Langkah-langkah menerapkan algoritma Naive Bayes Classifier menggunakan Python

A. Tujuan

Menerapkan algoritma Naive Bayes untuk megklasifikasikan dataset bunga Iris (Iris Flower Species Dataset).

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 Naive Bayes Classifier menggunakan Python

Langkah-langkah berikut ini adalah langkah dasar yang diperlukan untuk mengimplementasikan Naive Bayes dari awal yang akan diterapkan pada masalah pemodelan prediktif kelas bunga Iris.

Langkah 1: Pemisahan Data per Kelas.

Langkah 2: Perhitungan statistik Dataset.

Langkah 3: Perhitungan statistik Data per Kelas.

Langkah 4: Peritungan Gaussian Probability Density Function.

Langkah 5: Perhitungan Probabilitas Kelas.

Sebelum mengaplikasikan yang dibuat ke Iris Dataset, kode diujikan dulu kedalam dataset kecil.

Catatan: Koding menggunakan Python 3. Jika menggunakan Python 2.7 **Note**: penggunaan fungsi *items()* pada dictionary objects diubah menjadi *iteritems()*.

Langkah 1: Pemisahan Data per Kelas.

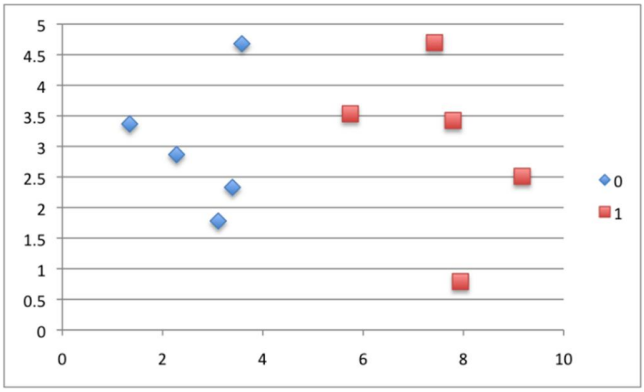
Diawali dengan menghitung probabilitas data berdasarkan kelasnya, disebut dengan *base rate*. Pisahkan data pelatihan berdasarkan kelas. Buat obyek *dictionary* dengan nilai kelas sebagai *key*, kemudian menambahkan sebuah *list* sebagai *value* dalam *dictionary*. Berikut ini adalah fungsi bernama ***separate_by_class()*** yang mengimplementasikan pendekatan ini. Diasumsikan bahwa kolom terakhir di setiap baris data adalah nilai kelas.

```
# 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
```

Uji output fungsi ***separate_by_class()*** menggunakan dataset ujicoba berukran kecil berikut:

X1	X2	Y
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

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



Selanjutnya, pisahkan data kecil tersebut berdasarkan kelasnya menggunakan fungsi ***separate_by_class()***

```
# Example of separating data by class value
# 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)
```

Setelah kode diatas dijalankan, dihasilkan kumpulan data yang terkelompok berdasarkan kelasnya, lalu cetak label kelas ke layar diikuti dengan semua data yang teridentifikasi sebagai anggota kelas tersebut.

```
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]
```

Selanjutnya kita buat fungsi lain yang diperlukan untuk melakukan perhitungan statistik semua data dalam dataset seperti pada langkah 2.

Langkah 2: Perhitungan Statistik.Dataset.

Dalam rangka menghitung probabilitas, dibutuhkan dua macam perhitungan statistik terhadap semua data yang ada di dataset masukan, yaitu menghitung *mean* dan *standard deviation* data per atribut/kolom untuk data dalam semua kelas. Rumus perhitungan *mean* adalah sebagai berikut:

$$mean = \mu = \frac{sum(x)}{n} * count(x)$$

Dengan x adalah sekumpulan nilai atau sebuah kolom yang akan kita cari.

Berikut ini adalah fungsi untuk menghitung *mean* dari sekumpulan nilai data

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

Perhitungan *standard deviation* adalah sebagai berikut:

$$standard\ deviation = \sigma = sqrt\frac{\sum_i^N (x_{i-}\bar{x})^2}{N-1}$$

Berikut ini adalah fungsi untuk menghitung *standard deviation* dari sekumpulan nilai data:

```
from math import sqrt

# 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)
```

Diperlukan perhitungan *mean* dan *standard deviation* untuk tiap kolom/atribut dalam dataset. Hasil perhitungan *mean* dan *standard deviation* pada semua kolom disimpan dalam sebuah list of tuples. Berikut ini adalah fungsi ***summarize_dataset()*** untuk menghitung perhitungan statistic yang meliputi *mean* dan *standard deviation* dari tiap kolom/atribut dalam dataset.

