ss

Classification and Name Entity Recognition on Code-Mixed English-Telugu Social Media Text

by

Aditya Relangi

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Dr. Vinod Variyam

Lincoln, Nebraska

August, 2019

Classification and Name Entity Recognition on Code-Mixed English-Telugu Social Media Text

Aditya Relangi, M.S.

University of Nebraska, 2019

Adviser: Dr. Vinod Variyam

Code-mixing (CM) is the phenomenon of using the morphemes of one language in another during speech or text. CM is a common occurrence among multilingual and bilingual communities. Even though a lot of advances have been made in natural language processing of monolingual English text, research is lacking when it comes to code-mixed data. In this project, we focus on collecting and analyzing a code-mixed dataset for English-Telugu. Telugu is a Dravidian language native to India and has about 81 million speakers. A large number of Telugu speakers are bilingual and do code-mixing in speech and text. We collect code-mixed social media text from Twitter, annotate the data into multiple classes for classification and name entity recognition. We evaluate the performance of multiple text classification approaches on code-mixed data. Our analysis shows the SVM classifier performs best for this classification task achieving an accuracy of 85.24 %.

The second part of the project focuses on extracting named entities from code-mixed data. Named entity recognition (NER) is a subtask of information extraction where nouns of extracted and identified into relevant categories. Name entity recognition is an important area in natural language processing and machine learning as it can be used for categorizing text, automated question answering and similar applications. We evaluate the accuracy of three different NER libraries on code-mixed data and come up with an alternative approach to improve their accuracy. This approach provides a marginal improvement in the accuracy. We also detail the real-time code-mixed tweet processing web application including its architecture, implementation and deployment details.

COPYRIGHT

ACKNOWLEDGMENTS

I want to extend my heartfelt thanks to Dr. Vinod Variyam. This project would not have been possible without his support and encouragement. If it were not for him, I would not have been able to complete this project for the Masters program.

I also thank my family for being patient with me as I kept postponing working on this program. I would also like to thank Dr. Ashok Samal and Dr. Stephen Scott for agreeing to be part of my defense committee.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Language is the primary medium through which ideas are communicated. Just as new ideas emerge and evolve over time, language too undergoes transformation. This transformation might involve borrowing vocabulary from other languages, shedding once popular terms because of a lack of usage, morphing words from other languages to express the meaning in a native language. This phenomenon is more commonly observed in multi-lingual societies. Multilingualism is the use of more than one language in speech or written text. Multilingualism arises because of the need to communicate across speech communities. Multilingual speakers often switch or mix between languages during speech or text and is called as code mixing [1]. The phenomenon of mixing languages in text is widely observed on social media websites. Users sometimes start a sentence in one language, switch to another in the middle and finally end it in another language.

Code Mixing is widely observed in south-east Asian countries. In India where multiple languages are spoken, code mixing is a commonly occurring phenomenon [2]. In this project, we focus on one such language pair English-Telugu. Telugu is a Dravidian language with about 81 million speakers in India according to the 2011 Indian census [3]. English is used by most Telugu speaking populace in their day to day conversations [4]. This is prominently observed in the day to day interactions on social media. Although there has been some research on the area of code mixed languages[5] [6], work on the language pair of English-Telugu is lacking.

When it comes to textual analysis of real world data, there is no demarcation of the kind of language used. On social media, users tend to use English, Telugu, a mix of both the languages also Romanized Telugu, which is phonetic Telugu written in English. In order to understand or extract any information from this text, we need to be able to classify the type of text and extract information from the text.

In this project, we collect a rich dataset of code mixed data and build different machine learning models to measure their performance on classifying the data. Among the six different classifiers evaluated, SVM achieves the highest accuracy at 85.24%. In the second part of the project, we measure the effectiveness of three different machine learning approaches in name entity recognition on code mixed data. We propose a new approach for named entity recognition on code mixed data which demonstrates improvement among all the three approaches. Even though we notice an improvement of 51.16% on code-mixed data for our best case scenario, the overall accuracy rate when compared against name entity recognition of English data is quite low. We present this project and the data as a baseline for further research in this area.

1.1 Terminology

Multilingualism: Multilingualism is the use of more than one language, either by an individual speaker or by a community of speakers.

Code-Mixing: Code-mixing is the embedding of words or phrases of one language into another.

Code Switching: Code switching is the alternation of one or more languages in the context of a single conversation.

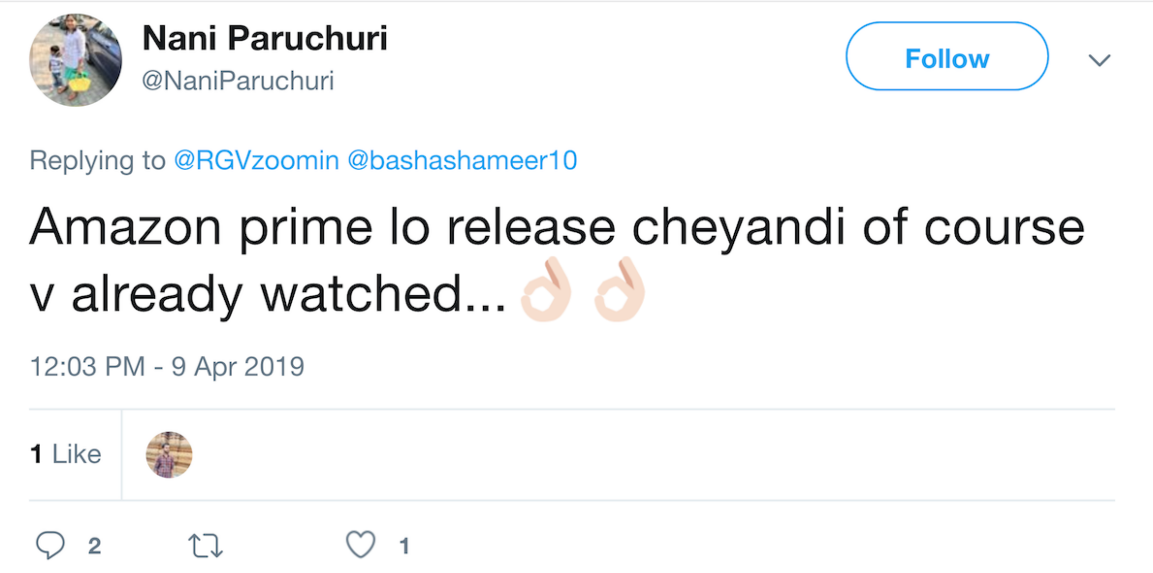
Romanized Telugu: The transliteration of Telugu into Roman script based on phonetics.

The distinction between code switching and code-mixing exists in certain fields of linguistics, for the purposes of this project, we use both the terms interchangeably.

Romanized Telugu is widely used on social media and instant messages. Telugu words are written in English for a phonetic representation.

1.2 Code-Mixed English-Telugu and Romanized Telugu Social Media Posts

Figure 11 is an example of a code-mixed English-Telugu tweet.



Nani Paruchuri
@NaniParuchuri

Replying to @RGVzoomin @bashashameer10

Amazon prime lo release cheyandi of course v already watched...👌👌

12:03 PM - 9 Apr 2019

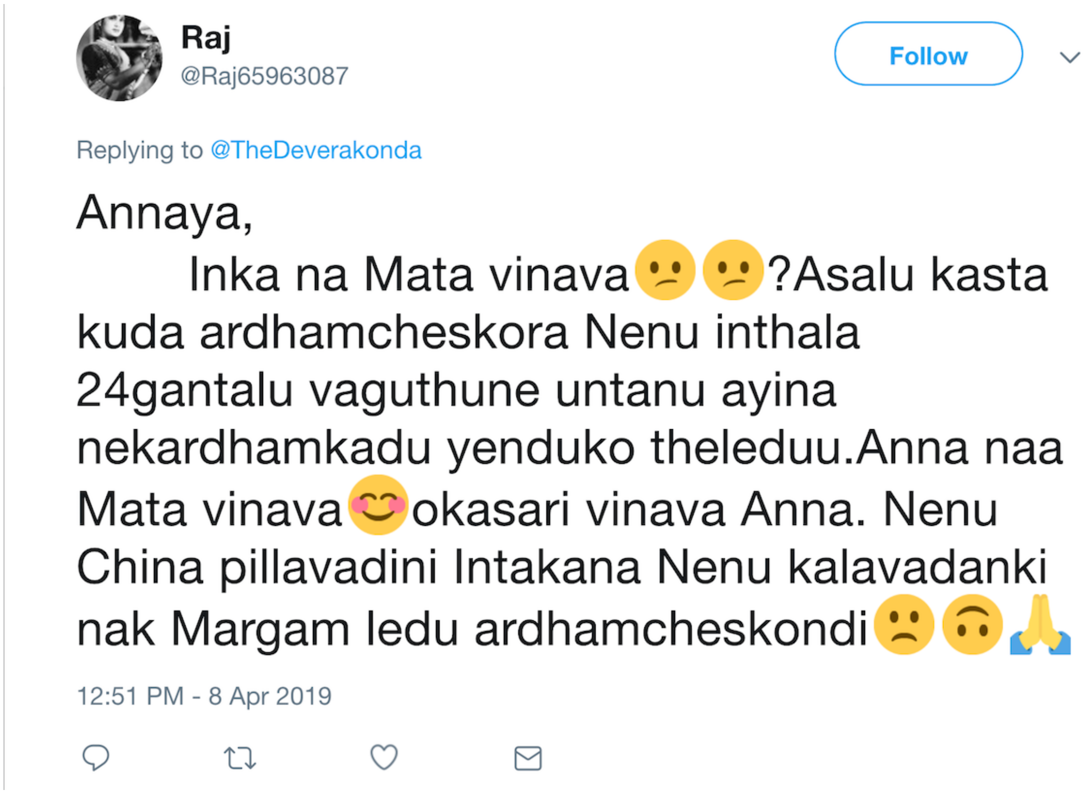
1 Like

Figure 11: Code-mixed tweet

The translation of the text in the tweet Figure 11 is “Release on Amazon Prime, of course we already watched it.”

The translation of the text in the Romanized Telugu tweet in Figure 12 is “Brother, will you listen to me? At lease understand me a little, I keep babbling 24 hours but I don’t get why you don’t understand. Listen to me brother. I’m a young kid, there is no other way for me to meet you, try to understand.”

Figure 13 is an example of a Telugu tweet and it translate into “Christians all across the world celebrate this festival every year.”



