Untitled Notebook

Anonymous

2025 年 6 月 20 日

任务三：基于 SQL 的多元线性回归

定义如下幸福指数数据表 happyness(Overall_rank, Country, Score, GDP_per_capita, Social_support, Healthy_life_expectancy, Freedom_to_make_life_choices, Generosity, Perceptions_of_corruption)。请以 score 作为因变量，GDP_per_capita, Social_support, Healthy_life_expectancy, Freedom_to_make_life_choices, Generosity, Perceptions_of_corruption 作为自变量，用 SQL 完成多元线性回归算法。

首先写出回归方程:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + u$$

注意：这里的y代表上面的因变量 score，而$x_1, x_2, x_3, x_4, x_5, x_6$分别代表自变量 GDP_per_capita, Social_support, Healthy_life_expectancy, Freedom_to_make_life_choices, Generosity, Perceptions_of_corruption.

我们设输入向量为$\mathbf{x} = (1, x_1, x_2, x_3, x_4, x_5, x_6)$，设系数向量为$\beta = (\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6)$，那么我们的目标是最小化所有样本上的**均方误差**:

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta)^2$$

首先计算出:

$$\frac{\partial J(\beta)}{\partial \beta_j} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta)x_{ij}, j = 0, 1, 2, 3, 4, 5, 6$$

这里认为$x_0 = 1$.

我们使用梯度下降的算法:

$$\beta_j \leftarrow \beta_j - \alpha\frac{\partial J(\beta)}{\partial \beta_j}, j = 0, 1, 2, 3, 4, 5, 6$$

$$\beta_j \leftarrow \beta_j + \frac{2\alpha}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta)x_{ij}, j = 0, 1, 2, 3, 4, 5, 6$$

[1]:
```
import sqlite3
```

```
def print_table_schema(db_path: str, table_name: str):
    """
    连接到指定的 SQLite 数据库，查询并打印出某张表的结构。
    输出内容包括：列 index、列名、数据类型、是否 NOT NULL、默认值、是否主键等。
    """
    # 1. 连接到 SQLite
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
```

```
    # 2. 执行 PRAGMA table_info
    cursor.execute(f"PRAGMA table_info('{table_name}');")
    rows = cursor.fetchall()

    if not rows:
        print(f"表 '{table_name}' 不存在或没有任何字段。")
    else:
        # 3. 打印表头
         header = ["cid", "column_name", "data_type", "not_null", "default_value",
"is_pk"]
        print(f"{header[0]:^3} | {header[1]:^30} | {header[2]:^10} | {header[3]:^8}
| {header[4]:^12} | {header[5]:^5}")
        print("-" * (len(header) * 15))

        # 4. 逐行打印结果
        for cid, name, col_type, notnull, dflt_value, pk in rows:
            print(f"{cid:^3} | {name:^30} | {col_type:^10} |   {notnull:^4}   | "
                  f"{str(dflt_value):^12}  |    {pk:^1}")
    # 5. 关闭连接
    cursor.close()
[2]:    conn.close()
```

```
def print_rows(db_path: str, table_name: str, n: int) -> None:
    """
    连接到指定的 SQLite 数据库，查询并打印出某张表的前 n 条记录，包括列名和每行的值。

    参数:
      - db_path:    SQLite 数据库文件路径
      - table_name: 要查询的表名
      - n:          要打印的行数

    输出示例（假设表有列 id, name, age):
    id | name    | age
    --------------------
    1  | Alice   | 30
    2  | Bob     | 25
    3  | Charlie | 40
    """
    # 1. 连接到 SQLite 数据库
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()

    try:
        # 2. 执行查询，取前 n 条记录
        cursor.execute(f"SELECT * FROM {table_name} LIMIT ?;", (n,))
        rows = cursor.fetchall()

        # 3. 如果没有结果，提示表为空或行数不足
        if not rows:
            print(f"表 '{table_name}' 中没有数据，或 n={n} 大于行数。")
            return

        # 4. 从 cursor.description 获取列名
        col_names = [desc[0] for desc in cursor.description]
        num_cols = len(col_names)
```

```
        # 5. 打印列名行
        print(" | ".join(col_names))
        print("-" * (len(" | ".join(col_names)) + 0))

        widths = []
        for i in range(num_cols):
            max_cell = max(len(str(row[i])) for row in rows)
            widths.append(max_cell)

        # 6. 打印每一行数据
        for row in rows:
            fmt_row = [str(row[i]).ljust(widths[i]) for i in range(num_cols)]
            print(" | ".join(fmt_row))

    except sqlite3.Error as e:
        print(f"查询过程中出现错误: {e}")
    finally:
        # 7. 关闭游标与连接
        cursor.close()
[3]:    conn.close()
```

```
conn = sqlite3.connect('happiness.db')
cursor = conn.cursor()
# 创建 happiness 表
cursor.execute('''
CREATE TABLE IF NOT EXISTS happiness (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    Overall_rank INTEGER NOT NULL,
    Country TEXT NOT NULL,
    Score REAL NOT NULL,
    GDP_per_capita REAL NOT NULL,
    Social_support REAL NOT NULL,
    Healthy_life_expectancy REAL NOT NULL,
    Freedom_to_make_life_choices REAL NOT NULL,
    Generosity REAL NOT NULL,
    Perceptions_of_corruption REAL NOT NULL
);
''')
conn.commit()

[4]: print_table_schema('happiness.db', 'happiness')
```

```
cid |          column_name          | data_type  | not_null | default_value | is_pk
-------------------------------------------------------------------------------------
 0  |               id              |  INTEGER   |    0     |      None     |    1
 1  |          Overall_rank         |  INTEGER   |    1     |      None     |    0
 2  |            Country            |    TEXT    |    1     |      None     |    0
 3  |             Score             |    REAL    |    1     |      None     |    0
 4  |         GDP_per_capita        |    REAL    |    1     |      None     |    0
 5  |         Social_support        |    REAL    |    1     |      None     |    0
 6  |    Healthy_life_expectancy    |    REAL    |    1     |      None     |    0
 7  |  Freedom_to_make_life_choices |    REAL    |    1     |      None     |    0
 8  |           Generosity          |    REAL    |    1     |      None     |    0
 9  |   Perceptions_of_corruption   |    REAL    |    1     |      None     |    0
```

