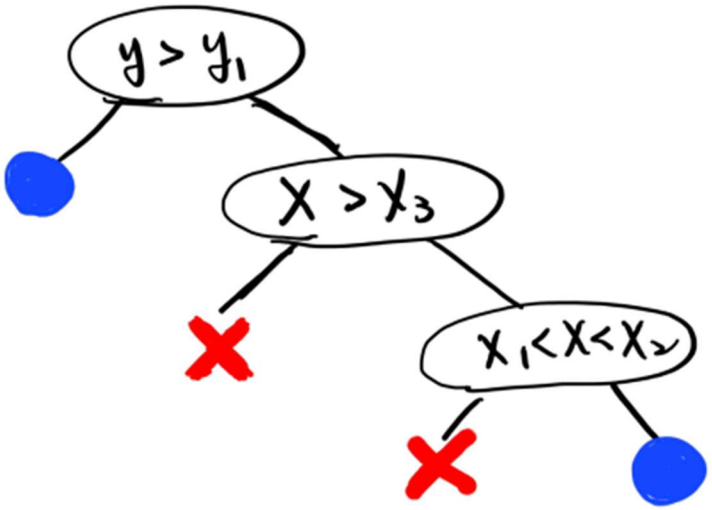
Machine learning HW03 111598067 王上澤

1. Mahalanobis' Method is better due to the formula :

$$(x-\mu)^T \Sigma^{-1} (x-\mu)$$

consider the all distributed points of the same group . so it find outliers more objectly .

2. Decision Tree of the graph.



3. one-dimensional data

$[0.9, 0.7, 1.2, 2.4, 1.8]$

Suppose

$\mu_1 = 1, \mu_2 = 2$

$\sigma_1^2 = \sigma_2^2 = 1$

$\alpha_1 = \alpha_2 = 0.5$

Start from $\beta_j(x) = p(j|x) = \frac{\alpha_j g_j(x)}{\sum_{k=1}^{2} \alpha_k g_k(x)}$

i.e. $g(x)$

(pdf of normal distribution) (univariate Gaussian)

$= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

3.

Put our assumption in $g_1(x)$, $g_2(x)$.

$$g_1(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-1}{1}\right)^2}$$

$$g_2(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-2}{1}\right)^2}$$

For $X = 0.9, 0.7, 1.2, 2.4, 1.8$.

