MINGGU KE 6

Pengklasifikasian k-NN (k-Nearest Neighbor)

"Show me who your friends are and I'll tell you who you are?"

Konsep k-NN dapat susah digambarkan dengan sederhana. Banyak peribahasa lama mengungkapnya dan ditemukan juga dalam hadits Rasulullah shallallaahu 'alaihi wa sallam bersabda :

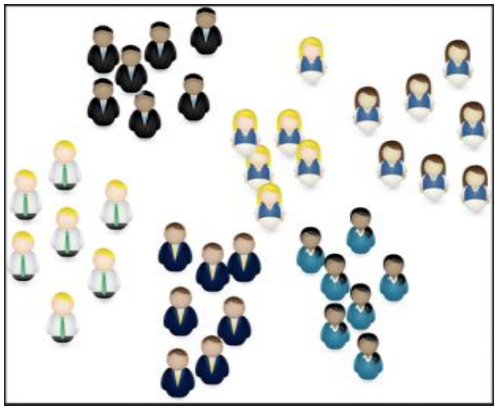
"Barangsiapa yang menyerupai suatu kaum, maka ia termasuk golongan mereka".

Hal ini menandakan bahwa sebenarnya k-NN adalah bagian dari kehidupan sehari-hari kita dan perkiraan yang dilakukan: Bayangkan apabila bertemu dengan kelompok dari orang, mereka sangat muda, bergaya, dan sportif. Mereka bicara tentang teman-teman mereka "Ben tidakkah kamu bersama mereka". Lalu apa imajinasimu tentang Ben? Betul, anda akan berimajinasi bahwa Ben sedang menjadi muda, bergaya, dan sportif sebagaimana mereka.

Apabila kita belajar bahwa Ben tinggal dalam suatu tetangga dimana orang memilih konservatif dan rata-rata pendapatan di atas 200000 dolar per tahun? Keduanya tetangganya membuat lebih dari 300000 dolar per tahun? Apa yang terpikir oleh anda tentang Ben? Hampir sebagian besarnya, anda tidak mempertimbangkan bahwa Ben adalah suatu underdog dan mungkin menyangkanya sebagai suatu conservatif juga?

Sekarang mari sedikit lebih matematis:

k-NN bekerja secara langsung pada sampel yang dipelajari, ketimbang membuat rule apabila dibandingkan dengan metode pengklasifikasian lain. Berikut ini adalah ilustrasinya.



Algoritme NN

Diberikan suatu himpunan kategori $\{c1,c2,...cn\}$, disebut juga kelas-kelas, sebagai contoh $\{\text{”male”, ”female”}\}$. Terdapat juga learnset *LS* terdiri dari contoh label. Tugas dari klasifikasi meliputi menempatkan suatu kategori atau kelas pada suatu contoh secara acak. Jika contoh itu *o* adalah elemen dari *LS*, label dari contoh akan digunakan.

Sekarang kita akan melihat kasus dimana *o* tidak dalam *LS*:

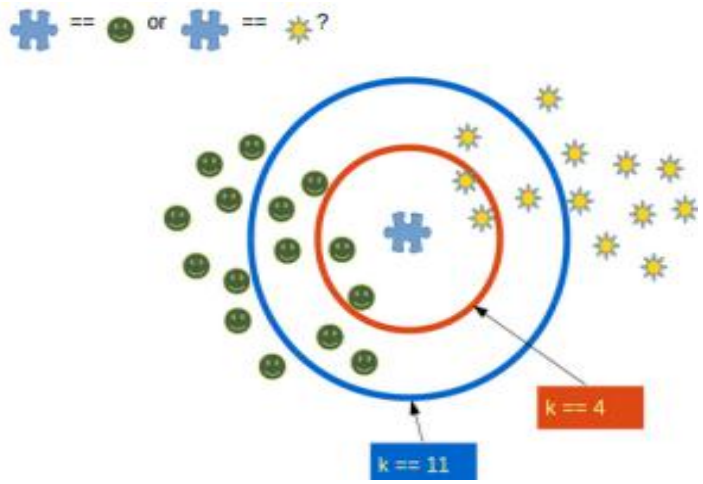
o dibandingkan dengan semua contoh dalam *LS*. Suatu jarak metric digunakan dalam perbandingan. Kita menetapkan yaitu *k* tetangga-tetangga terdekat dari *o*, yaitu item-item dengan jarak paling dekat. *k* adalah suatu kostanta yang didefinisikan user dan suatu bilangan bulat positif.