Raj
@Raj65963087

Replying to @TheDeverakonda

Annaya,
Inka na Mata vinava😕😕?Asalu kasta kuda ardhamcheskora Nenu inthala 24gantalu vaguthune untanu ayina nekardhamkadu yenduko theleduu.Anna naa Mata vinava😊okasari vinava Anna. Nenu China pillavadini Intakana Nenu kalavadanki nak Margam ledu ardhamcheskondi🙁🙃🙏

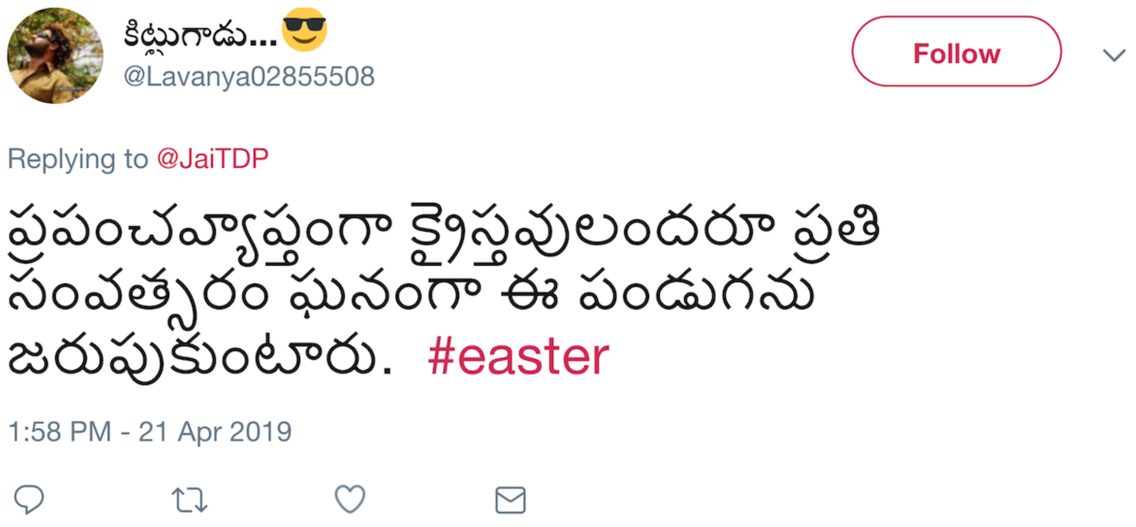
12:51 PM - 8 Apr 2019

Figure 12: Romanized tweet

1.3 Contribution of this project

The major contributions of this project are:

1. Collecting and annotating code-mixed English-Telugu social media data from Twitter with language classes.
2. Performance evaluation of various text classification algorithms to identify code-mixed tweets.
3. Annotating named entities for the tags Person, Organization and Location in code-mixed English-Telugu tweets.



కిట్టుగాడు...😎
@Lavanya02855508

Replying to @JaiTDP

ప్రపంచవ్యాప్తంగా క్రైస్తవులందరూ ప్రతి సంవత్సరం ఘనంగా ఈ పండుగను జరుపుకుంటారు. #easter

1:58 PM - 21 Apr 2019

Figure 13: Telugu tweet

4. Performance evaluation of different Name entity recognition(NER) models on code-mixed tweets.

The results from the project can act as a baseline for future research.

1.4 Organization of this project

This project is organized into seven chapters. We began this chapter by providing the contextual background on code-mixed data in social media and our motivation to choose this topic. We then introduce the terminology used in this project followed by examples of code-mixed and Romanized Telugu text. In Chapter 2, we present a literature survey of existing research in this area.

In Chapter 3, we focus on how the data is collected for the project, and an introduction to the annotation tool built for classifying the data. The steps taken to clean and normalize the data are introduced in detail. We also look at the open source tool Doccano[7] which is used for the purpose of NER tagging the code-mixed data.

Chapter 4 provides information about how the data is encoded for natural language processing. We explain the different text classification algorithms used to for the multi-label classification problem. The concepts behind each approach is explained and a comparison of their performance is presented.

We introduce the concept of NER and it's importance in text processing in chapter 5. We evaluate different models to extract named entities for Person, Organization and Location on code-mixed data and present our approach that shows improvement across all models with the highest improvement of 51.16%.

Chapter 6 details the architecture and implementation of the real time code-mixed text processing application that performs text classification and name entity recognition on live social media text streamed from the Twitter streaming API.

Finally in Cshapter 7, we present our conclusions and the potential future work that can be performed in this area. We also discuss how these approaches can possibly be applied for code-mixed data with other English language pairs.

Chapter 2

Literature Survey

Research into code mixing has early roots in the fields of linguistics and sociology [8] [9] [10]. Researchers like Annamalai have focused on [2] code-mixing in the context of Indian languages establishing English language pairing with the Indian languages Hindi, Telugu, Tamil, Bengali, Kannada and Marathi.

Linguists like Muysken [11] [12], Garafanga and Torras[13] established the terms 'Code mixing', 'Code switching' as umbrella terms to include any type of language mixing as it is not always clear where code mixing begins and word loaning ends [14].

Even though code mixing has been a focus of research in the field of linguistics for the past three decades, research in the field of computer science has been active only since the last decade. The earliest research observed in computer science in code mixing is observed in the work of Solorio et al [15]. Their initial work focuses on identifying the points of code-switching in English Spanish text. This work focuses on building a part of speech tagger for code mixed English Spanish text collected from a conversation between three speakers from a bilingual background [16]. The best performance observed in tagging the parts of speech text is by using an algorithm that combines the features of both an English and Spanish parts of speech tagger.

The research presented by Elfardy et al [17] and ElGhamdi et al [18] also shows that the best results are observed by combining the features of multiple monolingual models for token level language identification and part of speech tagging of code-mixed English Arabic datasets.

Word level language identification is vital in the part of speech tagging of code mixed text. Das et al[19] report a study to detect language boundaries at the word level in chat message corpora in mixed English-Bengali and English-Hindi. The dataset for this paper is collected from social media websites Facebook and Twitter[19]. A similarly compiled English Hindi dataset in the work done by Baheti et al observes the best results for training code-mixed models are achieved by training data on monolingual in sequence and finally training it on the code-mixed data.

Part of speech tagging lays the foundation for name entity recognition. Sasidhar et al presented a survey on name entity recognition on Indian languages and show the dearth of research due to the lack of annotated data and the agglutinative nature of the languages [20]. There has been considerable amount of work on NER in English [21]. Some of these approaches rely on linguistic features [22], while others use machine learning to extract named entities [23] [24]. There is limited research on NER in Telugu. Srikanth et al use a linguistic approach [25] while Shishtla et al use conditional random fields [26], both the approaches built on small datasets.

The FIRE 2016 workshop is the first attempt at extracting named entities from code-mixed English Indian language pairs[27]. The workshop established a baseline dataset and results for code mixed English-Hindi and English-Tamil. Different approaches were presented and evaluated with the neural network approach by Irshad et al providing the best performance for English-Hindi dataset [28], while the hybrid approach of using linguistic features combined with a conditional random field gave the best results for English-Tamil dataset [29].

Vijayanand et al have shown through their work that the writing system for both the Tamil and Telugu languages is same and share common properties during transliteration which would help in developing a common generator with different production algorithms based on the South Indian Languages like Kannada, Malayalam, Telugu and Tamil [30].

This project attempts to fill the gap in establishing a baseline dataset for document level classification of code-mixed dataset and name entity recognition on code-mixed English-Telugu dataset.

Chapter 3

Dataset

Code-mixed data is not as readily available as English data. In order to compile the dataset we relied on the social media micro-blogging platform Twitter.

3.1 Data Collection

The data is collected over a period of seven days. A list of twitter handles that are prominent public figures in the Telugu speaking areas were identified. These handles typically tweet in English, Telugu and Code-mixed English Telugu and are from different fields such as sports, films, politics. In order to collect a wide variety of the data, we chose to focus on collecting the replies received to the tweets made by these handles.

Twitter provides a streaming API endpoint [31]. We plugged our listener program written in Golang to collect the data from this stream and save this data into a Postgres database. The raw text was stored in the database as is without any modifications. We collected 992,538 samples of this dataset over the duration our application was live. The raw data is made available online as part of the Github repository.

3.2 Data Annotation

As mentioned in the previous chapter, we intend to perform two different tasks. One is to build a classifier to identify the type of the text and the second is to extract any named entities present in the dataset. Since the raw data we collected from the Twitter stream does not have any information that might help us classify this data, we need to annotate the data so we can use it to build our machine learning models.

Since both our tasks are different, we used different tools to annotate our data.

3.2.1 Text Classification

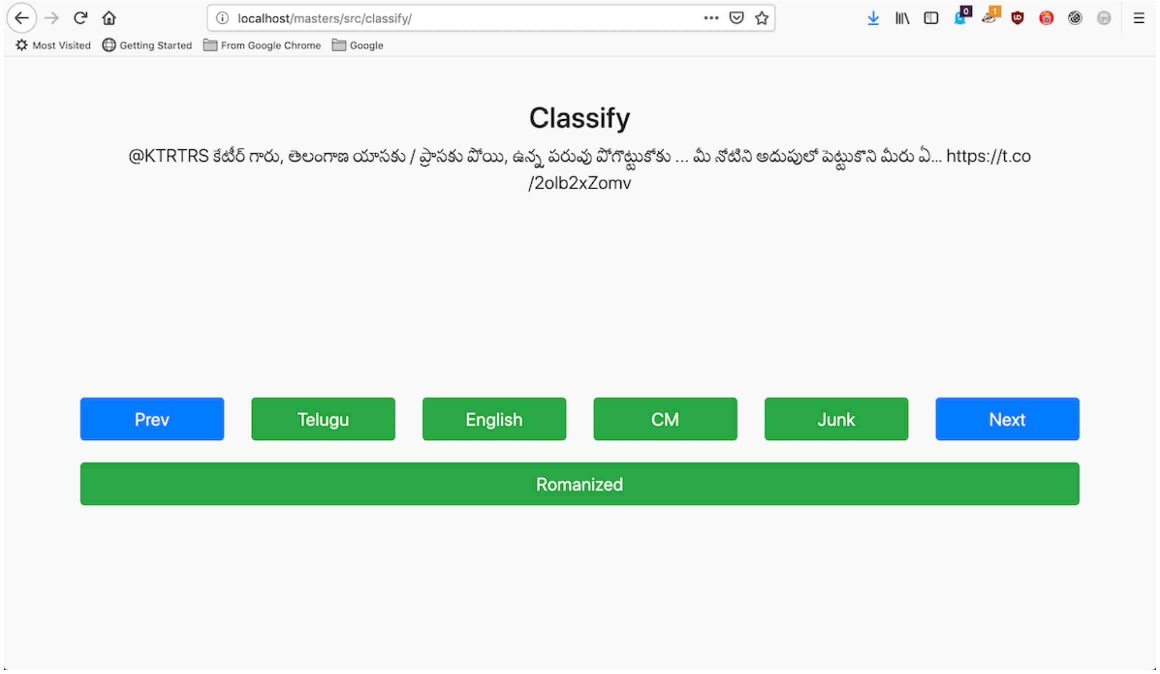
The first data labeling task is to classify the type of text. The text can be classified into one of the types listed in Table 3.2.1 with their respective titles.

Table 31: Text classification classes

Class	**Label**	**Example**
English	label_E	For all those who are speculating how we are releasing Lakshmis NTR ,please read these excerpts from the Honourable AP high court order
Telugu	label_T	కేవలం దీక్షలతో కాకుండా మత్స్యకారుల సమస్యని మరింత ఉధృతంగా రాష్ట్రం నలుమూలలా, దేశమంతా తెలిసే విధంగా చేయటానికి ప్రజా ఉద్యమాలకి పుట్టిల్లు అయిన శ్రీకాకుళం జిల్లాకి ఈ నెల 21న రావటానికి సిద్ధంగా ఉన్నాను. - @PawanKalyan
Code Mixed	label_CM	Election code vunnapudu CBN Em chesthadra? Review meetings ke permission tisukovali ayana. Police and all other govt officials working now under EC, with orders of EC. Nuvvu Ne publicity pichi. Bagane plan chesaru ycp vaallu
Romanized	label_R	Telengana lo Rajakeyalu cheyaru. Ayana andhra lo ne edina Rajakeya konam chustharu
Junk	label_J	RT @AnkurRoj: #justiceforebiz @hydcitypolice @KTRTRS @narendramodi @PMOIndia @SushmaSwaraj https://t.co/g8GGgfDThj

When classifying the text into these categories, the language of the twitter handles and the hash tags are ignored as they can only be in English. How we handle this is explained in the Data Normalization section of this chapter.

