情報工学実験4：データマイニング班

(week 5) 線形回帰モデルの多項式拡張、過学習とその回避

1. （復習）線形回帰モデルの実装
2. 入出力における線形と非線形
3. モデルの線形性
4. 多項式モデルによる線形回帰モデルの拡張
5. 実装演習1（多項式モデル導入）
6. 過学習と代表的な回避手段
    1. 実装演習2（ペナルティの導入）
    2. 実装演習3（交差確認）
7. 参考文献

実験ページ: http://ie.u-ryukyu.ac.jp/~tnal/2019/info4/dm/ - week5

目次

regression (ver.2: fit())

review

```
# testing
>>> import importlib
>>> importlib.reload(regression)
>>> model = regression.LinearRegression()
>>> model.fit(X, Y)
>>> model.theta
array([ 5.30412371,  0.49484536])
```

```
def fit(self, input, output):
    self.theta =
np.dot(np.dot(np.linalg.inv(np.dot(input.T,input)
),input.T),output)
```

$$\theta = (X^T X)^{-1} X^T Y$$

regression (ver.3: predict())

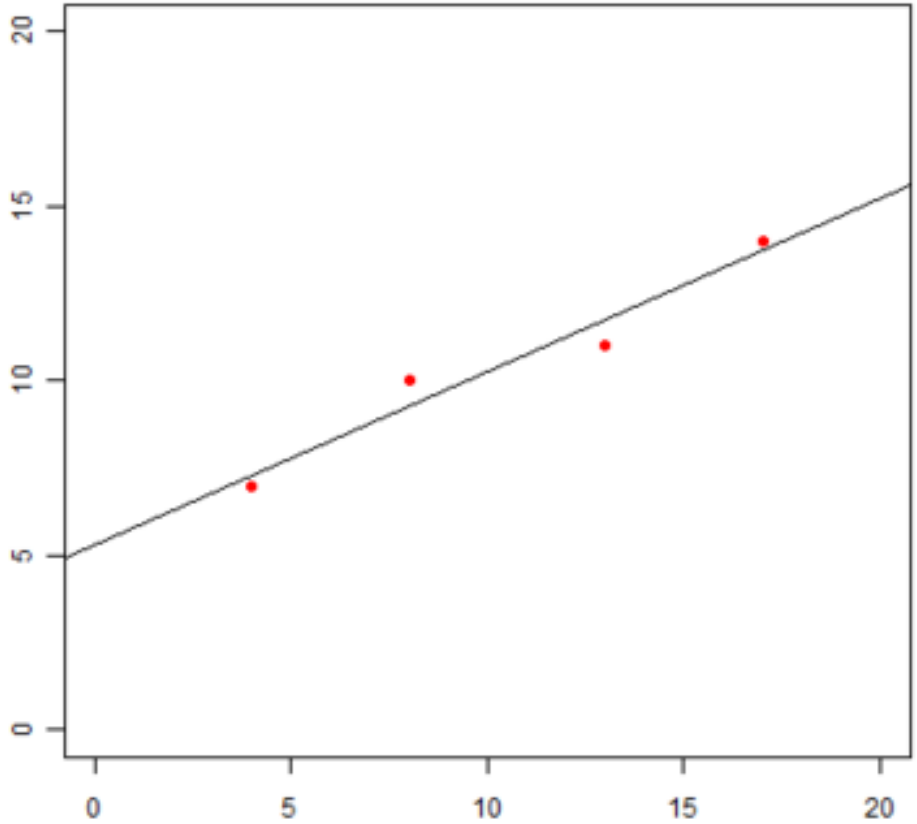
review

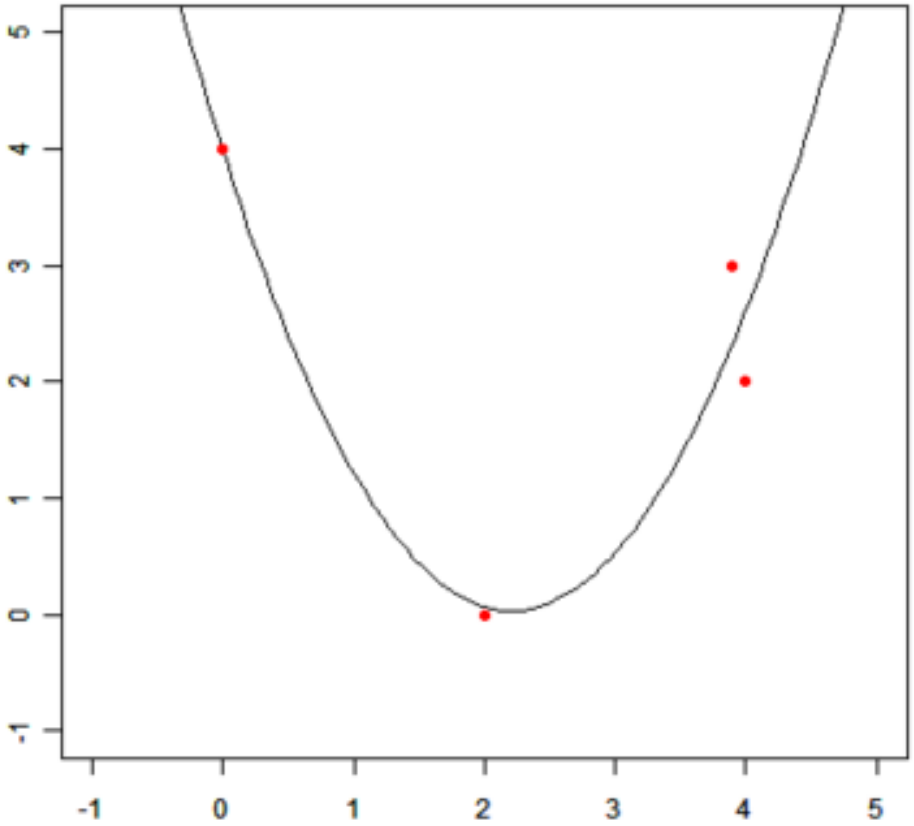
```
# testing
>>> importlib.reload(regression)
>>> model = regression.LinearRegression()
>>> model.fit(X, Y)
>>> model.predict(X)
array([  7.28350515,   9.2628866 ,
11.7371134 ,  13.71649485])
```

```
def predict(self, input):
    return np.dot(input, self.theta)
```

目次

Linear vs. Non-linear (input vs. output)





http://gihyo.jp/dev/serial/01/machine-learning/0008
http://gihyo.jp/dev/serial/01/machine-learning/0009?page=2

“Linear” Regression model

- Linearity
  - This means that the mean of the response variable is a “linear combination” of the parameters (regression coefficients).
  - When the feature(s) include non-linear variable(s) with raw feature, the model can explain a “non-linear” phenomena.
  - e.g., $y = a + bx + cx^2 + dx^3$

general definition