Yang paling umum dari kelas *LS* akan ditempatkan pada contoh *o*, Jika k=1 maka obyek dengan sederhananya ditempatkan pada kelas dari tetangga terdekat tunggal.

Algoritme k-NN ini termasuk algoritme yang paling sederhana dari pada algoritme pada ML yang lain. k-NN adalah tipe dari pembelajaran berbasis contoh, atau lazy learning, dimana fungsi hanya memperkirakan secara lokal dan semua komputasi diperform, ketika kita melakukan klasifikasi aktual.

Preparing Dataset

Berikut ini adalah ilustrasi dari k-NN:



Sebelum secara aktual kita memulai menulis suatu k-NN, kita membutuhkan suatu learnset. Kita akan menggunakan “iris” dataset yang disediakan dari modul sklearn.

Dataset terdiri dari 50 sampel dari setiap spesies dari iris berikut adalah contoh spesiesnya:

- iris setosa
- iris virginica
- iris versicolor

Empat fitur diukur dari sampel meliputi: panjang dan lebar dari sepal dan petal dalam cm.

```
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target
print(iris_data[0], iris_data[79], iris_data[100])
print(iris_labels[0], iris_labels[79], iris_labels[100])
```

Hasil dari skrip di atas adalah berikut ini:

```
[ 5.1  3.5  1.4  0.2] [ 5.7  2.6  3.5  1. ] [ 6.3  3.3  6.   2.5]
0 1 2
```

Kita akan membuat suatu learnset dari set di atas. Kita menggunakan permutasi dari np.random untuk membagi data secara acak.

```
np.random.seed(42)
```

```
indices = np.random.permutation(len(iris_data))
n_training_samples = 12
learnset_data = iris_data[indices[:-n_training_samples]]
learnset_labels = iris_labels[indices[:-n_training_samples]]
testset_data = iris_data[indices[-n_training_samples:]]
testset_labels = iris_labels[indices[-n_training_samples:]]
print(learnset_data[:4], learnset_labels[:4])
print(testset_data[:4], testset_labels[:4])
```

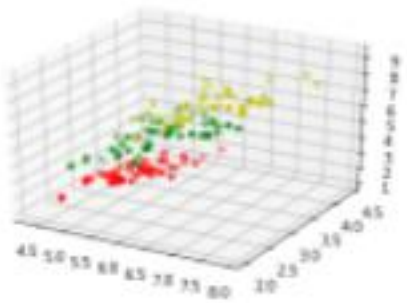
Hasil keluaran program di atas adalah berikut ini:

```
[[ 6.1   2.8   4.7   1.2]
 [ 5.7   3.8   1.7   0.3]
 [ 7.7   2.6   6.9   2.3]
 [ 6.    2.9   4.5   1.5]] [1 0 2 1]
[[ 5.7   2.8   4.1   1.3]
 [ 6.5   3.    5.5   1.8]
 [ 6.3   2.3   4.4   1.3]
 [ 6.4   2.9   4.3   1.3]] [1 2 1 1]
```

