Регрессия с опорными векторами (SVR)

Контекст задачи

Сегодня мы рассмотрим пример Support Vector Regression (SVR). В этом проекте мы будем решать ту же задачу, что и в Проекте 2 (где использовалась полиномиальная регрессия), но теперь применим SVR и сравним результаты.

Датасет: `Position_Salaries.csv`

Содержит 3 столбца:

1. `Position` — должность сотрудника.

2. `Level` — уровень должности (числовой).

3. `Salary` — зарплата.

Цель: Предсказать зарплату для сотрудника на уровне 6.5 (например, для Regional Manager с 2-летним опытом).

Шаг 1: Загрузка данных

Используем библиотеку `pandas` для чтения CSV-файла. Нам нужны только столбцы `Level` (признак) и `Salary` (целевая переменная).

```python
import pandas as pd

# Загрузка данных

dataset = pd.read_csv("Position_Salaries.csv")

# Выбор признака (X) и целевой переменной (y)

X = dataset.iloc[:, 1:2].values  # Все строки, столбец 1 (Level)

y = dataset.iloc[:, 2:].values   # Все строки, столбец 2 (Salary)
```

```

*Почему `1:2`?*

*- `1:2` означает выбор столбца с индексом 1 (`Level`), но в формате 2D-массива (чтобы избежать ошибок в scikit-learn).*

---

**Шаг 2: Масштабирование признаков**

SVR чувствителен к масштабу данных, поэтому применяем стандартизацию (приведение к среднему=0, дисперсии=1).

```python
from sklearn.preprocessing import StandardScaler

# Инициализация scaler для X и y
sc_X = StandardScaler()
sc_y = StandardScaler()

# Масштабирование данных
X_scaled = sc_X.fit_transform(X)
y_scaled = sc_y.fit_transform(y)
```

Зачем масштабировать `y`?

- В SVR целевая переменная также должна быть масштабирована для корректной работы ядра (особенно для RBF).

Шаг 3: Обучение модели SVR

Используем ядро Radial Basis Function (RBF) — оно хорошо работает для нелинейных данных.
```

```python
from sklearn.svm import SVR


# Создание и обучение модели
regressor = SVR(kernel="rbf")
regressor.fit(X_scaled, y_scaled.ravel())  # ravel() преобразует y в 1D-массив
```

Параметры SVR:

- `kernel="rbf"` — нелинейное ядро.

- `C` (по умолчанию=1.0) — штраф за ошибки.

- `epsilon` — допустимая погрешность.

Шаг 4: Визуализация результатов

Строим график предсказаний модели.

```python
import matplotlib.pyplot as plt
plt.scatter(X_scaled, y_scaled, color="red", label="Данные")
plt.plot(X_scaled, regressor.predict(X_scaled), color="blue", label="SVR")
plt.title("Предсказание зарплаты (SVR)")
plt.xlabel("Уровень должности (масштабированный)")
plt.ylabel("Зарплата (масштабированная)")
plt.legend()
plt.show()
```

Интерпретация:

- Красные точки — реальные данные.

- Синяя линия — предсказания SVR.

Шаг 5: Предсказание для уровня 6.5

Чтобы предсказать зарплату для уровня 6.5, нужно:

1. Масштабировать входное значение.

2. Сделать предсказание.

3. Обратно преобразовать результат в исходный масштаб.

```python
import numpy as np

# Значение для предсказания
level = np.array([[6.5]])

# Масштабирование
level_scaled = sc_X.transform(level)

# Предсказание (уже в масштабированном виде)
scaled_pred = regressor.predict(level_scaled)

# Обратное преобразование
predicted_salary =
sc_y.inverse_transform(scaled_pred.reshape(-1, 1))

print(f"Предсказанная зарплата:
${predicted_salary[0][0]:,.0f}")
```

Результат:

```

*Предсказанная зарплата: $170,000*

```

Сравнение с полиномиальной регрессией:

- В Проекте 2 (полиномиальная регрессия) предсказание было $158,000.

- SVR дал более высокую оценку, что может указывать на лучшее улавливание нелинейностей.

Критические замечания

1. **Недостаток данных — всего 10 строк в датасете. SVR требует больше данных для устойчивых предсказаний.**

2. **Выбор ядра — RBF не всегда оптимален. Можно испытать другие ядра (`linear`, `poly`).**

3. **Масштабирование — если забыть масштабировать `y`, предсказания будут некорректными.**

Регрессия с помощью дерева решений (Decision Tree Regression)

Введение в деревья решений

Дерево решений — это популярный алгоритм машинного обучения, используемый как для задач классификации, так и для регрессии. В отличие от линейных моделей (например, полиномиальной регрессии или SVR), дерево решений:

- Нелинейное: Может моделировать сложные зависимости.
- Непрерывное: Предсказывает значения на основе разделения данных на "листья" (конечные узлы).

Что сделать?

1. Используем тот же датасет Position_Salaries.csv, что и в предыдущих проектах.
2. Загрузите данные
3. Обучите модель:

   Используем DecisionTreeRegressor из библиотеки sklearn.tree.

   Критерий разделения: mse (Mean Squared Error) — минимизирует среднеквадратичную ошибку.

   ```
   Python
   ‘’’
   from sklearn.tree import DecisionTreeRegressor
   # Создание и обучение модели
   regressor = DecisionTreeRegressor(criterion="mse", random_state=0)
   regressor.fit(X, y)
   ‘’’
   ```

   Параметры:

   criterion="mse" — критерий оптимизации.

   random_state=0 — для воспроизводимости результатов.
4. Визуализуализируйте результат
5. Сделайте предсказание для уровня 6.5
6. Напиши итог сравнения с другими методами

Регрессия с помощью Random Forest

Что такое Random Forest?

Random Forest (Случайный лес) — это метод ансамблевого обучения, который объединяет множество деревьев решений для получения более точного и стабильного предсказания. В отличие от одного дерева решений, Random Forest строит сотни деревьев (по умолчанию 500) и усредняет их предсказания, что уменьшает переобучение и повышает точность.

Что сделать?

1. Используем тот же датасет Position_Salaries.csv, что и в предыдущих проектах.
2. Загрузите данные
3. Обучите модель:

   Используем RandomForestRegressor из sklearn.ensemble.
   Ключевые параметры:

- *n_estimators — количество деревьев (по умолчанию 100).*
- *random_state — для воспроизводимости результатов.*

```
Python
‘’’
from sklearn.ensemble import RandomForestRegressor
# Создание и обучение модели (10 деревьев)
regressor_10 = RandomForestRegressor(n_estimators=10, random_state=0)
regressor_10.fit(X, y)
‘’’’
```

4. Визуализуализируйте результат
5. Сделайте предсказание для уровня 6.5

6. Напиши итог сравнения с другими методами

Критические замечания

- Переобучение: Random Forest менее склонен к переобучению, чем одно дерево, но требует настройки n_estimators и max_depth.
- Интерпретируемость: Сложнее интерпретировать, чем одно дерево (но можно использовать feature_importances_).
- Вычислительная сложность: Большое число деревьев увеличивает время обучения.

Рекомендации:

- Для мелких датасетов (как здесь) достаточно 100 деревьев.
- Для больших данных можно увеличить n_estimators до 300-500.