

getBERTEmbeddingsForBatch(): Function to obtain BERT embeddings for a batch of Reddit posts

```
In [1]: def getBERTEmbeddingsForBatch(device, posts, tokenizer, model, isPsychBert = False):
    clearGPUMem()
    tokenizedPosts = tokenizer(posts, padding=True, truncation=True, return_tensors='pt', max_length=512)
    tokenizedPosts.to(device)
    model.to(device)
    with torch.no_grad():
        outputs = model(**tokenizedPosts)
    model.cpu() #Move the model off the GPU to free up memory space
    if isPsychBert == True: #Process for collecting CLS embeddings is slightly different for PsychBERT
        last_hidden_state = outputs['hidden_states'][-1]
        clsEmbeddings = last_hidden_state[:, 0, :]
    else:
        clsEmbeddings = outputs.last_hidden_state[:, 0, :] #For all other models other than PsychBERT

    clsEmbeddings = clsEmbeddings.cpu().numpy() #Send embeddings back to the cpu and convert to a Numpy array
    return clsEmbeddings
```

allEmbeddingsForModel(): Function to concatenate a list of ALL batches of embeddings, and return a complete list for a specific BERT model.

```
In [2]: def allEmbeddingsForModel(XPos, XNeg, currentTokenizer, currentModel, modelName, isPsychBert):
    embeddingsPos = [] #Initialize an empty list to store positive BERT embeddings
    embeddingsNeg = [] #Initialize an empty list to store negative BERT embeddings
    data = [XPos, XNeg]

    for index in range(2):
        #Obtain the Reddit post embeddings in sets of 256 posts. This is done to ensure everything can fit into
        batchSize = 256
        numBatches = (len(data[index]) - 1) // batchSize + 1

        if index == 0:
            classSet = 'POSITIVE'
        else:
            classSet = 'NEGATIVE'
        print(f'Obtaining {modelName} {classSet} vector embeddings for {len(data[index])} Reddit posts across {')

        for i in range(numBatches):
            start = i * batchSize
            end = min((i + 1) * batchSize, len(data[index]))
            print(f'Processing batch {i+1}/{numBatches}.')

            batchOfPosts = data[index][start:end]

            #Get BERT embeddings for the current batch of texts
            embeddings = getBERTEmbeddingsForBatch(device, batchOfPosts, currentTokenizer, currentModel, isPsychBert)
            if index == 0:
                embeddingsPos.extend(embeddings)
            else:
                embeddingsNeg.extend(embeddings)

        print(f'Finished obtaining all {modelName} embeddings.')
        print('\n-----\n')
    return embeddingsPos, embeddingsNeg
```

doDatasetInvestigation(): This function does not perform any calculations towards the results, but does help to confirm some of the expectations of what the dataset should look like. The assumption that the data can be labelled as 'positive' or 'negative' from its subreddit alone is somewhat reasonable only because of this dataset investigation. MinMax scaling is performed from 0-1 and the dimensionality is reduced to 2D for hexbin plotting. Confirming expectations of each class to be similar within itself but different from each other, these hexbin plots show ~1 cluster for the positive class (r/ptsd) and ~8 clusters for the negative class (8 various unrelated subreddits). This is more clearly shown for MentalBERT and PsychBERT, which ultimately have the best performances.

```
In [3]: def doDatasetInvestigation(X_Embeddings, y_Labels):
    X_scaled = MinMaxScaler().fit_transform(X_Embeddings) #Scale features to [0,1] range
    mapper = UMAP(n_components=2, metric="cosine").fit(X_scaled) #Initialize and fit UMAP
    df_emb = pd.DataFrame(mapper.embedding_, columns=["X", "Y"]) #Create a DataFrame of 2D embeddings
    df_emb["label"] = y_Labels
    df_emb.head() #Display initial dataframe rows

    fig, axes = plt.subplots(1, 2, figsize=(7,5))
    axes = axes.flatten()
    cmaps = ["Reds", "Greens"]
    labels = ['Negative', 'Positive']
```

```

for i, (label, cmap) in enumerate(zip(labels, cmaps)):
    df_emb_sub = df_emb.query(f"label == {i}")
    axes[i].hexbin(df_emb_sub["X"], df_emb_sub["Y"], cmap=cmap,
                  gridsize=20, linewidths=(0,))
    axes[i].set_title(label)
    axes[i].set_xticks([]), axes[i].set_yticks([])

plt.tight_layout()
plt.show()

```

doKMedoids(): Function that performs a typical model training/testing loop using Monte Carlo cross validation over 10 trials. The clusters that are formed after training are labelled based on the majority of the final observations that are assigned to it. Then, predictions for the test set are made. The accuracy, precision, recall, F1-score, and confusion matrix for the testing set is stored in a list and returned from the function.

```

In [4]: def doKMedoids(X_Embeddings, y_Labels, trials = 10, doPrint = False):
    allAccuracies = []
    allPrecision = []
    allRecall = []
    allFScore = []
    allMatrices = []
    for trial in range(trials):

        X_train, X_test, y_train, y_test = train_test_split(X_Embeddings, y_Labels, test_size=0.2, shuffle=True)

        #Classifier instance
        kmeans = KMedoids(n_clusters=9, metric='cosine', method='alternate', init='random', max_iter=300)

        #Training
        kmeans.fit(X_train)

        #Assign majority ground truth label from training data as the predicted label for each cluster
        cluster_majority_labels = {}
        for cluster in range(kmeans.n_clusters):
            cluster_data_indices = np.where(kmeans.labels_ == cluster)[0]
            cluster_ground_truth_labels = [y_train[cluster_data_index] for cluster_data_index in cluster_data_indices]
            majority_label = Counter(cluster_ground_truth_labels).most_common(1)[0][0]
            cluster_majority_labels[cluster] = majority_label

        #Predict cluster labels for test data
        test_cluster_labels = kmeans.predict(X_test)

        #Assign predicted labels based on majority ground truth labels of clusters
        predictions = [cluster_majority_labels[cluster] for cluster in test_cluster_labels]

        #Calculate and store all performance metrics for this trial
        allAccuracies.append(accuracy_score(y_test, predictions))
        allPrecision.append(precision_score(y_test, predictions, average='macro'))
        allRecall.append(recall_score(y_test, predictions, average='macro'))
        allFScore.append(f1_score(y_test, predictions, average='macro'))

        #print(f'trial {trial}: accuracy: {allAccuracies[-1]}') #TEMP DEBUG

        #Confusion matrices
        confusionMatrix = confusion_matrix(y_test, predictions)
        allMatrices.append(confusionMatrix)

        if doPrint == True:
            report = classification_report(y_test, predictions) #Classification report
            print(f"{trial + 1}. Confusion Matrix:")
            print(confusionMatrix)
            print(f"{trial + 1}. Classification Report:")
            print(report)

    return allAccuracies, allPrecision, allRecall, allFScore, allMatrices

