

MODUL PRAKTIKUM KECERDASAN BUATAN

PERTEMUAN 3 – KETIDAKPASTIAN

Tools : Jupyter notebook

Bahasa Pemrograman : Python

Dalam praktikum kali ini kita akan belajar tentang bagaimana cara membuat/membangun Naive Bayes *Classifier* dari awal menggunakan Python serta penerapannya untuk melakukan klasifikasi pada dataset bunga iris.

Algoritma Naive Bayes

Teorema Bayes memungkinkan kita untuk menghitung nilai probabilitas keanggotaan dari potongan data terhadap kelas tertentu berdasarkan pengetahuan yang diperoleh sebelumnya. Adapun teorema Bayes dapat dinyatakan dalam persamaan berikut:

$$P(\text{class}|\text{data}) = (P(\text{data}|\text{class}) \times P(\text{class})) / P(\text{data})$$

dimana $P(\text{class}|\text{data})$ merupakan nilai probabilitas suatu kelas dari data yang diberikan.

Naive Bayes adalah algoritma klasifikasi untuk masalah klasifikasi biner (dua kelas) dan multikelas.

Alasan disebut Naive Bayes karena perhitungan probabilitas untuk setiap kelas disederhanakan agar perhitungannya dapat dilakukan. Dimana masing-masing atribut dianggap independen dan tidak berinteraksi satu sama lain.

Dataset Bunga Iris

Dataset bunga Iris melibatkan prediksi jenis spesies berdasarkan atribut ukuran dari bunga tersebut.

Terdapat 5 atribut dalam dataset bunga Iris, yaitu:

1. Sepal length in cm.
2. Sepal width in cm.
3. Petal length in cm.
4. Petal width in cm.
5. Class

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	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	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa

Pengembangan Algoritma Naive Bayes

Pertama kita akan mengembangkan setiap bagian dari algoritma Naive Bayes, kemudian kita akan mengikat semua elemen menjadi implementasi kerja yang diterapkan pada dataset nyata di bagian berikutnya.

Terdapat 5 langkah utama dalam pengembangan Algoritma Naive Bayes:

1. Memisahkan data berdasarkan kelasnya
2. Meringkas dataset
3. Meringkas data berdasarkan kelas
4. Mendefinisikan fungsi kepadatan probabilitas Gaussian
5. Menentukan nilai probabilitas kelas

Langkah 1: Memisahkan data berdasarkan kelasnya

Pada tahap ini, kita akan memisahkan data berdasarkan kelasnya. Kita dapat membuat kamus obyek dengan sebuah nilai kunci berupa nilai kelas kemudian diikuti dengan menambahkan semua nilai dari record sesuai dengan nilai kelasnya. Adapun penerapan menggunakan python adalah sebagai berikut:

```
In [24]: # Example of separating data by class value
# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated

# Test separating data by class
dataset = [[3.393533211,2.331273381,0],
           [3.110073483,1.781539638,0],
           [1.343808831,3.368360954,0],
           [3.582294042,4.67917911,0],
           [2.280362439,2.866990263,0],
           [7.423436942,4.696522875,1],
           [5.745051997,3.533989803,1],
           [9.172168622,2.511101045,1],
           [7.792783481,3.424088941,1],
           [7.939820817,0.791637231,1]]
separated = separate_by_class(dataset)
for label in separated:
    print(label)
    for row in separated[label]:
        print(row)
```

Dataset terdiri atas 10 data yang dibuat secara manual dengan dua kelas (kelas 0 dan kelas 1) dengan rincian atribut sebagai berikut:

X1 (fitur 1)	X2 (fitur 2)	Y(kelas)
3.393533211,2.331273381,0,	[3.110073483,1.781539638,0],	
[1.343808831,3.368360954,0],		
[3.582294042,4.67917911,0],		
[2.280362439,2.866990263,0],		
[7.423436942,4.696522875,1],		
[5.745051997,3.533989803,1],		
[9.172168622,2.511101045,1],		
[7.792783481,3.424088941,1],		
[7.939820817,0.791637231,1]		

```
[7.939820817,0.791637231,1]]
```

Output

```
0
[3.393533211, 2.331273381, 0]
[3.110073483, 1.781539638, 0]
[1.343808831, 3.368360954, 0]
[3.582294042, 4.67917911, 0]
[2.280362439, 2.866990263, 0]
1
[7.423436942, 4.696522875, 1]
[5.745051997, 3.533989803, 1]
[9.172168622, 2.511101045, 1]
[7.792783481, 3.424088941, 1]
[7.939820817, 0.791637231, 1]
```

Langkah 2: Meringkas dataset

Pada tahap ini, kita akan melakukan dua perhitungan statistik terhadap data yang diberikan, yaitu: nilai rata-rata (mean) dan standar deviasi. Dua perhitungan statistik tersebut diperlukan untuk menghitung probabilitas.

Adapun cara menghitung rata-rata dapat dilihat pada persamaan berikut ini:

$$\text{mean} = \sum(x)/n \times \text{count}(x)$$

dimana x merupakan daftar nilai atau kolom yang kita cari.

