レポート No.1

目次

1.　応用数学

1.1.　線形代数学

・スカラーとベクトルの違い

　スカラー：四則演算ができる

　ベクトル：大きさや向きを表せる、数字の組み合わせ

・行列

　ベクトルの変換ができる

　連立１次方程式を行列で表現できる、連立１次方程式の係数

　行列の積：行と列の積で新たな成分を作る

　例題

$$\begin{pmatrix} 2 & 1 \\ 4 & 1 \end{pmatrix}\begin{pmatrix} 1 & 3 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 2\times1+1\times3 & 2\times3+1\times1 \\ 4\times1+1\times3 & 4\times3+1\times1 \end{pmatrix}$$

$$=\begin{pmatrix} 5 & 7 \\ 7 & 13 \end{pmatrix}$$

　行基本変形

　　１、i 行目をＣ倍

　　２、s 行目に t 行目のＣ倍を加える

　　３、p 行目と q 行目を入れ替える

i行目をc倍する

$$Q_{i,c} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

s行目にt行目のc倍を加える

$$R_{s,t,c} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & c & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

p行目とq行目を入れ替える

$$P_{p,q} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

逆行列　インバース

単位行列

はきだし法

問題

$\begin{pmatrix} 4 & 7 \\ 1 & 2 \end{pmatrix}$ の逆行列を求めよ

$$\begin{pmatrix} 4 & 7 \\ 1 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & -7 \\ -1 & 4 \end{pmatrix}$$

$$\left(\begin{array}{cc|cc} 4 & 7 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{array}\right)$$

1行目に2行目の-4倍を加える

$$\left(\begin{array}{cc|cc} 0 & -1 & 1 & -4 \\ 1 & 2 & 0 & 1 \end{array}\right)$$

2行目に1行目の2倍を加える

$$\left(\begin{array}{cc|cc} 0 & -1 & 1 & -4 \\ 1 & 0 & 2 & -7 \end{array}\right)$$

1行目を-1倍する

$$\left(\begin{array}{cc|cc} 0 & 1 & -1 & 4 \\ 1 & 0 & 2 & -7 \end{array}\right)$$

1行目と2行目を入れ替える

$$\left(\begin{array}{cc|cc} 1 & 0 & 2 & -7 \\ 0 & 1 & -1 & 4 \end{array}\right)$$

逆行列が存在しない条件

・解が１組に定まらない。(傾きが同じ、本質的に同じ式)

$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$　a:b≠c:d の時は逆行列を持つ、

a:b=c:d の時に逆行列をもたない

ad=bc　：同じ傾きだから解が定まらない

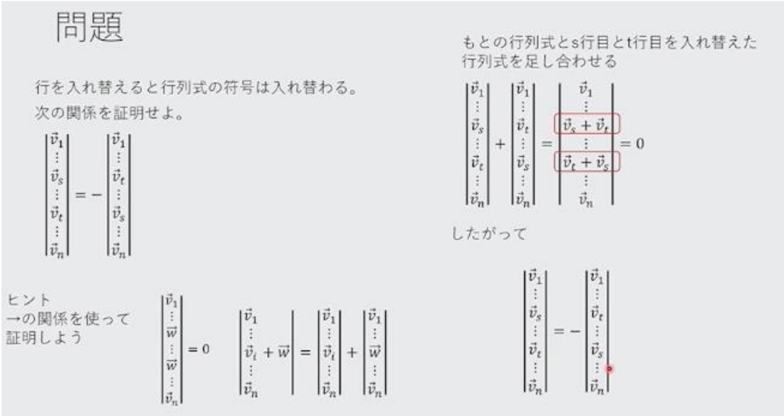
ad-bc=0

行列式　同じベクトルを含むとゼロ

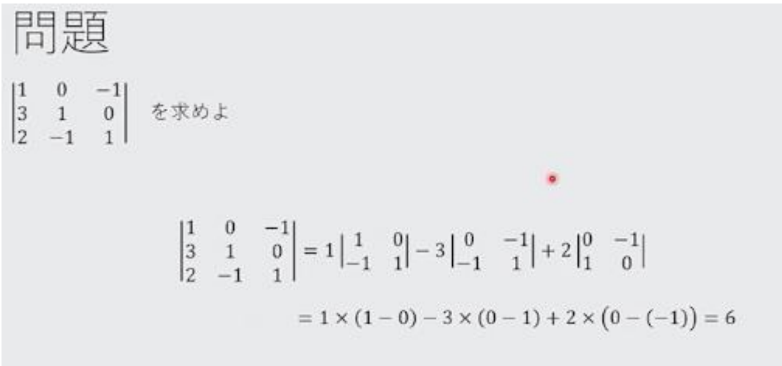
1 つのベクトルがλ倍されると行列式はλ倍

他の成分が全部同じで 1 番目のベクトルだけ違う場合、行列の和

行を入れ替えると符号が変わる



行列式の求め方



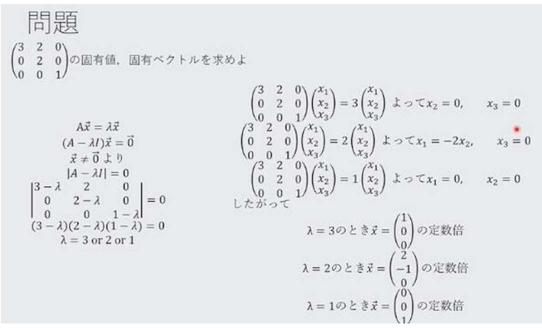
・固有値・固有ベクトルの求め方

$$A\vec{x} = \lambda\vec{x}$$

固有値λは 1 つ

固有ベクトルは一つではない、～～の定数倍

固有値λから求める



・固有値分解

分解によって、似たような特徴でてくる。

固有値は NxN の行列であれば、N 個ある。
行列は割り算しない、逆行列を掛ける。

問題

$\begin{pmatrix}2 & 1\\ 0 & 6\end{pmatrix}$を固有値分解せよ

$$A\vec{x} = \lambda\vec{x}$$
$$(A-\lambda I)\vec{x} = \vec{0}$$
$$\vec{x} \neq \vec{0} \text{ より}$$
$$|A-\lambda I| = 0$$
$$\begin{vmatrix}2-\lambda & 1\\ 0 & 6-\lambda\end{vmatrix} = 0$$
$$(2-\lambda)(6-\lambda) = 0$$
$$\lambda = 2 \text{ or } 6$$
$$\begin{pmatrix}2 & 1\\ 0 & 6\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} = 2\begin{pmatrix}x_1\\ x_2\end{pmatrix} \text{ よって} x_2 = 0$$
$$\begin{pmatrix}2 & 1\\ 0 & 6\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} = 6\begin{pmatrix}x_1\\ x_2\end{pmatrix} \text{ よって} 4x_1 = x_2$$

したがって

$\lambda = 2$のとき$\vec{x} = \begin{pmatrix}1\\ 0\end{pmatrix}$の定数倍

$\lambda = 6$のとき$\vec{x} = \begin{pmatrix}1\\ 4\end{pmatrix}$の定数倍

つまり

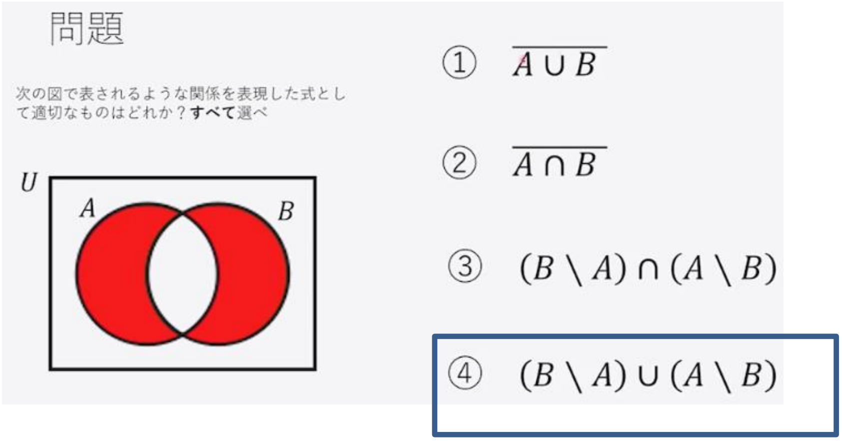
$$\begin{pmatrix}2 & 1\\ 0 & 6\end{pmatrix} = \begin{pmatrix}1 & 1\\ 0 & 4\end{pmatrix}\begin{pmatrix}2 & 0\\ 0 & 6\end{pmatrix}\begin{pmatrix}1 & 1\\ 0 & 4\end{pmatrix}^{-1}$$
$$\begin{pmatrix}2 & 1\\ 0 & 6\end{pmatrix} = \begin{pmatrix}1 & 1\\ 0 & 4\end{pmatrix}\begin{pmatrix}2 & 0\\ 0 & 6\end{pmatrix}\begin{pmatrix}1 & -1/4\\ 0 & 1/4\end{pmatrix}$$

・特異値・特異ベクトルの概要
　正方行列以外の固有値分解

・特異値分解
　固有値分解の親戚のように分解できる
　長方形の行列を転置してかけると固有値分解できる
　特異値分解でできることは固有値分解でもできる
　画像をぼかしてデータ量の削減ができる。

1.2.　統計学

・集合
　S={a,b,c,d,e,f,g}
　$a \in S$
　・和集合　$A \cup B$　（A カップ B）
　・共通部分　$A \cap B$　（A キャップ B）
　・絶対補　　自分以外　$U¥A = \bar{A}$
　・相対補　　B¥A　差集合　B から A を除く



・確率

頻度確率（客観確率）発生する頻度、きちんと測定する

ベイズ確率（主観確率）信念の度合い　いろいろな条件をもとに導き出す

確率＝P(A)=事象 A が起こる確率／すべての事象の数　　：(ゼロ～最大でも１)

$P(A \cap B) = P(A)P(B|A)$　A と B の共通部分

$P(A \cap B) = P(B \cap A)$

$P(A)P(B|A) = P(B)P(A|B)$

「条件付き確率」上記の縦棒

・ある事情 B が与えられた下で、A となる確率

$P(A|B) = \frac{P(A \cap B)}{P(B)}$

独立な事象の同時確率、お互いの発生には因果関係がない

$P(A \cap B) = P(B)P(A|B) = P(A)P(B)$　:掛け算のような

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$　:2 重に数えているので、引く

ベイズ則

P(飴玉)=1/4　P(笑顔｜飴玉)=1/2　 P(笑顔)=1/3

P(笑顔｜飴玉)×P(飴玉)＝P(笑顔, 飴玉)　 1/2×1/4=1/8

P(笑顔, 飴玉)＝P(飴玉, 笑顔)

P(飴玉, 笑顔)＝P(飴玉｜笑顔)×P(笑顔)

1/8 = P(飴玉｜笑顔) ×1/3

P(飴玉｜笑顔) ＝3/8

・統計

記述統計＝集団の性質を要約して記述

推測統計＝集団から一部を取り出し元の母集団の性質を推測

確率変数＝事象と結び付けられた数値

確率分布＝事象の発生する確率の分布

期待値＝平均、「ありえそうな値」

確率×確率変数の平均　連続した値も求められる

分散＝データの散らばり具合、期待値(平均)からのズレの 2 乗、(2 乗なので単位が違う)

共分散＝2 つのデータ系列の傾向の違い

$$
\begin{aligned}
&\text{分散}\mathrm{Var}(f)\\
&= \mathrm{E}\left(\left(f_{(X=x)} - \mathrm{E}_{(f)} \right)^2 \right)\\
&= \mathrm{E}\left(f_{(X=x)}^2 \right) - \left(\mathrm{E}_{(f)} \right)^2
\end{aligned}
$$

$$
\begin{aligned}
&\text{共分散}\mathrm{Cov}(f,g)\\
&= \mathrm{E}\left(\left(f_{(X=x)} - \mathrm{E}(f) \right) \left(g_{(Y=y)} - \mathrm{E}(g) \right) \right)\\
&= \mathrm{E}(fg) - \mathrm{E}(f)\mathrm{E}(g)
\end{aligned}
$$

