Langkah-langkah menerapkan algoritma Nearest Neighbor menggunakan Python

A. Tujuan

Menerapkan algoritma Nearest Neighbor untuk mengklasifikasikan dataset bunga Iris (Iris Flower Species Dataset) tanpa menggunakan library. Kegiatan dalam modul ini adalah:

1. Membuat kode yang menerapkan algoritma k-Nearest Neighbor langkah per langkah.
2. Mengevaluasi k-Nearest Neighbor pada real dataset.
3. Melakukan prediksi data baru menggunakan k-Nearest Neighbor.

B. Dataset Iris Flower Species

Dalam modul ini kita akan menggunakan Set Data Spesies Bunga Iris.

Iris Flower Species Dataset mengelompokkan dan memprediksi spesies bunga berdasarkan ukuran dari bagian kelopak (sepal) dan mahkota (petal) bunga iris menjadi beberapa kelas (multiclass classification problem). Jumlah data untuk setiap kelas seimbang. Terdapat 150 observasi dengan 4 variabel input dan 1 variabel output. Nama variabelnya adalah sebagai berikut:

- Sepal length in cm.
- Sepal width in cm.
- Petal length in cm.
- Petal width in cm.
- Class

Contoh dari 5 baris data pertama adalah sebagai berikut:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
...
```

Diketahui bahwa baseline performance untuk data ini sekitar 33%.

C. Langkah-langkah menerapkan algoritma k-Nearest Neighbor menggunakan Python

Langkah-langkah berikut ini adalah langkah dasar yang diperlukan untuk mengimplementasikan k-Nearest Neighbor dari awal yang akan diterapkan pada masalah pemodelan prediktif.

Langkah 1: Perhitungan Euclidean Distance.

Langkah 2: Menentukan tetangga terdekat (Nearest Neighbor).

Langkah 3: Membuat prediksi.

Langkah-langkah diatas merupakan dasar penerapan algoritma k-Nearest Neighbours untuk masalah klasifikasi dan regresi.

Catatan: Koding bisa menggunakan Python 3 atau Python 2.7 tanpa ada perubahan.

Langkah 1: Perhitungan Euclidean Distance.

Diawali dengan menghitung jarak antara dua baris data dalam dataset dengan rumus Euclidean Distance. Baris data direpresentasikan dalam angka riil. Cara mudah untuk menghitung jarak antara dua baris atau vektor angka adalah dengan menggambar garis lurus dalam ruang 2D atau 3D. Jarak Euclidean dihitung sebagai akar kuadrat dari jumlah selisih kuadrat antara dua vector seperti pada formula berikut ini:

$$d(x1, x2) = \sqrt{\sum_{i=1}^{n}(x1_i - x2_i)^2}$$

di mana x1 adalah baris data pertama, x2 adalah data baris kedua dan i adalah indeks untuk kolom tertentu saat kita menjumlahkan di semua kolom.

Semakin kecil nilai jarak Euclidean, artinya semakin mirip dua data tersebut. Nilai 0 berarti tidak ada perbedaan antara dua data tersebut. Di bawah ini adalah fungsi bernama ***euclidean_distance()*** untuk menghitung jarak dua buah data.

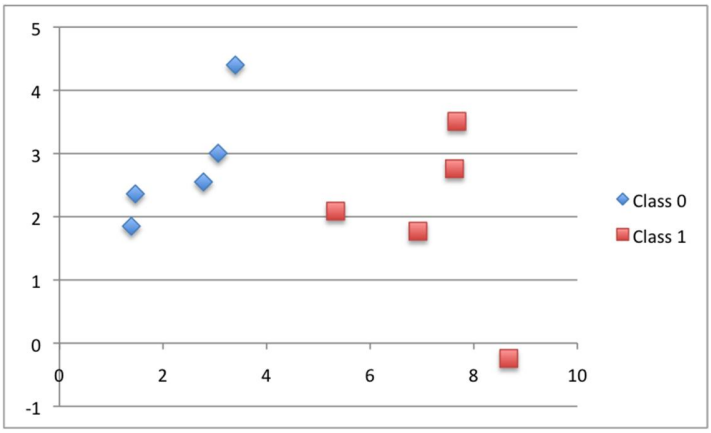
```
# Example of calculating Euclidean distance
from math import sqrt

# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):
    distance += (row1[i] - row2[i])**2
  return sqrt(distance)
```

Dapat dilihat bahwa fungsi tersebut mengasumsikan bahwa kolom terakhir di setiap baris adalah nilai kelas label yang harus diabaikan dari penghitungan jarak.

Fungsi perhitugan jarak ***euclidean_distance()*** ini akan kita ujikan pada data kecil dibawah ini.

X1	X2	Y
2.7810836	2.550537003	0
1.465489372	2.362125076	0
3.396561688	4.400293529	0
1.38807019	1.850220317	0
3.06407232	3.005305973	0
7.627531214	2.759262235	1
5.332441248	2.088626775	1
6.922596716	1.77106367	1
8.675418651	-0.242068655	1
7.673756466	3.508563011	1

Hasil scatter plot dataset kecil diatas dalam ruang dua dimensi adalah sebagai berikut:



Selanjutnya, aplikasikan fungsi ***euclidean_distance()*** pada seluruh data kecil tersebut dengan mencetak jarak antara baris pertama dan semua baris lainnya. Nampak jarak antara baris pertama dan baris itu sendiri adalah 0.

