Отчёт по лабораторной работе №4 по фундаментальным концепциям искусственного интеллекта на тему: “ Восстановление функции распределения вероятности”

Выполнил студент группы М80-114СВ-24 Сипкин Владислав.

1. Выполнение реализации метода восстановления плотности вероятности через ЕМ-алгоритм:

а) Этот код включает:

Листинг 1 – Программа реализации метода восстановления плотности вероятности через ЕМ-алгоритма

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# генерация значений
np.random.seed(42)
value = np.hstack([
    np.random.normal(loc=8, scale=0.8, size=100),  # первая функция норального
распределения со сгенерированным набором случайных точек
    np.random.normal(loc=4, scale=0.6, size=100),    # вторая функция норального
распределения со сгенерированным набором случайных точек
    np.random.normal(loc=0, scale=0.7, size=100)     # третья функция норального
распределения со сгенерированным набором случайных точек
])
K = 3  # количество компонент (распределений)
n = len(value)
# начачальные параметры смеси
w = np.ones(K) / K # вес компоненты
mu = np.random.choice(value, K) # мат.ожидание компоненты
sigma = np.random.random(K) # дисперсия компоненты

# ЕМ-алгоритм
N = 100     # количество итераций
for iteration in range(N):
    # На E-шаге вычисляется ожидаемое значение (expectation) вектора скрытых
переменных G
    g = np.zeros((n, K))     # апостериорная вероятность того, что обучающий
объект x_i получен из j-й компоненты смеси
    for k in range(K):
        g[:, k] = w[k] * norm.pdf(value, loc=mu[k], scale=np.sqrt(sigma[k]))
    g /= g.sum(axis=1, keepdims=True)
    # M-шаг: обновление параметров
    for k in range(K):
        w[k] = g[:, k].sum() / n
        mu[k] = (g[:, k] @ value) / g[:, k].sum()
        sigma[k] = (g[:, k] @ ((value - mu[k])**2)) / g[:, k].sum()
```

```
print("Вес", w)
print("Мат.ожидание:", mu)
print("Дисперсия:", sigma)

# визуалиизация восстановления плотности вероятности через EM-алгоритм
x = np.linspace(value.min() - 1, value.max() + 1, 1000)
pdf = np.zeros_like(x)
for k in range(K):
    pdf += w[k] * norm.pdf(x, loc=mu[k], scale=np.sqrt(sigma[k]))

plt.hist(value, bins=30, density=True, alpha=0.6, color="blue",
label="Гистограмма")
plt.plot(x, pdf, label="Функция плотности вероятности нормального распределения",
color="red")
plt.title("Восстановление плотности вероятности через EM-алгоритм")
plt.xlabel("x")
plt.ylabel("Плотность вероятности")
plt.legend()
plt.show()
```

б) Вывод результатов работы программы в консоле:

```
Вес [0.330017   0.33374321 0.33623979]
Мат.ожидание: [0.01887176 7.91327274 3.99917555]
Дисперсия: [0.50508282 0.53207359 0.33817438]
```

Рис. 1. Результаты подборки параметров для восстановления функции плотности вероятности нормального распределения через EM-алгоритм

в) Визуализация графиков:

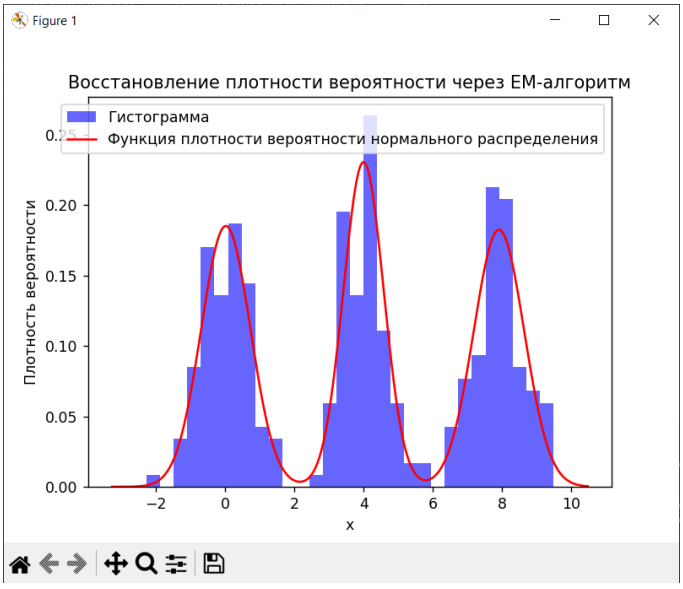

Рис. 2. Визуализация восстановления функции плотности вероятности нормального распределения через EM-алгоритм

2. Выполнение реализации метода восстановления плотности вероятности через ядерное сглаживание:

а) Этот код включает:

Листинг 2 – Программа реализации метода восстановления плотности вероятности через ядерное сглаживание:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde

# генерация данных
np.random.seed(42)
value = np.hstack([
    np.random.normal(loc=8, scale=0.8, size=100),
    np.random.normal(loc=4, scale=0.6, size=100),
    np.random.normal(loc=0, scale=0.7, size=100)
])

# восстановление плотности вероятности методом ядерного сглаживания
KDE = gaussian_kde(value)  # использование встроенной функции оценкия плотности вероятности через ядерное сглаживание
x = np.linspace(value.min() - 1, value.max() + 1, 1000)  # диапазон для построения графика
pdf = KDE(x)  # вычисление значений плотности

# визуализация восстановления плотности вероятности через методо ядерного сглаживания
plt.hist(value, bins=30, density=True, alpha=0.6, color='blue', label="Гистограмма")
plt.plot(x, pdf, label="Функция плотности вероятности нормального распределения", color='orange')
plt.title("Восстановление плотности вероятности методом ядерного сглаживания")
plt.xlabel("x")
plt.ylabel("Плотность вероятности")
plt.legend()
plt.show()
```

б) Визуализация результатов:

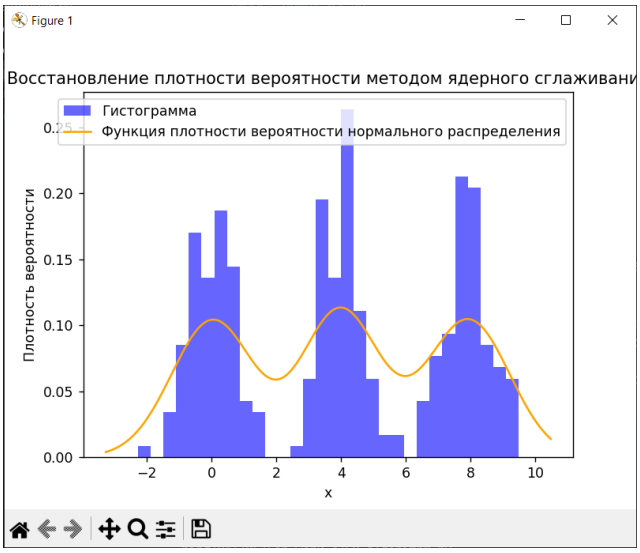

Рис. 3. Визуализация восстановления функции плотности вероятности нормального распределения через ядерное сглаживание

3. Выполнение реализации метода Метрополиса-Гастингса для несимметричного распределения на основе функции плотности, которая была восстановлена EM-алгоритмом, получив изначальные точки

а) Этот код включает:

Листинг 3 – Программа реализации метода Метрополиса-Гастингса для несимметричного распределения на основе функции плотности, которая была восстановлена EM-алгоритмом, получив изначальные точки

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# параметры целевого распределения (полученные из EM-алгоритма)
w = [0.3, 0.4, 0.3]  # веса компонент
mu = [0, 4, 8]       # мат.ожидания компонент
sigma = [0.7, 0.6, 0.8]  # дисперсии компонент

# целевая функция плотности вероятности P(x)
def target_distribution(x):
    pdf = 0
    for k in range(len(w)):
        pdf += w[k] * norm.pdf(x, loc=mu[k], scale=sigma[k])
    return pdf

# вычисление x' с помощью нормального распределения
def proposal_distribution(x, proposal_sigma=1.0):
```

```
    return np.random.normal(loc=x, scale=proposal_sigma)