In order to classify the text, we built a custom web application, the interface is shown in Figure 31



Classify

@KTRTRS కేటీర్ గారు, తెలంగాణ యాసకు / ప్రాసకు పోయి, ఉన్న పరువు పోగొట్టుకోకు ... మీ నోటిని అదుపులో పెట్టుకొని మీరు ఏ... https://t.co/2olb2xZomv

Figure 31: Web interface of the tweet classifier

This interface is build using the following technologies

- HTML5/CSS3/JS
- Golang
- Postgres

3.2.1.1 Front End

The front-end of the application is built using HTML5, CSS3 and JS. We used the CSS framework Bootstrap[32] for developing the user interface. Using the Bootstrap framework allowed us to spend less time tweaking the UI constructs of the application.

The user interface of the application is designed to be simple to avoid clutter and focus on the classification task. As a result, when the annotator visits the page, the text to be classified is presented in the center of the page with buttons to classify the text. When the annotator makes a selection by clicking on the class of the text, a POST call is made through Ajax to the Golang web server. The server processes the request and updates the class of the tweet in the Postgres database. If the operation is successful, the web server returns a HTTP 200 OK response. The response body also includes the next tweet to be classified which the front end Javascript function displays.

If the annotator wants to review the previous tweet, they can do so by simply clicking the *prev* button. The annotator can then either update their selection or click on *next* which takes them to the next tweet to be classified.

The POST request is listed here

```
POST /classify?id=13674 HTTP/1.1
Host: localhost:8881
cache—control: no—cache
Content—Type: multipart/form—data;
Content—Disposition: form—data; name=”id”
13674
Content—Disposition: form—data; name=”class”
T
```

The response received from the server is of the form

```
{
  ”id”: 13675,
  ”tweet”: ”RT @Amitkamble77777: By just showing the MOB, does the TRUTH will ↩
      change?\nThis deceptive company https://t.co/OoP2E2FNmy was using ↩
      …PYRAMID”,
  ”class”: ””
}
```

3.2.1.2 Back End

The web server for this application is built in Golang, an open source compiled programming language with built in concurrency primitives that make it an ideal choice for systems programming. The web server apart from rendering the static HTML, CSS and JS files also handles the requests received from the front end through the REST API endpoints. When a request is received the web server processes the data it received by connecting to the Postgres database. The port the web server is running can be configured by passing in flags during the deployment of the server. The server supports all the endpoints listed in Table 3.2.1.2.

Table 32: REST Endpoints

Endpoint	**Description**
/	Index page
/current	Returns the tweet to be classified along with it's id
/next	Returns the next tweet to be classified based on the id provided
/prev	Returns the previous tweet that was classified based on the id provided
/classify	Classifies the tweet provided the id and class

3.2.1.3 Database

The collected code mixed data was save to a Postgres database. Postgres is an open source relational database. The data modeling for our dataset is trivial as there are no dependencies on other datasets, so we chose to go with a single table to store all our data. The table is created using the following SQL command.

```sql
CREATE TABLE masters.tweetclassification(
  id  SERIAL,
  tweet TEXT,
  class CHAR(32),
  PRIMARY KEY(id)
);
```

Each REST API endpoint of the web server corresponds to a statement on the Postgres database. The following query returns the tweet with the highest id that is not classified.

```
SELECT id, tweet from masters.tweetclassification where id = (select max(id) ↩
    from masters.tweetclassification where class is not null limit 1)
```

We can get the previous tweet using

```
SELECT id, tweet from masters.tweetclassification where id = $1-1;
```

The next tweet is identified as

```
SELECT id, tweet from masters.tweetclassification where id = $1+1;
```

Once the class is identified it can be updated using the following query

```
UPDATE masters.tweetclassification set class=$1 where id=$2;
```

By using this application we were able to label 18,755 tweets. The distribution of the classified data is in Table 3.2.1.3.

Table 33: Class Distribution

Class	**Number of Samples**
label_J	8466
label_E	4210
label_CM	2736
label_T	2547
label_R	796

The data labeled as Junk cannot be used as it does not have any language specific information to classify it against. So, even though 18,755 tweets were classified, only

10,289 can be used for our purposes.

3.2.2 NER Tagging

The second data labeling task is name entity recognition (NER) tagging. NER is the process of identifying named entities in a given text.

3.2.2.1 Doccano

Doccano is an open source tool for text annotation. It is built using Python, HTML, CSS, JS. It can be run either locally or on a remote server. Doccano can either be installed locally on the machine or can be run using Docker. The following command will run an instance of Doccano locally.

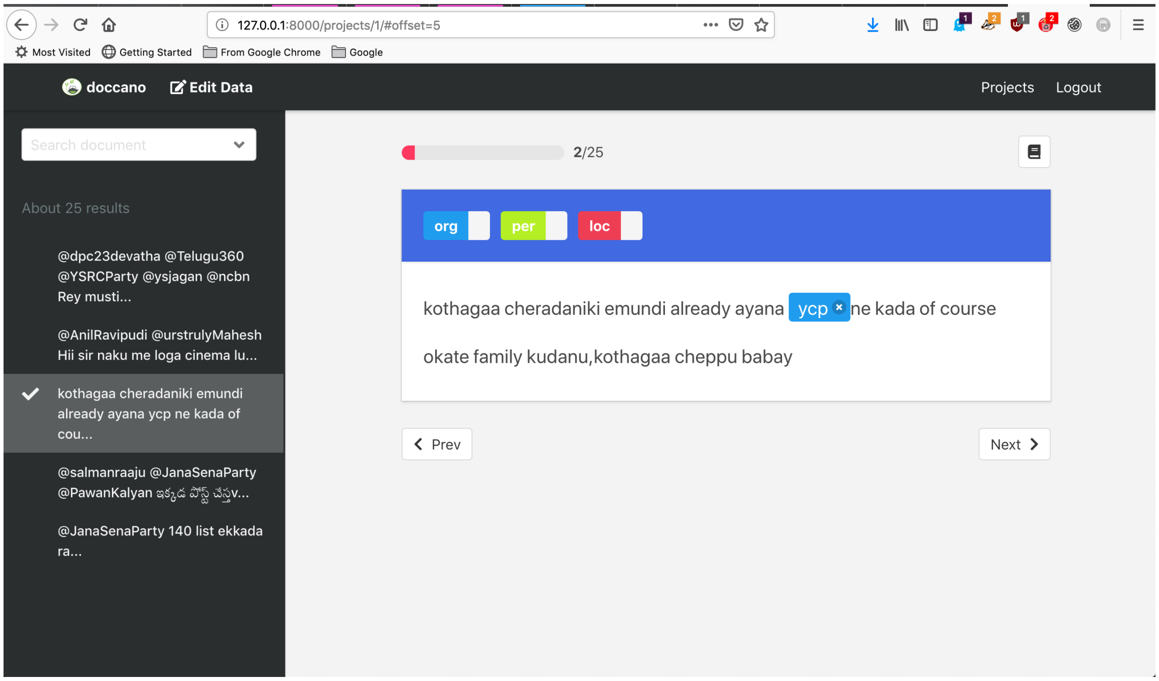
```
docker run -d --name doccano -p 8000:80 chakkiworks/doccano
```

Doccano provides several options to annotate data. In order to annotate the data, we need to upload the data to Doccano in this JSONL format.

```
{"text":"@SriVijayaNagesh @VSReddy_MP @PawanKalyan Avunu maaji mp garu ,↩
   badyaleni milanti drama mp lu vuntey alagey …rankelu https://t.co/↩
   HRNXciDzOH"}
{"text":"RT @Trending_Hypers: Koduraamaana Raaja @itisthatis' #↩
   SuperDeluxeOnMar29. Here's a sarcastic #DingDong promo video- https://t.co↩
   /4…rBas6ve"}
{"text":"@raps909 @KTRTRS Purthi Peru? Ni photo massage pettu."}
{"text":"@madhutata111 @urstruly_HNE @urstrulyMahesh Oh ah ammayi meerena..! ↩
   Live lo aayana raagane evaro ganthulu …vestunaru https://t.co/kFipaq4I09"}
{"text":"@actorbrahmaji @krsna_vamsi @urstrulyMahesh Left double chekka.. ↩
   Right ok ok chekka.. Meesam kavali dora"}
{"text":"@dpc23devatha @Telugu360 @YSRCParty @ysjagan @ncbn Rey musti LK, ↩
   family gurunchi vallaku telusu... dagulbagi …panulu https://t.co/↩
   R1XgUJFDun"}
```

```
{"text":"@AnilRavipudi @urstrulyMahesh Hii sir naku me loga cinema lu ↩
    chaiyalani na dream all read on the way lo cinema …stor https://t.co/↩
    ky2WkFfmz5"}
{"text":"kothagaa cheradaniki emundi already ayana ycp ne kada of course okate↩
     family kudanu,kothagaa cheppu babay"}
```

The Doccano interface for annotating can be seen in 32



@dpc23devatha @Telugu360 @YSRCParty @ysjagan @ncbn Rey musti...

@AnilRavipudi @urstrulyMahesh Hii sir naku me loga cinema lu...

✓ kothagaa cheradaniki emundi already ayana ycp ne kada of cou...

@salmanraaju @JanaSenaParty @PawanKalyan ఇక్కడ పోస్ట్ చేస్తv...

@JanaSenaParty 140 list ekkada ra...

kothagaa cheradaniki emundi already ayana ycp ne kada of course okate family kudanu,kothagaa cheppu babay

Figure 32: Doccano interface for NER Tagging

Once the data is tagged, it can be exported as a JSON file. The exported format is shown below

```
{"id": 1, "text": "@SriVijayaNagesh @VSReddy_MP @PawanKalyan Avunu maaji mp ↩
    garu ,badyaleni milanti drama mp lu vuntey alagey …rankelu https://t.co/↩
    HRNXciDzOH", "annotations": [], "meta": {}}
{"id": 2, "text": "RT @Trending_Hypers: Koduraamaana Raaja @itisthatis' #↩
    SuperDeluxeOnMar29. Here's a sarcastic #DingDong promo video— https://t.co↩
    /4…rBas6ve", "annotations": [{"label": 2, "start_offset": 21, "end_offset↩
    ": 39, "user": 1}], "meta": {}}
{"id": 3, "text": "@raps909 @KTRTRS Purthi Peru? Ni photo massage pettu.", "↩
    annotations": [], "meta": {}}
```

```
{"id": 4, "text": "@madhutata111 @urstruly_HNE @urstrulyMahesh Oh ah ammayi meerena..! Live lo aayana raagane evaro ganthulu …vestunaru https://t.co/kFipaq4I09", "annotations": [], "meta": {}}
{"id": 5, "text": "@actorbrahmaji @krsna_vamsi @urstrulyMahesh Left double chekka.. Right ok ok chekka.. Meesam kavali dora", "annotations": [], "meta": {}}
{"id": 6, "text": "@dpc23devatha @Telugu360 @YSRCParty @ysjagan @ncbn Rey musti LK, family gurunchi vallaku telusu... dagulbagi …panulu https://t.co/R1XgUJFDun", "annotations": [], "meta": {}}
{"id": 7, "text": "@AnilRavipudi @urstrulyMahesh Hii sir naku me loga cinema lu chaiyalani na dream all read on the way lo cinema …stor https://t.co/ky2WkFfmz5", "annotations": [], "meta": {}}
{"id": 8, "text": "kothagaa cheradaniki emundi already ayana ycp ne kada of course okate family kudanu,kothagaa cheppu babay", "annotations": [{"label": 1, "start_offset": 42, "end_offset": 46, "user": 1}], "meta": {}}
```

3.3 Data Normalization

In order to train the data for both the classification and the Name entity recognition tasks, we need to normalize the data. This is necessary as the performance of the machine learning models cannot be impacted by the user names, hash tags and emojis. Since these elements are neutral and are language independent they can be standardized in the dataset.

We replace any references of twitter handles with *[USERNAME]*, hash tags with *[HASHTAG]*, emojis with *[e]* and links with *[LINK]*. Table 3.3 shows an example of the original text and the corresponding normalized text. These elements are detected and replace using regular expression. The normalized elements are stored in the Postgres database.

