project

June 15, 2021

0.0.1 Retrieve Frequency Domain and Time Domain Data

[1]:
```
def readFile(file):
    """
    Reads a file in memory

    Arguments:
        file (string): file to be read in

    Returns:
        Python list containing file data points
    """
    data = []

    infile = open(file,"r")
    s = infile.read()

    numbers = [eval(x) for x in s.split()]
    for number in numbers:
        data.append(number)

    infile.close() # Close the file
    return data
```

Training Set

[2]:
```
# Set 1
music_nv_1_time = readFile('Data/Training/Set 1/y1.txt')

music_v_1_time = readFile('Data/Training/Set 1/y2.txt')

speech_1_time = readFile('Data/Training/Set 1/y3.txt')

speech_bkg_music_1_time = readFile('Data/Training/Set 1/y4.txt')

# Set 2
music_nv_2_time = readFile('Data/Training/Set 2/y1.txt')

music_v_2_time = readFile('Data/Training/Set 2/y2.txt')
```

```
speech_2_time = readFile('Data/Training/Set 2/y3.txt')

speech_bkg_music_2_time = readFile('Data/Training/Set 2/y4.txt')
```

Testing Set

[3]:
```
# Set 3
music_nv_3_time = readFile('Data/Testing/Set 3/y1.txt')

music_v_3_time = readFile('Data/Testing/Set 3/y2.txt')

speech_3_time = readFile('Data/Testing/Set 3/y3.txt')

speech_bkg_music_3_time = readFile('Data/Testing/Set 3/y4.txt')

# Set 4
music_nv_4_time = readFile('Data/Testing/Set 4/y1.txt')

music_v_4_time = readFile('Data/Testing/Set 4/y2.txt')

speech_4_time = readFile('Data/Testing/Set 4/y3.txt')

speech_bkg_music_4_time = readFile('Data/Testing/Set 4/y4.txt')
```

0.0.2 Feature Selection

0.0.3 Calculate Time Domain Signal Mean Value (Weighted)

[4]:
```
def meanValue(aList):
    """
    Calculates the mean value of the absolute signal

    Arguments:
        aList (array): 1-D array containing values to be averaged

    Returns:
        Average of values
    """
    listSum = 0
    count = 0
    for value in aList:
        listSum = listSum + abs(value)

    # The sample frequency is 22050 Hz
    # Therefore, the data points in a 5 s recording are 110250
    mean = listSum / 110250
```

```
    return mean
```

Training Set

[5]:
```
# Set 1
music_nv_1_time_mean = meanValue(music_nv_1_time)

music_v_1_time_mean = meanValue(music_v_1_time)

speech_1_time_mean = meanValue(speech_1_time)

speech_bkg_music_1_time_mean = meanValue(speech_bkg_music_1_time)

# Set 2
music_nv_2_time_mean = meanValue(music_nv_2_time)

music_v_2_time_mean = meanValue(music_v_2_time)

speech_2_time_mean = meanValue(speech_2_time)

speech_bkg_music_2_time_mean = meanValue(speech_bkg_music_2_time)
```

Testing Set

[6]:
```
# Set 3
music_nv_3_time_mean = meanValue(music_nv_3_time)

music_v_3_time_mean = meanValue(music_v_3_time)

speech_3_time_mean = meanValue(speech_3_time)

speech_bkg_music_3_time_mean = meanValue(speech_bkg_music_3_time)

# Set 4
music_nv_4_time_mean = meanValue(music_nv_4_time)

music_v_4_time_mean = meanValue(music_v_4_time)

speech_4_time_mean = meanValue(speech_4_time)

speech_bkg_music_4_time_mean = meanValue(speech_bkg_music_4_time)
```

0.0.4 Calculate Percentage of Signal Below Mean [Feature #2]

[7]:
```
def maxCount(aList, mean, scaling_factor):
    """
    Calculates the maximum number of consecutive low amplitude points
```

```
    Arguments:
        aList (array): 1-D array containing values to be iterated
        mean (float): The calculated absolute value average
        scaling_factor (float): Calculates 'null point' relative to signal mean

    Returns:
        Number of low amplitude (null) points
    """
    count = 0
    max_count = 0

    for i in range(len(aList)-1):
        if abs(aList[i]) < (mean*scaling_factor) and abs(aList[i+1]) <␣
↪(mean*scaling_factor):
            count = count + 1
        else:
            count = 0

        if count > max_count:
            max_count = count


    return max_count
```

