README:
The research paper is:
The presentation link is:
Documents submitted:

1. HL-KNN Modified report
2. HL-KNN Modified ppt
3. JuliaUmmaReddy_HLKNN.mp4 (https://drive.google.com/file/d/1j9pDrjTUn0aJtD8K7xb7eaHBoMgReNt_/view?usp=drive_link)
4. Jupyter notebook files:
    1. HLKNN_Iris_Baseline.ipynb
    2. HLKNN_Iris_Version1.ipynb
    3. HLKNN_Iris_Version2.ipynb
    4. HLKNN_ILPD_Baseline.ipynb
    5. HLKNN_ILPD_Version1.ipynb
    6. HLKNN_ILPD_Version2.ipynb
    7. HLKNN_Diabetes_Baseline.ipynb
    8. HLKNN_Diabetes_Version1.ipynb
    9. HLKNN_Diabetes_Version2.ipynb
    10. HLKNN_BankNoteAuthentication_Baseline.ipynb
    11. HLKNN_BankNoteAuthentication_Version1.ipynb
    12. HLKNN_BankNoteAuthentication_Version2.ipynb
    13. HLKNN_Beans_Version2.ipynb

Baseline is the HL-KNN algorithm
Version1 is the first improvement that I proposed.
Version2 is the second improvement that I proposed.

5. Datasets:
    1. iris.csv
    2. Indian Liver Patient Dataset (ILPD).csv
    3. NannaGift_transactionCode.csv (this is the diabetes dataset)
    4. data_banknote_authentication.txt
    5. Dry_Bean_Dataset2.csv

SCREENSHOTS:(Only for reference)
Below are the screenshots of all the jupyter files and recorded results

1. Dataset “iris.csv”

a. HL-KNN algorithm(Baseline)
   Accuracy: 100.0%
   Time taken: 11sec
b. Version 1 - No highlevel neighbors if all low-level are of same class
   Accuracy: 100.0%
   Time taken: 10.845sec

```python
def func_one (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    result = pd.DataFrame()
    result['Outcome'] = y_train
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-2]
        dist_arr = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr
        X_train['variety']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        #print(df_top7)
        arr1.append(df_top7['variety'].tolist())

        X_train = X_train.drop('variety',axis=1)
        if all(x == 1 for x in arr1):
            predict_arr.append(1)
        elif all(x == 0 for x in arr1):
            predict_arr.append(0)
        else:
            #count_outcome_0 = df_top7['Outcome'].eq(0).sum()
            #count_outcome_1 = df_top7['Outcome'].eq(1).sum()
            for j in range(0,len(df_top7)):
                top7 = df_top7.iloc[j,:-2]
               # print(top7)
                dist_arr1 = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr1
                X_train['variety']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
             #   print(df1_sorted)
                df1_top7 = df1_sorted.head(7)
                arr1.append(df1_top7['variety'].tolist())
                X_train = X_train.drop('variety',axis=1)
            #print(arr1)
```

```
        count_per_inner_list = [inner_list.count(1) for inner_list in arr1]
        total_count1 = sum(count_per_inner_list)
        count_per_inner_list = [inner_list.count(2) for inner_list in arr1]
        total_count2 = sum(count_per_inner_list)
        count_per_inner_list = [inner_list.count(3) for inner_list in arr1]
        total_count3 = sum(count_per_inner_list)
        #print(total_count3)

        if total_count1 > total_count2 and total_count1 > total_count3:
            predict_arr.append(1)
        elif total_count2 > total_count1 and total_count2 > total_count3:
            predict_arr.append(2)
        elif total_count3 > total_count1 and total_count3 > total_count2:
            predict_arr.append(3)
        elif total_count1 == total_count2 and total_count1 > total_count3:
            predict_arr.append(1)
        elif total_count1 == total_count3 and total_count1 > total_count2:
            predict_arr.append(1)
        elif total_count2 == total_count1 and total_count2 > total_count3:
            predict_arr.append(2)
        elif total_count2 == total_count3 and total_count2 > total_count1:
            predict_arr.append(2)
        elif total_count3 == total_count1 and total_count3 > total_count2:
            predict_arr.append(3)
        elif total_count3 == total_count2 and total_count3 > total_count1:
            predict_arr.append(3)
        elif total_count1 == total_count2 and total_count1 < total_count3:
            predict_arr.append(3)
        elif total_count1 == total_count3 and total_count1 < total_count2:
            predict_arr.append(2)
        elif total_count2 == total_count1 and total_count2 < total_count3:
            predict_arr.append(3)
        elif total_count2 == total_count3 and total_count2 < total_count1:
            predict_arr.append(1)
        elif total_count3 == total_count1 and total_count3 < total_count2:
            predict_arr.append(2)
        elif total_count3 == total_count2 and total_count3 < total_count1:
            predict_arr.append(1)
        else:
            predict_arr.append(3)
        arr1.clear()
```

```
        arr1.clear()


    #counts = [total_count3, total_count2, total_count1]
    #most_common_number = counts.index(max(counts)) + 1  # Adding 1 to get the actual number
    #predict_arr.append(most_common_number)

predicted_testdata['sepal.length'] = X_test['sepal.length']
predicted_testdata['petal.length'] = X_test['petal.length']
predicted_testdata['petal.width'] = X_test['petal.width']
predicted_testdata['variety'] = predict_arr

return predicted_testdata
```

