gaussian-classifiers-and-svm-using-numpy

June 10, 2020

1 Olimov Bekhzod ()

1.1 Kyungpook National University

2 1 Cross Validation Data Design

2.0.1 1,2,3,4. Splitting the dataset and creating folds

[1]:
```python
import numpy as np
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target
# Extracting all samples of the first class
X_class_1 = X[:50]
# Extracting all output values of the first class
y_class_1 = y[:50]
# Extracting all samples of the second class
X_class_2 = X[50:100]
# Extracting all output values of the second class
y_class_2 = y[50:100]
# Extracting all samples of the third class
X_class_3 = X[100:150]
# Extracting all output values of the second class
y_class_3 = y[100:150]

# Splitting the samples of class 1 into 5 different folds
f11 = X_class_1[:10]
f12 = X_class_1[10:20]
f13 = X_class_1[20:30]
f14 = X_class_1[30:40]
f15 = X_class_1[40:50]
# Splitting the output values of class 1 into 5 different folds
y_f11 = y_class_1[:10]
y_f12 = y_class_1[10:20]
y_f13 = y_class_1[20:30]
y_f14 = y_class_1[30:40]
```

```
y_f15 = y_class_1[40:50]

# Splitting the samples of class 2 into 5 different folds
f21 = X_class_2[:10]
f22 = X_class_2[10:20]
f23 = X_class_2[20:30]
f24 = X_class_2[30:40]
f25 = X_class_2[40:50]
# Splitting the output values of class 2 into 5 different folds
y_f21 = y_class_2[:10]
y_f22 = y_class_2[10:20]
y_f23 = y_class_2[20:30]
y_f24 = y_class_2[30:40]
y_f25 = y_class_2[40:50]

# Splitting the samples of class 3 into 5 different folds
f31 = X_class_3[:10]
f32 = X_class_3[10:20]
f33 = X_class_3[20:30]
f34 = X_class_3[30:40]
f35 = X_class_3[40:50]
# Splitting the output values of class 3 into 5 different folds
y_f31 = y_class_3[:10]
y_f32 = y_class_3[10:20]
y_f33 = y_class_3[20:30]
y_f34 = y_class_3[30:40]
y_f35 = y_class_3[40:50]

# Creating train and test sets for the first experiment
R1 = np.concatenate((f11, f12, f13, f14, f21, f22, f23, f24, f31, f32, f33,
 ↪f34))
T1 = np.concatenate((f15, f25, f35))
y_R1 = np.concatenate((y_f11, y_f12, y_f13, y_f14, y_f21, y_f22, y_f23, y_f24,
 ↪y_f31, y_f32, y_f33, y_f34))
y_T1 = np.concatenate((y_f15, y_f25, y_f35))

# Creating train and test sets for the second experiment
R2 = np.concatenate((f11, f12, f13, f15, f21, f22, f23, f25, f31, f32, f33,
 ↪f35))
T2 = np.concatenate((f14, f24, f34))
y_R2 = np.concatenate((y_f11, y_f12, y_f13, y_f15, y_f21, y_f22, y_f23, y_f25,
 ↪y_f31, y_f32, y_f33, y_f35))
y_T2 = np.concatenate((y_f14, y_f24, y_f34))

# Creating train and test sets for the third experiment
R3 = np.concatenate((f11, f12, f14, f15, f21, f22, f24, f25, f31, f32, f34,
 ↪f35))
```

```
T3 = np.concatenate((f13, f23, f33))
y_R3 = np.concatenate((y_f11, y_f12, y_f14, y_f15, y_f21, y_f22, y_f24, y_f25,␣
 ↪y_f31, y_f32, y_f34, y_f35))
y_T3 = np.concatenate((y_f13, y_f23, y_f33))

# Creating train and test sets for the fourth experiment
R4 = np.concatenate((f11, f13, f14, f15, f21, f23, f24, f25, f31, f33, f34,␣
 ↪f35))
T4 = np.concatenate((f12, f22, f32))
y_R4 = np.concatenate((y_f11, y_f13, y_f14, y_f15, y_f21, y_f23, y_f24, y_f25,␣
 ↪y_f31, y_f33, y_f34, y_f35))
y_T4 = np.concatenate((y_f12, y_f22, y_f32))

# Creating train and test sets for the fifth experiment
R5 = np.concatenate((f12, f13, f14, f15, f22, f23, f24, f25, f32, f33, f34,␣
 ↪f35))
T5 = np.concatenate((f11, f21, f31))
y_R5 = np.concatenate((y_f12, y_f13, y_f14, y_f15, y_f22, y_f23, y_f24, y_f25,␣
 ↪y_f32, y_f33, y_f34, y_f35))
y_T5 = np.concatenate((y_f11, y_f21, y_f31))
```