```

plotPerformance(): Function that displays a plot for a given performance metric (i.e. accuracy, precision, etc.). This function will display the average performance as well as a 95% confidence interval for each BERT variation being compared.

```

In [5]: def plotPerformance(performanceData, metricIndex):

    colors = ['b', 'g', 'r', 'c', 'm'] #Define the colors for each model

    x_labels = list(performanceData.keys()) #Define the x axis labels

    positions = np.arange(len(performanceData)) #Define the positions for the bars

```

```

bar_width = 0.5 #Define the width of the bars

metric = ['Accuracy', 'Precision', 'Recall', 'F1-score'][metricIndex] #Get the selected metric

#Initialize lists to store mean values and confidence intervals
means = []
confidenceIntervals = []

#Compute mean and confidence interval for each model
for values in performanceData.values():
    mean = np.mean(values)
    confidenceInterval = 1.96 * np.std(values) / np.sqrt(len(values))
    means.append(mean)
    confidenceIntervals.append(confidenceInterval)

# Plot the bars
plt.bar(positions, means, bar_width, yerr=confidenceIntervals, align='center', alpha=0.5, color=colors[:len(positions)])
plt.xticks(positions, x_labels)
plt.xlabel('Model')
plt.ylabel(metric)
plt.title(f'{metric} Comparison')
plt.ylim(.6, 1) #Set y-axis limits. These values cover all BERT model performances in this project.
plt.tight_layout()
plt.show()

```

clearGPUMem(): Clear GPU memory before doing embeddings

```

In [6]: def clearGPUMem():
        if device == 'cuda':
            torch.cuda.empty_cache()
        #clearGPUMem()

```

Imports

```

In [17]: #pip install tensorflow
#pip install umap-learn
#pip install scikit-learn-extra

import torch
import pandas as pd
import numpy as np
from transformers import AutoTokenizer, AutoModel #For MentalBERT and ClinicalBERT
from transformers import AutoModelForSequenceClassification #For PsychBERT
from transformers import BertTokenizer, BertModel #For Base BERT
from transformers import DistilBertTokenizer, DistilBertModel #For DistilBERT
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from umap import UMAP
from collections import Counter
from sklearn_extra.cluster import KMedoids

```

Set the device to CUDA for faster processing.

```

In [8]: #Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device

```

```

Out[8]: device(type='cuda')

```

The next few cells are declarations of the BERT variation models and tokenizers. These are added to dictionaries for ease of access throughout the remainder of the program.

```

In [9]: #MentalBERT - The warning produced by this cell is fine because this model is not used for a down-stream task.
mental_tokenizer = AutoTokenizer.from_pretrained("mental/mental-bert-base-uncased", token='hf_jxqlADYZSNdhbEjnF')
mental_model = AutoModel.from_pretrained("mental/mental-bert-base-uncased", token='hf_jxqlADYZSNdhbEjnFSRfealxM')

```

Some weights of BertModel were not initialized from the model checkpoint at mental/mental-bert-base-uncased and are newly initialized: ['bert.pooler.dense.weight', 'bert.pooler.dense.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

```

In [10]: #PsychBERT
psych_tokenizer = AutoTokenizer.from_pretrained("mnaylor/psychbert-finetuned-mentalhealth")
psych_model = AutoModelForSequenceClassification.from_pretrained("mnaylor/psychbert-finetuned-mentalhealth", ou

```

```
#Note the output_hidden_states argument that is required for PsychBERT but not the other models
```

```
In [11]: #ClinicalBERT
clinical_tokenizer = AutoTokenizer.from_pretrained("medicalai/ClinicalBERT")
clinical_model = AutoModel.from_pretrained("medicalai/ClinicalBERT")

In [12]: #DistilBERT
distil_tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
distil_model = DistilBertModel.from_pretrained("distilbert-base-uncased")

In [13]: #BERT
bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained("bert-base-uncased")

In [14]: allModels = {
    'MentalBERT' : mental_model,
    'PsychBERT' : psych_model,
    'ClinicalBERT' : clinical_model,
    'BERT' : bert_model,
    'DistilBERT' : distil_model
}

allTokenizers = {
    'MentalBERT' : mental_tokenizer,
    'PsychBERT' : psych_tokenizer,
    'ClinicalBERT' : clinical_tokenizer,
    'BERT' : bert_tokenizer,
    'DistilBERT' : distil_tokenizer
}
```

Configuration variables

```
In [15]: runForModels = [
    'MentalBERT',
    'PsychBERT',
    'ClinicalBERT',
    'BERT',
    'DistilBERT'
] #Comment out whichever models to exclude from the performance comparison

runForSampleOfDataset = False #When true, this will run the program for a random sample of the dataset (to save
randomSampleSize = 1000 #Only used when the above 'runForSampleOfDataset' is set to True
```

