JOBSHEET - 9
KECERDASAN BUATAN

Oleh:
FAREL PUTRA HIDAYAT
NIM. 2041723016

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG
NOVEMBER 2020

1. Pisahkan menurut kelas

```
In [1]: 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

        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],
                   [9.172167622,2.511101045,1],
                   [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)
```

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]
[9.172167622, 2.511101045, 1]
[7.792783481, 3.424088941, 1]
[7.939820817, 0.791637231, 1]
```

2. Meringkas Dataset

```
In [3]: from math import sqrt

        def mean(numbers):
            return sum(numbers)/float(len(numbers))

        def stdev(numbers):
            avg = mean(numbers)
            variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
            return sqrt(variance)

        def summarize_dataset(dataset):
            summaries = [(mean(column), stdev(column)) for column in zip(*dataset)]
            del(summaries[-1])
            return summaries

        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],
                   [9.172167622,2.511101045,1],
                   [7.792783481,3.424088941,1],
                   [7.939820817,0.791637231,1]]
        summary = summarize_dataset(dataset)
        print(summary)
```

Output:

```
[(5.1783332865, 2.766584345117987), (2.9984683241, 1.218556343617447)]
```

3. Meringkas Dataset Berdasarkan Kelas

```
In [3]: from math import sqrt

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

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)

def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries

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

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)
```

4. Fungsi Gaussian Probability Density

```
In [5]: from math import sqrt
        from math import pi
        from math import exp

        def calculate_probability(x, mean, stdev):
            exponent = exp(-((x-mean)**2 / (2 * stdev**2)))
            return (1 / (sqrt(2* pi) * stdev)) * exponent

        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
```

5. Probabilitas Kelas

In [4]:
```
from math import sqrt
from math import pi
from math import exp

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

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)

def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries

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

def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2)))
    return (1 / (sqrt(2* pi) * stdev)) * exponent

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

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]]
summaries = summarize_by_class(dataset)
probabilities = calculate_class_probabilities(summaries, dataset[0])
print(probabilities)
```

Output:

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

6. Studi Kasus Iris Flower Species
   a. Menerapkan algoritma Naive Beyes ke dataset bunga iris.

In [2]:
```
from csv import reader
from random import seed
from random import randrange
from math import sqrt
from math import exp
from math import pi

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

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

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
```

```
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

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

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
```

```
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

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)

def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries

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

def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent
```

```
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

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

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)

seed(1)
from sklearn.datasets import load_iris
filename =  'iris.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
str_column_to_int(dataset, len(dataset[0])-1)
n_folds = 5
scores = evaluate_algorithm(dataset, naive_bayes, n_folds)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

Output:

```
Scores: [93.33333333333333, 96.66666666666667, 100.0, 93.33333333333333, 93.33333333333333]
Mean Accuracy: 95.333%
```

b. Fitting model Naive Bayes pada seluruh Dataset dan membuat prediksi tunggal untuk pengamatan baru tercantum di bawah ini:

In [6]:
```
from csv import reader
from math import sqrt
from math import exp
from math import pi

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

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

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

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
```

```
def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)

def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries

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

def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent

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

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
```

```
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

filename = 'iris.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
str_column_to_int(dataset, len(dataset[0])-1)
model = summarize_by_class(dataset)
row = [5.7,2.9,4.2,1.3]
label = predict(model, row)
print('Data=%s, Predicted: %s' % (row, label))
```

Output:

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