2.0.2 Calculating means

[2]:
```
# Length of test data
N = len(T1)

# Calculating means of each class of R1
class1_mean_R1 = np.mean(R1[:40],axis=0)
class2_mean_R1 = np.mean(R1[40:80],axis=0)
class3_mean_R1 = np.mean(R1[80:120],axis=0)
# Calculating means of each class of R2
class1_mean_R2 = np.mean(R2[:40],axis=0)
class2_mean_R2 = np.mean(R2[40:80],axis=0)
class3_mean_R2 = np.mean(R2[80:120],axis=0)
# Calculating means of each class of R3
class1_mean_R3 = np.mean(R3[:40],axis=0)
class2_mean_R3 = np.mean(R3[40:80],axis=0)
class3_mean_R3 = np.mean(R3[80:120],axis=0)
# Calculating means of each class of R4
class1_mean_R4 = np.mean(R4[:40],axis=0)
class2_mean_R4 = np.mean(R4[40:80],axis=0)
class3_mean_R4 = np.mean(R4[80:120],axis=0)
# Calculating means of each class of R5
class1_mean_R5 = np.mean(R5[:40],axis=0)
class2_mean_R5 = np.mean(R5[40:80],axis=0)
class3_mean_R5 = np.mean(R5[80:120],axis=0)
```

[3]:
```python
# Fuction for accuracy calculation of the classifiers
def accuracy(t_x, t_y, c1_mean, c2_mean, c3_mean, c1_cov, c2_cov, c3_cov):
    data = t_x
    target = t_y
    WT = 0
    for n in range(N):
        c1_d = np.matmul((data[n] - c1_mean), np.linalg.inv(c1_cov))
        c1_d = np.matmul(c1_d, (data[n] - c1_mean))
        c2_d = np.matmul((data[n] - c2_mean), np.linalg.inv(c2_cov))
        c2_d = np.matmul(c2_d, (data[n] - c2_mean))
        c3_d = np.matmul((data[n] - c3_mean), np.linalg.inv(c3_cov))
        c3_d = np.matmul(c3_d, (data[n] - c3_mean))
        pred_class = np.argmin([c1_d, c2_d, c3_d])
        if pred_class == target[n]:
            WT = WT + 1
    acc = (WT/N)*100
    return acc

identity_matrix = np.array([[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]])
```

3 2. Experiments with 6 different types of uni-modal Gaussian classifiers

3.0.1 1,2,3. Perform 5-fold cross-validation experiments for all 6 methods, evaluate the average performance, determine which is the best out of 6 methods

