Language Detection using Natural Language Processing

Yashashvini Rachamallu, Bhanu Kanamarlapudi, Neel Joshi
Rachama2@msu.edu, kanamar1@msu.edu, joshine3@msu.edu

Abstract

Since the dawn of the internet, the world has come closer unlike never. Internet has completely removed the communication barriers between people in different corners of the world. With increase in the power of internet and social media, people from different language background have started to interact. However, since various people speak different languages, they have started to put content on the internet in their own language, which is very difficult for someone who does not know how to read that language. This has given rise to new form of data called multilingual data, where you have tons of data in different language. With the growing volume of multilingual data on the web, language detection has become a crucial task in various NLP applications, such as machine translation and sentiment analysis. However, previous works predominantly used machine learning models and very few variations in text vectorization for language detection. In our project, we explored the use of deep learning for the classification of languages and the effect of different embedding and text-vectorization methods. The dataset was taken as a mixture of four different datasets to facilitate better training procedures. Additionally, we have conducted experiments to evaluate the impact of different parameters on the models' performance. Our aim is to demonstrate the effectiveness of deep learning techniques for language detection in NLP and provide insights into the performance of various machine learning and deep learning models and text vectorizers for this task as well as provide insights into the most effective approaches for language detection.

1 Introduction

Language identification is a method that classifies text into a set of accessible languages. It plays a critical role in numerous NLP applications, such as autocorrection, machine translation, information retrieval, summarization, and question answering. Language identification allows appropriate autocorrection lexicons and language models to be loaded for predictive and multilingual typing. Language identification is an extensive research area, and it can be highly beneficial in breaking communication barriers caused by unknown languages.

There are two approaches to language identification: computational and non-computational. Non-computational approaches rely on linguistic expertise to recognize diacritics, symbols, and other elements. Computational approaches, on the other hand, use statistical methods to handle such challenges. Statistical approaches are more efficient when working with large amounts of text that require language classification. One advantage of statistical approach is that a single model can handle translation for numerous languages.

Over the years, several techniques have been proposed and employed for language identification. These techniques can be broadly categorized into feature extraction methods and advance deep learning based techniques. Feature extraction methods are used to transforms raw text data into numerical representations that can be processed by machine learning models. TF-IDF is a widely used statistical measure that evaluates the importance of a word in a document relative to a collection of documents. Word2Vec is a group of algorithms that learn continuous word embeddings from large text corpora. These embeddings capture the semantic and syntactic relationships between words and have been successfully utilized in various NLP tasks.

Deep learning-based approaches have gained significant popularity in recent years due to their ability to learn complex representations of data and their outstanding performance in various NLP tasks. Models such as Bidirectional Encoder Representations from Transformers (BERT), Long Short-Term Memory (LSTM), and Bidirectional Long Short-Term Memory (BiLSTM) have been successfully applied to language detection. BERT is a pre-trained model that can learn bidirectional contextualized word embeddings, while LSTM and BiLSTM are recurrent neural network architectures designed to capture long range dependencies in sequential data.

Language identification has numerous applications in various fields, such as marketing, customer service, and social media analysis. For instance, companies can use language identification to analyze customer reviews to improve their products or services.

In addition, language identification is crucial in natural language processing for sentiment analysis. Sentiment analysis involves analyzing the emotional tone of a piece of text. Accurate language identification is required to correctly analyze the sentiment of the text in different languages.

Moreover, language identification is useful in the development of chatbots and virtual assistants that can interact with users in multiple languages. Language identification can help these systems understand the language of the user's input and respond appropriately.

Finally, language identification is essential for preserving cultural heritage. It can help linguists and historians identify the language of ancient manuscripts and documents, which can provide valuable insights into the history and culture of a particular region.

The goal of this project is to use larger datasets containing more unique languages to create a classification model using custom neural networks. The project aims to compare the results of different text-vectorization methods with existing methods. This project's outcome will contribute to the development of more efficient language identification techniques for NLP applications allowing easier communication across different languages.

The next section summarizes the various models that have been proposed so far along with their metrics and drawbacks. Section 3 provides the dataset information. In section 4, we describe the methodology of our approach. Section 5 is devoted to analysis and results of numerous models. Section 6 presents our conclusive remarks and perspective ideas. Section 7 summarizes the entire project.

2 Previous Works

Language detection is a crucial task in NLP that has received significant attention from researchers in recent years. In previous works, machine learning models have been predominantly used for language detection tasks. Canvar (1994) used n-gram based text categorization, while Ahmed (2004) used n-gram based cumulative frequency addition for language identification from text.

