1. 设 $f(x)$ 连续.

则当 $x=3$ 时,

$$kx = 2 - \frac{x}{2}$$

$$3k = 2 - \frac{3}{2} \quad (\frac{1}{2})$$

$$k = \frac{1}{6}$$

$$E(x) = \int_0^{\cancel{4}} x f(x)\,dx$$

$$= \int_0^3 \frac{1}{6} x^2 dx + \int_3^4 2x - \frac{x^2}{2}\,dx$$

$$= \frac{1}{18} x^3 \Big|_0^3 \quad x^2 \Big|_3^4 - \frac{x^3}{6} \Big|_3^4$$

$$= \frac{27}{18} + \frac{\cancel{7}\ 126}{18} - \frac{\cancel{37}}{\cancel{6}} \quad \frac{111}{18}$$

$$= \frac{\cancel{42}}{\cancel{18}} \frac{7}{3} = \frac{7}{3}$$

$$Var(x) = \int_0^4 x^2 f(x)\, dx$$

$$= \int_0^3 \frac{x^3}{6}\, dx + \int_3^4 2x^2 - \frac{x^3}{2}\, dx$$

$$= \frac{1}{24} x^4 \Big|_0^3 + \frac{2}{3} x^3 \Big|_3^4 - \frac{x^4}{8} \Big|_3^4$$

$$= \frac{81}{24} + \frac{2}{3} \cdot 37 - \frac{1}{8}\, 175$$

$$= \frac{81 + 74 \cdot 8 - 3 \cdot 175}{24} = \cancel{81} + 5\cancel{9}2 - 5\cancel{2}5 \quad 148 \quad \frac{37}{6}$$

$$\cancel{6}7$$

$$= \frac{37}{6}$$

2.

$Var(x) = 2.65$

$x \sim N(\mu, \sigma^2)$

置信区间: $(\bar{x} - c\frac{\sigma}{\sqrt{n}}, \bar{x} + c\frac{\sigma}{\sqrt{n}})$

$\bar{x} = 24.9$

$c = \begin{cases} 2.58 & 99\% \\ 1.96 & 95\% \end{cases}$

$\frac{\sigma}{\sqrt{n}} = \frac{\sqrt{2.65}}{\sqrt{18}} = 0.383$

99% : $(23.91, 25.89)$

95% : $(24.15, 25.65)$

3 总体方差未知. 按 t 估计

$$SE_{x} = \frac{\sigma}{\sqrt{n-1}} = \frac{10}{\sqrt{19}}$$

$$t_{0.05/2}(19) = 2.093$$

$$t_{0.01/2}(19) = 2.861$$

95% : (150.198, 159.801)

99% : (148.436, 161.563)

$$155 + 2.093 \times \frac{10}{\sqrt{19}} = 159.801...$$

$$155 - 2.093 \times \frac{10}{\sqrt{19}} = 150.198...$$

$$155 + 2.861 \times \frac{10}{\sqrt{19}} = 161.563...$$

$$155 - 2.861 \times \frac{10}{\sqrt{19}} = 148.436...$$

4. A BD BAD BCD BE BADE BCDE BF BADF BCDF

$\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}$ BG BADFG BADEG BCDFG BCDEG BH BADFH BADEH BCDEH BCDEH BI BADEGI BADFGI BCDEGI BCDFGI