Collecting positive and negative dataframes

```
In [16]: #Read the CSV files into a DataFrame
df_pos = pd.read_csv('positive_cleaned.csv')
df_neg = pd.read_csv('negative_cleaned.csv')

#Add a new column 'label' to each DataFrame with the respective label
df_pos['label'] = 1
df_neg['label'] = 0

if runForSampleOfDataset == True:
    df_pos = df_pos.sample(n=randomSampleSize, random_state=42)
    df_neg = df_neg.sample(n=randomSampleSize, random_state=42)

#Converting positive observations to Python lists
XPos = df_pos['posts'].tolist()
yPos = df_pos['label'].tolist()

#Converting negative observations to Python lists
XNeg = df_neg['posts'].tolist()
yNeg = df_neg['label'].tolist()
```

All embeddings for each model are collected in the below cell and stored in a dictionary under the model that created them.

```
In [18]: modelData = {}
y_Labels = yPos + yNeg

for modelStr in runForModels:
    currentModel = allModels[modelStr]
    currentTokenizer = allTokenizers[modelStr]
    if modelStr == 'PsychBERT':
        isPsychBert = True
    else:
```

```

isPsychBert = False
print(f'***Collecting embeddings for {modelStr}***\n')
embeddingsPos, embeddingsNeg = allEmbeddingsForModel(XPos, XNeg, currentTokenizer, currentModel, modelStr, :
X_Embeddings = embeddingsPos + embeddingsNeg

modelData[modelStr] = {
    'embeddings' : X_Embeddings
}

```

Collecting embeddings for MentalBERT

Obtaining MentalBERT POSITIVE vector embeddings for 13697 Reddit posts across 54 batches.

Processing batch 1/54.
 Processing batch 2/54.
 Processing batch 3/54.
 Processing batch 4/54.
 Processing batch 5/54.
 Processing batch 6/54.
 Processing batch 7/54.
 Processing batch 8/54.
 Processing batch 9/54.
 Processing batch 10/54.
 Processing batch 11/54.
 Processing batch 12/54.
 Processing batch 13/54.
 Processing batch 14/54.
 Processing batch 15/54.
 Processing batch 16/54.
 Processing batch 17/54.
 Processing batch 18/54.
 Processing batch 19/54.
 Processing batch 20/54.
 Processing batch 21/54.
 Processing batch 22/54.
 Processing batch 23/54.
 Processing batch 24/54.
 Processing batch 25/54.
 Processing batch 26/54.
 Processing batch 27/54.
 Processing batch 28/54.
 Processing batch 29/54.
 Processing batch 30/54.
 Processing batch 31/54.
 Processing batch 32/54.
 Processing batch 33/54.
 Processing batch 34/54.
 Processing batch 35/54.
 Processing batch 36/54.
 Processing batch 37/54.
 Processing batch 38/54.
 Processing batch 39/54.
 Processing batch 40/54.
 Processing batch 41/54.
 Processing batch 42/54.
 Processing batch 43/54.
 Processing batch 44/54.
 Processing batch 45/54.
 Processing batch 46/54.
 Processing batch 47/54.
 Processing batch 48/54.
 Processing batch 49/54.
 Processing batch 50/54.
 Processing batch 51/54.
 Processing batch 52/54.
 Processing batch 53/54.
 Processing batch 54/54.

Obtaining MentalBERT NEGATIVE vector embeddings for 13696 Reddit posts across 54 batches.

Processing batch 1/54.
 Processing batch 2/54.
 Processing batch 3/54.
 Processing batch 4/54.
 Processing batch 5/54.
 Processing batch 6/54.
 Processing batch 7/54.
 Processing batch 8/54.
 Processing batch 9/54.
 Processing batch 10/54.
 Processing batch 11/54.
 Processing batch 12/54.
 Processing batch 13/54.
 Processing batch 14/54.
 Processing batch 15/54.
 Processing batch 16/54.
 Processing batch 17/54.

Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Finished obtaining all MentalBERT embeddings.

Collecting embeddings for PsychBERT

Obtaining PsychBERT POSITIVE vector embeddings for 13697 Reddit posts across 54 batches.

Processing batch 1/54.
Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.
Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.

Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Obtaining PsychBERT NEGATIVE vector embeddings for 13696 Reddit posts across 54 batches.
Processing batch 1/54.
Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.
Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Finished obtaining all PsychBERT embeddings.

Collecting embeddings for ClinicalBERT

Obtaining ClinicalBERT POSITIVE vector embeddings for 13697 Reddit posts across 54 batches.
Processing batch 1/54.
Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.

Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Obtaining ClinicalBERT NEGATIVE vector embeddings for 13696 Reddit posts across 54 batches.
Processing batch 1/54.
Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.
Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.

Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Finished obtaining all ClinicalBERT embeddings.

Collecting embeddings for BERT

Obtaining BERT POSITIVE vector embeddings for 13697 Reddit posts across 54 batches.

Processing batch 1/54.
Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.
Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.

Obtaining BERT NEGATIVE vector embeddings for 13696 Reddit posts across 54 batches.

Processing batch 1/54.

Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.
Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Finished obtaining all BERT embeddings.

Collecting embeddings for DistilBERT

Obtaining DistilBERT POSITIVE vector embeddings for 13697 Reddit posts across 54 batches.

Processing batch 1/54.
Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.
Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.

