I use the data of Stress.csv from Kaggle.com. https://www.kaggle.com/datasets/kreeshrajani/human-stress-prediction

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, log_loss
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
```

```
from google.colab import files
#Uploading data
uploaded = files.upload()
```

Choose Files Stress.csv

- **Stress.csv**(text/csv) - 1417803 bytes, last modified: 3/30/2023 - 100% done

```
Saving Stress.csv to Stress (1).csv
```

```
import pandas as pd
# Reading in the CSV File
df = pd.read_csv('Stress.csv', encoding='latin-1')
df = df[['subreddit','text']]
df.text = df.text.astype(str)
display(df)
print(df['subreddit'].value_counts().plot(kind='bar'))
```

	subreddit	text
0	ptsd	He said he had not felt that way before, sugge...
1	assistance	Hey there r/assistance, Not sure if this is th...
2	ptsd	My mom then hit me with the newspaper and it s...
3	relationships	until i met my new boyfriend, he is amazing, h...
4	survivorsofabuse	October is Domestic Violence Awareness Month a...

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models, preprocessing

from sklearn.preprocessing import LabelEncoder
import numpy as np

# Seed
np.random.seed(1234)
```

```
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("Train data size: ", train.shape)
print("Test data size: ", test.shape)
```

```
Train data size:  (2252, 2)
Test data size:  (586, 2)
```

```
# Training data with Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train.text)

x_train = tokenizer.texts_to_matrix(train.text, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.text, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.subreddit)
y_train = encoder.transform(train.subreddit)
y_test = encoder.transform(test.subreddit)

y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
```

```
train shapes: (2252, 10863) (2252, 10)
test shapes: (586, 10863) (586, 10)
```

```
# CNN
CNN = models.Sequential()
CNN.add(layers.Embedding(400, 32, input_length=10863))
CNN.add(layers.Conv1D(64, 10, activation='relu'))
CNN.add(layers.MaxPooling1D(5))
CNN.add(layers.Conv1D(32, 10, activation='relu'))
CNN.add(layers.GlobalMaxPooling1D())
CNN.add(layers.Dense(10))

CNN.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = CNN.fit(x_train,
                  y_train,
                  epochs=5,
                  batch_size=32,
                  validation_split=0.2)
```

```
Epoch 1/5
57/57 [==============================] - 121s 2s/step - loss: 6.5869 - accuracy: 0.0278 - val_loss: 6.8261 - val_accuracy: 0.0155
Epoch 2/5
57/57 [==============================] - 114s 2s/step - loss: 6.5153 - accuracy: 0.0255 - val_loss: 6.8261 - val_accuracy: 0.0155
Epoch 3/5
57/57 [==============================] - 111s 2s/step - loss: 6.5153 - accuracy: 0.0255 - val_loss: 6.8261 - val_accuracy: 0.0155
```

```
Epoch 4/5
57/57 [==============================] - 111s 2s/step - loss: 6.5153 - accuracy: 0.0255 - val_loss: 6.8261 - val_accuracy: 0.0155
Epoch 5/5
57/57 [==============================] - 113s 2s/step - loss: 6.5153 - accuracy: 0.0255 - val_loss: 6.8261 - val_accuracy: 0.0155
```

```
result = CNN.evaluate(x_test, y_test, verbose=1)
print('Accuracy is ', result[1])
```

```
19/19 [==============================] - 11s 593ms/step - loss: 6.1062 - accuracy: 0.0188
Accuracy is  0.018771331757307053
```

```
# Embedded
embed = models.Sequential()
embed.add(layers.Embedding(400, 16, input_length=10863))
embed.add(layers.Flatten())
embed.add(layers.Dense(20, activation='relu'))
embed.add(layers.Dense(10, activation='softmax'))

embed.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
embed.summary()

history = embed.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 10863, 16)         6400

 flatten_3 (Flatten)         (None, 173808)            0

 dense_34 (Dense)            (None, 20)                3476180

 dense_35 (Dense)            (None, 10)                210

=================================================================
Total params: 3,482,790
Trainable params: 3,482,790
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
57/57 [==============================] - 7s 113ms/step - loss: 2.4787 - accuracy: 0.1116 - val_loss: 2.2081 - val_accuracy: 0.1131
Epoch 2/10
57/57 [==============================] - 7s 129ms/step - loss: 2.1879 - accuracy: 0.1771 - val_loss: 2.1979 - val_accuracy: 0.1729
Epoch 3/10
57/57 [==============================] - 9s 151ms/step - loss: 2.1474 - accuracy: 0.1938 - val_loss: 2.1378 - val_accuracy: 0.1729
Epoch 4/10
57/57 [==============================] - 7s 127ms/step - loss: 2.1024 - accuracy: 0.1943 - val_loss: 2.1044 - val_accuracy: 0.1729
Epoch 5/10
57/57 [==============================] - 8s 147ms/step - loss: 2.0378 - accuracy: 0.1938 - val_loss: 2.0651 - val_accuracy: 0.1729
Epoch 6/10
57/57 [==============================] - 11s 200ms/step - loss: 1.9709 - accuracy: 0.1999 - val_loss: 2.0342 - val_accuracy: 0.1885
Epoch 7/10
57/57 [==============================] - 6s 110ms/step - loss: 1.9016 - accuracy: 0.2066 - val_loss: 2.0202 - val_accuracy: 0.1863
Epoch 8/10
57/57 [==============================] - 5s 86ms/step - loss: 1.8360 - accuracy: 0.2082 - val_loss: 2.0073 - val_accuracy: 0.1996
Epoch 9/10
57/57 [==============================] - 5s 86ms/step - loss: 1.7814 - accuracy: 0.2088 - val_loss: 2.0029 - val_accuracy: 0.2151
Epoch 10/10
57/57 [==============================] - 6s 103ms/step - loss: 1.7335 - accuracy: 0.2127 - val_loss: 1.9997 - val_accuracy: 0.2528
```

```
result = embed.evaluate(x_test, y_test, verbose=1)
print('Accuracy is ', result[1])
```

```
19/19 [==============================] - 0s 19ms/step - loss: 1.9004 - accuracy: 0.2799
Accuracy is  0.27986347675323486
```

```
# Logistic Regression
X = df.text
y = df.subreddit

# divide into train and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# vectorizer
vector = TfidfVectorizer(binary=True)
X_train = vector.fit_transform(X_train)  # fit and transform the train data
X_test = vector.transform(X_test)        # transform only the test data

#train
LRC = LogisticRegression(solver='lbfgs', class_weight='balanced')
LRC.fit(X_train, y_train)
```

```
Exception ignored in: <function _xla_gc_callback at 0x7f93331c45e0>
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/jax/_src/lib/__init__.py", line 97, in _
    def _xla_gc_callback(*args):
KeyboardInterrupt:
```

```
LogisticRegression
LogisticRegression(class_weight='balanced')
```

```
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('log loss: ', log_loss(y_test, probs))
```

```
accuracy score:  0.5193661971830986
log loss:  1.7238526562227248
```

After completing all of the approaches, I see that the logistic Regression of accuracy of 0.5 has the highest accuracy out of CNN and embedded approaches. The CNN has the lowest accuracy from the 3 approaches.