**Полиномиальная регрессия**

Цель полиномиальной регрессии и полиномиального преобразования состоит в том, чтобы представить данные не подчиняются линейной зависимости, в виде полиномиальной функции. Такие данные могут иметь нелинейные взаимосвязи между независимыми и зависимыми переменными.

*Существует несколько способов определить, является ли связь между двумя переменными линейной или нелинейной. Один из наиболее распространенных способов - построение графика рассеяния (scatter plot).*

*import matplotlib.pyplot as plt*

*# Создаем данные для графика = [1, 2, 3, 45]*

*y = [10, 20, 15, 25, Создаем график*

*plt.plot(x, y)*

*# Настраиваем оси и заголовок*

*plt.xlabel('X-значения')*

*plt.ylabel('Y-значения')*

*plt.title('График данных')*

*# Отображаем график*

*plt.show()*

*Если точки на графике распределены вокруг прямой линии, то связь между переменными скорее всего является линейной. В случае, если точки образуют нелинейную форму, например, кривую, это указывает на нелинейную связь статистические методы для определения типа связи, а именно коэффициент корреляции. Если коэффициент корреляции близок к 1 или -1, то связь между переменными скорее всего линейна. Если же коэффициент близок к 0, то связь скорее всего нелинейна.*

**Пример:** Рассмотрим набор данных с одной независимой переменной X (например, стаж работы) и одной зависимой переменной Y (например, заработная плата). Если наблюдается, что связь между X и Y не является линейной, можно воспользоваться полиномиальной регрессией.

Определенную степень полинома можно выбрать в зависимости от данных и предполагаемой формы связи между переменными. Например, если есть подозрение, что связь является квадратичной, можно использовать полиномиальное преобразование второй степени (X^2).

Используя полиномиальную регрессию, можно построить модель, которая лучше соответствует данным и позволяет более точно предсказывать значения зависимой переменной Y на основе значения независимой переменной

**Вот полный цикл работы с полиномиальной регрессией:**

1. Импортируйте необходимые библиотеки:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

2. Загрузите данные и разделите их на обучающую и тестовую выборки (если необходимо):

X = # массив признаков

y = # массив целевых переменных

# Разделение на обучающую и тестовую выборки (опционально)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

3. Создайте объект полиномиальных признаков и преобразуйте данные:

degree = 3 # степень полинома

poly\_features = PolynomialFeatures(degree=degree, include\_bias=False)

X\_poly = poly\_features.fit\_transform(X\_train)

4. Обучите модельейной регрессии на полиномиальных признаках:

LinearRegression()

model.fit(X\_poly, y\_train)

5. Прогнруйте значения на тестовой выборке:

X\_test\_poly = poly\_features.transform(X\_test)

y\_pred = model.predict(X\_test\_poly)

6. Вычислите метрики качества модели (например, коэффициент детерминации R^2):

from sklearn.metrics import r2\_score

r2 = r2\_score(y\_test, y\_pred)

print(f" Score: {r2}")

7. Визуализируйте результаты:

plt.scatter(X\_test, y\_test, color='blue', label='Actual')

plt.plot(X\_test, y\_pred, color='red', label='Predicted')

plt.xlabel('X')

plt.ylabel('y')

plt.title('Polynomial Regression')

plt.legend()

plt.show()

Вот подробное объяснение каждого шага:

1. Линейная регрессия: Вы начинаете с модели LinearRegression и обучаете ее на самых коррелирующих параметрах. Это может быть один параметр или несколько параметров, которые имеют сильную корреляцию с зависимой переменной. Обучение модели на одном параметре позволяет оценить его влияние на зависимую переменную. Затем вы можете расширить модель, добавив больше коррелирующих параметров.

2. Полиномиальное преобразование: Если линейная регрессия не дает хорошего соответствия с тестовыми данными, вы можете использовать ту же модель LinearRegression, но применить полиномиальное преобразование к параметрам. Полиномиальное преобразование позволяет модели аппроксимировать нелинейные зависимости между параметрами и зависимой переменной. Вы можете экспериментировать с разными степенями полинома и выбрать оптимальную степень, которая дает наилучший результат.

3. Гребневая регрессия и случайный лес: Если линейная регрессия с полиномиальным преобразованием все еще не дает хорошего соответствия с ценой (высокие значения MSE и низкие значения R2), вы можете попробовать другие модели. Гребневая регрессия (Ridge Regression) является методом регуляризации, который может помочь уменьшить переобучение модели. Вы можете обучить модель на гребневой регрессии и оценить ее производительность.

Если результаты все еще не удовлетворительны, вы можете попробовать применить модель случайного леса (Random Forest Regression). Случайный лес является ансамблевой моделью, которая комбинирует несколько деревьев решений для получения более точных прогнозов. Вы можете доучить объект, обученный на гребневой регрессии, с использованием случайного леса и продолжать экспериментировать с этой моделью, пока не достигнете оптимального результата.

Важно отметить, что порядок и выбор моделей зависят от конкретной задачи и данных. Вам может потребоваться провести дополнительные эксперименты и анализировать результаты, чтобы выбрать наилучшую модель для вашей задачи.

**ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ МОДЕЛИ**

MSE - это метрика, которая измеряет среднеквадратичную ошибку между прогнозируемыми значениями и фактическими значениями в задачах регрессии. Чем меньше значение MSE, тем лучше модель соответствует данным.

Для вычисления MSE, мы сначала находим разницу между прогнозируемыми значениями и фактическими значениями для каждого наблюдения. Затем мы возводим каждую разницу в квадрат, чтобы избежать отрицательных значений. После этого мы находим среднее значение квадратов разностей, и это и будет нашим MSE.

