Read Me File

Overview

The cyber challenges project aims to compare the experiences of individuals with IDD and Down Syndrome on social media by extracting insights from twitter-based datasets: IDD dataset and Down_syndrome dataset. The datasets were uploaded on Kaggle, where we performed the analysis. The analysis was performed in a Python environment using the BERT model for Negative/Positive/Neutral sentiment predictions, and LDA model for topic modelling.

Initial Setup: Importing Libraries

The process of importing is demonstrated in the code snippet shown below:

```
import re
import nltk
import string
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from gensim import corpora, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, confusion_matrix
from gensim.models import LdaModel
from gensim.corpora import Dictionary
from gensim.models import word2vec
from tabulate import tabulate
from sklearn.decomposition import LatentDirichletAllocation
from keras.preprocessing.text import Tokenizer
import tensorflow as tf
```

The code snippet above was used to set up a sentiment analysis and natural language processing pipeline. In the cell, necessary modules and libraries are imported to perform various NLP tasks and data visualization. Various tools, including nltk, and SentimentIntensityAnalyzer were used for sentiment analysis, genism library was imported for word embedding and topic modelling

tasks, and it imported components like LDA Model and Dictionary, and the LatentDirichletAllocation was imported to perform LDA topic modelling. Seaborn and matplotlib.pyplot were imported for data visualization purposes, TensorFlow (tf) and Keras were imported for deep learning tasks, and Tokenizer was imported for preprocessing and data cleaning tasks.

Loading data

```
data = pd.read_csv("/kaggle/input/disability-data/ID dataset0.CSV", encoding="ISO
-8859-1")
data_2 = pd.read_csv("/kaggle/input/disability-data/Down Syndrome Dataset0.csv",
encoding="ISO-8859-1")
data.head()
data_2.head()
```

The code snippet is loading the data (CSV files) into the pandas DataFrame and then displaying the first few rows of the data using the '*.head ()*' approach. The first two lines read the CSV files located at a specified file path. The files contain data associated with IDD and Down Syndrome tweets. The pd.read_CSV function reads CSV files and generates a DataFrame based on the data in the CSV file. The lines '*data.head ()'* and *'data_2.head ()*' display the top 5 rows of the dataset, which helps to quickly inspect the contents and structure of our datasets.

Data Cleaning and Preprocessing

```
def clean_text(text):
    '''
        Make text lowercase, remove text in square brackets,remove links,remove p
unctuation
        and remove words containing numbers.
    '''
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

The code in the cell above defines the '*clean_text'* function, which is used for text data cleaning and preprocessing. The function performs a series of text preprocessing steps using the *'re.sub ()*' command to remove unwanted aspects, including HTML tags, URLs, words with numbers, texts in brackets, and punctuations. It also utilizes the '*text.lower ()*' command to change all the text in the dataset into lowercase.

```
# Defining a function to remove emoji's
def removeEmoji(text):
    emoji = re.compile(
        "["
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F600-\U0001F64F"  # emotion icons
        u"\U0001F1E0-\U0001F1FF"  # flags
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        "]+", flags=re.UNICODE
    )
    return emoji.sub(r'', text)
```

The above cell defines '*removeEmoji ()*', which helps remove emojis in the tweets. The function performs a series of steps to remove symbols & pictographs, emoticon icons, flags, and transport & map symbols. The *u ""* command was used to create a Unicode string and the re.UNICODE flag was passed and the data was converted into Unicode. The *return emoji.sub(r'', text)* snippet removed emojis from the text by substituting them with empty string.

LDA Model and Topic Modelling

```
#Apply preprocessing to the two datasets
data = data.dropna(subset=['tweet'])
data['clean_text'] = data['tweet'].apply(clean_text)
data_2 = data_2.dropna(subset=['tweet'])
data_2['clean_text'] = data_2['tweet'].apply(clean_text)
```

The snippet helps preprocess the two datasets by handling the missing values in the data. The *data = data.dropna (subset=['tweet'])* and *data_2 = data.dropna (subset=['tweet'])* were used to remove missing values in the 'tweet' column from the DataFrame. If any row in 'tweet' column has missing value, the entire row is dropped. This step is performed to clean the data prior to analysis since missing values can interfere with calculations and algorithms.

