Теория вероятностей и математическая статистика

Задание 1.

Даны значения величины заработной платы заемщиков банка (zp) и значения их поведенческого кредитного скоринга (ks): zp = [35, 45, 190, 200, 40, 70, 54, 150, 120, 110], ks = [401, 574, 874, 919, 459, 739, 653, 902, 746, 832]. Используя математические операции, посчитать коэффициенты линейной регрессии, приняв за X заработную плату (то есть, zp - признак), а за y - значения скорингового балла (то есть, ks - целевая переменная). Произвести расчет как с использованием intercept, так и без.

Подсказка № 1

Ковариация между двумя переменными помогает определить, насколько они изменяются вместе. В Python ковариацию между `zp` и `ks` можно найти через формулу: $Cov(X,Y)=\frac{\sum(X-\bar{X})\cdot(Y-\bar{Y})}{n-1}$, где X и Y — переменные, а $\bar{X}$ и $\bar{Y}$ — их средние значения.

Подсказка № 2

Рассчитать коэффициент наклона без intercept. Для получения коэффициента наклона (b) без intercept используйте формулу: $b=\frac{Cov(X,Y)}{Var(X)}$. Дисперсию (Var) можно найти как $\frac{\sum(X-\bar{X})^2}{n-1}$. В Python это делается через `np.sum` и `np.mean`.

Подсказка № 3

При расчете коэффициентов с intercept, формула для intercept (a) следующая: $a=\bar{Y}-b\cdot\bar{X}$. Это значение добавляет смещение к регрессионной линии, чтобы она проходила через средние значения переменных.

Эталонное решение:

```
import numpy as np

# Данные
zp = np.array([35, 45, 190, 200, 40, 70, 54, 150, 120, 110])
ks = np.array([401, 574, 874, 919, 459, 739, 653, 902, 746, 832])
```

```
# 1.1 Ковариация и коэффициент наклона без intercept
mean_zp = np.mean(zp)
mean_ks = np.mean(ks)
b = np.sum((zp - mean_zp) * (ks - mean_ks)) / np.sum((zp - mean_zp)
** 2)
print(f"Коэффициент наклона (b) без intercept: {b}")

# 1.2 Коэффициенты регрессии с intercept
a = mean_ks - b * mean_zp
print(f"Коэффициент наклона (b) с intercept: {b}")
print(f"Intercept (a): {a}")
```

Задача 2.

Посчитать коэффициент линейной регрессии при заработной плате (zp), используя градиентный спуск (без intercept).

Подсказка № 1

Градиентный спуск — это метод оптимизации, который помогает находить значения параметров, минимизирующие функцию потерь. В этом случае, функция потерь — это среднеквадратичная ошибка. Градиентный спуск обновляет коэффициенты на основе градиента функции потерь относительно этих коэффициентов.

Подсказка № 2

Градиент для линейной регрессии с одной переменной (без intercept) вычисляется как $\frac{1}{n}\sum(y-(b \cdot x)) \cdot x$, где b — текущий коэффициент наклона.

Подсказка № 3

Скорость обучения определяет, насколько большие шаги делаются при обновлении коэффициента наклона. Выберите скорость обучения так, чтобы градиентный спуск сходился, но не прыгал вокруг оптимального значения. Если скорость обучения слишком велика, алгоритм может не сойтись.

Подсказка № 4

Количество итераций должно быть достаточным, чтобы алгоритм успел сойтись. При небольшом числе итераций может не хватить времени для нахождения оптимального решения. Следите за значением MSE и, если необходимо, увеличьте число итераций.

Подсказка № 5

Среднеквадратичная ошибка (MSE) вычисляется как $\frac{1}{n}\sum(y - y\hat{})^2$, где — $y\hat{}$ предсказанные значения. Это значение помогает отслеживать, насколько хорошо модель подгоняет данные на каждом шаге градиентного спуска.

Подсказка № 6

На каждом шаге градиентного спуска важно выводить текущие значения коэффициента и MSE. Это позволит увидеть, как меняется ошибка и как быстро сходится модель.

