Regressão Polinomial e Curvas de Aprendizado

Slides extraídos do livro "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*" de Aurélien Géron

Sumário

Sumário

Regressão Polinomial

E se seus dados forem mais complexos que uma linha reta?

Ideia Principal

Você pode usar um modelo linear para ajustar dados não lineares. Uma maneira simples de fazer isso é adicionar potências de cada feature como novas features e, em seguida, treinar um modelo linear neste conjunto estendido de features. Essa técnica é chamada de **Regressão Polinomial**.

Vamos ver um exemplo. Primeiro, vamos gerar alguns dados não lineares, baseados em uma equação quadrática simples (mais um pouco de ruído).

Exemplo: Dados Quadráticos Não Lineares

```
import numpy as np
np.random.seed(42)
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X ** 2 + X + 2 + np.random.randn(m, 1)
```

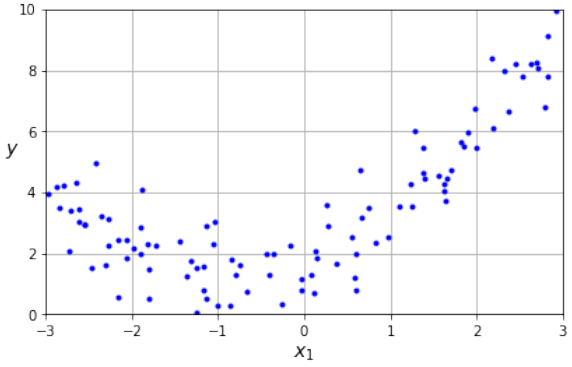


Figure: Dataset não linear e ruidoso gerado.

Usando PolynomialFeatures

Vamos usar a classe PolynomialFeatures do Scikit-Learn para transformar nossos dados de treinamento, adicionando o quadrado (polinômio de segundo grau) de cada feature.

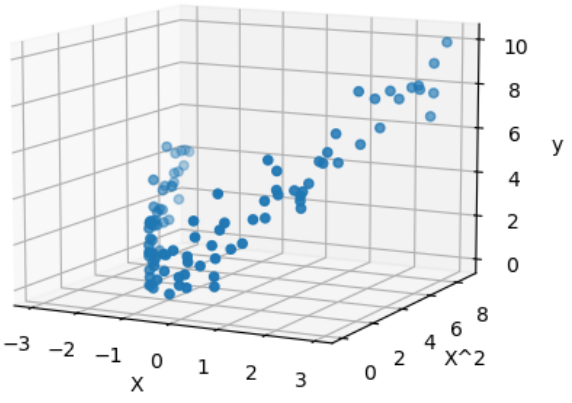
```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

include_bias é False para evitar adicionar uma coluna de 1s (bias), porque o LinearRegression já faz isso.

X_poly agora contém a feature original de X mais o quadrado dessa feature.

```
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929, 0.56664654])
```

Usando PolynomialFeatures



Usando `PolynomialFeatures`

PolynomialFeatures para múltiplos atributos

Neste caso, temos os atributos originais (ex.: x_1 e x_2), suas potências (ex.: x_1^2, x_2^2) e todos os produtos possíveis (ex.: $x_1 x_2$). Os produtos fornecem relações entre os atributos (interações).

Número de atributos $\binom{n+d}{d}$, onde d é o grau e n é o número de atributos originais.
Esta quantidade cresce rapidamente com d e n. Ex.: $n = 10$ e $d = 4$ resulta em 1001 atributos!

Demonstração (opcional)

- Os termos são $x_1^{p_1} x_2^{p_2} \cdots x_n^{p_n}$, onde $p_1 + p_2 + \cdots + p_n \leq d$.
- É o mesmo que $p_1 + p_2 + \cdots + p_n + p_{n+1} = d$, onde p_{n+1} é uma variável de folga não negativa.
- Método das estrelas e barras: d estrelas e n barras, total de $n + d$ símbolos.
  Escolher d posições para as estrelas (ou n para as barras) dá $\binom{n+d}{d}$ combinações.

Ajustando o Modelo Linear aos Dados Transformados

Agora podemos ajustar um modelo `LinearRegression` a esses dados de treinamento estendidos.

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```

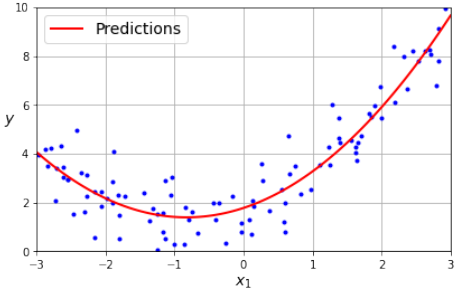


Figure: Previsões do modelo de Regressão Polinomial.

O modelo estima $\hat{y} = 0.56x_1^2 + 0.93x_1 + 1.78$ quando a função original era $y = 0.5x_1^2 + 1.0x_1 + 2.0 +$ ruído.

Regressão Polinomial de Alto Grau

Se você aplicar um modelo polinomial de grau muito alto, ele provavelmente se ajustará aos dados de treinamento muito melhor.

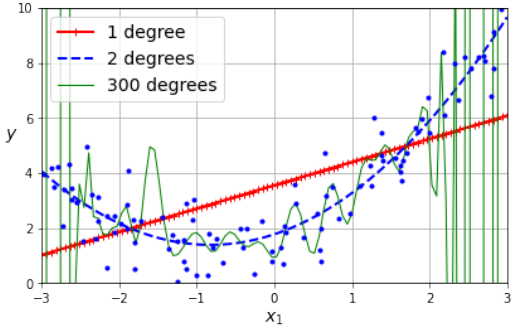


Figure: Regressão Polinomial de Alto Grau.

- Polinômio de grau 300 se contorce para chegar o mais perto possível das instâncias de treinamento.
- Este modelo está severamente com **overfitting** (sobreajuste) nos dados de treinamento.
- O modelo linear está com **underfitting** (subajuste).
- O modelo quadrático é o que melhor generaliza neste caso.

Sumário

Curvas de Aprendizado

Como podemos decidir a complexidade do nosso modelo?
Como saber se o modelo está com overfitting ou underfitting?

- Usar **validação cruzada** para estimar o desempenho de generalização do modelo.
  - Se o modelo tem bom desempenho nos dados de treinamento, mas generaliza mal, ele está com overfitting.
  - Se tem mau desempenho em ambos, está com underfitting.
- Outra maneira é observar as **curvas de aprendizado**.
  - São gráficos do **erro de treinamento e do erro de validação** do modelo **em função da iteração** do treinamento.
  - Para gerar os gráficos, **treine** o modelo várias vezes **em subconjuntos cada vez maiores do conjunto de treinamento.**

Exemplo de Curvas de Aprendizado: Modelo Subajustado

Vamos observar as curvas de aprendizado do modelo de Regressão Linear simples.

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, valid_scores = learning_curve(
    LinearRegression(), X, y, train_sizes=np.linspace(0.01, 1.0, 40),
    cv=5, scoring="neg_root_mean_squared_error")

train_errors = -train_scores.mean(axis=1)
valid_errors = -valid_scores.mean(axis=1)
```

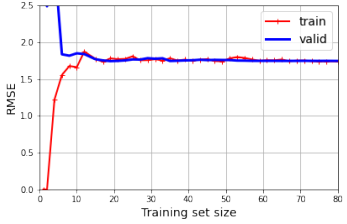


Figure: Curvas de aprendizado para um modelo linear.

Estas curvas são típicas de um modelo que está com **underfitting**.

Analisando Curvas de Subajuste

Analisando o gráfico anterior:

- **Erro de Treinamento**: Começa em zero e sobe até atingir um platô. Quando atinge o platô, adicionar novas instâncias não melhora o erro médio.
- **Erro de Validação**: Começa muito alto e depois diminui até atingir um platô, muito próximo da outra curva.

Características de Underfitting

Ambas as curvas atingiram um platô; elas estão próximas e o erro é razoavelmente alto.

Dica

Se o seu modelo está com underfitting nos dados de treinamento, adicionar mais exemplos de treinamento não ajudará. Você precisa usar um modelo melhor ou criar features melhores.

Exemplo de Curvas de Aprendizado: Modelo Sobreajustado

Agora, vamos ver as curvas de aprendizado de um modelo polinomial de 10º grau nos mesmos dados.

```
from sklearn.pipeline import make_pipeline

polynomial_regression = make_pipeline(
    PolynomialFeatures(degree=10, include_bias=False),
    LinearRegression())
```

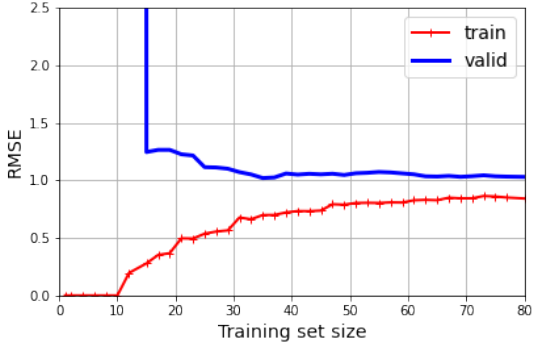


Figure: Curvas de aprendizado para o modelo polinomial de 10º grau.

Analisando Curvas de Sobreajuste

Analisando o gráfico anterior, existem duas diferenças muito importantes:

- O erro nos dados de treinamento é muito menor do que antes.
- Existe uma lacuna (gap) entre as curvas. Isso significa que o modelo tem um desempenho significativamente melhor nos dados de treinamento do que nos dados de validação, o que é a marca registrada de um modelo com **overfitting**.

Dica

Uma maneira de melhorar um modelo com overfitting é alimentá-lo com mais dados de treinamento até que o erro de validação atinja o erro de treinamento.

O Trade-off Viés/Variância

O erro de generalização de um modelo pode ser expresso como a soma de três erros:

Viés (Bias)

Esta parte do erro de generalização se deve a suposições erradas, como assumir que os dados são lineares quando na verdade são quadráticos. Um modelo com alto viés tem maior probabilidade de subajustar os dados de treino.

Variância (Variance)

Esta parte se deve à sensibilidade excessiva do modelo a pequenas variações nos dados de treinamento.
Um modelo com muitos graus de liberdade (como um modelo polinomial de alto grau) provavelmente terá alta variância e, portanto, sobreajustará os dados de treinamento.

Erro Irredutível (Irreducible Error)

Esta parte se deve ao ruído dos próprios dados. A única maneira de reduzir esta parte do erro é limpar os dados.

Aumentar a complexidade de um modelo normalmente aumentará sua variância e reduzirá seu viés. Reduzir a complexidade de um modelo aumenta seu viés e reduz sua variância. É por isso que é chamado de trade-off.

Fim