```
#create dictionary for the datasets
documents_data = [text.split() for text in data['clean_text']]

data_dictionary = corpora.Dictionary(documents_data)

documents_data_2 = [text.split() for text in data['clean_text']]

data_2_dictionary = corpora.Dictionary(documents_data_2)

# Convert the dataset into a  bag-of-words (BoW) representation
data_corpus = [data_dictionary.doc2bow(doc) for doc in documents_data]
data_2_corpus = [data_2_dictionary.doc2bow(doc) for doc in documents_data_2]
```

The above snippet creates a dictionary for the two datasets. The code is preparing the data for topic modelling based on the Gensim library. *Documents_data* and *documents_data_2* are created as a list comprehension using the '.split ()' metho, which splits the text into a list of words in the 'clean_text' column. The lines *data_dictionary = corpora.Dictionary(documents_data)* and *data_2_dictionary = corpora.Dictionary(documents_data_2)* are create the Gensim dictionary from tokenized documents. The words are mapped to unique IDs and the generated dictionary is used in later stages of topic modelling. The datasets are then converted into a bag-of-words representation, where the corpus contains the word Id and frequency in the dataset.

```
# Train the LDA models for each dataset
num_topics = 20

data_lda_model = models.LdaModel(data_corpus, num_topics=num_topics, id2word=data
_dictionary, passes=10)
```

```
data_2_lda_model = models.LdaModel(data_2_corpus, num_topics=num_topics, id2word=
data_2_dictionary, passes=10)
```

The snippet above trains the LDA for each dataset and specifies that the model should create 20 topics in each dataset. It utilises the corpus of tokenized documents ('data_corpus'), a dictionary ('data_dictionary'), and the specified of number of topics ('num_topics') as input. The created model identifies the suitable topics in the datasets and their related words.

```
# Print the topics and keywords for Williams syndrome and Intellectual disability
print("ID dataset Topics:")
for idx, topic in data_lda_model.print_topics(-1):
    print(f'Topic {idx + 1}: {topic}')
```

The code is printing the 20 most frequently discussed topics and their respective keywords in the IDD dataset.

```
print("Down Syndrome Dataset Topics:")
for idx, topic in data_2_lda_model.print_topics(-1):
    print(f'Topic {idx + 1}: {topic}')
```

The code is printing the 20 most frequently discussed topics and their respective keywords in the down_syndrome dataset.

```
# dominant topics for in the datasets
print("ID Dataset Dominant Topics:")
for i, row in enumerate(data_lda_model[data_corpus]):
    row = sorted(row, key=lambda x: (x[1]), reverse=True)
    print(f"Document {i + 1}: Topic {row[0][0] + 1} (Probability: {row[0][1]})")

print("Down Syndrome Dataset Dominant Topics:")
for i, row in enumerate(data_2_lda_model[data_2_corpus]):
    row = sorted(row, key=lambda x: (x[1]), reverse=True)
    print(f"Document {i + 1}: Topic {row[0][0] + 1} (Probability: {row[0][1]})")
```

The code identifies the topics with highest probability in the documents and outputs information regarding the dominant topics in the IDD and down_syndrome datasets. It prints the topic ID and probability for each documents in the given datasets. The most prevalent topics based on the LDA model will be printed out.

```
# data['Text'] = data['Text'].astype(str)
data['tweet'] = data['tweet'].apply(lambda x: clean_text(x))


data['tweet']=data['tweet'].apply(lambda x: removeEmoji(x))
data['tweet'].apply(lambda x:len(str(x).split())).max()

data['tweet'].head()
```

The cell involves text preprocessing and analysis of the 'data' DataFrame. The `.apply(lambda x: clean_text(x))` function applies the '*clean_text'* function the *'tweet'* column to clean the text data. The `.apply(lambda x:len(str(x).split())).max()` computes the number of words in each cleaned text, splits the cleaned data using whitespaces as delimiter, and counts them to find the maximum value.

```
data_2['tweet'] = data_2['tweet'].apply(lambda x: clean_text(x))

data_2['tweet']=data_2['tweet'].apply(lambda x: removeEmoji(x))
data_2['tweet'].apply(lambda x:len(str(x).split())).max()
data_2['tweet'][20:80]
```

The cell involves text preprocessing and analysis of the 'data_2' DataFrame. The `.apply(lambda x: clean_text(x))` function applies the '*clean_text'* function the *'tweet'* column to clean the text data. The `.apply(lambda x:len(str(x).split())).max()` computes the number of words in each cleaned text, splits the cleaned data using whitespaces as delimiter, and counts them to find the maximum value.

