kNN-Study

November 23, 2018

1 K-Nearest Neighbours

Basically trying to implement common distance calculation algorithms and would like to try KNN using those on Iris Data Set
Referece [http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/]

- Implemented required distance functions (yet to review its accuracy)
- Loaded Irsi dataset
- splitted into training and test dataset 80%, 20%
- for each distance algorith, running k-NN from 1 to 9 and printing its accuracy

```
In [1]: # Importing required modules
        from math import * # for math operation
        from decimal import Decimal # for decimal approximation
        import operator # for selection
        import pandas as pd # for handling iris dataset
        from sklearn.model_selection import train_test_split
```

```
In [2]: # My test vector

        v1 = [1.0, 3.2, 4.8, 0.1, 3.2, 0.6, 2.2, 1.1]
        v2 = [0.1, 5.2, 1.9, 4.2, 1.9, 0.1, 0.1, 6.0]
```

1.1 Distance Algorithm Implementations

1.1.1 Manhattan Distance

Also referred as L1 Norm

$$L_1 Norm = ||x - y||_1 = \left(\sum_{i=1}^{n} |(x_i - y_i)| \right)$$

```
In [3]: def manhanttan_dist(v1, v2):
            '''
            returns manhattan distance between vector v1 and v2 having same dimension d
            numeric components for vectors v1 and v2 are assumed
            '''
            return round(Decimal(sum(abs(a-b) for a, b in zip(v1, v2))),3)

        print(manhanttan_dist(v1,v1), manhanttan_dist(v2,v2))
        print(manhanttan_dist(v1, v2), manhanttan_dist(v2, v1))
```

```
0.000 0.000
18.700 18.700
```

1.1.2 Eucliean Distance

Also referred as L2 Norm

$$L_2 Norm = ||x - y||_2 = \sqrt{\left(\sum_{i=1}^{d} (x_{1i} - y_{2i})^2 \right)} = \sqrt{(x - y)^T (x - y)}$$

$$L_2 Norm = ||x - y||_2 = \left(\sum_{i=1}^{d} (x_{1i} - y_{2i})^2 \right)^{\frac{1}{2}}$$

```
In [4]: def eucd_dist(v1,v2):
            '''
            returns euclidean distance between vector v1 and v2 having same dimension d
            numeric components for vectors v1 and v2 are assumed
            '''
            return round(Decimal(sqrt(sum(pow(a-b,2) for a,b in zip(v1,v2)))),3)
```

```
In [5]: print(eucd_dist(v1,v1), eucd_dist(v2,v2))
        print(eucd_dist(v1, v2), eucd_dist(v2, v1))
```

```
0.000 0.000
7.771 7.771
```

1.1.3 Minkowski Distance

Also referred as Lp Norm

$$L_p Norm = ||x - y||_p = \left(\sum_{i=1}^{d} |x_{1i} - y_{2i}|^p \right)^{\frac{1}{p}}$$

Observations of Minkowski:

$$L_1 Norm = ManhattanDistance$$

$$L_2 Norm = EuclideanDistance$$

$$L_\infty Norm = ChebyshevDistance = L_{max} Norm$$

```
In [6]: def getNthRoot(val, n_root):
            '''
            returns n_th root of the given value
            '''
            return round(Decimal(val) ** Decimal(Decimal(1.0)/n_root),3)

        def minkowski_dist(v1, v2, p):
            '''
            returns minkowski distance between vectors v1 and v2 of same dimension d
                numeric components for vectors v1 and v2 are assumed
                v1, v2 ==> vectors
                p ==> p-form that need to be calcualted
            '''
            return getNthRoot(sum(pow(abs(a-b),p) for a,b in zip(v1, v2)), p)

        #print(getNthRoot(2,9))
        #print(minkowski_dist([0,3,4,5], [7,6,3,-1], 3))


        for p in range(1,5):
            print("p :", p, minkowski_dist(v1, v2, p), minkowski_dist(v2, v1, p))
```

```
p : 1 18.700 18.700
p : 2 7.771 7.771
p : 3 6.138 6.138
p : 4 5.579 5.579
```