標準偏差＝分散のルート、（元の単位に戻る）

確率分布

・ベルヌーイ分布＝コイントス、裏表だけ

・マルチヌーイ（カテゴリカル）分布＝サイコロ、いろんな面がある

・二項分布＝ベルヌーイ分布の多試行

・ガウス分布＝釣鐘型の連続分布、サンプルが多ければ正規分布に近づく

面積が 1 になるようにしてある

推定＝母集団を特徴づける母数(パラメータ)を統計学的に推測

点推定＝１つの値に推定

区間推定＝範囲（区間）を推定

推定量＝estimator 、関数みたいなもの、推定関数

推定値＝estimate、実際に試行して定まった値、

真の値に対して＾(ハット)を付けて区別する

標本平均＝母集団から取り出した標本の平均値

サンプル数が多ければ母集団に近づく、一致性

サンプル数が多くても期待値は母集団の値と同様、不偏性

標本分散＝一致性はある、不偏性はない

数が小さいと分散が大きいか小さいか

不偏分散＝$\frac{1}{n-1}\sum_{i=1}^{n}(xi-\bar{x})^2$　　n-1 がある

1.3. 情報科学

増加の「比率」の例。情報の増え方のわかりやすさ。

自己情報量＝エントロピーと似ている、情報のめずらしさ

底が e のときの単位 nat(ナット)、2 のときの単位 bit

$I(x)=-\log(P(x))=\log(W(x))$

シャノンエントロピー＝情報量の期待値（平均）

シャノンエントロピーが最大になるところを探す、誤差関数の使い道

カルバック・ライブラー　ダイバージェンス　＝距離に近い

同じ事象・確率変数における異なる確率分布 P,Q の違いを表す

想定した分布 Q、実際の分布 P、それぞれの分布がどれだけ似ているか

シャノンエントロピーに似ている

交差エントロピー＝KL ダイバージェンスの一部分

Q の自己情報量を P の分布で平均。

2. 機械学習

問題設定->データ選定->データの前処理->

->機械学習モデルの選定->モデルの学習(パラメータ推定)->モデルの評価

ルールベースモデルと機械学習モデルの違い

機械学習とは：トム・ミッチェル 1997 の定義

2.1. 線形回帰モデル

入力：説明変数、特徴量、m 次元のベクトル

出力：目的変数、スカラー値

説明変数 $\boldsymbol{x} = (x_1, x_2, \cdots, x_m)^T \in \mathbb{R}^m$

目的変数 $y \in \mathbb{R}^1$

教師データ

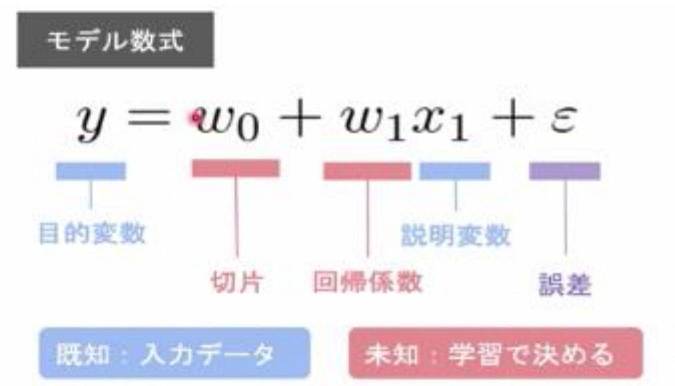
$$\{(\boldsymbol{x}_i, y_i); i = 1, ..., n\}$$

パラメータ $\boldsymbol{w} = (w_1, w_2, \cdots, w_m)^T \in \mathbb{R}^m$

線形結合 $\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + w_0 = \sum_{j=1}^{m} w_j x_j + w_0$

予測値はハットを付ける

パラメータは未知：最小二乗法により推定



汎化性能測定：データの分割

平均二乗誤差（残差平方和）　MSE

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{n_{\mathrm{train}}} \sum_{i=1}^{n_{\mathrm{train}}} (\hat{y}_i^{(\mathrm{train})} - y_i^{(\mathrm{train})})^2$$

最尤法(さいゆうほう)でも同じようにできる。尤度関数の最大化を利用。

「最尤法の解」と「最小二乗法の解」は一致する。

Scikit-learn (さいきっとらーん)

ハンズオン（実装演習）

ボストンハウジングデータを用いて、線形単回帰を行う

jupyter skl_regression Last Checkpoint: 18時間前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help　Not Trusted　Python 3

線形回帰モデル-Boston Hausing Data-

1. 必要モジュールとデータのインポート

In [1]:
```python
#from モジュール名 import クラス名（もしくは関数名や変数名）

from sklearn.datasets import load_boston
from pandas import DataFrame
import numpy as np
```

In [2]:
```python
# ボストンデータを"boston"というインスタンスにインポート
boston = load_boston()
```

In [3]:
```python
#インポートしたデータを確認(data / target / feature_names / DESCR)
print(boston)
```

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]]), 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
```

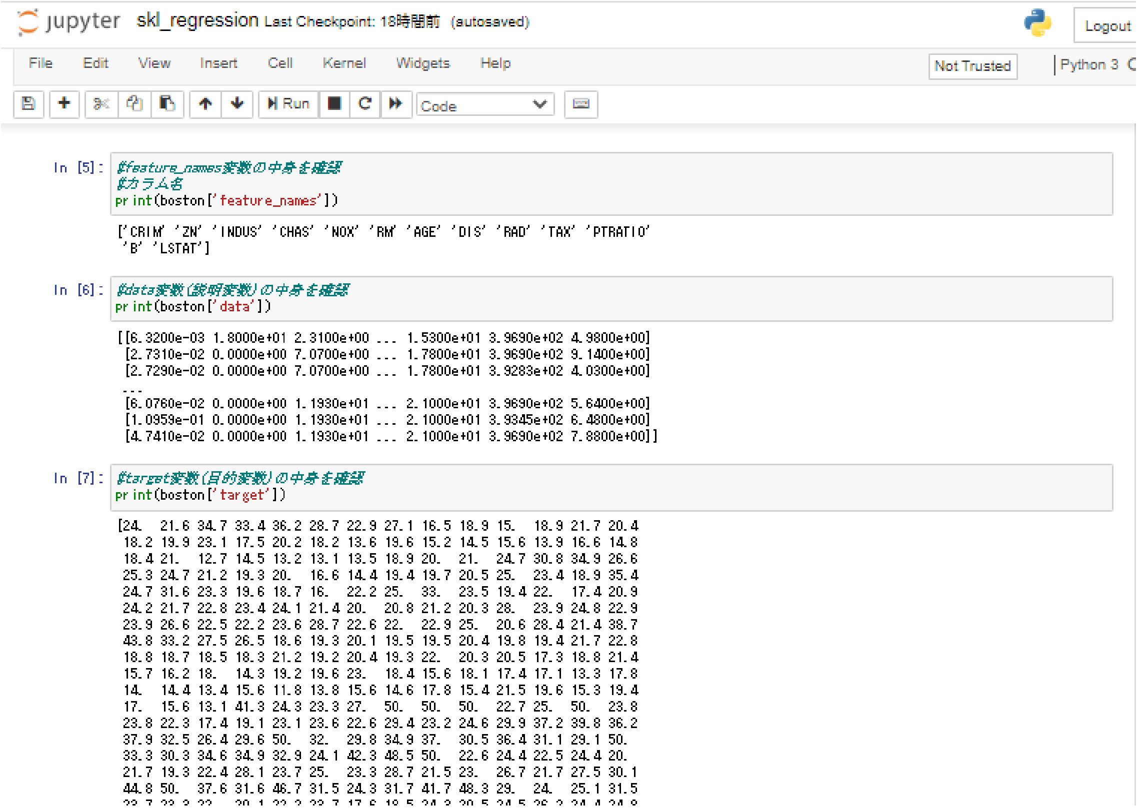
jupyter skl_regression Last Checkpoint: 18時間前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help　Not Trusted　Python 3

```
International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.¥n", 'filename': '/op
t/conda/lib/python3.6/site-packages/sklearn/datasets/data/boston_house_prices.csv'}
```

In [4]:
```python
#DESCR変数の中身を確認
print(boston['DESCR'])
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
```



boston の中に説明変数と目的変数が格納されていることを確認した。

jupyter skl_regression Last Checkpoint: 18時間前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Logout

2. データフレームの作成

In [8]:
```
#  説明変数らをDataFrameへ変換
df = DataFrame(data=boston.data, columns = boston.feature_names)
```

In [9]:
```
# 目的変数をDataFrameへ追加
df['PRICE'] = np.array(boston.target)
```

In [10]:
```
# 最初の5行を表示
df.head(5)
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

線形単回帰分析

In [11]:
```
#カラムを指定してデータを表示
df[['RM']].head()
```

Out[11]:

	RM
0	6.575
1	6.421
2	7.185
3	6.998
4	7.147

In [12]:
```
# 説明変数
data = df.loc[:, ['RM']].values
```

jupyter skl_regression Last Checkpoint: 18時間前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

```
In [12]: # 説明変数
         data = df.loc[:, ['RM']].values

In [13]: #dataリストの表示(1-5)
         data[0:5]

Out[13]: array([[6.575],
                [6.421],
                [7.185],
                [6.998],
                [7.147]])

In [14]: # 目的変数
         target = df.loc[:, 'PRICE'].values

In [15]: target[0:5]

Out[15]: array([24. , 21.6, 34.7, 33.4, 36.2])

In [16]: ## sklearnモジュールからLinearRegressionをインポート
         from sklearn.linear_model import LinearRegression

In [17]: # オブジェクト生成
         model = LinearRegression()
         #model.get_params()
         #model = LinearRegression(fit_intercept = True, normalize = False, copy_X = True,  n_jobs = 1)

In [18]: # fit関数でパラメータ推定
         model.fit(data, target)

Out[18]: LinearRegression()

In [21]: #予測
         model.predict([[1]])

Out[21]: array([-25.5685118])
```

単回帰での予測が行えた。

jupyter skl_regression Last Checkpoint: 18時間前 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted | Python 3

Run Code

Out [21]:
```
array([-25.5685118])
```

重回帰分析(2変数)

In [20]:
```
#カラムを指定してデータを表示
df[['CRIM', 'RM']].head()
```

Out [20]:

	CRIM	RM
0	0.00632	6.575
1	0.02731	6.421
2	0.02729	7.185
3	0.03237	6.998
4	0.06905	7.147

In [21]:
```
# 説明変数
data2 = df.loc[:, ['CRIM', 'RM']].values
# 目的変数
target2 = df.loc[:, 'PRICE'].values
```

In [22]:
```
# オブジェクト生成
model2 = LinearRegression()
```

In [23]:
```
# fit関数でパラメータ推定
model2.fit(data2, target2)
```

Out [23]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [24]:
```
model2.predict([[0.2, 7]])
```

Out [24]:
```
array([29.42867599])
```

回帰係数と切片の値を確認

In [25]:
```
# 単回帰の回帰係数と切片を出力
print('推定された回帰係数: %.3f, 推定された切片 : %.3f' % (model.coef_, model.intercept_))
```

```
推定された回帰係数: 9.102, 推定された切片 : -34.671
```

重回帰での予測が行えた。

In [26]:
```
# 重回帰の回帰係数と切片を出力
print(model.coef_)
print(model.intercept_)
```
```
[9.10210898]
-34.67062077643857
```

モデルの検証

1. 決定係数

決定係数

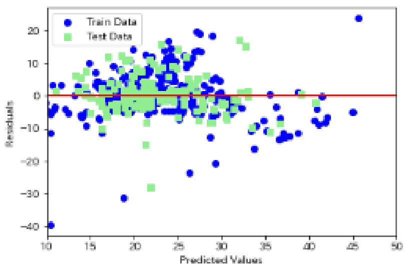
print('単回帰決定係数: %.3f, 重回帰決定係数 : %.3f' % (model.score(data,target), model2.score(data2,target2)))

In [27]:
```
# train_test_splitをインポート
from sklearn.cross_validation import train_test_split
```
```
C:¥Users¥takam¥Anaconda3¥lib¥site-packages¥sklearn¥cross_validation.py:41: DeprecationWarning: This module was deprecated in versi
on 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the i
nterface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [28]:
```
# 70%を学習用、30%を検証用データにするよう分割
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size = 0.3, random_state = 666)
# 学習用データでパラメータ推定
model.fit(X_train, y_train)
# 作成したモデルから予測（学習用、検証用モデル使用）
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

In [29]:
```
# matplotlibをインポート
import matplotlib.pyplot as plt
# Jupyterを利用していたら、以下のおまじないを書くとnotebook上に図が表示
%matplotlib inline
# 学習用、検証用それぞれで残差をプロット
plt.scatter(y_train_pred, y_train_pred - y_train, c = 'blue', marker = 'o', label = 'Train Data')
plt.scatter(y_test_pred, y_test_pred - y_test, c = 'lightgreen', marker = 's', label = 'Test Data')
plt.xlabel('Predicted Values')
```

In [29]:
```
# matplotlibをインポート
import matplotlib.pyplot as plt
# Jupyterを利用していたら、以下のおまじないを書くとnotebook上に図が表示
%matplotlib inline
# 学習用、検証用それぞれで残差をプロット
plt.scatter(y_train_pred, y_train_pred - y_train, c = 'blue', marker = 'o', label = 'Train Data')
plt.scatter(y_test_pred, y_test_pred - y_test, c = 'lightgreen', marker = 's', label = 'Test Data')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
# 凡例を左上に表示
plt.legend(loc = 'upper left')
# y = 0に直線を引く
plt.hlines(y = 0, xmin = -10, xmax = 50, lw = 2, color = 'red')
plt.xlim([10, 50])
plt.show()
```



In [30]:
```
# 平均二乗誤差を評価するためのメソッドを呼び出し
from sklearn.metrics import mean_squared_error
# 学習用、検証用データに関して平均二乗誤差を出力
print('MSE Train : %.3f, Test : %.3f' % (mean_squared_error(y_train, y_train_pred), mean_squared_error(y_test, y_test_pred)))
# 学習用、検証用データに関してR^2を出力
print('R^2 Train : %.3f, Test : %.3f' % (model.score(X_train, y_train), model.score(X_test, y_test)))
```
```
MSE Train : 44.983, Test : 40.412
R^2 Train : 0.500, Test : 0.434
```

In [ ]:

残差の可視化を行えた。

2.2. 非線形回帰モデル

基底展開法：最小２乗法や最尤法により推定

多項式関数、ガウス型基底関数、スプライン関数/B スプライン関数

ガウス型基底関数＝正規分布、バンド幅

多項式 (1~9次)

$$\phi_j = x^j$$

ガウス型基底

$$\phi_j(x) = \exp\left\{\frac{(x-\mu_j)^2}{2h_j}\right\}$$

2次元ガウス型基底関数

$$\phi_j(\boldsymbol{x}) = \exp\left\{\frac{(\boldsymbol{x}-\boldsymbol{\mu}_j)^T(\boldsymbol{x}-\boldsymbol{\mu}_j)}{2h_j}\right\}$$