```
word_cloud = WordCloud(
     background_color='white',
     stopwords=set(STOPWORDS),
     max_words=50,
     max_font_size=40,
     scale=5,
     random_state=1
 ).generate(str(data['tweet']))

fig = plt.figure(1, figsize=(10,10))
plt.axis('off')
fig.suptitle('Word Cloud for top 50 prevelant words in the tweets of Intellectual
Disability', fontsize=20)
fig.subplots_adjust(top=2.3)
plt.imshow(word_cloud)
plt.show()
```

The code snippet creates and displays a word cloud visualization based on the text in the '*data*' DataFrame (IDD dataset). Several parameters were used to customize the word cloud appearance: *background_color* sets the background color, *stopwords* provides words to be ignored in the word cloud, *max_words* specifies the number of words to be included in the word cloud, and the *.generate ()* method creates the word cloud based on the input text data. It uses matplotlib library to create and display the plot, and configure it to have a title.

```
word_cloud = WordCloud(
     background_color='white',
     stopwords=set(STOPWORDS),
     max_words=50,
     max_font_size=40,
     scale=5,
     random_state=1
 ).generate(str(data_2['tweet']))

fig = plt.figure(1, figsize=(10,10))
plt.axis('off')
fig.suptitle('Word Cloud for top 50 prevelant words in the tweets of Down Syndrom
e', fontsize=20)
fig.subplots_adjust(top=2.3)
plt.imshow(word_cloud)
plt.show()
```

The code snippet creates and displays a word cloud visualization based on the text in the '*data_2'* DataFrame (IDD dataset). Several parameters were used to customize the word cloud appearance: *background_color* sets the background color, *stopwords* provides words to be ignored in the word cloud, *max_words* specifies the number of words to be included in the word cloud, and the *.generate ()* method creates the word cloud based on the input text data. It uses matplotlib library to create and display the plot, and configure it to have a title.

Sentiment Labelling

```
nltk.download("vader_lexicon")

sentiments = SentimentIntensityAnalyzer()

data["Positive"] = [sentiments.polarity_scores(i)["pos"] for i in data["tweet"]]
data["Negative"] = [sentiments.polarity_scores(i)["neg"] for i in data["tweet"]]
data["Neutral"] = [sentiments.polarity_scores(i)["neu"] for i in data["tweet"]]
data['Compound'] = [sentiments.polarity_scores(i)["compound"] for i in data["twee
t"]]
```

The cell above computes the sentiment scores for the text data in the '*tweet'* column of the '*data'* DataFrame using the VADER sentiment model. The code computes the positive, negative, neutral, and compound sentiment scores for each tweet and the obtained scores are added as new column to the DataFrame.

```
# Labeling sentiments into the dataset

score = data["Compound"].values
sentiment = []
for i in score:
    if i >= 0.05 :
        sentiment.append('Positive')
    elif i <= -0.05 :
        sentiment.append('Negative')
    else:
        sentiment.append('Neutral')
data["Sentiment"] = sentiment
```

```
data.head()

sns.countplot(data=data, x='Sentiment')
```

The code processes the sentiment scores computed earlier, classifies them as ‘positive’, ‘negative’, or ‘neutral’ and creates a count plot to visualize how sentiments are distributed in the cleaned data.

```
sentiment_counts = data["Sentiment"].value_counts()
sentiment_percentages = (sentiment_counts / len(data)) * 100

# Plotting the pie chart
plt.figure(figsize=(8, 6))
plt.pie(sentiment_percentages, labels=sentiment_percentages.index, autopct='%1.1f
%%')
plt.title("Sentiment Distribution for ID profiles")
plt.axis('equal')
plt.show()
```

The cell computes the percentages of positive, negative and neutral sentiments in the cleaned datasets, and creates a pie chart to visualize the percentages of the sentiments.

```
data_2["Positive"] = [sentiments.polarity_scores(i)["pos"] for i in data_2["tweet
"]]
data_2["Negative"] = [sentiments.polarity_scores(i)["neg"] for i in data_2["tweet
"]]
data_2["Neutral"] = [sentiments.polarity_scores(i)["neu"] for i in data_2["tweet"
]]
data_2['Compound'] = [sentiments.polarity_scores(i)["compound"] for i in data_2["
tweet"]]
```

The cell above computes the sentiment scores for the text data in the ‘*tweet’* column of the ‘*data_2’* DataFrame using the VADER sentiment tool. The code computes the positive, negative, neutral, and compound sentiment scores for each tweet and the obtained scores are added as new column to the DataFrame.