1.1.4 Consine Similarity

$$\cos\theta = \frac{a.b}{||a||\ ||b||}$$

$$\cos\theta = \frac{a^T b}{||a||\ ||b||}$$

$$\cos\theta = \left(\frac{a}{||a||}\right)^T \left(\frac{b}{||b||}\right)$$

```
In [7]: def dot_product(v1, v2):
            '''
            returns algebraic dot product of two vectors v1 and v2
            '''
            return Decimal(sum(a*b for a,b in zip(v1,v2)))

        def getLength(v1):
            '''
            returns length/magniture of the given vector
            '''
            return Decimal(sqrt(sum(x*x for x in v1)))

        def scalarMultiply(v1, c):
            '''
            performs scalar multiplication over given vector v1
            '''
            return [round(Decimal(x*c),3) for x in v1]

        def normalize(v1):
            '''
            returns the unit vector of given vector v1
            '''
            l = getLength(v1)
```

```
    if(l == 0):
        return 0; # TO DO - Raise Exception

    return scalarMultiply(v1,(Decimal(1.0)/l))

def cosine_similarity(v1,v2):
    '''
    returns consine similarity between vectors v1 and v2
    '''
    numerator = dot_product(v1, v2)
    denominator = getLength(v1) * getLength(v2)
    return round(Decimal(numerator / denominator), 3)
```

In [8]:
```
# Validation

a = [5,3]
b = [1,4]

print('Euclidean_Distance(a,b): ', eucd_dist(a,b))
print('Unit Vecor of a: ', normalize(a))
print('Unit Vecor of b: ', normalize(b))
print('cos_similarity(a,b): ', cosine_similarity(a,b))
```

```
Euclidean_Distance(a,b):  4.123
Unit Vecor of a:  [Decimal('0.857'), Decimal('0.514')]
Unit Vecor of b:  [Decimal('0.243'), Decimal('0.970')]
cos_similarity(a,b):  0.707
```

1.1.5 Cosine Dissimlarity

$$1 - cosine\_similarity(x, y)$$

In [9]:
```
def consine_dissimilarity(v1, v2):
    '''
    returns cosine dissimilarity between vectors v1 and v2
    '''
    return (1-cosine_similarity(v1,v2))
```

In [10]:
```
print('cos_similarity(a,b): ', consine_dissimilarity(a,b))
```

```
cos_similarity(a,b):  0.293
```

1.2 k-NN Implementation (for Iris DataSet)

1.3 Calculating Accuracy

2 Iris Data Set

2.1 Load DataSet

In [11]:
```
df = pd.read_csv('./iris.data')
df.head()
```

Out[11]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

2.2 Split DataSet

In [12]:
```
# Split the data and labels for easy handling
# 80% training
# 20% for testing
df_train, df_test = train_test_split(df, test_size=0.2)

#df_data = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
#df_labels = df[['species']]
#print(df_data.head())
#print(df_labels.head())
```

In [13]:
```
print('Training Dataset:')
print(df_train.shape)
print(df_train.head())
print(df_train.describe())
```

```
Training Dataset:
(120, 5)
     sepal_length  sepal_width  petal_length  petal_width          species
135           7.7          3.0           6.1          2.3   Iris-virginica
80            5.5          2.4           3.8          1.1  Iris-versicolor
90            5.5          2.6           4.4          1.2  Iris-versicolor
84            5.4          3.0           4.5          1.5  Iris-versicolor
34            4.9          3.1           1.5          0.1      Iris-setosa
       sepal_length  sepal_width  petal_length  petal_width
count    120.000000   120.000000    120.000000   120.000000
mean       5.828333     3.046667      3.704167     1.171667
std        0.835673     0.441337      1.763467     0.756172
min        4.300000     2.000000      1.100000     0.100000
25%        5.100000     2.800000      1.500000     0.275000
50%        5.700000     3.000000      4.200000     1.300000
75%        6.400000     3.300000      5.025000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

In [14]:
```
print('Test Dataset:')
print(df_test.shape)
print(df_test.head())
print(df_test.describe())
```