# алгоритм Метрополиса-Гастингса
def metropolis_hastings(num_samples, proposal_sigma=1.0):
    samples = []         # выборка, которую нужно получиться после выполнения
алгоритма
    x_current = 0        # начальное значение x
    for i in range(num_samples):
        x_proposed = proposal_distribution(x_current, proposal_sigma)       #
генерация нового x' из вспомогательной функции распределения

        p_current = target_distribution(x_current)      # вычисление знаяения
целевой функции при текущем x
        p_proposed = target_distribution(x_proposed)    # вычисление знаяения
целевой функции при x'
        q_last = norm.pdf(x_current, loc=x_proposed, scale=proposal_sigma)      #
вычисление значения вспомогательной функции распределения при x
        q_new = norm.pdf(x_proposed, loc=x_current, scale=proposal_sigma)      #
вычисление значения вспомогательной функции распределения при x'

        alpha = min(1, (p_proposed * q_last) / (p_current * q_new))     #
вычисление вероятности принятия точки x' за новый x
        if np.random.uniform(0, 1) < alpha:     # сравнение сгенерирования числа
и равномерного распределения с вероятностью принятия точки x' за новый x
            x_current = x_proposed      # принятие точки x' за новый x
        samples.append(x_current)       # новый x = x (старое значение)
    return samples

# генерация выборки
num_samples = 10000
samples = metropolis_hastings(num_samples, proposal_sigma=1.0)

# визуалиизация гистограммы выборки и наложение целевой плотности вероятности
x = np.linspace(-2, 12, 1000)
pdf = target_distribution(x)

plt.hist(samples, bins=50, density=True, alpha=0.6, color='blue',
label='Метрополис-Гастингс выборка')
plt.plot(x, pdf, color='red', label='Целевая функция плотности')
plt.title("Метрополис-Гастингс для смеси нормальных распределений")
plt.xlabel("x")
plt.ylabel("Плотность вероятности")
plt.legend()
plt.show()
```

б) Визуализация результатов:

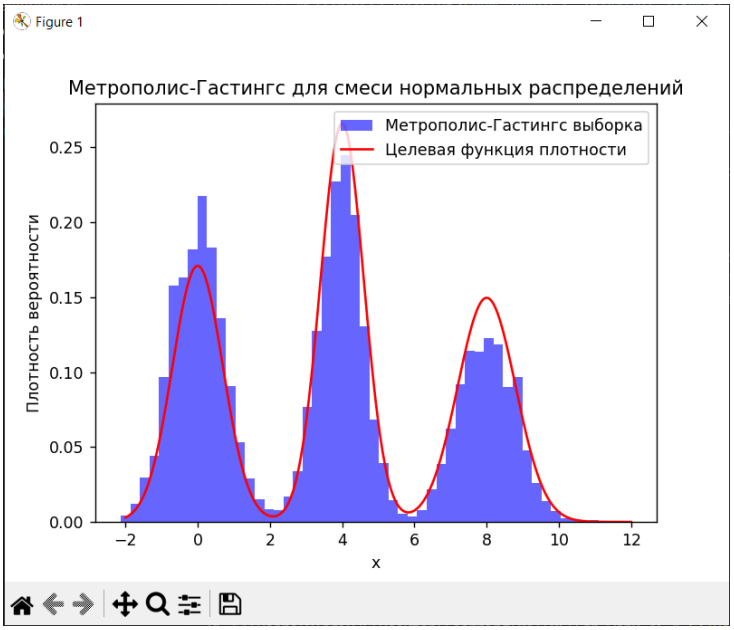

Рис. 4. Визуализация метода Метрополиса-Гастингса для несимметричного распределения на основе функции плотности, которая была восстановлена ЕМ-алгоритмом, получив изначальные точки

4. Выполнение реализации метода Метрополиса-Гастингса для несимметричного распределения на основе функции плотности, которая была восстановлена через ядерное сглаживание, получив изначальные точки

а) Этот код включает:

Листинг 4 – Программа реализации метода Метрополиса-Гастингса для несимметричного распределения на основе функции плотности, которая была восстановлена через ядерное сглаживание, получив изначальные точки

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde, norm

# генерация данных
np.random.seed(42)
value = np.hstack([
    np.random.normal(loc=8, scale=0.8, size=100),
    np.random.normal(loc=4, scale=0.6, size=100),
    np.random.normal(loc=0, scale=0.7, size=100)
])

# восстановление плотности вероятности методом ядерного сглаживания
KDE = gaussian_kde(value)
```

```
x = np.linspace(value.min() - 1, value.max() + 1, 1000)
pdf = KDE(x)

# алгоритм Метрополиса-Гастингса
def metropolis_hastings(kde, num_samples, proposal_sigma=1.0):
    samples = []  # выборка, которую нужно получить
    x_current = np.random.choice(value)  # начальное значение x
    for i in range(num_samples):
        x_proposed = np.random.normal(x_current, proposal_sigma)    # генерация
нового x' из вспомогательной функции распределения

        p_current = kde(x_current)      # вычисление знаяения целевой функции при
текущем
        p_proposed = kde(x_proposed)    # вычисление знаяения целевой функции при
x'
        q_last = norm.pdf(x_current, loc=x_proposed, scale=proposal_sigma)      #
вычисление значения вспомогательной функции распределения при x
        q_new = norm.pdf(x_proposed, loc=x_current, scale=proposal_sigma)      #
вычисление значения вспомогательной функции распределения при x'

        alpha = min(1, (p_proposed * q_last) / (p_current * q_new))     #
вычисление вероятности принятия точки x' за новый x
        if np.random.uniform(0, 1) < alpha:     # сравнение сгенерирования числа
и равномерного распределения с вероятностью принятия точки x' за новый x
            x_current = x_proposed      # принятие точки x' за новый x
        samples.append(x_current)       # новый x = x (старое значение)
    return np.array(samples)

# генерация выборки
num_samples = 10000
proposal_sigma = 1.0
samples = metropolis_hastings(KDE, num_samples, proposal_sigma)

# визуалиизация гистограммы выборки и наложение целевой плотности вероятности
plt.hist(samples, bins=50, density=True, alpha=0.6, color='blue',
label="Метрополис-Гастингс выборка")
plt.plot(x, pdf, color='red', label="Восстановленная плотность")
plt.title("Метрополис-Гастингс для ядерного сглаживания")
plt.xlabel("x")
plt.ylabel("Плотность вероятности")
plt.legend()
plt.show()
```

б) Визуализация результатов:

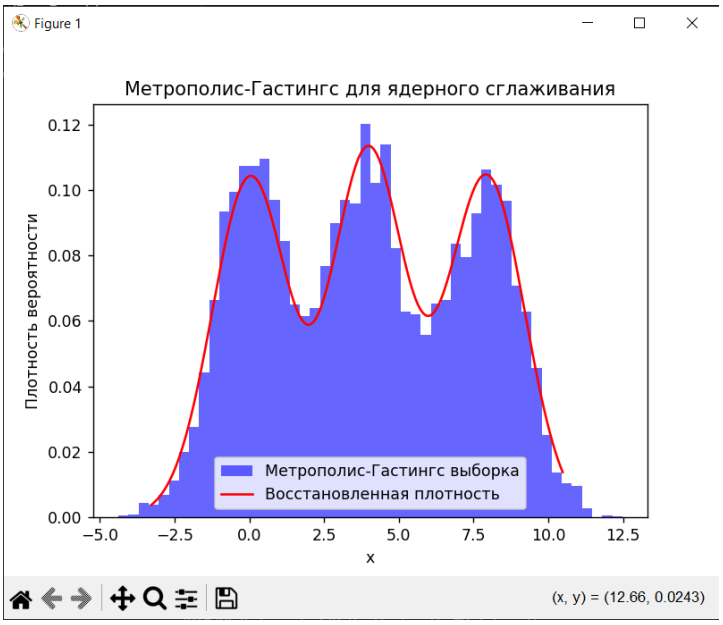

Рис. 5. Визуализация метода Метрополиса-Гастингса для несимметричного распределения на основе функции плотности, которая была восстановлена через ядерное сглаживание, получив изначальные точки