```
# Labeling sentiments into the dataset

score = data_2["Compound"].values
sentiment = []
for i in score:
    if i >= 0.05 :
        sentiment.append('Positive')
    elif i <= -0.05 :
        sentiment.append('Negative')
    else:
        sentiment.append('Neutral')
data_2["Sentiment"] = sentiment
data_2.head()

sns.countplot(data=data_2, x='Sentiment')
```

The code processes the sentiment scores computed earlier, classifies them as 'positive', 'negative', or 'neutral' and creates a count plot to visualize how sentiments are distributed in the cleaned data.

```
# Calculate sentiment percentages
sentiment_counts = data_2["Sentiment"].value_counts()
sentiment_percentages = (sentiment_counts / len(data_2)) * 100

# Plotting the pie chart
plt.figure(figsize=(8, 6))
plt.pie(sentiment_percentages, labels=sentiment_percentages.index, autopct='%1.1f
%%')
plt.title("Sentiment Distribution for down syndrome Profiles")
plt.axis('equal')
plt.show()
```

The cell computes the percentages of positive, negative and neutral sentiments in the cleaned datasets, and creates a pie chart to visualize the percentages of the sentiments.

Model Training

```
lb= LabelEncoder()

lb.fit(data['Sentiment'])
```

```
y_train_down_syndrome = lb.transform(data['Sentiment'].to_list())
y_train_idd = lb.transform(data_2['Sentiment'].to_list())

# Splits Dataset into Training and Testing set
x_train_down_syndrome, x_test_down_syndrome, y_train_down_syndrome, y_test_down_s
yndrome = train_test_split(
    data["tweet"], y_train_down_syndrome, test_size=0.2, random_state=7
)

# Splits Dataset into Training and Testing set
x_train_idd, x_test_idd, y_train_idd, y_test_idd = train_test_split(
    data_2["tweet"], y_train_idd, test_size=0.2, random_state=7
)
```

The LabelEncoder converts categorical sentiment labels into numerical values, which helps in training the model. The categorical sentiment labels from 'sentiment' column in the two datasets are transformed into corresponding numerical labels, and stored into *y_train_down_syndrome*, and *y_train_idd*. The two datasets are then split into train and test sets. This ensures that the model is trained on portion of data and assessed on another to evaluate its performance on unseen data.

Transformer classification model

```
MAX_SEQUENCE_LENGTH = 40
```

Transformer classification tends to present a unique opportunity to perform different forms of sentiment analysis. In this case, it was vital to define the length of character; hence the maximum character length was set at 40. While focusing on the maximum sequence length, the need to focus on the underlying vocabulary and text data emerged. As a result, a tokenization model was applied to the dataset. Therefore, the snippet below was oriented towards splitting the dataset

into smaller units for better analysis. This was the foundation to applying the transformer classification model.

```
down_syndrome_tokenizer = Tokenizer()
down_syndrome_tokenizer.fit_on_texts(data.tweet)

word_index_down_syndrome = down_syndrome_tokenizer.word_index
vocab_size_down_syndrome = len(down_syndrome_tokenizer.word_index) + 1
print("Vocabulary Size :", vocab_size_down_syndrome)
```

Like any other machine learning classification model, Transformer tend to be accompanied by a unique set of features for accurate data analysis. However, the model cannot be applied on the collected data at random. For this reason, it was critical to train the text data based on the underlying characteristics and tokens. Therefore, the *x_train_down_syndrome* line is set to split the data and acquire data for training the model. Similarly, the *x_test_down_syndrome* was used acquire the testing data for the transformer model (see the snippet below).

```
x_train_down_syndrome = tf.keras.utils.pad_sequences(down_syndrome_tokenizer.text
s_to_sequences(x_train_down_syndrome.ravel()), maxlen=MAX_SEQUENCE_LENGTH)
x_test_down_syndrome = tf.keras.utils.pad_sequences(down_syndrome_tokenizer.texts
_to_sequences(x_test_down_syndrome.ravel()), maxlen=MAX_SEQUENCE_LENGTH)
```

Once the training and testing dataset have been obtained, one can continue with the analysis. In this case, the first step is to train the obtained dataset bets on the tokens and sequences. Therefore, the snippet below indicates the code that initiated the training and testing process for the transformer model.

```
x_train_idd = tf.keras.utils.pad_sequences(idd_tokenizer.texts_to_sequences(x_tra
in_idd.ravel()), maxlen=MAX_SEQUENCE_LENGTH)
x_test_idd = tf.keras.utils.pad_sequences(idd_tokenizer.texts_to_sequences(x_test
_idd.ravel()), maxlen=MAX_SEQUENCE_LENGTH)
```

The completion of training indicated that the focus can shift to further analysis. In this case, it was vital to focus on data normalization. As a result, key libraries and dependencies such as *MultiHeadAttention, LayerNormalization* and *Sequential* were imported to aid with the process.