Sedangkan cara menghitung standar deviasi dapat dilihat pada persamaan berikut ini:

$$\text{standard deviation} = \sqrt{\sum_{i=1}^N (x_i - \text{mean}(x))^2 / N-1}$$

Penerapan langkah ini menggunakan python adalah sebagai berikut:

```
In [3]: # Example of summarizing a dataset
from math import sqrt

# Calculate the mean of a list of numbers
def mean(numbers):
    return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a list of numbers
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)

# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
    # the use of the zip() function that will aggregate elements from each provided argument.
    # We pass in the dataset to the zip() function with the * operator that separates the dataset (that is a list of lists)
    # into separate lists for each row.
    # The zip() function then iterates over each element of each row and returns a column
    # from the dataset as a list of numbers.
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries
|
# Test summarizing a dataset
dataset = [[3.393533211,2.331273381,0],
           [3.110073483,1.781539638,0],
           [1.343808831,3.368360954,0],
           [3.582294042,4.67917911,0],
           [2.280362439,2.866990263,0],
           [7.423436942,4.696522875,1],
           [5.745051997,3.533989803,1],
           [9.172168622,2.511101045,1],
           [7.792783481,3.424088941,1],
           [7.939820817,0.791637231,1]]
summary = summarize_dataset(dataset)
print(summary)
```

Output

```
[ (5.17833386499999, 2.7665845055177263, 10), (2.9984683241, 1.218556343617447, 10) ]
```

Output di atas menunjukkan bahwa bahwa nilai rata-rata X1 adalah 5,17833386499999 dan standar deviasi X1 adalah 2,7665845055177263.

Langkah 3: Meringkas data berdasarkan kelas

Sebelumnya kita telah mengembangkan fungsi "separate_by_class()" untuk memisahkan kumpulan data menjadi baris berdasarkan kelas dan fungsi "summary_dataset()" untuk menghitung statistik ringkasan dari setiap kolom.

Berikutnya kita bisa meringkas kolom dalam kumpulan data yang diatur berdasarkan nilai kelas melalui fungsi "summary_by_class()". Pertama-tama, dataset dibagi berdasarkan kelas, kemudian statistik dihitung pada setiap subset. Hasil berupa daftar tupel statistik kemudian disimpan dalam kamus berdasarkan nilai kelasnya.

Penerapan langkah ini menggunakan python adalah sebagai berikut:

```
In [5]: # Example of summarizing data by class value
from math import sqrt

# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated

# Calculate the mean of a list of numbers
def mean(numbers):
    return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a list of numbers
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)

# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
```

```

# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
        summaries[class_value] = summarize_dataset(rows)
    return summaries

# Test summarizing by class
dataset = [[3.393533211, 2.331273381, 0],
           [3.110073483, 1.781539638, 0],
           [1.343808831, 3.368360954, 0],
           [3.582294042, 4.67917911, 0],
           [2.280362439, 2.866990263, 0],
           [7.423436942, 4.696522875, 1],
           [5.745051997, 3.533989803, 1],
           [9.172168622, 2.511101045, 1],
           [7.792783481, 3.424088941, 1],
           [7.939820817, 0.791637231, 1]]
summary = summarize_by_class(dataset)
for label in summary:
    print(label)
    for row in summary[label]:
        print(row)

```

Output

```

0
(2.7420144012, 0.9265683289298018, 5)
(3.0054686692, 1.1073295894898725, 5)
1
(7.6146523718, 1.2344321550313704, 5)
(2.9914679790000003, 1.4541931384601618, 5)

```

Output di atas menunjukkan bahwa nilai rata-rata X1 pada kelas 0 adalah 2.7420144012 dan standar deviasi X1 pada kelas 0 adalah 0.9265683289298018.

Langkah 4: Mendefinisikan fungsi kepadatan probabilitas Gaussian

Menghitung probabilitas nilai riil tertentu seperti X1 itu sulit. Salah satu cara kita dapat melakukannya adalah dengan mengasumsikan bahwa nilai X1 diambil dari suatu distribusi, seperti distribusi Gaussian.

Distribusi Gaussian dapat diringkas hanya dengan menggunakan dua angka: rata-rata dan standar deviasi. Oleh karena itu, dengan sedikit fungsi matematika, kita dapat memperkirakan probabilitas dari suatu nilai tertentu. Fungsi matematika yang dimaksud adalah Fungsi Distribusi Probabilitas Gaussian (atau PDF Gaussian) dan dapat dihitung sebagai berikut:

$$f(x) = (1 / \sqrt{2 \times \pi}) \times \exp(-(x - \text{mean})^2 / (2 \times \sigma^2))$$

Dimana sigma adalah standar deviasi untuk x , rata-rata adalah rata-rata untuk x dan PI adalah nilai pi.