= 3

B CA CBA CDA $\frac{1}{2}$

C BD . BE , BF . BH , BG . BI 同 A 3

D AE . AF . AH . AG . AI 5

B 5

C 5

15

E A|B|C|D - H|G|I $4 \times 3 \times \frac{1}{2} = 6$

E|F

F 同E . 6

H 0

G A|B|C|D|E|F|H - I 7

I 0

HW1-BAYES

SA22011090 余致远

朴素贝叶斯分类器（Naive Bayes Classifier）

数据集：Bayesian_Dataset_train.csv, Bayesian_Dataset_test.csv。

数据描述：列名分别为“年纪、工作性质、家庭收入、学位、工作类型、婚姻状况、族裔、性别、工作地点”，最后一列是标签，即收入是否大于 50k 每年。

任务描述：使用朴素贝叶斯（Naïve Bayesian）预测一个人的收入是否高于 50K 每年。

要求输出：

1）结果统计，例如 precision、recall、F1 score 等；

2）csv 文件，在 test 文件最后增加一列，填入模型预测的收入标签（<=50K 或>50K）

Optional：探索不同参数对结果的影响。

In [264…
```
import pandas as pd

train_file_path = 'datasets/Bayesian_Dataset_train.csv'
test_file_path  = 'datasets/Bayesian_Dataset_test.csv'
train_df = pd.read_csv(train_file_path, names=['age','job_nature','family_income','c
test_df = pd.read_csv(test_file_path, names=['age','job_nature','family_income','deg
# train_df

train_df
```

Out[264]:

	age	job_nature	family_income	degree	marriage	job_type	in_family	race	gend
0	39	State-gov	77516	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Ma
1	50	Self-emp-not-inc	83311	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Ma
2	38	Private	215646	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Ma
3	53	Private	234721	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Ma
4	28	Private	338409	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Fema
...	...	...	...	...	...	...	...	...	
27197	27	Private	257302	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	White	Fema
27198	40	Private	154374	HS-grad	Married-civ-spouse	Machine-op-inspct	Husband	White	Ma
27199	58	Private	151910	HS-grad	Widowed	Adm-clerical	Unmarried	White	Fema
27200	22	Private	201490	HS-grad	Never-married	Adm-clerical	Own-child	White	Ma
27201	52	Self-emp-inc	287927	HS-grad	Married-civ-spouse	Exec-managerial	Wife	White	Fema

27202 rows × 11 columns

In [265…

```
# jobNatureMap = {elem:index+1 for index, elem in enumerate(set(train_df['job_nature
# train_df['job_nature'] = train_df['job_nature'].map(jobNatureMap)

# pd.factorize() is a better way

map_dict = {}
for column in train_df.columns:
    if column == 'age' or column == 'family_income':
        continue
    factorized_column, map = train_df[column].factorize()
    train_df[column] = factorized_column
    map_dict[column] = {map[i]:i for i in range(len(map))}

# train_df
# factorized_column
map_dict['race']
```

Out[265]:

```
{'White': 0,
 'Black': 1,
 'Asian-Pac-Islander': 2,
 'Amer-Indian-Eskimo': 3,
 'Other': 4}
```

In [266…
```
# map_dict['race'][[1,2,1]]
```

In [267…
```
for column in test_df.columns:
    if column == 'age' or column == 'family_income':
        continue
    test_df[column] = test_df[column].replace(map_dict[column])

test_df
```

Out[267]:

	age	job_nature	family_income	degree	marriage	job_type	in_family	race	gender	wor
0	52	1	209642	1	1	1	1	0	0	
1	59	2	109015	1	2	10	4	0	1	
2	56	5	216851	0	1	10	1	0	0	
3	23	5	190709	6	0	11	0	0	0	
4	31	2	507875	4	1	8	1	0	0	
...	...	...	...	...	...	...	...	...	...	
2955	24	2	381895	2	2	8	4	0	1	
2956	18	2	436163	2	0	3	3	0	0	
2957	25	5	514716	0	0	0	3	1	1	
2958	46	2	42972	3	1	3	2	0	1	
2959	30	2	77266	1	2	6	0	0	0	

2960 rows × 11 columns

In [268…
```
train_features, train_labels = train_df.iloc[:,:-1], train_df.iloc[:,-1]
test_features, test_labels = test_df.iloc[:,:-1], test_df.iloc[:,-1]

# test_features
```