This dataset in both its original and normalized form is made available in CSV format on Github.

Table 34: Original Text and Normalized Text

Original Text	**Normalized text**
Watch the LIVE in All SocialMedia 😃😃 Unveiling the wax statue in few minutes ! @urstrulyMahesh Facebook:... https://t.co/HsgtuIUJYb	watch the live in all socialmedia [e][e]unveiling the wax statue in few minutes ! [username] facebook:... [link]

Chapter 4

Text Classification

4.1 Why Classification?

In this chapter we look at the ways to identify whether the given text is code-mixed, monolingual or Romanized. Classifying the text allows us to perform further operations on the data. This classification problem is different from language identification as code-mixed data is interspersed with multiple languages only word level language identification is possible. For our classification problem we intend to identify the text at a sentence level into code-mixed, monolingual or Romanized.

4.2 Bag of Words

Bag of words is a simple representation of the words in the data set disregarding the grammar and the order of the words while retaining the multiplicity [33]. This method is used widely for text classification tasks. In order to compile the bag of words, we perform tokenization and vectorization of the data.

Tokenization is the task of separating the words of a sentence into individual words called tokens. For example, tokenizing the text in Table 4.2 will give us the following token set.

```
{”best”,”wishes”,”for”,”your”,”birthday”,”i”,”wish”,”you”,”a”,”happy”,”↩
    birthday”,”i”,”wish”,”you”,”the”,”best”,”.”,”may”,”your”,”birthday”,”wish↩
    ”,”come”}
```

Table 41: Sample text

Index	**Text**
S1	I wish you a happy birthday
S2	Best wishes for your birthday
S3	I wish you the best. May your birthday wish come

Vectorization is the process of representing text as vector. The vector representation of the text in Table 4.2 can be represented as document matrix as shown in Table4.2

Table 42: Text Vectorization

Index	best	wishes	for	your	birthday	i	wish	you	a	happy	the	may	come
S1	0	0	0	0	1	1	1	1	1	1	0	0	0
S2	1	1	1	1	1	0	0	0	0	0	0	0	0
S3	1	1	0	1	1	1	2	1	0	0	1	1	1

Six algorithms have been chosen to measure the performance on the dataset to establish a baseline of how the classification tasks are performed. The details of the algorithms are presented in the following sections.

4.3 Naive Bayes

Naive Bayes classifiers are linear classifies that are simple and efficient [34]. Naive Bayes classifiers are based on the Bayes' theorem and the *naive* refers to the assumption that the features in a dataset are independent. Even though this assumption is false in many cases, the models hold up well when working in real world scenarios. Another reason to choose Naive Bayes classifiers is that they work well with small data sets, which is what we are working with.

When it comes to text classification, a document $d \in D$ corresponds to a data instance, where D denotes the training dataset. The document d can be represented as a bag of words. Each word $w \in d$ comes from a set W of all feature words. Each document d is associated with a class label $c \in C$ where C denotes the class label set. The Naive Bayes classifiers estimate the conditional probability $P(c|d)$ which represents the probability that a documents d belongs to class c. Using Bayes rule, we have

$$P(c|d) \propto P(c)P(d|c)$$

The weighted results of training multinomial naive Bayes classifier on our dataset are shown in Table 4.3. The label wise score are shown in Table 4.3

Table 43: Weighted scores for multinomial naive Bayes classification

Accuracy	82.84%
Precision	75.8%
Recall	82.84%
F1-Score	79.06%

Table 44: Class wise recall, precision and F1 scores for Multinomial Naive Bayes Classifier

Class	**Precision**	**Recall**	**F1-Score**
English	85%	99.13%	91.58%
Code Mixed	70.96%	69.77%	70.36%
Telugu	90.36%	96.72%	93.43%
Romanized	68.36%	21.72%	32.37%

Figure 41 shows the confusion matrix for the Naive Bayes classifier.

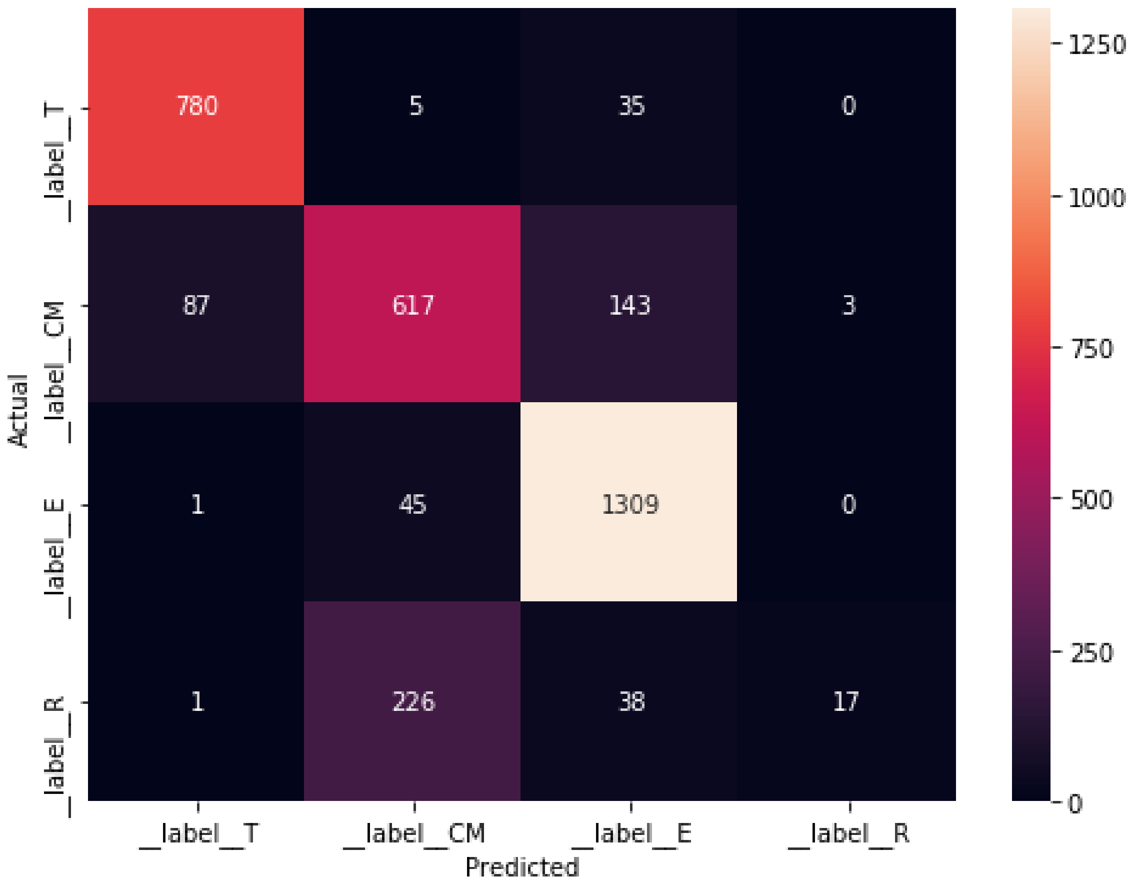


Figure 41: Confusion matrix for the naive Bayes classifier

4.4 SVM

A support vector machine (SVM) is a supervised machine learning algorithm that works well on classification problems [35]. SVMs work well with smaller dataset[36]. In this algorithm, each data item is plotted as a point in n-dimensional space (where n is number of features) with the value of each feature being the value of a particular coordinate. The classification is performed by finding the hyper-plane that differentiates the different classes.

SVMs are effective in high dimensional space as they perform feature extraction inherently. This is useful as we are not pruning any features. They are also memory efficient as they use only a few data points (support vectors) in the decision function. For, each word there are only a few non-zero entries. SVMs work well with dense concepts and sparse instances.

The weighted results of training the SVM classifier on our dataset are shown in Table 4.4. The label wise score are shown in Table 4.4

Table 45: Weighted scores for support vector machine classification

Accuracy	85.24%
Precision	85.03%
Recall	85.23%
F1-Score	83.91%

Table 46: Class wise recall, precision and F1 scores for Support Vector Machine Classifier

Class	**Precision**	**Recall**	**F1-Score**
English	91.94%	94.69%	93.3%
Code Mixed	70.59%	79.54%	74.8%
Telugu	91.84%	95.9%	93.83%
Romanized	75.8%	23.12%	35.47%

Figure 42 shows the confusion matrix for the SVM classifier.

4.5 Random Forest Classifier

Random Forest is a supervised machine learning algorithm that can be use for classification or regression tasks [37]. Random forest is comprised of multiple decision trees, the larger the number of trees the robust the classifier is. Random forest classifiers work well with sparse data[38].

Random forests work by creating decision trees on randomly selected data samples, getting the prediction from each tree and selects the best solution by means of voting. They are a good indicator of the feature importance. Random forest is an ensemble method where the collection of decision trees generated by randomly splitting the dataset are known as the forest. Each tree depends on an independent random sample. As each

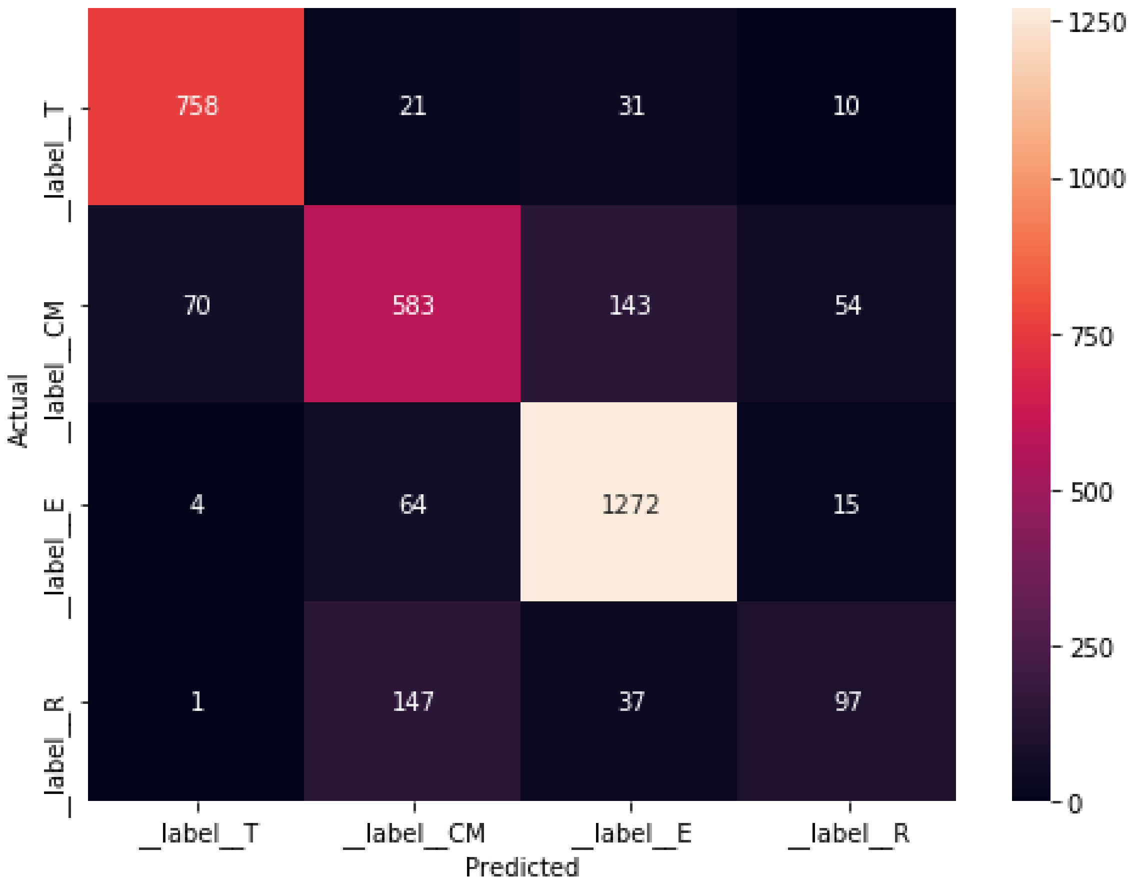


Figure 42: Confusion matrix for the SVM classifier

tree votes, the the most popular class is chosen as the final result of the random forest classifier.

