CISC/CMPE 452/COGS 400 Assignment 1 - Perceptron (10 points)

Please put your name and student id here

```
FirstName LastName, #12345678
```

- The notebook file has clearly marked blocks where you are expected to write code. Do not write or modify any code outside of these blocks.
- Make sure to restart and run all the cells from the beginning before submission. Do not clear out the outputs.
- Mark will be deducted based on late policy (-1% of the course total marks per day after due date until the end date after which no assignments will be accepted)

Build Model (6 points)

Implement **Simple Feedback Learning** for emotion classification (dataset from: https://www.kaggle.com/praveengovi/emotions-dataset-for-nlp (https://www.kaggle.com/praveengovi/emotions-dataset-for-nlp))

Use the correct/incorrect feedback and info about (y>d) or (y<d) to change weights.
Refer to the **Perceptron slides**

- 1. Implement forward and calculate the output (2 point)
- 2. Update the weights and bias (2 points)
- 3. Predict function (1 point)
- 4. Activation function (1 point)

Evaluator Function (2 point)

Implement the evaluator function with Pytorch or Numpy only

- Evaluation metrics include confusion matrix, accuracy, recall score, precision and F1 score

Train and Evaluate the Model (2 point)

Train the model with customized learning rate and number of iterations
Use the predict function to predict the labels with the test dataset
Evaluate the prediction results

- Evaluation metrics include confusion matrix, accuracy, recall score, precision and F1 score

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
# load dataset
df_train = pd.read_csv('data/train.txt', names=['Text', 'Emotion'], se
df_test = pd.read_csv('data/test.txt', names=['Text', 'Emotion'], sep=
```

In [ ]:
```
x_train = df_train['Text']
y_train = df_train['Emotion']

x_test = df_test['Text']
y_test = df_test['Emotion']

df_train.head()
```

In [ ]:
```
df_train.Emotion.value_counts()
```

Data Preprocessing

In [ ]:
```
# encode label
y_train = y_train.replace({'joy':1, 'sadness':0, 'anger':0, 'fear':0,
y_test = y_test.replace({'joy':1, 'sadness':0, 'anger':0, 'fear':0, 'l
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5)#, stop_words='eng

# We transform each text into a vector
x_train = tfidf.fit_transform(x_train).toarray()
x_test = tfidf.transform(x_test).toarray()
```

In [ ]:
```
def evaluator(y_test, y_pred):
    ##################################################################
    # enter code here to implement the evaluation matrices including c
    # DO NOT use any python packages such as scikit-learn


    ##################################################################
```

In [ ]:
```
class SimpleFeedbackLearning(object):
    def __init__(self):
        self.history = {}
        self.history['train_acc'] = []
        self.history['test_acc'] = []

    def f(self, x):
        ##################################################################
        # 4. enter code here to implement the activation function


        ##################################################################
        return fx

    def train(self, x, y, x_test, y_test, learning_rate=0.1, n_iters=1
```

```
    def train(self, x, y, x_test, y_test, learning_rate=0.1, n_iters=1
        n_train, input_size = x.shape
        n_test = x_test.shape[0]
        # weight initialization
        self.W = np.zeros(input_size)
        self.b = np.zeros(1)

        for i in range(n_iters):
            for xi, yi in zip(x, y):
                # forward
                ###########################################################
                # 1. enter code here to calculate the output


                ###########################################################

                ###########################################################
                # 2. enter code here to adjust the weights and bias


                ###########################################################

            train_acc = (self.predict(x) == y).sum() / n_train
            test_acc = (self.predict(x_test) == y_test).sum() / n_test
            self.history['train_acc'].append(train_acc)
            self.history['test_acc'].append(test_acc)
            if verbose:
                print('epoch %d, train acc %.4f, test acc %.4f' % (i +

    def predict(self, x):
        ###############################################################
        # 3. enter code here to complete the predict function
        # TODO: use the trained weights to predict labels and return t


        ###############################################################
        return y_pred
```

In [ ]:

```
#####################################################################
# enter code here to initialize and train the model


#####################################################################
```

In [ ]:
```
# plot the accuracy
plt.plot(model1.history['train_acc'], label='train_acc')
plt.plot(model1.history['test_acc'], label='test_acc')
plt.legend()
plt.show()
```

In [ ]:
```
#####################################################################
# enter code here to evaluate the model with the evaluator function


#####################################################################
```