In [527]:
```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = func_one(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

Out[527]:

73	6.1	4.7	1.2	2
146	6.3	5.0	1.9	3

In [528]:
```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['variety'] = y_test
```

```
Time taken: 10.845224857330322 seconds
```

In [531]:
```
match_percent = efficiency_test (predicted_test, X_test)
```

In [532]:
```
match_percent
```

Out[532]: 100.0

c. Version2: Only checking for neighbors of neighbors that are maximum in the first level
Accuracy: 100%
Time taken: 10.497sec

```
def calculate_euclidean_distance(training_features, test_features):
    dist_arr = []
    #filtered_training_features = training_features[training_features['Distance'] != -1].copy()
    for i in range(0,len(training_features)):

        tf = training_features.iloc[i,:-2]

        dist = np.sqrt(np.sum(np.square(test_features - tf)))
        dist_arr.append(dist)
    return dist_arr
```

```
def func_one (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    arr12 = []
    result = pd.DataFrame()
    result['Outcome'] = y_train
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-2]
        dist_arr = calculate_euclidean_distance(X_train, test)
        X_train = X_train[X_train['Distance'] != -1].copy()
        X_train['Distance']= dist_arr
        X_train['variety']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        #print(df_top7)
        arr1.append(df_top7['variety'].tolist())
        total1_count = calculate_sum(arr1)
        new_df = df_top7[df_top7['variety'] == total1_count].copy()

        X_train = X_train.drop('variety',axis=1)
        if all(x == 1 for x in arr1):
            predict_arr.append(1)
        elif all(x == 0 for x in arr1):
            predict_arr.append(0)
        else:
            for j in range(0,len(new_df)):
                top7 = new_df.iloc[j,:-2]
                dist_arr1 = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr1
                X_train['variety']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
                df1_top7 = df1_sorted.head(7)
                arr12.append(df1_top7['variety'].tolist())
                X_train = X_train.drop('variety',axis=1)
            total_count = calculate_sum(arr12)
            predict_arr.append(total_count)
            arr1.clear()
            arr12.clear()
```

```
            arr1.clear()
            arr12.clear()

        #if total1_count != total_count:
        #    X_train.loc[X_train['UniqueID'].isin(df_top7['UniqueID']) & (df_top7['variety'] == total1_count), 'Dist

    predicted_testdata['sepal.length'] = X_test['sepal.length']
    predicted_testdata['petal.length'] = X_test['petal.length']
    predicted_testdata['petal.width'] = X_test['petal.width']
    predicted_testdata['variety'] = predict_arr

    return predicted_testdata
```

```
#'efficiency_test' function calculates the accuracy of the prediction made in the function "func_one"
#Here I pass predicted test data and the test data and compare their 'Outcome' values. Calculate the percentage of r
#return the percentage.
```

```
def efficiency_test (predicted_testdata, X_test):
    counttt = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,5] == predicted_test.iloc[i,3]:
            counttt = counttt +1
    percentage = (counttt/len(X_test)) * 100
    return percentage
```

```
# Calling the 'func_one' function to calculate the distance of each test data from all the training data and print t
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = func_one(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['variety'] = y_test
```

```
Time taken: 10.496729135513306 seconds
```

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
100.0
```

