Лабораторна 4: Відновлення щільності розподілення

Даними в даному завданні є вимірювання деяких перевірених параметрів на конвейєрах збірки обладнання Bosh (див. [конкурс (https://www.kaggle.com/c/bosch-production-line-performance)](https://www.kaggle.com/c/bosch-production-line-performance) «Bosch Production Line Performance» на Kaggle).

Всі початкові дані Bosh не вміщуються до оперативної пам'яті комп'ютера, тому в файлі `data.csv` — приведено лише декілька ознак. Прочитайте дані з файлу `data.csv` . Цільовою ознакою тут є `Response` — наявність браку на виробництві.

In [1]:
```
import pandas as pd
```

In [6]:
```
data = pd.read_csv("data.csv")
```

1

побудуйте на одному графіку дла наближення до щільності розподілення ознаки `L1_S24_F1846` для `Response = 0` та для `Response = 1` , використовуючи одне з наступних ядер (номер ядра оберіть за формулою: `(n mod 6) + 1` , де `n` — ваш номер в списку групи):

```
1.      кусочно-постійне (прямокутне) -  tophat
2.      гаусовске -  gaussian
3.      лінійне (трикутник) -  linear
4.      косінусоідальне -  cosine
5.      квадратичне (Епанечникова) -  epanechnikov
6.      експоненціальне -  exponential
```

In [5]:
```
from sklearn.neighbors import KernelDensity
import numpy as np

import matplotlib.pyplot as plt
```

Help:

```
i0 = data['Response'] == 0
kde0 = KernelDensity(kernel='gaussian', bandwidth=0.1).fit(data.loc[i
0, 'L1_S24_F1846'].values.reshape(-1, 1))
X_plot = np.linspace(-1, 1, 1000).reshape(-1, 1)
Dens0 = np.exp(kde0.score_samples(X_plot))  # score_samples возвращает
логарифм плотности
```

In [23]:
```
i0 = data['Response'] == 0
kde0 = KernelDensity(kernel='gaussian', bandwidth=0.1).fit(data.loc[i0, 'L1_S2
X_plot = np.linspace(-1, 1, 1000).reshape(-1, 1)
Dens0 = np.exp(kde0.score_samples(X_plot))
Dens0
```

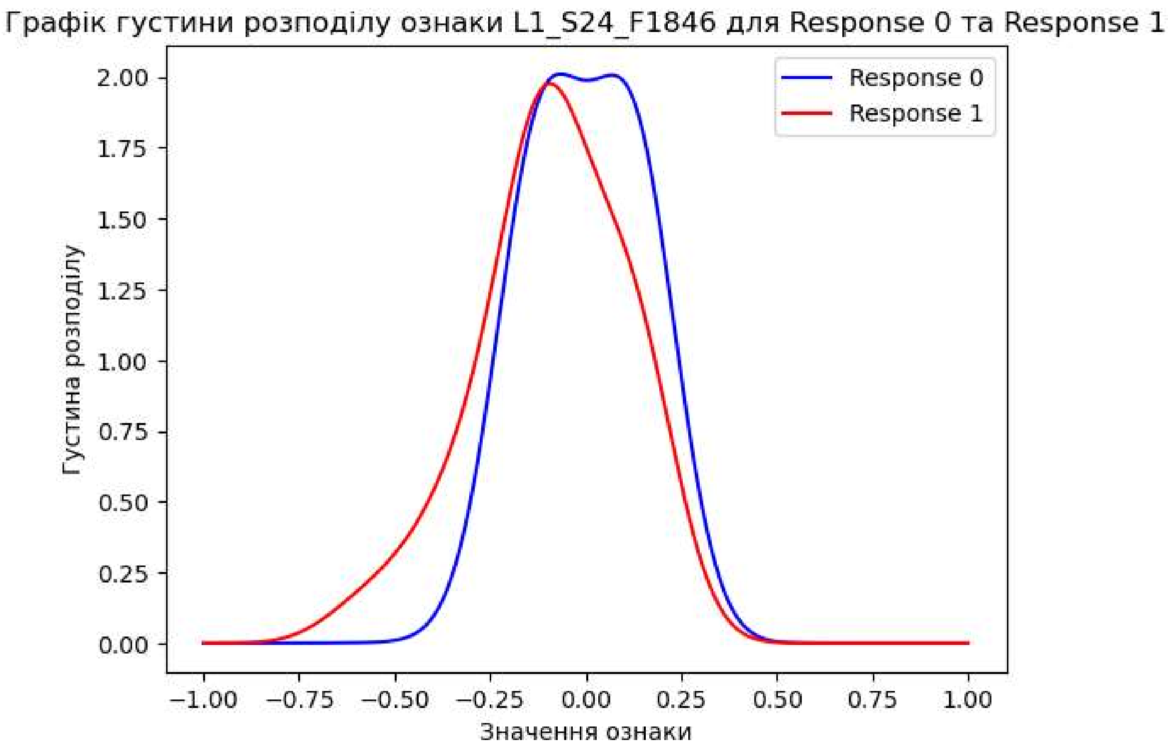
```
       5.58988227e-06, 5.94702423e-06, 6.32461334e-06, 6.72365817e-06,
       7.14520671e-06, 7.59034733e-06, 8.06020966e-06, 8.55596549e-06,
       9.07882967e-06, 9.63006096e-06, 1.02109628e-05, 1.08228843e-05,
       1.14672207e-05, 1.21454145e-05, 1.28589557e-05, 1.36093828e-05,
       1.43982836e-05, 1.52272950e-05, 1.60981045e-05, 1.70124499e-05,
       1.79721200e-05, 1.89789551e-05, 2.00348470e-05, 2.11417393e-05,
       2.23016279e-05, 2.35165603e-05, 2.47886365e-05, 2.61200081e-05,
       2.75128788e-05, 2.89695038e-05, 3.04921892e-05, 3.20832923e-05,
       3.37452204e-05, 3.54804303e-05, 3.72914278e-05, 3.91807668e-05,
       4.11510482e-05, 4.32049187e-05, 4.53450703e-05, 4.75742383e-05,
       4.98952003e-05, 5.23107748e-05, 5.48238193e-05, 5.74372286e-05,
       6.01539334e-05, 6.29768980e-05, 6.59091181e-05, 6.89536191e-05,
       7.21134536e-05, 7.53916989e-05, 7.87914549e-05, 8.23158410e-05,
       8.59679941e-05, 8.97510652e-05, 9.36682171e-05, 9.77226209e-05,
       1.01917454e-04, 1.06255895e-04, 1.10741123e-04, 1.15376313e-04,
       1.20164635e-04, 1.25109244e-04, 1.30213286e-04, 1.35479887e-04,
       1.40912156e-04, 1.46513175e-04, 1.52286001e-04, 1.58233661e-04,
       1.64359147e-04, 1.70665416e-04, 1.77155384e-04, 1.83831926e-04,
       1.90697868e-04, 1.97755990e-04, 2.05009021e-04, 2.12459633e-04,
       2.20110446e-04, 2.27964017e-04, 2.36022847e-04, 2.44289371e-04,
```