Penerapan langkah ini menggunakan python adalah sebagai berikut:

```

In [1]: # Example of Gaussian PDF
from math import sqrt
from math import pi
from math import exp

# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2)))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent

# Test Gaussian PDF
print(calculate_probability(1.0, 1.0, 1.0))
print(calculate_probability(2.0, 1.0, 1.0))
print(calculate_probability(0.0, 1.0, 1.0))

```

Output

```
0.3989422804014327
0.24197072451914337
0.24197072451914337
```

Output tersebut menunjukkan bahwa ketika nilai rata-ratanya 1 dan nilai standar deviasinya adalah 1 maka memiliki probabilitas 0,39. Sedangkan untuk nilai standar deviasi 1 dan nilai rata-ratanya 2 ataupun 0 memiliki probabilitas yang sama yaitu 0,24

Langkah 5: Menentukan nilai probabilitas kelas

Probabilitas dihitung secara terpisah untuk setiap kelas. Ini berarti bahwa pertama-tama kita menghitung probabilitas dari data baru termasuk dalam kelas pertama, kemudian menghitung probabilitas dari data baru termasuk dalam kelas kedua, dan seterusnya untuk semua kelas. Adapun cara perhitungan nilai probabilitas dapat dilihat pada persamaan berikut ini:

$$P(\text{kelas}|\text{data}) = (P(\text{data}|\text{kelas}) \times P(\text{kelas})) / P(\text{data})$$

dimana $P(\text{kelas}|\text{data})$ merupakan nilai probabilitas suatu kelas dari data yang diberikan.

Nilai probabilitas tertinggi akan suatu kelas yang nantinya akan diambil sebagai hasil prediksi.

Sebagai contoh pada dataset di atas, di mana kita memiliki 2 variabel input, maka perhitungan probabilitas sebuah baris milik kelas 0 pertama dapat dihitung sebagai berikut:

$$P(\text{class}=0|X1, X2) = P(X1|\text{class}=0) \times P(X2|\text{class}=0) \times P(\text{class}=0)$$

Adapun penerapan langkah ini adalah sebagai berikut:

```
In [2]: # Example of calculating class probabilities
from math import sqrt
from math import pi
from math import exp

# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated

# Calculate the mean of a List of numbers
def mean(numbers):
    return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a List of numbers
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)
```

```

# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries

# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
        summaries[class_value] = summarize_dataset(rows)
    return summaries

# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
    exponent = exp(-(pow((x-mean), 2) / (2 * pow(stdev, 2))))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent

# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
        for i in range(len(class_summaries)):
            mean, stdev, _ = class_summaries[i]
            probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
    return probabilities

# Test calculating class probabilities
dataset = [[3.93533211, 2.331273381, 0],
           [3.110073483, 1.781539638, 0],
           [1.343808831, 3.368360954, 0],
           [3.582294042, 4.67917911, 0],
           [2.280362439, 2.866990263, 0],
           [7.423436942, 4.696522875, 1],
           [5.745051997, 3.533989893, 1],
           [9.172168622, 2.511101045, 1],
           [7.792783481, 3.424088941, 1],
           [7.939820817, 0.791637231, 1]]
summaries = summarize_by_class(dataset)
probabilities = calculate_class_probabilities(summaries, dataset[0])
print(probabilities)

```

Output

```
{0: 0.05032427673372076, 1: 0.00011557718379945765}
```

Kode di atas akan mencetak probabilitas yang dihitung untuk setiap kelas. Kita dapat melihat bahwa probabilitas baris pertama menjadi kelas 0 (0,0503) lebih tinggi daripada probabilitas baris pertama menjadi kelas 1 (0,0001). Karena itu dapat disimpulkan bahwa data tersebut merupakan kelas 0.

Latihan: Penerapan Algoritma Naive Bayes pada Dataset Bunga Iris (iris.csv)

Terapkan algoritma Naive Bayes pada dataset bunga Iris (iris.csv) sesuai dengan langkah-langkah berikut:

```

# Naive Bayes On The Iris Dataset
from csv import reader
from random import seed
from random import randrange
from math import sqrt
from math import exp
from math import pi

```

```

# 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

# 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

```

```
# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated
```

```
# Calculate the mean of a list of numbers
def mean(numbers):
    return sum(numbers)/float(len(numbers))
```

```
# Calculate the standard deviation of a list of numbers
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)
```

```
# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries
```

```
# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
        summaries[class_value] = summarize_dataset(rows)
    return summaries
```

```
# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent
```

```
# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
        for i in range(len(class_summaries)):
            mean, stdev, _ = class_summaries[i]
            probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
    return probabilities
```

```
# Predict the class for a given row
def predict(summaries, row):
    probabilities = calculate_class_probabilities(summaries, row)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label
```

```

# Naive Bayes Algorithm
def naive_bayes(train, test):
    summarize = summarize_by_class(train)
    predictions = list()
    for row in test:
        output = predict(summarize, row)
        predictions.append(output)
    return(predictions)

# Make a prediction with Naive Bayes on Iris Dataset
filename = 'iris.csv'
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)

# fit model
model = summarize_by_class(dataset)

# define a new record
row = [5.7,2.9,4.2,1.3]

# predict the label
label = predict(model, row)
print('Data=%s, Predicted: %s' % (row, label))

```

Jelaskan setiap tahapan pada kode diatas!
Apa *output* yang didapat dari kode di atas? Jelaskan!

Referensi:

<https://machinelearningmastery.com/>