Sarma (2018) focused on word level language identification in code-mixed social media text of Assamese, Bengali, Hindi, and English languages. They used a machine learning approach based on Support Vector Machine (SVM) and achieved a maximum accuracy of 97.5%. A recent work by H. Singh (2020) used machine learning algorithms such as Decision Tree (DT), K-Nearest Neighbor (KNN), Random Forest (RF), and Support Vector Machine (SVM) for language identification. They used a dataset of five languages namely Hindi, Punjabi, Bengali, Telugu, and Tamil, and achieved a maximum accuracy of 97.6% while using the Random Forest algorithm.

Another recent work by A. Singh (2022) gives insights of the various important terminologies of

NLP and NLG and can be useful for the readers interested to start their early career in NLP and work on relevant applications. It also focuses on the history, applications, and recent developments in the field of NLP. The third objective is to discuss datasets, approaches and evaluation metrics used in NLP.

Christian (2017) published a language identification system (LID to help us to identify spoken language, say an audio file in any language. This makes it first choice to use in technologies like Speech Recognition. Priyanka (2015) proposed a robust language identifier Stanford Language Identification Engine (SLIDE) to identify languages such as Nepali, Bengali etc. used in micro blogging websites.

Bhat (2018) studied code switching between Hindi and English multilingual speakers in twitter. They proposed a neural stacking model to uses parts of speech efficiently during parsing. A work on language identification of English and Gujarati code mix data in Patel (2020) shows how a word with same set of characters in English and Gujarati have different set of meanings and how it will be difficult for the model to identify.

Kowsari (2019) demonstrated the effectiveness of ensemble learning for text classification, where an ensemble of different models, such as LSTMs, BiLSTMs, and TFIDF-based classifiers, can be used to achieve better language detection accuracy compared to individual models.

Howard (2018) introduced the Universal Language Model Fine-tuning (ULMFiT) method, which allows for the fine tuning of pre-trained language models to achieve state-of-the-art performance on various NLP tasks. The success of BERT in language identification tasks can also be attributed to its effective use of transfer learning.

3 Dataset

Below are the four different datasets used to generate our final datasets: The links to the datasets are :

1. Data-1: https://www.kaggle.com/datasets/basilb2s/language-detection
2. Data-2: https://www.kaggle.com/code/martinkk5575/language-detection/data
3. Data-3: https://www.kaggle.com/datasets/lailaboullous/language-detection-dataset
4. Data-4: https://www.kaggle.com/datasets/chazzer/big-language-detection-dataset?select=sentences.csv

We created two different datasets by merging the above four datasets. First, we combined Data-1, Data-2 and Data-3, which contains 10267, 21859 and 12646 unique rows respectively. Figure 1 describes the boxplot of combined dataset containing approximately 45000 rows.

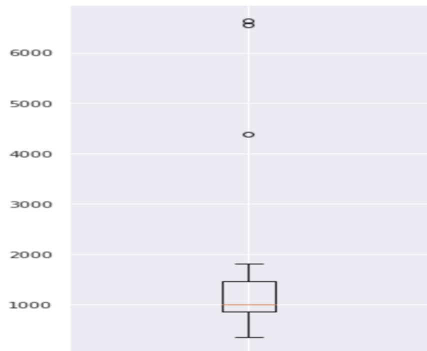

Figure 1: Boxplot of Combined dataset

The Data-4 has two files, the CSV with sentences and a JSON which has the corresponding language for abbreviation in the dataset. The Data-4 has 10^7 sentences in it, which is extremely challenging for our systems to process. To remove the outliers in above combined dataset, we used the Data-4 and picked only languages whose sentences count is in between 4000-6000 and we ended up generating Dataset-1 with around 2 lakh sentences.

The Dataset-2 is generated by combining Data-1, Data-2, Data-3 and few sentences from Data-4, to make each language in Dataset-2 to be between 10000 and 14000. Boxplot of the Dataset-2 is shown in Figure 2 and distribution plot is shown in Figure 3.

Figure 2: Boxplot of Dataset-2

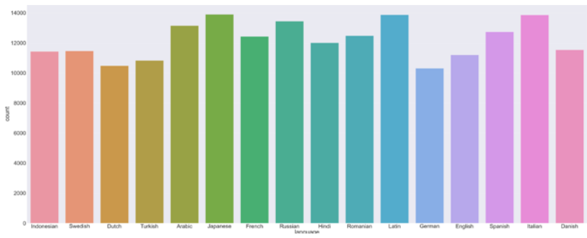

Figure 3: Distribution plot of Dataset-2

Dataset-2 contains around 2 lakh rows. From Figure 2 and Figure 3, we can observe that Dataset-2 is more normalized and no outliers.

