**Applying BERT Model on Hinglish Dataset:**

The process of applying the BERT model to a Hinglish (a combination of Hindi and English) sentiment analysis dataset. Sentiment analysis involves determining the sentiment or emotional tone of a given text, which can be positive, negative, or neutral.

# **Data Preparation**

# **Data Loading:** We begin by unzipping the dataset using **!unzip 'data.zip'** and installing the necessary libraries, including **ktrain** for BERT model training.

|  |
| --- |
| !unzip 'data.zip' !pip install ktrain |

1. **Importing Libraries:** We import various libraries for data preprocessing, evaluation, and visualization.

|  |
| --- |
| from sklearn.metrics import f1\_score, confusion\_matrix, classification\_report, precision\_score, recall\_score, accuracy\_score import numpy as np from matplotlib import pyplot as plt import re import string from nltk.corpus import stopwords from nltk.stem import PorterStemmer from nltk.tokenize import TweetTokenizer from nltk import download download('stopwords') |

1. **Data Cleaning:** We define a **Tweet** class and functions for cleaning the tweet content. This cleaning process involves removing mentions, special characters, URLs, digits, and punctuation, as well as tokenization and stemming for English word.

|  |
| --- |
| class Tweet:  def \_\_init\_\_(self):     self.uid = None     self.content = ''     self.sentiment = ''   def cleanTweet(tweet): # print(tweet.content,"\n") # tweet.content = re.sub(r'@ [0-9a-zA-Z]+', '', tweet.content) # remove @ mentions  tweet.content = re.sub(r'\\_', '', tweet.content) # remove underscores  tweet.content = re.sub(r'...', '', tweet.content) # remove elipses/dots  tweet.content = re.sub(r'\.', '', tweet.content) # remove elipses/dots  tweet.content = re.sub(r'^RT[\s]+', '', tweet.content) # remove RT  tweet.content = re.sub("[#@(c)àâ€¦¥°¤ð¹ÿœ3/4¨‡†§<²¿¸^]", '', tweet.content) # remove weird symbols  tweet.content = tweet.content.split("http")[0].split('https')[0] # remove http/https  tweet.content = ''.join([i for i in tweet.content if not i.isdigit()]) # remove digits  tweet.content = ''.join([word for word in tweet.content if word not in string.punctuation]) # remove punctuations  tweet.content = TweetTokenizer(preserve\_case=False, strip\_handles=True,reduce\_len=True).tokenize(tweet.content)  tweet.content = ' '.join([i for i in tweet.content]) # convert to string # print(tweet.content) #     print("============================================================================")  return tweet |

1. **Loading Stopwords:** We load stopwords for both English and Hinglish languages from separate files.

|  |
| --- |
| def load\_stop\_words():  stopwords\_english = stopwords.words('english')  stopwords\_hinglish = []  with open('data/hinglish\_stopwords.txt','r') as fp:     while True:         line = fp.readline()         if not line:             break             stopwords\_hinglish.append(line.strip())  return stopwords\_english, stopwords\_hinglish |

1. **Reading the Dataset:** We read the Hinglish dataset, which contains tweets in both English and Hinglish, and perform data preprocessing.

|  |
| --- |
| def readFile(filename, test\_data=False):  stemmer\_english = PorterStemmer()  stopwords\_english, stopwords\_hinglish = load\_stop\_words()  all\_tweets = []  with open(filename, 'r', encoding="utf8") as fp:     tweet = Tweet()     last\_one = False     while True:         line = fp.readline()         if not line:             last\_one = True         if len(line.split()) > 1 or last\_one==True:             if last\_one==True or line.split()[0] == 'meta':                 if len(tweet.content) > 0 or last\_one==True:                         all\_tweets.append(cleanTweet(tweet))                     if last\_one==True:                         break                     tweet = Tweet()                 tweet.uid = line.split()[1]                 tweet.sentiment = line.split()[2] if test\_data==False else None             else:                 if line.split()[1] == "Eng":                     if line.split()[0] not in stopwords\_english: #                         line.split()[0] = autoCorrect(line.split()[0])                         tweet.content += stemmer\_english.stem(line.split()[0]) + " "                 elif line.split()[1] == "Hin":                     if line.split()[0] not in stopwords\_hinglish:                         tweet.content += line.split()[0] + " "                 else:                     tweet.content += line.split()[0] + " "     return all\_tweets |

**Data Analysis and Visualization**

Before training the BERT model, it's essential to analyze and visualize the dataset's characteristics. You can explore the distribution of sentiment labels, the length of tweets, and other relevant statistics.

**BERT Model Training**

1. **Importing BERT Model:** We import the BERT model for Hinglish sentiment analysis using the **ktrain** library.

|  |
| --- |
| import os os.environ["CUDA\_DEVICE\_ORDER"]="PCI\_BUS\_ID"; os.environ["CUDA\_VISIBLE\_DEVICES"]="0"; import ktrain from ktrain import text |

1. **Preprocessing Data:** We preprocess the training data by converting text inputs to numerical format suitable for BERT.

|  |
| --- |
| x\_train = [i.content for i in all\_tweets] y\_train = [i.sentiment for i in all\_tweets] t = text.Transformer('vicgalle/xlm-roberta-large-xnli-anli') trn = t.preprocess\_train(x\_train, y\_train) |

1. **Model Building:** We create the BERT classifier model.

|  |
| --- |
| model = t.get\_classifier() |

1. **Training the Model:** We use a one-cycle learning rate schedule to train the model.

|  |
| --- |
| learner = ktrain.get\_learner(model, train\_data=trn, batch\_size=6) learner.fit\_onecycle(2e-7, 1) |

**Model Evaluation**

After training the model, it's crucial to evaluate its performance on a test dataset. We use metrics such as F1-score, precision, recall, accuracy, and confusion matrix to assess the model's performance.

|  |
| --- |
| # Loading actual labels for the test dataset actual\_labels\_dict = dict() with open(r'data/test/test\_labels\_hinglish.txt','r') as fp:  # ...   # Reading the test dataset and assigning actual labels all\_test\_tweets = readFile(r'data/test/Hindi\_test\_unalbelled\_conll\_updated.txt', test\_data=True) for i in all\_test\_tweets:  i.sentiment = actual\_labels\_dict[i.uid]   # Making predictions using the trained model predictor = ktrain.get\_predictor(learner.model, preproc=t) predictions = [] for i in all\_test\_tweets:     predictions.append(predictor.predict(i.content))   # Converting sentiment labels to numerical values for evaluation actual\_num = [] for i in all\_test\_tweets:  # ...   predictions\_num = [] for i in predictions:  # ...   # Evaluating the model show\_results(actual\_num, predictions\_num) |

**Result Analysis**

After evaluation, you can analyze and visualize the results, including misclassified examples, to gain insights into the model's performance.

|  |
| --- |
| for i in range(len(actual\_num)):  if (actual\_num[i]!=predictions\_num[i]):         print("Actual:",all\_test\_tweets[i].sentiment)         print("Predicted:",predictions[i][0][0][9:])     print(all\_test\_tweets[i].content)         print("=====================================================================") |

**Conclusion**

In this document, we have outlined the steps to apply the BERT model to a Hinglish sentiment analysis dataset. The process involves data preparation, BERT model training, evaluation, and result analysis. Fine-tuning the model and optimizing hyperparameters can further improve its performance on Hinglish sentiment analysis tasks.