$$h_\theta(x) = \sum \theta_i \Phi_i(x) = \sum \theta_i x_i$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

simple case

Polynomial expansion (polynomial regression)

- expansion of h(x) with 2nd degree of polynomial function.

$$h_\theta(x) = \sum \theta_i \Phi_i(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

目次

Policy of implementation

- don’t change the regression model (regression.py)
- the input X must be expanded with polynomial function, before model.fit().

[before] Class design / How to use


```
# from numpy as np
# X = np.array([[1,4],[1,8],[1,13],[1,17]])
# Y = np.array([7, 10, 11, 14])
>>> import datasets
>>> X, Y = datasets.load_linear_example1()
>>> import regression
>>> model = regression.LinearRegression()
>>> model.fit(X, Y)
>>> model.theta
array([ 5.30412371,  0.49484536])
>>> model.predict(X)
array([  7.28350515,   9.2628866 ,  11.7371134 ,  13.71649485])
>>> model.score(X, Y) # RSS
1.2474226804123705
```

https://github.com/naltoma/regression-test.git

[after] Class design / How to use

```
# from numpy as np
# X = np.array([[1, 0.0], [1, 2.0], [1, 3.9], [1, 4.0]])
# Y = np.array([4.0, 0.0, 3.0, 2.0])
>>> import datasets
>>> X, Y = datasets.load_nonlinear_example1()
>>> ex_X = datasets.polynomial2_features(X)
>>> import regression
>>> model = regression.LinearRegression()
>>> model.fit(ex_X, Y)
>>> model.theta
array([ 3.98420277, -3.57732329,  0.8088239 ])
>>> model.predict(ex_X)
array([ 3.98420277,  0.06485179,  2.33485345,  2.616092  ])
>>> model.score(ex_X, Y) # RSS
0.8264445942622757 9
```