We used Textblob to check about the sentiment of all the sentences, to make sure if we are using neutral sentences. By plotting the bar graph (Figure 4) of sentiment score, we could see that the sentences in the data are almost neutral.

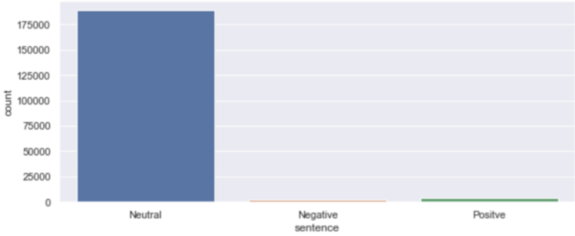

Figure 4: Bar plot for Sentiment Score

We also plotted the word cloud for different languages. On observing them we could see that some of the most frequent words are same in different languages, but the meaning of the word is different in each language. So, it's important to know about the relation and position of the words to classify the language for the text. Figure 5 and Figure 6 are sample word clouds for Arabic, French, English and Spanish.

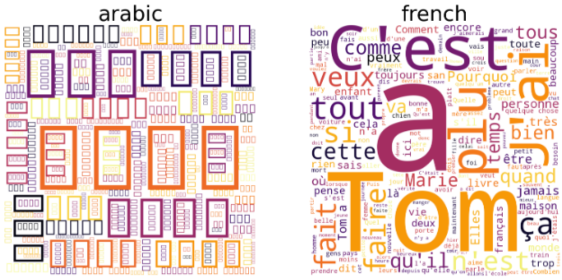

Figure 5: Arabic and French Word Clouds

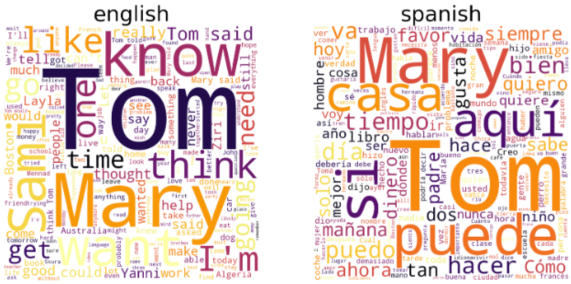

Figure 6: English and Spanish Word Clouds

4 Methodology

The following steps have been implemented on both datasets:

4.1 Text Preprocessing: We have processed the text data by converting into lower, removing special characters, punctuation, htmls, email addresses. We have also preprocessed data by applying stemming and by removing stop words for each language separately.

4.2 Classification Modeling:

4.2.1. Machine Learning: We implemented the following machine learning classification models that uses Bag-of-Words, TF-IDF vectorizer, transformer, Tokenizer, Word2Vec and N-gram analysis:

- Naïve Bayes
- Logistic Regression
- Decision Trees
- Random Forest
- Ensemble
- SVM

4.2.2. Deep Learning: We experimented with with various text vectorization techniques namely embeddings from BERT, tokenizer, combination of character, word and position analyzer to represent the text data as numerical features to use as an input to our deep learning models. We implemented a neural network with 4 variations. The variations include LSTM layers, dropout layers, spatial dropout layer, Bi-LSTM layers. We have trained our models on 5 epochs with different learning rates and used Adam optimizer and the loss function as categorical cross entropy. We also fined tuned DISTILBERT using dataset-2 on 3 epochs.

4.3 Training & Testing: The dataset is divided into 80% for training and 20% for testing.

4.4 Models implemented on Dataset-1:

On Dataset-1, both BagofWords and TF-IDF methods have been implemented for Naïve Bayes and Decision Trees classification.

Tokenizer method has been used to implement Bi-LSTM neural network model which contains 6 layers, used Adam optimizer and cross entropy loss function. Figure 7 describes the model summary.

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_13 (Embedding)    (None, 250, 100)          5000000

 spatial_dropout1d_17 (Spati  (None, 250, 100)         0
 alDropout1D)

 dense_36 (Dense)            (None, 250, 120)          12120

 spatial_dropout1d_18 (Spati  (None, 250, 120)         0
 alDropout1D)

 bidirectional_12 (Bidirecti  (None, 200)              176800
 onal)

 dense_37 (Dense)            (None, 88)                17688

=================================================================
Total params: 5,206,608
Trainable params: 5,206,608
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Bi-LSTM model summary

We also implemented a models which contains LSTM layers and different number of dense layers in order to compare performance with model containing with Bi-LSTM layers.