Kode berikutnya hanya dibutuhkan untuk memvisualisasikan data dari learnset di atas. Data akan terdiri dari nilai-nilai per item iris, sehingga kita akan mengurangi data ke tiga nilai dengan summing up nilai ketiga dan keempat. Cara ini akan dapat menggambarkan data dalam ruang 3 dimensi

```
# following line is only necessary, if you use ipython notebook!!!
#%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
colours = ("r", "b")
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(learnset_data)):
        if learnset_labels[i] == iclass:
            X[iclass][0].append(learnset_data[i][0])
            X[iclass][1].append(learnset_data[i][1])
            X[iclass][2].append(sum(learnset_data[i][2:]))
colours = ("r", "g", "y")
fig = plt.figure()ax = fig.add_subplot(111, projection='3d')
for iclass in range(3):
        ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])
plt.show()
```

Berikut ini adalah gambar dari skrip di atas:



Menentukan Tetangga

Untuk menentukan tetangga similaritas diantara dua contoh, kita butuh suatu fungsi distance. Dalam contoh kita euclidian distance adalah ideal.

```
def distance(instance1, instance2):
    # just in case, if the instances are lists or tuples:
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

     return np.linalg.norm(instance1 - instance2)
print(distance([3, 5], [1, 1]))
print(distance(learnset_data[3], learnset_data[44]))
```

Akan mendapatkan hasil sebagai berikut:

```
4.472135955
3.41906419946
```

Fungsi get_neighbors mengembalikan suatu list dengan tetangga ‘k’, yang terdekat pada contoh ‘test_instance’:

```
def get_neighbors(training_set,
                  labels,
                  test_instance,
                  k,
                  distance=distance):
    """ get_neighors calculates a list of the k nearest neighbors of an instance
'test_instance'. The list neighbors contains 3-tuples with (index, dist, label)
where index is the index from the training_set, dist is the distance
```

```
between the test_instance and the instance training_set[index] distance is
a reference to a function used to calculate the distances """
    distances = []
    for index in range(len(training_set)):
        dist = distance(test_instance, training_set[index])
        distances.append((training_set[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return neighbors
```

Kita akan menguji fungsi dengan sample iris:

```
for i in range(5):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              3,
                              distance=distance)
    print(i,
          testset_data[i],
          testset_labels[i],
          neighbors)
```

Berikut ini adalah outputnya:

```
0 [ 5.7  2.8  4.1  1.3] 1 [(array([ 5.7,  2.9,  4.2,  1.3]), 0.14142135623730995, 1),
(array([ 5.6,  2.7,  4.2,  1.3]), 0.17320508075688815, 1), (array([ 5.6,  3. ,  4.1,  1.3]),
0.22360679774997935, 1)]
1 [ 6.5  3.   5.5  1.8] 2 [(array([ 6.4,  3.1,  5.5,  1.8]), 0.14142135623730931, 2),
(array([ 6.3,  2.9,  5.6,  1.8]), 0.24494897427831783, 2), (array([ 6.5,  3. ,  5.2,  2. ]),
0.36055512754639879, 2)]
2 [ 6.3  2.3  4.4  1.3] 1 [(array([ 6.2,  2.2,  4.5,  1.5]), 0.26457513110645864, 1),
(array([ 6.3,  2.5,  4.9,  1.5]), 0.57445626465380295, 1), (array([ 6. ,  2.2,  4. ,  1. ]),
0.5916079783099617, 1)]
3 [ 6.4  2.9  4.3  1.3] 1 [(array([ 6.2,  2.9,  4.3,  1.3]), 0.20000000000000018, 1),
(array([ 6.6,  3. ,  4.4,  1.4]), 0.26457513110645869, 1), (array([ 6.6,  2.9,  4.6,  1.3]),
0.3605551275463984, 1)]
4 [ 5.6  2.8  4.9  2. ] 2 [(array([ 5.8,  2.7,  5.1,  1.9]), 0.31622776601683755, 2),
(array([ 5.8,  2.7,  5.1,  1.9]), 0.31622776601683755, 2), (array([ 5.7,  2.5,  5. ,  2. ]),
0.33166247903553986, 2)]
```