Even though Random forest classifiers work well with Sparse data, missing values and tend not to over fit, their performance on classifying code mixed data is poor. The weighted results of training a Random Forest classifier on our dataset are shown in Table 4.5. The label wise score are shown in Table 4.5

Table 47: Weighted scores for random forest classifier

Accuracy	44.08%
Precision	63%
Recall	46%
F1-Score	33%

Table 48: Class wise recall, precision and F1 scores for Random Forest Classifier

Class	Precision	Recall	F1-Score
English	43%	100%	60%
Code Mixed	88%	3%	6%
Telugu	94%	15%	27%
Romanized	0%	0%	0%

Figure 43 shows the confusion matrix for the Random Forest classifier.

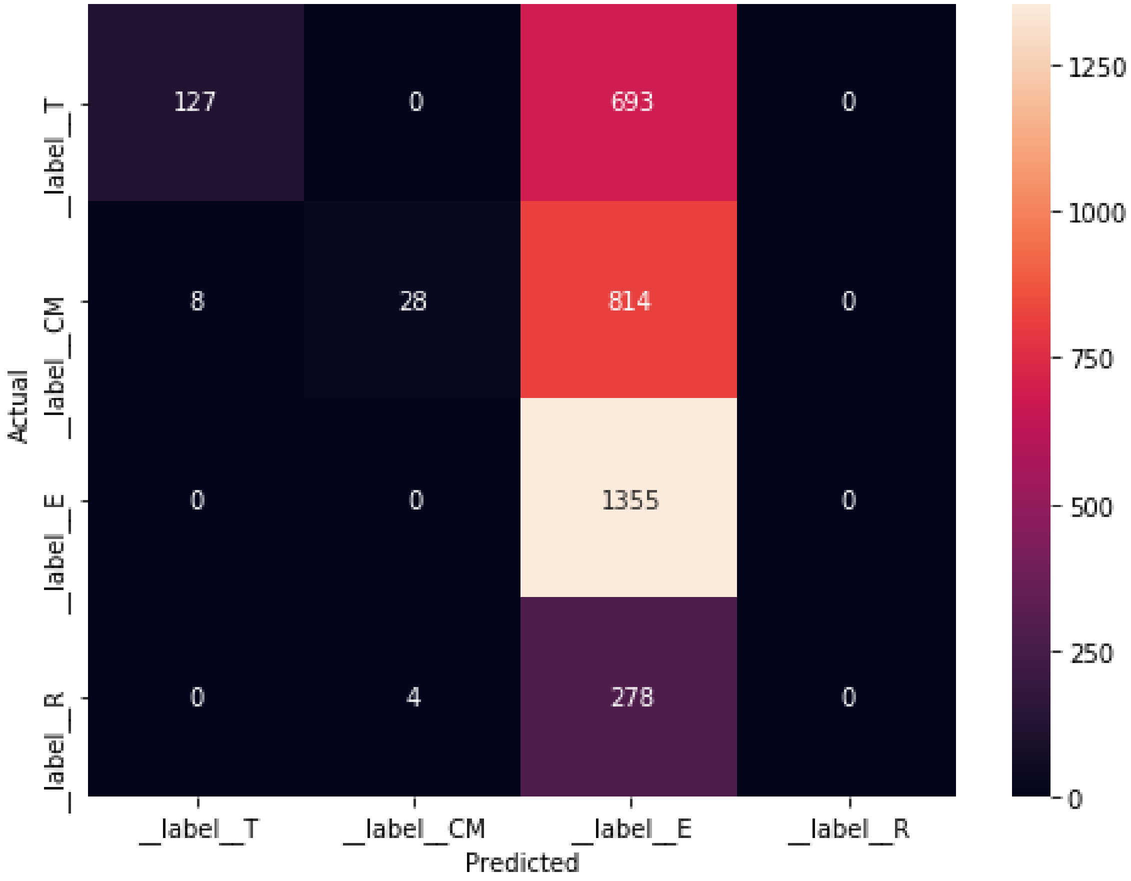


Figure 43: Confusion matrix for the random forest classifier

4.6 Logistic Regression

Logistic regression is a statistical method that is used for supervised machine learning applications [39]. Logistic regression can be used as a linear regression equation to produce discrete binary outputs.Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

The performance of the logistic regression classifier for the application on code mixed data is comparable to the Naive Bayes classifier and SVM classifier. The weighted results of training a logistic regression classifier on our dataset are shown in Table 4.6. The label wise score are shown in Table 4.6

Table 49: Weighted scores for logistic regression classifier

Accuracy	81.66%
Precision	80%
Recall	81%
F1-Score	81%

Table 410: Class wise recall, precision and F1 scores for Logistic Regression Classifier

Class	**Precision**	**Recall**	**F1-Score**
English	82%	97%	89%
Code Mixed	72%	69%	70%
Telugu	90%	92%	91%
Romanized	62%	15%	25%

Figure 44 shows the confusion matrix for the random forest classifier.

4.7 fastText

fastText is an open source library for efficient learning of word representations and sentence classification [40]. In fastText each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram; words being represented as the sum of these representations. Most models that learn such representations ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large vocabularies and many rare

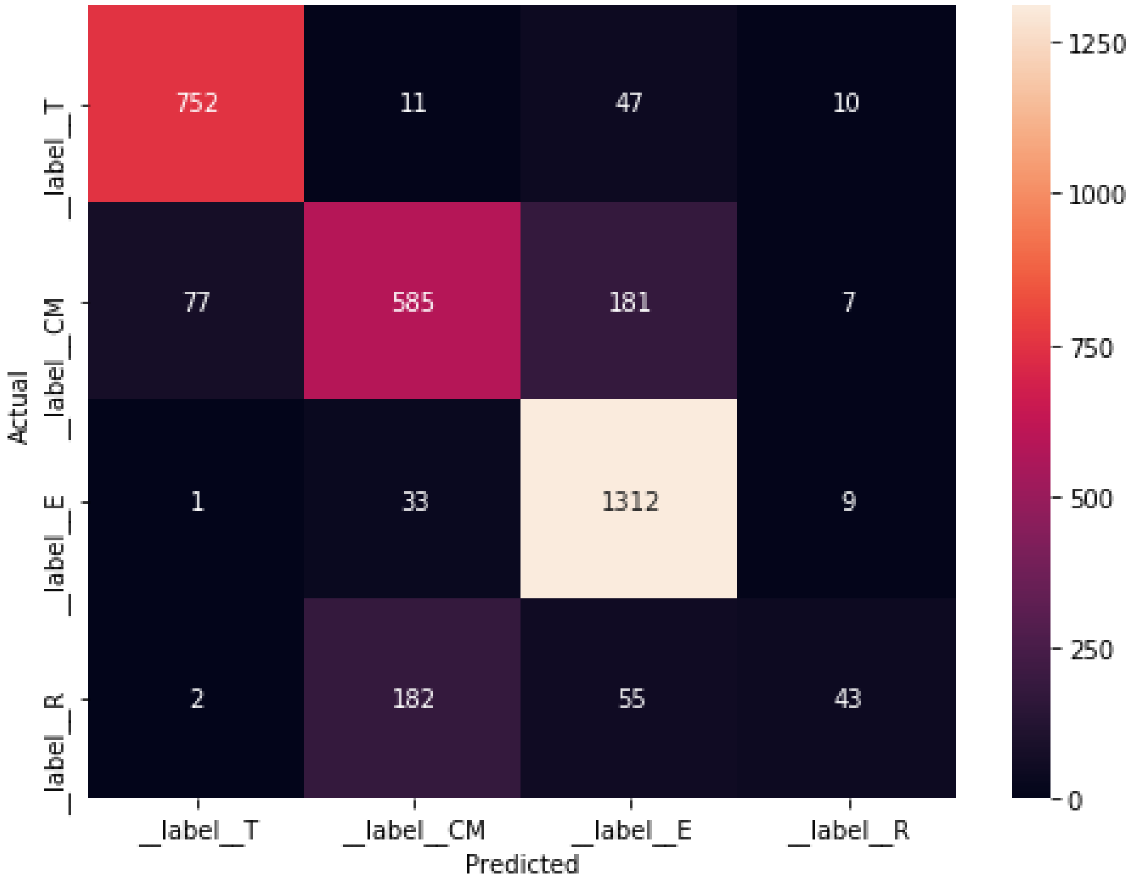


Figure 44: Confusion matrix for the logistic regression classifier

words. Especially for code mixed languages the vocabularies are quite large and fastText is a good fit in this scenario. fastText uses a new approach based on the skipgram model. It is fast, allowing to train models on large corpora quickly and allows to compute word representations for words that did not appear in the training data.

One of the key features of fastText word representation is its ability to produce vectors for any words, even made-up ones. Indeed, fastText word vectors are built from vectors of substrings of characters contained in it. This allows to build vectors even for misspelled words or concatenation of words. We train our model for 25 epochs and with a learning rate of 1.0.

The weighted results of training multinomial naive Bayes classifier on our dataset are shown in Table 4.7. The label wise score are shown in Table 4.7

Table 411: Weighted scores for fastText classifier

Accuracy	78.6%
Precision	79.6%
Recall	79.6%
F1-Score	71.25%

Table 412: Class wise recall, precision and F1 scores for fastText Classifier

Class	**Precision**	**Recall**	**F1-Score**
English	83.82%	93.69%	88.48%
Code Mixed	77.11%	65.7%	70.94%
Telugu	86.65%	88.12%	87.38%
Romanized	39.63%	42.1%	40.83%

4.8 LSTM

Long Short Term Memory (LSTM) is a recurrent artificial neural network architecture.