We utilized AutoTokenizer to load the pre-trained XLM-RoBERTa to create embeddings to the preprocessed input and designed a custom neural network for classification task.

4.5 Models implemented using Dataset-2

Using Dataset-2, numerous N-gram techniques have been implemented in order to compare the performance between word and character analysis. SVM, Logistic Regression, Random Forest and Naïve Bayes models have been implemented to evaluate the performance.

Word2Vec is used for text vectorization and Logistic regression model has been implemented, which helped us with sequential relation among words in a sentence.

We created embeddings where we are taking care of characters, words and positions using TF-IDF transformer, tokenizer and Count Vectorizer. We implemented Naïve Bayes (Figure 8), Logistic Regression (Figure 9), Random Forest (Figure 10) and Ensemble methods (Figure 11).

```
Pipeline(steps=[('features',
                 FeatureUnion(transformer_list=[('char_pipeline',
                                                 Pipeline(steps=[('vect_char',
                                                                  CountVectorizer(analyzer='char',
                                                                                  ngram_range=(2,
                                                                                               6))),
                                                                 ('tfidf_char',
                                                                  TfidfTransformer())])),
                                                ('word_pipeline',
                                                 Pipeline(steps=[('vect_word',
                                                                  CountVectorizer(ngram_range=(1,
                                                                                               3))),
                                                                 ('tfidf_word',
                                                                  TfidfTransformer())])),
                                                ('pos_pipeline',
                                                 Pipeline(steps=[('pos',
                                                                  CountVectorizer(tokenizer=<function word_tokenize at 0x77382b741320>))]))])),
                ('clf', MultinomialNB())])
```

Figure 8: Pipeline of Naïve Bayes

```
Pipeline(steps=[('features',
                 FeatureUnion(transformer_list=[('char_pipeline',
                                                 Pipeline(steps=[('vect_char',
                                                                  CountVectorizer(analyzer='char',
                                                                                  ngram_range=(2,
                                                                                               6))),
                                                                 ('tfidf_char',
                                                                  TfidfTransformer())])),
                                                ('word_pipeline',
                                                 Pipeline(steps=[('vect_word',
                                                                  CountVectorizer(ngram_range=(1,
                                                                                               3))),
                                                                 ('tfidf_word',
                                                                  TfidfTransformer())])),
                                                ('pos_pipeline',
                                                 Pipeline(steps=[('pos',
                                                                  CountVectorizer(tokenizer=<function word_tokenize at 0x77382b741320>))]))])),
                ('LC',
                 LogisticRegression(multi_class='ovr', solver='liblinear'))])
```

Figure 9: Pipeline of Logistic Regression

```
Pipeline(steps=[('features',
                 FeatureUnion(transformer_list=[('char_pipeline',
                                                 Pipeline(steps=[('vect_char',
                                                                  CountVectorizer(analyzer='char',
                                                                                  ngram_range=(2,
                                                                                               6))),
                                                                 ('tfidf_char',
                                                                  TfidfTransformer())])),
                                                ('word_pipeline',
                                                 Pipeline(steps=[('vect_word',
                                                                  CountVectorizer(ngram_range=(1,
                                                                                               3))),
                                                                 ('tfidf_word',
                                                                  TfidfTransformer())])),
                                                ('pos_pipeline',
                                                 Pipeline(steps=[('pos',
                                                                  CountVectorizer(tokenizer=<function word_tokenize at 0x77382b741320>))]))])),
                ('RF', RandomForestClassifier(max_depth=10, random_state=0))])
```