説明変数

$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \cdots, x_{im}) \in \mathbb{R}^m$$

説明変数の数

非線形関数ベクトル

$$\boldsymbol{\phi}(\boldsymbol{x}_i) = (\phi_1(\boldsymbol{x}_i), \phi_2(\boldsymbol{x}_i), \cdots, \phi_k(\boldsymbol{x}_i))^T \in \mathbb{R}^k$$

基底関数の数

非線形関数の計画行列

$$\Phi^{(train)} = (\boldsymbol{\phi}(\boldsymbol{x}_1), \boldsymbol{\phi}(\boldsymbol{x}_2), \cdots, \boldsymbol{\phi}(\boldsymbol{x}_n))^T \in \mathbb{R}^{n\times k}$$

最尤法による予測値

$$\hat{\boldsymbol{y}} = \Phi(\Phi^{(train)T}\Phi^{(train)})^{-1}\Phi^{(train)T}\boldsymbol{y}^{(train)}$$

未学習＝表現力の低いモデル、学習データに対して、

十分小さな誤差が得られていないモデル

⇒対策：表現力の高いモデルを利用する

過学習＝表現力の高いモデル、小さな誤差は得られたけど、

テスト集合誤差との差が大きいモデル

⇒対策：学習データを増やす

⇒対策：不要な基底関数(変数)を削除して表現力を抑止

⇒対策：正規化法を利用して表現力を抑止

基底関数(変数)の数、位置やバンド幅によってモデルの複雑さが変化

正則化法（罰則化法）

リッジ推定量＝縮小推定、最小２乗推定量、L1 ノルムを利用

パラメータをゼロに近づけるよう推定

ラッソ推定量＝スパース推定

いくつかのパラメータを正確にゼロに推定

変数を選択する指標にも使える

正則化項（罰則項）

無い→最小２乗推定量、L2 ノルム利用→リッジ、L1 ノルムを利用→ラッソ

正則化パラメータ

小さく→制約面が大きく、大きく→制約面が小さく

非線形回帰モデルのパラメータ

基底関数の個数、基底関数の位置、基底関数のバンド幅、正則化パラメータ

汎化性能＝汎化誤差が小さいものが良い

ホールドアウト法

クロスバリデーション（交差検証）→CV 値

ハンズオン（実装演習）

データを作成して、非線形回帰を行う

jupyter skl_nonlinear regression (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [2]:
```
#seaborn設定
sns.set()
#背景変更
sns.set_style("darkgrid", {'grid.linestyle': '--'})
#大きさ(スケール変更)
sns.set_context("paper")
```

In [3]:
```
n=100

def true_func(x):
    z = 1-48*x+218*x**2-315*x**3+145*x**4
    return z

def linear_func(x):
    z = x
    return z
```

In [4]:
```
# 真の関数からノイズを伴うデータを生成

# 真の関数からデータ生成
data = np.random.rand(n).astype(np.float32)
data = np.sort(data)
target = true_func(data)

#  ノイズを加える
noise = 0.5 * np.random.randn(n)
target = target  + noise

# ノイズ付きデータを描画

plt.scatter(data, target)
```

jupyter skl_nonlinear regression (unsaved changes)

Logo

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

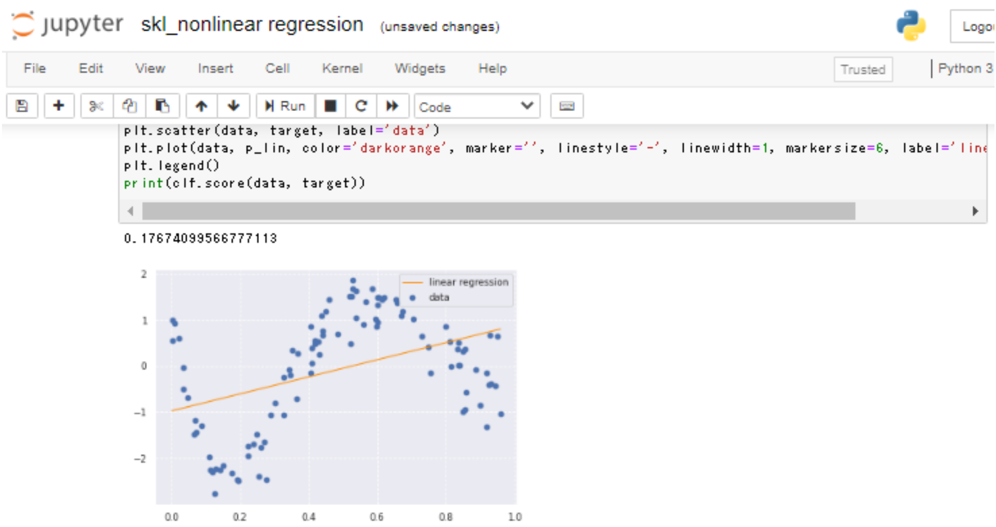
```
target = target  + noise

# ノイズ付きデータを描画

plt.scatter(data, target)

plt.title('NonLinear Regression')
plt.legend(loc=2)
```

```
No handles with labels found to put in legend.
```

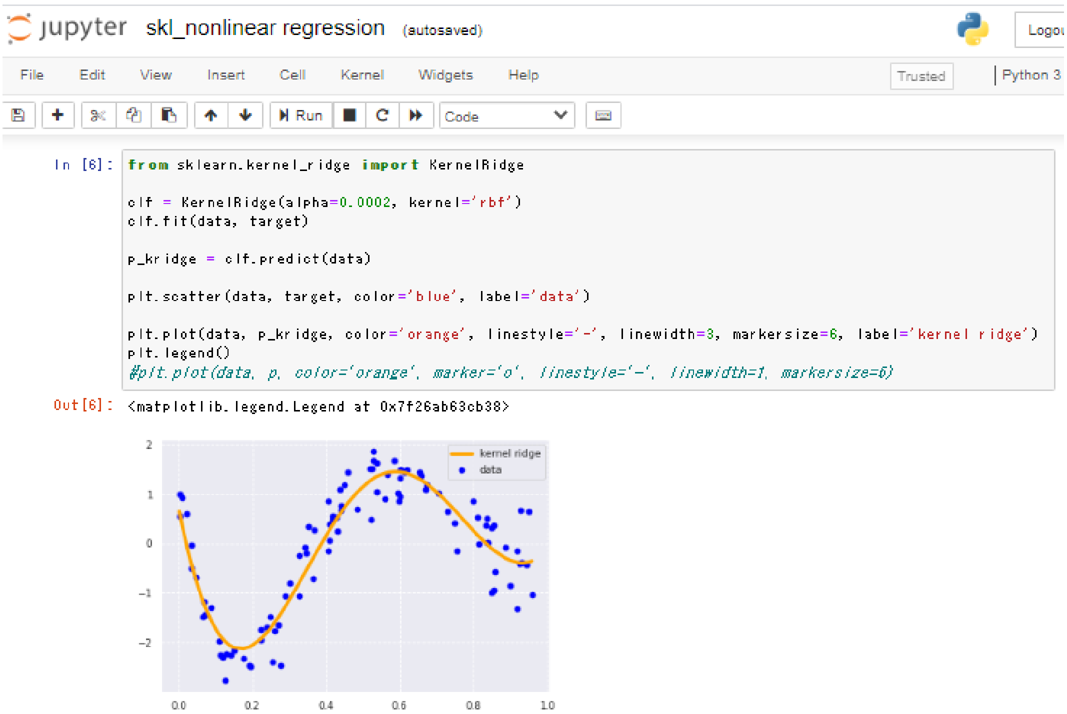
Out[4]: <matplotlib.legend.Legend at 0x7f26b0a30748>

In [5]:
```
from sklearn.linear_model import LinearRegression

clf = LinearRegression()
data = data.reshape(-1,1)
target = target.reshape(-1,1)
clf.fit(data, target)

p_lin = clf.predict(data)

plt.scatter(data, target, label='data')
plt.plot(data, p_lin, color='darkorange', marker='', linestyle='-', linewidth=1, markersize=6, label='line
plt.legend()
```

線形モデルの場合：曲線にはならない、スコア 0.17



カーネルリッジ回帰の場合：

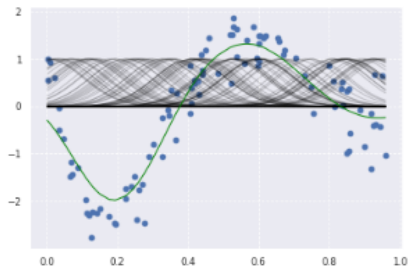


リッジ推定量を用いた線形回帰：スコア 0.84

jupyter skl_nonlinear regression (autosaved)

In [7]:
```
#Ridge

from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Ridge

kx = rbf_kernel(X=data, Y=data, gamma=50)
#KX = rbf_kernel(X, x)

#clf = LinearRegression()
clf = Ridge(alpha=30)
clf.fit(kx, target)

p_ridge = clf.predict(kx)

plt.scatter(data, target, label='data')
for i in range(len(kx)):
    plt.plot(data, kx[i], color='black', linestyle='-', linewidth=1, markersize=3, label='rbf', alpha=0.2)

#plt.plot(data, p, color='green', marker='o', linestyle='-', linewidth=0.1, markersize=3)
plt.plot(data, p_ridge, color='green', linestyle='-', linewidth=1, markersize=3, label='ridge regression')
#plt.legend()

print(clf.score(kx, target))
```

```
0.8414379768929768
```



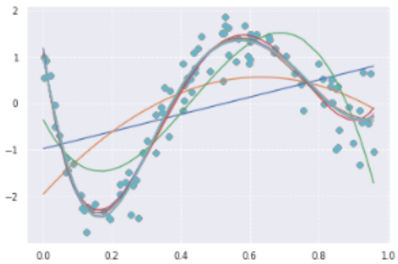
多項式回帰の場合：

jupyter skl_nonlinear regression (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [8]:
```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
```

In [9]:
```python
#PolynomialFeatures(degree=1)

deg = [1,2,3,4,5,6,7,8,9,10]
for d in deg:
    regr = Pipeline([
        ('poly', PolynomialFeatures(degree=d)),
        ('linear', LinearRegression())
    ])
    regr.fit(data, target)
    # make predictions
    p_poly = regr.predict(data)
    # plot regression result
    plt.scatter(data, target, label='data')
    plt.plot(data, p_poly, label='polynomial of degree %d' % (d))
```



ロッソ推定量を用いた線形回帰の場合：スコア 0.0000000000000000222

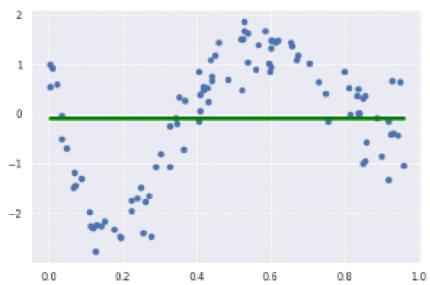
jupyter skl_nonlinear regression (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [10]:

```
#Lasso

from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Lasso

kx = rbf_kernel(X=data, Y=data, gamma=5)
#KX = rbf_kernel(X, x)

#lasso_clf = LinearRegression()
lasso_clf = Lasso(alpha=10000, max_iter=1000)
lasso_clf.fit(kx, target)

p_lasso = lasso_clf.predict(kx)

plt.scatter(data, target)

#plt.plot(data, p, color='green', marker='o', linestyle='-', linewidth=0.1, markersize=3)
plt.plot(data, p_lasso, color='green', linestyle='-', linewidth=3, markersize=3)

print(lasso_clf.score(kx, target))
```

```
2.220446049250313e-16
```



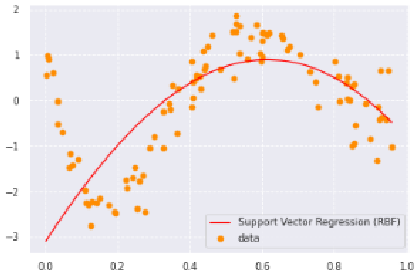
サポートベクターマシーン(RBF カーネル)の場合：

jupyter skl_nonlinear regression (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [11]:
```python
from sklearn import  preprocessing, linear_model, svm

# SVR-rbf
clf_svr = svm.SVR(kernel='rbf', C=1e3, gamma=0.1, epsilon=0.1)
clf_svr.fit(data, target)
y_rbf = clf_svr.fit(data, target).predict(data)

# plot

plt.scatter(data, target, color='darkorange', label='data')
plt.plot(data, y_rbf, color='red', label='Support Vector Regression (RBF)')
plt.legend()
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```



重回帰を用いた場合：

jupyter skl_nonlinear regression (autosaved)

In [12]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size=0.1, random_state=0)
```

In [14]:
```
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint

import os
model_dir = './out/checkpoints/'
if os.path.exists(model_dir) == False:os.mkdir(model_dir)
cb_cp = ModelCheckpoint('./out/checkpoints/weights.{epoch:02d}-{val_loss:.2f}.hdf5', verbose=1, save_weigh
cb_tf  = TensorBoard(log_dir='./out/tensorBoard', histogram_freq=0)

# relu_reg_model を定義
def relu_reg_model():
    model = Sequential()
    model.add(Dense(10, input_dim=1, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(1000, activation='linear'))
    model.add(Dense(1))

    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

In [15]:
```
from keras.models import Sequential
from keras.layers import Input, Dense, Dropout, BatchNormalization
from keras.wrappers.scikit_learn import KerasRegressor