2. Dataset: “Diabetes.csv”

a. HLKNN - accuracy is 77.92%
   time taken is 257.59sec

```
]: def calculate_euclidean_distance(training_features, test_features):
    dist_arr = []
    for i in range(0,len(training_features)):
        tf = training_features.iloc[i,:-2]
        dist = np.sqrt(np.sum(np.square(test_features - tf)))
        dist_arr.append(dist)
    return dist_arr
```

```
]: def func_one (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    result = pd.DataFrame()
    result['Outcome'] = y_train
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-2]
        dist_arr = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr
        X_train['Outcome']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        arr1.append(df_top7['Outcome'].tolist())
        X_train = X_train.drop('Outcome',axis=1)
        #if all(x == 1 for x in arr1):
        #    predict_arr.append(1)
        #elif all(x == 0 for x in arr1):
        #    predict_arr.append(0)
        #else:
        #    #count_outcome_0 = df_top7['Outcome'].eq(0).sum()
        #    #count_outcome_1 = df_top7['Outcome'].eq(1).sum()
        for j in range(0,len(df_top7)):
            top7 = df_top7.iloc[j,:-2]
            dist_arr1 = calculate_euclidean_distance(X_train, top7)
            X_train['Distance']= dist_arr1
            X_train['Outcome']= y_train
            df1 = pd.DataFrame(X_train)
            df1_sorted = df1.sort_values(by='Distance')
            #print(df1_sorted)
            df1_top7 = df1_sorted.head(7)
            arr1.append(df1_top7['Outcome'].tolist())
```

```
            X_train = X_train.drop('Outcome',axis=1)

        count_per_inner_list = [inner_list.count(0) for inner_list in arr1]
        total_count0 = sum(count_per_inner_list)
        count_per_inner_list = [inner_list.count(1) for inner_list in arr1]
        total_count1 = sum(count_per_inner_list)
        #arr1 = result.nsmallest(7,'Distance')
        #count1 = 0 if not (1 in arr1['Outcome'].values) else arr1['Outcome'].value_counts()[1]
        #count0 = 0 if not (0 in arr1['Outcome'].values) else arr1['Outcome'].value_counts()[0]
        if total_count1 > total_count0:
            predict_arr.append(1)
        elif total_count0 > total_count1:
            predict_arr.append(0)
        else:
            predict_arr.append(1)
        arr1.clear()
    predicted_testdata['Pregnancies'] = X_test['Pregnancies']
    predicted_testdata['Glucose'] = X_test['Glucose']
    predicted_testdata['BloodPressure'] = X_test['BloodPressure']
    predicted_testdata['SkinThickness'] = X_test['SkinThickness']
    predicted_testdata['Insulin'] = X_test['Insulin']
    predicted_testdata['BMI'] = X_test['BMI']
    predicted_testdata['Age'] = X_test['Age']
    #predicted_testdata['DiabetesPedigreeFunction'] = X_test['DiabetesPedigreeFunction']
    predicted_testdata['Outcome'] = predict_arr

    return predicted_testdata
```

```
def efficiency_test (predicted_testdata, X_test):
    counttt = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,10] == predicted_test.iloc[i,7]:
            counttt = counttt +1
    percentage = (counttt/len(X_test)) * 100
    return percentage
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = func_one(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['Outcome'] = y_test
```

```
Time taken: 257.5920810699463 seconds
```

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
77.92207792207793
```

b. Version1
   Accuracy is 77.92%
   Time taken is 231.69 sec
   Percentage improvement in time taken is 10.05%

```python
def calculate_euclidean_distance(training_features, test_features):
    dist_arr = []
    for i in range(0,len(training_features)):
        tf = training_features.iloc[i,:-2]
        dist = np.sqrt(np.sum(np.square(test_features - tf)))
        dist_arr.append(dist)
    return dist_arr
```

```python
def func_one (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    result = pd.DataFrame()
    result['Outcome'] = y_train
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-2]
        dist_arr = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr
        X_train['Outcome']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        arr1.append(df_top7['Outcome'].tolist())
        X_train = X_train.drop('Outcome',axis=1)
        if all(x == 1 for x in arr1):
            predict_arr.append(1)
        elif all(x == 0 for x in arr1):
            predict_arr.append(0)
        else:
            #count_outcome_0 = df_top7['Outcome'].eq(0).sum()
            #count_outcome_1 = df_top7['Outcome'].eq(1).sum()
            for j in range(0,len(df_top7)):
                top7 = df_top7.iloc[j,:-2]
                dist_arr1 = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr1
                X_train['Outcome']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
                #print(df1_sorted)
```

```
                df1_top7 = df1_sorted.head(7)
                arr1.append(df1_top7['Outcome'].tolist())
                X_train = X_train.drop('Outcome',axis=1)

            count_per_inner_list = [inner_list.count(0) for inner_list in arr1]
            total_count0 = sum(count_per_inner_list)
            count_per_inner_list = [inner_list.count(1) for inner_list in arr1]
            total_count1 = sum(count_per_inner_list)
            if total_count1 > total_count0:
                predict_arr.append(1)
            elif total_count0 > total_count1:
                predict_arr.append(0)
            else:
                predict_arr.append(1)
            arr1.clear()
    predicted_testdata['Pregnancies'] = X_test['Pregnancies']
    predicted_testdata['Glucose'] = X_test['Glucose']
    predicted_testdata['BloodPressure'] = X_test['BloodPressure']
    predicted_testdata['SkinThickness'] = X_test['SkinThickness']
    predicted_testdata['Insulin'] = X_test['Insulin']
    predicted_testdata['BMI'] = X_test['BMI']
    predicted_testdata['Age'] = X_test['Age']
    predicted_testdata['Outcome'] = predict_arr

    return predicted_testdata
```

```
: def efficiency_test (predicted_testdata, X_test):
    counttt = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,10] == predicted_test.iloc[i,7]:
            counttt = counttt +1
    percentage = (counttt/len(X_test)) * 100
    return percentage
```