Figure 10: Pipeline of Random Forest

```
VotingClassifier(estimators=[('lr',
                              Pipeline(steps=[('features',
                                               FeatureUnion(transformer_list=[('char_pipeline',
                                                                               Pipeline(steps=[('vect_char',
                                                                                                CountVectorizer(analyzer='char',
                                                                                                                ngram_range=(2,
                                                                                                                             6))),
                                                                                               ('tfidf_char',
                                                                                                TfidfTransformer())])),
                                                                              ('word_pipeline',
                                                                               Pipeline(steps=[('vect_word',
                                                                                                CountVectorizer(ngram_range=(1,
                                                                                                                             3))),
                                                                                               ('tfidf_word',
                                                                                                TfidfTransformer())])),
                                                                              ('...
                                                                                                                ngram_range=(2,
                                                                                                                             6))),
                                                                                               ('tfidf_char',
                                                                                                TfidfTransformer())])),
                                                                              ('word_pipeline',
                                                                               Pipeline(steps=[('vect_word',
                                                                                                CountVectorizer(ngram_range=(1,
                                                                                                                             3))),
                                                                                               ('tfidf_word',
                                                                                                TfidfTransformer())])),
                                                                              ('pos_pipeline',
                                                                               Pipeline(steps=[('pos',
                                                                                                CountVectorizer(tokenizer=<function word_tokenize at 0x77382b741320
>))]))])),
                                              ('RF',
                                               RandomForestClassifier(max_depth=10,
                                                                      random_state=0))]))])
```

Figure 11: Pipeline of Ensemble Model

We used AutoTokenizer to load the pre trained DistilBERT-base-uncased model and to fine-tune DistilBERT-base-uncased model, we used AutoModelForSequenceClassification to load. Figure 12 explains pre-trained DistilBERT-base-uncased classification model with number of labels = 16.

```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=16, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

Figure 12: Pre-trained DistilBERT-base-uncased classification model

We also utilized AutoTokenizer to load the pre-trained XLM- RoBERTa to create embeddings to the preprocessed input and designed a custom neural network for classification task. Figure 13 provides the custom model summary.

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_ids (InputLayer)          [(None, 200)]        0

__________________________________________________________________________________________________
attention_mask (InputLayer)     [(None, 200)]        0

__________________________________________________________________________________________________
tfxlm_roberta_for_sequence_clas TFSequenceClassifier 278045186   input_ids[0][0]
                                                                 attention_mask[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 2)            0           tfxlm_roberta_for_sequence_classi
__________________________________________________________________________________________________
dense (Dense)                   (None, 512)          1536        flatten[0][0]
__________________________________________________________________________________________________
dropout_38 (Dropout)            (None, 512)          0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 128)          65664       dropout_38[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 17)           2193        dense_1[0][0]
==================================================================================================
Total params: 278,114,579
Trainable params: 278,114,579
Non-trainable params: 0
```

Figure 13: Custom Model Summary

5 Results And Analysis

Below table 1 represents the classification accuracies of models by varying embedding styles on Dataset-1:

Model	Accuracy
BoW + Naïve Bayes	87.1
BoW + Decision Trees	78.5
TF-IDF + Naïve Bayes	85
TF-IDF + Decision Trees	76
XLM-RoBERTa + Custom NN	72
LSTM + 1 Dense	87
LSTM + 2 Dense	86.7
Bi-LSTM + 2 Dense	85.6

Table 1: Dataset-1 Results

From the above table, the models that used bag of words as text vectorizer method has higher accuracy compared to models that used TF-IDF as text vectorizer. The BoW + Naïve Bayes and LSTM + 1 Dense models have almost similar accuracy and outperformed other models.

It is interesting to note that the accuracy of model using XLM-RoBERTa embeddings is quite less compared to other models. The reason for such an output is due to less amount of training data for few languages and model’s inability to perform identification.

Below table 2 represents the classification accuracies of models by varying embedding styles on Dataset-2:

Model	Accuracy
Unigram(Word) + NB	82
Unigram(Word) + LR	88

Unigram(Word) + SVM	88
Unigram(Word) + RF	87
Bigram (Word) + NB	33
Bigram (Word) + LR	35
Bigram (Word) + SVM	35
Bigram (Word) + RF	35
Trigram (Word) + NB	13
Trigram (Word) + LR	15
Trigram (Word) + SVM	15
Trigram (Word) + RF	15
3 Char gram + NB	84
3 Char gram + LR	92
3 Char gram + SVM	92
3 Char gram + RF	89
4 Char gram + NB	74
4 Char gram + LR	82
4 Char gram + SVM	83
4 Char gram + RF	80
Word2Vec + LR	68.3
Char+Word+Pos+ NB	**97.7**
Char+Word+Pos + LR	97.3
Char+Word+Pos + RF	91
Char+Word+Pos + Ensemble	97.6
Fine-tuned DistilBERT	88.7
XLM-RoBERTa + Custom NN	80.1

Table 2: Dataset-2 Results

We started to analyze the performance on Dataset-2 by implementing numerous character and word N-grams. Among word N-grams, Unigram out-performed and among char N-grams, 3-char gram out-performed. Between the word and character N-grams, 3 char gram performed best.

To understand the relation between the words, we have implemented Word2Vec. Our analysis from N-gram and Word2Vec led us to consider position along with character analysis and word analysis during the embedding phase. From the above table, it can be seen that models using these as embeddings outperformed the rest. Among the models using these embeddings, Naïve Bayes performed the best.

Due to constraint on computational resources, we had to reduce our sentences to 8000 that resulted with 88.7% accuracy for Fine tuned DistilBERT and 80.1% accuracy on XLM-RoBERTa and custom NN. Reasons for these models to not performing better compared to ML models could be due to lesser size of dataset and due to less number of layers for the custom NN model.

6 Conclusion

Language detection is an important tool for facilitating communication and ensuring that content is appropriate and accessible to diverse audiences. Language detection plays a crucial role in analyzing language usage and trends to provide insights into customer behavior, sentiment analysis. Companies operating globally use language detection to provide users with localized content and services. Language detection can also be used for security purposes. These applications of language detection attracted us to choose this topic. We have experimented with various embedding techniques, machine learning and deep learning models. From our experiments, we have observed that considering position along with character and word analysis plays a crucial role during embedding phase. Among all our implementations, the model with Char+Word+Pos+NB has given the best accuracy of **97.7%.**

7 Future Work

Language detection models can be improved over time by continuously learning from new data. We want to develop models that can learn and adapt the new language variations. Our current language detection model detects only one language at a time. However, in many real-world scenarios, it is necessary to detect multiple languages within the same document. We plan to implement a multi-language model identifier. Like many language detection models, our models also struggle with accurately identifying low-resource languages. Our future work is to improve the performance of low-resource language identifications. We also plan to develop a user interface to accept text or document as input and identify and parse the text to desired language. We also like to explore domain specific language detection as language usage vary significantly depending on domain or industry.

8 Workload Distribution

Below table 3 provides the workload distribution of the team members for the entire project.

Student	Task
Yashashvini Rachamallu	Proposal, Datasets collection, Preprocessing, classification models implementation, Training & Testing
Bhanu Kanamarlapudi	Preprocessing, vectorization, classification models implementation, Training & Testing , Mid report
Neel Joshi	Training & Testing, Final report

Table 3: Work Distribution

References

N. Sarma, S. R. Singh and D. Goswami, "Word Level Language Identification in Assamese-Bengali-Hindi-English Code-Mixed Social Media Text," 2018 International Conference on Asian Language Processing (IALP), Bandung, Indonesia, 2018, pp. 261-266, doi: 10.1109/IALP.2018.8629104.

W. B. Canvar and J. M. Trenkle. N-gram based Text Categorization. Proceedings of Symposium on Document Analysis and Information Retrieval, University of Nevada, Las Vegas, pp. 161-176, 1994.

B. Ahmed, S.-H. Cha, and C. Tappert. Language Identification from Text Using N-gram Based Cumulative Frequency Addition. Proceedings of Student/Faculty Research Day, CSIS, Pace University, 2004.

Pujeri*, B. P., & Sai D, J. (2020). An anatomization of language detection and translation using NLP techniques. *International Journal of Innovative Technology and Exploring Engineering*, *10*(2), 69–77. https://doi.org/10.35940/ijitee.b8265.1210220.

Khurana, D., Koli, A., Khatter, K., & Singh, S. (2022). Natural language processing: State of the art, current trends and challenges. *Multimedia Tools and Applications*, *82*(3), 3713–3744. https://doi.org/10.1007/s11042-022-13428-4.

Christian Bartz, Tom Herold, Haojin Yang, Christoph Meinel(2017). Language Identification Using Deep Convolution Recurrent Neural Networks. https://arxiv.org/abs/1708.04811.

Priyanka Mathur, Arkajyoti Misra, Emrah Budur(2015). Language Identification from Text Document. https://cs229.stanford.edu/proj2015/324_report.pdf

Bhat, Irshad Ahmad, et al. "Universal Dependency parsing for Hindi-English code-switching." *arXiv preprint arXiv:1804.05868* (2018).

D. Patel and R. Parikh, "Language Identification and Translation of English and Gujarati code-mixed data," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 2020, pp. 1-4, doi: 10.1109/ic-ETITE47903.2020.410.

Kowsari, K.; Jafari Meimandi, K.; Heidarysafa, M.; Mendu, S.; Barnes, L.; Brown, D. Text Classification Algorithms: A Survey. *Information* **2019**, *10*, 150. https://doi.org/10.3390/info10040150

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.