[4]:
```python
# The first type of uni-modal Gaussian classifier
# Calculating variance
R1_var = np.var(R1)
R2_var = np.var(R2)
R3_var = np.var(R3)
R4_var = np.var(R4)
R5_var = np.var(R5)
# Calculating covariance matrices
R1_cov_first = R1_var*identity_matrix
R2_cov_first = R2_var*identity_matrix
R3_cov_first = R3_var*identity_matrix
R4_cov_first = R4_var*identity_matrix
R5_cov_first = R5_var*identity_matrix

accuracy_1 = accuracy(T1, y_T1, class1_mean_R1, class2_mean_R1, class3_mean_R1,␣
 ↪R1_cov_first, R1_cov_first, R1_cov_first)
accuracy_2 = accuracy(T2, y_T2, class1_mean_R2, class2_mean_R2, class3_mean_R2,␣
 ↪R2_cov_first, R2_cov_first, R2_cov_first)
accuracy_3 = accuracy(T3, y_T3, class1_mean_R3, class2_mean_R3, class3_mean_R3,␣
 ↪R3_cov_first, R3_cov_first, R3_cov_first)
```

```
accuracy_4 = accuracy(T4, y_T4, class1_mean_R4, class2_mean_R4, class3_mean_R4, R4_cov_first, R4_cov_first, R4_cov_first)
accuracy_5 = accuracy(T5, y_T5, class1_mean_R5, class2_mean_R5, class3_mean_R5, R5_cov_first, R5_cov_first, R5_cov_first)
avg_accuracy = (accuracy_1 + accuracy_2 + accuracy_3 + accuracy_4 + accuracy_5)/5
print("The first type of uni-modal Gaussian classifier\n")
print("Accuracy of the classifier on the first experiment: {:.2f}%".format(accuracy_1))
print("Accuracy of the classifier on the second experiment: {:.2f}%".format(accuracy_2))
print("Accuracy of the classifier on the third experiment: {:.2f}%".format(accuracy_3))
print("Accuracy of the classifier on the fourth experiment: {:.2f}%".format(accuracy_4))
print("Accuracy of the classifier on the fifth experiment: {:.2f}%".format(accuracy_5))
print("Average accuracy of the first type uni-modal Gaussian classifier: {:.2f}%".format(avg_accuracy))
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


# The second type of uni-modal Gaussian classifier
# Calculating variance
R1_var_axis0 = np.var(R1, axis=0)
R2_var_axis0 = np.var(R2, axis=0)
R3_var_axis0 = np.var(R3, axis=0)
R4_var_axis0 = np.var(R4, axis=0)
R5_var_axis0 = np.var(R5, axis=0)
# Calculating covariance matrices
R1_cov_second = R1_var_axis0*identity_matrix
R2_cov_second = R2_var_axis0*identity_matrix
R3_cov_second = R3_var_axis0*identity_matrix
R4_cov_second = R4_var_axis0*identity_matrix
R5_cov_second = R5_var_axis0*identity_matrix

accuracy_1 = accuracy(T1, y_T1, class1_mean_R1, class2_mean_R1, class3_mean_R1, R1_cov_second, R1_cov_second, R1_cov_second)
accuracy_2 = accuracy(T2, y_T2, class1_mean_R2, class2_mean_R2, class3_mean_R2, R2_cov_second, R2_cov_second, R2_cov_second)
accuracy_3 = accuracy(T3, y_T3, class1_mean_R3, class2_mean_R3, class3_mean_R3, R3_cov_second, R3_cov_second, R3_cov_second)
accuracy_4 = accuracy(T4, y_T4, class1_mean_R4, class2_mean_R4, class3_mean_R4, R4_cov_second, R4_cov_second, R4_cov_second)
accuracy_5 = accuracy(T5, y_T5, class1_mean_R5, class2_mean_R5, class3_mean_R5, R5_cov_second, R5_cov_second, R5_cov_second)
```

```python
avg_accuracy = (accuracy_1 + accuracy_2 + accuracy_3 + accuracy_4 + accuracy_5)/5
print("The second type of uni-modal Gaussian classifier\n")
print("Accuracy of the classifier on the first experiment: {:.2f}%".format(accuracy_1))
print("Accuracy of the classifier on the second experiment: {:.2f}%".format(accuracy_2))
print("Accuracy of the classifier on the third experiment: {:.2f}%".format(accuracy_3))
print("Accuracy of the classifier on the fourth experiment: {:.2f}%".format(accuracy_4))
print("Accuracy of the classifier on the fifth experiment: {:.2f}%".format(accuracy_5))
print("Average accuracy of the second type uni-modal Gaussian classifier: {:.2f}%".format(avg_accuracy))
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# The third type of uni-modal Gaussian classifier
# Calculating covariance matrices
R1_cov_third = np.cov(R1, rowvar=False)
R2_cov_third = np.cov(R2, rowvar=False)
R3_cov_third = np.cov(R3, rowvar=False)
R4_cov_third = np.cov(R4, rowvar=False)
R5_cov_third = np.cov(R5, rowvar=False)

accuracy_1 = accuracy(T1, y_T1, class1_mean_R1, class2_mean_R1, class3_mean_R1, R1_cov_third, R1_cov_third, R1_cov_third)
accuracy_2 = accuracy(T2, y_T2, class1_mean_R2, class2_mean_R2, class3_mean_R2, R2_cov_third, R2_cov_third, R2_cov_third)
accuracy_3 = accuracy(T3, y_T3, class1_mean_R3, class2_mean_R3, class3_mean_R3, R3_cov_third, R3_cov_third, R3_cov_third)
accuracy_4 = accuracy(T4, y_T4, class1_mean_R4, class2_mean_R4, class3_mean_R4, R4_cov_third, R4_cov_third, R4_cov_third)
accuracy_5 = accuracy(T5, y_T5, class1_mean_R5, class2_mean_R5, class3_mean_R5, R5_cov_third, R5_cov_third, R5_cov_third)
avg_accuracy = (accuracy_1 + accuracy_2 + accuracy_3 + accuracy_4 + accuracy_5)/5
print("The third type of uni-modal Gaussian classifier\n")
print("Accuracy of the classifier on the first experiment: {:.2f}%".format(accuracy_1))
print("Accuracy of the classifier on the second experiment: {:.2f}%".format(accuracy_2))
print("Accuracy of the classifier on the third experiment: {:.2f}%".format(accuracy_3))
print("Accuracy of the classifier on the fourth experiment: {:.2f}%".format(accuracy_4))
```

```
print("Accuracy of the classifier on the fifth experiment: {:.2f}%".
↪format(accuracy_5))
print("Average accuracy of the third type uni-modal Gaussian classifier: {:.
↪2f}%".format(avg_accuracy))
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# The fourth type of uni-modal Gaussian classifier
# Calculating variances for each class
class1_R1_var = np.var(R1[:40])
class2_R1_var = np.var(R1[40:80])
class3_R1_var = np.var(R1[80:120])
class1_R2_var = np.var(R2[:40])
class2_R2_var = np.var(R2[40:80])