0.0.5 Scaling Factor of 0.5

Training Set

[8]:
```
# Set 1
music_nv_1_null_time_0_5 = maxCount(music_nv_1_time, music_nv_1_time_mean, 0.5)

music_v_1_null_time_0_5 = maxCount(music_v_1_time, music_v_1_time_mean, 0.5)

speech_1_null_time_0_5 = maxCount(speech_1_time, speech_1_time_mean, 0.5)

speech_bkg_music_1_null_time_0_5 = maxCount(speech_bkg_music_1_time,␣
↪speech_bkg_music_1_time_mean, 0.5)

# Set 2
music_nv_2_null_time_0_5 = maxCount(music_nv_2_time, music_nv_2_time_mean, 0.5)

music_v_2_null_time_0_5 = maxCount(music_v_2_time, music_v_2_time_mean, 0.5)

speech_2_null_time_0_5 = maxCount(speech_2_time, speech_2_time_mean, 0.5)

speech_bkg_music_2_null_time_0_5 = maxCount(speech_bkg_music_2_time,␣
↪speech_bkg_music_2_time_mean, 0.5)
```

Testing Set

[9]:
```
# Set 3
music_nv_3_null_time_0_5 = maxCount(music_nv_3_time, music_nv_3_time_mean, 0.5)

music_v_3_null_time_0_5 = maxCount(music_v_3_time, music_v_3_time_mean, 0.5)

speech_3_null_time_0_5 = maxCount(speech_3_time, speech_3_time_mean, 0.5)

speech_bkg_music_3_null_time_0_5 = maxCount(speech_bkg_music_3_time,␣
 ↪speech_bkg_music_3_time_mean, 0.5)

# Set 4
music_nv_4_null_time_0_5 = maxCount(music_nv_4_time, music_nv_4_time_mean, 0.5)

music_v_4_null_time_0_5 = maxCount(music_v_4_time, music_v_4_time_mean, 0.5)

speech_4_null_time_0_5 = maxCount(speech_4_time, speech_4_time_mean, 0.5)

speech_bkg_music_4_null_time_0_5 = maxCount(speech_bkg_music_4_time,␣
 ↪speech_bkg_music_4_time_mean, 0.5)
```

0.0.6 Data Partitioning

There are currently four sound categories we are working with: - music with no vocals - music with vocals - speech - speech with background music

At this point we have worked enough through the time domain statistics to gather a sufficient set of traits to run the system through. We will take the following strategy to pin point the exact class the test data points to.

Project: Diverge from the sound basis Is the sound speech-oriented or music-oriented?

The music-oriented category in this case consists of: - music with no vocals - music with vocals

The main message being transmitted is the rhythm, instrumentation, and/or vocalisation of the musical material.

The speech-oriented category in this case consists of: - speech - speech with background music

The main message being transmitted is the speaker's words.

[10]:
```
import numpy as np
```

Training Set

[11]:
```
# Using training set 1 and 2, populate the training arrays with % time below 0.
 ↪5 * mean

# ~ Class 0 = speech based
# ~ Class 1 = music based
```

```
speach_music_train = np.array([speech_1_null_time_0_5,
                               speech_2_null_time_0_5,
                               speech_bkg_music_1_null_time_0_5,
                               speech_bkg_music_2_null_time_0_5,
                               music_nv_1_null_time_0_5,
                               music_nv_2_null_time_0_5,
                               music_v_1_null_time_0_5,
                               music_v_2_null_time_0_5])

label_sm_train = np.array([0, 0, 0, 0, 1, 1, 1, 1])
```

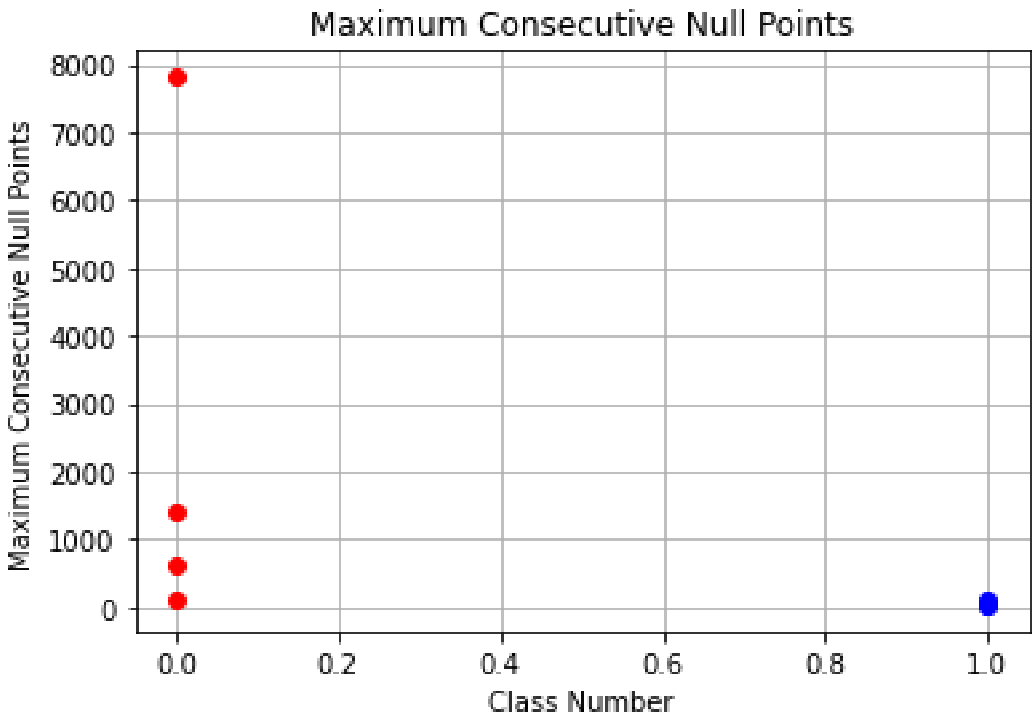
Testing Set

[12]:
```
# Using testing set 3 and 4, populate the testing arrays with % time below 0.5␣
↪* mean

# ~ Class 0 = speech
# ~ Class 1 = music

speach_music_test = np.array([speech_3_null_time_0_5,
                              speech_4_null_time_0_5,
                              speech_bkg_music_3_null_time_0_5,
                              speech_bkg_music_4_null_time_0_5,
                              music_nv_3_null_time_0_5,
                              music_nv_4_null_time_0_5,
                              music_v_3_null_time_0_5,
                              music_v_4_null_time_0_5])

label_sm_test = np.array([0, 0, 0, 0, 1, 1, 1, 1])
```

Data Visualisation

[13]:
```
import matplotlib.pyplot as plt
```

[14]:
```
plt.plot(label_sm_train[0:4], speach_music_train[0:4], 'ro')
plt.plot(label_sm_train[4:8], speach_music_train[4:8], 'bo')
plt.xlabel('Class Number')
plt.ylabel('Maximum Consecutive Null Points')
plt.title('Maximum Consecutive Null Points')
plt.grid(True)
plt.show()
```



The graphs above might be a little misleading. The quantity of interest is plotted along the **verical axis**, the horizontal axis merely benefits the reader by making the points more discernable.

The kNN algorithm will; therefore, be **adapted** to measure vertical distance between points as the distance metric - the horizontal (and hypotenuse) distance is irrelevant to our analysis.