In [269…
```
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import precision_recall_fscore_support, accuracy_score

model = CategoricalNB()
model.fit(train_features, train_labels)

y_pred = model.predict(test_features)
y_pred
```

Out[269]:
```
array([1, 0, 1, ..., 0, 1, 0], dtype=int64)
```

In [270…
```
precision, recall, F1_score, _ = precision_recall_fscore_support(test_labels, y_pred
acc = accuracy_score(test_labels, y_pred)
print("precision: {} \nrecall: {} \nF1 score: {} \naccuracy: {}".format(precision, r
```

```
precision: [0.90317776 0.5845666 ]
recall: [0.82233273 0.73930481]
F1 score: [0.86086133 0.65289256]
accuracy: 0.8013513513513514
```

In [271…
```
# mostly borrowed from https://juejin.cn/post/6865193399784472583
import numpy as np
import math
```

```python
from scipy.stats import multivariate_normal

class NaiveBayes:
    def __init__(self):
        self.model = None

    # process X_train
    def summarize(self, X):
        # summaries = [(self.mean(i), self.stdev(i)) for i in zip(*X)]
        summaries = [(np.mean(i), np.var(i)) for i in zip(*X)]
        return summaries

    def fit(self, X, y):
        labels = list(set(y))
        data = {label: [] for label in labels}
        for feature, label in zip(X, y):
            data[label].append(feature)
        self.model = {label: self.summarize(feature) for label, feature in data.ite
        return self.model

    # 计算概率
    def calculate_probabilities(self, X):
        probabilities = {}
        for label, feature in self.model.items():
            probabilities[label] = 1
            for i in range(len(feature)):
                mean, stdev = feature[i]
                probabilities[label] *= multivariate_normal.pdf(X[i], mean, stdev)

        return probabilities

    # 类别
    def predict(self, x_test):
        # 将预测数据在所有类别中的概率进行排序，并取概率最高的类别
        label = sorted(self.calculate_probabilities(x_test).items(), key=lambda x:
        return label
```

In [272…

```python
model = NaiveBayes()
model.fit(train_features.to_numpy(),train_labels.to_numpy())

y_pred = [model.predict(i) for i in test_features.to_numpy()]
# y_pred
```

In [273…

```python
precision, recall, F1_score, _ = precision_recall_fscore_support(test_labels, y_pred
acc = accuracy_score(test_labels, y_pred)
print("precision: {} \nrecall: {} \nF1 score: {} \naccuracy: {}".format(precision, r
```

```
precision: [0.89432485 0.41065172]
recall: [0.61980108 0.78342246]
F1 score: [0.73217623 0.53885057]
accuracy: 0.6611486486486486
```

HW1-GMM

SA22011090 余致远

高斯混合模型与 EM 算法

数据集：Iris 数据集

数据描述：https://www.kaggle.com/datasets/uciml/iris，可通过 sklearn 直接导入数据集

任务描述：使用高斯混合模型与 EM 算法对数据进行分类计算，mixture components 设置为 3。

要求输出：不同高斯分布的 mean 和 variance，每个高斯分布对应的权重，plot 出分布的图。

EM 算法可以参考 https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html。

Optional：尝试不同的 covariance structures，包括 spherical、diagonal、tied 与 full。

In [2]:

```
from sklearn.mixture import GaussianMixture
from sklearn import datasets
import numpy as np

iris = datasets.load_iris()
features = iris.data
labels = iris.target

gmm = GaussianMixture(n_components=3, covariance_type='full')
# iris['feature_names']
iris.data.shape
```

Out[2]:

```
(150, 4)
```

$$\ell(\theta) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{2} \pi_k \underbrace{N(x_i; \mu_k, \sigma_k^2)}_{L[i,k]} \right)$$

