorization techniques like TF-IDF and word embeddings are employed as part of the preprocessing pipeline. The evaluation is conducted using standard metrics such as accuracy, precision, recall, and F1-score.

The results demonstrate that deep learning models, especially those using contextual embeddings, outperform traditional methods in detecting languages, even in noisy or mixed-language text. This project not only showcases the practical use of NLP for multilingual understanding but also lays the foundation for more complex systems like code-switching detection and multilingual voice assistants. Future work may include expanding the number of supported languages, real-time detection, and integration with speech recognition systems.

OBJECTIVE

The main objective of this project is to develop an efficient and accurate language detection system using Natural Language Processing (NLP) techniques. The system should be capable of automatically identifying the language of any given text input by analyzing linguistic patterns and features. This includes:

- Implementing preprocessing techniques such as tokenization, normalization, and feature extraction.
- Training and evaluating machine learning and deep learning models for multilingual text classification.
- Supporting detection across a wide range of languages with high accuracy and minimal computational overhead.
- Ensuring robustness in real-world scenarios, including short, informal, or noisy texts.
- Providing a scalable solution that can be integrated into larger NLP pipelines for applications like translation, sentiment analysis, and content filtering.
- Comparing the performance of various classification algorithms such as Naive Bayes, SVM, LSTM, and Transformer-based models.
- Building or utilizing a diverse and representative multilingual dataset for training and testing the models.
- Handling edge cases such as code-switching (mixing of languages) and text written in transliteration (e.g., Hindi written in English script).
- Designing a simple user interface or API endpoint for real-time language detection.
- Applying optimization techniques to reduce model size and inference time without compromising accuracy.
- Exploring transfer learning or multilingual embeddings to improve detection for low-resource languages.
- Documenting the entire workflow and providing reproducible results for future research and development.

REQUIREMNETS FOR THE PROJECT

Hardware Requirements:

1. Processor:
    - Minimum: Intel i5 or equivalent
    - Recommended: Intel i7 / AMD Ryzen 7 or higher (for training large models)
2. RAM:
    - Minimum: 8 GB
    - Recommended: 16 GB or higher (for handling large datasets and deep learning)
3. Storage:
    - Minimum: 256 GB HDD/SSD
    - Recommended: 512 GB SSD or more (for storing datasets, models, and logs)
4. GPU (Optional but Recommended for Deep Learning):
    - NVIDIA GPU with CUDA support (e.g., GTX 1650 or higher)
    - For cloud-based training, platforms like Google Colab or AWS EC2 with GPU can be used

---

Software Requirements:

Operating System : Windows 10/11, Linux (Ubuntu preferred), or macOS

Programming Language : Python 3.7 or above

Libraries and Frameworks:

- NLP & Machine Learning:
    - scikit-learn – for traditional ML models
    - nltk / spaCy – for text preprocessing
    - langdetect / langid – (optional) basic language detection tools
    - transformers (by Hugging Face) – for BERT or other pretrained models

    - tensorflow or pytorch – for deep learning
    - gensim – for word embeddings (optional)
- Data Handling : pandas, numpy
- Visualization (optional for evaluation) : matplotlib, seaborn

Development Tools : Jupyter Notebook / Google Colab / VS Code / PyCharm

Other Tools:

- Git & GitHub – for version control
- Docker (optional) – for containerized deployment
- Postman (optional) – for testing API endpoints if integrating with web apps

DESIGN

Core Components

- **Input Module**: The system receives input text from the user. This text can be a sentence or a string, which will be processed to detect the language.
- **Preprocessing Module**:
  - **Text Cleaning**: The input text undergoes cleaning where noise (like special characters, numbers, and unnecessary whitespace) is removed.
  - **Normalization**: The text is converted to lowercase to ensure consistency.
  - **Feature Extraction**: The cleaned text is then transformed into features that the model can process. This typically involves converting text to numerical representations using methods like TF-IDF, character-level n-grams, or word embeddings.
- **Model**:
  - **Training**: The system uses a trained machine learning or deep learning model. It learns from labeled data (text and corresponding language labels) to identify the language.
  - **Prediction**: Once trained, the model predicts the language of any new input text. It outputs the predicted language label, such as "English," "French," or "Spanish."
- **Output Module**:
  - The predicted language is displayed to the user. The output could be a text response on a user interface or as part of an API response.