```
Test Dataset:
(30, 5)
     sepal_length  sepal_width  petal_length  petal_width          species
116           6.5          3.0           5.5          1.8   Iris-virginica
57            4.9          2.4           3.3          1.0  Iris-versicolor
73            6.1          2.8           4.7          1.2  Iris-versicolor
5             5.4          3.9           1.7          0.4      Iris-setosa
144           6.7          3.3           5.7          2.5   Iris-virginica
       sepal_length  sepal_width  petal_length  petal_width
count     30.000000    30.000000     30.000000    30.000000
mean       5.903333     3.083333      3.976667     1.306667
std        0.807928     0.406909      1.781259     0.794348
min        4.600000     2.200000      1.000000     0.100000
25%        5.100000     2.925000      1.700000     0.425000
50%        6.000000     3.050000      4.550000     1.400000
75%        6.475000     3.375000      5.400000     1.875000
max        7.600000     3.900000      6.600000     2.500000
```

2.3 Calculating Neighbors

In [15]:
```
def getNeighbours(training_data_set, query_point, k, algo='euct', p=3):
    '''
    returns list having k neighbors to the given query data point
    input:
        training_data_set: Pandas DataFrame
        query_point: Pandas DataSeries
        k: Number of Neighbors to calculate
        algo: type of distance algorithm to use
            euct (euclidean distance default)
            maht (manhattan)
            mink (minkowski)
            coss (cosine similarity)
        p: minkowski required p norm (default 3)
    Output:
        List of nearest data points
    '''
    distances = [] # list to hold all the neighbors

    # calcualte distance between query_point and every point in data set
    # create a list
    for x in range(len(training_data_set)):
        # stip non-numeric label - in training data
        v1 = training_data_set.iloc[x]
        v1 = v1[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
        #print(type(v1), v1)

        # stip non-numeric label - in query data
        q_v = query_point[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
        if algo == 'maht':
            dist = manhanttan_dist(q_v, v1)
        elif algo == 'mink':
```

```
                dist = minkowski_dist(q_v, v1, p)
            elif algo == 'coss':
                dist = cosine_similarity(q_v, v1)
                #print('Coss: ', dist)
            else:
                dist = eucd_dist(q_v, v1)
            distances.append((dist, training_data_set.iloc[x]))

        # sort the list in ascending order
        distances.sort(key=lambda tup:tup[0])
        #print(distances)

        # select k nearest neighbors and return it
        neighbors = []
        for i in range(k):
            neighbors.append(distances[i][1])
        return neighbors
```

2.4 Calculating Responses

```
In [16]: def getClassLabel(neighbors):
             '''
             returns the class label having majority vote
             Note that it doesn't handle 'Not Sure' case yet
             '''
             class_votes = {} # dictionary keys are flowers, values are its counts
             for x in range(len(neighbors)):
                 class_label = neighbors[x][-1]
                 if class_label in class_votes:
                     class_votes[class_label] += 1
                 else:
                     class_votes[class_label] = 1

             response = max(class_votes.items(), key=operator.itemgetter(1))[0]
             return response
```

2.5 Accuracy of Predictions

- Try to check accuray for k in range 1 to 9
  - Euclidean Distance
  - Cosine Similarity
  - Manhattan Distance
  - L_3 Norm (minkowski distance)

2.5.1 Euclidean Distance

```
In [17]: %%time

         for k in range(1,10):
             correct_predictions = 0
             for t_index in range(len(df_test)):
                 test_data_point = df_test.iloc[t_index]
                 neighbors = getNeighbours(df_train, test_data_point, k)
                 predicted_class = getClassLabel(neighbors)
                 if predicted_class == test_data_point['species']:
                     correct_predictions += 1
                 #print('Predicted: ', predicted_class, ' Actual: ', test_data_point['species'])

             print('k: ', k, 'Percent: ', round((correct_predictions/len(df_test)) * 100,3),'Total correct predictions: ', correct_prediction

k:  1 Percent:  96.667 Total correct predictions:  29  out of  30
k:  2 Percent:  96.667 Total correct predictions:  29  out of  30
k:  3 Percent:  96.667 Total correct predictions:  29  out of  30
k:  4 Percent:  96.667 Total correct predictions:  29  out of  30
k:  5 Percent:  93.333 Total correct predictions:  28  out of  30
k:  6 Percent:  96.667 Total correct predictions:  29  out of  30
k:  7 Percent:  93.333 Total correct predictions:  28  out of  30
k:  8 Percent:  93.333 Total correct predictions:  28  out of  30
k:  9 Percent:  93.333 Total correct predictions:  28  out of  30
CPU times: user 1min 11s, sys: 666 ms, total: 1min 12s
Wall time: 1min 11s
```

2.5.2 Cosine Similarity

In [18]:
```
%%time

for k in range(1,10):
    correct_predictions = 0
    for t_index in range(len(df_test)):
        test_data_point = df_test.iloc[t_index]
        neighbors = getNeighbours(df_train, test_data_point, k, 'coss')
        predicted_class = getClassLabel(neighbors)
        if predicted_class == test_data_point['species']:
            correct_predictions += 1
        #print('Predicted: ', predicted_class, ' Actual: ', test_data_point['species'])

    print('k: ', k, 'Percent: ', round((correct_predictions/len(df_test)) * 100,3),'Total correct predictions: ', correct_prediction
```