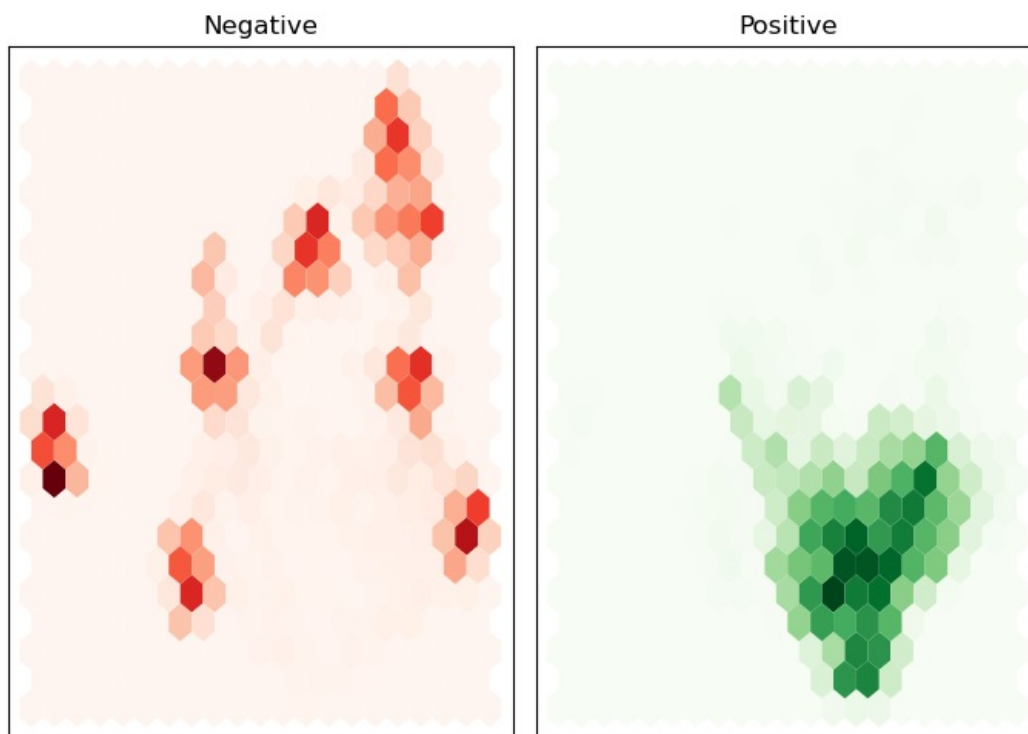
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.
Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Obtaining DistilBERT NEGATIVE vector embeddings for 13696 Reddit posts across 54 batches.
Processing batch 1/54.
Processing batch 2/54.
Processing batch 3/54.
Processing batch 4/54.
Processing batch 5/54.
Processing batch 6/54.
Processing batch 7/54.
Processing batch 8/54.
Processing batch 9/54.
Processing batch 10/54.
Processing batch 11/54.
Processing batch 12/54.
Processing batch 13/54.
Processing batch 14/54.
Processing batch 15/54.
Processing batch 16/54.
Processing batch 17/54.
Processing batch 18/54.
Processing batch 19/54.
Processing batch 20/54.
Processing batch 21/54.
Processing batch 22/54.
Processing batch 23/54.
Processing batch 24/54.
Processing batch 25/54.
Processing batch 26/54.
Processing batch 27/54.
Processing batch 28/54.
Processing batch 29/54.
Processing batch 30/54.
Processing batch 31/54.
Processing batch 32/54.
Processing batch 33/54.
Processing batch 34/54.
Processing batch 35/54.
Processing batch 36/54.
Processing batch 37/54.
Processing batch 38/54.
Processing batch 39/54.
Processing batch 40/54.
Processing batch 41/54.
Processing batch 42/54.
Processing batch 43/54.
Processing batch 44/54.
Processing batch 45/54.
Processing batch 46/54.
Processing batch 47/54.
Processing batch 48/54.
Processing batch 49/54.
Processing batch 50/54.
Processing batch 51/54.

Processing batch 52/54.
Processing batch 53/54.
Processing batch 54/54.
Finished obtaining all DistilBERT embeddings.

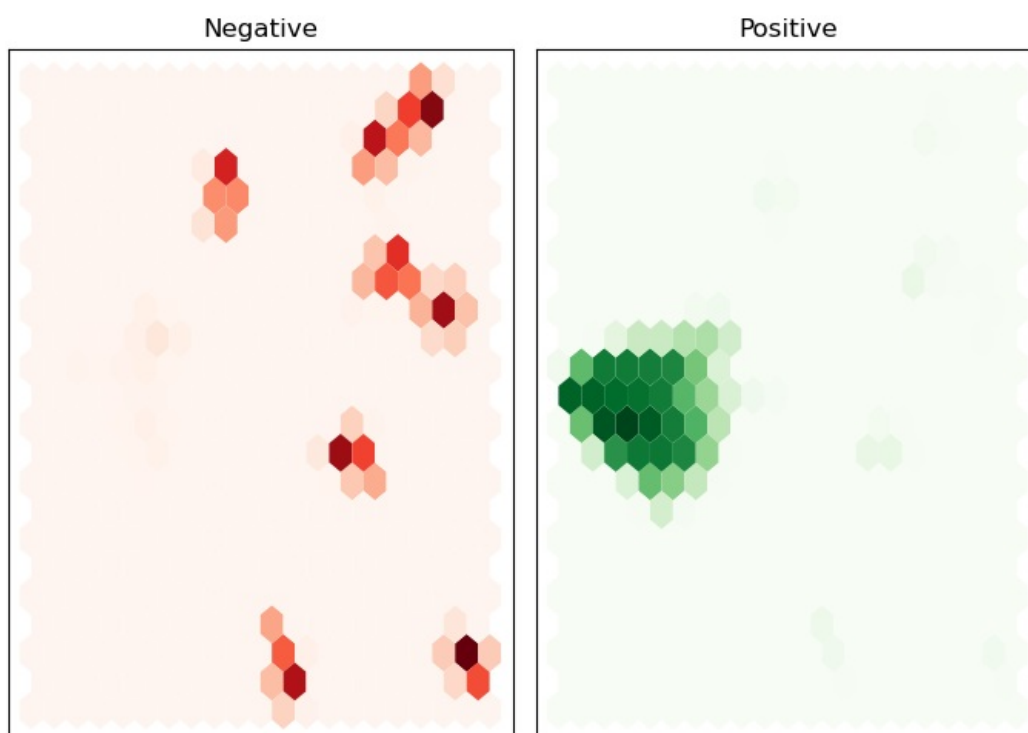
This section performs a dataset investigation/visualization to improve confidence in the data collection methods for this research. It does not contribute to the results directly. More details can be found in `doDatasetInvestigation()` with the function definitions in this notebook.

```
In [19]: for modelStr in runForModels:
          print(f'***2-Dimensional dataset embeddings from {modelStr}***\n')
          modelEmbeddings = modelData[modelStr]['embeddings']
          doDatasetInvestigation(modelEmbeddings, y_Labels)
```