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

Uji terlebih dahulu semua fungsi yang telah dibuat pada data kecil, sebagai berikut:

```
# Example of summarizing a dataset
# 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)
```

Berikut ini adalah hasil perhitungan statistik dari dari dua buah atribut/kolom yang dimiliki oleh dataset setelah menjalankan kode program diatas:

```
[(5.178333386499999, 2.7665845055177263, 10),
(2.9984683241, 1.218556343617447, 10)]
```

Dapat dilihat bahwa nilai rata-rata (*mean*) kolom/atribut X1 adalah 5.178333386499999 dan standar deviasi (*standard deviation*) X1 adalah 2.7665845055177263.

Langkah 3: Perhitungan Statistik Data per Kelas.

Langkah berikutnya adalah menghitung statistik dari dataset per kelas. Pada tahap sebelumnya kita telah memiliki fungsi ***separate_by_class()*** untuk memisahkan dataset berdasarkan kelasnya. Dan juga telah membuat fungsi ***summarize_dataset*()** untuk menghitung statistik (*mean* dan *standard deviation*) untuk tiap kolom untuk keseluruhan data dalam dataset (untuk semua kelas). Selanjutnya, gunakan dua buah fungsi berikut ini untuk menghitung statistik tiap kelas/atribut yang dimiliki tiap kelas. Berikut ini adalah fungsi ***summarize_by_class*()** untuk meringkas hasil perhitungan *mean* dan *standard deviation* dari semua kolom/atribut dalam 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
```

Tes fungsi ***summarize_by_class*()** pada data kecil.

```
# Example of summarizing data by class value
from math import sqrt

# 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)
```

Hasil menjalankan fungsi ini adalah tampilan nilai kelas beserta nilai *mean* dan *standard deviation* masing-masing kelas.

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

Fungsi-fungsi yang telah dibuat diatas merupakan fungsi-fungsi yang dibutuhkan dalam menghitung probabilitas.

Langkah 4: Perhitungan Gaussian Probability Density Function

Menghitung probabilitas atau likelihood dari sekumpulan data bertipe data real seperti nilai-nilai ada atribut X1 data kecil diatas adalah sulit. Cara yang bisa dipakai adalah mengasumsikan data berdistribusi Gaussian.

Distribusi Gaussian dapat dihitung menggunakan dua variavel: yaitu *mean* dan *standard deviation*. Berikut ini adalah formula dalam menghitung Gaussian Probability Distribution Function:

$$f(x|\mu,\sigma^2 = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Dengan σ adalah *standard deviation* untuk x, μ (*mean*) adalah nilai rata-rata untuk x dan π adalah nilai konstanta pi.

Berikut ini fungsi ***calculate_probability()*** untuk menghitung Gaussian Probability Density:

```
# 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
```

Ujicoba fungsi ***calculate_probability()***

```
# 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))
```

Hasil running kode diatas adalah sebagai berikut:

```
0.3989422804014327
0.24197072451914337
0.24197072451914337
```

Langkah 5: Menghitung Probabilitas Kelas

Sekarang kita akan menggunakan perhitungan statistik dari data training untuk menghitung probabilitas dari data baru. Perhitungan probabilitas akan dilakukan untuk tiap kelas. Ini artinya bahwa pertama kali kita harus menghitung probabilitas bahwa sebuah data milik kelas pertama. Selanjutnya menghitung probabilitas bahwa sebuah data milik kelas kedua, dan seterusnya sampai semua kelas.

Probabilitas sebuah data baru adalah milik sebuah kelas diformulasikan sebagai berikut:

$$P(class|data) = P(X|class) * P(class)$$

Formula tersebut agak berbeda dengan rumus dasar Bayes Theorem. Konstanta pembagi, yaitu $P(data)$ dihilangkan. Hal ini berarti bahwa hasil kalkulasi tidak lagi merupakan probabilitas data yang dimiliki suatu kelas. Nilai tersebut masih dimaksimalkan, artinya penghitungan kelas yang menghasilkan nilai terbesar diambil sebagai prediksi. Ini adalah penyederhanaan implementasi fornulasi yang umum dilakukan karena kita sering kali lebih tertarik pada prediksi kelas daripada probabilitas.

Perhitungan probabilitas untuk tiap variabel dilakukan secara terpisah (“naïve"). Pada contoh di atas dimana dataset hanya memiliki 2 variabel input, perhitungan probabilitas bahwa suatu data baru termasuk dalam kelas pertama yaitu kelas 0 dapat dihitung sebagai berikut:

$$P(class = 0|X1, X2) = P(X1|class = 0) * P(X2|class = 0) * P(class = 0)$$

Inilah alasan mengapa kita perlu memisahkan data berdasarkan nilai per kelas. Fungsi ***calculate_probability()*** untuk menghitung Gaussian Probability Density yang telah kita buat

pada langkah sebelumnya digunakan untuk menghitung probabilitas dari atribut bertipe data real seperti X1 dan untuk perhitungan statistik yang dibutuhkan.

Di bawah ini adalah fungsi bernama ***calculate_class_probabilities()*** yang menggabungkan semua kebutuhan perhitungan tersebut diatas dalam sebuah fungsi. Argumen yang dibutuhkan fungsi ini adalah sekumpulan hasil perhitungan statistik (*summaries*) dan sebuah data baru (*row*).

Pertama, jumlah total data training dihitung berdasarkan ukuran data yang tersimpan dalam perhitungan statistik (*summaries*). Nilai ini digunakan dalam perhitungan probabilitas kelas tertentu ($P(class)$) dalam bentuk rasio antara jumlah data kelas tertentu terhadap total jumlah data dalam data pelatihan.

Selanjutnya, probabilitas dihitung untuk setiap data input menggunakan fungsi Gaussian probability density function dan fungsi perhitungan statistik untuk kolom/atribut milik kelas tersebut. Selanjutnya, semua nilai probabilitas dikalikan. Proses ini diulang sebanyak jumlah kelas yang dimiliki oleh dataset. Pada tahap akhir, sekumpulan nilai probabilitas yang tersimpan dalam *dictionary* sejumlah banyaknya kelas (satu nilai probabilitas per kelas) akan menjadi nilai balik dari fungsi ini.