we have

$g_1(X) = 0.3970 \quad 0.3814 \quad 0.3910 \quad 0.1497 \quad 0.3897$

$g_2(X) = 0.3479 \quad 0.1714 \quad 0.3897 \quad 0.3683 \quad 0.3910$

than is to compute $\beta_j(x) = P(j|x)$

$$= \frac{\alpha_j g_j(x)}{\sum_{k=1}^{2} \alpha_k g_k(x)}$$

and $\alpha_1 = \alpha_2 = 0.5$

therefore, $\beta_1(x) = \frac{g_1(x)}{g_1(x)+g_2(x)}$

$$\beta_2(x) = \frac{g_2(x)}{g_1(x)+g_2(x)}.$$

$\beta_1(x) =$ 0.6457 0.6900 0.5744 0.3891 0.4256

$\beta_2(x) =$ 0.3543 0.3100 0.4256 0.7109 0.5744

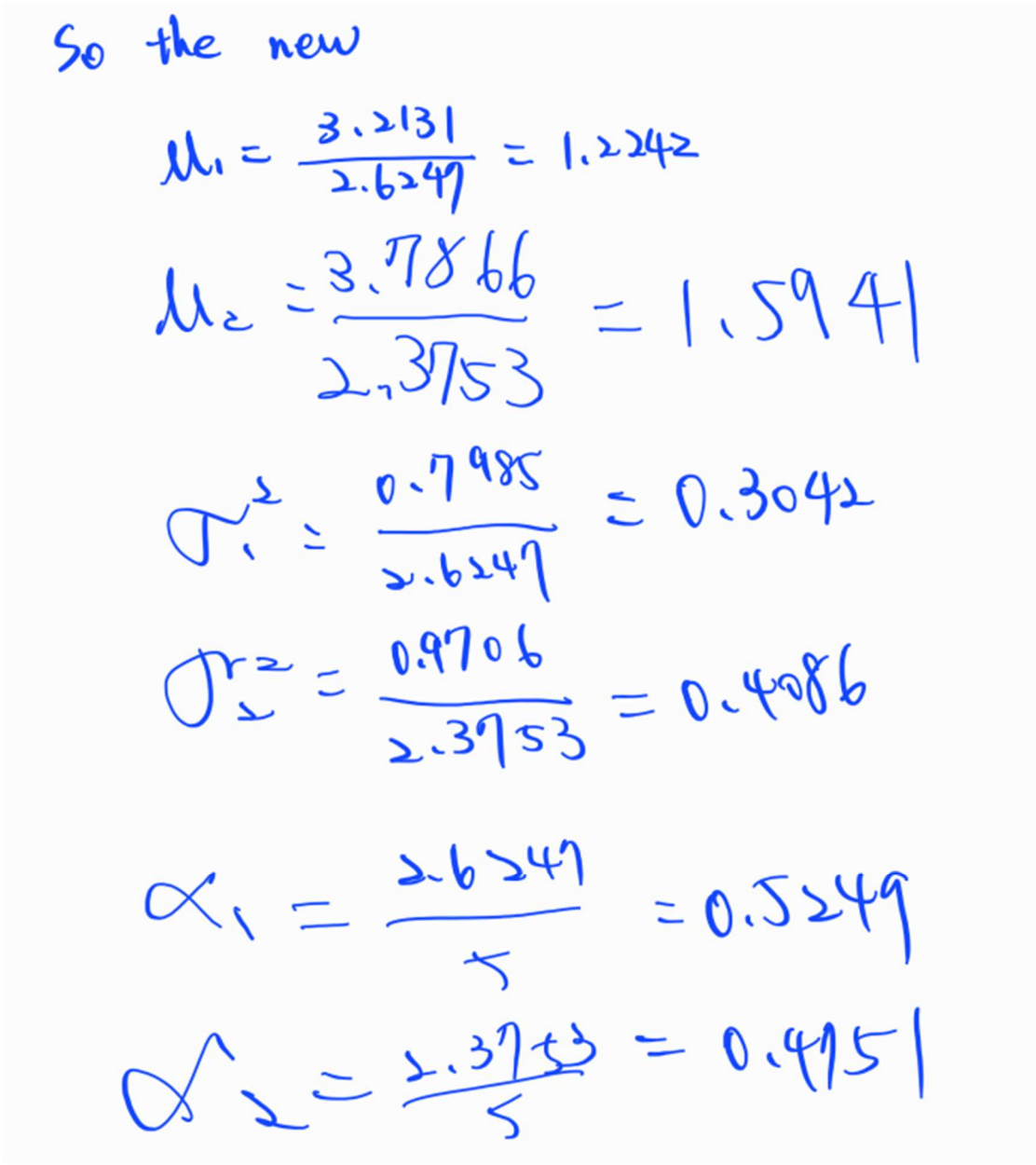
$x =$ 0.9 0.7 1.2 2.4 1.8

update method of

$$\mu_j = \frac{\sum_{i=0}^{n} \beta_j x_i}{\sum_{i=0}^{n} \beta_j}$$

$$\sigma_j = \frac{\sum_{i=0}^{n} \beta_j (x_i - \mu_j)^2}{\sum_{i=0}^{n} \beta_j}$$

