3.3 Обучение регрессионных моделей

Переобучение и дисперсия

Минимизация квадратов отклонений $\sum_{i=0}^{n} R_i^2$ ничего не гарантирует. Цель состоит не в минимизации суммы квадратов, а в том чтобы делать правильные предсказания на новых данных.

Переобученные модели очень чувствительны к выбросам, в прогнозах будет очень высокая дисперсия, поэтому к моделям специально дополняется смещение.

Центра́льные преде́льные теоре́мы (Ц.П.Т.) — класс теорем в теории вероятностей, утверждающих, что сумма большого количества независимых случайных величин имеет распределение близкое к нормальному. Так как многие случайные величины в приложениях являются суммами нескольких случайных факторов, центральные предельные теоремы обосновывают популярность нормального распределения.

Смещение модели

Смещение модели означает, что при попытке построить модель предпочтение отдается определенной схеме, например, что наша модель выражается прямой линией или проходит через определенную точку, а не график со сложной структурой минимизирующей остатки

$\sum_{i=0}^{n} R_i^2$
Если мы вносим смещение, мы можем недообучить модель, получатся что задача прогнозирования сводится к балансировке минимизации функции потерь.

Виды регрессии:

- Гребневая регрессия (ridge) добавляется смещение в виде штрафа, из-за этого хуже идет подгонка под имеющиеся данные
- Лассо - регрессия - удаление некоторых переменных

Механически применить линейную регрессию к данным, значит сделать на основе полученной модели прогноз, и думать что все в порядке - нельзя

Градиентный спуск (пакетный градиентный спуск)

Для работы используются все доступные обучающие данные. На практике используется стохастический (вероятностный) спуск, на каждой итерации мы обучаемся только по одной выборке из данных.

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from numpy.linalg import inv, qr
import random


data = np.array(
   [
      [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
      [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
   ]
)

x = data[:, 0]
y = data[:, 1]
n = len(x)

w1 = 0.0
w0 = 0.0

L = 0.001
iteration = 100_000
sample_size = 1 # размер выборки

for i in range(iteration):
```

```
    idx = np.random.choice(n, sample_size, replace=False)
    D_w0 = -2 * sum((y[idx] - (w0 + w1 * x[idx])))
    D_w1 = -2 * sum(x[idx] * (y[idx] - (w0 + w1 * x[idx])))

    w0 -= L * D_w0
    w1 -= L * D_w1

print(w0, w1)
# 0.783474440838093 2.461637212688212

plt.scatter(x, y)

line_y = w1 * x + w0
plt.plot(x, line_y, color='red')
plt.show()
```

- Сокращение числа вычислений
- Вносим смещение ⇒ боремся с переобучением

Мини-пакетный спуск - на каждой итерации используется несколько выборок

Как оценить результат ?

Как оценить на сколько сильно “промахиваются” прогнозы при использовании линейной регрессии

Коэффициент корреляция

Коэффициент корреляции помогает понять есть ли связь между двумя переменными, если большая то связь есть, если ближе к нулю, то нет

```
import pandas as pd
import numpy as np

data = np.array(
    [
        [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
        [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
    ]
)

data_df = pd.DataFrame(data)
print(data_df.corr(method="pearson"))
#          0        1
# 0  1.00000  0.97684
```

```
# 1  0.97684  1.00000

data_df[1] = data_df.values[::-1]
print(data_df.corr(method='pearson'))

#      0    1
# 0  1.0 -1.0
# 1 -1.0  1.0
```

Формула подсчета коэффициента корреляции:

$$r_{xy} = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n\sum x_i^2 - (\sum x_i)^2][n\sum y_i^2 - (\sum y_i)^2]}}$$

Значение коэффициента корреляции rxy всегда находится в диапазоне от **-1 до +1**, где:

- **+1** — идеальная положительная линейная связь.
- **1** — идеальная отрицательная линейная связь.
- **0** — отсутствие линейной связи.

Обучающие и тестовые выборки

Основной метод борьбы с переобучение, заключается что набор данных делится на обучающую и тестовые выборки.

Во всех видах машинного обучения с учителем это встречается.

Обычная пропорция - это 2/3 обучения и 1/3 на тест или (4/5 к 1/5)

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np

data = np.array(
   [
      [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
      [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
   ]
)

data_df = pd.DataFrame(data)

X = data_df.values[:, 0]
Y = data_df.values[:, 1]

X_traint, X_test, Y_traint, Y_test = train_test_split(X, Y, test_size=1/3)
```

```
print(X_traint, Y_traint)
# [10  1  6  2  7  9] [28  5 14  7 17 22]

print(X_test, Y_test)
# [8 3 5 4] [19  7 11 10]
```