Дайте відповідь в коментарцяз на питання: чи є вибірка такою що гарно розділена за однакою `L1_S24_F1846` ?

In [42]:
```
i1 = data['Response'] == 1
kde1 = KernelDensity(kernel='gaussian', bandwidth=0.1).fit(data.loc[i1, 'L1_S24
X_plot = np.linspace(-1, 1, 1000).reshape(-1, 1)
Dens1 = np.exp(kde1.score_samples(X_plot))
Dens1
```

```
       1.89024924e-04, 2.02739907e-04, 2.17363502e-04, 2.32953663e-04,
       2.49567071e-04, 2.67259239e-04, 2.86094616e-04, 3.06138702e-04,
       3.27460157e-04, 3.50130913e-04, 3.74226289e-04, 3.99825108e-04,
       4.27009815e-04, 4.55866592e-04, 4.86485477e-04, 5.18960490e-04,
       5.53389744e-04, 5.89875573e-04, 6.28524646e-04, 6.69448092e-04,
       7.12761616e-04, 7.58585618e-04, 8.07045312e-04, 8.58270838e-04,
       9.12397379e-04, 9.69565271e-04, 1.02992011e-03, 1.09361287e-03,
       1.16079997e-03, 1.23164343e-03, 1.30631092e-03, 1.38497586e-03,
       1.46781751e-03, 1.55502105e-03, 1.64677766e-03, 1.74328456e-03,
       1.84474510e-03, 1.95136880e-03, 2.06337140e-03, 2.18097489e-03,
       2.30440752e-03, 2.43390386e-03, 2.56970479e-03, 2.71205748e-03,
       2.86121539e-03, 3.01743825e-03, 3.18099202e-03, 3.35214885e-03,
       3.53118700e-03, 3.71839078e-03, 3.91405046e-03, 4.11846218e-03,
       4.33192778e-03, 4.55475477e-03, 4.78725607e-03, 5.02974993e-03,
       5.28255973e-03, 5.54601378e-03, 5.82044510e-03, 6.10619122e-03,
       6.40359393e-03, 6.71299902e-03, 7.03475600e-03, 7.36921780e-03,
       7.71674050e-03, 8.07768296e-03, 8.45240650e-03, 8.84127456e-03,
       9.24465228e-03, 9.66290615e-03, 1.00964036e-02, 1.05455125e-02,
       1.10106010e-02, 1.14920366e-02, 1.19901862e-02, 1.25054152e-02,
       1.30380875e-02, 1.35885643e-02, 1.41572042e-02, 1.47443627e-02,
```

In [43]:
```
plt.plot(X_plot, Dens0, label='Response 0', color='b')
plt.plot(X_plot, Dens1, label='Response 1', color='r')
plt.xlabel('Значення ознаки')
plt.ylabel('Густина розподілу')
plt.title('Графік густини розподілу ознаки L1_S24_F1846 для Response 0 та Respo
plt.legend()
plt.show()
```



2

Розбийте вибірку `data` на дві рівні части: тренувальну `dataTrain` и перевірочну `dataTest` .