class3_R2_var = np.var(R2[80:120])
class1_R3_var = np.var(R3[:40])
class2_R3_var = np.var(R3[40:80])
class3_R3_var = np.var(R3[80:120])
class1_R4_var = np.var(R4[:40])
class2_R4_var = np.var(R4[40:80])
class3_R4_var = np.var(R4[80:120])
class1_R5_var = np.var(R5[:40])
class2_R5_var = np.var(R5[40:80])
class3_R5_var = np.var(R5[80:120])

# Calculating covariance matrices for each class
class1_R1_cov_fourth = class1_R1_var*identity_matrix
class2_R1_cov_fourth = class2_R1_var*identity_matrix
class3_R1_cov_fourth = class3_R1_var*identity_matrix
class1_R2_cov_fourth = class1_R2_var*identity_matrix
class2_R2_cov_fourth = class2_R2_var*identity_matrix
class3_R2_cov_fourth = class3_R2_var*identity_matrix
class1_R3_cov_fourth = class1_R3_var*identity_matrix
class2_R3_cov_fourth = class2_R3_var*identity_matrix
class3_R3_cov_fourth = class3_R3_var*identity_matrix
class1_R4_cov_fourth = class1_R4_var*identity_matrix
class2_R4_cov_fourth = class2_R4_var*identity_matrix
class3_R4_cov_fourth = class3_R4_var*identity_matrix
class1_R5_cov_fourth = class1_R5_var*identity_matrix
class2_R5_cov_fourth = class2_R5_var*identity_matrix
class3_R5_cov_fourth = class3_R5_var*identity_matrix

accuracy_1 = accuracy(T1, y_T1, class1_mean_R1, class2_mean_R1, class3_mean_R1,␣
↪class1_R1_cov_fourth, class2_R1_cov_fourth, class3_R1_cov_fourth)
accuracy_2 = accuracy(T2, y_T2, class1_mean_R2, class2_mean_R2, class3_mean_R2,␣
↪class1_R2_cov_fourth, class2_R2_cov_fourth, class3_R2_cov_fourth)
accuracy_3 = accuracy(T3, y_T3, class1_mean_R3, class2_mean_R3, class3_mean_R3,␣
↪class1_R3_cov_fourth, class2_R3_cov_fourth, class3_R3_cov_fourth)
```

```
accuracy_4 = accuracy(T4, y_T4, class1_mean_R4, class2_mean_R4, class3_mean_R4,␣
 ↪class1_R4_cov_fourth, class2_R4_cov_fourth, class3_R4_cov_fourth)
accuracy_5 = accuracy(T5, y_T5, class1_mean_R5, class2_mean_R5, class3_mean_R5,␣
 ↪class1_R5_cov_fourth, class2_R5_cov_fourth, class3_R5_cov_fourth)
avg_accuracy = (accuracy_1 + accuracy_2 + accuracy_3 + accuracy_4 + accuracy_5)/
 ↪5
print("The fourth type of uni-modal Gaussian classifier\n")
print("Accuracy of the classifier on the first experiment: {:.2f}%".
 ↪format(accuracy_1))
print("Accuracy of the classifier on the second experiment: {:.2f}%".
 ↪format(accuracy_2))
print("Accuracy of the classifier on the third experiment: {:.2f}%".
 ↪format(accuracy_3))
print("Accuracy of the classifier on the fourth experiment: {:.2f}%".
 ↪format(accuracy_4))
print("Accuracy of the classifier on the fifth experiment: {:.2f}%".
 ↪format(accuracy_5))
print("Average accuracy of the fourth type uni-modal Gaussian classifier: {:.
 ↪2f}%".format(avg_accuracy))
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# The fifth type of uni-modal Gaussian classifier
# Calculating covariances for each class
class1_R1_cov_fifth = np.cov(R1[:40], rowvar=False)
class2_R1_cov_fifth = np.cov(R1[40:80], rowvar=False)
class3_R1_cov_fifth = np.cov(R1[80:120], rowvar=False)
class1_R2_cov_fifth = np.cov(R2[:40], rowvar=False)
class2_R2_cov_fifth = np.cov(R2[40:80], rowvar=False)
class3_R2_cov_fifth = np.cov(R2[80:120], rowvar=False)
class1_R3_cov_fifth = np.cov(R3[:40], rowvar=False)
class2_R3_cov_fifth = np.cov(R3[40:80], rowvar=False)
class3_R3_cov_fifth = np.cov(R3[80:120], rowvar=False)
class1_R4_cov_fifth = np.cov(R4[:40], rowvar=False)
class2_R4_cov_fifth = np.cov(R4[40:80], rowvar=False)
class3_R4_cov_fifth = np.cov(R4[80:120], rowvar=False)
class1_R5_cov_fifth = np.cov(R5[:40], rowvar=False)
class2_R5_cov_fifth = np.cov(R5[40:80], rowvar=False)
class3_R5_cov_fifth = np.cov(R5[80:120], rowvar=False)

accuracy_1 = accuracy(T1, y_T1, class1_mean_R1, class2_mean_R1, class3_mean_R1,␣
 ↪class1_R1_cov_fifth, class2_R1_cov_fifth, class3_R1_cov_fifth)
accuracy_2 = accuracy(T2, y_T2, class1_mean_R2, class2_mean_R2, class3_mean_R2,␣
 ↪class1_R2_cov_fifth, class2_R2_cov_fifth, class3_R2_cov_fifth)
accuracy_3 = accuracy(T3, y_T3, class1_mean_R3, class2_mean_R3, class3_mean_R3,␣
 ↪class1_R3_cov_fifth, class2_R3_cov_fifth, class3_R3_cov_fifth)
```

```
accuracy_4 = accuracy(T4, y_T4, class1_mean_R4, class2_mean_R4, class3_mean_R4,␣
 ↪class1_R4_cov_fifth, class2_R4_cov_fifth, class3_R4_cov_fifth)
accuracy_5 = accuracy(T5, y_T5, class1_mean_R5, class2_mean_R5, class3_mean_R5,␣
 ↪class1_R5_cov_fifth, class2_R5_cov_fifth, class3_R5_cov_fifth)
avg_accuracy = (accuracy_1 + accuracy_2 + accuracy_3 + accuracy_4 + accuracy_5)/
 ↪5
print("The fifth type of uni-modal Gaussian classifier:\n")
print("Accuracy of the classifier on the first experiment: {:.2f}%".
 ↪format(accuracy_1))
print("Accuracy of the classifier on the second experiment: {:.2f}%".
 ↪format(accuracy_2))
print("Accuracy of the classifier on the third experiment: {:.2f}%".
 ↪format(accuracy_3))
print("Accuracy of the classifier on the fourth experiment: {:.2f}%".
 ↪format(accuracy_4))
print("Accuracy of the classifier on the fifth experiment: {:.2f}%".
 ↪format(accuracy_5))
print("Average accuracy of the fifth type uni-modal Gaussian classifier: {:.
 ↪2f}%".format(avg_accuracy))
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# The sixth type of uni-modal Gaussian classifier
# Calculating variances for each class
class1_R1_var_axis0 = np.var(R1[:40], axis=0)
class2_R1_var_axis0 = np.var(R1[40:80], axis=0)
class3_R1_var_axis0 = np.var(R1[80:120], axis=0)
class1_R2_var_axis0 = np.var(R2[:40], axis=0)
class2_R2_var_axis0 = np.var(R2[40:80], axis=0)
class3_R2_var_axis0 = np.var(R2[80:120], axis=0)
class1_R3_var_axis0 = np.var(R3[:40], axis=0)
class2_R3_var_axis0 = np.var(R3[40:80], axis=0)