```
: predicted_test = pd.DataFrame()
  start_time = time.time()
  predicted_test = func_one(X_train,X_test,y_train)
  end_time = time.time()
  predicted_test
```

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['Outcome'] = y_test
```

```
Time taken: 231.69658088684082 seconds
```

```
: match_percent = efficiency_test (predicted_test, X_test)
```

```
: match_percent
```

```
: 77.92207792207793
```

c. Version2– Only checking for neighbors of neighbors that are maximum in the first level
   Accuracy:75.9740%
   Time Taken: 195.356 sec
   Percentage Improvement in the time taken is 24.16%
   Percentage decrease in accuracy is 2.12%

```python
def calculate_euclidean_distance(training_features, test_features):
    dist_arr = []
    #filtered_training_features = training_features[training_features['Distance'] != -1].copy()
    #filtered_values = training_features.loc[training_features['Distance'] == -1, 'UniqueID']
    #print(filtered_values)
    #count_of_minus_1 = len(training_features[training_features['Distance'] == -1])
    #print(count_of_minus_1)
    for i in range(0,len(training_features)):
        tf = training_features.iloc[i,:-2]
        dist = np.sqrt(np.sum(np.square(test_features - tf)))
        dist_arr.append(dist)
    return dist_arr
```

```python
def calculate_sum(arr1):
    count_per_inner_list = [inner_list.count(0) for inner_list in arr1]
    total_count0 = sum(count_per_inner_list)
    count_per_inner_list = [inner_list.count(1) for inner_list in arr1]
    total_count1 = sum(count_per_inner_list)

    if total_count1 > total_count0:
        return 1
    elif total_count0 > total_count1:
        return 0
    else:
        return 1
```

```python
def func_one (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    arr12 = []
    result = pd.DataFrame()
    result['Outcome'] = y_train
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-2]
        dist_arr = calculate_euclidean_distance(X_train, test)
        #X_train = X_train[X_train['Distance'] != -1].copy()
        X_train['Distance']= dist_arr
        X_train['Outcome']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        arr1.append(df_top7['Outcome'].tolist())
        total1_count = calculate_sum(arr1)
        new_df = df_top7[df_top7['Outcome'] == total1_count].copy()

        X_train = X_train.drop('Outcome',axis=1)
        if all(x == 1 for x in arr1):
            predict_arr.append(1)
        elif all(x == 0 for x in arr1):
            predict_arr.append(0)
        else:
            for j in range(0,len(new_df)):
                top7 = new_df.iloc[j,:-2]
                dist_arr1 = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr1
                X_train['Outcome']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
                #print(df1_sorted)
                df1_top7 = df1_sorted.head(7)
                arr12.append(df1_top7['Outcome'].tolist())
                X_train = X_train.drop('Outcome',axis=1)
            total_count = calculate_sum(arr12)
            predict_arr.append(total_count)
            arr1.clear()
            arr12.clear()
```

```
            predict_arr.append(total_count)
            arr1.clear()
            arr12.clear()

        #if total1_count != total_count:
        #   X_train.loc[X_train['UniqueID'].isin(df_top7['UniqueID']) & (df_top7['Outcome'] == total1_count), 'Dista

    predicted_testdata['Pregnancies'] = X_test['Pregnancies']
    predicted_testdata['Glucose'] = X_test['Glucose']
    predicted_testdata['BloodPressure'] = X_test['BloodPressure']
    predicted_testdata['SkinThickness'] = X_test['SkinThickness']
    predicted_testdata['Insulin'] = X_test['Insulin']
    predicted_testdata['BMI'] = X_test['BMI']
    predicted_testdata['Age'] = X_test['Age']
    predicted_testdata['Outcome'] = predict_arr

    return predicted_testdata
```

```
def efficiency_test (predicted_testdata, X_test):
    counttt = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,10] == predicted_test.iloc[i,7]:
            counttt = counttt +1
    percentage = (counttt/len(X_test)) * 100
    return percentage
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = func_one(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Age	Outcome
285	7	136	74.000000	26.00000	135.000000	26.0	51	0
101	1	151	60.000000	29.15342	155.548223	26.1	22	1
581	6	109	60.000000	27.00000	155.548223	25.0	27	0
352	3	61	82.000000	28.00000	155.548223	34.4	46	0

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['Outcome'] = y_test
```

```
Time taken: 195.35651898384094 seconds
```

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
75.97402597402598
```

3. Dataset: “ILPD.csv”

a. HL-KNN
   Accuracy: 68.37%
   Time taken: 153.513 sec

```python
#This function is the main algorithm which implements HL-KNN algorithm(Baseline). Takes the training data,
#test data and predicts the class of the given test data.
#More description detailed in the report
def high_level_knn_modified (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    arr12 = []
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-1]
        dist_arr = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr
        X_train['1']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        arr1.append(df_top7['1'].tolist())
        total1_count = calculate_sum(arr1)

        X_train = X_train.drop('1',axis=1)
        for j in range(0,len(df_top7)):
            top7 = df_top7.iloc[j,:-1]
            dist_arr1 = calculate_euclidean_distance(X_train, top7)
            X_train['Distance']= dist_arr1
            X_train['1']= y_train
            df1 = pd.DataFrame(X_train)
            df1_sorted = df1.sort_values(by='Distance')
            df1_top7 = df1_sorted.head(7)
            arr1.append(df1_top7['1'].tolist())
            X_train = X_train.drop('1',axis=1)
        total_count = calculate_sum(arr1)
        predict_arr.append(total_count)
        arr1.clear()

    predicted_testdata['Female'] = X_test['Female']
    predicted_testdata['6.8'] = X_test['6.8']
    predicted_testdata['3.3'] = X_test['3.3']
    predicted_testdata['0.9'] = X_test['0.9']
    predicted_testdata['1'] = predict_arr

    return predicted_testdata
```

```python
#This function calculates the accuracy of the predicted data by dividing the correctly
#number of test samples multiplied by 100
def efficiency_test (predicted_testdata, X_test):
    counttt = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,5] == predicted_test.iloc[i,4]:
            counttt = counttt +1
    percentage = (counttt/len(X_test)) * 100
    return percentage
```

```python
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = high_level_knn_modified(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

	Female	6.8	3.3	0.9	1
457	1	3.6	0.9	0.30	1
225	1	8.0	4.0	1.00	1
162	1	7.4	4.3	1.38	1
195	1	6.0	2.1	0.53	1
267	1	5.7	2.1	0.50	1
...	...	...	...	...	...
464	2	5.1	2.0	0.50	1
528	1	5.1	1.8	0.50	1
349	1	5.7	2.9	1.00	1
273	1	8.0	4.0	1.00	1
5	2	7.0	3.5	1.00	1

117 rows × 5 columns

```python
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['variety'] = y_test
```

```
Time taken: 153.51389265060425 seconds
```

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['variety'] = y_test
```

```
Time taken: 153.51389265060425 seconds
```

```
X_test
```

	Female	6.8	3.3	0.9	Distance	variety
457	1	3.6	0.9	0.30	None	1
225	1	8.0	4.0	1.00	None	1
162	1	7.4	4.3	1.38	None	1
195	1	6.0	2.1	0.53	None	1
267	1	5.7	2.1	0.50	None	1
...	...	...	...	...	...	...
464	2	5.1	2.0	0.50	None	2
528	1	5.1	1.8	0.50	None	1
349	1	5.7	2.9	1.00	None	1
273	1	8.0	4.0	1.00	None	1
5	2	7.0	3.5	1.00	None	1

117 rows × 6 columns

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
68.37606837606837
```

b. Version1
Accuracy: 68.376%

Time Taken: 164.055sec

```
: def func_one (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    arr12 = []
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-1]
        dist_arr = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr
        X_train['1']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        #print(df_top7)
        arr1.append(df_top7['1'].tolist())
        total1_count = calculate_sum(arr1)
        #new_df = df_top7[df_top7['1'] == total1_count].copy()

        X_train = X_train.drop('1',axis=1)
        if all(x == 1 for x in arr1):
            predict_arr.append(1)
        elif all(x == 0 for x in arr1):
            predict_arr.append(0)
        else:
            for j in range(0,len(df_top7)):
                top7 = df_top7.iloc[j,:-1]
                dist_arr1 = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr1
                X_train['1']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
                df1_top7 = df1_sorted.head(7)
                arr1.append(df1_top7['1'].tolist())
                X_train = X_train.drop('1',axis=1)
            total_count = calculate_sum(arr1)
            predict_arr.append(total_count)
            arr1.clear()
            #arr12.clear()

    predicted_testdata['Female'] = X_test['Female']
    predicted_testdata['6.8'] = X_test['6.8']
    predicted_testdata['3.3'] = X_test['3.3']
```

```
                df1_sorted = df1.sort_values(by='Distance')
                df1_top7 = df1_sorted.head(7)
                arr1.append(df1_top7['1'].tolist())
                X_train = X_train.drop('1',axis=1)
            total_count = calculate_sum(arr1)
            predict_arr.append(total_count)
            arr1.clear()
            #arr12.clear()

    predicted_testdata['Female'] = X_test['Female']
    predicted_testdata['6.8'] = X_test['6.8']
    predicted_testdata['3.3'] = X_test['3.3']
    predicted_testdata['0.9'] = X_test['0.9']
    predicted_testdata['1'] = predict_arr

    return predicted_testdata
```

```
def efficiency_test (predicted_testdata, X_test):
    counttt = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,5] == predicted_test.iloc[i,4]:
            counttt = counttt +1
    percentage = (counttt/len(X_test)) * 100
    return percentage
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = func_one(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

	Female	6.8	3.3	0.9	1
457	1	3.6	0.9	0.30	1
225	1	8.0	4.0	1.00	1
162	1	7.4	4.3	1.38	1
195	1	6.0	2.1	0.53	1
267	1	5.7	2.1	0.50	1
...	...	...	...	...	...
464	2	5.1	2.0	0.50	1

273	1	8.0	4.0	1.00	1
5	2	7.0	3.5	1.00	1

117 rows × 5 columns

]:
```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['variety'] = y_test
```

```
Time taken: 164.05501127243042 seconds
```

]:
```
X_test
```

]:

	Female	6.8	3.3	0.9	Distance	variety
457	1	3.6	0.9	0.30	None	1
225	1	8.0	4.0	1.00	None	1
162	1	7.4	4.3	1.38	None	1
195	1	6.0	2.1	0.53	None	1
267	1	5.7	2.1	0.50	None	1
...	...	...	...	...	...	...
464	2	5.1	2.0	0.50	None	2
528	1	5.1	1.8	0.50	None	1
349	1	5.7	2.9	1.00	None	1
273	1	8.0	4.0	1.00	None	1
5	2	7.0	3.5	1.00	None	1

117 rows × 6 columns

]:
```
match_percent = efficiency_test (predicted_test, X_test)
```

]:
```
match_percent
```

]: 68.37606837606837

c. Version2
Accuracy:67.521%

Time taken: 114.372 sec

```
: def func_one (X_train,X_test,y_train):
    dist_arr = []
    predict_arr = []
    dist_arr1 = []
    arr1 = []
    arr12 = []
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-1]
        dist_arr = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr
        X_train['1']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        #print(df_top7)
        arr1.append(df_top7['1'].tolist())
        total1_count = calculate_sum(arr1)
        new_df = df_top7[df_top7['1'] == total1_count].copy()

        X_train = X_train.drop('1',axis=1)
        if all(x == 1 for x in arr1):
            predict_arr.append(1)
        elif all(x == 0 for x in arr1):
            predict_arr.append(0)
        else:
            for j in range(0,len(new_df)):
                top7 = new_df.iloc[j,:-1]
                dist_arr1 = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr1
                X_train['1']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
                df1_top7 = df1_sorted.head(7)
                arr12.append(df1_top7['1'].tolist())
                X_train = X_train.drop('1',axis=1)
            total_count = calculate_sum(arr12)
            predict_arr.append(total_count)
            arr1.clear()
            arr12.clear()

    predicted_testdata['Female'] = X_test['Female']
    predicted_testdata['6.8'] = X_test['6.8']
    predicted_testdata['3.3'] = X_test['3.3']
```

```
    predicted_testdata['Female'] = X_test['Female']
    predicted_testdata['6.8'] = X_test['6.8']
    predicted_testdata['3.3'] = X_test['3.3']
    predicted_testdata['0.9'] = X_test['0.9']
    predicted_testdata['1'] = predict_arr

    return predicted_testdata
```

```
def efficiency_test (predicted_testdata, X_test):
    counttt = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,5] == predicted_test.iloc[i,4]:
            counttt = counttt +1
    percentage = (counttt/len(X_test)) * 100
    return percentage
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = func_one(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

	Female	6.8	3.3	0.9	1
457	1	3.6	0.9	0.30	1
225	1	8.0	4.0	1.00	1
162	1	7.4	4.3	1.38	2
195	1	6.0	2.1	0.53	1
267	1	5.7	2.1	0.50	1
...	...	...	...	...	...
464	2	5.1	2.0	0.50	1
528	1	5.1	1.8	0.50	1
349	1	5.7	2.9	1.00	1
273	1	8.0	4.0	1.00	1
5	2	7.0	3.5	1.00	1

117 rows × 5 columns

5	2	7.0	3.5	1.00	1

117 rows × 5 columns

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['variety'] = y_test
```

```
Time taken: 114.37283110618591 seconds
```

```
X_test
```

	Female	6.8	3.3	0.9	Distance	variety
457	1	3.6	0.9	0.30	None	1
225	1	8.0	4.0	1.00	None	1
162	1	7.4	4.3	1.38	None	1
195	1	6.0	2.1	0.53	None	1
267	1	5.7	2.1	0.50	None	1
...	...	...	...	...	...	...
464	2	5.1	2.0	0.50	None	2
528	1	5.1	1.8	0.50	None	1
349	1	5.7	2.9	1.00	None	1
273	1	8.0	4.0	1.00	None	1
5	2	7.0	3.5	1.00	None	1

117 rows × 6 columns

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
67.52136752136752
```

4. Dataset: “BankNoteAuthentication”

a. HL-KNN
   Accuracy: 99.27%
   Time taken: 653.25 sec

```python
#More description detailed in the report
def high_level_knn_modified (X_train,X_test,y_train):
    dist_arr_lowlevel = []
    predict_arr = []
    dist_arr_highlevel = []
    neighbors_class_list = []
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-1]
        dist_arr_lowlevel = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr_lowlevel
        X_train['Column5']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        neighbors_class_list.append(df_top7['Column5'].tolist())
        total1_count = calculate_occurence_of_eachclass(neighbors_class_list)

        X_train = X_train.drop('Column5',axis=1)
        for j in range(0,len(df_top7)):
            top7 = df_top7.iloc[j,:-1]
            dist_arr_highlevel = calculate_euclidean_distance(X_train, top7)
            X_train['Distance']= dist_arr_highlevel
            X_train['Column5']= y_train
            df1 = pd.DataFrame(X_train)
            df1_sorted = df1.sort_values(by='Distance')
            df1_top7 = df1_sorted.head(7)
            neighbors_class_list.append(df1_top7['Column5'].tolist())
            X_train = X_train.drop('Column5',axis=1)
        total_count = calculate_occurence_of_eachclass(neighbors_class_list)
        predict_arr.append(total_count)
        neighbors_class_list.clear()
    predicted_testdata['Column1'] = X_test['Column1']
    predicted_testdata['Column2'] = X_test['Column2']
    predicted_testdata['Column3'] = X_test['Column3']
    predicted_testdata['Column5'] = predict_arr

    return predicted_testdata
```

```
    percentage = (counter/len(X_test)) * 100
    return percentage
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = high_level_knn_modified(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

	Column1	Column2	Column3	Column5
1240	-3.5510	1.89550	0.186500	1
703	1.3114	4.54620	2.293500	0
821	-4.0173	-8.31230	12.454700	1
1081	-5.1190	6.64860	-0.049987	1
37	3.6289	0.81322	1.627700	0
...	...	...	...	...
654	3.5127	2.90730	1.057900	0
1100	1.4378	0.66837	-2.026700	1
90	1.4806	7.63770	-2.787600	0
1107	1.2198	2.09820	-3.195400	1
262	1.8114	7.60670	-0.978800	0

275 rows × 4 columns

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['Outcome'] = y_test
```

```
Time taken: 653.2583322525024 seconds
```

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
99.27272727272727
```

b. Version1
   Accuracy:99.27%
   Time taken:700.47 sec

```python
def high_level_knn_modified (X_train,X_test,y_train):
    dist_arr_lowlevel = []
    predict_arr = []
    dist_arr_highlevel = []
    neighbors_class_list = []
    #highlevel_neighbors_class = []
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-1]
        dist_arr_lowlevel = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr_lowlevel
        X_train['Column5']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        neighbors_class_list.append(df_top7['Column5'].tolist())
        total1_count = calculate_occurence_of_eachclass(neighbors_class_list)
        #new_df = df_top7[df_top7['Column5'] == total1_count].copy()

        X_train = X_train.drop('Column5',axis=1)
        if all(x == 1 for x in neighbors_class_list):
            predict_arr.append(1)
        elif all(x == 0 for x in neighbors_class_list):
            predict_arr.append(0)
        else:
            for j in range(0,len(df_top7)):
                top7 = df_top7.iloc[j,:-1]
                dist_arr_highlevel = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr_highlevel
                X_train['Column5']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
                df1_top7 = df1_sorted.head(7)
                neighbors_class_list.append(df1_top7['Column5'].tolist())
                X_train = X_train.drop('Column5',axis=1)
            total_count = calculate_occurence_of_eachclass(neighbors_class_list)
            predict_arr.append(total_count)
            neighbors_class_list.clear()
        #highlevel_neighbors_class.clear()
    predicted_testdata['Column1'] = X_test['Column1']
    predicted_testdata['Column2'] = X_test['Column2']
    predicted_testdata['Column3'] = X_test['Column3']
    #predicted_testdata['Column4'] = X_test['Column4']
    predicted_testdata['Column5'] = predict_arr
```

```
    return percentage
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = high_level_knn_modified(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

	Column1	Column2	Column3	Column5
1240	-3.5510	1.89550	0.186500	1
703	1.3114	4.54620	2.293500	0
821	-4.0173	-8.31230	12.454700	1
1081	-5.1190	6.64860	-0.049987	1
37	3.6289	0.81322	1.627700	0
...	...	...	...	...
654	3.5127	2.90730	1.057900	0
1100	1.4378	0.66837	-2.026700	1
90	1.4806	7.63770	-2.787600	0
1107	1.2198	2.09820	-3.195400	1
262	1.8114	7.60670	-0.978800	0

275 rows × 4 columns

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['Outcome'] = y_test
```

```
Time taken: 700.4708249568939 seconds
```

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
99.27272727272727
```