5. Выполнение реализации семплирования по Гиббсу для несимметричного распределения на основе функции плотности, которая была восстановлена через EM-алгоритм, получив изначальные точки

а) Этот код включает:

Листинг 5 – Программа реализации семплирования по Гиббсу для несимметричного распределения на основе функции плотности, которая была восстановлена через EM-алгоритм, получив изначальные точки

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# параметры целевого распределения (полученные из EM-алгоритма)
w = [0.3, 0.4, 0.3]  # веса компонент
mu = [0, 4, 8]       # мат.ожидания компонент
sigma = [0.7, 0.6, 0.8]  # дисперсии компонент
K = len(w)  # количество компонент

# целевая функция плотности вероятности P(x)
def target_pdf(x):
    pdf = 0
    for k in range(K):
        pdf += w[k] * norm.pdf(x, loc=mu[k], scale=np.sqrt(sigma[k]))
```

```
    return pdf

# семплирование по Гиббсу
def gibbs_sampling(target_pdf, n_samples=1000, init_value=0):
    samples = [init_value]  # начальное значение
    for i in range(1, n_samples):
        proposal = np.random.normal(samples[-1], 1.0)  # генерация нового
значения из нормального распределения
        acceptance_ratio = target_pdf(proposal) / target_pdf(samples[-1])  #
вычисление отношения плотности вероятности
        if np.random.uniform(0, 1) < acceptance_ratio:  # принятие или отклонение
нового значения
            samples.append(proposal)  # принятие точки x' за новый x
        else:
            samples.append(samples[-1])  # новый x = x (старое значение)
    return np.array(samples)

# семплирование по Гиббсу
samples = gibbs_sampling(target_pdf, n_samples=1000)

# визуалиизация гистограммы выборки и наложение целевой плотности вероятности
x = np.linspace(min(samples) - 1, max(samples) + 1, 1000)
pdf = np.zeros_like(x)
for k in range(K):
    pdf += w[k] * norm.pdf(x, loc=mu[k], scale=np.sqrt(sigma[k]))