class3_R3_var_axis0 = np.var(R3[80:120], axis=0)
class1_R4_var_axis0 = np.var(R4[:40], axis=0)
class2_R4_var_axis0 = np.var(R4[40:80], axis=0)
class3_R4_var_axis0 = np.var(R4[80:120], axis=0)
class1_R5_var_axis0 = np.var(R5[:40], axis=0)
class2_R5_var_axis0 = np.var(R5[40:80], axis=0)
class3_R5_var_axis0 = np.var(R5[80:120], axis=0)
# Calculating covariance matrices for each class
class1_R1_cov_sixth = class1_R1_var_axis0*identity_matrix
class2_R1_cov_sixth = class2_R1_var_axis0*identity_matrix
class3_R1_cov_sixth = class3_R1_var_axis0*identity_matrix
class1_R2_cov_sixth = class1_R2_var_axis0*identity_matrix
class2_R2_cov_sixth = class2_R2_var_axis0*identity_matrix
class3_R2_cov_sixth = class3_R2_var_axis0*identity_matrix
class1_R3_cov_sixth = class1_R3_var_axis0*identity_matrix
```

```
class2_R3_cov_sixth = class2_R3_var_axis0*identity_matrix
class3_R3_cov_sixth = class3_R3_var_axis0*identity_matrix
class1_R4_cov_sixth = class1_R4_var_axis0*identity_matrix
class2_R4_cov_sixth = class2_R4_var_axis0*identity_matrix
class3_R4_cov_sixth = class3_R4_var_axis0*identity_matrix
class1_R5_cov_sixth = class1_R5_var_axis0*identity_matrix
class2_R5_cov_sixth = class2_R5_var_axis0*identity_matrix
class3_R5_cov_sixth = class3_R5_var_axis0*identity_matrix

accuracy_1 = accuracy(T1, y_T1, class1_mean_R1, class2_mean_R1, class3_mean_R1,␣
 ↪class1_R1_cov_sixth, class2_R1_cov_sixth, class3_R1_cov_sixth)
accuracy_2 = accuracy(T2, y_T2, class1_mean_R2, class2_mean_R2, class3_mean_R2,␣
 ↪class1_R2_cov_sixth, class2_R2_cov_sixth, class3_R2_cov_sixth)
accuracy_3 = accuracy(T3, y_T3, class1_mean_R3, class2_mean_R3, class3_mean_R3,␣
 ↪class1_R3_cov_sixth, class2_R3_cov_sixth, class3_R3_cov_sixth)
accuracy_4 = accuracy(T4, y_T4, class1_mean_R4, class2_mean_R4, class3_mean_R4,␣
 ↪class1_R4_cov_sixth, class2_R4_cov_sixth, class3_R4_cov_sixth)
accuracy_5 = accuracy(T5, y_T5, class1_mean_R5, class2_mean_R5, class3_mean_R5,␣
 ↪class1_R5_cov_sixth, class2_R5_cov_sixth, class3_R5_cov_sixth)
avg_accuracy = (accuracy_1 + accuracy_2 + accuracy_3 + accuracy_4 + accuracy_5)/
 ↪5
print("The sixth type of uni-modal Gaussian classifier:\n")
print("Accuracy of the classifier on the first experiment: {:.2f}%".
 ↪format(accuracy_1))
print("Accuracy of the classifier on the second experiment: {:.2f}%".
 ↪format(accuracy_2))
print("Accuracy of the classifier on the third experiment: {:.2f}%".
 ↪format(accuracy_3))
print("Accuracy of the classifier on the fourth experiment: {:.2f}%".
 ↪format(accuracy_4))
print("Accuracy of the classifier on the fifth experiment: {:.2f}%".
 ↪format(accuracy_5))
print("Average accuracy of the sixth type uni-modal Gaussian classifier: {:.
 ↪2f}%".format(avg_accuracy))
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
The first type of uni-modal Gaussian classifier

Accuracy of the classifier on the first experiment: 96.67%
Accuracy of the classifier on the second experiment: 93.33%
Accuracy of the classifier on the third experiment: 86.67%
Accuracy of the classifier on the fourth experiment: 93.33%
Accuracy of the classifier on the fifth experiment: 90.00%
Average accuracy of the first type uni-modal Gaussian classifier: 92.00%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
```

```
The second type of uni-modal Gaussian classifier

Accuracy of the classifier on the first experiment: 90.00%
Accuracy of the classifier on the second experiment: 86.67%
Accuracy of the classifier on the third experiment: 86.67%
Accuracy of the classifier on the fourth experiment: 90.00%
Accuracy of the classifier on the fifth experiment: 80.00%
Average accuracy of the second type uni-modal Gaussian classifier: 86.67%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
The third type of uni-modal Gaussian classifier

Accuracy of the classifier on the first experiment: 93.33%
Accuracy of the classifier on the second experiment: 83.33%
Accuracy of the classifier on the third experiment: 86.67%
Accuracy of the classifier on the fourth experiment: 86.67%
Accuracy of the classifier on the fifth experiment: 73.33%
Average accuracy of the third type uni-modal Gaussian classifier: 84.67%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
The fourth type of uni-modal Gaussian classifier

Accuracy of the classifier on the first experiment: 100.00%
Accuracy of the classifier on the second experiment: 93.33%
Accuracy of the classifier on the third experiment: 90.00%
Accuracy of the classifier on the fourth experiment: 93.33%
Accuracy of the classifier on the fifth experiment: 90.00%
Average accuracy of the fourth type uni-modal Gaussian classifier: 93.33%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
The fifth type of uni-modal Gaussian classifier:

Accuracy of the classifier on the first experiment: 100.00%
Accuracy of the classifier on the second experiment: 96.67%
Accuracy of the classifier on the third experiment: 93.33%
Accuracy of the classifier on the fourth experiment: 96.67%
Accuracy of the classifier on the fifth experiment: 100.00%
Average accuracy of the fifth type uni-modal Gaussian classifier: 97.33%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
The sixth type of uni-modal Gaussian classifier:

Accuracy of the classifier on the first experiment: 100.00%
Accuracy of the classifier on the second experiment: 93.33%
Accuracy of the classifier on the third experiment: 93.33%
Accuracy of the classifier on the fourth experiment: 96.67%
Accuracy of the classifier on the fifth experiment: 86.67%
Average accuracy of the sixth type uni-modal Gaussian classifier: 94.00%
```

~~~~~~
~~~~~~~~~~~~~~~~~~~~~