```
# Test distance function
dataset = [[2.7810836,2.550537003,0],
  [1.465489372,2.362125076,0],
  [3.396561688,4.400293529,0],
  [1.38807019,1.850220317,0],
  [3.06407232,3.005305973,0],
  [7.627531214,2.759262235,1],
  [5.332441248,2.088626775,1],
  [6.922596716,1.77106367,1],
  [8.675418651,-0.242068655,1],
  [7.673756466,3.508563011,1]]
row0 = dataset[0]
for row in dataset:
  distance = euclidean_distance(row0, row)
  print(distance)
```

Setelah kode diatas dijalankan, program mencetak jarak antara baris pertama dan setiap baris dalam kumpulan data, termasuk dirinya sendiri (bernilai 0.0).

```
0.0
1.3290173915275787
1.9494646655653247
1.5591439385540549
0.5356280721938492
4.850940186986411
2.592833759950511
4.214227042632867
6.522409988228337
4.985585382449795
```

Selanjutnya, fungsi ***euclidean_distance()*** kita gunakan sebagai perhitungan jarak untuk menemukan tetangga dalam dataset.

Langkah 2: Menentukan tetangga terdekat (Nearest Neighbor).

Tetangga dari sebuah data baru dalam sebuah dataset adalah sejumlah k data terdekat dengan data baru tersebut. Untuk menemukan tetangga dari data baru dalam sebuah dataset, pertama-tama kita harus menghitung jarak antara setiap data dalam dataset dengan data baru. Kita gunakan fungsi jarak ***euclidean_distance()*** yang telah dibuat di langkah 1.

Setelah jarak dihitung, kita harus mengurutkan data dalam dataset secara ascending berdasarkan jarak yang ditemukan. Kemudian kita pilih k buah data teratas (k buah nilai jarak terkecil) sebagai tetangga yang paling mirip.

Kita dapat melakukan kegiatan ini dengan cara menyimpan jarak untuk setiap data dalam dataset sebagai tupel, mengurutkan daftar tupel berdasarkan jarak (dalam urutan ascending) dan kemudian mengambil tetangga terdekat (jarak terkecil).

Berikut ini adalah fungsi bernama ***get_neighbours()*** yang mengimplementasikan maksud ini.

```
# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
  distances.sort(key=lambda tup: tup[1])
  neighbors = list()
  for i in range(num_neighbors):
    neighbors.append(distances[i][0])
  return neighbors
```

Perhatikan bahwa fungsi ***euclidean_distance()*** digunakan untuk menghitung jarak antara setiap data dalam dataset dan data baru.

Daftar tupel data dalam dataset (*train_row*) dan jarak diurutkan dimana kunci khusus digunakan untuk memastikan bahwa item kedua dalam tupel v digunakan dalam operasi pengurutan. Di akhir fungsi, daftar *num_neighbours* yang paling mirip dengan *test_row* dikembalikan. Selanjutnya fungsi ini kita uji dengan dataset kecil yang telah dibuat sebelumnya. Seperti pada kode dibawah ini:

```
# Test distance function
dataset = [[2.7810836,2.550537003,0],
  [1.465489372,2.362125076,0],
  [3.396561688,4.400293529,0],
  [1.38807019,1.850220317,0],
  [3.06407232,3.005305973,0],
  [7.627531214,2.759262235,1],
  [5.332441248,2.088626775,1],
  [6.922596716,1.77106367,1],
  [8.675418651,-0.242068655,1],
  [7.673756466,3.508563011,1]]
neighbors = get neighbors(dataset, dataset[0], 3)
for neighbor in neighbors:
  print(neighbor)
```

Menjalankan fungsi-fungsi tersebut pada data kecil akan mencetak 3 data yang paling mirip dengan data baru. Seperti yang diharapkan, data pertama adalah yang paling mirip dengan data baru dan berada di urutan teratas daftar.