# use data split and fit to run the model
estimator = KerasRegressor(build_fn=relu_reg_model, epochs=100, batch_size=5, verbose=1)

history = estimator.fit(x_train, y_train, callbacks=[cb_cp, cb_tf], validation_data=(x_test, y_test))
```

jupyter skl_nonlinear regression (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

```
# use data split and fit to run the model
estimator = KerasRegressor(build_fn=relu_reg_model, epochs=100, batch_size=5, verbose=1)

history = estimator.fit(x_train, y_train, callbacks=[cb_cp, cb_tf], validation_data=(x_test, y_test))
```

```
Epoch 96/100
17/18 [==========================>..] - ETA: 0s - loss: 0.1850
Epoch 00096: saving model to ./out/checkpoints/weights.96-0.21.hdf5
18/18 [==============================] - 0s 15ms/step - loss: 0.1808 - val_loss: 0.2095
Epoch 97/100
17/18 [==========================>..] - ETA: 0s - loss: 0.1931
Epoch 00097: saving model to ./out/checkpoints/weights.97-0.23.hdf5
18/18 [==============================] - 0s 16ms/step - loss: 0.1887 - val_loss: 0.2256
Epoch 98/100
17/18 [==========================>..] - ETA: 0s - loss: 0.1967
Epoch 00098: saving model to ./out/checkpoints/weights.98-0.22.hdf5
18/18 [==============================] - 0s 15ms/step - loss: 0.1935 - val_loss: 0.2207
Epoch 99/100
17/18 [==========================>..] - ETA: 0s - loss: 0.1981
Epoch 00099: saving model to ./out/checkpoints/weights.99-0.23.hdf5
18/18 [==============================] - 0s 16ms/step - loss: 0.2073 - val_loss: 0.2296
Epoch 100/100
17/18 [==========================>..] - ETA: 0s - loss: 0.2219
Epoch 00100: saving model to ./out/checkpoints/weights.100-0.25.hdf5
18/18 [==============================] - 0s 16ms/step - loss: 0.2197 - val_loss: 0.2503
```

In [16]:
```
y_pred = estimator.predict(x_train)
```

```
18/18 [==============================] - 0s 2ms/step
```

In [17]:
```
plt.title('NonLiner Regressions via DL by ReLU')
plt.plot(data, target, 'o')
plt.plot(data, true_func(data), '.')
plt.plot(x_train, y_pred, "o", label='predicted: deep learning')
#plt.legend(loc=2)
```

jupyter skl_nonlinear regression (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

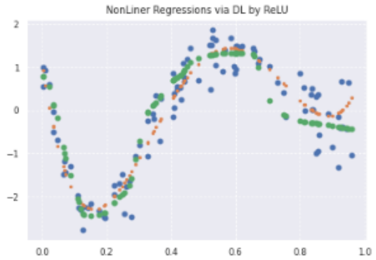
```
Epoch 00100: saving model to ./out/checkpoints/weights.100-0.25.hdf5
18/18 [==============================] - 0s 16ms/step - loss: 0.2197 - val_loss: 0.2503
```

In [16]:
```
y_pred = estimator.predict(x_train)
```

```
18/18 [==============================] - 0s 2ms/step
```

In [17]:
```
plt.title('NonLiner Regressions via DL by ReLU')
plt.plot(data, target, 'o')
plt.plot(data, true_func(data), '.')
plt.plot(x_train, y_pred, "o", label='predicted: deep learning')
#plt.legend(loc=2)
```


Out[17]:
```
[<matplotlib.lines.Line2D at 0x7f25e47b79b0>]
```



In [18]:
```
print(lasso_clf.coef_)
```

```
[-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. -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.
  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.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

2.3.　ロジスティック回帰モデル

分類問題（クラス分類）で利用する。回帰ではない。

説明変数　$\boldsymbol{x} = (x_1, x_2, \cdots, x_m)^T \in \mathbb{R}^m$

目的変数　$y \in \{0, 1\}$　0か1

教師データ

$\{(\boldsymbol{x}_i, y_i); i = 1, ..., n\}$

シグモイド関数：出力は必ずゼロか１。単調増加関数。

パラメータが変わるとシグモイド関数の形が変わる。

a を増加→x=0 付近の曲線の勾配が増加

バイアス変化は段差の位置

シグモイド関数の微分は、シグモイド関数自身で表現できる

求めたい値　シグモイド関数

$$\underbrace{P(Y = 1|\boldsymbol{x})}_{\text{説明変数の実現値が与えられた際にY=1になる確率}} = \sigma(\underbrace{w_0 + w_1 x_1 + \cdots + w_m x_m}_{\text{データのパラメーターに対する線形結合}})$$

表記　$p_i = \sigma(w_0 + w_1 x_{i1} + \cdots + w_m x_{im})$

ロジスティック回帰モデルでは、ベルヌーイ分布を用いる

ベルヌーイ分布のパラメータの推定は、最尤推定

確率変数は独立である＝確率の掛け算

尤度関数を最大化する＝最尤推定

対数を取ると微分の計算が簡単になる

対数尤度関数が最大化＝尤度関数が最大化

尤度関数にマイナスを掛けて最小化（最小２乗法の最小化と合わせる）

勾配降下法

線形回帰モデル（最小２乗法）→MSE のパラメータに関するゼロになる値

ロジスティック回帰モデル（最尤法）→対数尤度関数をパラメータで微分して

ゼロになる値→困難。

パラメータ更新するのに N 個全てのデータ和が必要

N が大きい場合メモリ不足になる。確率的勾配降下法を用いる。

確率的勾配降下法（SGD）

データをランダムに確率的に選んでパラメータを更新

モデルの評価

混同行列

成果率＝(TP+TN)/(TP+FN+FP+TN)

再現率（Recall）＝TP/(TP+FN)

適合率（Precision）　=TP/(TP+FP)

F 値＝再現率と適合率の調和平均

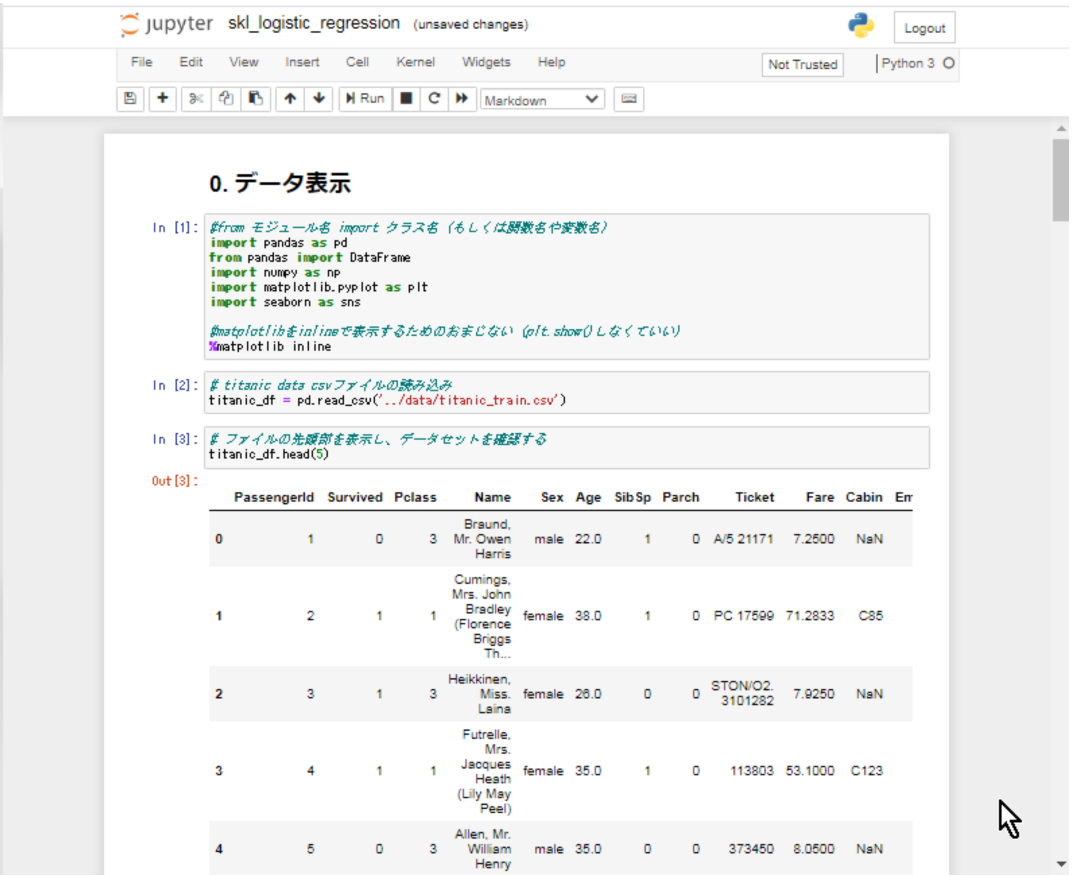
ハンズオン（実装演習）

タイタニックのデータを用いる。

シグモイド関数を作る、同時確率を作る、尤度関数を作る、

尤度関数を最大にするような点 W 重みを付ける、微分してゼロになるところを

勾配降下法を使って求める。



0. データ表示

In [1]:
```python
#from モジュール名 import クラス名（もしくは関数名や変数名）
import pandas as pd
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#matplotlibをinlineで表示するためのおまじない (plt.show()しなくていい)
%matplotlib inline
```

In [2]:
```python
# titanic data csvファイルの読み込み
titanic_df = pd.read_csv('../data/titanic_train.csv')
```

In [3]:
```python
# ファイルの先頭部を表示し、データセットを確認する
titanic_df.head(5)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

jupyter skl_logistic_regression (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3

Run Code

1. ロジスティック回帰

不要なデータの削除・欠損値の補完

In [4]:
```
#予測に不要と考えるからうをドロップ (本当はここの情報もしっかり使うべきだと思っています)
titanic_df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

#一部カラムをドロップしたデータを表示
titanic_df.head()
```

Out[4]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

In [5]:
```
#nullを含んでいる行を表示
titanic_df[titanic_df.isnull().any(1)].head(10)
```

Out[5]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
5	0	3	male	NaN	0	0	8.4583	Q
17	1	2	male	NaN	0	0	13.0000	S
19	1	3	female	NaN	0	0	7.2250	C
26	0	3	male	NaN	0	0	7.2250	C
28	1	3	female	NaN	0	0	7.8792	Q
29	0	3	male	NaN	0	0	7.8958	S
31	1	1	female	NaN	1	0	146.5208	C
32	1	3	female	NaN	0	0	7.7500	Q
36	1	3	male	NaN	0	0	7.2292	C
42	0	3	male	NaN	0	0	7.8958	C

In [ ]:

jupyter skl_logistic_regression (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

19	1	3	female	NaN	0	0	7.2250	C
26	0	3	male	NaN	0	0	7.2250	C
28	1	3	female	NaN	0	0	7.8792	Q
29	0	3	male	NaN	0	0	7.8958	S
31	1	1	female	NaN	1	0	146.5208	C
32	1	3	female	NaN	0	0	7.7500	Q
36	1	3	male	NaN	0	0	7.2292	C
42	0	3	male	NaN	0	0	7.8958	C

In [6]:

```
#Ageカラムのnullを中央値で補完

titanic_df['AgeFill'] = titanic_df['Age'].fillna(titanic_df['Age'].mean())

#再度nullを含んでいる行を表示 (Ageのnullは補完されている)
titanic_df[titanic_df.isnull().any(1)]

#titanic_df.dtypes
```

Out[6]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill
5	0	3	male	NaN	0	0	8.4583	Q	29.699118
17	1	2	male	NaN	0	0	13.0000	S	29.699118
19	1	3	female	NaN	0	0	7.2250	C	29.699118
26	0	3	male	NaN	0	0	7.2250	C	29.699118
28	1	3	female	NaN	0	0	7.8792	Q	29.699118
...	...	...	...	...	...	...	...	...	...
859	0	3	male	NaN	0	0	7.2292	C	29.699118
863	0	3	female	NaN	8	2	69.5500	S	29.699118
868	0	3	male	NaN	0	0	9.5000	S	29.699118
878	0	3	male	NaN	0	0	7.8958	S	29.699118
888	0	3	female	NaN	1	2	23.4500	S	29.699118

179 rows × 9 columns

1. ロジスティック回帰

jupyter skl_logistic_regression (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

1. ロジスティック回帰

実装(チケット価格から生死を判別)

In [7]:
```
#運賃だけのリストを作成
data1 = titanic_df.loc[:, ["Fare"]].values
```

In [8]:
```
#生死フラグのみのリストを作成
label1 =  titanic_df.loc[:,["Survived"]].values
```

In [9]:
```
from sklearn.linear_model import LogisticRegression
```

In [10]:
```
model=LogisticRegression()
```

In [11]:
```
model.fit(data1, label1)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column
-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for e
xample using ravel().
  y = column_or_1d(y, warn=True)
```

Out[11]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [13]:
```
model.predict([[61]])
```

Out[13]:
```
array([0])
```

In [15]:
```
model.predict_proba([[62]])
```

Out[15]:
```
array([[0.49978123, 0.50021877]])
```

In [16]:
```
X_test_value = model.decision_function(data1)
```

In [17]:
```
# # 決定関数値（絶対値が大きいほど識別境界から離れている）
# X_test_value = model.decision_function(X_test)
# # 決定関数値をシグモイド関数で確率に変換
# X_test_prob = normal_sigmoid(X_test_value)
```

jupyter skl_logistic_regression (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [16]:
```
X_test_value = model.decision_function(data1)
```

In [17]:
```
# # 決定関数値（絶対値が大きいほど識別境界から離れている）
# X_test_value = model.decision_function(X_test)
# # 決定関数値をシグモイド関数で確率に変換
# X_test_prob = normal_sigmoid(X_test_value)
```

In [18]:
```
print (model.intercept_)

print (model.coef_)
```

```
[-0.94131796]
[[0.01519666]]
```

In [19]:
```
w_0 = model.intercept_[0]
w_1 = model.coef_[0,0]