4 3. Support Vector Machines

5 1. Determine the best C and degree of the polynomial kernel functions

[5]:
```
from sklearn.svm import SVC
import pandas as pd

# Setting different values for the parameters
C = [1,10,100,1000]
degree = [1,2,3]

# Creating folds for the results of each experiment
first_exp = []
second_exp = []
third_exp = []
fourth_exp = []
fifth_exp = []

# for loops to conduct experiments with different parameters
for c in C:
    for d in degree:
        svc = SVC(kernel='poly', C=c, degree=d, gamma='scale').fit(R1, y_R1)
        results1 = ("{:.3f}").format(svc.score(T1, y_T1))
        first_exp.append(results1)
        svc = SVC(kernel='poly', C=c, degree=d, gamma='scale').fit(R2, y_R2)
        results2 = ("{:.3f}").format(svc.score(T2, y_T2))
        second_exp.append(results2)
        svc = SVC(kernel='poly', C=c, degree=d, gamma='scale').fit(R3, y_R3)
        results3= ("{:.3f}").format(svc.score(T3, y_T3))
        third_exp.append(results3)
        svc = SVC(kernel='poly', C=c, degree=d, gamma='scale').fit(R4, y_R4)
        results4 = ("{:.3f}").format(svc.score(T4, y_T4))
        fourth_exp.append(results4)
        svc = SVC(kernel='poly', C=c, degree=d, gamma='scale').fit(R5, y_R5)
        results5 = ("{:.3f}").format(svc.score(T5, y_T5))
        fifth_exp.append(results5)

# Changing values into float
first_exp = np.float64(first_exp)
second_exp = np.float64(second_exp)
third_exp = np.float64(third_exp)
fourth_exp = np.float64(fourth_exp)
```

```
fifth_exp = np.float64(fifth_exp)

# Creating dataframe to compare results
exp = ["First experiment", "Second experiment", "Third experiment", "Fourth␣
↪experiment", "Fifth experiment"]
df = pd.DataFrame([first_exp, second_exp, third_exp, fourth_exp, fifth_exp],␣
↪index=exp)

# Rename columns of the dataframe
df.rename(columns={0: "C=1, degree=1"}, inplace=True)
df.rename(columns={1: "C=1, degree=2"}, inplace=True)
df.rename(columns={2: "C=1, degree=3"}, inplace=True)
df.rename(columns={3: "C=10, degree=1"}, inplace=True)
df.rename(columns={4: "C=10, degree=2"}, inplace=True)
df.rename(columns={5: "C=10, degree=3"}, inplace=True)
df.rename(columns={6: "C=100, degree=1"}, inplace=True)
df.rename(columns={7: "C=100, degree=2"}, inplace=True)
df.rename(columns={8: "C=100, degree=3"}, inplace=True)
df.rename(columns={9: "C=1000, degree=1"}, inplace=True)
df.rename(columns={10: "C=1000, degree=2"}, inplace=True)
df.rename(columns={11: "C=1000, degree=3"}, inplace=True)

# Creating dictionary to find out the best parameters
mean_scores = {"C=1, degree=1": df["C=1, degree=1"].mean(), "C=1, degree=2":␣
↪df["C=1, degree=2"].mean(), "C=1, degree=3": df["C=1, degree=3"].mean(),
              "C=10, degree=1": df["C=10, degree=1"].mean(), "C=10, degree=2":␣
↪df["C=10, degree=2"].mean(), "C=10, degree=3": df["C=10, degree=3"].mean(),
              "C=100, degree=1": df["C=100, degree=1"].mean(), "C=100,␣
↪degree=2": df["C=100, degree=2"].mean(), "C=100, degree=3": df["C=100,␣
↪degree=3"].mean(),
              "C=1000, degree=1": df["C=1000, degree=1"].mean(), "C=1000,␣
↪degree=2": df["C=1000, degree=2"].mean(), "C=1000, degree=3": df["C=1000,␣
↪degree=3"].mean()}
the_best_parameters = max(mean_scores, key=mean_scores.get)
print(the_best_parameters)
df
```

```
C=1, degree=2
```