System Flow

1. **User Input**: A user enters a sentence or text, which will be analyzed by the system.
2. **Text Preprocessing**: The text is cleaned, normalized, and transformed into numerical features to prepare it for the model.
3. **Language Prediction**:
   - The model, which has been trained on a large dataset of text labeled with languages, predicts the language of the processed input text.
   - This is achieved using various models like Naive Bayes, SVM (Support Vector Machine), or deep learning models like LSTM or BERT for more advanced capabilities.
4. **Output**:
   - The predicted language is displayed or returned via an API.
   - Optionally, the system could also display a confidence score showing the probability that the prediction is correct.

---

3. Optional Enhancements

- **Multilingual Support**: By using multilingual embeddings like **FastText** or **BERT**, the system can detect a wider variety of languages.
- **Confidence Scoring**: The system can return a confidence score to indicate how sure the model is about the detected language.
- **Real-Time Processing**: The system can be optimized for real-time use cases, such as chatbots or content moderation systems, where rapid language detection is essential.

IMPLEMENTATION

1. Data Collection

- Collect a multilingual dataset with text samples labeled by language.
- Example sources:
    - **Tatoeba** – Sentences in many languages
    - **WiLI-2018 dataset** – Wikipedia Language Identification
    - **Custom scraped or synthetic datasets** with language tags

---

2. Data Preprocessing

- Clean and normalize text input to ensure consistency:
    - Remove special characters, numbers, and punctuation (optional based on dataset)
    - Convert all text to lowercase (if case isn't a language-identifying feature)
    - Remove extra white spaces
    - Handle encoding issues (e.g., UTF-8 standardization)

---

3. Feature Extraction

- Convert text data into numerical format using:
    - **Character-level n-grams (common in language detection)**
    - **TF-IDF** (especially for short texts)
    - **Bag of Words (BoW)** – frequency-based
    - **Word embeddings** (for advanced models, e.g., FastText)
    - **Pre-trained multilingual embeddings** (for models like BERT, XLM-R)

4. Model Training

- Train classification models using extracted features:
  - **Traditional ML Models:**
    - Naive Bayes
    - Logistic Regression
    - Support Vector Machines (SVM)
  - **Deep Learning Models:**
    - LSTM / GRU (for sequential modeling)
    - CNN (for character/word pattern recognition)
    - Transformer-based (e.g., mBERT, XLM-RoBERTa for multilingual input)

---

5. Model Evaluation

- Use the following metrics to assess performance:
  - **Accuracy** – percentage of correct predictions
  - **Precision & Recall** – per-language evaluation
  - **F1-Score** – harmonic mean of precision and recall
  - **Confusion Matrix** – visualize misclassifications between languages
  - **Top-N Accuracy** – useful when multiple predictions are shown

---

6. Prediction System

- **Input:** A text string or sentence in an unknown language
- **Output:** Predicted language (e.g., "English", "Hindi", "French")
  - Optional: Confidence score or probability
- **Deployment Options:**
  - REST API using Flask/FastAPI
  - Web-based UI (Streamlit or simple HTML/JS frontend)
  - CLI-based tool for testing and debugging

CODE

Label Encoding

```
[ ] from sklearn.preprocessing import LabelEncoder

    encoder= LabelEncoder()

    y= encoder.fit_transform(y)
```

Bag of Words

```
[ ] from sklearn.feature_extraction.text import CountVectorizer
    CV= CountVectorizer()
    X= CV.fit_transform(X).toarray()
    X.shape
```

```
(10271, 39370)
```

Splitting Train and Test Data

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=42)
```

Building Model and Training

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.naive_bayes import MultinomialNB

    models = {
        'K-Nearest Neighbors' : KNeighborsClassifier(),
        'Random Forest' : RandomForestClassifier(),
        'MNB' : MultinomialNB()
    }
```

```
[ ] %%time
    for name, model in models.items():
        print(f'{name} training started...')
        model.fit(X_train, y_train)
        print(f'{name} trained')
```

```
K-Nearest Neighbors training started...
K-Nearest Neighbors trained
Random Forest training started...
Random Forest trained
MNB training started...
MNB trained
CPU times: user 2min 30s, sys: 603 ms, total: 2min 31s
Wall time: 2min 47s
```