plt.hist(samples, bins=30, density=True, alpha=0.6, color="purple",
label="Сэмплированные данных по Гиббсу")
plt.plot(x, pdf, color="red", label="Восстановленная плотность через EM-
алгоритм")
plt.title("Сэмплирование методом Гиббса из смеси нормальных распределений")
plt.xlabel("x")
plt.ylabel("Плотность вероятности")
plt.legend()
plt.show()
```

б) Визуализация результатов:

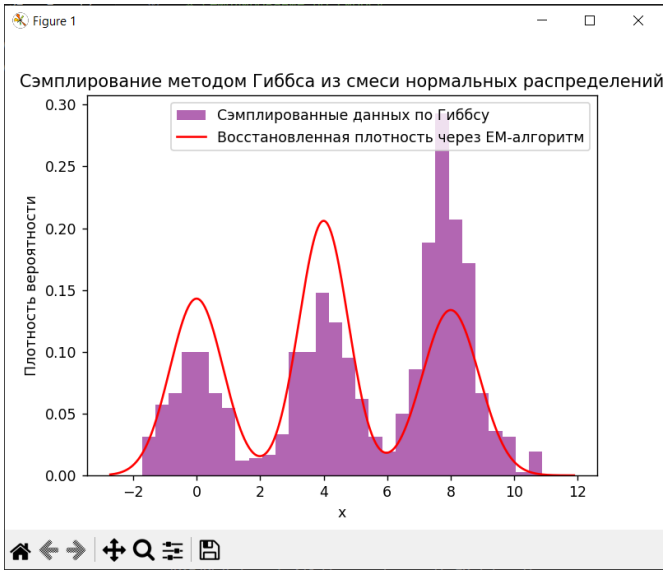

Рис. 6. Визуализация семплирования по Гиббсу для несимметричного распределения на основе функции плотности, которая была восстановлена через ЕМ-алгоритм, получив изначальные точки

6. Выполнение реализации семплирования по Гиббсу для несимметричного распределения на основе функции плотности, которая была восстановлена через ядерное сглаживание, получив изначальные точки

а) Этот код включает:

Листинг 6 – Программа реализации семплирования по Гиббсу для несимметричного распределения на основе функции плотности, которая была восстановлена через ядерное сглаживание, получив изначальные точки

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde

# генерация данных и восстановление плотности через ядерное сглаживание
np.random.seed(42)
value = np.hstack([
    np.random.normal(loc=8, scale=0.8, size=100),
    np.random.normal(loc=4, scale=0.6, size=100),
    np.random.normal(loc=0, scale=0.7, size=100)
])

# восстановление плотности вероятности методом ядерного сглаживания
KDE = gaussian_kde(value)
```

```
x = np.linspace(value.min() - 1, value.max() + 1, 1000)
pdf = KDE(x)

# семплирование методом Гиббса
def gibbs_sampling(kde, n_samples=1000, init_value=0):
    samples = [init_value]  # Начальное значение
    for i in range(1, n_samples):
        # Предложение нового значения из нормального распределения
        proposal = np.random.normal(samples[-1], 1.0)       # генерация нового
значения из нормального распределения
        acceptance_ratio = kde(proposal) / kde(samples[-1])     # вычисление
отношения плотности вероятности (метод Метрополиса-Гастингса)
        if np.random.uniform(0, 1) < acceptance_ratio:      # принятие или
отклонение нового значения на основе выборки с отклонением
            samples.append(proposal)           # принятие точки x' за новый x
        else:
            samples.append(samples[-1])     # новый x = x (старое значение)
    return np.array(samples)

samples = gibbs_sampling(KDE, n_samples=5000, init_value=0)     # семплирование
методом Гиббса

# визуалиизация гистограммы выборки и наложение целевой плотности вероятноси
plt.figure(figsize=(10, 6))
plt.hist(samples, bins=50, density=True, alpha=0.6, color='purple',
label="Сэмплированные данных по Гиббсу")
plt.plot(x, pdf, color='orange', label="Восстановленная плотность через ядерное
сглаживание")
plt.title("Семплирование по Гиббсу из несимметричного распределения")
plt.xlabel("x")
plt.ylabel("Плотность вероятности")
plt.legend()
plt.show()
```

б) Визуализация результатов:

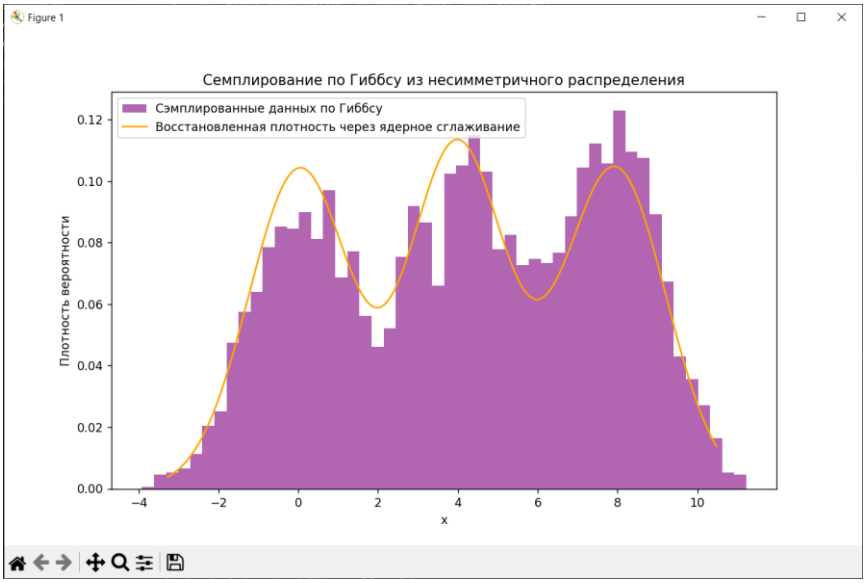

Рис. 7. Визуализация семплирования по Гиббсу для несимметричного распределения на основе функции плотности, которая была восстановлена через ядерное сглаживание, получив изначальные точки