[5]:

	C=1, degree=1	C=1, degree=2	C=1, degree=3 \
First experiment	1.000	1.000	1.000
Second experiment	0.933	0.967	0.967
Third experiment	0.933	1.000	0.967
Fourth experiment	0.967	1.000	1.000
Fifth experiment	0.933	0.967	0.967

```
                    C=10, degree=1  C=10, degree=2  C=10, degree=3  \
```

```
First experiment              1.000          1.000          1.000
Second experiment             0.967          0.967          0.933
Third experiment              1.000          0.900          0.900
Fourth experiment             1.000          1.000          1.000
Fifth experiment              0.967          0.967          1.000

                   C=100, degree=1  C=100, degree=2  C=100, degree=3  \
First experiment             1.000            1.000            1.000
Second experiment            0.967            0.933            0.933
Third experiment             0.900            0.900            0.900
Fourth experiment            1.000            1.000            1.000
Fifth experiment             1.000            1.000            1.000

                   C=1000, degree=1  C=1000, degree=2  C=1000, degree=3
First experiment              1.000             1.000             1.000
Second experiment             0.933             0.933             0.933
Third experiment              0.900             0.900             0.900
Fourth experiment             1.000             1.000             0.967
Fifth experiment              1.000             1.000             0.967
```

5.0.1 2. Determine the best C and standard deviation of the Gaussian kernel function