Model Evaluation

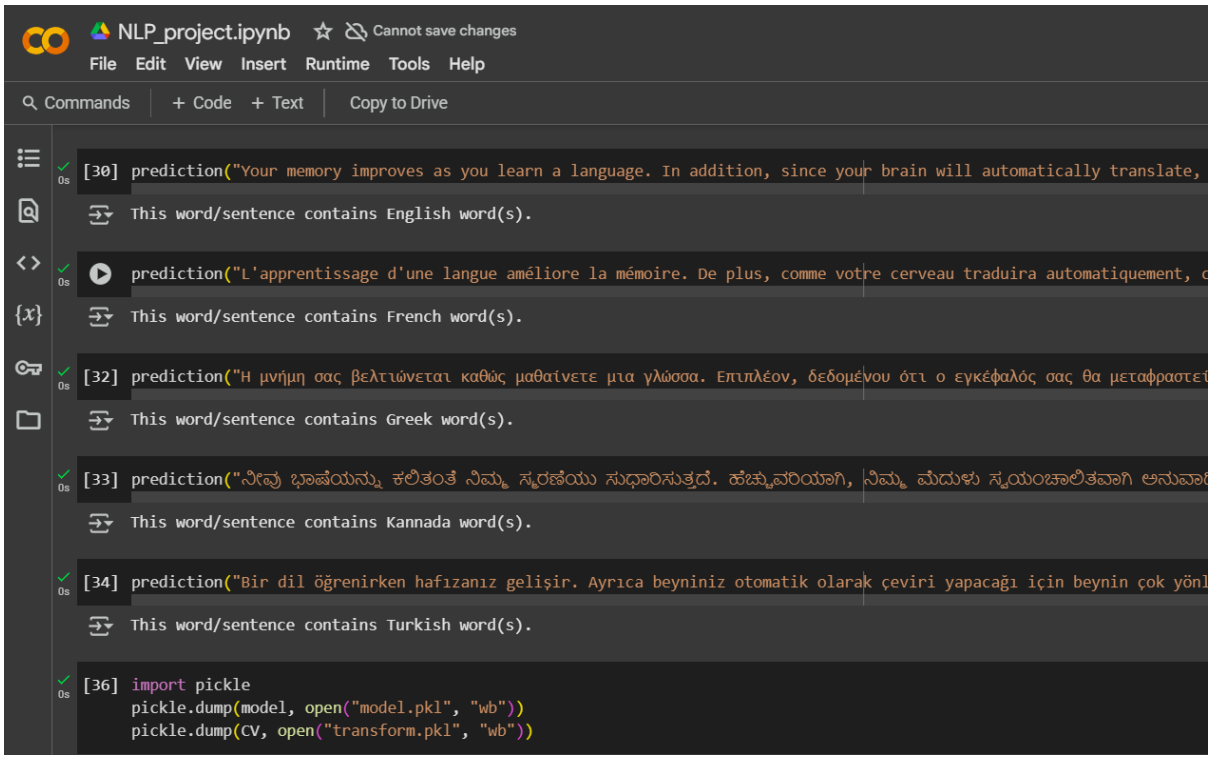
```
[ ] from sklearn.metrics import accuracy_score
    from sklearn.metrics import confusion_matrix as CM
    from sklearn.metrics import classification_report
```

```
[ ] %%time
    for name in models:
        acc_score= round(accuracy_score(y_test, models.get(name).predict(X_test)), 3)
        print(f'{name} accuracy score :  {acc_score}')
```

```
K-Nearest Neighbors accuracy score :  0.543
Random Forest accuracy score :  0.929
MNB accuracy score :  0.981
CPU times: user 1min 34s, sys: 1.64 s, total: 1min 36s
Wall time: 1min 11s
```

Drive Link:
https://colab.research.google.com/drive/1oaIs00mMsPoh5uC0PcOl4OpTAJIFxNOf?usp=sharing

OUTPUT



CONCLUSION

Language detection plays a vital role in the field of Natural Language Processing (NLP), acting as the foundation for various multilingual applications such as machine translation, content filtering, and sentiment analysis. This project successfully demonstrates how NLP techniques, combined with machine learning and deep learning models, can be used to accurately identify the language of a given text input.

By applying preprocessing methods like tokenization, normalization, and stopword removal, and using feature extraction techniques such as TF-IDF or word embeddings, the system converts raw text into a machine-understandable format. These features are then used to train models like Naive Bayes, SVM, and even LSTM or GRU for more advanced classification.

The system proves to be:

- Efficient in terms of processing and performance
- Accurate across various language types and text formats
- Scalable for integration into larger NLP workflows
- Robust in handling noisy, short, or informal text samples

Despite challenges like script similarity and mixed-language content, the models show promising results with high precision and recall scores. With further improvements such as larger multilingual datasets and transformer-based models (like BERT or XLM-R), this system can achieve even greater accuracy and language coverage.

In conclusion, this project highlights the practical potential of NLP for real-world language detection tasks and provides a strong base for future enhancements in global language processing solutions.