Memilih untuk mendapatkan suatu hasil tunggal

Kita akan menulis suatu fungsi vote sekarang. Fungsi ini menggunakan kelas ‘Counter’ dari koleksi untuk menghitung jumlah dari kelas-kelas dalam suatu list contoh. List contoh ini akan menjadi tetangga-tetangga tentunya. Fungsi ‘vote’ mengembalikan kelas paling umum.

```
from collections import Counter
def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]
```

Kita akan menguji vote dari sampel training kita:

```
for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              3,
                              distance=distance)
    print("index: ", i,
          ", result of vote: ", vote(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", testset_data[i])
```

Hasil keluaran python sebagai berikut:

```
index:  0 , result of vote:  1 , label:  1 , data:  [ 5.7  2.8  4.1  1.3]
index:  1 , result of vote:  2 , label:  2 , data:  [ 6.5  3.   5.5  1.8]
index:  2 , result of vote:  1 , label:  1 , data:  [ 6.3  2.3  4.4  1.3]
index:  3 , result of vote:  1 , label:  1 , data:  [ 6.4  2.9  4.3  1.3]
index:  4 , result of vote:  2 , label:  2 , data:  [ 5.6  2.8  4.9  2. ]
index:  5 , result of vote:  2 , label:  2 , data:  [ 5.9  3.   5.1  1.8]
index:  6 , result of vote:  0 , label:  0 , data:  [ 5.4  3.4  1.7  0.2]
index:  7 , result of vote:  1 , label:  1 , data:  [ 6.1  2.8  4.   1.3]
index:  8 , result of vote:  1 , label:  2 , data:  [ 4.9  2.5  4.5  1.7]
index:  9 , result of vote:  0 , label:  0 , data:  [ 5.8  4.   1.2  0.2]
index:  10 , result of vote:  1 , label:  1 , data:  [ 5.8  2.6  4.   1.2]
index:  11 , result of vote:  2 , label:  2 , data:  [ 7.1  3.   5.9  2.1]
```

Kita melihat bahwa prediksi berhubungan dengan hasil-hasil yang terlabel, kecuali dalam kasus dari item dengan index 8. ‘vote_prob’ adalah suatu fungsi

seperti ‘vote’ tetapi mengembalikan class name dan probabilitas dari kelas ini.

```
def vote_prob(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    labels, votes = zip(*class_counter.most_common())
    winner = class_counter.most_common(1)[0][0]
    votes4winner = class_counter.most_common(1)[0][1]
    return winner, votes4winner/sum(votes)

for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              5,
                              distance=distance)
    print("index: ", i,
          ", vote_prob: ", vote_prob(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", testset_data[i])
```

Kita akan mendapatkan output sebagai berikut:

```
index:  0 , vote_prob:  (1, 1.0) , label:  1 , data:  [ 5.7  2.8  4.1  1.3]
index:  1 , vote_prob:  (2, 1.0) , label:  2 , data:  [ 6.5  3.   5.5  1.8]
index:  2 , vote_prob:  (1, 1.0) , label:  1 , data:  [ 6.3  2.3  4.4  1.3]
index:  3 , vote_prob:  (1, 1.0) , label:  1 , data:  [ 6.4  2.9  4.3  1.3]
index:  4 , vote_prob:  (2, 1.0) , label:  2 , data:  [ 5.6  2.8  4.9  2. ]
index:  5 , vote_prob:  (2, 0.8) , label:  2 , data:  [ 5.9  3.   5.1  1.8]
index:  6 , vote_prob:  (0, 1.0) , label:  0 , data:  [ 5.4  3.4  1.7  0.2]
index:  7 , vote_prob:  (1, 1.0) , label:  1 , data:  [ 6.1  2.8  4.   1.3]
index:  8 , vote_prob:  (1, 1.0) , label:  2 , data:  [ 4.9  2.5  4.5  1.7]
index:  9 , vote_prob:  (0, 1.0) , label:  0 , data:  [ 5.8  4.   1.2  0.2]
index:  10 , vote_prob:  (1, 1.0) , label:  1 , data:  [ 5.8  2.6  4.   1.2]
index:  11 , vote_prob:  (2, 1.0) , label:  2 , data:  [ 7.1  3.   5.9  2.1]
```