The first layer is the embedded layer that uses length 100 vectors to represent each word. SpatialDropout1D performs variational dropout in NLP models.The next layer is the LSTM layer with 100 memory units. The output layer must create 4 output values, one for each class. Activation function is softmax for multi-class classification. Because it is a multi-class classification problem, categorical_crossentropy is used as the loss function.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_17 (Embedding)     (None, 250, 100)          5000000
_________________________________________________________________
spatial_dropout1d_9 (Spatial (None, 250, 100)          0
_________________________________________________________________
lstm_13 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_14 (Dense)             (None, 4)                 404
=================================================================
Total params: 5,080,804
```

```
Trainable params: 5,080,804

Non—trainable params: 0

________________________________________________________________
```

The weighted results of training LSTM classifier on our dataset are shown in Table 4.8. The label wise score are shown in Table 4.8

Table 413: Weighted scores for LSTM classifier

Accuracy	81.85%
Precision	86%
Recall	79%
F1-Score	78%

Table 414: Class wise recall, precision and F1 scores for LSTM Classifier

Class	**Precision**	**Recall**	**F1-Score**
English	100%	1%	3%
Code Mixed	88%	96%	92%
Telugu	76%	65%	70%
Romanized	90%	91%	90%

Figure 45 shows the confusion matrix for the LSTM classifier.

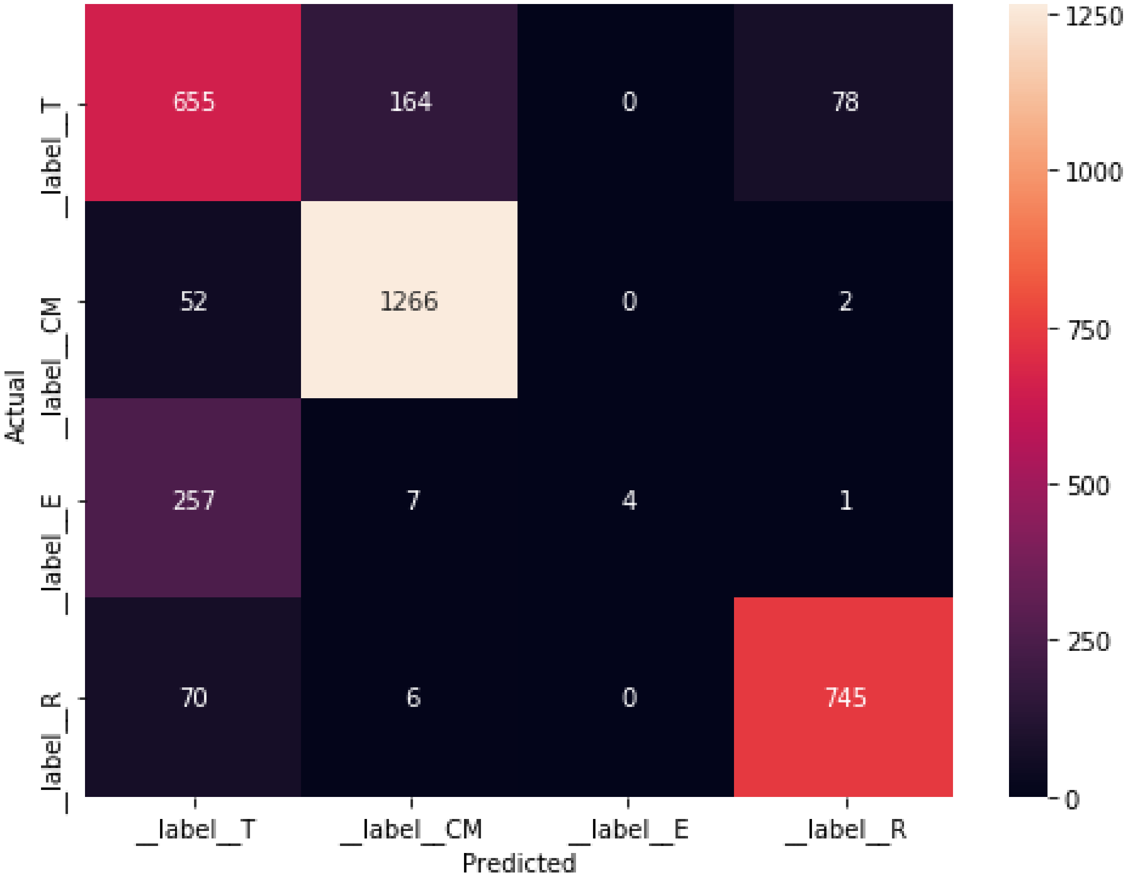


Figure 45: Confusion matrix for the LSTM classifier

Chapter 5

Named Entity Recognition

Named Entity Recognition is the process of information extraction from unstructured text to extract named entities into predetermined categories such as organization names, locations, dates, person names, etc., This is useful in various applications like text classification, indexing, automatic question answering and many other natural language applications. For this project, we intend to extract only the entities Person, Organization and Location.

5.1 Approaches to Named Entity Recognition

Named entity recognition in English has received a lot of attention in the past two decades. The popular approaches to name entity recognition use the following approaches

Rule based: Ruled based approaches use hand crafted rules on plain text or build parsers on part of speech tagged text to recognize predetermined patterns. For example, a sequence of capitalized words ending in “Inc.” is considered as a name of an organization. These rules do not necessarily translate to other domains and are hard to extend or apply.
Gazetteers: Another approach to Named Entity Recognition is to use a stored list of

entities of different types to compare against and identify the entities. This is a simple method that is fast and easy to extend by adding entities to the list. The limitations of this approach are that the lists cannot be exhaustive as the entities are numerous and language is constantly evolving.

Statistical Models: Statistical models treat name entity recognition as a sequence tagging problem, where each word is tagged with entity type. A typical approach to this is to use parts of speech taggers on the text and use statistical models to identify patterns to identify named entities. However, the problem with statistical models is to generate large scale, high quality training data.

5.2 Challenges with Code-Mixed Languages

When it comes to name entity recognition of code mixed English Telugu text, there are several challenges. Telugu is an agglutinative language. An agglutinative language is one where complex words are formed by stringing together morphemes without changing their spelling or phonetics. As a result when performing part of speech tagging, Telugu tends to generate more tags for the same words. An example can be seen below.

కిరణ్[NNP] బంతిని[NN][DT] కొట్‌టాడు[VBD][PRP]

The above text translates and is part of speech tagged as follows

Kiran[NNP] hit[VBD] the[DT] ball[NN]

As you can notice Telugu language generates more tags for the same text. Name entity recognition becomes more challenging on code mixed English-Telugu text as part of speech tagging is considered the first step of the process. In recent years, there has been some research into building a POS tagger for code-mixed Telugu language but there is no extensive dataset or model to use definitively.

5.3 Performance Comparison

For code-mixed languages, we can take the approach of using a NER model built using a corpus of code-mixed languages. The challenge with this is, such a corpus is tough to build, as a lot of the text generated is primarily in one language. In order to build a code-mixed language corpus, we will have to rely on short social media blurbs. Apart from being limited, building such a dataset is time consuming and intensive.

Another approach, the one we implemented, is to use a named entity recognition model built on the corpus of a single language. Different named entity recognition algorithms use different corpora to build out their NER models. Because of the abundance of English language named entity recognition models, we can use a model built using English language as an alternative to code-mixed NER model. This accuracy of this approach is low but provides a start towards name entity recognition of code-mixed text.

From the code-mixed data we have collected in Chapter 2, we identified 500 named entities from 1847 samples of code-mixed data. The distribution of the named entities is listed in Table 5.3

Table 51: Distribution of named entity types

Person	57%
Location	25.6%
Organization	17.4%

We attempted to improve the performance of the name entity recognition by translating the code-mixed text into a single language and using an NER model built on the corpus of that language. In order to do this, we use the Google Translate API to convert our text into English at the sentence level and then extracting the entities. The raw text is normalized and cleaned before being translated to English language to improve the accuracy of the translation. An example of this is listed in Table 5.3

Table 52: Data processing for NER

Raw Text	RT @JanasenaParty: రంజాన్ శుభాకాంక్షలు - Janasena chief @PawanKalyan https://t.co/MFXsMDeWfc
Normalized Text	rt [username]: రంజాన్ శుభాకాంక్షలు - Janasena chief [username] [link]
Cleaned Text	రంజాన్ శుభాకాంక్షలు - Janasena chief
Translated Text	Ramadan greetings - Janasena chief

5.3.1 NLTK

Natural Language Toolkit (NLTK) [41] as the name implies is a library for performing natural language processing written in Python. It provides an easy to use API for performing text processing operations like classification, tokenization, stemming, name entity recognition etc., NLTK also provides ability to extend it's functionality to different languages other than English. We evaluated the performance of the NLTK name entity recognition on raw code-mixed text and translated English text. The results can be observed in Table 5.3.1. As can be observed, the entity extraction is marginally better on translated text than code-mixed text for all three entities we expect to recognize.

Table 53: Accuracy of NLTK

	Code-mixed Text	**Translated English Text**
Person	20.8 %	23 %
Location	12 %	14.3%
Organization	7.2 %	10.42 %

5.3.2 MITIE

MIT Information extraction(MITIE) [42] is a C++ library that can be used for information extraction. MITIE provides API bindings in multiple languages like Python, Java, C, C++ and Matlab. MITIE can be used to perform text classification, tokenization and entity recognition. MITIE also provides with the ability to build custom entity recognition models on a base model built using a language corpus. MITIE operates using distributional word embeddings and support vector machines. The performance of the MITIE name entity recognition can be observed in Table 5.3.2. MITIE performs the best among the three models that we evaluated.

Table 54: Accuracy of MITIE

	Code-mixed Text	**Translated English Text**
Person	27.68 %	42.12 %
Location	18.45 %	22.79%
Organization	14.72 %	25.1 %

5.3.3 SpaCy

SpaCy is a CPython language library that is built on a neural network framework to provide faster performance on natural language processing tasks while maintaining a high level of accuracy[43]. SpaCy can be used to perform text classification, tokenization, lemmatization, POS tagging and many other operations. It comes pre-built with language models for several languages such as English, German, French, Spanish, Portuguese, Italian, Dutch and Greek. It is also possible to extend the support to other languages in SpaCy. The performance of SpaCy over different entities is presented in Table 5.3.3.

Table 55: Accuracy of Spacy

	Code-mixed Text	**Translated English Text**
Person	21 %	29.37 %
Location	11.41 %	15.6%
Organization	11.8 %	15.4 %

5.4 Conclusion

The previous section shows the performance of all three name entity recognizers on code mixed English-Telugu text and translate English text. The results shows that MITIE performs better compared to the other two approaches but the overall accuracy is still low for all three libraries.

On code-mixed text the low accuracy can be attributed to the fact that we are attempting to extract entities on text whose vocabulary is outside the language corpus that the NER models have been trained on. The low accuracy on the translated text can be attributed to several factors such as the inaccuracy and loss of information in translation, unfamiliarity with Indian names commonly encountered in the dataset.

Chapter 6

Real-time processing of Code-mixed English-Telugu tweets

In this chapter, we look into the web application built to visualize real time processing of code-mixed English-Telugu tweets. The application is built using a microservice architecture with a REST API to communicate between the different modules.

6.1 Architecture

This application is built using the microservice architecture where an application is comprised of loosely coupled services. In a microservice architecture, multiple small applications each performing a single functional task come together to form the overall application. These individual applications are loosely coupled over light weight transfer protocols. This allows for greater flexibility in developing, deploying and maintaining an application. The failure of a single module does not bring down the entire application.

A microservice architecture also allows for greater flexibility in choosing the right tools for the use case preventing in lock-in of technologies. This is observed in the architecture presented where each technology is chosen for specific reasons. Figure 61 shows the architecture of the system.

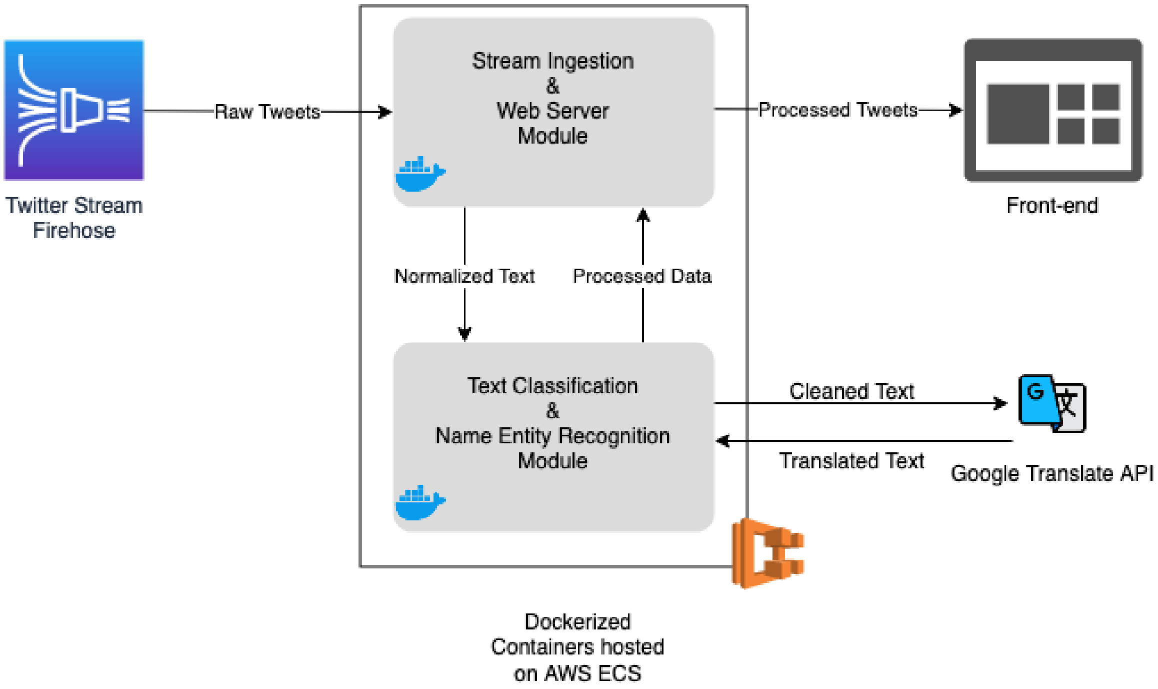


Figure 61: Architecture diagram

6.2 Real-Time Data Ingestion

In order to process the data in real-time, we need to ingest the data in real-time. Twitter provides a streaming API endpoint from where data can be ingested. Subscribing to this endpoint by setting filters on certain twitter handles that tweet English, Telugu and code-mixed English-Telugu tweets allows us to limit the volume of data we need to ingest and helps in processing only the data of interest.

In order to handle the large volumes of the data received from the streaming API, we need a language that can process a high volume of data concurrently. In order for this, we use Golang.

6.2.1 Go

Go is an open source programming language created at Google in 2009. Go is a statically compiled language with excellent support for concurrency primitives. Apart from its focus on simplicity and readability, Go's focus on performance makes it a great choice for processing large volumes of data. Go is also used extensively for systems development. Popular applications like Kubernetes, Docker and InfluxDB are built using Go.

We use Go to build our stream ingestion and web server module. Each tweet received through the stream ingestion module spins off a Goroutine that makes a call to the machine learning module which returns the processed data. The processed data is then formatted and broadcast to all the front-end clients.

6.3 Real-Time Processing

The processing for the real-time data is done using a Python application. The reason to use Python for this microservice is because Python has good support for machine learning libraries and all our machine learning models have been developed in Python as noted in chapters 3 and 4.

The Python microservice runs a Flask web server on port 7070. The ingestor module makes a HTTP POST request to the Python module, which runs the normalized text through different classifiers and Name Entity Recognition models to extract the entities. That data is compiled together and returned.

The request is of the following form