https://github.com/naltoma/regression-test.git

Numpy Tips (array concatenation)

```
>>> a =
np.array([[1, 2, 3],
[4, 5, 6]])
>>> b =
np.array([[7, 8, 9],
[10, 11, 12]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> b
array([[7, 8, 9],
       [10, 11, 12]])
```

```
>>> np.r_[a, b]
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
>>> np.c_[a, b]
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
>>> np.eye(2)
array([[ 1.,  0.],
       [ 0.,  1.]])
```

datasets.py (ver.2)

```
# testing
>>> import datasets
>>> X, Y = datasets.load_nonlinear_example1()
>>> ex_X = datasets.polynomial2_features(X)
>>> ex_X
array([[ 1. ,  0. ,  0. ],
       [ 1. ,  2. ,  4. ],
       [ 1. ,  3.9 ,  15.21],
       [ 1. ,  4. ,  16. ]])
>>> Y
array([ 4.,  0.,  3.,  2.])
```

if correct, then add & commit!

```
def load_nonlinear_example1():
    X = np.array([[1, 0.0], [1, 2.0], [1, 3.9], [1, 4.0]])
    Y = np.array([4.0, 0.0, 3.0, 2.0])
    return X, Y

def polynomial2_features(input):
    poly2 = input[:,1:]**2
    return np.c_[input, poly2]
```

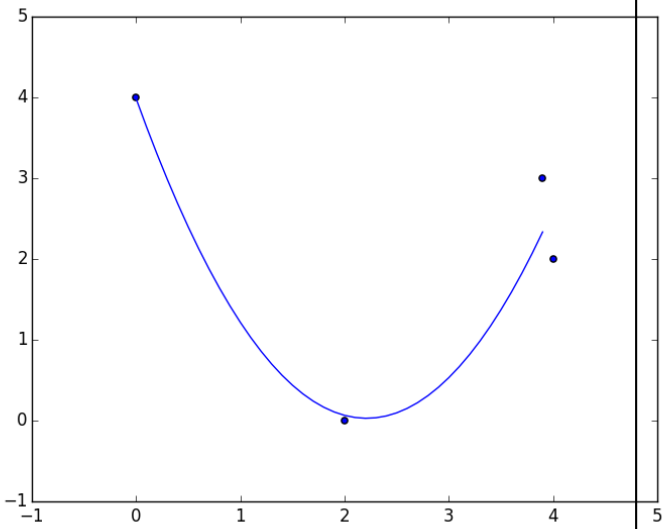
illustrate the model.predict(): nonlinear_ex.py

```
import numpy as np
import datasets
import regression

X, Y = datasets.load_nonlinear_example1()
ex_X = datasets.polynomial2_features(X)
model = regression.LinearRegression()
model.fit(ex_X, Y)

samples = np.arange(0, 4, 0.1)
x_samples = np.c_[ np.ones(len(samples)), samples ]
ex_x_samples = datasets.polynomial2_features(x_samples)

import matplotlib.pyplot as plt
plt.scatter(X[:,1], Y)
plt.plot(samples, model.predict(ex_x_samples))
plt.show()
```



datasets.py (ver.3)

```
# testing
>>> import datasets
>>> X, Y = datasets.load_nonlinear_example1()
>>> ex_X = datasets.polynomial3_features(X)
>>> ex_X
array([[ 1.   ,  0.   ,  0.   ,  0.   ],
       [ 1.   ,  2.   ,  4.   ,  8.   ],
       [ 1.   ,  3.9  , 15.21 , 59.319],
       [ 1.   ,  4.   , 16.   , 64.   ]])
>>> Y
array([ 4.,  0.,  3.,  2.])
```

```
def polynomial3_features(input):
    poly2 = input[:,1:]**2
    poly3 = input[:,1:]**3
    return np.c_[input, poly2, poly3]
```

more general?