k-NN Terbobot

Dengan menggunkan metode sebelumnya kelas yang akan terpilih adalah kelas dengan voting paling banyak walaupun secara realita bisa saja kelas paling banyak tersebut mempunya jarak yang sebetulnya lebih jauh dengan kelas yang tidak terpilih sebagai contoh ketika menggunakan k=11, maka kelas yang terdekat suatu titik dperoleh 5 titik terdekat adalah A dan 6 titik lainnya

adalah B, apabila menggunakan vote maka sudah pasti yang terpilih adalah B walaupun sebenarnya A lebih dekat. Cara yang dapat dilakukan adalah dengan menggunakan bobot seri harmonic :

$$\sum 1/(i+1)=1+1/2+1/3+...+1/k \quad dimana\ i=1\ sampai\ k$$

Berikut ini adalah kode programnya:

```
def vote_harmonic_weights(neighbors, all_results=True):
    class_counter = Counter()
    number_of_neighbors = len(neighbors)
    for index in range(number_of_neighbors):
        class_counter[neighbors[index][2]] += 1/(index+1)
    labels, votes = zip(*class_counter.most_common())
    #print(labels, votes)
    winner = class_counter.most_common(1)[0][0]
    votes4winner = class_counter.most_common(1)[0][1]
    if all_results:
        total = sum(class_counter.values(), 0.0)
        for key in class_counter:
            class_counter[key] /= total
        return winner, class_counter.most_common()
    else:
        return winner, votes4winner / sum(votes)

for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              6,
                              distance=distance)
    print("index: ", i,
          ", result of vote: ",
          vote_harmonic_weights(neighbors,
                                all_results=True))
```

Output program adalah sebagai berikut:

```
index:  0 , result of vote:  (1, [(1, 1.0)])
index:  1 , result of vote:  (2, [(2, 1.0)])
index:  2 , result of vote:  (1, [(1, 1.0)])
index:  3 , result of vote:  (1, [(1, 1.0)])
```

```
index:  4 , result of vote:  (2, [(2, 0.9319727891156463), (1, 0.06802721088435375)])
index:  5 , result of vote:  (2, [(2, 0.8503401360544217), (1, 0.14965986394557826)])
index:  6 , result of vote:  (0, [(0, 1.0)])
index:  7 , result of vote:  (1, [(1, 1.0)])
index:  8 , result of vote:  (1, [(1, 1.0)])
index:  9 , result of vote:  (0, [(0, 1.0)])
index:  10 , result of vote:  (1, [(1, 1.0)])
index:  11 , result of vote:  (2, [(2, 1.0)])
```

Pembobotan sebelumnya hanya berdasarkan ranking dari jarak. Kita dapat memperbaikinya dengan fungsi voting baru:

```
def vote_distance_weights(neighbors, all_results=True):
    class_counter = Counter()
    number_of_neighbors = len(neighbors)
    for index in range(number_of_neighbors):
        dist = neighbors[index][1]
        label = neighbors[index][2]
        class_counter[label] += 1 / (dist**2 + 1)
    labels, votes = zip(*class_counter.most_common())
    #print(labels, votes)
    winner = class_counter.most_common(1)[0][0]
    votes4winner = class_counter.most_common(1)[0][1]
    if all_results:
        total = sum(class_counter.values(), 0.0)
        for key in class_counter:
            class_counter[key] /= total
        return winner, class_counter.most_common()
    else:
        return winner, votes4winner / sum(votes)

for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              6,
                              distance=distance)
    print("index: ", i,
          ", result of vote: ", vote_distance_weights(neighbors,
                                                      all_results=True))
```