[5]:
```
print_rows('happiness.db', 'happiness', 5)
```

```
id | Overall_rank | Country | Score | GDP_per_capita | Social_support
| Healthy_life_expectancy | Freedom_to_make_life_choices | Generosity |
Perceptions_of_corruption
--------------------------------------------------------------------------------------------------
1 | 1 | Finland     | 7.769 | 1.34  | 1.587 | 0.986 | 0.596 | 0.153 | 0.393
2 | 2 | Denmark     | 7.6   | 1.383 | 1.573 | 0.996 | 0.592 | 0.252 | 0.41
3 | 3 | Norway      | 7.554 | 1.488 | 1.582 | 1.028 | 0.603 | 0.271 | 0.341
4 | 4 | Iceland     | 7.494 | 1.38  | 1.624 | 1.026 | 0.591 | 0.354 | 0.118
5 | 5 | Netherlands | 7.488 | 1.396 | 1.522 | 0.999 | 0.557 | 0.322 | 0.298
```

```
# 插入数据，从当前目录下的 世界幸福指数数据集 读取
# 里面有 2015 - 2019 年的世界幸福指数数据
import csv

# 2019 年数据 结构如下:
# Overall rank,Country or region,Score,GDP per capita,Social support,Healthy life
expectancy,Freedom to make life choices,Generosity,Perceptions of corruption

with open('./世界幸福指数数据集/2019.csv', 'r', encoding='utf-8') as f:
    reader = csv.reader(f)
    header = next(reader)  # 读取表头
    for row in reader:
        cursor.execute('''
            INSERT INTO happiness (Overall_rank, Country, Score, GDP_per_capita,
Social_support,
                                   Healthy_life_expectancy, Freedom_to_make_life_choices,
                                     Generosity, Perceptions_of_corruption)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);
      ''', (row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8]))

conn.commit()
```