$$\alpha_j = \frac{1}{n} \sum_{i=0}^{n} \beta_j$$



4. Code:

```
from sklearn import datasets, model_selection, metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.mixture import GaussianMixture
from numpy import average
import pandas as pd
import numpy as np

iris = datasets.load_iris()
acc_arr = []
iter_times = 10
```

```
for i in range(iter_times):
    # split the datasets
    data_train, data_test, target_train, target_test =
model_selection.train_test_split(
        iris.data, iris.target, test_size=0.3, random_state=i+22333333)
    # three classifier for each class
    Setosa_classifier = GaussianMixture(n_components=2,
random_state=i+22333333, max_iter=100)
    Versicolour_classifier = GaussianMixture(n_components=2,
random_state=i+22333333, max_iter=100)
    Virginica_classifier = GaussianMixture(n_components=2,
random_state=i+22333333, max_iter=100)
    # fit the trainning data
    Setosa_classifier.fit(data_train[target_train==0])
    Versicolour_classifier.fit(data_train[target_train==1])
    Virginica_classifier.fit(data_train[target_train==2])
    # get each class' score
    Setosa_score = Setosa_classifier.score_samples(X=data_test)
    Versicolour_score =
Versicolour_classifier.score_samples(X=data_test)
    Virginica_score = Virginica_classifier.score_samples(X=data_test)
    # combine three classes' score
    combine_pred = np.column_stack((Setosa_score, Versicolour_score,
Virginica_score))
    # get total accuracy list by picking out the maxium score's from
where they came from
    prediction = np.argmax(combine_pred, axis=1)

    accur = metrics.accuracy_score(target_test, prediction)
    print("{iter} 's train accuracy : {acc}".format(iter=i, acc=accur))
    acc_arr.append(accur)

print("average accuracy of {iter} times' GMM classification :
{aver_acc}".format(iter=iter_times,aver_acc=average(acc_arr)))
result :
0 's train accuracy : 0.9777777777777777
1 's train accuracy : 1.0
2 's train accuracy : 0.9777777777777777
```

```
3 's train accuracy : 1.0
4 's train accuracy : 0.9555555555555556
5 's train accuracy : 1.0
6 's train accuracy : 0.9777777777777777
7 's train accuracy : 0.9555555555555556
8 's train accuracy : 0.9111111111111111
9 's train accuracy : 0.9777777777777777
average accuracy of 10 times' GMM classification : 0.9733333333333334
```

5. Code:

```
from sklearn import datasets, model_selection, metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from numpy import average
import pandas as pd
import numpy as np
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.feature_selection import ColumnSelector as CSR

cancer_data = pd.read_csv("./data/breast-cancer-wisconsin.data",
names=[i for i in range(11)])
# print(len(cancer_data))
cancer_data = cancer_data.drop(cancer_data.columns[0], axis=1)
cancer_data = cancer_data.replace(to_replace='?', value=pd.NA)
cancer_data.dropna(inplace = True) ## remove the row that miss the
attribute
cancer_data = cancer_data.reset_index(drop=True)

# print(cancer_data[[1,2,3,4,5,6,7,8,9]])
acc_arr = []
iter_time = 10
for i in range(iter_time):
    # split the datasets
    data_train, data_test, target_train, target_test =
model_selection.train_test_split(
        cancer_data[[i for i in range(1, 10)]], cancer_data[10],
test_size=0.2
        )
```

```
    data_train, data_valid, target_train, target_valid =
model_selection.train_test_split(
        data_train, target_train, test_size=0.25
        )
    model = GaussianNB()
    sfs = SFS(model, forward=False, cv=0, k_features=3,
scoring='accuracy', verbose=False, n_jobs=-1)
    sfs.fit(data_valid, target_valid) # 20%
    print(f"Best score achieved: {sfs.k_score_}, \nFeature's names:
{sfs.k_feature_names_}")
    model.fit(CSR(cols=sfs.k_feature_names_).transform(data_train),targe
t_train) # 60%

    prediction =
model.predict(CSR(cols=sfs.k_feature_names_).transform(data_test)) #
20%
    accur = metrics.accuracy_score(target_test, prediction)
    acc_arr.append(accur)

print(f"\nthe average accuracy of 10 times trial =",average(acc_arr))
```

result:

```
Best score achieved: 0.9708029197080292,
Feature's names: (1, 3, 6)
Best score achieved: 0.9854014598540146,
Feature's names: (1, 2, 6)
Best score achieved: 0.9708029197080292,
Feature's names: (1, 2, 7)
Best score achieved: 0.9562043795620438,
Feature's names: (1, 2, 6)
Best score achieved: 0.948905109489051,
Feature's names: (1, 3, 8)
Best score achieved: 0.9781021897810219,
Feature's names: (2, 3, 4)
Best score achieved: 0.9708029197080292,
Feature's names: (1, 3, 6)
Best score achieved: 0.9635036496350365,
Feature's names: (1, 4, 6)
```

```
 Best score achieved: 0.9708029197080292,
 Feature's names: (1, 5, 6)
 Best score achieved: 0.9781021897810219,
 Feature's names: (2, 3, 6)

the average accuracy of 10 times trial = 0.9547445255474452
```