MSE - это средняя квадратичная ошибка, которая измеряет среднеквадратичное отклонение предсказанных значений от фактических значений. Чем меньше значение MSE, тем лучше модель. Однако, абсолютное значение MSE может быть сложно интерпретировать без контекста.

RMSE - это квадратный корень из MSE. Он имеет ту же шкалу, что и исходные данные, что делает его более интерпретируемым. RMSE также измеряет среднеквадратичное отклонение предсказанных значений от фактических значений, но в тех же единицах измерения, что и целевая переменная. Чем меньше значение RMSE, тем лучше модель.

**Вот пример кода на Python, который показывает, как вычислить MSE:**

from sklearn.metrics import mean\_squared\_error

# Пример фактических значений и прогнозируемых значений

actual\_values = [2, 4, 6, 8]

predicted\_values = [1.5, 3.5, 5.5, 7.5]

# Вычисление MSE

mse = mean\_squared\_error(actual\_values, predicted\_values)

print("MSE:", mse)

R2 (Coefficient of Determination) - это метрика, которая измеряет долю дисперсии зависимой переменной, которая может быть объяснена моделью. Значение R2 находится в диапазоне от 0 до 1, где 1 означает, что модель идеально соответствует данным, а 0 означает, что модель не объясняет никакую дисперсию.

Для вычисления R2, мы сначала находим сумму квадратов разностей между фактическими значениями и средним значением фактических значений. Затем мы находим сумму квадратов разностей между прогнозируемыми значениями и средним значением фактических значений. Далее, мы делим вторую сумму на первую сумму и вычитаем результат из 1. Это и будет нашим R2.

**Вот пример кода на Python, который показывает, как вычислить R2:**

from sklearn.metrics import r2\_score

# Пример фактических значений и прогнозируемых значений

actual\_values = [2, 4, 6, 8]

predicted\_values = [1.5, 3.5, 5.5, 7.5]

# Вычисление R2

r2 = r2\_score(actual\_values, predicted\_values)

print("R2:", r2)

Напиши мне простой для понимания механизм работы с данными вместе с кодом для регрессионного анализа каждого вида, оценкой модели, смыслом этого анализа и их визуализацией

**МАТЕМАТИЧЕСКРЕ РЕШЕНИЕ**

Давайте рассмотрим математическую формулу для вычисления MSE и R2.

MSE (Mean Squared Error) вычисляется следующим образом:

MSE = (1/n) \* Σ(yi - ŷi)^2

где:

- n - количество наблюдений

- yi - фактическое значение зависимой переменной для i-го наблюдения

- ŷi - прогнозируемое значение зависимой переменной для i-го наблюдения

- Σ - сумма всех значений для i от 1 до n

Мы вычитаем прогнозируемое значение от фактического значения, возводим разницу в квадрат и суммируем все такие значения для всех наблюдений. Затем мы делим эту сумму на количество наблюдений, чтобы получить среднее значение квадратов разностей.

R2 (Coefficient of Determination) вычисляется следующим образом:

R2 = 1 - (SSR/SST)

где:

- SSR (Sum of Squared Residuals) - сумма квадратов остатков (разницы между фактическими значениями и прогнозируемыми значениями)

- SST (Total Sum of Squares) - общая сумма квадратов (разница между фактическими значениями и средним значением фактических значений)

Мы вычисляем сумму квадратов остатков (SSR), которая представляет собой сумму квадратов разностей между фактическими значениями и прогнозируемыми значениями. Затем мы вычисляем общую сумму квадратов (SST), которая представляет собой сумму квадратов разностей между фактическими значениями и средним значением фактических значений. Затем мы вычитаем отношение SSR к SST из 1, чтобы получить R2.

**КОНВЕЙЕРЫ:**

Если вы создаете объект Pipeline в scikit-learn, в котором включаете шаг нормализации данных (например, с использованием StandardScaler), то вы можете применять различные типы регрессии к этому объекту с помощью метода fit для обучения модели и метода predict для прогнозирования на новых данных.

Когда вы достигаете оптимальных результатов производительности, вы можете использовать этот объект Pipeline для прогнозирования на новых данных, передавая данные в исходном, реальном масштабе. Объект Pipeline автоматически применит нормализацию к новым данным, используя те же параметры, которые были вычислены во время обучения модели.

Например, предположим, что у вас есть объект pipeline, который включает шаг нормализации данных и шаг регрессии:

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

pipeline = Pipeline([

('scaler', StandardScaler()),

('regressor', LinearRegression())

])

Вы можете обучить модель, вызвав метод fit на объекте pipeline:

pipeline.fit(X\_train, y\_train)

Затем вы можете использовать обученную модель для прогнозирования на новых данных, передавая их в исходном масштабе:

predictions = pipeline.predict(X\_test)

Объект Pipeline автоматически применит нормализацию к данным X\_test с использованием тех же параметров, которые были вычислены во время обучения модели. Таким образом, вы получите прогнозы в исходном масштабе данных.

Это удобно, поскольку вам не нужно вручную нормализовывать новые данные перед прогнозированием. Объект Pipeline берет на себя эту задачу и обеспечивает согласованность масштабирования данных между обучением и прогнозированием.

Если вы используете объект Pipeline с шагом нормализации данных (например, StandardScaler), то предсказанная целевая переменная будет выдана в стандартном масштабе, а не нормализованной.

Объект Pipeline применяет нормализацию только к признакам (факторам), а не к целевой переменной. Нормализация выполняется для обеспечения согласованности масштабирования признаков между обучением и прогнозированием, но не влияет на целевую переменную.

Поэтому, когда вы вызываете метод predict на объекте Pipeline, он возвращает предсказанные значения целевой переменной в исходном, стандартном масштабе данных.