# def normal_sigmoid(x):
#     return 1 / (1+np.exp(-x))

def sigmoid(x):
    return 1 / (1+np.exp(-(w_1*x+w_0)))

x_range = np.linspace(-1, 500, 3000)

plt.figure(figsize=(9,5))
#plt.xkcd()
plt.legend(loc=2)


# plt.ylim(-0.1, 1.1)
# plt.xlim(-10, 10)

# plt.plot([-10,10],[0,0], "k", lw=1)
# plt.plot([0,0],[-1,1.5], "k", lw=1)
plt.plot(data1,np.zeros(len(data1)), 'o')
plt.plot(data1, model.predict_proba(data1), 'o')
plt.plot(x_range, sigmoid(x_range), '-')
#plt.plot(x_range, normal_sigmoid(x_range), '-')
#
```

```
No handles with labels found to put in legend.
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x7f8c5518ff28>]
```

jupyter sklearn_logistic_regression Last Checkpoint: 数秒前 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

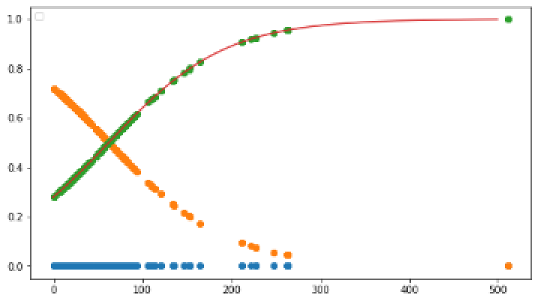
Trusted | Python 3

Run Code

In [19]:
```
w_0 = model.intercept_[0]
w_1 = model.coef_[0,0]

# def normal_sigmoid(x):
#     return 1 / (1+np.exp(-x))

def sigmoid(x):
    return 1 / (1+np.exp(-(w_1*x+w_0)))

x_range = np.linspace(-1, 500, 3000)

plt.figure(figsize=(9,5))
#plt.xkcd()
plt.legend(loc=2)


# plt.ylim(-0.1, 1.1)
# plt.xlim(-10, 10)

# plt.plot([-10, 10], [0, 0], "k", lw=1)
# plt.plot([0, 0], [-1, 1.5], "k", lw=1)
plt.plot(data1,np.zeros(len(data1)), 'o')
plt.plot(data1, model.predict_proba(data1), 'o')
plt.plot(x_range, sigmoid(x_range), '-')
#plt.plot(x_range, normal_sigmoid(x_range), '-')
#
```

```
No handles with labels found to put in legend.
```

Out[19]: [<matplotlib.lines.Line2D at 0x7f8c5518ff28>]



jupyter skl_logistic_regression Last Checkpoint: 1分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

1. ロジスティック回帰

実装(2変数から生死を判別)

In [20]:

```
#AgeFillの欠損値を埋めたので
#titanic_df = titanic_df.drop(['Age'], axis=1)
```

In [21]:

```
titanic_df['Gender'] = titanic_df['Sex'].map({'female': 0, 'male': 1}).astype(int)
```

In [22]:

```
titanic_df.head(3)
```

Out[22]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill	Gender
0	0	3	male	22.0	1	0	7.2500	S	22.0	1
1	1	1	female	38.0	1	0	71.2833	C	38.0	0
2	1	3	female	26.0	0	0	7.9250	S	26.0	0

In [23]:

```
titanic_df['Pclass_Gender'] = titanic_df['Pclass'] + titanic_df['Gender']
```

In [24]:

```
titanic_df.head()
```

Out[24]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill	Gender	Pclass_Gender
0	0	3	male	22.0	1	0	7.2500	S	22.0	1	4
1	1	1	female	38.0	1	0	71.2833	C	38.0	0	1
2	1	3	female	26.0	0	0	7.9250	S	26.0	0	3
3	1	1	female	35.0	1	0	53.1000	S	35.0	0	1
4	0	3	male	35.0	0	0	8.0500	S	35.0	1	4

In [25]:

```
titanic_df = titanic_df.drop(['Pclass', 'Sex', 'Gender','Age'], axis=1)
```

In [26]:

```
titanic_df.head()
```

Out[26]:

	Survived	SibSp	Parch	Fare	Embarked	AgeFill	Pclass_Gender
0	0	1	0	7.2500	S	22.0	4
1	1	1	0	71.2833	C	38.0	1

jupyter skl_logistic_regression Last Checkpoint: 1分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help   Trusted   Python 3

In [25]:
```
titanic_df = titanic_df.drop(['Pclass', 'Sex', 'Gender','Age'], axis=1)
```

In [26]:
```
titanic_df.head()
```

Out[26]:

	Survived	SibSp	Parch	Fare	Embarked	AgeFill	Pclass_Gender
0	0	1	0	7.2500	S	22.0	4
1	1	1	0	71.2833	C	38.0	1
2	1	0	0	7.9250	S	26.0	3
3	1	1	0	53.1000	S	35.0	1
4	0	0	0	8.0500	S	35.0	4

In [27]:
```
# 重要だよ！！！
# 境界線の式
#   w_1・x + w_2・y + w_0 = 0
#   ⇒ y = (-w_1・x - w_0) / w_2

# # 境界線 プロット
# plt.plot([-2,2], map(lambda x: (-w_1 * x - w_0)/w_2, [-2,2]))

# # データを重ねる
# plt.scatter(X_train_std[y_train==0, 0], X_train_std[y_train==0, 1], c='red', marker='x', label='train 0')
# plt.scatter(X_train_std[y_train==1, 0], X_train_std[y_train==1, 1], c='blue', marker='x', label='train 1')
# plt.scatter(X_test_std[y_test==0, 0], X_test_std[y_test==0, 1], c='red', marker='o', s=60, label='test 0')
# plt.scatter(X_test_std[y_test==1, 0], X_test_std[y_test==1, 1], c='blue', marker='o', s=60, label='test 1')
```

In [28]:
```
np.random.seed = 0

xmin, xmax = -5, 85
ymin, ymax = 0.5, 4.5

index_survived = titanic_df[titanic_df["Survived"]==0].index
index_notsurvived = titanic_df[titanic_df["Survived"]==1].index

from matplotlib.colors import ListedColormap
fig, ax = plt.subplots()
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
sc = ax.scatter(titanic_df.loc[index_survived, 'AgeFill'],
                titanic_df.loc[index_survived, 'Pclass_Gender']+(np.random.rand(len(index_survived))-0.5)*0.1,
                color='r', label='Not Survived', alpha=0.3)
sc = ax.scatter(titanic_df.loc[index_notsurvived, 'AgeFill'],
                titanic_df.loc[index_notsurvived, 'Pclass_Gender']+(np.random.rand(len(index_notsurvived))-0.5)*0.1,
                color='b', label='Survived', alpha=0.3)
ax.set_xlabel('AgeFill')
```

jupyter skl_logistic_regression Last Checkpoint: 2分前 (unsaved changes) Logout

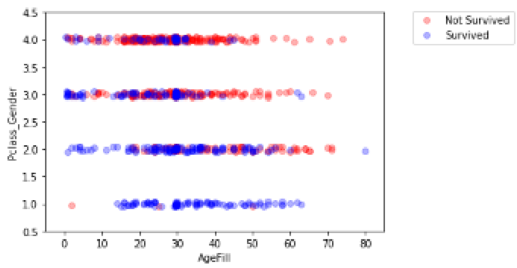
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```python
# plt.scatter(X_train_std[y_train==1, 0], X_train_std[y_train==1, 1], c='blue', marker='x', label='train 1')
# plt.scatter(X_test_std[y_test==0, 0], X_test_std[y_test==0, 1], c='red', marker='o', s=60, label='test 0')
# plt.scatter(X_test_std[y_test==1, 0], X_test_std[y_test==1, 1], c='blue', marker='o', s=60, label='test 1')
```

In [28]:

```python
np.random.seed = 0

xmin, xmax = -5, 85
ymin, ymax = 0.5, 4.5

index_survived = titanic_df[titanic_df["Survived"]==0].index
index_notsurvived = titanic_df[titanic_df["Survived"]==1].index

from matplotlib.colors import ListedColormap
fig, ax = plt.subplots()
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
sc = ax.scatter(titanic_df.loc[index_survived, 'AgeFill'],
                titanic_df.loc[index_survived, 'Pclass_Gender']+(np.random.rand(len(index_survived))-0.5)*0.1,
                color='r', label='Not Survived', alpha=0.3)
sc = ax.scatter(titanic_df.loc[index_notsurvived, 'AgeFill'],
                titanic_df.loc[index_notsurvived, 'Pclass_Gender']+(np.random.rand(len(index_notsurvived))-0.5)*0.1,
                color='b', label='Survived', alpha=0.3)
ax.set_xlabel('AgeFill')
ax.set_ylabel('Pclass_Gender')
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
ax.legend(bbox_to_anchor=(1.4, 1.03))
```

Out[28]: <matplotlib.legend.Legend at 0x7f8c50a2de48>



In [29]:

```python
#運賃だけのリストを作成
data2 = titanic_df.loc[:, ["AgeFill", "Pclass_Gender"]].values
```

jupyter skl_logistic_regression Last Checkpoint: 7分前 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Markdown

```
data2 = titanic_df.loc[:, ["AgeFill", "Pclass_Gender"]].values
```

In [30]:
```
data2
```

Out[30]:
```
array([[22.        ,  4.        ],
       [38.        ,  1.        ],
       [26.        ,  3.        ],
       ...,
       [29.69911765,  3.        ],
       [26.        ,  2.        ],
       [32.        ,  4.        ]])
```

In [31]:
```
#生死フラグのみのリストを作成
label2 =  titanic_df.loc[:,["Survived"]].values
```

In [32]:
```
model2 = LogisticRegression()
```

In [33]:
```
model2.fit(data2, label2)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[33]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [34]:
```
model2.predict([[10,1]])
```

Out[34]:
```
array([1])
```

In [35]:
```
model2.predict_proba([[10,1]])
```

Out[35]:
```
array([[0.03754749, 0.96245251]])
```

In [36]:
```
titanic_df.head(3)
```

Out[36]:

	Survived	SibSp	Parch	Fare	Embarked	AgeFill	Pclass_Gender
0	0	1	0	7.2500	S	22.0	4
1	1	1	0	71.2833	C	38.0	1
2	1	0	0	7.9250	S	26.0	3

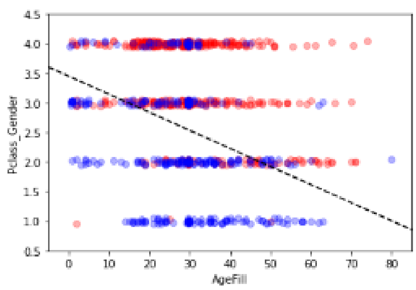
jupyter skl_logistic_regression Last Checkpoint: 7分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted

Run Markdown

In [37]:
```python
h = 0.02
xmin, xmax = -5, 85
ymin, ymax = 0.5, 4.5
xx, yy = np.meshgrid(np.arange(xmin, xmax, h), np.arange(ymin, ymax, h))
Z = model2.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
Z = Z.reshape(xx.shape)

fig, ax = plt.subplots()
levels = np.linspace(0, 1.0)
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
#contour = ax.contourf(xx, yy, Z, cmap=cm, levels=levels, alpha=0.5)

sc = ax.scatter(titanic_df.loc[index_survived, 'AgeFill'],
                titanic_df.loc[index_survived, 'Pclass_Gender']+(np.random.rand(len(index_survived))-0.5)*0.1,
                color='r', label='Not Survived', alpha=0.3)
sc = ax.scatter(titanic_df.loc[index_notsurvived, 'AgeFill'],
                titanic_df.loc[index_notsurvived, 'Pclass_Gender']+(np.random.rand(len(index_notsurvived))-0.5)*0.1,
                color='b', label='Survived', alpha=0.3)

ax.set_xlabel('AgeFill')
ax.set_ylabel('Pclass_Gender')
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
#fig.colorbar(contour)

x1 = xmin
x2 = xmax
y1 = -1*(model2.intercept_[0]+model2.coef_[0][0]*xmin)/model2.coef_[0][1]
y2 = -1*(model2.intercept_[0]+model2.coef_[0][0]*xmax)/model2.coef_[0][1]
ax.plot([x1, x2] ,[y1, y2], 'k--')
```

Out[37]: [<matplotlib.lines.Line2D at 0x7f8c509e35f8>]



jupyter skl_logistic_regression Last Checkpoint: 8分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

2. モデル評価

混同行列とクロスバリデーション

In [38]:
```
from sklearn.model_selection import train_test_split
```

In [39]:
```
traindata1, testdata1, trainlabel1, testlabel1 = train_test_split(data1, label1, test_size=0.2)
traindata1.shape
trainlabel1.shape
```

Out[39]: (712, 1)

In [40]:
```
traindata2, testdata2, trainlabel2, testlabel2 = train_test_split(data2, label2, test_size=0.2)
traindata2.shape
trainlabel2.shape
#本来は同じデータセットを分割しなければいけない。(簡易的に別々に分割している。)
```

Out[40]: (712, 1)

In [41]:
```
data = titanic_df.loc[:, ].values
label =  titanic_df.loc[:,["Survived"]].values
traindata, testdata, trainlabel, testlabel = train_test_split(data, label, test_size=0.2)
traindata.shape
trainlabel.shape
```

Out[41]: (712, 1)

In [42]:
```
eval_model1=LogisticRegression()
eval_model2=LogisticRegression()
#eval_model=LogisticRegression()
```

In [43]:
```
predictor_eval1=eval_model1.fit(traindata1, trainlabel1).predict(testdata1)
predictor_eval2=eval_model2.fit(traindata2, trainlabel2).predict(testdata2)
#predictor_eval=eval_model.fit(traindata, trainlabel).predict(testdata)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

jupyter skl_logistic_regression Last Checkpoint: 9分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help   Trusted | Python 3

```
1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [44]:
```python
eval_model1.score(traindata1, trainlabel1)
```
Out[44]:
```
0.6685393258426966
```

In [45]:
```python
eval_model1.score(testdata1,testlabel1)
```
Out[45]:
```
0.6424581005586593
```

In [46]:
```python
eval_model2.score(traindata2, trainlabel2)
```
Out[46]:
```
0.776685393258427
```

In [47]:
```python
eval_model2.score(testdata2,testlabel2)
```
Out[47]:
```
0.770949720670391
```

In [48]:
```python
from sklearn import metrics
print(metrics.classification_report(testlabel1, predictor_eval1))
print(metrics.classification_report(testlabel2, predictor_eval2))
```
```
              precision    recall  f1-score   support