Эталонное решение:

```
import numpy as np

zp = np.array([35, 45, 190, 200, 40, 70, 54, 150, 120, 110])
ks = np.array([401, 574, 874, 919, 459, 739, 653, 902, 746, 832])

slope = 0
learning_rate = 0.0001
interactions = 100

# # среднеквадратичная ошибка
def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred)**2)

# # градиент
def gradient(zp, ks, slope):
    return np.mean((slope * zp - ks) * zp)

# # градиентный спуск
for i in range (interactions):
    slope -= learning_rate * gradient (zp, ks, slope)
    mse = mean_squared_error (ks, slope * zp)
```

```
    print (f'Итерация {i+1}: Коэффициент склонности = {slope:.2f},
Среднеквадратичная ошибка = {mse:.4f}')

print(f'Коэффициент склонности: {slope:.2f}')
```

Задача 3.

Произвести вычисления как в пункте 2, но с вычислением intercept. Учесть, что изменение коэффициентов должно производиться на каждом шаге одновременно (то есть изменение одного коэффициента не должно влиять на изменение другого во время одной итерации).

Подсказка № 1

Градиентный спуск требует вычисления градиента функции потерь для каждого параметра. Убедитесь, что расчет градиента для коэффициента наклона (`w`) и intercept (`b`) производится отдельно, но обновление этих параметров происходит одновременно в каждом шаге.

Подсказка № 2

Градиент для `w` (наклона) и `b` (intercept) следует вычислять отдельно. Формула для `w` — $\frac{2}{n}\sum(ypred-y)\cdot x$, а для `b` — $\frac{2}{n}\sum(ypred-y)$. Убедитесь, что градиенты правильно рассчитываются и используются для обновления коэффициентов.

Подсказка № 3

После вычисления градиентов для обоих параметров, обновите их одновременно, но не пересчитывайте их до следующего шага. Это важно для правильного выполнения градиентного спуска, чтобы изменения в одном параметре не влияли на изменения в другом на одном шаге.

Подсказка № 4

Скорость обучения (`alfa`) контролирует, насколько быстро коэффициенты обновляются. Небольшое значение скорости обучения может замедлить процесс, а слишком большое значение может привести к расходимости. Подберите значение `alfa` так, чтобы градиентный спуск стабильно сходился.

Подсказка № 5

На каждом шаге полезно вычислять и отслеживать среднеквадратичную ошибку (MSE). Это позволяет вам увидеть, как меняется качество модели и помогает понять, сходится ли алгоритм. Вывод значений MSE периодически, например, каждые 10 итераций, поможет вам следить за прогрессом.

Подсказка № 6

Начальные значения коэффициентов (например, `w` и `b`) могут быть любыми, но их правильная инициализация важна для стабильности обучения. Обычно инициализация производится нулями или малыми случайными числами.

Эталонное решение:

```
import numpy as np


zp = np.array([35, 45, 190, 200, 40, 70, 54, 150, 120, 110])
ks = np.array([401, 574, 874, 919, 459, 739, 653, 902, 746, 832])
# Начальные коэффициенты
w = 0
b = 0

# Скорость обучения
alfa = 0.001

# Количество итераций
n_iter = 100

# Среднеквадратичная ошибка
def compute_mse (y_true, y_pred):
    return np.mean ((y_true - y_pred)**2)

# Градиентный спуск
for i in range (n_iter):
    y_predict = w * zp + b
    errors = y_predict - ks
    dw = (2 / len (zp)) * np.dot(zp, errors)
    db = (2 / len (zp)) * np.sum(errors)
    w -= alfa * dw
    b -= alfa * db
```

```
    if i % 10 == 0:
        mse = compute_mse (ks, y_predict)
        print (f'Interaction {i}: MSE  = {mse}, w = {w}, b = {b}')
print(f'Итоговые значения коэффициента при zp = {w}, b = {b}')
```