Лабораторна робота №3

Тема: Застосування дискримінантного аналізу в процесі визначення переліку станів модельованого об’єкта.

Мета: Закріпити навички проектування програм для дискримінантного аналізу опису станів досліджуваних об’єктів.

Хід роботи

Після ознайомлення із теорією дискримінатного аналізу, було вирішено написати невеликий python-скрипт, що вирішує дану задачу, використовуючи сучасні та потужні бібліотеки обробки даних. Це дозволило швидко виконати завдання та не писати при цьому великі обсяги коду. Варто зазначити, що використовувалась бібліотека `numpy` для обчислень та `sklearn.discriminant_analysis` для виконання аналізу. Клас LinearDiscriminantAnalysis може містити в собі використання дещо інших підходів чи методів, ніж було вказано у теоретичній частині, проте це не впливає на отриманий результат. Кроки виконання аналізу ідентичні до теорії, але значення можуть різнитись в залежності від формул, що використовує бібліотека.

```
# Створення та тренування моделі LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X, y)

# Підготовка нових об'єктів для передбачення
new_objects = np.array([[75, 9.6, 18.5], [95, 12.5, 16.1]])

# Передбачення класів для нових об'єктів
predictions = lda.predict(new_objects)

# Ймовірності кожного класу для кожного нового об'єкта
class_probabilities = lda.predict_proba(new_objects)

# Отримання оцінок коваріаційних матриць для кожного класу
for class_label in np.unique(y):
    class_indices = np.where(y == class_label)
    class_data = X[class_indices]
    class_covariance = np.cov(class_data, rowvar=False)
    covariances.append(class_covariance)

# Розрахунок незміщеної оцінки об'єднаної коваріаційної матриці
n1 = len(firstClass)
n2 = len(secondClass)
cov_matrix_group1 = np.cov(firstClass, rowvar=False)
cov_matrix_group2 = np.cov(secondClass, rowvar=False)
pooled_covariance = ((n1 - 1) * cov_matrix_group1 + (n2 - 1) * cov_matrix_group2) / (n1 + n2 - 2)

# Вивід результатів
print("\nНові об'єкти:\n", new_objects)

# Вивід класів та їх середніх значень
class_names = np.unique(y)
for i, class_label in enumerate(class_names):
    class_mean_values = lda.means_[i]
    class_means_with_labels = np.vstack((firstClass if class_label == "перший" else secondClass, class_mean_values))
    print(f"\nКлас '{class_label} із середніми значеннями':\n{class_means_with_labels}")

# Вивід коваріаційних матриць
for i, class_label in enumerate(class_names):
    print(f"\nКоваріаційна матриця для класу '{class_label}':\n{covariances[i]}")
```

Рисунок 1. Використання LinearDiscriminantAnalysis для аналізу

Запустивши скріпт для дискримінатного аналізу за прикладом із теоретичної частини, отримали такий результат:

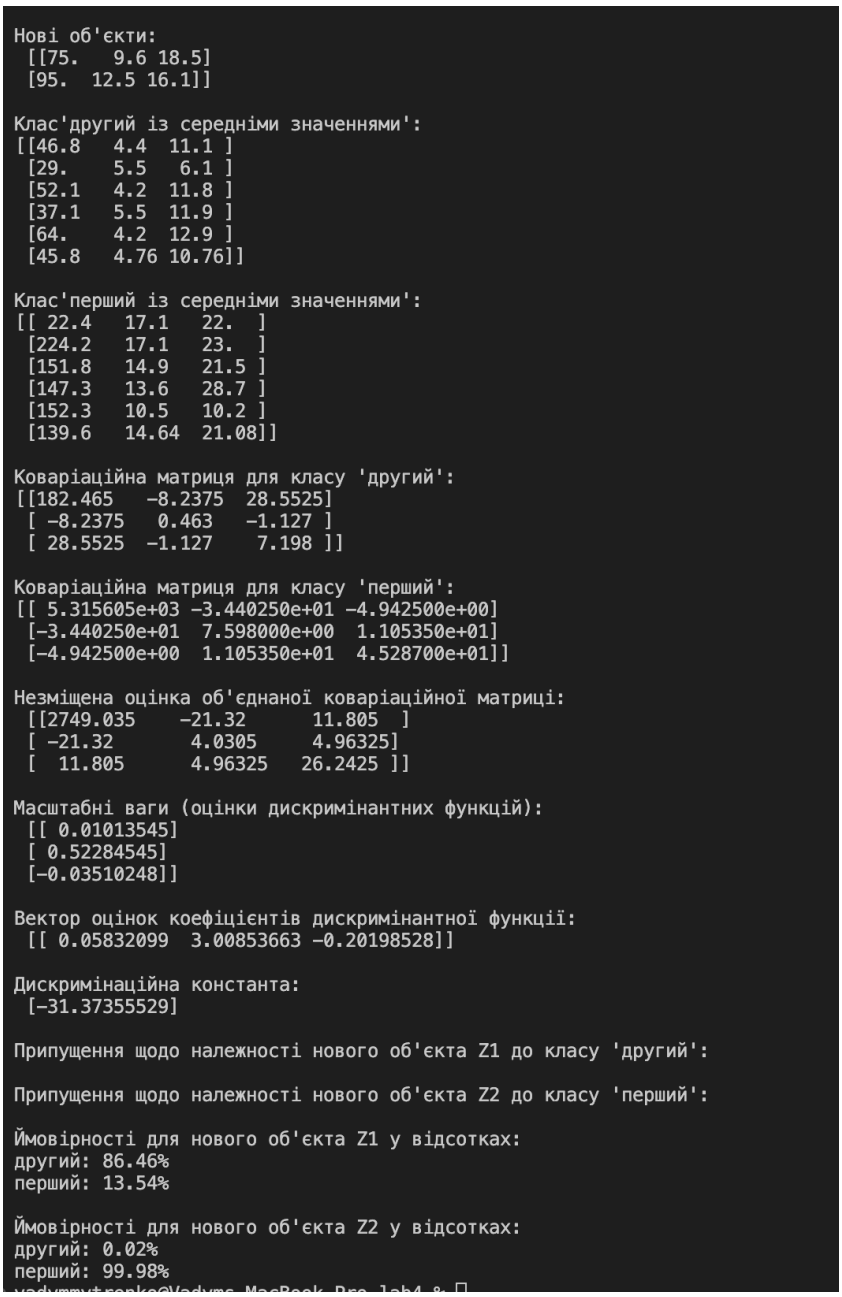

Рисунок 2. Результат виконання скрипта для значень із теоретичної частини

Як уже і було зазначено, деякі проміжні результати відрізняються із-за різних формул, що використовує бібліотека, проте припущення до належності об’єктів виконано правильно, що показує на коректність роботи алгоритму.

Далі, було виконано обчислення за варіантом:

2

0,31	15,5	1,2	105
0,52	12,3	1,15	55
0,43	11,2	1,18	75
0,73	17,7	1,17	100
0,94	16,8	1,22	115
0,83	12,5	1,16	85
0,54	12,2	1,17	80

0,25	15,3	1,5	134
0,18	14,0	1,8	93
0,45	16,8	1,55	126
0,18	18,1	1,18	90
0,35	19,0	1,35	139
0,43	18,2	1,43	113
0,29	14,3	1,29	104

0,4	14,23	1,21	100
0,22	19,02	1,38	87

Рисунок 3. Варіант для тестування системи

Замінимо перший та другий клас у коді, а також відповідні мітки класів і вхідні значення Z:

```
# Задання даних для першого та другого класу
firstClass = np.array([
    [0.31, 15.5, 1.2, 105],
    [0.52, 12.3, 1.15, 55],
    [0.43, 11.2, 1.18, 75],
    [0.73, 17.7, 1.17, 100],
    [0.94, 16.8, 1.22, 115],
    [0.83, 12.5, 1.16, 85],
    [0.54, 12.2, 1.17, 80]
])

secondClass = np.array([
    [0.25, 15.3, 1.5, 134],
    [0.18, 14.0, 1.8, 93],
    [0.45, 16.8, 1.55, 126],
    [0.18, 18.1, 1.18, 90],
    [0.35, 19.0, 1.35, 139],
    [0.43, 18.2, 1.43, 113],
    [0.29, 14.3, 1.29, 104]
])

# Об'єднання даних у одну матрицю
X = np.vstack((firstClass, secondClass))

# Задання міток класів
y = np.array(["перший", "перший", "перший", "перший", "перший", "перший", "перший",
              "другий", "другий", "другий", "другий", "другий", "другий", "другий"])

# Створення та тренування моделі LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X, y)

# Підготовка нових об'єктів для передбачення
new_objects = np.array([[0.4, 14.23, 1.21, 100], [0.22, 19.02, 1.38, 87]])
```