In [52]:

```
from scipy.stats import multivariate_normal

class myGaussianMixture(object):
    # mostly borrowed from https://xavierbourretsicotte.github.io/gaussian_mixture.h

    def __init__(self, n_components = 3, max_iter = 100, tol = 0.001):

        # Parameters
        self.n_components = n_components
        self.max_iter = max_iter
        self.tol = tol

        # Attributes
        self.means_ = None
        self.covariances_ = None
```

```
        self.weights_ = None

        self.log_likelihoods = []

    def fit(self, X= None):

        n,d = X.shape           # [150, 4]
        k = self.n_components   # 3 types

        # initialize means
        mu = X[np.random.choice(n,k,replace = False)]   # [points=150, types=3]

        # initialize a covariance matrix for each gaussian, uppercase for mat
        Sigma = [np.eye(d)] * k                         # k=3 types * [eye(features=

        # initialize the probability for each gaussian pi
        pi = np.array([1 / k] * k)                      # when summing up, count \su

        # initialize responsibility matrix: n points for each gaussian
        W = np.zeros((n,k))                             # [points=150, types=3]

        # initialize list of log-likelihoods
        log_likelihoods = []

        iter = 0
        while iter < self.max_iter:

            # E-step

            # lambda function for gaussian pdf
            P = lambda m, s: multivariate_normal.pdf(X, mean = m, cov = s, allow_si

            # nominator of responsibilities: j is the j-th gaussian
            # for each node, count the expectation of belonging to type j
            for j in range(k):
                W[:, j] = pi[j] * P(mu[j], Sigma[j])

            # log likelihood computation (same as nominator of responsibilities)
            l = np.sum(np.log(np.sum(W, axis = 1)))

            # store log likelihood in list
            log_likelihoods.append(l)

            # compute W matrix by dividing by denominator (the sum along j)
            W = (W.T / W.sum(axis = 1)).T

            # sum of w^i entries along j (used for parameter updates)
            # these are the soft weighted number of datapoints belonging to each gau
            W_sum = np.sum(W, axis = 0)


            # M step

            for j in range(k):

                ## Update means
                mu[j] = (1. / W_sum[j]) * np.sum(W[:, j] * X.T, axis = 1).T

                ## Update covariances
                Sigma[j] = ((W[:,j] * ((X - mu[j]).T)) @ (X - mu[j])) / W_sum[j]

                ## Update probabilities of each gaussian
                pi[j] = W_sum[j] / n
```

```
            # check for convergence
            if len(log_likelihoods) < 2: continue
            if np.abs(l - log_likelihoods[-2]) < self.tol: break

        self.means_ = mu
        self.covariances_ = Sigma
        self.weights_ = pi
        self.log_likelihoods = log_likelihoods

    def predict(self, X):
        probs = np.array([ multivariate_normal.pdf(X, mean = self.means_[j], cov =
        return np.argmax(probs, axis = 0)
```

In [59]:
```
import matplotlib.pyplot as plt
# from matplotlib.colors import ListedColormap
# from matplotlib import patches

X = iris.data
gmm = myGaussianMixture()
gmm.fit(X)
y = gmm.predict(X)
print(y)

print(labels)
```