```
[2.7810836, 2.550537003, 0]
[3.06407232, 3.005305973, 0]
[1.465489372, 2.362125076, 0]
```

Sampai pada tahap ini kita tahu bagaimana mendapatkan tetangga dari dataset. Cara ini bisa kita gunakan untuk membuat prediksi.

Langkah 3: Membuat Prediksi.

Tetangga paling mirip yang dipilih dari dataset pelatihan dapat digunakan untuk membuat prediksi. Dalam permasalahan klasifikasi, kita bisa mengembalikan kelas yang paling terwakili di antara tetangga. Kita dapat menyelesaikan permasalahan ini dengan menggunakan fungsi *max()* pada daftar nilai keluaran dari neighbors Di bawah ini adalah fungsi bernama ***predict_classification()*** yang mengimplementasikan kegiatan ini.

```
# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
  neighbors = get neighbors(train, test row, num neighbors)
  output_values = [row[-1] for row in neighbors]
  prediction = max(set(output_values), key=output_values.count)
  return prediction
```

Tes fungsi ***predict_classification()*** pada data kecil dengan kode berikut ini:

```
# Test distance function
dataset = [[2.7810836,2.550537003,0],
	[1.465489372,2.362125076,0],
	[3.396561688,4.400293529,0],
	[1.38807019,1.850220317,0],
	[3.06407232,3.005305973,0],
	[7.627531214,2.759262235,1],
	[5.332441248,2.088626775,1],
	[6.922596716,1.77106367,1],
	[8.675418651,-0.242068655,1],
	[7.673756466,3.508563011,1]]
prediction = predict classification(dataset, dataset[0], 3)
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

Hasil menjalankan fungsi ini adalah tampilan klasifikasi yang diharapkan, yaitu kelas 0 dan klasifikasi aktual yang diprediksi berdasarkan 3 tetangga paling mirip dalam dataset, yaitu 0.

```
Expected 0, Got 0.
```

Fungsi-fungsi yang telah dibuat diatas merupakan fungsi-fungsi yang dibutuhkan dalam prediksi kelas data baru dengan KNN. Selanjutnya kita gunakan fungsi-fungsi ini kedalam Iris Flower Species Dataset.

D. Iris Flower Species Case Study

Bagian ini menerapkan algoritma k-Nearest Neighbor ke Iris Flower Species Dataset.

Langkah pertama adalah load dataset dan mengubah data menjadi angka. Untuk ini, digunakan fungsi ***load_csv()*** untuk load file, ***str_column_to_float()*** untuk mengubah string menjadi float dan ***str_column_to_int()*** untuk mengubah kolom kelas menjadi nilai integer.

Metrik evaluasi yang digunakan adalah *k-fold cross-validation* dengan k benilai 5. Sehingga tiap fold terdiri dari 150/5 = 30 record. Fungsi bantuan bernama ***evaluate_algorithm()*** untuk mengevaluasi algoritma dengan validasi silang dan ***accuracy_metric***() untuk menghitung akurasi prediksi.

Fungsi baru lainnya bernama ***k_nearest_neighbors()*** akan dibuat sebagai penerapan algoritma k-Nearest Neighbor, pertama-tama menghitung statistik dari set data pelatihan dan menggunakannya untuk membuat prediksi untuk set data pengujian.

Berikut ini adalah kode lengkapnya:

```
# k-nearest neighbors on the Iris Flowers Dataset
from random import seed
from random import randrange
from csv import reader
from math import sqrt


# Load a CSV file
def load_csv(filename):
  dataset = list()
  with open(filename, 'r') as file:
    csv_reader = reader(file)
    for row in csv_reader:
      if not row:
        continue
      dataset.append(row)
  return dataset


# Convert string column to float
def str_column_to_float(dataset, column):
  for row in dataset:
    row[column] = float(row[column].strip())


# Convert string column to integer
def str_column_to_int(dataset, column):
  class_values = [row[column] for row in dataset]
  unique = set(class_values)
  lookup = dict()
  for i, value in enumerate(unique):
    lookup[value] = i
  for row in dataset:
    row[column] = lookup[row[column]]
  return lookup


# Find the min and max values for each column
def dataset_minmax(dataset):
  minmax = list()
  for i in range(len(dataset[0])):
    col_values = [row[i] for row in dataset]
    value_min = min(col_values)
    value_max = max(col_values)
    minmax.append([value_min, value_max])
  return minmax
```

```
# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
  for row in dataset:
    for i in range(len(row)):
      row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])


# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
  dataset_split = list()
  dataset_copy = list(dataset)
  fold_size = int(len(dataset) / n_folds)
  for _ in range(n_folds):
    fold = list()
    while len(fold) < fold_size:
      index = randrange(len(dataset_copy))
      fold.append(dataset_copy.pop(index))
    dataset_split.append(fold)
  return dataset_split


# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
  correct = 0
  for i in range(len(actual)):
    if actual[i] == predicted[i]:
      correct += 1
  return correct / float(len(actual)) * 100.0


# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
  folds = cross_validation_split(dataset, n_folds)
  scores = list()
  for fold in folds:
    train_set = list(folds)
    train_set.remove(fold)
    train_set = sum(train_set, [])
    test_set = list()
    for row in fold:
      row_copy = list(row)
      test_set.append(row_copy)
      row_copy[-1] = None
    predicted = algorithm(train_set, test_set, *args)
    actual = [row[-1] for row in fold]
    accuracy = accuracy_metric(actual, predicted)
    scores.append(accuracy)
  return scores
```

```
# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):
    distance += (row1[i] - row2[i])**2
  return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
  distances.sort(key=lambda tup: tup[1])
  neighbors = list()
  for i in range(num_neighbors):
    neighbors.append(distances[i][0])
  return neighbors

# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
  neighbors = get_neighbors(train, test_row, num_neighbors)
  output_values = [row[-1] for row in neighbors]
  prediction = max(set(output_values), key=output_values.count)
  return prediction

# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
  predictions = list()
  for row in test:
    output = predict_classification(train, row, num_neighbors)
    predictions.append(output)
  return(predictions)
```

```
# Test the kNN on the Iris Flowers dataset
seed(1)
filename = 'iris.txt'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
  str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# evaluate algorithm
n_folds = 5
num_neighbors = 5
scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds, num_
neighbors)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

Menjalankan kode diatas menghasilkan skor akurasi klasifikasi rata-rata pada setiap validasi silang (5 skor) dan juga skor akurasi rata-rata. Akurasi rata-rata yang dihasilkan sekitar 96.6%, skor ini jauh lebih baik daripada skor akurasi baseline sebesar 33%. Perhatikan bahwa akurasi 96.6% ini adalah akurasi tahap pelatihan, yaitu akurasi saat model berlatih mengenali pola data dari keseluruhan dataset training. Sampai pada tahap ini belum dilakukan prediksi pada data baru.