Рисунок 4. Зміна вхідних відповідно до варіанту

```
Нові об'єкти:
 [[  0.4   14.23   1.21 100.  ]
 [  0.22  19.02   1.38  87.  ]]

Клас 'другий' із середніми значеннями:
[[  0.25        15.3         1.5        134.        ]
 [  0.18        14.          1.8         93.        ]
 [  0.45        16.8         1.55       126.        ]
 [  0.18        18.1         1.18        90.        ]
 [  0.35        19.          1.35       139.        ]
 [  0.43        18.2         1.43       113.        ]
 [  0.29        14.3         1.29       104.        ]
 [  0.30428571  16.52857143  1.44285714 114.14285714]]

Клас 'перший' із середніми значеннями:
[[  0.31        15.5         1.2        105.        ]
 [  0.52        12.3         1.15        55.        ]
 [  0.43        11.2         1.18        75.        ]
 [  0.73        17.7         1.17       100.        ]
 [  0.94        16.8         1.22       115.        ]
 [  0.83        12.5         1.16        85.        ]
 [  0.54        12.2         1.17        80.        ]
 [  0.61428571  14.02857143  1.17857143  87.85714286]]

Коваріаційна матриця для класу 'другий':
[[ 1.21952381e-02  9.66904762e-02 -4.80952381e-04  1.23595238e+00]
 [ 9.66904762e-02  4.05238095e+00 -2.11261905e-01  1.27452381e+01]
 [-4.80952381e-04 -2.11261905e-01  4.05904762e-02  7.61904762e-02]
 [ 1.23595238e+00  1.27452381e+01  7.61904762e-02  3.81142857e+02]]

Коваріаційна матриця для класу 'перший':
[[5.11619048e-02 2.28357143e-01 8.40476190e-04 1.74404762e+00]
 [2.28357143e-01 6.66571429e+00 3.40476190e-02 4.28214286e+01]
 [8.40476190e-04 3.40476190e-02 5.80952381e-04 4.04761905e-01]
 [1.74404762e+00 4.28214286e+01 4.04761905e-01 4.15476190e+02]]

Незміщена оцінка об'єднаної коваріаційної матриці:
 [[ 3.16785714e-02  1.62523810e-01  1.79761905e-04  1.49000000e+00]
 [ 1.62523810e-01  5.35904762e+00 -8.86071429e-02  2.77833333e+01]
 [ 1.79761905e-04 -8.86071429e-02  2.05857143e-02  2.40476190e-01]
 [ 1.49000000e+00  2.77833333e+01  2.40476190e-01  3.98309524e+02]]

Масштабні ваги (оцінки дискримінантних функцій):
 [[ 4.71505166]
 [-0.27279119]
 [-4.52494928]
 [-0.01371412]]

Вектор оцінок коефіцієнтів дискримінантної функції:
 [[ 17.4457334   -1.00932985 -16.74235289  -0.05074238]]

Дискримінаційна константа:
 [34.47796328]

Припущення щодо належності нового об'єкта Z1 до класу 'перший':

Припущення щодо належності нового об'єкта Z2 до класу 'другий':

Ймовірності для нового об'єкта Z1 у відсотках:
другий: 14.67%
перший: 85.33%

Ймовірності для нового об'єкта Z2 у відсотках:
другий: 99.98%
перший: 0.02%
```

Рисунок 5. Результат роботи системи дискрімантного аналізу для визначення класів

У результаті роботи, новий об’єкт Z1 належить до першого класу із імовірністю 85.33%, а об’єкт Z2 належить до другого із імовірністю 99.98%. Високі імовірності свідчать про правильну роботу алгоритму із

дискримінатного аналізу.

Вихідний код даної системи можна побачити на GitHub репозиторії за посиланням: https://github.com/VadymMytr/ISPPR

Висновки: під час виконання даної лабораторної роботи було розібрано використання дискримінантного аналізу для класового розподілу. Також було створено короткий скрипт, що виконує аналіз за допомогою сучасних та потужних бібліотек. Правильність роботи системи перевірена на тестових даних, а також було виконано прогін системи по даних за варіантом.