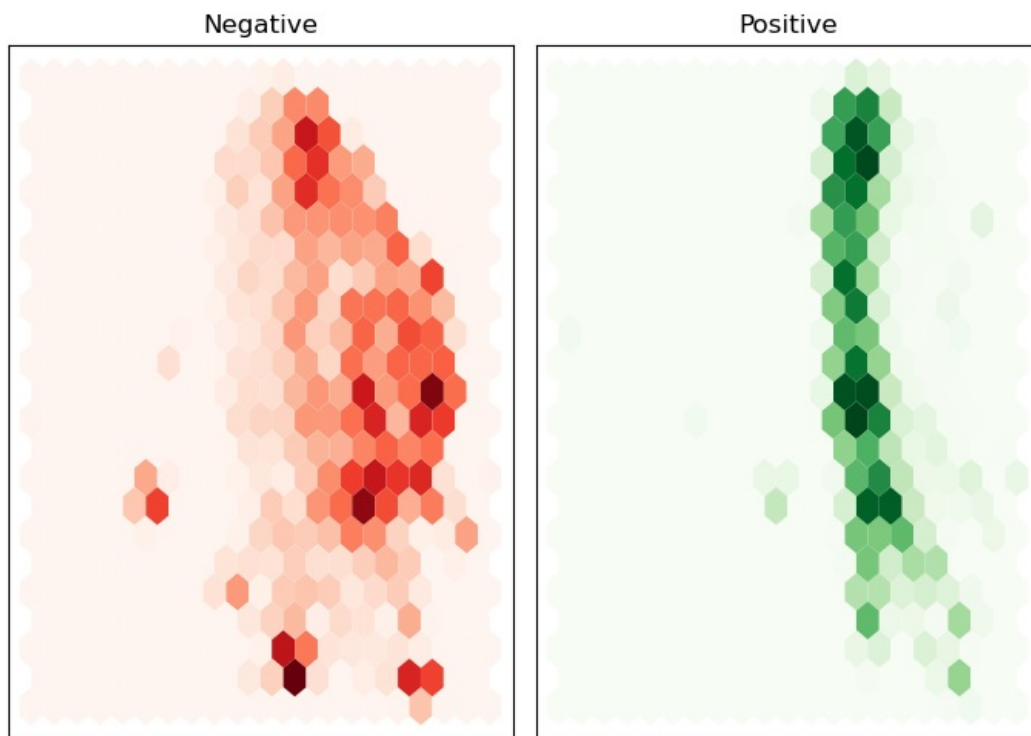
2-Dimensional dataset embeddings from MentalBERT



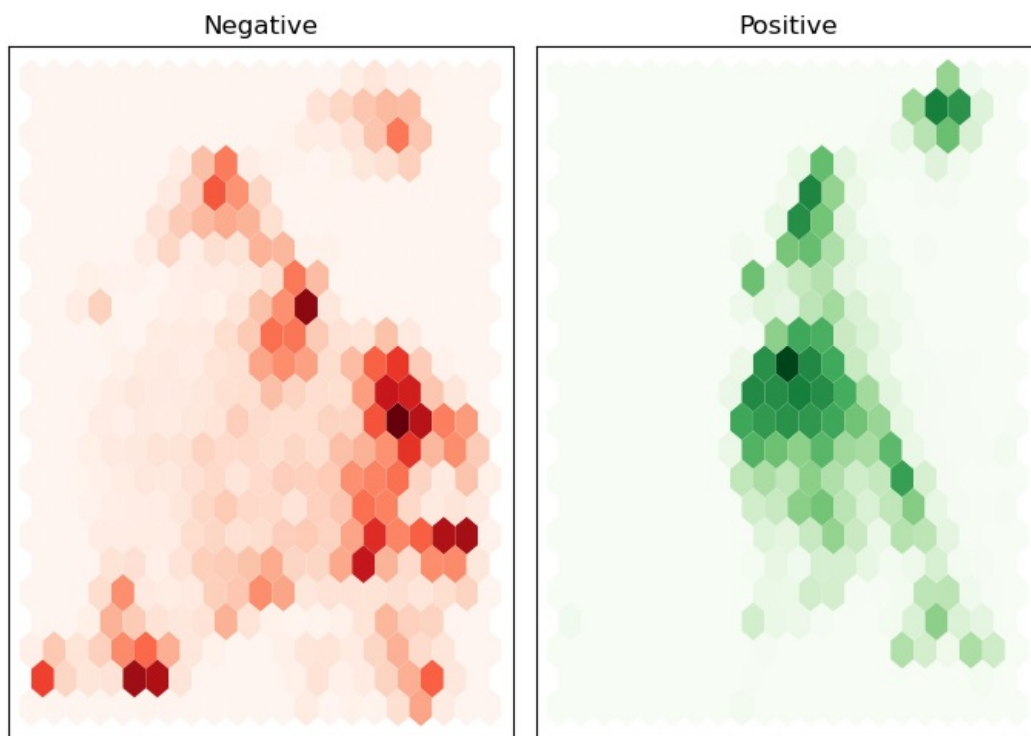
2-Dimensional dataset embeddings from PsychBERT