           0       0.62      0.95      0.75       103
           1       0.77      0.22      0.35        76

    accuracy                           0.64       179
   macro avg       0.70      0.59      0.55       179
weighted avg       0.69      0.64      0.58       179

              precision    recall  f1-score   support

           0       0.79      0.86      0.82       112
           1       0.72      0.63      0.67        67

    accuracy                           0.77       179
   macro avg       0.76      0.74      0.75       179
weighted avg       0.77      0.77      0.77       179
```

In [49]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix1=confusion_matrix(testlabel1, predictor_eval1)
confusion_matrix2=confusion_matrix(testlabel2, predictor_eval2)
```

jupyter skl_logistic_regression Last Checkpoint: 9分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help   Trusted | Python 3

In [50]:
```python
confusion_matrix1
```
Out[50]:
```
array([[98,  5],
       [59, 17]])
```

In [51]:
```python
confusion_matrix2
```
Out[51]:
```
array([[96, 16],
       [25, 42]])
```

In [52]:
```python
fig = plt.figure(figsize = (7,7))
#plt.title(title)
sns.heatmap(
    confusion_matrix1,
    vmin=None,
    vmax=None,
    cmap="Blues",
```

jupyter skl_logistic_regression Last Checkpoint: 8分前 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

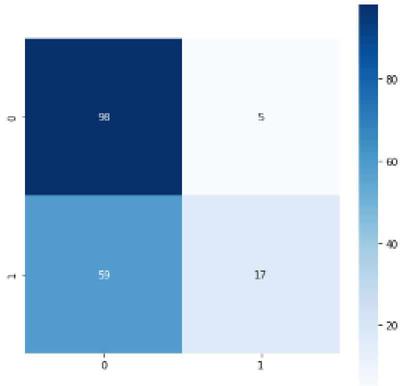
Trusted | Python 3

Run Code

Out [51]:
```
array([[96, 16],
       [25, 42]])
```

In [52]:
```
fig = plt.figure(figsize = (7,7))
#plt.title(title)
sns.heatmap(
    confusion_matrix1,
    vmin=None,
    vmax=None,
    cmap="Blues",
    center=None,
    robust=False,
    annot=True, fmt='.2g',
    annot_kws=None,
    linewidths=0,
    linecolor='white',
    cbar=True,
    cbar_kws=None,
    cbar_ax=None,
    square=True, ax=None,
    #xticklabels=columns,
    #yticklabels=columns,
    mask=None)
```

Out [52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c4fb82e80>
```



jupyter skl_logistic_regression Last Checkpoint: 9分前 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

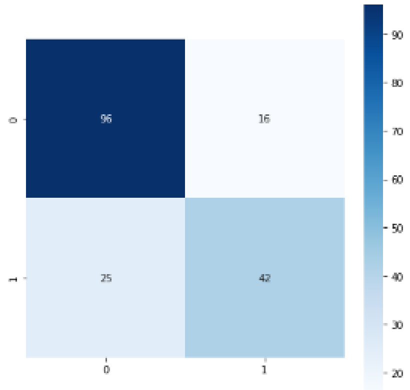
Run Code

In [53]:

```
fig = plt.figure(figsize = (7,7))
#plt.title(title)
sns.heatmap(
    confusion_matrix2,
    vmin=None,
    vmax=None,
    cmap="Blues",
    center=None,
    robust=False,
    annot=True, fmt='.2g',
    annot_kws=None,
    linewidths=0,
    linecolor='white',
    cbar=True,
    cbar_kws=None,
    cbar_ax=None,
    square=True, ax=None,
    #xticklabels=columns,
    #yticklabels=columns,
    mask=None)
```

Out[53]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8c4fb00048>
```

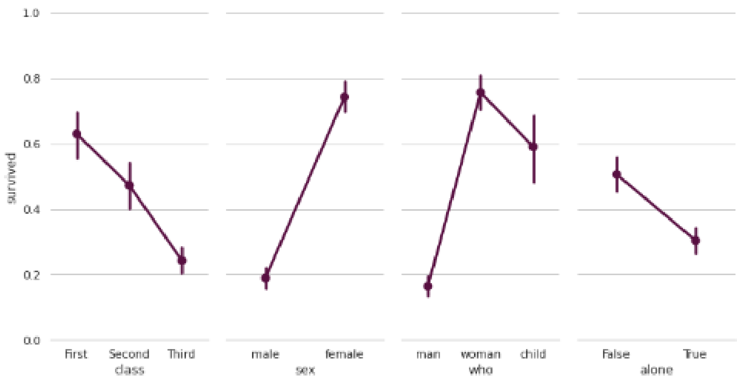


jupyter skl_logistic_regression Last Checkpoint: 9分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python

In [54]:
```
#Paired categorical plots

import seaborn as sns
sns.set(style="whitegrid")

# Load the example Titanic dataset
titanic = sns.load_dataset("titanic")

# Set up a grid to plot survival probability against several variables
g = sns.PairGrid(titanic, y_vars="survived",
                 x_vars=["class", "sex", "who", "alone"],
                 size=5, aspect=.5)

# Draw a seaborn pointplot onto each Axes
g.map(sns.pointplot, color=sns.xkcd_rgb["plum"])
g.set(ylim=(0, 1))
sns.despine(fig=g.fig, left=True)

plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/seaborn/axisgrid.py:1259: UserWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(UserWarning(msg))
```



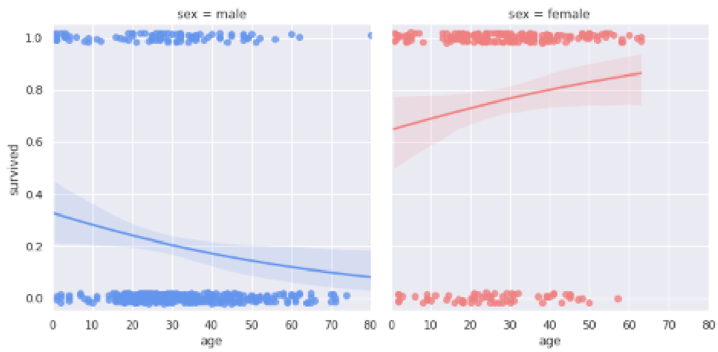
jupyter skl_logistic_regression Last Checkpoint: 9分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

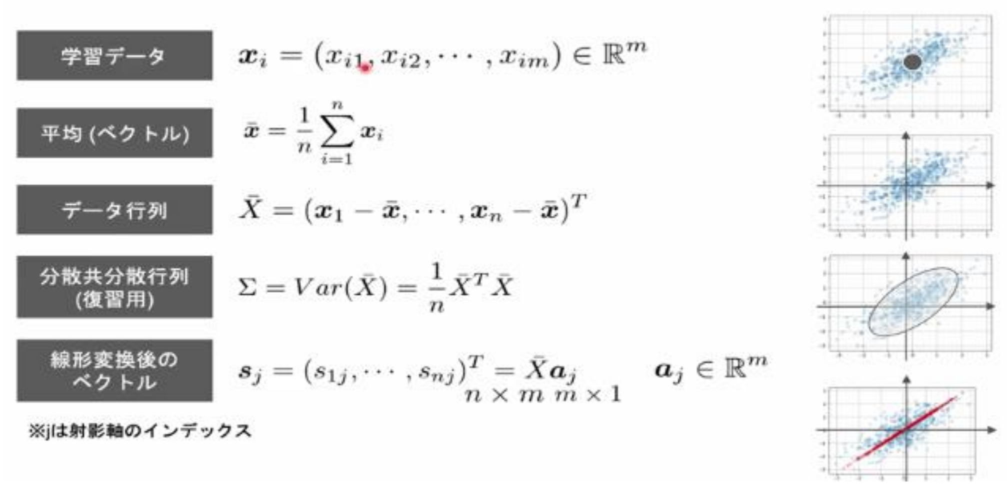
In [55]:

```python
#Faceted logistic regression

import seaborn as sns
sns.set(style="darkgrid")

# Load the example titanic dataset
df = sns.load_dataset("titanic")

# Make a custom palette with gendered colors
pal = dict(male="#6495ED", female="#F08080")

# Show the survival probability as a function of age and sex
g = sns.lmplot(x="age", y="survived", col="sex", hue="sex", data=df,
               palette=pal, y_jitter=.02, logistic=True)
g.set(xlim=(0, 80), ylim=(-.05, 1.05))
plt.show()
```

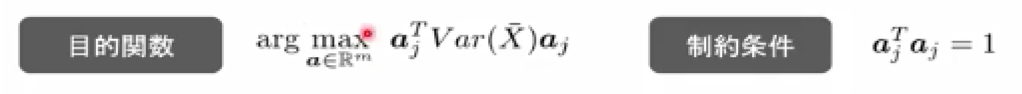


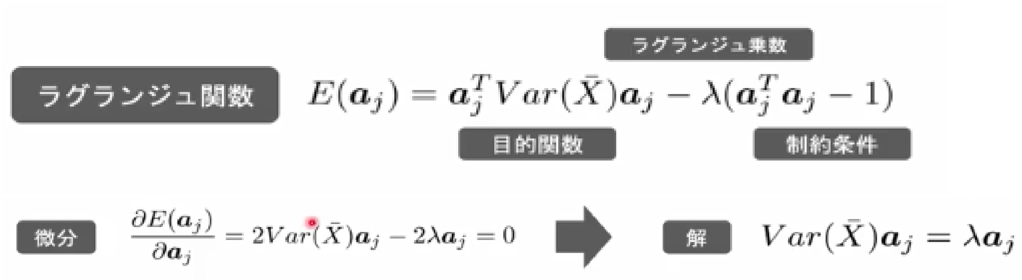
2.4. 主成分分析

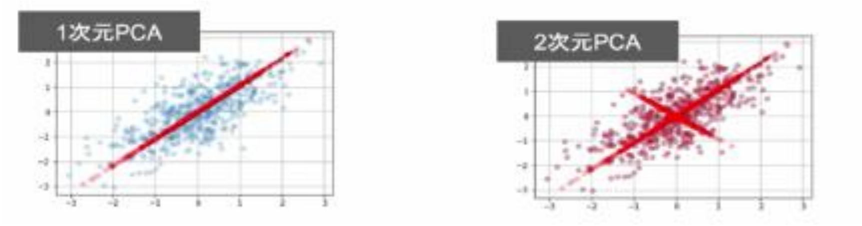
次元の圧縮、情報の損失をなるべく小さくする



線形変換後の変数の分散が最大となる放射軸







分散共分散行列を計算、固有値問題を解く

寄与率：圧縮した結果、情報がどれだけロスしたか。

第 k 主成分の分散の全文さんに対する割合

累積寄与率：第 1-k 主成分まで圧縮した際の情報損失量の割合

$$c_k = \frac{\lambda_k}{\sum_{i=1}^{m} \lambda_i}$$

第k主成分の分散

主成分の総分散

$$r_k = \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{i=1}^{m} \lambda_i}$$

第1~k主成分の分散

主成分の総分散

オートエンコーダも次元圧縮に使える場合がある。

ハンズオン（実装演習）

乳がんデータを用いる。

jupyter skl_pca (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Markdown

In [1]:
```
#https://ohke.hateblo.jp/entry/2017/08/11/230000を参考に利用しています。
```

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
cancer_df = pd.read_csv('../data/cancer.csv')
```

In [4]:
```
print('cancer df shape: {}'.format(cancer_df.shape))
```

```
cancer df shape: (569, 33)
```

In [5]:
```
cancer_df
```

Out[5]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	conc
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	
...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	

569 rows × 33 columns

jupyter skl_pca (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Code

In [6]:
```python
cancer_df.drop('Unnamed: 32', axis=1, inplace=True)
cancer_df
```

Out[6]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	conc
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	
...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	

569 rows × 32 columns

・diagnosis: 診断結果 (良性がB / 悪性がM) ・説明変数は3列以降、目的変数を2列目としロジスティック回帰で分類

In [7]:
```python
# 目的変数の抽出
y = cancer_df.diagnosis.apply(lambda d: 1 if d == 'M' else 0)
```

In [8]:
```python
# 説明変数の抽出
X = cancer_df.loc[:, 'radius_mean':]
```

In [9]:
```python
# 学習用とテスト用でデータを分離
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# 標準化
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ロジスティック回帰で学習
```

・diagnosis: 診断結果 (良性がB / 悪性がM) ・説明変数は3列以降、目的変数を2列目としロジスティック回帰で分類