[6]:
```
print_rows('happiness.db', 'happiness', 5)
```

```
id | Overall_rank | Country | Score | GDP_per_capita | Social_support
| Healthy_life_expectancy | Freedom_to_make_life_choices | Generosity |
Perceptions_of_corruption
--------------------------------------------------------------------------------------------------
1 | 1 | Finland     | 7.769 | 1.34  | 1.587 | 0.986 | 0.596 | 0.153 | 0.393
2 | 2 | Denmark     | 7.6   | 1.383 | 1.573 | 0.996 | 0.592 | 0.252 | 0.41
3 | 3 | Norway      | 7.554 | 1.488 | 1.582 | 1.028 | 0.603 | 0.271 | 0.341
4 | 4 | Iceland     | 7.494 | 1.38  | 1.624 | 1.026 | 0.591 | 0.354 | 0.118
5 | 5 | Netherlands | 7.488 | 1.396 | 1.522 | 0.999 | 0.557 | 0.322 | 0.298
```

```
def run_gradient_descent(db_path: str, max_iter: int = 1000, alpha: float = 0.0005):
    """
    在 SQLite 数据库中执行梯度下降算法.
    1) 创建辅助表 betas (只含一行) 存放 beta0..beta6, 初始值为 0.
    2) 创建辅助表 constants 存放 alpha 和 n (样本数).
    3) Python 层面循环 max_iter 次, 每次执行一次 UPDATE betas ... 来更新系数.
    4) 最终从 betas 读出 beta0...beta6.
```

```
    """
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    try:
        cursor.execute('DROP TABLE IF EXISTS betas;')
        cursor.execute('DROP TABLE IF EXISTS constants;')
        cursor.execute('DROP TABLE IF EXISTS sample_count;')

        # 计算样本数 n，并存入 sample_count
        cursor.execute('''
        CREATE TABLE sample_count AS
        SELECT COUNT(*) AS n
        FROM happiness;
        ''')

        # 创建 betas 表，只含一行，初始化 7 个 beta 都为 0.0
        cursor.execute('''
        CREATE TABLE betas (
            beta0 REAL DEFAULT 0.0,
            beta1 REAL DEFAULT 0.0,
            beta2 REAL DEFAULT 0.0,
            beta3 REAL DEFAULT 0.0,
            beta4 REAL DEFAULT 0.0,
            beta5 REAL DEFAULT 0.0,
            beta6 REAL DEFAULT 0.0
        );
        ''')
        cursor.execute('INSERT INTO betas DEFAULT VALUES;')

        # 创建 constants 表，保存 alpha 和 n（从 sample_count 中读取）
        cursor.execute(f'''
        CREATE TABLE constants AS
        SELECT
            {alpha} AS alpha,
            n
        FROM sample_count;
        ''')

        conn.commit()

        update_sql = '''
        UPDATE betas
        SET
             beta0 = beta0 + (2.0 * (SELECT alpha FROM constants) / (SELECT n FROM
constants)) * (
                     SELECT SUM(Score - (beta0 + beta1 * GDP_per_capita + beta2 *
Social_support +
                                         beta3 * Healthy_life_expectancy + beta4 *
Freedom_to_make_life_choices +
                                beta5 * Generosity + beta6 * Perceptions_of_corruption))
                FROM happiness
            ),
             beta1 = beta1 + (2.0 * (SELECT alpha FROM constants) / (SELECT n FROM
constants)) * (
                    SELECT SUM((Score - (beta0 + beta1 * GDP_per_capita + beta2 *
Social_support +
```

```
                                        beta3 * Healthy_life_expectancy + beta4 *
Freedom_to_make_life_choices +
                            beta5 * Generosity + beta6 * Perceptions_of_corruption))
* GDP_per_capita)
                FROM happiness
            ),
             beta2 = beta2 + (2.0 * (SELECT alpha FROM constants) / (SELECT n FROM
constants)) * (
                    SELECT SUM((Score - (beta0 + beta1 * GDP_per_capita + beta2 *
Social_support +
                                        beta3 * Healthy_life_expectancy + beta4 *
Freedom_to_make_life_choices +
                            beta5 * Generosity + beta6 * Perceptions_of_corruption))
* Social_support)
                FROM happiness
            ),
             beta3 = beta3 + (2.0 * (SELECT alpha FROM constants) / (SELECT n FROM
constants)) * (
                    SELECT SUM((Score - (beta0 + beta1 * GDP_per_capita + beta2 *
Social_support +
                                        beta3 * Healthy_life_expectancy + beta4 *
Freedom_to_make_life_choices +
                            beta5 * Generosity + beta6 * Perceptions_of_corruption))
* Healthy_life_expectancy)
                FROM happiness
            ),
             beta4 = beta4 + (2.0 * (SELECT alpha FROM constants) / (SELECT n FROM
constants)) * (
                    SELECT SUM((Score - (beta0 + beta1 * GDP_per_capita + beta2 *
Social_support +
                                        beta3 * Healthy_life_expectancy + beta4 *
Freedom_to_make_life_choices +
                            beta5 * Generosity + beta6 * Perceptions_of_corruption))
