LAPORAN PRAKTIKUM
KECERDASAN BUATAN
"KETIDAKPASTIAN"

NAMA : **AFRIDHO IKHSAN**
NPM : **2210631170002**
KELAS : **4A**

PROGRAM STUDI INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS SINGAPERBANGSA KARAWANG

2023

DAFTAR ISI

LATIHAN

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
```

```python
# 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
```

```python
# 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
```

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

```python
# 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)
```

```python
# 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
```

```python
# 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
```

```python
# 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
```

```python
# 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
```

```python
# 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!

JAWABAN

Berikut adalah penjelasan untuk setiap tahapan pada kode di atas:

- Impor fungsi pembantu dari beberapa modul

```
# 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
```

  a. from csv import reader: Mengimpor fungsi reader dari modul csv. Fungsi ini akan digunakan untuk membaca file CSV.
  b. from random import seed: Mengimpor fungsi seed dari modul random. Fungsi ini digunakan untuk menginisialisasi generator nomor acak.
  c. from random import randrange: Mengimpor fungsi randrange dari modul random. Fungsi ini digunakan untuk memilih angka acak dari rentang tertentu.
  d. from math import sqrt, exp, pi: Mengimpor fungsi sqrt, exp, dan konstanta pi dari modul math. Fungsi dan konstanta ini akan digunakan dalam perhitungan statistik dan probabilitas dalam algoritma Naive Bayes.

- Pembuatan fungsi untuk memuat File CSV

```
# 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
```

  Fungsi load_csv digunakan untuk membaca dataset dari file CSV yang telah diberikan dan mengembalikan list of lists yang berisi data dari file tersebut.

- Pembuatan fungsi untuk pemrosesan data

```
# 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
```

  Fungsi str_column_to_float digunakan untuk mengonversi kolom dataset dari string ke float. Sedangkan fungsi str_column_to_int digunakan untuk mengonversi kolom kelas (label) dari string ke integer.

- Pembuatan fungsi untuk pembagian Validasi Silang

```
# 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
```

Fungsi cross_validation_split membagi dataset menjadi beberapa bagian untuk keperluan validasi silang (cross-validation).

- Pembuatan fungsi untuk perhitungan akurasi metriks

```
# 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
```

Fungsi accuracy_metric menghitung akurasi prediksi dengan membandingkan nilai aktual dengan nilai prediksi.

- Pembuatan fungsi untuk mengevaluasi Algoritma

```
# 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
```

Fungsi evaluate_algorithm mengevaluasi algoritma menggunakan cross-validation split.

- Pembuatan fungsi untuk pemisahan data berdasarkan class

```
# 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
```

Fungsi separate_by_class memisahkan dataset menjadi subset berdasarkan nilai kelas (label).

- Pembuatan fungsi-fungsi yang diperlukan untuk meringkas kumpulan data

```
# 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
```

Fungsi yang diperlukan diantaranya adalah fungsi untuk perhitungan mean, standar deviasi, dan pada akhirnya digunakan pada fungsi summarize_dataset untuk meringkas data-data numerik pada setiap kolom menjadi data mengenai rata-rata, deviasi standar, dan jumlah datanya saja.

- Pembuatan fungsi untuk peringkasan berdasarkan class

```
# 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
```

Fungsi summarize_by_class menghitung statistik (rata-rata, deviasi standar, jumlah data) untuk setiap kolom berdasarkan kelasnya.

- Pembuatan fungsi untuk perhitungan probabilitas

```
# 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
```

Fungsi calculate_probability menghitung distribusi probabilitas Gaussian untuk suatu nilai x.

- Pembuatan fungsi untuk perhitungan probabilitas kelas

```
# 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
```

Fungsi calculate_class_probabilities menghitung probabilitas prediksi untuk setiap kelas berdasarkan distribusi Gaussian.

- Pembuatan fungsi untuk melakukan prediksi

```
# 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
```

Fungsi predict bertujuan untuk melakukan prediksi kelas untuk sebuah baris data tertentu menggunakan model Naive Bayes yang telah dilatih.

- Pembuatna fungsi untuk menerapkan algoritma Naive Bayes

```
# 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)
```

Fungsi naive_bayes mengimplementasikan algoritma Naive Bayes dengan menggunakan fungsi-fungsi sebelumnya.

- Memuat kumpulan data iris & transformasi data

```
# 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)
```

Membaca dataset Iris dari file 'iris.csv' dan mengonversi kolom-kolom pada dataset yang telah dibaca yang berbentuk numerik menjadi float dan kolom kelas menjadi integer.

- Perlatihan model

```
# fit model
model = summarize_by_class(dataset)
```

Melatih model Naive Bayes dengan menggunakan fungsi summarize_by_class.

- Penggunaan fungsi predict untuk melakukan prediksi pada data baru

```
# 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))
```

Membuat data baru dan dari data baru tersebut dilakukan prediksi (contoh baris [5.7, 2.9, 4.2, 1.3]) menggunakan model yang telah dilatih menggunakan fungsi summarize_by_class sebelumnya.

Output

Output dari kode di atas adalah prediksi kelas untuk data baru yang diberikan dalam variabel row. Hasil prediksi tersebut akan dicetak pada layar. Output ini memberikan informasi tentang kelas yang diprediksi oleh model Naive Bayes untuk data input tertentu.

Jika, menggunakan kasus pada kode di atas, outputnya seperti berikut:

```
Data=[5.7, 2.9, 4.2, 1.3], Predicted: 1
```

Yang dimana, hasilnya menyatakan bahwa data baru yang telah dibuat diprediksi merupakan bagian dari kelas 1 (integer dari hasil konversi menggunakan fungsi str_column_to_int() yang pasangannya adalah Iris-versicolor)