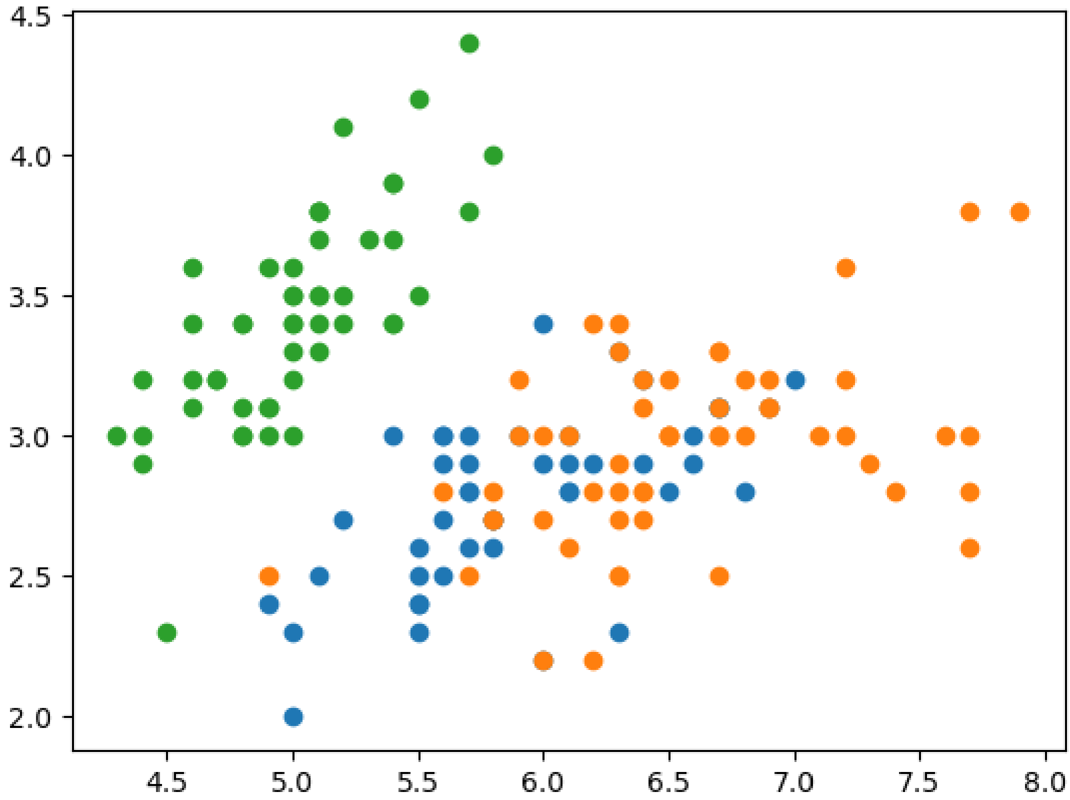
```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [61]:
```
type = {'0':[],'1':[],'2':[]}
for i in range(len(X)):
    # print(i, str(y[i]))
    type[str(y[i])].append(i)
# type

iris_result = [X[type['0']],X[type['1']],X[type['2']]]
# iris_result

# from sklearn.decomposition import PCA

# pca = PCA(n_components=2)
for i in range(len(iris_result)):
    # iris_result[i] = pca.fit_transform(iris_result[i])
    plt.scatter(iris_result[i][:,0],iris_result[i][:,1])
```



In [62]:
```
gmm.means_
```

Out[62]:
```
array([[5.91513787, 2.77785927, 4.20189782, 1.29710082],
       [6.54476071, 2.94874317, 5.47998145, 1.98487718],
       [5.006     , 3.428     , 1.462     , 0.246     ]])
```

In [63]:
```
gmm.covariances_
```

Out[63]:
```
[array([[0.2753255 , 0.09690444, 0.18471488, 0.05441802],
        [0.09690444, 0.09263655, 0.09112867, 0.04299585],
        [0.18471488, 0.09112867, 0.20076472, 0.06103621],
        [0.05441802, 0.04299585, 0.06103621, 0.03202467]]),
 array([[0.38705358, 0.09220718, 0.30274019, 0.0615557 ],
        [0.09220718, 0.11034341, 0.08423873, 0.05598218],
        [0.30274019, 0.08423873, 0.32757013, 0.07433559],
        [0.0615557 , 0.05598218, 0.07433559, 0.08569279]]),
 array([[0.121764, 0.097232, 0.016028, 0.010124],
        [0.097232, 0.140816, 0.011464, 0.009112],
        [0.016028, 0.011464, 0.029556, 0.005948],
        [0.010124, 0.009112, 0.005948, 0.010884]])]
```

In [64]:
```
gmm.weights_
```

Out[64]:
```
array([0.29939692, 0.36726975, 0.33333333])
```