Portfolio Assignment: Text Classification 1

```
import pandas
import sklearn
import seaborn as sb
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
import math
```

The dataset I am using are coronavirus tweets and the model should be able to detect if a tweet's sentiment is extremely postiive, extremely negative, or in between.

```
# Read in data from csv files

df_test = pandas.read_csv('corona_test.csv', header=0, usecols=[4,5], encoding='latin-1')
df_train = pandas.read_csv('corona_train.csv', header=0, usecols=[4,5], encoding='latin-1')[:10000]

df = pandas.concat([df_test, df_train])

display(df)
```

	OriginalTweet	Sentiment
0	TRENDING: New Yorkers encounter empty supermar...	Extremely Negative
1	When I couldn't find hand sanitizer at Fred Me...	Positive
2	Find out how you can protect yourself and love...	Extremely Positive
3	#Panic buying hits #NewYork City as anxious sh...	Negative
4	#toiletpaper #dunnypaper #coronavirus #coronav...	Neutral
...	...	...
9995	Popped out for food in #oldham \r\r\nPolite , ...	Positive
9996	Found my first paper towel in weeks at a super...	Positive
9997	Through the #Coronavirus chaos, IÂ m grateful ...	Negative
9998	Therapist, Lisa Olivera gave gratitude cards t...	Positive
9999	Older people and those with other conditions, ...	Extremely Negative

13798 rows × 2 columns

Display graph showing distributions of target classes (Counts of tweets of different sentiment)

```
import seaborn as sb
sb.displot(x = 'Sentiment', data = df, aspect = 16/10)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fda09be6790>
```

Text Processing and Vectorizing

```
X_train = df_train['OriginalTweet']
Y_train = df_train['Sentiment']
X_test = df_test['OriginalTweet']
Y_test = df_test['Sentiment']

from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))

vectorizer = TfidfVectorizer(stop_words = list(stopwords))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

#print vectorized train and test
print(X_train.shape)
print(X_test.shape)
```

```
(10000, 27576)
(3798, 27576)
```

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, Y_train)

prediction = naive_bayes.predict(X_test)
print('Accuracy: ', accuracy_score(Y_test, prediction))
```

```
Accuracy:  0.344391785150079
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Initialize logistic regression mode
lr = LogisticRegression()

# Fit the model to the training data

lr.fit(X_train, Y_train)
#Predict labels for the test data
y_pred = lr.predict(X_test)

# Calculate accuracy score of the model on the test data
accuracy = accuracy_score(Y_test, y_pred)

print('Accuracy: ', accuracy)
```

```
Accuracy:  0.4615587151132175
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (s
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Neural Network

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

nn = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=10000)

# Fit the model to the training data
nn.fit(X_train, Y_train)

# Predict labels for the test data
y_pred = nn.predict(X_test)

    # Calculate accuracy score of the model on the test data
accuracy = accuracy_score(Y_test, y_pred)

print('Accuracy: ', accuracy)
```

```
Accuracy:  0.43364928909952605
```

Logistic Regression achieved the highest accuracy of 46%, a little over Neural Network classification approach by 3%. I had anticipated Naive Bayes to return a higher accuracy than 34% since the dataset is small.