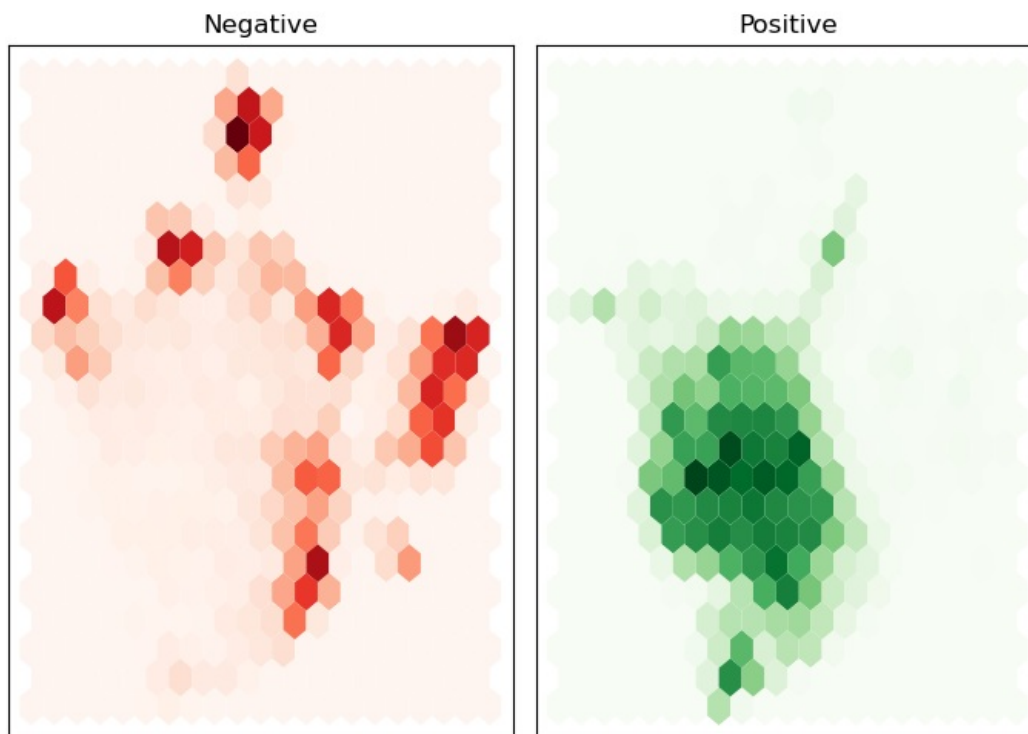
2-Dimensional dataset embeddings from ClinicalBERT



2-Dimensional dataset embeddings from BERT



2-Dimensional dataset embeddings from DistilBERT



This section runs a K-Medoids classifier for 10 trials per model and prints the average performance for each model.

```
In [20]: for modelStr in runForModels:
    print(f'***Performing KMedoids training and testing using {modelStr} embeddings***\n')
    modelEmbeddings = modelData[modelStr]['embeddings']
    allAccuracies, allPrecision, allRecall, allFScore, allMatrices = doKMedoids(modelEmbeddings, y_Labels, tria
    modelData[modelStr]['accuracy'] = allAccuracies
    modelData[modelStr]['precision'] = allPrecision
    modelData[modelStr]['recall'] = allRecall
    modelData[modelStr]['fscore'] = allFScore

    avgMatrix = np.mean(allMatrices, axis=0)
    avgAccuracy = np.mean(allAccuracies)
    avgPrecision = np.mean(allPrecision)
    avgPrecision = np.mean(allPrecision)
    avgRecall = np.mean(allRecall)
    avgFScore = np.mean(allFScore)

    print(f'Average Confusion Matrix:\n{avgMatrix}')
    print(f'Accuracy: {avgAccuracy}')
    print(f'Precision: {avgPrecision}')
    print(f'Recall: {avgRecall}')
    print(f'F1-Score: {avgFScore}\n')
```

Performing KMedoids training and testing using MentalBERT embeddings

```
Average Confusion Matrix:
[[2310.2  409.6]
 [ 402.5 2356.7]]
Accuracy: 0.8517795218105494
Precision: 0.8567487221230632
Recall: 0.8519784890081011
F1-Score: 0.8512341640394169
```

Performing KMedoids training and testing using PsychBERT embeddings

```
Average Confusion Matrix:
[[2658.9   80. ]
 [  98.3 2641.8]]
Accuracy: 0.9674575652491331
Precision: 0.9678845540376153
Recall: 0.9673623115071031
F1-Score: 0.9674408071870706
```

Performing KMedoids training and testing using ClinicalBERT embeddings

```
Average Confusion Matrix:
[[1568.7 1164.9]
 [ 624.3 2121.1]]
Accuracy: 0.6734440591348786
Precision: 0.6830472454766252
Recall: 0.6734344189519321
F1-Score: 0.6689786762230485
```

Performing KMedoids training and testing using BERT embeddings

```
Average Confusion Matrix:
[[1851.6  892.6]
 [ 846.4 1888.4]]
Accuracy: 0.6826063150209892
Precision: 0.6886347290243983
Recall: 0.6828596136643066
F1-Score: 0.6801286133365956
```