* Freedom_to_make_life_choices)
                FROM happiness
            ),
             beta5 = beta5 + (2.0 * (SELECT alpha FROM constants) / (SELECT n FROM
constants)) * (
                    SELECT SUM((Score - (beta0 + beta1 * GDP_per_capita + beta2 *
Social_support +
                                        beta3 * Healthy_life_expectancy + beta4 *
Freedom_to_make_life_choices +
                            beta5 * Generosity + beta6 * Perceptions_of_corruption))
* Generosity)
                FROM happiness
            ),
             beta6 = beta6 + (2.0 * (SELECT alpha FROM constants) / (SELECT n FROM
constants)) * (
                    SELECT SUM((Score - (beta0 + beta1 * GDP_per_capita + beta2 *
Social_support +
                                        beta3 * Healthy_life_expectancy + beta4 *
Freedom_to_make_life_choices +
                            beta5 * Generosity + beta6 * Perceptions_of_corruption))
* Perceptions_of_corruption)
                FROM happiness
            );'''
```

```
        for _ in range(max_iter):
            cursor.execute(update_sql)
            conn.commit()

          cursor.execute("SELECT beta0, beta1, beta2, beta3, beta4, beta5, beta6
FROM betas;")
        beta_vals = cursor.fetchone()
        print("训练结束，最终 beta 参数：")
        print(f"beta0 = {beta_vals[0]:.6f}")
        print(f"beta1 = {beta_vals[1]:.6f}")
        print(f"beta2 = {beta_vals[2]:.6f}")
        print(f"beta3 = {beta_vals[3]:.6f}")
        print(f"beta4 = {beta_vals[4]:.6f}")
        print(f"beta5 = {beta_vals[5]:.6f}")
        print(f"beta6 = {beta_vals[6]:.6f}")
    except sqlite3.Error as e:
        print("执行过程中出错：", e)
    finally:
        cursor.close()
[7]:    conn.close()
```

```
[8]: run_gradient_descent('happiness.db', max_iter=1000, alpha=0.0005)
```

```
训练结束，最终 beta 参数：
beta0 = 1.302814
beta1 = 1.199887
beta2 = 1.590536
beta3 = 0.964165
beta4 = 0.537504
beta5 = 0.250273
beta6 = 0.158444
```

```
# 预测一个国家的幸福指数
def predict_happiness(db_path: str, gdp: float, social_support: float, healthy_life:
float,
                      freedom: float, generosity: float, corruption: float) -> float:
    """
    预测一个国家的幸福指数.
    """
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    try:
          cursor.execute("SELECT beta0, beta1, beta2, beta3, beta4, beta5, beta6
FROM betas;")
        beta_vals = cursor.fetchone()
        if not beta_vals:
            raise ValueError("未找到训练好的模型参数.")

        # 使用线性回归模型进行预测
        prediction = (beta_vals[0] +
                      beta_vals[1] * gdp +
                      beta_vals[2] * social_support +
                      beta_vals[3] * healthy_life +
                      beta_vals[4] * freedom +
```

```
                    beta_vals[5] * generosity +
                    beta_vals[6] * corruption)
        return prediction
    except sqlite3.Error as e:
        print("执行过程中出错: ", e)
    finally:
        cursor.close()
[9]:    conn.close()
```

```
# 使用 2019 年 中国 的数据 Score = 5.191
pred = predict_happiness('happiness.db', gdp=1.029, social_support=1.125,
healthy_life=0.893,
                freedom=0.521, generosity=0.058, corruption=0.100)
[10]: print(f"预测的幸福指数为: {pred:.6f}")
```

```
预测的幸福指数为: 5.498249
```

```
# 使用 2019 年 United States 的数据 Score = 6.892
# 1.433,1.457,0.874,0.454,0.280,0.128
pred = predict_happiness('happiness.db', gdp=1.433, social_support=1.457,
healthy_life=0.874,
                freedom=0.454, generosity=0.280, corruption=0.128)
[11]: print(f"预测的幸福指数为: {pred:.6f}")
```

```
预测的幸福指数为: 6.516726
```

```
# France,6.592,1.324,1.472,1.045,0.436,0.111,0.183
pred = predict_happiness('happiness.db', gdp=1.324, social_support=1.472,
healthy_life=1.045,
                freedom=0.436, generosity=0.111, corruption=0.183)
[12]: print(f"预测的幸福指数为: {pred:.6f}")
```

```
预测的幸福指数为: 6.531412
```

```
# Qatar,6.374,1.684,1.313,0.871,0.555,0.220,0.167
pred = predict_happiness('happiness.db', gdp=1.684, social_support=1.313,
healthy_life=0.871,
                freedom=0.555, generosity=0.220, corruption=0.167)
[13]: print(f"预测的幸福指数为: {pred:.6f}")
```

```
预测的幸福指数为: 6.631419
```

```
# South Sudan,2.853,0.306,0.575,0.295,0.010,0.202,0.091
pred = predict_happiness('happiness.db', gdp=0.306, social_support=0.575,
healthy_life=0.295,
                freedom=0.010, generosity=0.202, corruption=0.091)
[14]: print(f"预测的幸福指数为: {pred:.6f}")
```

```
预测的幸福指数为: 2.939314
```