```
POST /predict HTTP/1.1
Host: localhost:7070
Content-Type: application/json
cache-control: no-cache
Postman-Token: 3d885ed9-ab2a-4abc-b8a0-386ba0ccd00c
{
    ”normalized_text”:”[username] [username] [username] [username] *janasena ↩
        election campaigning special video    at 10.30 pm.*.. ..”
}------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

The response from the processing module is of the form listed below

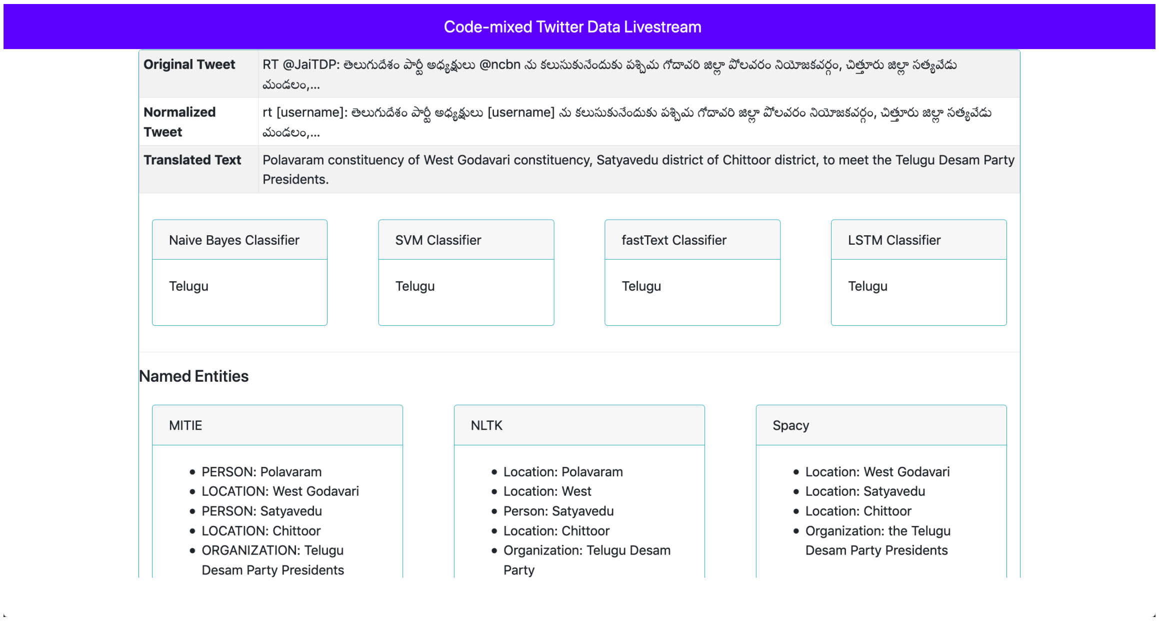
```
{
  ”classifiers”: {
    ”ft_prediction”: ”Code−mixed”,
    ”lstm_prediction”: ”Code−mixed”,
    ”nb_prediction”: ”English”,
    ”svm_prediction”: ”Telugu”
  },
  ”ner”: {
    ”mitie”: {
      ”person”: ”pawan”,
      ”org”: ”tdp”,
      ”loc”: ”none”
    },
    ”stanford_ner”: {
      ”person”: ”pawan”,
      ”org”: ”tdp”,
      ”loc”: ”none”
    },
    ”spacy”: {
      ”person”: ”pawan”,
      ”org”: ”tdp”,
      ”loc”: ”none”
    }
  }
}
```

Even though we've tested six different classifiers in Chapter 3, we only use four classifiers. The Random Forest Classifier is not used owing to it's poor performance on this data, logistic regression classifier is also not presented to keep the design simple and it's performance is similar to the LSTM.

6.4 Real-time Visualization

The visualization of this data allows us to juxtapose the performance of different classifier and NER models in real time. The front-end visualization is built using HTML5, CSS3 and JS. We also use Jquery, a javascript library to build the front-end.

Figure 62 shows the interface of the real-time streaming page.



Code-mixed Twitter Data Livestream

Original Tweet	RT @JaiTDP: తెలుగుదేశం పార్టీ అధ్యక్షులు @ncbn ను కలుసుకునేందుకు పశ్చిమ గోదావరి జిల్లా పోలవరం నియోజకవర్గం, చిత్తూరు జిల్లా సత్యవేడు మండలం,...
Normalized Tweet	rt [username]: తెలుగుదేశం పార్టీ అధ్యక్షులు [username] ను కలుసుకునేందుకు పశ్చిమ గోదావరి జిల్లా పోలవరం నియోజకవర్గం, చిత్తూరు జిల్లా సత్యవేడు మండలం,...
Translated Text	Polavaram constituency of West Godavari constituency, Satyavedu district of Chittoor district, to meet the Telugu Desam Party Presidents.

Naive Bayes Classifier	SVM Classifier	fastText Classifier	LSTM Classifier
Telugu	Telugu	Telugu	Telugu

Named Entities

MITIE

- PERSON: Polavaram
- LOCATION: West Godavari
- PERSON: Satyavedu
- LOCATION: Chittoor
- ORGANIZATION: Telugu Desam Party Presidents

NLTK

- Location: Polavaram
- Location: West
- Person: Satyavedu
- Location: Chittoor
- Organization: Telugu Desam Party

Spacy

- Location: West Godavari
- Location: Satyavedu
- Location: Chittoor
- Organization: the Telugu Desam Party Presidents

Figure 62: Real-time streaming page

The interface displays real-time visualization of the streaming data. This has been achieved using Web sockets for the data streaming. There are three ways in which to update the front-end.

6.4.1 Polling

Polling is the process of making periodic requests to the web server for any new data. The server responds with the data immediately if it is available or does not return any data if there is none available. The frequency of making this request can be set depending on how frequently the data changes.

For a real-time update, the polling frequency needs to be sub-second. This means the client will be making multiple requests to the server in a short time frame. If there are multiple clients accessing the application, the server can be overwhelmed by the load of these requests. This will impact the server and waste network resources especially when there is no new data.

6.4.2 Long Polling

Long polling is similar to polling a request. The client sends the request to the server, but instead of the server responding back immediately when there is no data, the server responds only when it has data to transfer. This is achieved by increasing the timeout for the request.

Long polling alleviates some of the problems caused by polling. By eliminating empty responses, long polling reduces some of the bandwidth challenges caused through polling. The disadvantage of this method is for high volume data, each the long polling acts similar to polling as each request is immediately fulfilled. This causes an overhead where a new connection has to be established between the client and the server for each request.

6.4.3 Web socket

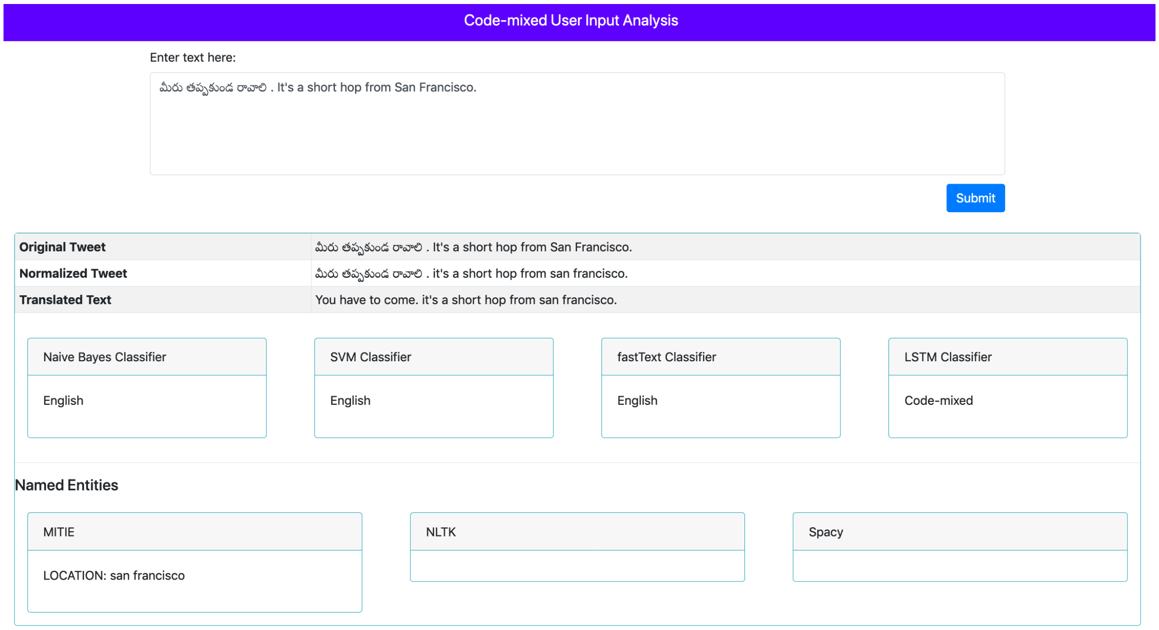
Web sockets unlike polling and long-polling do not close a connection immediately once the data is transferred. Web sockets keep a connection live between the client and server and use the same connection to transfer the data.