Performing KMedoids training and testing using DistilBERT embeddings

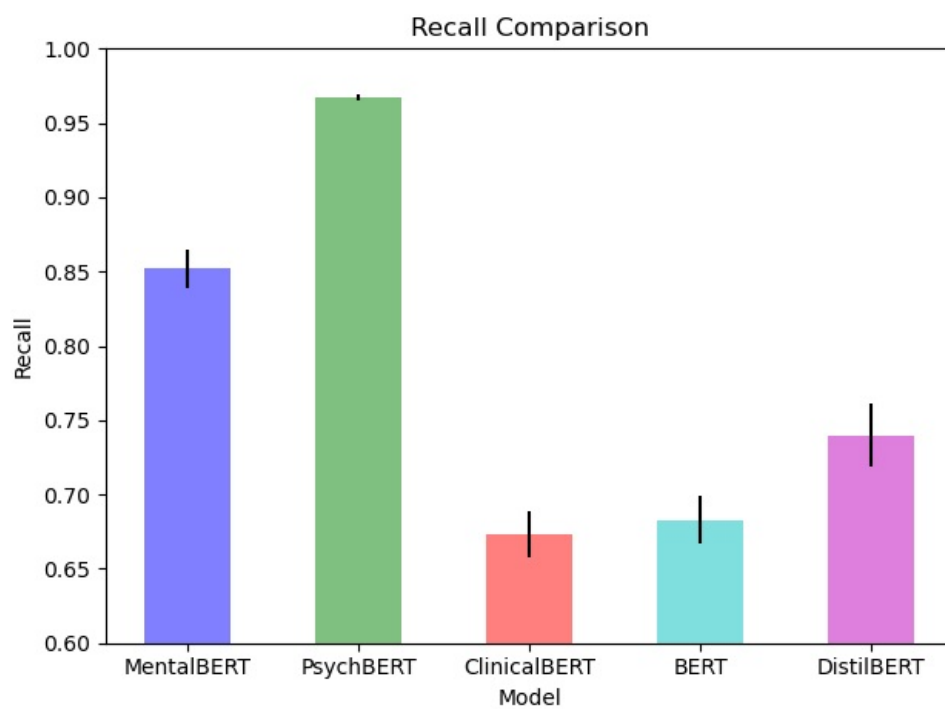
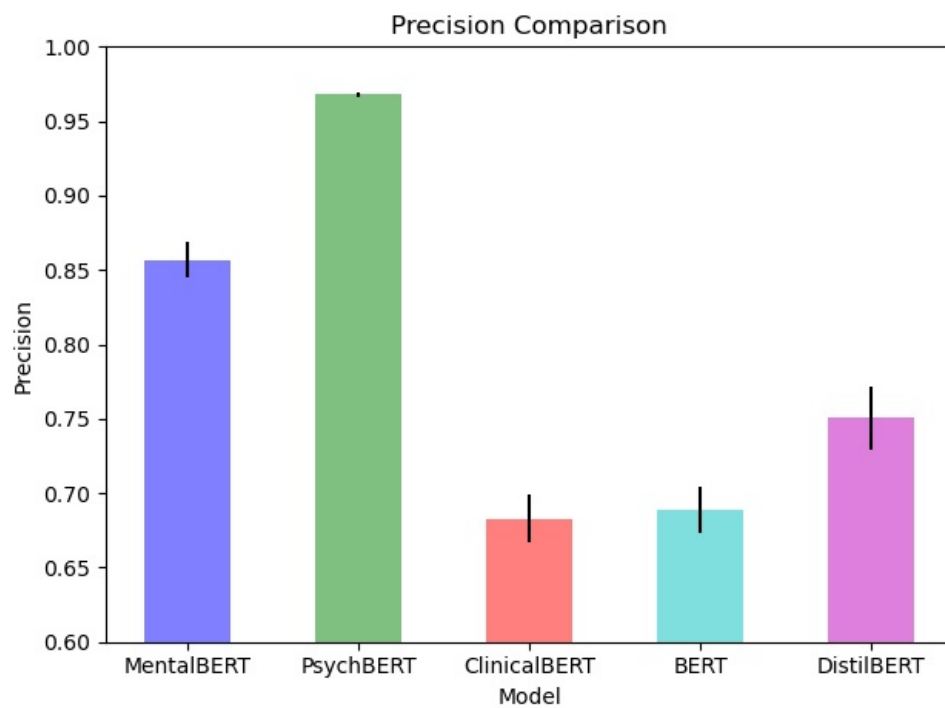
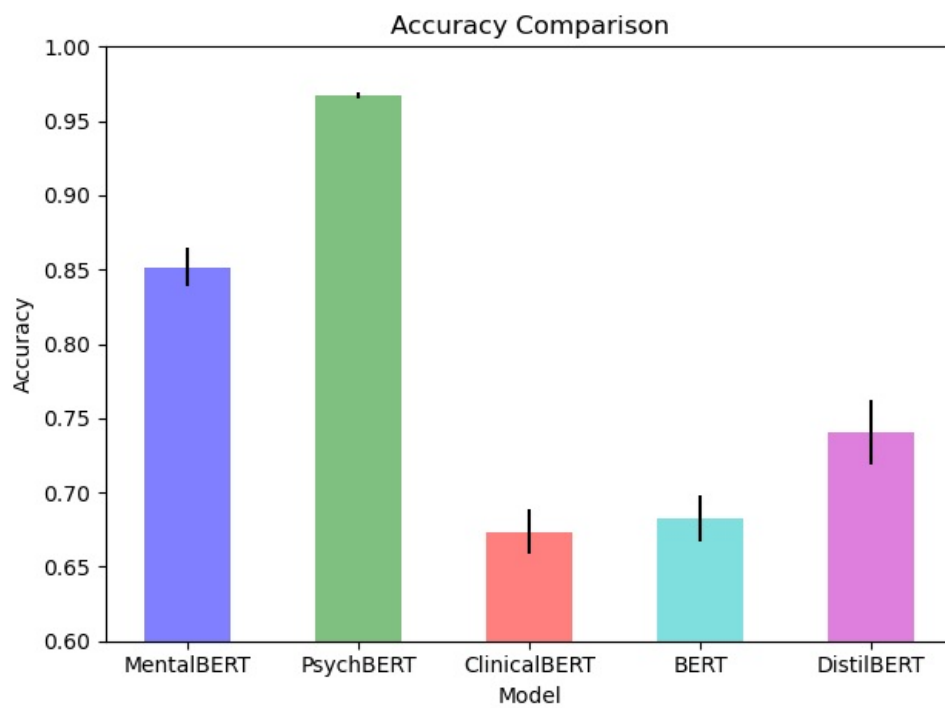
```
Average Confusion Matrix:
[[1889.4  843.7]
 [ 579.1 2166.8]]
Accuracy: 0.7403175762000365
Precision: 0.7504485640875669
Recall: 0.7398457285702855
F1-Score: 0.737313237525679
```