```
# 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(tot
al_rows)
    for i in range(len(class_summaries)):
      mean, stdev, count = class_summaries[i]
      probabilities[class_value] *= calculate_probability(row[i], mean,
 stdev)
  return probabilities
```

Contoh kode di bawah ini diawali dengan menghitung statistik data per kelas untuk set data training, kemudian menggunakan nilai statistik tersebut untuk menghitung probabilitas data terhadap setiap kelas. Berikut ini kode untuk menjalankan data kecil pada fungsi-fungsi yang telah dibuat dan menampilkan nilai probabilitas tiap kelas.

```
# Test calculating class 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)
```

Hasilnya dapat kita lihat bahwa probabilitas data baru termasuk kelas 0 (0,0503) lebih tinggi daripada probabilitas data baru termasuk kelas 1 (0,0001). Oleh karena itu, dapat disimpulkan bahwa data baru ini adalah milik kelas 0.

Pada tahap ini, tahap demi tahap fungsi-fungsi yang dibutuhkan untuk menerapkan algoritma Naive Bayes telah diterapkan dalam kode program. Selanjutnya kode ini akan kita terapkan ke Iris Flower Species Dataset.

D. Iris Flower Species Case Study

Bagian ini menerapkan algoritma Naive Bayes ke Iris Flower Species Dataset.

Langkah pertama adalah load dataset dan mengubah data menjadi angka yang dapat diterima oleh fungsi perhitungan *mean* dan *standard deviation*. 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 bernama ***predict()*** akan dibuat untuk menangani kalkulasi probabilitas data baru terhadap setiap kelas yang ada dan memilih kelas dengan nilai probabilitas terbesar sebagai prediksi pemilik kelas dari data baru tersebut.

Fungsi baru lainnya bernama ***naive_bayes()*** akan dibuat untuk menangani penerapan algoritma Naive Bayes, pertama-tama menghitung statistik dari set data pelatihan dan menggunakannya untuk membuat prediksi untuk set data pengujian.

Berikut ini adalah kode programnya:

```
# 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 zi
p(*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, st
dev)
  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)
```

```
# Test Naive Bayes on Iris 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
scores = evaluate_algorithm(dataset, naive_bayes, n_folds)
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 95%, skor ini jauh lebih baik daripada skor akurasi baseline sebesar 33%. Perhatikan bahwa akurasi 95% ini adalah akurasi tahap pelatihan, yaitu akurasi saat model erlatih mengenali pola data dari keseluruhan dataset training. Sampai pada tahap ini belum dilakukan data testing.

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

Kita dapat melakukan training untuk membangun model pada seluruh dataset dan kemudian menggunakan model tersebut untuk membuat prediksi pada data pengamatan baru (data testing). Misalnya, pertama-tama *model* dibentuk dari sekumpulan probabilitas yang dihitung mengguakan fungsi ***summarize_by_class ()***.

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

Setelah dihitung, *model* digunakan sebagai argument fungsi ***predict()*** dengan argument *row* adalah data baru yang akan diredikasi label kelas nya.

```
...
# predict the label
label = predict(model, row)
```

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 dari proses pelatihan *(fit)* model Naive Bayes pada seluruh dataset dan membuat prediksi terhadap satu buah data baru:

```
# Make Predictions with Naive Bayes On The Iris Dataset
from csv import reader
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
    print('[%s] => %d' % (value, i))
  for row in dataset:
    row[column] = lookup[row[column]]
  return lookup

# 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(tot
al_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
```

```
# Make a prediction with Naive Bayes 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)
# 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))
```

Proses yang terjadi selama program dijalankan diawali dengan menghitung statistik data training dan melakukan training (dengan pemanggilan fungsi *fit()*) pada keseluruhan dataset. Selanjutnya, sebuah data baru didefinisikan dan label kelas prediksi dihitung. Hasil prediksi menyatakan bahwa data baru adalah milik kelas 1 yaitu “Iris-versicolor”. Berikut ini adalah output running program diatas:

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