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

1. **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
```

2. **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')
```

3. **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
```

4. **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
```

5. **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
```

2. **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)
```

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

```
model = t.get_classifier()
```

4. **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.