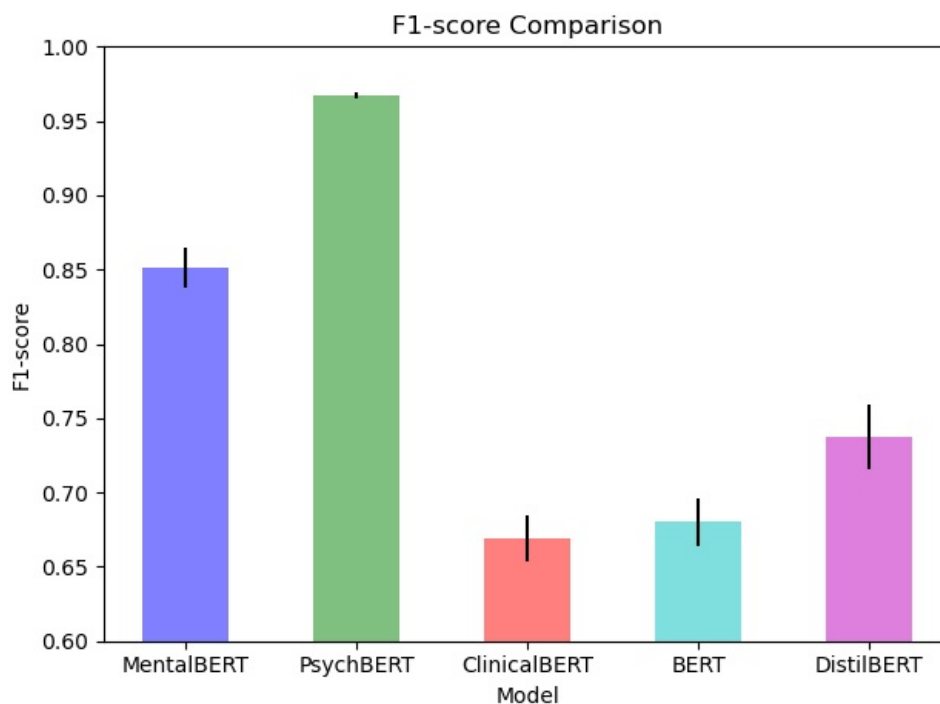
This section displays the final results in the form of 4 bar charts. Each compares the included BERT variants in terms of a different performance metric (accuracy, precision, recall, and F1-score).

```
In [21]: accuracyData = {}
precisionData = {}
recallData = {}
fscoreData = {}

for modelStr in runForModels:
    accuracyData[modelStr] = modelData[modelStr]['accuracy']
    precisionData[modelStr] = modelData[modelStr]['precision']
    recallData[modelStr] = modelData[modelStr]['recall']
    fscoreData[modelStr] = modelData[modelStr]['fscore']

#Compare performance for different metrics
plotPerformance(accuracyData, 0) #Accuracy
plotPerformance(precisionData, 1) #Precision
plotPerformance(recallData, 2) #Recall
plotPerformance(fscoreData, 3) #F1-Score
```





This below cell contains code that is not intended to be run. It is listed here as documentation of some of the failed methods of confirming differences between the positive and negative class, as well as similarity within themselves. The main approaches used here are as follows:

- Standardization (z-scoring) the embeddings
- Unit vectors of the embeddings
- PCA dimensionality reduction

Eventually, the Min-Max scaling and 2D Hexbin projection showed the first confirmations of differences in the data (this can be seen by running the doDatasetInvestigation function). The positive class formed 1 cluster, which is expected because it was collected from r/ptsd alone. The negative class formed ~8 clusters, representing the 8 various subreddits that it was collected from.

```
In [25]: '''
#Instantiate PCA object
pca = PCA(n_components=2)

#Fit PCA to data
pca.fit(X_Embeddings)

# Transform data
transformed_data = pca.transform(X_Embeddings)

#Instantiate PCA object
pcaNeg = PCA(n_components=2)

#Fit PCA to data
pcaNeg.fit(X_Embeddings_Neg)

# Transform data
transformed_data_Neg = pcaNeg.transform(X_Embeddings_Neg)

#Calculate mean and standard deviation across all embeddings
meanEmbeddings = np.mean(np.array(X_Embeddings), axis=0)
stdEmbeddings = np.std(np.array(X_Embeddings), axis=0)

#Standardize the embeddings
standardizedEmbeddings = (np.array(X_Embeddings) - meanEmbeddings) / stdEmbeddings

posNP = np.array(X_Embeddings)
posMagnitudes = np.linalg.norm(posNP, axis=1, keepdims=True) #Compute the magnitude of each vector (L2 norm)
posUnitVectors = posNP / posMagnitudes

#Calculate mean and standard deviation across all embeddings
meanEmbeddingsNeg = np.mean(np.array(X_Embeddings_Neg), axis=0)
stdEmbeddingsNeg = np.std(np.array(X_Embeddings_Neg), axis=0)

#Standardize the embeddings
standardizedEmbeddingsNeg = (np.array(X_Embeddings_Neg) - meanEmbeddingsNeg) / stdEmbeddingsNeg

negNP = np.array(X_Embeddings_Neg)
negMagnitudes = np.linalg.norm(negNP, axis=1, keepdims=True) #Compute the magnitude of each vector (L2 norm)
```

```

negUnitVectors = negNP / negMagnitudes

cosineSim = cosine_similarity(standardizedEmbeddings) #Compute cosine similarity matrix
averageSimilarity = np.mean(cosineSim) #Compute the average cosine similarity

cosineSimNeg = cosine_similarity(standardizedEmbeddingsNeg) #Compute cosine similarity matrix
averageSimilarityNeg = np.mean(cosineSimNeg) #Compute the average cosine similarity

averageSimilarity, averageSimilarityNeg

cosineSim = cosine_similarity(transformed_data) #Compute cosine similarity matrix
averageSimilarity = np.mean(cosineSim) #Compute the average cosine similarity

cosineSimNeg = cosine_similarity(transformed_data_Neg) #Compute cosine similarity matrix
averageSimilarityNeg = np.mean(cosineSimNeg) #Compute the average cosine similarity

averageSimilarity, averageSimilarityNeg

cosineSimsForEachBERT = np.array([.679285, .99204206, .78988516, .9190559])
np.mean(cosineSimsForEachBERT)

#Compute pairwise cosine similarity between positive and negative sections of the dataset
similarities = 1 - cdist(standardizedEmbeddings, standardizedEmbeddingsNeg, metric='cosine')
similarities

average_similarity = np.mean(similarities)
average_similarity

cosineSimsForEachBERT_CDIST = np.array([.65682, .99201, .78425, .91553])
np.mean(cosineSimsForEachBERT_CDIST)
'''
print()

```