```
In [7]: # 目的変数の抽出
y = cancer_df.diagnosis.apply(lambda d: 1 if d == 'M' else 0)
```

```
In [8]: # 説明変数の抽出
X = cancer_df.loc[:, 'radius_mean':]
```

```
In [9]: # 学習用とテスト用でデータを分離
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# 標準化
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ロジスティック回帰で学習
logistic = LogisticRegressionCV(cv=10, random_state=0)
logistic.fit(X_train_scaled, y_train)

# 検証
print('Train score: {:.3f}'.format(logistic.score(X_train_scaled, y_train)))
print('Test score: {:.3f}'.format(logistic.score(X_test_scaled, y_test)))
print('Confustion matrix:¥n{}'.format(confusion_matrix(y_true=y_test, y_pred=logistic.predict(X_test_scaled))))
```

```
atus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

Train score: 0.988
Test score: 0.972
Confustion matrix:
[[89  1]
 [ 3 50]]

/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (st
atus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

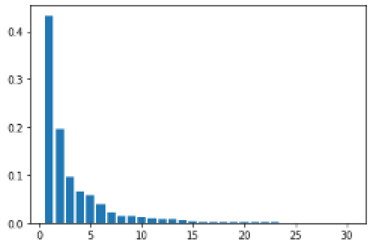
・検証スコア97%で分類できることを確認

jupyter skl_pca (autosaved)

File Edit View Insert Cell Kernel Widgets Help   Trusted   Python 3

```
In [10]: pca = PCA(n_components=30)
pca.fit(X_train_scaled)
plt.bar([n for n in range(1, len(pca.explained_variance_ratio_)+1)], pca.explained_variance_ratio_)
```

Out[10]: <BarContainer object of 30 artists>

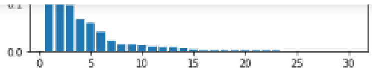



```
In [11]: # PCA
# 次元数2まで圧縮
pca = PCA(n_components=2)
```

上位２位で、６５％程度の情報量となる。

jupyter skl_pca (autosaved)

File Edit View Insert Cell Kernel Widgets Help — Trusted — Python 3

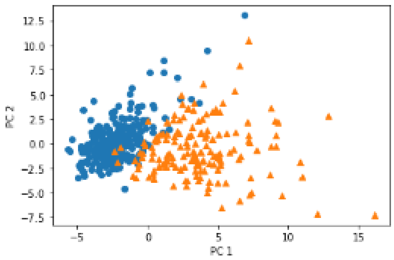


In [11]:

```python
# PCA
# 次元数2まで圧縮
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
print('X_train_pca shape: {}'.format(X_train_pca.shape))
# X_train_pca shape: (426, 2)

# 寄与率
print('explained variance ratio: {}'.format(pca.explained_variance_ratio_))
# explained variance ratio: [ 0.43315126  0.19586506]

# 散布図にプロット
temp = pd.DataFrame(X_train_pca)
temp['Outcome'] = y_train.values
b = temp[temp['Outcome'] == 0]
m = temp[temp['Outcome'] == 1]
plt.scatter(x=b[0], y=b[1], marker='o') # 良性は○でマーク
plt.scatter(x=m[0], y=m[1], marker='^') # 悪性は△でマーク
plt.xlabel('PC 1') # 第1主成分をx軸
plt.ylabel('PC 2') # 第2主成分をy軸
```

```
X_train_pca shape: (426, 2)
explained variance ratio: [0.43315126 0.19586506]
```

Out[11]: Text(0, 0.5, 'PC 2')



2.5. アルゴリズム

・k 近傍法（kNN）

　k はパラメータ、増えると過学習に近くなる

・k-平均法（k-means）クラスタリング

　k はパラメータ、クラスタの数

　クラスタの再割りあて、中心の更新を繰り返す。

　初期値の決め方で変わる。k-means++

ハンズオン（実装演習）

jupyter np_knn (unsaved changes)

k近傍法

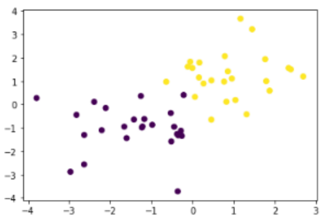
In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

訓練データ生成

In [2]:
```
def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 1
    x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
    x_train = np.concatenate([x0, x1])
    y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
    return x_train, y_train
```

In [3]:
```
X_train, ys_train = gen_data()
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x7fb02212fa58>
```



jupyter np_knn (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help   Not Trusted   Python 3

学習

陽に訓練ステップはない

予測

予測するデータ点との、距離が最も近い*k*個の、訓練データのラベルの最頻値を割り当てる

In [4]:
```
def distance(x1, x2):
    return np.sum((x1 - x2)**2, axis=1)

def knc_predict(n_neighbors, x_train, y_train, X_test):
    y_pred = np.empty(len(X_test), dtype=y_train.dtype)
    for i, x in enumerate(X_test):
        distances = distance(x, X_train)
        nearest_index = distances.argsort()[:n_neighbors]
        mode, _ = stats.mode(y_train[nearest_index])
        y_pred[i] = mode
    return y_pred

def plt_resut(x_train, y_train, y_pred):
    xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
    xx = np.array([xx0, xx1]).reshape(2, -1).T
    plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
    plt.contourf(xx0, xx1, y_pred.reshape(100, 100).astype(dtype=np.float), alpha=0.2, levels=np.linspace(0, 1, 3))
```

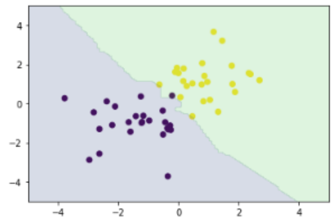
In [5]:
```
n_neighbors = 3

xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
X_test = np.array([xx0, xx1]).reshape(2, -1).T

y_pred = knc_predict(n_neighbors, X_train, ys_train, X_test)
plt_resut(X_train, ys_train, y_pred)
```

jupyter np_knn (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help   Not Trusted   Python 3

```
    plt.contourf(xx0, xx1, y_pred.reshape(100, 100).astype(dtype=np.float), alpha=0.2, levels=np.linspace(0, 1, 3))
```

In [5]:
```
n_neighbors = 3

xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
X_test = np.array([xx0, xx1]).reshape(2, -1).T

y_pred = knc_predict(n_neighbors, X_train, ys_train, X_test)
plt_resut(X_train, ys_train, y_pred)
```

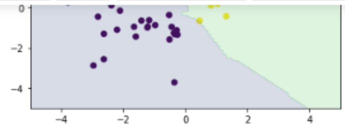
jupyter np_knn (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

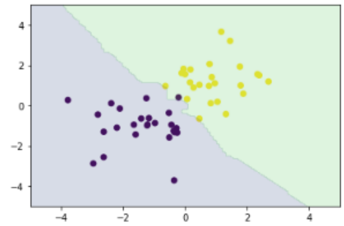
Not Trusted | Python 3

Run Code



numpy実装

In [6]:
```
from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X_train, ys_train)
plt_resut(X_train, ys_train, knc.predict(X_test))
```



2.6.　サポートベクターマシン

正負で 2 値分類

マージンが最大となる線形判別関数

目的関数

$$\max_{\boldsymbol{w},b}\left[\min_{i}\frac{t_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)}{||\boldsymbol{w}||}\right]$$

マージンが最大となる直線(パラメータ)を探す

$$\max_{\boldsymbol{w},b}\frac{1}{||\boldsymbol{w}||}$$

マージンで正規化すると
成り立つ

マージン 1/||w||の最大化のために。

最小化問題、二次計画問題

SVMの主問題

・主問題の目的関数と制約条件

目的関数　$$\min_{\boldsymbol{w},b}\frac{1}{2}||\boldsymbol{w}||^2$$

制約条件　$$t_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)\geq 1 \qquad (i=1,2,...,n)$$

双対問題：制約条件のもと別の等価な関数に置き換えて解決

・ラグランジュ未定乗数法：問題を解くための手法

・KKT 条件 Karush-Kuhn-Tucker 条件：問題において最適解が満たす条件

補問題：ラグランジュ未定乗数法を用いて定義したラグランジュ関数を最大化する

主問題の最適解と双対問題の最適解は一対一対応

ハードマージン SVM：クラスが完全に線形分離可能、マージン最大化を満たす

ソフトマージン SVM：線形分離できない場合、ノイズを含む

制約条件が違う、C が小さい時：誤差許容

非線形分離、非線形 SVM：線形分離できない場合

特徴空間に写像し、その空間で線形分離

・カーネルトリック：K(xi,xj)

目的関数は以下のように変わる

$$\max_{\boldsymbol{a}} \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j t_i t_j \underbrace{\phi(\boldsymbol{x}_i)^{\mathrm{T}} \phi(\boldsymbol{x}_j)}_{\text{ここのみ変化}}$$

・高次元ベクトルの内積をスカラー関数で表す

・特徴空間が高次元でも計算コストが少ない

非線形カーネル：非線形分離できる

・放射基盤関数カーネル（RFB カーネル、ガウシアンカーネル）

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right)$$

σ ：分布データの分散

多項式カーネル：$K(xi,xj)=(xixj+C)^d$

d ：入力データ次元、C：重みパラメータ

ハンズオン（実装演習）

jupyter np_svm (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

サポートベクターマシン(SVM)

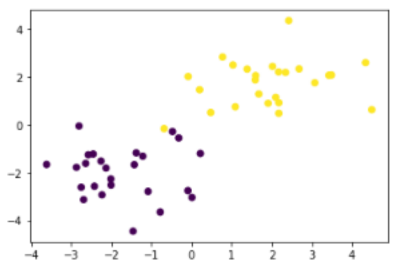
In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

訓練データ生成① （線形分離可能）

In [2]:
```
def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 2.
    x1 = np.random.normal(size=50).reshape(-1, 2) + 2.
    X_train = np.concatenate([x0, x1])
    ys_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
    return X_train, ys_train
```

In [3]:
```
X_train, ys_train = gen_data()
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
```

Out[3]: <matplotlib.collections.PathCollection at 0x7f6c47f4f860>



jupyter np_svm (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python

学習

特徴空間上で線形なモデル$y(\boldsymbol{x}) = \boldsymbol{w}\phi(\boldsymbol{x}) + b$を用い、その正負によって2値分類を行うことを考える。

サポートベクターマシンではマージンの最大化を行うが、それは結局以下の最適化問題を解くことと同じである。

ただし、訓練データを$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^{\mathrm{T}}, \boldsymbol{t} = [t_1, t_2, \ldots, t_n]^{\mathrm{T}} (t_i = \{-1, +1\})$とする。

$\min_{\boldsymbol{w},b} \quad \frac{1}{2}||\boldsymbol{w}||^2$

subject to $\quad t_i(\boldsymbol{w}\phi(\boldsymbol{x}_i) + b) \geq 1 \quad (i = 1, 2, \cdots, n)$

ラグランジュ乗数法を使うと、上の最適化問題はラグランジュ乗数$\boldsymbol{a} (\geq 0)$を用いて、以下の目的関数を最小化する問題となる。

$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{n} a_i t_i (\boldsymbol{w}\phi(\boldsymbol{x}_i) + b - 1) \qquad \cdots (1)$

目的関数が最小となるのは、$\boldsymbol{w}, b$に関して偏微分した値が0となるときなので、

$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} a_i t_i \phi(\boldsymbol{x}_i) = \boldsymbol{0}$

$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} a_i t_i = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{t} = 0$

これを式(1) に代入することで、最適化問題は結局以下の目的関数の最大化となる。

$$\begin{aligned}\tilde{L}(\boldsymbol{a}) &= \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j) \\ &= \boldsymbol{a}^{\mathrm{T}}\boldsymbol{1} - \frac{1}{2}\boldsymbol{a}^{\mathrm{T}} H \boldsymbol{a}\end{aligned}$$

ただし、行列Hのi行j列成分は$H_{ij} = t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j) = t_i t_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$である。また制約条件は、$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t} = 0 (\frac{1}{2}||\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t}||^2 = 0)$である。

jupyter np_svm (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

ただし、行列Hのi行j列成分は$H_{ij} = t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j) = t_i t_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$である。また制約条件は、$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t} = 0 (\frac{1}{2}||\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t}||^2 = 0)$である。

この最適化問題を最急降下法で解く。目的関数と制約条件を$\boldsymbol{a}$で微分すると、

$\frac{d\tilde{L}}{d\boldsymbol{a}} = \boldsymbol{1} - H\boldsymbol{a}$

$\frac{d}{d\boldsymbol{a}}(\frac{1}{2}||\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t}||^2) = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t})\boldsymbol{t}$

なので、$\boldsymbol{a}$を以下の二式で更新する。

$\boldsymbol{a} \leftarrow \boldsymbol{a} + \eta_1(\boldsymbol{1} - H\boldsymbol{a})$

$\boldsymbol{a} \leftarrow \boldsymbol{a} - \eta_2(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t})\boldsymbol{t}$