```
Scores: [96.66666666666667, 96.66666666666667, 100.0, 90.0, 100.0]
Mean Accuracy: 96.667%
```

Kita gunakan training dataset untuk membuat prediksi pada data baru dengan memanggil fungsi ***predict_classification(),*** dengan argument *row* adalah data baru yang akan direndiksi label kelas nya.

```
...
# predict the label
label = predict_classification(dataset, row, num_neighbors)
```

Akan lebih informatif jika info nama label kelas (string) hasil prediksi ditampilkan. Kita bisa mengupdate fungsi ***str_column_to_int()*** untuk mencetak pemetaan nama kelas string menjadi integer sehingga kita bisa mengetahui prediksi yang dihasilkan. Berikut ini adalah modifikasi fungsi ***str_column_to_int()*** yang dimaksud:

```
# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
        print('[%s] => %d' % (value, i))
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup
```

Berikut ini adalah kode lengkap dalam menggunakan k-Nearest Neighbor pada seluruh dataset dan membuat prediksi terhadap satu buah data baru:

```
# Make Predictions with k-nearest neighbors on the Iris Flowers Dataset
from csv import reader
from math import sqrt

# Load a CSV file
def load_csv(filename):
  dataset = list()
  with open(filename, 'r') as file:
    csv_reader = reader(file)
    for row in csv_reader:
      if not row:
        continue
      dataset.append(row)
  return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
  for row in dataset:
    row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
  class_values = [row[column] for row in dataset]
  unique = set(class_values)
  lookup = dict()
  for i, value in enumerate(unique):
    lookup[value] = i
    print('[%s] => %d' % (value, i))
  for row in dataset:
    row[column] = lookup[row[column]]
  return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
  minmax = list()
  for i in range(len(dataset[0])):
    col_values = [row[i] for row in dataset]
    value_min = min(col_values)
    value_max = max(col_values)
    minmax.append([value_min, value_max])
  return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
  for row in dataset:
    for i in range(len(row)):
      row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
```

```
# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):
    distance += (row1[i] - row2[i])**2
  return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
  distances.sort(key=lambda tup: tup[1])
  neighbors = list()
  for i in range(num_neighbors):
    neighbors.append(distances[i][0])
  return neighbors

# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
  neighbors = get_neighbors(train, test_row, num_neighbors)
  output_values = [row[-1] for row in neighbors]
  prediction = max(set(output_values), key=output_values.count)
  return prediction

# Make a prediction with KNN on Iris Dataset
filename = 'iris.txt'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
  str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# define model parameter
num_neighbors = 5
# define a new record
row = [5.7,2.9,4.2,1.3]
# predict the label
label = predict_classification(dataset, row, num_neighbors)
print('Data=%s, Predicted: %s' % (row, label))
```

Hasil prediksi menyatakan bahwa data baru adalah milik kelas 1 yaitu “Iris-setosa”. Berikut ini adalah output running program diatas:

```
[Iris-setosa] => 0
[Iris-versicolor] => 1
[Iris-virginica] => 2
Data=[5.7, 2.9, 4.2, 1.3], Predicted: 1
```