

多项式曲线拟合

《模式识别与机器学习》 *page* : 10

使用 $\sin(x)$ 函数产生数据集：

```
1. import numpy as np
2.
3. n_dots = 200 # 产生 200 个数据点
4. X = np.linspace(-2 * np.pi, 2 * np.pi, n_dots) # X 取值范围：[-2*pi, 2*pi]
5. Y = np.sin(X) + 0.2 * np.random.rand(n_dots) - 0.1 # Y 加上随机噪声 [-0.1, 0.1]
```

使用 `sklearn.preprocessing.PolynomialFeatures` 进行特征的构造：

```
1. from sklearn.preprocessing import PolynomialFeatures
2. import numpy as np
3.
4. n_dots = 200
5. X = np.linspace(-2 * np.pi, 2 * np.pi, n_dots)
6. Y = np.sin(X) + 0.2 * np.random.rand(n_dots) - 0.1
```

```
1. X = X.reshape(-1, 1)
2. Y = Y.reshape(-1, 1)
```

在标准线性回归的情况下，你可能会有一类似于二维数据的模型：

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2$$

如果我们想把数据拟合成抛物面而不是平面，可以结合二阶多项式的特征，使模型看起来像这样：

$$\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

观察，这仍然还是一个线性模型，创造一个新的变量：

$$z = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]$$

使用这些数据的重新标记，原问题可以写成：

$$\hat{y}(w, x) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5$$

以上模型是线性模型，可以使用 `sklearn.linear_model` 进行求解。

`PolynomialFeatures` 通过构造系数来扩展一个线性回归，其有 3 个参数：

- `degree`: 控制多项式的度；
- `interaction_only`: 默认为 `False`，如果指定为 `True`：不会出现特征与自身结合的项，即没有平方项：`x2`；
- `include_bias`: 默认为 `True`，如果指定为 `True`，就会出现 1 那一项。

```
1.  from sklearn.linear_model import LinearRegression
2.  from sklearn.preprocessing import PolynomialFeatures
3.  from sklearn.pipeline import Pipeline
4.
5.  def polynomial_model(degree=1):
6.      polynomial_features = PolynomialFeatures(degree=degree, include_bia
s=False)
7.      linear_regression = LinearRegression(normalize=True)
8.      pipeline = Pipeline([("polynomial_features",
polynomial_features), #添加多项式特征
9.                           ("linear_regression", linear_regression)])
10.     return pipeline
```

```
1.  from sklearn.metrics import mean_squared_error
2.
3.  degrees = [2, 3, 5, 10]
4.  results = []
5.
6.  for d in degrees:
7.      model = polynomial_model(degree=d)
8.      model.fit(X, Y)
9.      train_score = model.score(X, Y)
10.     mse = mean_squared_error(Y, model.predict(X))
11.     results.append({"model": model, "degree": d, "score": train_score,
"mse": mse})
12.
13.  for r in results:
```

```
14.     print("degree: {}; train score: {}; mean squared error: {}".format  
        (r["degree"], r["score"], r["mse"]))
```

```
degree: 2; train score: 0.14623503228246737; mean squared error: 0.42716432  
74566476
```

```
degree: 3; train score: 0.27454911350351763; mean squared error: 0.36296492  
79959968
```

```
degree: 5; train score: 0.8963163750167511; mean squared error: 0.051876040  
37284656
```

```
degree: 10; train score: 0.9934426628788955; mean squared error: 0.00328083  
3258749786
```

比较不同二项式阶数的拟合效果：

```
1.     import matplotlib.pyplot as plt  
2.     from matplotlib.figure import SubplotParams  
3.  
4.     plt.figure(figsize=(12, 6), dpi=200, subplotpars=SubplotParams(hspace=0  
        .3))  
5.     for i, r in enumerate(results):  
6.         fig = plt.subplot(2, 2, i+1)  
7.         plt.xlim(-8, 8)  
8.         plt.title("LinearRegression degree={}".format(r["degree"]))  
9.         plt.scatter(X, Y, s=5, c='b', alpha=0.5)  
10.        plt.plot(X, r["model"].predict(X), 'r-')
```