Коэффициент детерминации

$$r_{xy}^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

$$0 < r_{xy}^2 < 1$$

чем ближе к 1, тем лучше регрессия работает на тестовых данных

Обозначения:

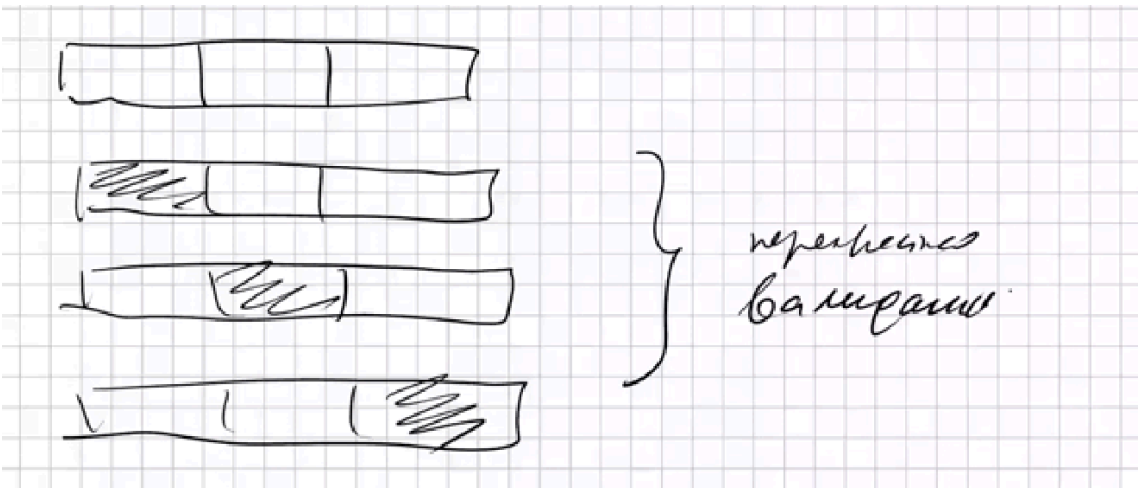
- y_i — реальное значение.
- y^i — значение, предсказанное моделью.
- y^- — среднее значение всех y.

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
data = np.array(
   [
      [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
      [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
   ]
)

data_df = pd.DataFrame(data)

X = data_df.values[:, :-1]
Y = data_df.values[:, -1]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

model = LinearRegression()
```

```
model.fit(X_test, Y_test)

r = model.score(X_test, Y_test)
print(r)
# 0.9500909037593103
```

Перекрестная валидация

Модель обучают трижды и трижды тестируют (трехкратная перекрестная валидация)




```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, KFold, cross_val_score
import matplotlib.pyplot as plt
import pandas as pd

data = np.array(
   [
      [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
      [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
   ]
)

data_df = pd.DataFrame(data)

X = data_df.values[:, :-1]
Y = data_df.values[:, -1]

kfold = KFold(n_splits=3, random_state=1, shuffle=True)
```

```
model = LinearRegression()
results = cross_val_score(model, X, Y, cv=kfold)

print(results) # средник кв. ошибки
# [ 0.88414769 -2.35154626  0.75792214]

print(results.mean(), results.std())
# -0.23649214168138835 1.4964566263570558
```

Метрики показывают насколько единообразно ведет себя модель на разных выборках.

Возможно использование поэлементной перекрестной валидации - когда мало данных.

Если большая дисперсия, то можно делать случайную валидацию

Вариационная выборка - для сравнения различных моделей конфигураций

Многомерная линейная регрессия

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, KFold, cross_val_score
import matplotlib.pyplot as plt
import pandas as pd

data_df = pd.read_csv("multiple_independent_variable_linear.csv")
X = data_df.values[:, :-1]
Y = data_df.values[:, -1]

kfold = KFold(n_splits=3, random_state=1, shuffle=True)

model = LinearRegression().fit(X, Y)

print(model.coef_, model.intercept_)
# [2.00672647 3.00203798] 20.109432820035963

fig = plt.figure()
ax = plt.axes(projection="3d")

x1 = X[:, 0]
x2 = X[:, 1]
y = Y
```

```
ax.scatter3D(x1, x2, y)
plt.show()
```