```
from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization, Dropo
ut, Layer
from tensorflow.keras.layers import Embedding, Input, GlobalAveragePooling1D, Den
se
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.models import Sequential, Model
import warnings
warnings.filterwarnings("ignore", category=np.VisibleDeprecationWarning)
```

The normalization process was performed based on two distinct dynamics. Definition of a *Transformer block and token embedding.* The transformer block was focused on defining the overall structure if the model during the normalization process. This was accompanied by focusing on the scale-product attention unit and ensuring minimal reliance on reoccurrence. Where the token embedding was concerned, the focus was on positional embedding. This aspect of the code was primarily concerned with the dataset to ensure they were oriented towards analysis instead of being considered as a "*bag of words.*" In this case, the snippet focused on ensuring the position of the word in a given sentence is indexed instead of the whole sentence.

```
class TransformerBlock(Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = Sequential(
            [Dense(ff_dim, activation="relu"),
             Dense(embed_dim),]
        )
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
```

```
                out1 = self.layernorm1(inputs + attn_output)
                ffn_output = self.ffn(out1)
                ffn_output = self.dropout2(ffn_output, training=training)
                return self.layernorm2(out1 + ffn_output)

class TokenAndPositionEmbedding(Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super(TokenAndPositionEmbedding, self).__init__()
        self.token_emb = Embedding(input_dim=vocab_size, output_dim=embed_dim)
        self.pos_emb = Embedding(input_dim=maxlen, output_dim=embed_dim)

    def call(self, x):
        maxlen = tf.shape(x)[-1]
        positions = tf.range(start=0, limit=maxlen, delta=1)
        positions = self.pos_emb(positions)
        x = self.token_emb(x)
        return x + positions
```

The completion of normalization and token embedding implied that other facets of the model could be defined. In this case, the *Global average pooling, dropout, Denses and their activation* were defined. In this case, the *"relu" and softmax* were use as activation functions at defend stages of the transformer block. Subsequently, the model was defined and complied for further analysis and evaluation. Specifically, the *Adam optimizer* was used, with a key focus on the overall accuracy of the model. This was accompanied by the use of *sparse_categorical_crossentropy* for the calculation of losses during the compilation. Therefore, the model was further trained and tested based on the new metrics, with the application of *ReduceLROnPlateau* dependency.

```
embed_dim = 32  # Embedding size for each token
num_heads = 2  # Number of attention heads
ff_dim = 32  # Hidden layer size in feed forward network inside transformer

inputs = Input(shape=(MAX_SEQUENCE_LENGTH,))
embedding_layer = TokenAndPositionEmbedding(MAX_SEQUENCE_LENGTH, vocab_size_down_
syndrome, embed_dim)
x = embedding_layer(inputs)
transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
x = transformer_block(x)
x = GlobalAveragePooling1D()(x)
x = Dropout(0.1)(x)
x = Dense(20, activation="relu")(x)
```

```
x = Dropout(0.1)(x)
outputs = Dense(3, activation="softmax")(x)

model_down_syndrome = Model(inputs=inputs, outputs=outputs)
```

```
model_down_syndrome.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
```

```
ReduceLROnPlateau_ = ReduceLROnPlateau(factor=0.1, min_lr = 0.01, monitor = 'val_loss', verbose = 1)
history_transformer = model_down_syndrome.fit(
    x_train_down_syndrome, y_train_down_syndrome, batch_size=32, epochs=10,
    validation_data=(x_test_down_syndrome, y_test_down_syndrome), callbacks=[ReduceLROnPlateau_]
```

The task would not be completed without deifying the provided prediction model. Therefore, the subsequent snippet (see below) was utilized to define two prediction models based on the classifiers adopted. This was accompanied by plotting key metrics to define the accuracy and precision of the model.

```
# Model 1 predictions
scores_model_idd = model_idd.predict(x_test_idd, verbose=1, batch_size=10000)
pred_model_idd = np.argmax(scores_model_idd, axis=1)
report_model_idd = classification_report(y_test_idd, pred_model_idd, output_dict=True)

# Model 2 predictions
scores_model_down_syndrome = model_down_syndrome.predict(x_test_down_syndrome, verbose=1, batch_size=10000)
pred_model_down_syndrome = np.argmax(scores_model_down_syndrome, axis=1)
report_model_down_syndrome = classification_report(y_test_down_syndrome, pred_model_down_syndrome, output_dict=True)

# Get class labels and metrics from the classification reports
class_labels = list(report_model_idd.keys())[:-3]  # Exclude 'accuracy', 'macro avg', and 'weighted avg'
precision_scores_model_idd = [report_model_idd[label]['precision'] for label in class_labels]
recall_scores_model_idd = [report_model_idd[label]['recall'] for label in class_labels]
f1_scores_model_idd = [report_model_idd[label]['f1-score'] for label in class_labels]
```

```
precision_scores_model_down_syndrome = [report_model_down_syndrome[label]['precis
ion'] for label in class_labels]
recall_scores_model_down_syndrome = [report_model_down_syndrome[label]['recall']
for label in class_labels]
f1_scores_model_down_syndrome = [report_model_down_syndrome[label]['f1-score'] fo
r label in class_labels]

# Create bar charts to visualize the metrics for both models
plt.figure(figsize=(12, 6))
x = range(len(class_labels))
width = 0.35

plt.bar(x, precision_scores_model_idd, width, label='Precision - Model IDD')
plt.bar(x, recall_scores_model_idd, width, label='Recall - Model IDD', alpha=0.7)
plt.bar(x, f1_scores_model_idd, width, label='F1-score - Model IDD', alpha=0.5)
plt.bar([val + width for val in x], precision_scores_model_down_syndrome, width,
label='Precision - Model down_syndrome')
plt.bar([val + width for val in x], recall_scores_model_down_syndrome, width, lab
el='Recall - Model down_syndrome', alpha=0.7)
plt.bar([val + width for val in x], f1_scores_model_down_syndrome, width, label='
F1-score - Model down_syndrome', alpha=0.5)

plt.xlabel('Class')
plt.ylabel('Score')
plt.title('Comparison of Classification Metrics')
plt.xticks([val + width/2 for val in x], class_labels, rotation=45)
plt.legend()

plt.tight_layout()
plt.show()
```

Sentiment distribution using transformers

The completion of classification was followed by obtaining the classification report. In the snippet below the code focuses on the testing and the predicative aspects of the model. The report focuses on the frequencies of the idd and class frequencies used during classification. Once each aspect of the model had been determined, key plots associated with the model were obtained.

```
# Obtain the classification report for Model IDD
report_model_idd = classification_report(y_test_idd, pred_model_idd, output_dict=
True)

# Get the class labels and their corresponding frequencies
class_labels_idd = list(report_model_idd.keys())[:-3]
```

```
class_frequencies_idd = [report_model_idd[label]['support'] for label in class_la
bels_idd]
class_labels_idd = [int(label) for label in class_labels_idd]

# Plot the class frequencies for Model IDD
plt.figure(figsize=(8, 6))
plt.bar(lb.inverse_transform(class_labels_idd), class_frequencies_idd)
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.title('Class Frequencies - Model IDD')
plt.xticks(rotation=45)
plt.show()
```

The above snippet was followed by a shift in focus to examine model Down syndrome.

Therefore, the snippet for obtaining such metrics and performance plots are shown below:

```
# Obtain the classification report for Model down syndrome
report_model_down_syndrome = classification_report(y_test_down_syndrome, pred_mod
el_down_syndrome, output_dict=True)

# Get the class labels and their corresponding frequencies
class_labels_down_syndrome = list(report_model_down_syndrome.keys())[:-3]
class_frequencies_down_syndrome = [report_model_down_syndrome[label]['support'] f
or label in class_labels_down_syndrome]
class_labels_down_syndrome = [int(label) for label in class_labels_down_syndrome]

# Plot the class frequencies for Model down syndrome
plt.figure(figsize=(8, 6))
plt.bar(lb.inverse_transform(class_labels_down_syndrome), class_frequencies_down_
syndrome)
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.title('Class Frequencies - Model down_syndrome')
plt.xticks(rotation=45)
plt.show()
```

Results

The experiment utilised Twitter-based datasets, collected from individuals with intellectual disabilities. The study involved three key analysis: sentiment analysis, topic modelling and comparative analysis. Sentiment analysis aimed to establish the emotions or sentiments in the

datasets, topic modelling was conducted to extract the common topics discussed by the two groups, and a comparative analysis aimed to establish the differences and similarities in experiences between individuals with IDD and Down syndrome while interacting online.

	username	user_description	tweet	is_retweet
0	cosmicfizzypop	dnfi if you're nsfw	sometimes i wish i didn't have a learning disa...	Yes
1	Naltonpi	Fat Falco and Joker. I enjoy both Yugioh and S...	I have a learning disability. It'scalled Heart...	No
2	arminiu_ss	Cogito Ergo Sum	We all are intellectually challenged to some e...	Yes
3	KiwiFunknuckle	The original and only Kiwifunknuckle. Small fl...	I really struggled to read and write. I repeat...	Yes
4	CatherineHorbu3	I am a Peer Support worker for learning disabi...	I am proud of being a woman with a lived exper...	Yes

Figure 2: IDD Dataset

	username	user_description	tweet	is_retweet
0	baker_banter	Hi! My name is Derek, I am 23, I have Down syn...	HAPPY WORLD DOWN SYNDROME DAY AND HAPPY 2nd Bl...	No
1	MaxAPeters	Max Peters is a dynamic young man with Down Sy...	World Down Syndrome Day is coming up March 21!...	Yes
2	MikaylaHDancer	I am a self advocate. I am the first woman wit...	I had so much fun at Down syndrome conference ...	No
3	DiedraMckesson	My name is Diedra Mckesson I have Down Syndro...	Would bring awareness about the importance of ...	Yes
4	clbagelboy	I am an KMR/SAG Actor who has Down syndrome. I...	Happy Disability Pride Month!! I am so happy t...	No

Figure 1: Down Syndrome Group

The figures above show the features of the two datasets: IDD dataset and the down_syndrome dataset used in the experiment. The datasets contained four key components: username, user_description, tweet, and is_retweet. The tweet column was used for analysis and topic modelling to extract sentiments and main topics in the datasets.

Topic Modelling

Figure 3: IDD Word cloud

Figure 4: Down syndrome word cloud

- The word cloud produced the top 50 prevalent words in both datasets. The topics were derived from the clusters of top keywords in the Down syndrome group and IDD dataset.
- The two topics based on manual inspection are awareness, and cyberbullying and online safety.
- From inspection of the commonly identified words, it is clear that individuals with ID recorded high bullying and online safety-related cases in cyberspace compared to individuals with Down syndrome. The Down syndrome dataset exhibited more keywords associated with awareness than the IDD dataset. This suggests possible differences in how individuals with different conditions interact on social platforms.

Sentiment Analysis

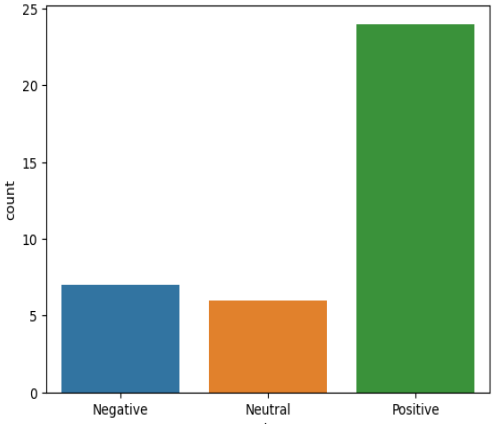

Figure 6: Sentiment Distribution for IDD Group

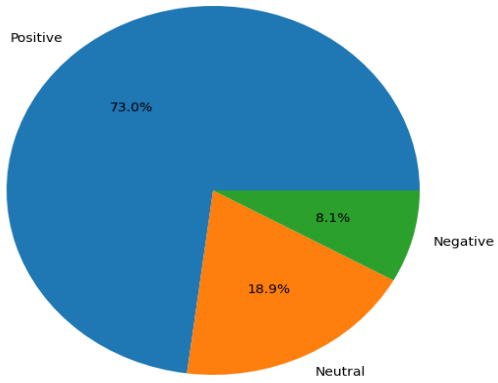

Figure 5: Sentiment Distribution for Down Syndrome Group

- Individuals with IDD profiles expressed significant positive sentiments (64.9%) in cyberspace. The remaining segment of individuals reported neutral and negative sentiments, with 18.9% having neutral emotions and only 16.2% reporting negative interactions in the online space.

- Down syndrome profiles had an increased frequency of positive sentiment scores (73%) in social media. A small section of participants recorded 18.9% neutral expressions and 9.1% negative expressions on social media.

Model Performance and Metric Comparison

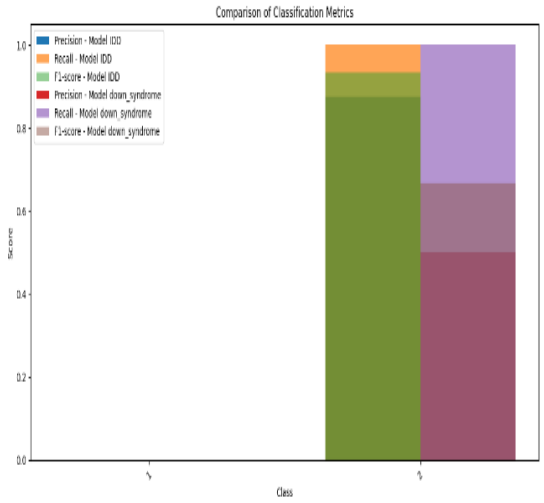

Table 1: Metrics Comparison for IDD Dataset

Class	Precision-Model IDD	Recall-Model IDD	F1-score-Model IDD
1	0.0	0.0	0.0
2	87.5	100	93.3333

Table 2: Metrics Comparison for Down syndrome Dataset

	Syndrome	Syndrome	Syndrome
1	0.000000	0.000000	0.000000
2	50	100	66.6667

- From the table above, the model recorded 0 % and 87.5 % precision scores for the sentiments in the IDD dataset.
- For the IDD dataset, the recall values are 0% and 100% for sentiments (negative and positive).
- The IDD model yielded 0.0%, and 93.333% F1-scores for the sentiments. This indicates that the model is effective in predicting positive sentiments but less effective in identifying negative sentiments
- For the Down syndrome group, the model recorded 0% and 50% precision scores for the sentiments (negative, positive).
- The model recorded recall scores of 0.0% and 100% for the Down-Syndrome dataset.
- For the Down-Syndrome dataset, the F1 scores attained were 0%, and 66.6667%, revealing that the model struggles to identify negative but performs better in identifying positive sentiments.

Sentiment Distribution Using Transformers

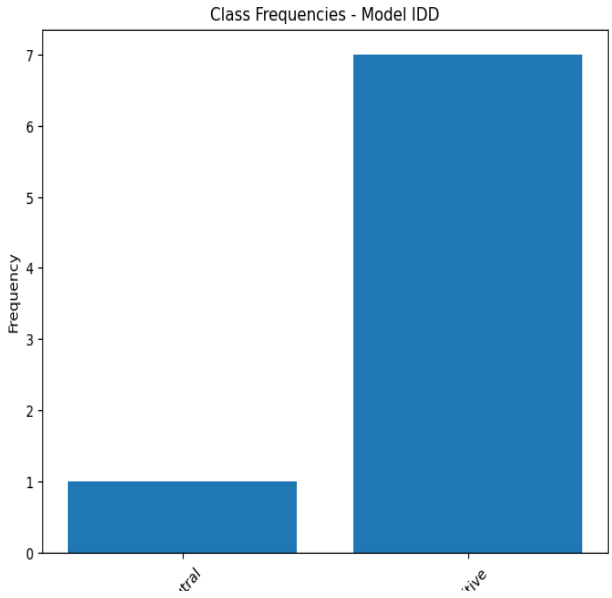

Figure 7: IDD

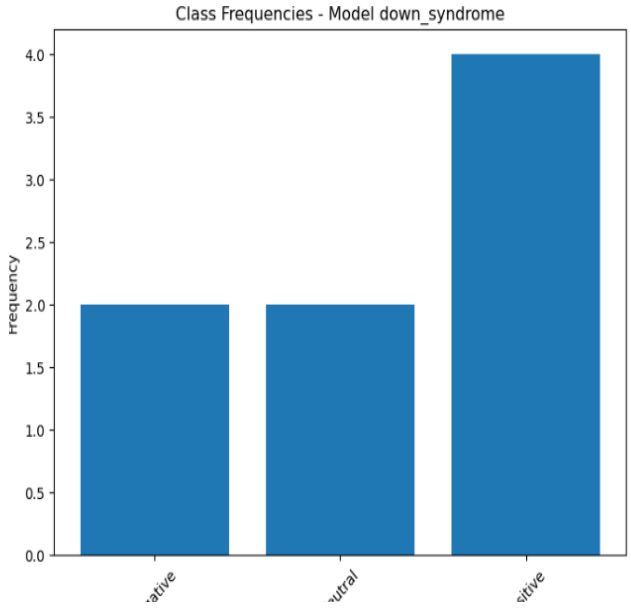

Figure 8: Down syndrome