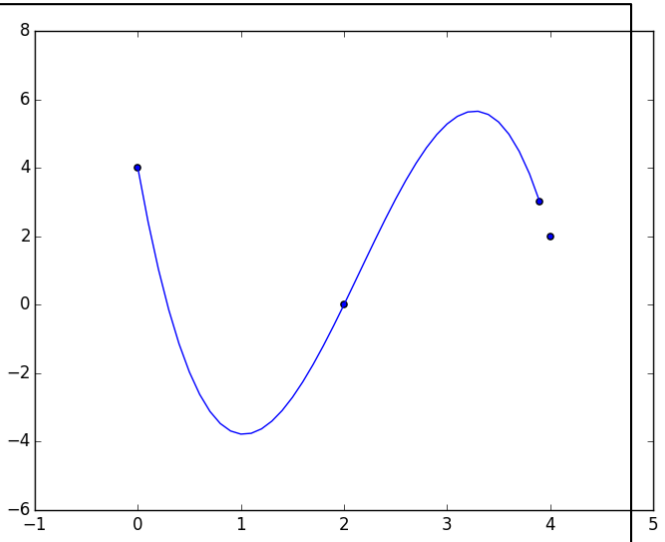
illustrate the model.predict(): nonlinear_ex2.py

```
import numpy as np
import datasets
import regression

X, Y = datasets.load_nonlinear_example1()
ex_X = datasets.polynomial3_features(X)
model = regression.LinearRegression()
model.fit(ex_X, Y)

samples = np.arange(0, 4, 0.1)
x_samples = np.c_[ np.ones(len(samples)), samples ]
ex_x_samples = datasets.polynomial3_features(x_samples)

import matplotlib.pyplot as plt
plt.scatter(X[:,1], Y)
plt.plot(samples, model.predict(ex_x_samples))
plt.show()
```



目次

- （復習）クラスタリング、検討演習
- （復習）線形回帰モデルの実装
- 入出力における線形と非線形
- モデルの線形性
- 線形回帰モデルへの多項式モデル導入（入力調整で対応）
- 実装演習1
  - 実装ポリシー、クラスデザイン
  - Numpy Tips（配列結合、単位行列）
  - datasets.py へ多項式モデル拡張関数（h(x)=x+x^2）の追加
  - 回帰ライン描画による動作確認
  - datasets.py へ多項式モデル拡張関数（h(x)=x+x^2+x^3）の追加
  - 回帰ライン描画による動作確認

- 過学習と代表的な回避手段
- ペナルティ項（L2-norm）の導入
- 実装演習2（ペナルティの導入）
  - 実装ポリシー、クラスデザイン
  - regression.py の拡張（RidgeRegression()）
  - RidgeRegression.fit() 更新
  - ペナルティの効果検証
  - 演習課題
- 実装演習3（交差確認）
  - 素朴なコード例
  - scikit-learn ライブラリを用いるためのクラス修正
  - sklearn.cross_validation を使った交差確認
- 参考文献

one of measurement

the parameters vs. over-fitting

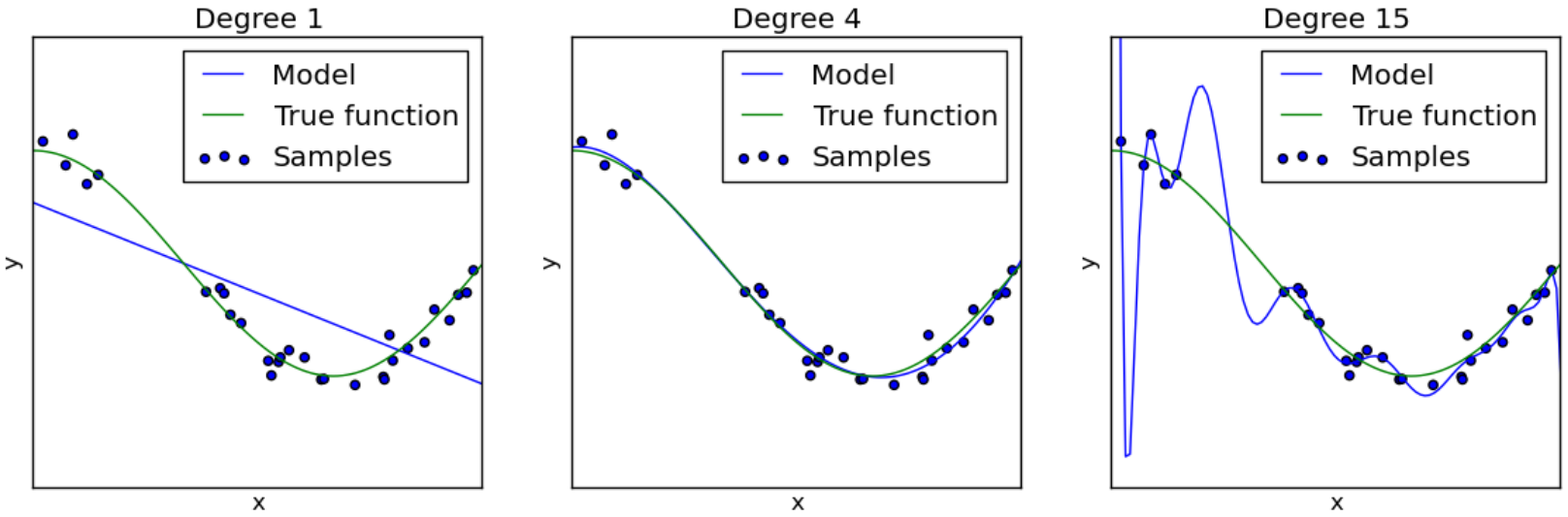
- nonlinear_ex.py (2nd degree of polynomial)
  - model.theta = [3.98420277, -3.57732329, 0.8088239]
  - model.score = 0.8264459426227279
- nonlinear_ex2.py (3rd degree of polynomial)
  - model.theta = [4.   , -16.91430499, 10.81072874, -1.67678812]
  - model.score = 4.1869905232912016e-22
- Over-fitting
  - A modeling error which occurs when a function is too closely fit to a limited set of data points. Over-fitting the model generally takes the form of making an overly complex model to explain idiosyncrasies in the data under study. In reality, the data being studied often has some degree of error or random noise within it. Thus attempting to make the model conform too closely to slightly inaccurate data can infect the model with substantial errors and reduce its predictive power.
  - http://www.investopedia.com/terms/o/overfitting.asp

how to avoid?

http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

Underfitting vs. Overfitting

This example demonstrates the problems of underfitting and overfitting and how we can use linear regression with polynomial features to approximate nonlinear functions. The plot shows the function that we want to approximate, which is a part of the cosine function. In addition, the samples from the real function and the approximations of different models are displayed. The models have polynomial features of different degrees. We can see that a linear function (polynomial with degree 1) is not sufficient to fit the training samples. This is called **underfitting**. A polynomial of degree 4 approximates the true function almost perfectly. However, for higher degrees the model will **overfit** the training data, i.e. it learns the noise of the training data.



Some ways to avoid over-fitting

- ready for HUGE dataset.
- develop the dataset to quality. (more noiseless)
- penalize overly complex models.
  - e.g., complex models ≒ largest parameters
- test the model's ability on unseen dataset.
  - e.g., cross-validation tests

a penalty for the parameters (generalization)

- introduce a penalty term to cost function J(θ).
  - sum of squared parameters (L2-norm)

before: $$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Linear Regression

after: $$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\|\theta\|^2$$

$$\|\theta\|^2 = \theta_0^2 + \theta_1^2 + \ldots + \theta_M^2$$

Ridge Regression

目次

Policy of implementation 2

- don’t change LinearRegression() class.
  - because the difference between LinearRegression() and RidgeRegression() is only fit().
- use class inheritance mechanism to implement RidgeRegression().

[after2] Class design / How to use

```
# from numpy as np
# X = np.array([[1, 0.0], [1, 2.0], [1, 3.9], [1, 4.0]])
# Y = np.array([4.0, 0.0, 3.0, 2.0])
>>> import datasets
>>> X, Y = datasets.load_nonlinear_example1()
>>> ex_X = datasets.polynomial3_features(X)
>>> import regression
>>> model = regression.RidgeRegression(alpha=0.5)
>>> model = regression.RidgeRegression()    #default: alpha=0.1
>>> model.fit(ex_X, Y)
>>> model.theta
array([ 3.54259714, -1.24971967, -0.68925104,  0.23695052])
>>> model.predict(ex_X)
array([ 3.54259714,  0.1817578 ,  2.24085012,  2.68053522])
>>> model.score(ex_X, Y) # RSS
1.2816900115950909
```

https://github.com/naltoma/regression-test.git

Ridge regression
(regression.py, ver. 5)

```
# testing
>>> import importlib
>>> importlib.reload(regression)
>>> model = regression.RidgeRegression()
>>> model.alpha
0.1
```

if correct, then add & commit!

```
class RidgeRegression(LinearRegression):
    alpha = None

    def __init__(self, alpha=0.1):
        self.alpha = alpha

    def fit(self, input, output):
        pass
```

Ridge regression
(regression.py, ver. 6)

```
# testing
>>> importlib.reload(regression)
>>> model = regression.RidgeRegression()
>>> model.fit(ex_X, Y)
>>> model.theta
array([ 3.54259714, -1.24971967, -
0.68925104,  0.23695052])
```

if correct, then add & commit!

```
    def fit(self, input, output):
        xTx = np.dot(input.T, input)
        I = np.eye(len(xTx))
        self.theta = np.dot(np.dot(np.linalg.inv(xTx + self.alpha*I),
input.T),output)
```

$$\theta = (X^T X + \alpha I)^{-1} X^T Y$$

$$\alpha := alpha$$

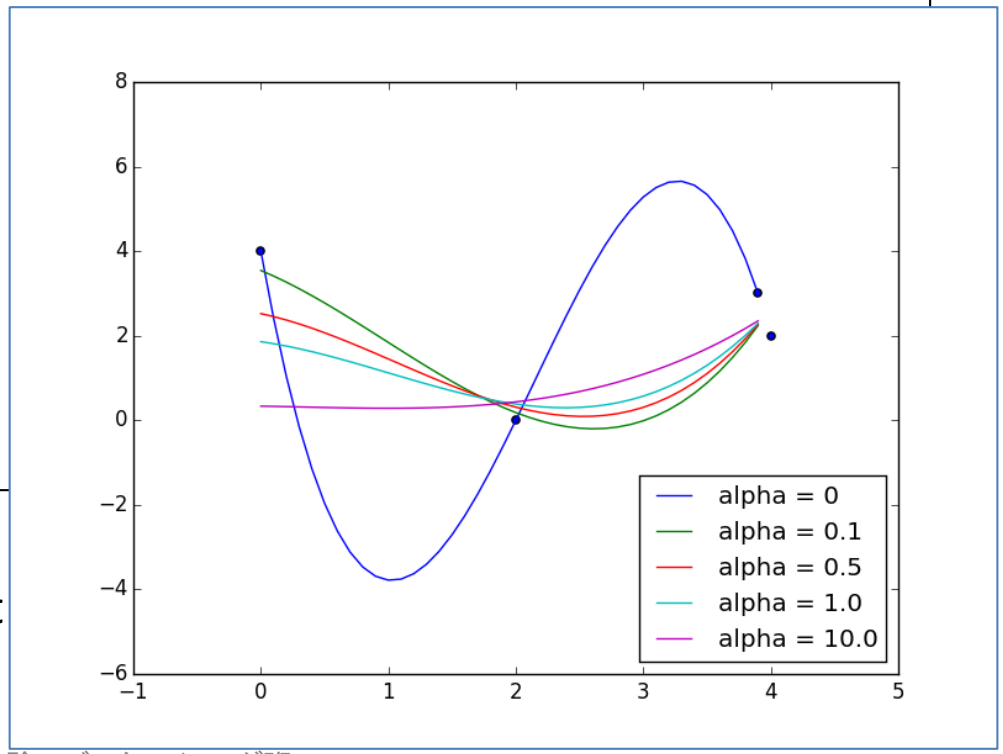
$$I := identity\_matrix$$

Effect of the penalty

- alpha=0 # == LinearRegression()
  - theta: [4. -16.91430499 10.81072874 -1.67678812]
  - score: 7.4647921109305001e-22
- alpha=0.1
  - theta: [3.54259714, -1.24971967, -0.68925104, 0.23695052]
  - score: 1.2816900115950909
- alpha=0.5
  - theta: [2.52220383, -0.63725353, -0.63511135, 0.20043682]
  - score: 3.2271319080413789
- alpha=1.0
  - theta: [1.85895448, -0.43056141, -0.46106559, 0.1538384]
  - score: 5.5987129498416079
- alpha=10.0
  - theta: [0.33402625, -0.04968343, -0.04987846, 0.05004393]
  - score: 14.342958761816003

Exercise

- preconditions
  - X, Y = datasets.load_nonlinear_example1()
  - ex_X = datasets.polynomial3_features(X)
- Illustrate the samples and regression lines on RidgeRegression() with alpha = {0, 0.1, 0.5, 1.0, 10.0}.



code example
https://github.com/naltoma/regression-test.git

目次

- （復習）線形回帰モデルの実装
- 入出力における線形と非線形
- モデルの線形性
- 線形回帰モデルへの多項式モデル導入（入力調整で対応）
- 実装演習1
  - 実装ポリシー、クラスデザイン
  - Numpy Tips（配列結合、単位行列）
  - datasets.py へ多項式モデル拡張関数（h(x)=x+x^2）の追加
  - 回帰ライン描画による動作確認
  - datasets.py へ多項式モデル拡張関数（h(x)=x+x^2+x^3）の追加
  - 回帰ライン描画による動作確認

- 過学習と代表的な回避手段
- ペナルティ項（L2-norm）の導入
- 実装演習2（ペナルティの導入）
  - 実装ポリシー、クラスデザイン
  - regression.py の拡張（RidgeRegression()）
  - RidgeRegression.fit() 更新
  - ペナルティの効果検証
  - 演習課題
- 実装演習3（交差確認）
  - 素朴なコード例
  - scikit-learn ライブラリを用いるためのクラス修正
  - sklearn.cross_validation を使った交差確認
- 参考文献

Some ways to avoid over-fitting

- ready for HUGE dataset.
- develop the dataset to quality. (more noiseless)
- penalize overly complex models.
  - e.g., complex models ≒ largest parameters
- test the model's ability on unseen dataset.
  - e.g., cross-validation tests

Cross-validation

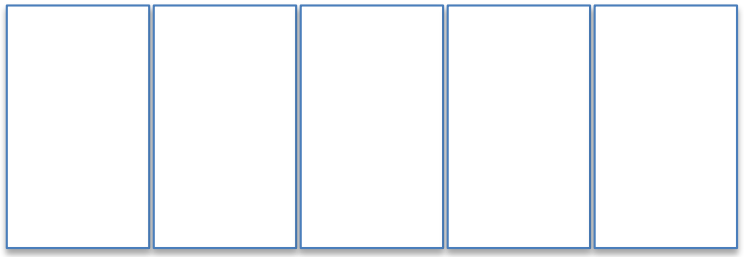
- scikit-learn: 3.1. Cross-validation: evaluating estimator performance
  - http://scikit-learn.org/stable/modules/cross_validation.html
  - Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called **overfitting**. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a **test set** X_test, y_test.

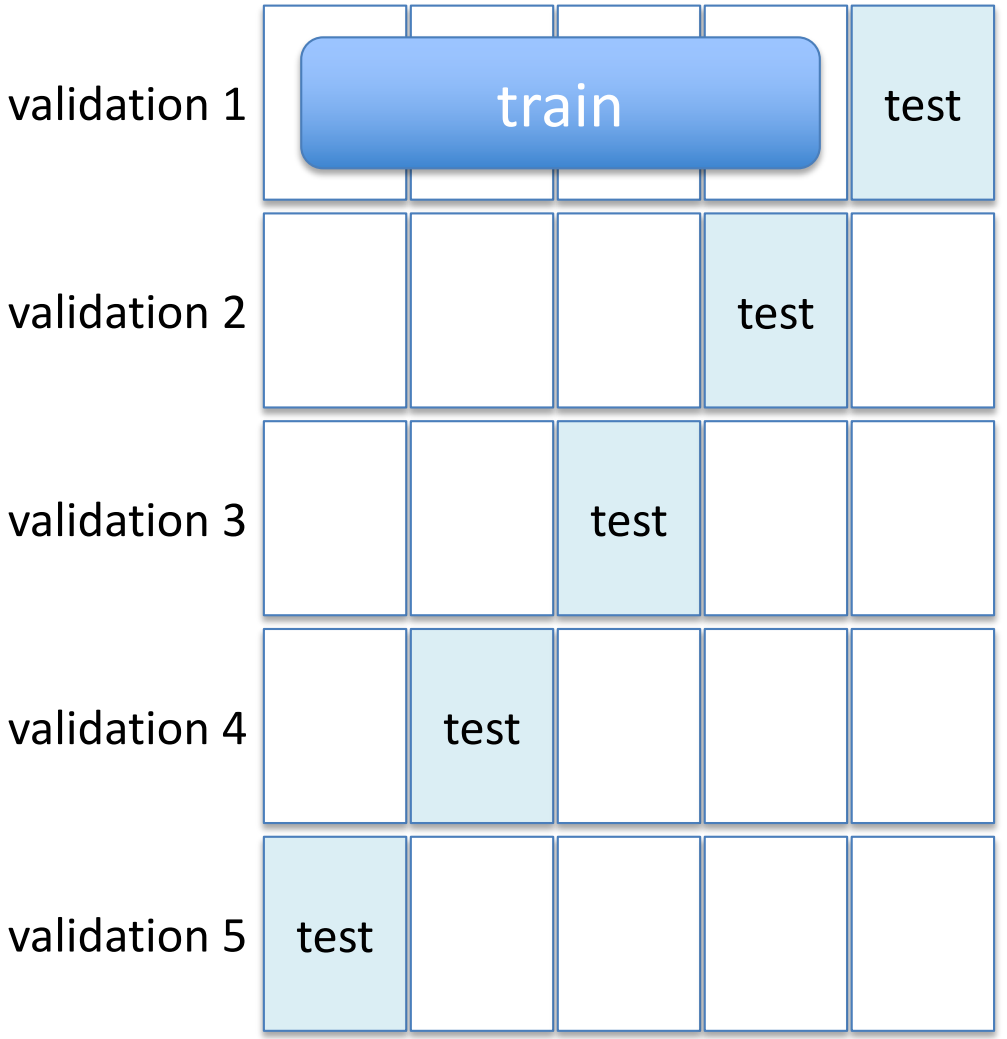
K-fold cross-validation (K=5)

(step 2) The model.fit() is learned using K-1 folds (4 folds), and the fold left out is used for test.

total dataset

(step 1) divides all the samples in K groups (folds) of equal sizes, if possible.





Example of K=2 cross-validation: crossvalidation.py

```
from sklearn import datasets
boston = datasets.load_boston()
x = boston.data
y = boston.target
half = int(len(x)/2)

import regression
model = regression.RidgeRegression(alpha=0.1)
# case 1: learn on the first half, test on the last half
model.fit(x[:half], y[:half])
score = model.score(x[half:], y[half:])
# case 2: learn on the last half, test on the first half
model.fit(x[half:], y[half:])
score += model.score(x[:half], y[:half])
print("RidgeRegression(alpha=0.1) score =", score)
#-> RidgeRegression(alpha=0.1) score = 78656.6246552
#-> RidgeRegression(alpha=1.0) score = 42334.1689238
```

Cross-validation using scikit-learn module (1/2)

update RidgeRegression class (regression.py, ver.7)

```
class RidgeRegression(LinearRegression):

    # for scikit-learn
    def get_params(self, deep=True):
        return {'alpha':self.alpha}

    # (OPTION) the coefficient of determination R^2.
    # see sklearn.linear_model.Ridge().score()
    # http://goo.gl/v93tNM
    def score2(self, input, output):
        u = ((output - self.predict(input)) ** 2).sum()
        v = ((output - output.mean()) ** 2).sum()
        return (1 - u/v)
```

Cross-validation using scikit-learn module (2/2)

crossvalidation_sklearn.py

```
from sklearn import datasets
boston = datasets.load_boston()
x = boston.data
y = boston.target

import numpy as np
ones = np.ones((len(x),1))
ex_x = np.c_[ones, x]

import regression
alpha = 0.1
model = regression.RidgeRegression(alpha=alpha)

#from sklearn import cross_validation #will be removed in 0.20
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, ex_x, y, cv=10, n_jobs=-1)
print("*** Ridge(alpha=%0.2f) ***" % alpha)
print("scores=", scores)
print("mean score = %f (+/- %0.2f)" % (scores.mean(), scores.std()*2))
```

References

- Machine Learning in Action, http://www.manning.com/pharrington/
- Tikhonov regularization – Wikipedia, http://en.wikipedia.org/wiki/Tikhonov_regularization
- 機械学習 by Masafumi Noda, http://www.slideshare.net/masafuminoda/machine-learning-11767735
- 線形回帰による曲線フィッティング, http://aidiary.hatenablog.com/entry/20140402/1396445570
- 過学習を防ぐ正則化, http://gihyo.jp/dev/serial/01/machine-learning/0009?page=3
- 正則化 (regularization) – 機械学習の「朱鷺の杜Wiki」, http://ibisforest.org/index.php?正則化
- リッジ回帰 (ridge regression) – 機械学習の「朱鷺の杜Wiki」, http://ibisforest.org/index.php?リッジ回帰
- Cross-validation: evaluating estimator performance – scikit-learn, http://scikit-learn.org/stable/modules/cross_validation.html
- Underfitting vs. Overfitting – scikit-learn, http://scikit-learn.org/stable/auto_examples/plot_underfitting_overfitting.html
- 交差確認 (cross validation) – 機械学習の「朱鷺の杜Wiki」, http://ibisforest.org/index.php?交差確認