Output dari program di atas adalah:

```
index:  0 , result of vote:  (1, [(1, 1.0)])
```

```
index:   1 , result of vote:   (2, [(2, 1.0)])
index:   2 , result of vote:   (1, [(1, 1.0)])
index:   3 , result of vote:   (1, [(1, 1.0)])
index:   4 , result of vote:   (2, [(2, 0.84901545921183608), (1, 0.15098454078816387)])
index:   5 , result of vote:   (2, [(2, 0.67361374621844783), (1, 0.32638625378155212)])
index:   6 , result of vote:   (0, [(0, 1.0)])
index:   7 , result of vote:   (1, [(1, 1.0)])
index:   8 , result of vote:   (1, [(1, 1.0)])
index:   9 , result of vote:   (0, [(0, 1.0)])
index:   10 , result of vote:   (1, [(1, 1.0)])
index:   11 , result of vote:   (2, [(2, 1.0)])
```

Contoh k-NN yang lain:

```
train_set = [(1, 2, 2),
                    (-3, -2, 0),
                    (1, 1, 3),
                    (-3, -3, -1),
                    (-3, -2, -0.5),
                    (0, 0.3, 0.8),
                    (-0.5, 0.6, 0.7),
                    (0, 0, 0)
                   ]labels = ['apple',    'banana', 'apple',
               'banana', 'apple', "orange",
               'orange', 'orange']k = 1for test_instance in [(0, 0, 0), (2, 2, 2),
                              (-3, -1, 0), (0, 1, 0.9),
                              (1, 1.5, 1.8), (0.9, 0.8, 1.6)]:
      neighbors = get_neighbors(train_set,
                                          labels,
                                          test_instance,
                                          2)
      print("vote distance weights: ", vote_distance_weights(neighbors))
```

Hasil python di atas menghasilkan:

```
vote distance weights:   ('orange', [('orange', 1.0)])
vote distance weights:   ('apple', [('apple', 1.0)])
vote distance weights:   ('banana', [('banana', 0.52941176470588236), ('apple',
0.47058823529411764)])
vote distance weights:   ('orange', [('orange', 1.0)])
vote distance weights:   ('apple', [('apple', 1.0)])
vote distance weights:   ('apple', [('apple', 0.50847457627118653), ('orange',
0.49152542372881353)])
```

k-NN dalam Linguistik

Algoritme ini dapat digunakan untuk mengenali kata yang mispelled. Kita menggunakan modul dari Levenshtein Distance.

```
def levenshtein(s,t):
    if s == "":
        return len(t)
    if t == "":
        return len(s)
    if s[-1] == t[-1]:
        cost = 0
    else:
        cost = 1

    res = min([levenshtein(s[:-1], t)+1,
               levenshtein(s, t[:-1])+1,
               levenshtein(s[:-1], t[:-1]) + cost])
    return res

cities = []
with open("/home/am86/piton/city_names.txt") as fh:
    for line in fh:
        city = line.strip()
        if " " in city:
            # like Freiburg im Breisgau add city only as well
            cities.append(city.split()[0])
        cities.append(city)
#cities = cities[:20]
for city in ["Freiburg", "Frieburg", "Freiborg",
             "Hamborg", "Sahrluis"]:
    neighbors = get_neighbors(cities,
                              cities,
                              city,
                              2,
                              distance=levenshtein)
    print("vote_distance_weights: ", vote_distance_weights(neighbors))
```

Hasil ouputnya:

```
vote_distance_weights:  ('Freiburg', [('Freiburg', 0.6666666666666666), ('Freiberg',
0.3333333333333333)])
```

vote_distance_weights: ('Freiburg', [('Freiburg', 0.6666666666666666), ('Lüneburg', 0.3333333333333333)])
vote_distance_weights: ('Freiburg', [('Freiburg', 0.5), ('Freiberg', 0.5)])
vote_distance_weights: ('Hamburg', [('Hamburg', 0.7142857142857143), ('Bamberg', 0.2857142857142857)])
vote_distance_weights: ('Saarlouis', [('Saarlouis', 0.8387096774193549), ('Bayreuth', 0.16129032258064516)])