0.0.7 Establish a Definition for our kNN algorithm

Distance Between Two Points

[15]:
```
def verticalDistance(y_a, y_b):
    """
    Calculates the vertical distance between two points (1 feature)

    Arguments:
        y_a (float): 1-D array containing values to be iterated
        y_b (float): The calculated absolute value average

    Returns:
        Distance between two points
    """
    return abs(y_a - y_b)
```

Calculate List of Distances

[16]:
```python
def listOfDistances(x_test, X_in):
    """
    Calculates a list of distances from a test point to each
    member of an input array

    Arguments:
        x_test (float): Test point
        X_in (array): 1-D array containing values to be iterated

    Returns:
        List of distances between test point and input array
    """
    distance_list = []

    length = len(X_in)

    while length > 0:
        distance_list.insert(0,verticalDistance(x_test, X_in[length-1]))
        length = length-1

    return distance_list
```

[17]:
```python
y_test = 1.0
Y_in = np.array([2.0, -1.5, -2, 0])
listOfDistances(y_test, Y_in)
```

[17]: [1.0, 2.5, 3.0, 1.0]

Determine k Nearest Neighbours

Sorting the distances and returning indices

[18]:
```python
def kNearestIndices(distance_list, k):
    """
    Calculates a list of the nearest indices to the test point

    Arguments:
        distance_list (array): 1-D array containing list of distances
        k (int): Number of nearest points desired

    Returns:
        List of the k-nearest indices to the test point
    """
    # Step 1: Create an empty numpy array
    arr_empty = np.array([])

    # Step 2: Append the distance_list python array
    arr_append = np.append (arr_empty, distance_list)
```

```
    # Step 3: Sort the array elements in ascending order
    arr_sort = np.argsort(arr_append)

    # Step 4: Retain first k elements
    k_nearest_indices = arr_sort[:k]

    return k_nearest_indices
```

[19]:
```
# Test Cell
print(kNearestIndices([5.0, 3.5, 2.5, 1.0], 3))
```

```
[3 2 1]
```

Returning the Values at the K-Nearest Indeces

[20]:
```
def kNearestNeighbours(k_nearest_indices, X_in, Y_in):
    """
    Calculates a list of the nearest neighbours to the test point

    Arguments:
        k_nearest_indices (array): 1-D array containing list of distances
        X_in (array): 1-D array containing data array points
        Y_in (array): 1-D array containing label points

    Returns:
        List of the k-nearest neighbours to the test point
    """
    # Step 1: Create two Python lists
    X = []
    Y = []

    # Step 2: Identify how many nearest neighbours
    #         are to be used
    length = len(k_nearest_indices)

    # Step 3: Place the nearest neighbour in the created
    #         Python list, head first
    while length > 0:
        X.insert(0, X_in[k_nearest_indices[length - 1]])
        Y.insert(0, Y_in[k_nearest_indices[length - 1]])

        length = length - 1

    # Step 4: Use the Python lists to create ceooresponding
    #.        numpy arrays
    X_k = np.array (X)
    Y_k = np.array (Y)
```

```python
    return X_k, Y_k
```

[21]:
```python
# Test Cell
X_train_grade = np.array([1, 2, 3, 4, 5, 6, 7, 8])

Y_train_grade = np.array([0, 0, 0, 0, 1, 1, 1, 0])

X_k_grade, Y_k_grade = kNearestNeighbours([0, 1, 3], X_train_grade,␣
 ↪Y_train_grade)

print(X_k_grade)
print(Y_k_grade)
```

```
[1 2 4]
[0 0 0]
```

[22]:
```python
from scipy import stats
```

Predict Category

[23]:
```python
def predict(x_test, X_in, Y_in, k):
    """
    Determine the most likely identity of the test point

    Arguments:
        x_test (array): 1-D array containing test point
        X_in (array): 1-D array containing data array points
        Y_in (array): 1-D array containing label points
        k (int): Number of nearest points desired

    Returns:
        Most likely identity of test point
    """
    # Step 1: Task 2 - Calculate list of distances
    distance_list = []
    distance_list = listOfDistances(x_test, X_in)
    # print(distance_list)

    # Step 2: Task 3.1 - Determine k Nearest Indices
    k_nearest_indices = kNearestIndices(distance_list, k)
    # print(k_nearest_indices)

    # Step 3: Task 3.2 - Determine k Nearest Neighbours
    X_k, Y_k = kNearestNeighbours(k_nearest_indices, X_in, Y_in)
    # print(Y_k)

    # Step 4: Determine mode of classes
```

```python
    prediction = np.array(stats.mode(Y_k))
    # print(prediction)

    # Step 5: Return only the class member
    return prediction[0]
```