[6]:
```python
from sklearn.svm import SVC
import pandas as pd

# Setting different values for the parameters
C = [1,10,100,1000]
sigma = [1e-3, 1e-2, 1e-1, 1, 2]

# Creating folds for the results of each experiment
first_exp = []
second_exp = []
third_exp = []
fourth_exp = []
fifth_exp = []

# for loops to conduct experiments with different parameters
for c in C:
    for sigma_value in sigma:
        svc = SVC(kernel='rbf', C=c, gamma=sigma_value).fit(R1, y_R1)
        results1 = ("{:.3f}").format(svc.score(T1, y_T1))
        first_exp.append(results1)
        svc = SVC(kernel='rbf', C=c, gamma=sigma_value).fit(R2, y_R2)
        results2 = ("{:.3f}").format(svc.score(T2, y_T2))
        second_exp.append(results2)
        svc = SVC(kernel='rbf', C=c, gamma=sigma_value).fit(R3, y_R3)
        results3= ("{:.3f}").format(svc.score(T3, y_T3))
```

```
        third_exp.append(results3)
        svc = SVC(kernel='rbf', C=c, gamma=sigma_value).fit(R4, y_R4)
        results4 = ("{:.3f}").format(svc.score(T4, y_T4))
        fourth_exp.append(results4)
        svc = SVC(kernel='rbf', C=c, gamma=sigma_value).fit(R5, y_R5)
        results5 = ("{:.3f}").format(svc.score(T5, y_T5))
        fifth_exp.append(results5)

# Changing values into float
first_exp = np.float64(first_exp)
second_exp = np.float64(second_exp)
third_exp = np.float64(third_exp)
fourth_exp = np.float64(fourth_exp)
fifth_exp = np.float64(fifth_exp)

# Creating a dataframe to compare results
exp = ["First experiment", "Second experiment", "Third experiment", "Fourth␣
 ↪experiment", "Fifth experiment"]
df = pd.DataFrame([first_exp, second_exp, third_exp, fourth_exp, fifth_exp],␣
 ↪index=exp)

# Rename columns of the dataframe
df.rename(columns={0: "C=1, sigma=1e-3"}, inplace=True)
df.rename(columns={1: "C=1, sigma=1e-2"}, inplace=True)
df.rename(columns={2: "C=1, sigma=1e-1"}, inplace=True)
df.rename(columns={3: "C=1, sigma=1"}, inplace=True)
df.rename(columns={4: "C=1, sigma=2"}, inplace=True)
df.rename(columns={5: "C=10, sigma=1e-3"}, inplace=True)
df.rename(columns={6: "C=10, sigma=1e-2"}, inplace=True)
df.rename(columns={7: "C=10, sigma=1e-1"}, inplace=True)
df.rename(columns={8: "C=10, sigma=1"}, inplace=True)
df.rename(columns={9: "C=10, sigma=2"}, inplace=True)
df.rename(columns={10: "C=100, sigma=1e-3"}, inplace=True)
df.rename(columns={11: "C=100, sigma=1e-2"}, inplace=True)
df.rename(columns={12: "C=100, sigma=1e-1"}, inplace=True)
df.rename(columns={13: "C=100, sigma=1"}, inplace=True)
df.rename(columns={14: "C=100, sigma=2"}, inplace=True)
df.rename(columns={15: "C=1000, sigma=1e-3"}, inplace=True)
df.rename(columns={16: "C=1000, sigma=1e-2"}, inplace=True)
df.rename(columns={17: "C=1000, sigma=1e-1"}, inplace=True)
df.rename(columns={18: "C=1000, sigma=1"}, inplace=True)
df.rename(columns={19: "C=1000, sigma=2"}, inplace=True)

# Creating dictionary to find out the best parameters
mean_scores = {"C=1, sigma=1e-3": df["C=1, sigma=1e-3"].mean(), "C=1,␣
 ↪sigma=1e-2": df["C=1, sigma=1e-2"].mean(),
```

```
                "C=1, sigma=1e-1": df["C=1, sigma=1e-1"].mean(), "C=1, sigma=1":␣
↪df["C=1, sigma=1"].mean(),
                "C=1, sigma=2": df["C=1, sigma=2"].mean(), "C=10, sigma=1e-3":␣
↪df["C=10, sigma=1e-3"].mean(),
                "C=10, sigma=1e-2": df["C=10, sigma=1e-2"].mean(), "C=10,␣
↪sigma=1e-1": df["C=10, sigma=1e-1"].mean(),
                "C=10, sigma=1": df["C=10, sigma=1"].mean(), "C=10, sigma=2":␣
↪df["C=10, sigma=2"].mean(),
                "C=100, sigma=1e-3": df["C=100, sigma=1e-3"].mean(), "C=100,␣
↪sigma=1e-2": df["C=100, sigma=1e-2"].mean(),
                "C=100, sigma=1e-1": df["C=100, sigma=1e-1"].mean(), "C=100,␣
↪sigma=1": df["C=100, sigma=1"].mean(),
                "C=100, sigma=2": df["C=100, sigma=2"].mean(), "C=1000,␣
↪sigma=1e-3": df["C=1000, sigma=1e-3"].mean(),
                "C=1000, sigma=1e-2": df["C=1000, sigma=1e-2"].mean(), "C=1000,␣
↪sigma=1e-1": df["C=1000, sigma=1e-1"].mean(),
                "C=1000, sigma=1": df["C=1000, sigma=1"].mean(), "C=1000,␣
↪sigma=2": df["C=1000, sigma=2"].mean()}
the_best_parameters = max(mean_scores, key=mean_scores.get)
print(the_best_parameters)
df
```

```
C=1, sigma=1e-1
```

[6]:

```
                   C=1, sigma=1e-3  C=1, sigma=1e-2  C=1, sigma=1e-1  \
First experiment             0.933            1.000            1.000
Second experiment            0.967            0.933            0.967
Third experiment             0.833            0.867            0.967
Fourth experiment            0.967            0.967            1.000
Fifth experiment             0.867            0.900            0.967

                   C=1, sigma=1  C=1, sigma=2  C=10, sigma=1e-3  \
First experiment          1.000         1.000             1.000
Second experiment         0.967         0.967             0.933
Third experiment          0.900         0.900             0.867
Fourth experiment         1.000         1.000             0.967
Fifth experiment          0.967         0.967             0.900

                   C=10, sigma=1e-2  C=10, sigma=1e-1  C=10, sigma=1  \
First experiment              1.000             1.000          1.000
Second experiment             0.967             0.967          0.933
Third experiment              0.967             0.967          0.900
Fourth experiment             1.000             1.000          0.967
Fifth experiment              0.967             0.967          0.967

                   C=10, sigma=2  C=100, sigma=1e-3  C=100, sigma=1e-2  \
First experiment           1.000              1.000              1.000
```

```
Second experiment          0.933              0.967              0.967
Third experiment           0.900              0.967              0.967
Fourth experiment          0.967              1.000              1.000
Fifth experiment           1.000              0.967              0.967

                   C=100, sigma=1e-1  C=100, sigma=1  C=100, sigma=2  \
First experiment               1.000           0.933           0.967
Second experiment              0.933           0.933           0.933
Third experiment               0.900           0.900           0.900
Fourth experiment              1.000           0.967           0.967
Fifth experiment               1.000           0.967           1.000

                   C=1000, sigma=1e-3  C=1000, sigma=1e-2  C=1000, sigma=1e-1  \
First experiment                1.000               1.000               1.000
Second experiment               0.967               0.933               0.933
Third experiment                0.967               0.900               0.900
Fourth experiment               1.000               1.000               0.967
Fifth experiment                0.967               1.000               0.967

                   C=1000, sigma=1  C=1000, sigma=2
First experiment             0.933            0.967
Second experiment            0.933            0.933
Third experiment             0.867            0.900
Fourth experiment            0.967            0.967
Fifth experiment             0.967            1.000
```