Using web sockets we can transfer a large amount of data without additional connection requests. Each client has to open a web socket connection the first time it connects to the server. This allows the server to handle more connections than through polling or long polling. Since new connections do not have to be established for every request, the latency of the data is faster as well. The server or the client can close the web socket connections if there is no activity for a specified duration freeing up resources.

6.5 User input processing

The application apart from real-time processing of code-mixed text, also supports the processing of ad-hoc user inputs. This page contains a text area, where the user can submit their text and the results of the analysis are presented. When a user submits a request, an Ajax Post call is made to the back end server which processes the request and returns a JSON response which is rendered on the page.

Figure 63 shows the user input processing page with the result of the user input displayed.



Code-mixed User Input Analysis

Enter text here:

మీరు తప్పకుండ రావాలి . It's a short hop from San Francisco.

Original Tweet	మీరు తప్పకుండ రావాలి . It's a short hop from San Francisco.
Normalized Tweet	మీరు తప్పకుండ రావాలి . it's a short hop from san francisco.
Translated Text	You have to come. it's a short hop from san francisco.

Naive Bayes Classifier	SVM Classifier	fastText Classifier	LSTM Classifier
English	English	English	Code-mixed

Named Entities

MITIE	NLTK	Spacy
LOCATION: san francisco		

Figure 63: Ad-hoc analysis page

The request sent to the back end can be seen below

```
POST /predict HTTP/1.1
```

```
Host: localhost:8090
Content-Type: application/json
User-Agent: PostmanRuntime/7.13.0
Accept: */*
Cache-Control: no-cache
Host: localhost:8090
cache-control: no-cache
{
  "text":"it's a short hop from san francisco.   "
}
```

The response received from the back end is of the form listed below

```
{
    "tweet": {
        "original_text": "it's a short hop from san francisco.    ",
        "normalized_text": "it's a short hop from san francisco.    "
    },
    "predictions": {
        "ft_prediction": "English",
        "lstm_prediction": "English",
        "nb_prediction": "English",
        "svm_prediction": "English",
        "translated_text": "it's a short hop from san francisco.    ",
        "non_english_percent": 0.9375,
        "spacy_entities": [],
        "nltk_entities": []
    },
    "entities": {
        "mitie": [
            "LOCATION: san francisco "
        ],
        "nltk": [],
        "prose": null,
        "spacy": []
    }
}
```

6.6 Deployment

Deploying an application built using the microservice architecture can often times involve multiple steps, since each microservice needs to be compiled along with it's dependencies and the executable deployed on the server. The advent of cloud and containerization services have made this process easier.

6.6.1 Docker

Docker is a containerization platform that performs operating system level virtualization. This allows for any application to be package into a docker container and run on any platform where the docker engine is installed. Since the application dependencies are also package as part of the docker container, dependency management becomes easier.

The same dockerized application can run on any underlying operating system allowing greater flexibility in deploying the application. Docker also allows versioning containers which makes rolling back corrupt deployments easier. Docker also provides security by isolating our application from other programs running on the server. We can strictly control the access to our dockerized application preventing any data leakages.

Our application comprises of two docker containers one for the real-time ingestion and web server, the other for the Python Flask web server to perform machine learning operations. These dockerized containers can then be deployed on stand-alone servers or on the cloud using a container orchestration service like AWS Elastic Container Service.

Chapter 7

Conclusion and Future Work

7.1 Summary and Conculsion

Code mixing in a language is the embedding of lingustic units of one language in utterance or text into another language. This phenomenon is often observed in bilingual and multi-lingual communities. This project focuses on two parts of code-mixed data from social media text.

The first part focuses on data collection and classification of the text, we introduced the concept of code-mixing and the lack of research on code-mixed English-Telugu text. We then detailed how the data was collected from the Twitter streaming API and described the characteristics of collected data. We also document how the data was annotated and the applications that were built and used to annotate the data fro both the classification and name entity purposes. We then present the performance of different text classification algorithms in identifying code-mixed, romanized, English or Telugu text. The SVM approach to text classification provides the highest accuracy of 85.24%.

The second part of the project focuses on extracting named entities and the real time processing of social media text. We detail the process of name entity recognition, the different approaches in practice and list the challenges of name entity recognition in code-mixed data. We list our approach to extract the entities from code-mixed data and compare the performance of three popular name entity recognition libraries and the reason behind their poor accuracy rates are explained. We finally present the web application built to process streaming social media text from the Twitter streaming API and detail the architecture, implementation and the deployment process.

7.2 Future work

The dataset and the results presented can be used as a benchmark for code-mixed English-Telugu text processing. The same approach to classification can also be used for other code-mixed English language pairs. This work can also be extended to build a POS tagged corpus that can be used to build name entity models. We can also evaluate the performance of word level translation of code-mixed text to measure the performance of entity extraction. Another approach that can be evaluated as part of future work is to build a corpus of annotated code-mixed data and use that as the basis for the NER model.

Bibliography

[1] P. Muysken, C. P. Díaz, P. C. Muysken, *et al.*, *Bilingual speech: A typology of code-mixing*, vol. 11. Cambridge University Press, 2000.

[2] E. Annamalai, "The language factor in code mixing," 1989.

[3] C. Chandramouli and R. General, "Census of india 2011," *Provisional Population Totals. New Delhi: Government of India*, 2011.

[4] D. P. Pattanayak, *Multilingualism in India*. Multilingual Matters, 1990.

[5] U. Barman, A. Das, J. Wagner, and J. Foster, "Code mixing: A challenge for language identification in the language of social media," in *Proceedings of the first workshop on computational approaches to code switching*, pp. 13–23, 2014.

[6] Y. Vyas, S. Gella, J. Sharma, K. Bali, and M. Choudhury, "Pos tagging of english-hindi code-mixed social media content," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 974–979, 2014.

[7] H. Nakayama, "chakki-works/doccano," Jun 2019.

[8] N. M. Kamwangamalu, "Code-mixing and modernization," *World Englishes*, vol. 8, no. 3, pp. 321–332, 1989.

[9] S. N. Sridhar and K. K. Sridhar, "The syntax and psycholinguistics of bilingual code mixing.," *Canadian Journal of Psychology/Revue canadienne de psychologie*, vol. 34, no. 4, p. 407, 1980.

[10] E. G. Bokamba, "Are there syntactic constraints on code-mixing?," *World Englishes*, vol. 8, no. 3, pp. 277–292, 1989.

[11] L. Milroy and P. Muysken, *One speaker, two languages: Cross-disciplinary perspectives on code-switching*. Cambridge University Press, 1995.

[12] P. C. Muysken, "Code-switching and grammatical theory," 1995.

[13] J. Gafaranga and M.-C. Torras, "Interactional otherness: Towards a redefinition of codeswitching," *International Journal of Bilingualism*, vol. 6, no. 1, pp. 1–22, 2002.

[14] B. Alex, *Automatic detection of English inclusions in mixed-lingual data with an application to parsing*. PhD thesis, University of Edinburgh, 2008.

[15] T. Solorio and Y. Liu, "Learning to predict code-switching points," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 973–981, Association for Computational Linguistics, 2008.

[16] T. Solorio and Y. Liu, "Part-of-speech tagging for english-spanish code-switched text," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1051–1060, Association for Computational Linguistics, 2008.

[17] H. Elfardy and M. Diab, "Token level identification of linguistic code switching," *Proceedings of COLING 2012: Posters*, pp. 287–296, 2012.

[18] F. AlGhamdi, G. Molina, M. Diab, T. Solorio, A. Hawwari, V. Soto, and J. Hirschberg, "Part of speech tagging for code switched data," in *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pp. 98–107, 2016.

[19] A. Das and B. Gambäck, "Identifying languages at the word level in code-mixed indian social media text," in *Proceedings of the 11th International Conference on Natural Language Processing*, pp. 378–387, 2014.

[20] B. Sasidhar, P. Yohan, A. V. Babu, and A. Govardhan, "A survey on named entity recognition in indian languages with particular reference to telugu," *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 2, p. 438, 2011.

[21] H. Isozaki and H. Kazawa, "Efficient support vector classifiers for named entity recognition," in *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pp. 1–7, Association for Computational Linguistics, 2002.

[22] T. Zhang and D. Johnson, "A robust risk minimization based named entity recognition system," in *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pp. 204–207, Association for Computational Linguistics, 2003.

[23] G. Petasis, F. Vichot, F. Wolinski, G. Paliouras, V. Karkaletsis, and C. D. Spyropoulos, "Using machine learning to maintain rule-based named-entity recognition and classification systems," in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pp. 426–433, Association for Computational Linguistics, 2001.

[24] A. Mikheev, M. Moens, and C. Grover, "Named entity recognition without gazetteers," in *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pp. 1–8, Association for Computational Linguistics, 1999.

[25] P. Srikanth and K. N. Murthy, "Named entity recognition for telugu," in *Proceedings of the IJCNLP-08 Workshop on Named Entity Recognition for South and South East Asian Languages*, 2008.

[26] P. M. Shishtla, K. Gali, P. Pingali, and V. Varma, "Experiments in telugu ner: A conditional random field approach," in *Proceedings of the IJCNLP-08 Workshop on Named Entity Recognition for South and South East Asian Languages*, 2008.

[27] P. R. Rao and S. L. Devi, "Cmee-il: Code mix entity extraction in indian languages from social media text@ fire 2016-an overview.," in *FIRE (Working Notes)*, pp. 289–295, 2016.

[28] I. A. Bhat, M. Shrivastava, and R. A. Bhat, "Code mixed entity extraction in indian languages using neural networks.," in *FIRE (Working Notes)*, pp. 296–297, 2016.

[29] D. Gupta, S. Tripathi, A. Ekbal, and P. Bhattacharyya, "A hybrid approach for entity extraction in code-mixed social media data," *MONEY*, vol. 25, p. 66, 2016.

[30] K. Vijayanand and R. Seenivasan, "Named entity recognition and transliteration for telugu language," *Parsing in Indian Languages*, p. 64, 2011.

[31] “Docs - twitter developers.”

[32] M. Otto and J. Thornton, “Bootstrap.”

[33] H. M. Wallach, “Topic modeling: beyond bag-of-words,” in *Proceedings of the 23rd international conference on Machine learning*, pp. 977–984, ACM, 2006.

[34] A. McCallum, K. Nigam, *et al.*, “A comparison of event models for naive bayes text classification,” in *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48, Citeseer, 1998.

[35] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” in *European conference on machine learning*, pp. 137–142, Springer, 1998.

[36] M. Chi, R. Feng, and L. Bruzzone, “Classification of hyperspectral remote-sensing data with primal svm for small-sized training dataset problem,” *Advances in space research*, vol. 41, no. 11, pp. 1793–1799, 2008.

[37] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, “Text classification algorithms: A survey,” *Information*, vol. 10, no. 4, p. 150, 2019.

[38] J. W. Richards, D. L. Starr, N. R. Butler, J. S. Bloom, J. M. Brewer, A. Crellin-Quick, J. Higgins, R. Kennedy, and M. Rischard, “On machine-learned classification of variable stars with sparse and noisy time-series data,” *The Astrophysical Journal*, vol. 733, no. 1, p. 10, 2011.

[39] A. Genkin, D. D. Lewis, and D. Madigan, “Large-scale bayesian logistic regression for text categorization,” *Technometrics*, vol. 49, no. 3, pp. 291–304, 2007.

[40] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification,” *arXiv preprint arXiv:1607.01759*, 2016.

[41] E. Loper and S. Bird, “Nltk: the natural language toolkit,” *arXiv preprint cs/0205028*, 2002.

[42] D. E. King, “Dlib-ml: A machine learning toolkit,” *Journal of Machine Learning*

Research, vol. 10, no. Jul, pp. 1755–1758, 2009.

[43] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer, "Allennlp: A deep semantic natural language processing platform," *arXiv preprint arXiv:1803.07640*, 2018.