[24]:
```python
# Test Cell
X_train_grade = np.array([1, 2, 3, 4, 5, 6, 7, 8])

Y_train_grade = np.array([0, 0, 0, 0, 1, 1, 1, 0])

x1_grade = np.array([0])
k_grade = 3
print(predict(x1_grade, X_train_grade, Y_train_grade, k_grade))
```

```
[0]
```

Predict for an entire batch of test examples

[25]:
```python
def predictBatch(X_t, X_in, Y_in, k):
    """
    Determine the most likely identities of the test point list

    Arguments:
        X_t (array): 1-D array containing test point list
        X_in (array): 1-D array containing data array points
        Y_in (array): 1-D array containing label points
        k (int): Number of nearest points desired

    Returns:
        Most likely identity of test point list values
    """
    length = len(X_t)

    predictions_py = []

    while (length > 0):
        # Step 1: Identify the prediction for the last index
        prediction = predict(X_t[length-1], X_in, Y_in, k)

        # Step 2: Concatenate the index to the head of the list
        predictions_py = [*prediction, *predictions_py]

        length = length - 1

    predictions = np.array(predictions_py)

    return predictions
```

[26]:
```python
# Test Cell
X_train_grade = np.array([1, 2, 3, 4, 15, 16, 17, 5])

Y_train_grade = np.array([0, 0, 0, 0, 1, 1, 1, 0])

X_test_grade = np.array([1, 2, 3, 14])

Y_test_grade = np.array([0, 0, 0, 1])

k_grade=1

print(predictBatch(X_test_grade, X_train_grade, Y_train_grade, k=k_grade))
```

```
[0 0 0 1]
```

Accuracy Metric

[27]:
```python
def accuracy(Y_pred, Y_test):
    """
    Determine the accuracy of the prediction with respect
    to the true value (label)

    Arguments:
        Y_pred (array): 1-D array containing prediction point list
        Y_test (array): 1-D array containing label point list

    Returns:
        Accuracy of prediction
    """
    length = len(Y_pred)
    correct = 0

    while length > 0:

        # Step 1: Identify if the array alements are equal
        if np.equal(Y_pred[length - 1], Y_test[length - 1]):

            # Step 2: If equal increment correct
            correct = correct + 1

        length = length - 1

    accuracy = correct / len(Y_pred)

    return accuracy
```

[28]:
```python
# Test Cell
Y_test_grade = np.array([0, 0, 0, 1])
```

```
Y_pred_grade = np.array([0, 0, 0, 1])

print(accuracy(Y_pred_grade, Y_test_grade))
```

```
1.0
```

Single Method Implementation

[29]:
```
def run(X_train, X_test, Y_train, Y_test, k):
    """
    Defines the kNN algortithm in one method

    Arguments:
        X_train (array): 1-D array containing training data points
        X_test (array): 1-D array containing testing data points
        Y_train (array): 1-D array containing training label points
        Y_test (array): 1-D array containing testing label points
        k (int): Number of nearest points desired

    Returns:
        Accuracy of prediction
    """
    # Step 1: Task 5 - Predict for an entire batch of test examples
    Y_pred = predictBatch(X_test, X_train, Y_train, k)

    # Step 2: Task 6 - Accuracy metric
    test_accuracy = accuracy(Y_pred, Y_test)

    return test_accuracy
```

[30]:
```
# Test Cell
X_train_grade = np.array([1, 2, 3, 4, 15, 16, 17, 5])
Y_train_grade = np.array([0, 0, 0, 0, 1, 1, 1, 0])

X_test_grade = np.array([1, 2, 3, 14])
Y_test_grade = np.array([0, 0, 0, 1])
k_grade=3

print("Accuracy Test:")
print(f'{run(X_train_grade, X_test_grade, Y_train_grade, Y_test_grade,␣
 ↪k_grade)}')
```

```
Accuracy Test:
1.0
```

0.1 Project: Diverge from the sound basis

[31]:
```
k = 1
print("kNN algorithm accuracy:")
print(f'{run(speach_music_train, speach_music_test, label_sm_train,
 ↪label_sm_test, k)}')
```

```
kNN algorithm accuracy:
0.875
```

Thank you for reading through to the end of this notebook

0.1.1 Fin

[ ]:
```

```