多项式回归与模型泛化

多项式回归

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x = np.random.uniform(-3, 3, size=100)
X = x.reshape(-1, 1)
y = 0.5 * x**2 + x + 2 + np.random.normal(0, 1, size=100)
plt.scatter(x, y)
X2 = np.hstack([X, X**2])
lin_reg = LinearRegression()
lin_reg.fit(X2, y)
y_predict = lin_reg.predict(X2)
plt.plot(np.sort(x),y_predict[np.argsort(x)],color='r')
plt.show()
print(lin_reg.coef_)
print(lin_reg.intercept_)
```

sklearn中的多项式回归

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x = np.random.uniform(-3, 3, size=100)
X = x.reshape(-1, 1)
y = 0.5 * x**2 + x + 2 + np.random.normal(0, 1, size=100)
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
poly.fit(X)
X2 = poly.transform(X)
print(X2.shape)
lin_reg = LinearRegression()
lin_reg.fit(X2, y)
y_predict = lin_reg.predict(X2)
plt.scatter(x, y)
plt.plot(np.sort(x), y_predict[np.argsort(x)], color='r')
plt.show()
```

PolynomialFeatures(degree=3)

$$x_1, x_2 \begin{cases} 1, x_1, x_2 \\ x_1^2, x_2^2, x_1x_2 \\ x_1^3, x_2^3, x_1^2x_2, x_1x_2^2 \end{cases}$$

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
X = np.arange(1, 11).reshape(-1, 2)
poly = PolynomialFeatures(degree=2)
poly.fit(X)
X2 = poly.transform(X)
print(X2)
```

Pipeline

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.random.uniform(-3, 3, size=100)
X = x.reshape(-1, 1)
y = 0.5 * x**2 + x + 2 + np.random.normal(0, 1, size=100)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
poly_reg = Pipeline([
    ("poly", PolynomialFeatures(degree=2)),
    ("std_scaler", StandardScaler()),
    ("lin_reg", LinearRegression())
])
poly_reg.fit(X, y)
y_predict = poly_reg.predict(X)
```

误差

```
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y, y_predict))
```

欠拟合与过拟合

欠拟合：算法所训练的模型不能完整表述数据关系

过拟合：算法训练的模型过多地表达了数据间的噪音关系

学习曲线

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression
x = np.random.uniform(-3, 3, size=100)
X = x.reshape(-1, 1)
y = 0.5 * x**2 + x + 2 + np.random.normal(0, 1, size=100)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
poly_reg = Pipeline([
    ("poly", PolynomialFeatures(degree=2)),
    ("std_scaler", StandardScaler()),
    ("lin_reg", LinearRegression())
])
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=666)
train_score = []
test_score = []
for i in range(1, 76):
    lin_reg = poly_reg
    lin_reg.fit(X_train[:i],y_train[:i])
    y_train_predict = lin_reg.predict(X_train[:i])
    train_score.append(mean_squared_error(y_train[:i], y_train_predict))
    y_test_predict = lin_reg.predict(X_test)
    test_score.append(mean_squared_error(y_test, y_test_predict))

plt.plot([i for i in range(1, 76)], np.sqrt(train_score), label="train")
plt.plot([i for i in range(1, 76)], np.sqrt(test_score), label="test")
plt.legend()
plt.show()
```

验证数据集与交叉验证

1. 数据划分

训练数据

验证数据

测试数据：不参与模型创建，作为衡量最终模型性能的数据集

2. 交叉验证

训练数据化为A、B、C，BC训练A验证，AC训练B验证，AB训练C验证，k个模型均值为结果调参

```python
import numpy as np
from sklearn import datasets
digits = datasets.load_digits()
X = digits.data
y = digits.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=666)
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier()
print(cross_val_score(knn_clf, X_train, y_train, cv=5))
```

k-folds交叉验证：把训练数据集分成k份，每次训练k个模型，相当于整体性能慢了k倍

留一法LOO-CV：将训练数据集分成m份，Leave-One-Out Cross Validation，完全不受随机的影响，最接近模型真正的性能指标，但计算量巨大

偏差方差平衡

1. 特点

偏差（Bias）：对问题本身假设不正确，欠拟合，如对非线性数据使用线性回归

方差（Variance）：数据的一点点扰动都会较大地影响模型，通常原因在于使用的模型太复杂，过拟合，如高阶多项式回归

2. 相关算法

天生高方差算法：KNN，非参数学习通常都是高方差算法，因为不对数据进行任何假设

天生高偏差算法：线性回归，参数学习通常都是高偏差的算法，因为对数据具有极强的假设

3. 调整偏差和方差

KNN中对k的调整：k越小，模型越复杂，偏差越小；k越大，模型越简单，方差越小。

线性回归中使用多项式回归，degree越小，模型越简单，偏差越大；degree越大，模型越复杂，方差越大。

4. 解决高方差
    - 降低模型复杂度
    - 减少数据维度，降噪
    - 增加样本数
    - 使用验证集
    - 模型正则化

模型正则化

限制参数的大小

岭回归：目标，使$J(\theta) = MSE(y, \hat{y}; \theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$尽可能小

```
from sklearn.linear_model import Ridge
def RidgeRegression(degree, alpha):
    return Pipeline([
        ("poly", PolynomialFeatures(degree=degree)),
        ("std_scaler", StandardScaler()),
        ("ridge_reg", Ridge(alpha=alpha))
    ])
ridgel_reg = RidgeRegression(20, 0.001)
```

LASSO回归：目标，使$J(\theta) = MSE(y, \hat{y}; \theta) + \alpha \sum_{i=1}^{n} |\theta_i|$尽可能小

```
from sklearn.linear_model import Lasso
def RidgeRegression(degree, alpha):
    return Pipeline([
        ("poly", PolynomialFeatures(degree=degree)),
        ("std_scaler", StandardScaler()),
        ("ridge_reg", Lasso(alpha=alpha))
    ])
ridgel_reg = RidgeRegression(20, 0.01)
```

LASSO趋向于使一部分theta值变为0，可作为特征选择用

弹性网

$J(\theta) = MSE(y, \hat{y}; \theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2$