c. Version2
Accuracy is 99.27%
Time taken is 703.6089 sec

```python
def calculate_euclidean_distance(training_features, test_features):
    dist_list = []
    for i in range(0,len(training_features)):
        #Not including the Distance column
        tf = training_features.iloc[i,:-1]
        dist = np.sqrt(np.sum(np.square(test_features - tf)))
        dist_list.append(dist)
    return dist_list
```

```python
#This function calculates the number of occurences of each of the k-nearest neighbors class and
#retruns the class value that has maximum occurence
def calculate_occurence_of_eachclass(knearest_neighbors_class_list):
    count_per_inner_list = [inner_list.count(0) for inner_list in knearest_neighbors_class_list]
    total_count0 = sum(count_per_inner_list)
    count_per_inner_list = [inner_list.count(1) for inner_list in knearest_neighbors_class_list]
    total_count1 = sum(count_per_inner_list)

    if total_count1 > total_count0:
        return 1
    elif total_count0 > total_count1:
        return 0
    else:
        return 1
```

```python
#This function is the main algorithm which is an improved version of HL-KNN algorithm. Takes the training data, test
#data and predicts the class of the given test data.
#More description detailed in the report
def high_level_knn_modified (X_train,X_test,y_train):
    dist_arr_lowlevel = []
    predict_arr = []
    dist_arr_highlevel = []
    lowlevel_neighbors_class = []
    highlevel_neighbors_class = []
    predicted_testdata = pd.DataFrame()
    for i in range(0,len(X_test)):
        test = X_test.iloc[i,:-1]
        dist_arr_lowlevel = calculate_euclidean_distance(X_train, test)
        X_train['Distance']= dist_arr_lowlevel
        X_train['Column5']= y_train
        df = pd.DataFrame(X_train)
        df_sorted = df.sort_values(by='Distance')
        df_top7 = df_sorted.head(7)
        lowlevel_neighbors_class.append(df_top7['Column5'].tolist())
        total1_count = calculate_occurence_of_eachclass(lowlevel_neighbors_class)
        new_df = df_top7[df_top7['Column5'] == total1_count].copy()

        X_train = X_train.drop('Column5',axis=1)
        if all(x == 1 for x in lowlevel_neighbors_class):
            predict_arr.append(1)
        elif all(x == 0 for x in lowlevel_neighbors_class):
            predict_arr.append(0)
        else:
            for j in range(0,len(new_df)):
                top7 = new_df.iloc[j,:-1]
                dist_arr_highlevel = calculate_euclidean_distance(X_train, top7)
                X_train['Distance']= dist_arr_highlevel
                X_train['Column5']= y_train
                df1 = pd.DataFrame(X_train)
                df1_sorted = df1.sort_values(by='Distance')
                df1_top7 = df1_sorted.head(7)
                highlevel_neighbors_class.append(df1_top7['Column5'].tolist())
                X_train = X_train.drop('Column5',axis=1)
            total_count = calculate_occurence_of_eachclass(highlevel_neighbors_class)
            predict_arr.append(total_count)
            lowlevel_neighbors_class.clear()
            highlevel_neighbors_class.clear()
    predicted_testdata['Column1'] = X_test['Column1']
    predicted_testdata['Column2'] = X_test['Column2']
```

```
            total_count = calculate_occurence_of_eachclass(highlevel_neighbors_class)
            predict_arr.append(total_count)
            lowlevel_neighbors_class.clear()
            highlevel_neighbors_class.clear()
    predicted_testdata['Column1'] = X_test['Column1']
    predicted_testdata['Column2'] = X_test['Column2']
    predicted_testdata['Column3'] = X_test['Column3']
    #predicted_testdata['Column4'] = X_test['Column4']
    predicted_testdata['Column5'] = predict_arr

    return predicted_testdata
```

```
#This function calculates the accuracy of the predicted data by dividing the correctly predicetd data over total
#number of test samples multiplied by 100
def efficiency_test (predicted_testdata, X_test):
    counter = 0
    for i in range (0,len(X_test)):
        if X_test.iloc[i,4] == predicted_test.iloc[i,3]:
            counter = counter +1
    percentage = (counter/len(X_test)) * 100
    return percentage
```

```
predicted_test = pd.DataFrame()
start_time = time.time()
predicted_test = high_level_knn_modified(X_train,X_test,y_train)
end_time = time.time()
predicted_test
```

	Column1	Column2	Column3	Column5
1240	-3.5510	1.89550	0.186500	1
703	1.3114	4.54620	2.293500	0
821	-4.0173	-8.31230	12.454700	1
1081	-5.1190	6.64860	-0.049987	1
37	3.6289	0.81322	1.627700	0
...	...	...	...	...
654	3.5127	2.90730	1.057900	0
1100	1.4378	0.66837	-2.026700	1
90	1.4806	7.63770	-2.787600	0
1107	1.2198	2.09820	-3.195400	1
262	1.8114	7.60670	-0.978800	0

275 rows × 4 columns

```
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
X_test ['Outcome'] = y_test
```

```
Time taken: 703.6089808940887 seconds
```

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
99.27272727272727
```

6. Dataset: “DryBean.csv”

Version2

Accuracy is 58.348%

Time taken is 2041.540 sec

	AspectRation	Eccentricity	Compactness	ShapeFactor3	Distance
486	1.393202	0.696279	0.845816	0.715404	None
1548	1.359475	0.677440	0.855788	0.732374	None
1016	1.810887	0.833701	0.736268	0.542091	None
857	1.484348	0.739008	0.820075	0.672523	None
598	1.778692	0.826994	0.742668	0.551555	None
...	...	...	...	...	...
960	1.891548	0.848829	0.724326	0.524647	None
905	1.238765	0.590202	0.897732	0.805924	None
1096	1.487818	0.740437	0.818590	0.670090	None
235	1.487882	0.740464	0.819402	0.671419	None
1061	1.484127	0.738917	0.820220	0.672761	None

2177 rows × 5 columns

```
X_test ['variety'] = y_test
```

```
Time taken: 2041.5407149791718 seconds
```

```
X_test
```

	AspectRation	Eccentricity	Compactness	ShapeFactor3	Distance	variety
1443	1.993807	0.865127	0.704707	0.496611	None	6
1152	1.259739	0.608159	0.890561	0.793099	None	1
1348	1.671816	0.801383	0.772827	0.597262	None	5
1334	1.619577	0.786614	0.783865	0.614444	None	3
2700	1.686608	0.805272	0.767052	0.588368	None	3
...	...	...	...	...	...	...
1398	1.528662	0.756350	0.808248	0.653265	None	5
533	2.056002	0.873747	0.693484	0.480920	None	6
2490	1.191598	0.543809	0.915346	0.837858	None	1
702	1.743995	0.819278	0.755729	0.571127	None	7
859	1.497643	0.744416	0.814682	0.663707	None	3

545 rows × 6 columns

```
match_percent = efficiency_test (predicted_test, X_test)
```

```
match_percent
```

```
58.34862385321101
```