In [46]:
```
dataTrain = data.loc[0:data.shape[0] / 2, ].reset_index(drop=True)
dataTest = data.loc[data.shape[0] / 2:data.shape[0], ].reset_index(drop=True)
```

Використовуючи крос-валідацію, підберіть **для каодного класу `Response` (`r=0` и `r=1`)** значення ширини ядра `bandwidth` , при якому логарифм правдоподібності максимальний на перевірочній виборці.

Help:

```
r = 0
kde0 = KernelDensity(kernel='gaussian', bandwidth=0.05)
kde0.fit(dataTrain.loc[dataTrain['Response']==r, 'L1_S24_F1846'].value
s.reshape(-1, 1))
logProbability0 = kde0.score_samples(dataTest.loc[dataTest['Response']
```

In [87]:
```
r = 0
kde0 = KernelDensity(kernel='gaussian', bandwidth=0.05)
kde0.fit(dataTrain.loc[dataTrain['Response']==r, 'L1_S24_F1846'].values.reshap
logProbability0 = kde0.score_samples(dataTest.loc[dataTest['Response']==r, 'L1_
logProbability0[np.isinf(logProbability0)] = -100 # замінюємо нескінченність
logLikehood0 = logProbability0.sum()
print(logLikehood0)
```

```
20544.80746112119
```

In [86]:
```
r = 1
kde1 = KernelDensity(kernel='gaussian', bandwidth=0.05)
kde1.fit(dataTrain.loc[dataTrain['Response']==r, 'L1_S24_F1846'].values.reshap
logProbability1 = kde1.score_samples(dataTest.loc[dataTest['Response']==r, 'L1_
logProbability1[np.isinf(logProbability1)] = -100 # замінюємо нескінченність
logLikehood1 = logProbability1.sum()
print(logLikehood1)
```

```
103.98787944779149
```

3

Для знайдених найкращих `bandwidth` обчисліть `p(x|0)` та `p(x|1)` для тестової вибірки.

In [80]:
```
pX0 = np.exp(kde0.score_samples(dataTest['L1_S24_F1846'].values.reshape(-1, 1)
pX1 = np.exp(kde1.score_samples(dataTest['L1_S24_F1846'].values.reshape(-1, 1)
```

За формулою Байєса знайдіть потім `p(0|x)` та `p(1|x)` . Підсортуйте всі об'єкт тестової вибірки за зростанням ймовірності, що передбачено `p(1|x)` , виведіть на екран ймовірності для наступних 10 об'єктів та розрахуйте кількість відбракованих деталей серед останніх 100 об'єктів у відсортованому ряді.

Теорема Байєса:

p(0|x) = (p(x|0) * p(0)) / (p(x|0) * p(0) + p(x|1) * p(1))

p(1|x) = (p(x|1) * p(1)) / (p(x|0) * p(0) + p(x|1) * p(1))

Help:

ind = np.argsort(predictionProb1afterX) - сортування, що повертає індекси елементів

print(predictionProb1afterX[ind[-10:]]) - виведення останніх 10 елемен

In [97]:
```
pX0
```

Out[97]:
```
array([1.86372499, 2.36423376, 2.59101227, ..., 2.64556202, 2.38273243,
       2.68395152])
```

In [96]:
```
total_samples = len(dataTest)
negative_samples = len(dataTest[dataTest['Response'] == 0])

p0 = negative_samples / total_samples
p0
```

Out[96]:
```
0.990425376406958
```

In [95]:
```
total_samples = len(dataTest)
negative_samples = len(dataTest[dataTest['Response'] == 1])

p1 = negative_samples / total_samples
p1
```

Out[95]:
```
0.009574623593041953
```

In [104]:
```
p0X = pX0*p0/(pX0*p0+pX1*p1)
p1X = pX1*p1/(pX1*p1+pX0*p0)
print(f"Ймовірністи p(0|x) = {p0X} \nЙмовірністи p(1|x) = {p1X}")
```

```
Ймовірністи p(0|x) = [0.99370453 0.9946718  0.99470981 ... 0.99472872 0.99107
957 0.99071466]
Ймовірністи p(1|x) = [0.00629547 0.0053282  0.00529019 ... 0.00527128 0.00892
043 0.00928534]
```

4

У лабораторній роботі було розглянуто метод відновлення щільності розподілення випадкової величини на основі її дискретних значень. Для цього було використано гаусонівський метод.

У першій частині лабораторної роботи було розглянуто теоретичні основи методу. Було показано, що гаусонівська функція розподілу є найбільш ймовірним розподілом випадкової величини з обмеженим набором дискретних значень. Це дозволяє використовувати її для побудови щільності розподілення.

У другій частині лабораторної роботи було реалізовано гаусонівський метод в Python. Було проведено експерименти з відновлення щільності розподілення для різних випадкових величин.

За результатами експериментів було встановлено, що гаусонівський метод є ефективним методом відновлення щільності розподілення. Він дозволяє отримати досить точну оцінку щільності розподілення навіть для невеликого набору дискретних значень.

Виконав студент групи ІСТ-21-1, Дешков Максім

In [ ]: