2章 教師あり学習（P27）

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

2.1 クラス分類と回帰（P27）

- 教師あり：クラス分類、回帰分類
- クラス分類
  - 2クラス分類
  - 多クラス分類：「教師」情報として、クラス名「猫 Cat」「犬 Dog」「人 Human」のように教師データを与える
- 回帰分類
  - 1つ1つの値
- 教師なし
  - 教師「情」情報なし学習
  - 入力データのみ与えられる
  - 2クラス分類問題や回帰問題

2.2 汎化、過剰適合、過小学習（P28）

- 本ページの機械学習のゴールは、**過剰適合（過学習）**
- 過剰適合（過学習）：トレーニングセットに対する精度が高くなってしまう
  - 過学習で汎化性能（未知データに対する精度）が低くなる
  - 過学習は、モデルが複雑すぎる場合に発生する
- 逆に、汎化性能が高くなるように調整することを「汎化」という（英語 Generalization）
- 汎化性能を高めるためにモデルを単純にすることを「正則化」という（英語 Regularization）
- 逆に汎化性能を高めるためにモデルを複雑にすることを「過小学習」という（英語 Underfitting）
- 過小学習は、適切な学習が行われていない状態のことを指す

2.2.1 モデルの複雑さとデータセットの大きさ（P31）

- 複雑なモデルほど、より柔軟にデータを表現し、適切な精度を出すことができるが、逆に過学習になりやすい
- データ量が大事

2.3 教師あり機械学習アルゴリズム（P31）

- このあとのページからは、このセクションに沿ってクラス分類、回帰分類を説明していきます。
- このあと説明するアルゴリズムの説明の順序はアルゴリズムの複雑さの順番になっています。
- **k近傍法（k-NN）は比較的簡単なアルゴリズムです。**
- 線形モデルは、クラス分類問題、回帰分類問題のどちらにも使えるアルゴリズムです。
- scikit-learnのデフォルトでは、クラス分類問題のほうが多くの問題を解くように調整されている。

2.3.1 サンプルデータセット（P31）

- 2クラス分類問題の例として使われます。

```
%ls -l mglearn/data
```

```
total 0
drwxr-xr-x 1 root root 4096 Oct 16 14:30 .
drwxr-xr-x 1 root root 4096 Oct 16 14:30 ..
```

```
%ls -l /usr/local/lib/python3.6/dist-packages/mglearn/data
```

```
total 0
drwxr-xr-x 1 root root 4096 Oct 16 14:30 .
drwxr-xr-x 1 root root 4096 Oct 16 14:30 ..
```

```
!pip install mglearn
```

```
Collecting mglearn
  Downloading https://files.pythonhosted.org/packages/ed/a1/b38d963f2a2d5f0e8ca8f0c9f3a2ee0d7d2e03293d8f3d0f0fb7ef59e9b0/mglearn-0.1.7.tar.gz (295kB)
     |100%| 296kB 2.8MB/s
Building wheel for mglearn (setup.py) ...
  Stored in directory: /root/.cache/pip/wheels/ad/1d/0d/dabb1a7ca68f6ed2d8a1085e8bb0d97fb8d26a6113e5d2f7ed
Successfully built mglearn
Installing collected packages: mglearn
Successfully installed mglearn-0.1.7
```

```
%ls -l /usr/local/lib/python3.6/dist-packages/mglearn/data
```

```
total 0
drwxr-xr-x 1 root root 4096 Oct 16 14:30 .
drwxr-xr-x 1 root root 4096 Oct 16 14:30 ..
```

```
import mglearn
mglearn.data_sets.load_extended_boston()
```

```
(array([[ 0.00632, 18.00000,  2.31000,  0.00000,  0.53800,  6.57500,
         65.20800,  4.09000,  1.00000, 296.00000, 15.30000, 396.90000,
          4.98000],
        [ 0.02731,  0.00000,  7.07000,  0.00000,  0.46900,  6.42100,
         78.90000,  4.96710,  2.00000, 242.00000, 17.80000, 396.90000,
          9.14000],
        [ 0.02729,  0.00000,  7.07000,  0.00000,  0.46900,  7.18500,
         61.10000,  4.96710,  2.00000, 242.00000, 17.80000, 392.83000,
          4.03000],
        [ 0.06905,  0.00000,  2.18000,  0.00000,  0.45800,  6.87500,
         45.80000,  6.06220,  3.00000, 222.00000, 18.70000, 394.12000,
          5.21000],
        [ 0.02985,  0.00000,  2.18000,  0.00000,  0.45800,  6.43000,
         58.70000,  6.06220,  3.00000, 222.00000, 18.70000, 394.63000,
          2.94000],
        [ 0.08829, 12.00000,  7.87000,  0.00000,  0.52400,  6.01200,
         66.60000,  5.56070,  5.00000, 311.00000, 15.20000, 395.60000,
         12.43000],
        [ 0.14455, 12.00000,  7.87000,  0.00000,  0.52400,  6.172...
```

- 回帰問題（regression）データセットとして、mglearnのデータセットを使用する。

```
%ls -l mglearn/data/boston_extended.gz
```

```
ls: cannot access 'mglearn/data/boston_extended.gz': No such file or directory
```

```
%ls -l /usr/local/lib/python3.6/dist-packages/mglearn/data/boston_extended.gz
```

```
-rw-r--r-- 1 root root 1.7M Oct 16 14:30 /usr/local/lib/python3.6/dist-packages/mglearn/data/boston_extended.gz
```

```
import mglearn
X, y = mglearn.data_sets.load_extended_boston()
print(X.shape, y.shape)
print(type(X), type(y))
print(X[0], y[0])
```

```
(506, 104) (506,)
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
[0.00632 18.      2.31      0.      0.538     6.575    65.2     4.09
 1.      296.     15.3     396.9     4.98      0.      0.      0.      0.
 0.      0.      0.      0.      0.      0.      0.      0.      0.
 0.      0.      0.      0.      0.      0.      0.      0.      0.
 0.      0.      0.      0.      0.
```

In [37]:

```
print('Test set R^2: {:.2f}'.format(reg.score(X_test, y_test)))
# 比較的よい
```

```
Test set R^2: 0.83
```

2.3.2.4 KNeighborsRegressorの解析（P44）

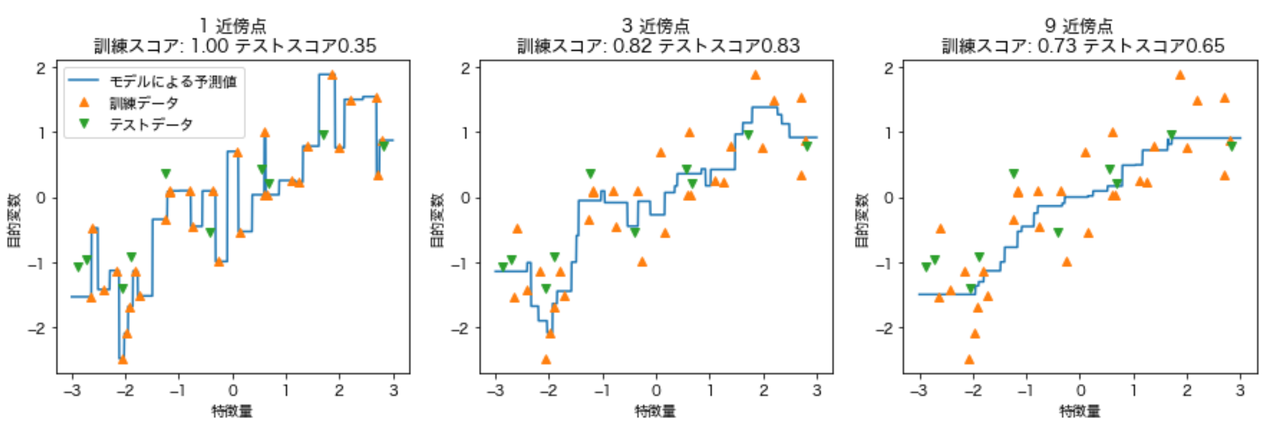
- 1次元のデータセットに対して、すべての値に対する予測値がどのようになるかを見ていく（？）
- これにはたくさんのデータポイントを持つデータセットをつくればよい

In [95]:

```
# 日本語フォントの設定
font_options = {'family': 'Hiragino Sans'}
plt.rc('font', **font_options)

# プロット先を3つ作る
fig, axes = plt.subplots(1, 3, figsize = (15, 4))

# -3〜3までの間にデータポイントを1000点作る
# reshapeの-1は自動の意（行：自動、列：1、の意）
line = np.linspace(-3, 3, 1000).reshape(-1,1)

for n_neighbors, ax in zip([1, 3, 9], axes):
    # モデルを構築
    reg = KNeighborsRegressor(n_neighbors = n_neighbors)
    # 訓練データセットをプロットし学習させる
    reg.fit(X_train, y_train)

    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^')
    ax.plot(X_test, y_test, 'v')
    ax.set_title(
    "{} 近傍点\n 訓練スコア: {:.2f} テストスコア{:.2f}".format(
    n_neighbors, reg.score(X_train, y_train), reg.score(X_test, y_test)))
    ax.set_xlabel("特徴量")
    ax.set_ylabel("目的変数")

axes[0].legend(["モデルによる予測値", "訓練データ", "テストデータ"], loc="best")
```

Out[95]:

```
<matplotlib.legend.Legend at 0x7f98722f3790>
```



- このグラフからわかるように 1 最近傍点による回帰では予測はすべて訓練データの点を通っている
- このため、予測は不安定になる。逆に最近傍点を増やすと予測はスムースだが訓練データに対する適合度は下がる

2.3.2.5 利点と欠点とパラメータ（P45）

- **KNeighbors分類器の重要なパラメーター**
  - 近傍点の数
  - データポイント間の距離測度
- 実用上は、近傍点の数は3や5程度の小さな数値で十分な場合がほとんどだが調整する必要がある
- **適切な距離測度の選択**はこの本では説明しないが、デフォルトではユークリッド距離を用いることが多い

- k-最近傍法の利点：モデルがわかりやすい、あまり調整しなくても性能がよい
- より高度な技術を利用する前に、このアルゴリズムをベースラインとして試すと良い
- 最近傍法のモデル構築は高速だが、訓練セットが大きくなると遅くなる
- k-最近傍法アルゴリズムを使う場合は、データの前処理が重要（3章 教師なし学習と前処理、を参照）
- k-最近傍法は、多数の特徴量（数百以上）を持つデータセットではうまく機能しない
- またほとんどの特徴量が多くの場合0となるような疎なデータセット（sparse dataset）だと特に性能が悪い
- つまり、k-最近傍法は理解しやすいメリットはあるが、処理速度が遅く多数の特徴量を扱うことができないため
- **実際にはほとんど利用されていない**。次の手法にはこれらの問題点はない