```
k:  1 Percent:  0.0 Total correct predictions:  0  out of  30
k:  2 Percent:  0.0 Total correct predictions:  0  out of  30
k:  3 Percent:  0.0 Total correct predictions:  0  out of  30
k:  4 Percent:  0.0 Total correct predictions:  0  out of  30
k:  5 Percent:  0.0 Total correct predictions:  0  out of  30
k:  6 Percent:  0.0 Total correct predictions:  0  out of  30
k:  7 Percent:  0.0 Total correct predictions:  0  out of  30
k:  8 Percent:  0.0 Total correct predictions:  0  out of  30
k:  9 Percent:  0.0 Total correct predictions:  0  out of  30
CPU times: user 1min 14s, sys: 654 ms, total: 1min 15s
Wall time: 1min 14s
```

2.5.3 Manhattan Distance

In [19]:
```
%%time

for k in range(1,10):
    correct_predictions = 0
    for t_index in range(len(df_test)):
        test_data_point = df_test.iloc[t_index]
        neighbors = getNeighbours(df_train, test_data_point, k, 'maht')
        predicted_class = getClassLabel(neighbors)
        if predicted_class == test_data_point['species']:
            correct_predictions += 1
        #print('Predicted: ', predicted_class, ' Actual: ', test_data_point['species'])

    print('k: ', k, 'Percent: ', round((correct_predictions/len(df_test)) * 100,3),'Total correct predictions: ', correct_prediction
```

```
k:  1 Percent:  96.667 Total correct predictions:  29  out of  30
k:  2 Percent:  96.667 Total correct predictions:  29  out of  30
k:  3 Percent:  96.667 Total correct predictions:  29  out of  30
k:  4 Percent:  96.667 Total correct predictions:  29  out of  30
k:  5 Percent:  93.333 Total correct predictions:  28  out of  30
k:  6 Percent:  93.333 Total correct predictions:  28  out of  30
k:  7 Percent:  93.333 Total correct predictions:  28  out of  30
k:  8 Percent:  96.667 Total correct predictions:  29  out of  30
k:  9 Percent:  93.333 Total correct predictions:  28  out of  30
CPU times: user 1min 11s, sys: 639 ms, total: 1min 12s
Wall time: 1min 12s
```

2.5.4 Minkowski Distance with p=3

In [20]:
```
%%time

for k in range(1,10):
    correct_predictions = 0
    for t_index in range(len(df_test)):
        test_data_point = df_test.iloc[t_index]
        neighbors = getNeighbours(df_train, test_data_point, k, 'mink')
        predicted_class = getClassLabel(neighbors)
        if predicted_class == test_data_point['species']:
            correct_predictions += 1
        #print('Predicted: ', predicted_class, ' Actual: ', test_data_point['species'])

    print('k: ', k, 'Percent: ', round((correct_predictions/len(df_test)) * 100,3),'Total correct predictions: ', correct_prediction
```

```
k:  1 Percent:  96.667 Total correct predictions:  29  out of  30
k:  2 Percent:  96.667 Total correct predictions:  29  out of  30
k:  3 Percent:  96.667 Total correct predictions:  29  out of  30
```

```
k:  4 Percent:  96.667 Total correct predictions:  29  out of  30
k:  5 Percent:  93.333 Total correct predictions:  28  out of  30
k:  6 Percent:  96.667 Total correct predictions:  29  out of  30
k:  7 Percent:  93.333 Total correct predictions:  28  out of  30
k:  8 Percent:  96.667 Total correct predictions:  29  out of  30
k:  9 Percent:  93.333 Total correct predictions:  28  out of  30
CPU times: user 1min 20s, sys: 651 ms, total: 1min 20s
Wall time: 1min 20s
```

3 Observation

- Training Data highly influences the prediction accuracy
  - if we rerun this test multiple times, you can see differences in accuracy for each test
- Cosine Similairy
  - Incomplete implementation, so nothing deduced yet