In [4]:

```
t = np.where(ys_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

eta1 = 0.01
eta2 = 0.001
n_iter = 500

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.where(a > 0, a, 0)
```

jupyter np_svm (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

予測

新しいデータ点$\boldsymbol{x}$に対しては、$y(\boldsymbol{x}) = \boldsymbol{w}\phi(\boldsymbol{x}) + b = \sum_{i=1}^{n} a_i t_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b$の正負によって分類する。

ここで、最適化の結果得られた$a_i (i = 1, 2, \ldots, n)$の中で$a_i = 0$に対応するデータ点は予測に影響を与えないので、$a_i > 0$に対応するデータ点（サポートベクトル）のみ保持しておく。bはサポートベクトルのインデックスの集合をSとすると、$b = \frac{1}{S} \sum_{s \in S} \left(t_s - \sum_{i=1}^{n} a_i t_i k(\boldsymbol{x}_i, \boldsymbol{x}_s) \right)$によって求める。

In [5]:

```
index = a > 1e-6
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()
```

In [6]:

```
xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * sv.dot(X_test[i])
y_pred = np.sign(y_project)
```

In [7]:

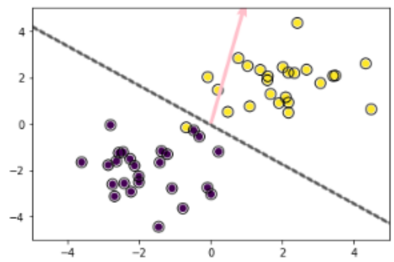
```
# 訓練データを可視化
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
                    s=100, facecolors='none', edgecolors='k')
# 領域を可視化
#plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
                     levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

jupyter np_svm (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
# マージンと決定境界を可視化
plt.quiver(0, 0, 0.1, 0.35, width=0.01, scale=1, color='pink')
```

Out[7]: <matplotlib.quiver.Quiver at 0x7f6c47a449b0>



jupyter np_svm (unsaved changes)

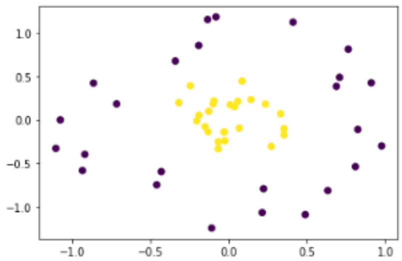
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

訓練データ生成②（線形分離不可能）

In [8]:
```
factor = .2
n_samples = 50
linspace = np.linspace(0, 2 * np.pi, n_samples // 2 + 1)[:-1]
outer_circ_x = np.cos(linspace)
outer_circ_y = np.sin(linspace)
inner_circ_x = outer_circ_x * factor
inner_circ_y = outer_circ_y * factor

X = np.vstack((np.append(outer_circ_x, inner_circ_x),
               np.append(outer_circ_y, inner_circ_y))).T
y = np.hstack([np.zeros(n_samples // 2, dtype=np.intp),
               np.ones(n_samples // 2, dtype=np.intp)])
X += np.random.normal(scale=0.15, size=X.shape)
x_train = X
y_train = y
```

In [9]:
```
plt.scatter(x_train[:,0], x_train[:,1], c=y_train)
```

Out[9]: <matplotlib.collections.PathCollection at 0x7f6c477a0160>



jupyter np_svm (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

学習

元のデータ空間では線形分離は出来ないが、特徴空間上で線形分離することを考える。

今回はカーネルとしてRBFカーネル（ガウシアンカーネル）を利用する。

In [10]:
```
def rbf(u, v):
        sigma = 0.8
        return np.exp(-0.5 * ((u - v)**2).sum() / sigma**2)

X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# RBFカーネル
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        K[i, j] = rbf(X_train[i], X_train[j])

eta1 = 0.01
eta2 = 0.001
n_iter = 5000

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.where(a > 0, a, 0)
```

jupyter np_svm (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

予測

In [11]:
```
index = a > 1e-6
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()
```

In [12]:
```
xx0, xx1 = np.meshgrid(np.linspace(-1.5, 1.5, 100), np.linspace(-1.5, 1.5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * rbf(X_test[i], sv)
y_pred = np.sign(y_project)
```

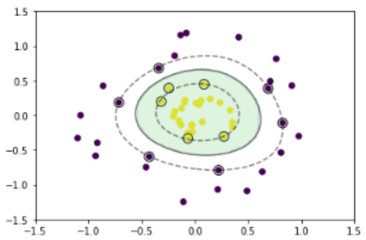
jupyter np_svm (autosaved)

File Edit View Insert Cell Kernel Widgets Help Notebook saved Trusted Python 3

y_pred = np.sign(y_project)

In [13]:
```
# 訓練データを可視化
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
                    s=100, facecolors='none', edgecolors='k')
# 領域を可視化
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
                     levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

Out[13]: <matplotlib.contour.QuadContourSet at 0x7f6c47771358>



jupyter np_svm (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

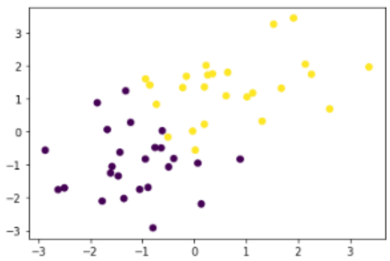
ソフトマージンSVM

訓練データ生成③（重なりあり）

In [14]:
```
x0 = np.random.normal(size=50).reshape(-1, 2) - 1.
x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
x_train = np.concatenate([x0, x1])
y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
```

In [15]:
```
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
```

Out[15]: <matplotlib.collections.PathCollection at 0x7f6c476e5b70>



jupyter np_svm (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

学習

分離不可能な場合は学習できないが、データ点がマージン内部に入ることや誤分類を許容することでその問題を回避する。

スラック変数$\xi_i \geq 0$を導入し、マージン内部に入ったり誤分類された点に対しては、$\xi_i = |1 - t_i y(\boldsymbol{x}_i)|$とし、これらを許容する代わりに対して、ペナルティを与えるように、最適化問題を以下のように修正する。

jupyter np_svm (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

スラック変数$\xi_i \geq 0$を導入し、マージン内部に入ったり誤分類された点に対しては、$\xi_i = |1 - t_i y(\boldsymbol{x}_i)|$とし、これらを許容する代わりに対して、ペナルティを与えるように、最適化問題を以下のように修正する。

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad t_i(\boldsymbol{w}\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \quad (i = 1, 2, \cdots, n)$$

ただし、パラメータCはマージンの大きさと誤差の許容度のトレードオフを決めるパラメータである。この最適化問題をラグランジュ乗数法などを用いると、結局最大化する目的関数はハードマージンSVMと同じとなる。

$$\tilde{L}(\boldsymbol{a}) = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}} \phi(\boldsymbol{x}_j)$$

ただし、制約条件が$a_i \geq 0$の代わりに$0 \leq a_i \leq C (i = 1, 2, \ldots, n)$となる。（ハードマージンSVMと同じ$\sum_{i=1}^{n} a_i t_i = 0$も制約条件）

In [16]:
```
X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

C = 1
eta1 = 0.01
eta2 = 0.001
n_iter = 1000

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.clip(a, 0, C)
```

jupyter np_svm (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

予測

In [17]:
```
index = a > 1e-8
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()
```

In [18]:
```
xx0, xx1 = np.meshgrid(np.linspace(-4, 4, 100), np.linspace(-4, 4, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * sv.dot(X_test[i])
y_pred = np.sign(y_project)
```

jupyter np_svm (autosaved)

Logou

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

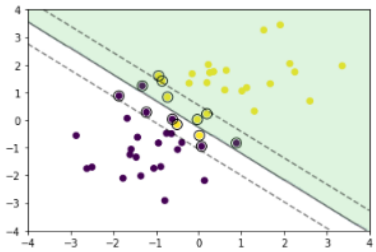
Run Code

```python
y_pred = np.sign(y_project)
```

In [19]:

```python
# 訓練データを可視化
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
                    s=100, facecolors='none', edgecolors='k')
# 領域を可視化
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
                     levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

Out[19]: <matplotlib.contour.QuadContourSet at 0x7f6c47661668>



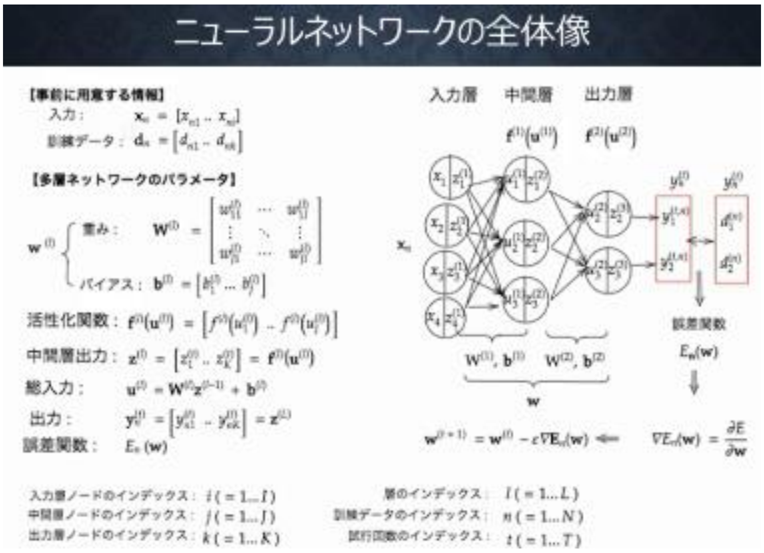
3. 深層学習（前編１）

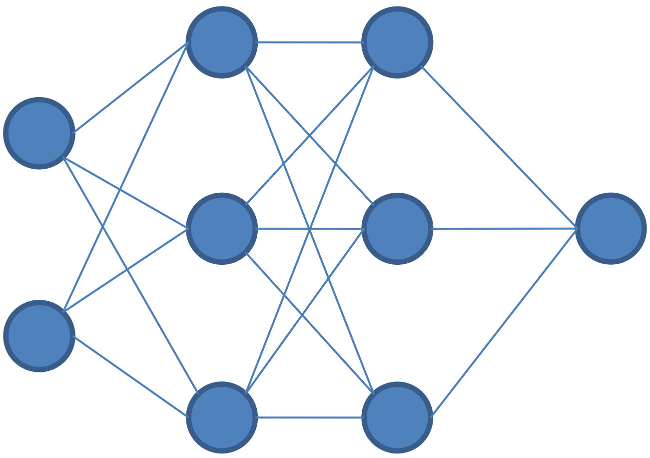
3.1. Section1：入力層～中間層

・X1～Xn―入力層―W1,B1―中間層－W2,B2―出力層

DL の最適化の最終目的：誤差を最小化するパラメータを発見すること

パラメータ：重み、バイアス





NN ができること

回帰：結果予測（売上予測、株価）、ランキング（競馬順位、人気順位）

分類：猫写真の判別、花の種類分類、手書き文字認識

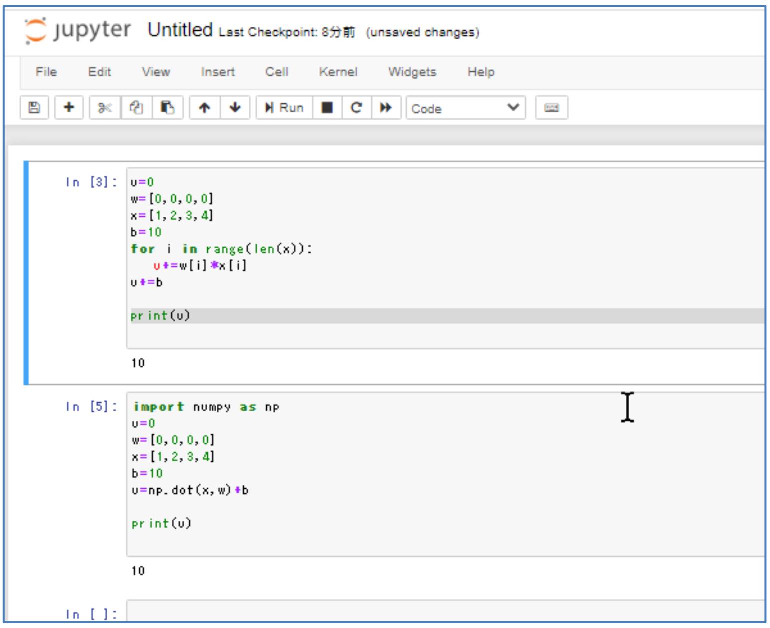
実用例：自動売買、チャットボット、翻訳、音声解釈、囲碁将棋

ハンズオン（実装演習）

入力：xi 、重み：wi、バイアス：b、総入力：u、出力：z、活性化関数：f

入力層のインデックス：i

u＝w1x1+w2x2+w3x3+w4x4+b＝Wx+b



内積で書ける。

順伝播（単層・単ユニット）の実装演習

jupyter 1_1_forward_propagation Last Checkpoint: 2018/11/24 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]:
```python
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import numpy as np
from common import functions

def print_vec(text, vec):
    print("*** " + text + " ***")
    print(vec)
    #print("shape: " + str(x.shape))
    print("")
```

In [2]:
```python
# 順伝播（単層・単ユニット）

# 重み
W = np.array([[0.1], [0.2]])

## 試してみよう_配列の初期化
#W = np.zeros(2)
#W = np.ones(2)
#W = np.random.rand(2)
#W = np.random.randint(5, size=(2))

print_vec("重み", W)


# バイアス
b = 0.5

## 試してみよう_数値の初期化
```

jupyter 1_1_forward_propagation Last Checkpoint: 2018/11/24 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```python
## 試してみよう_数値の初期化
#b = np.random.rand() # 0~1のランダム数値
#b = np.random.rand() * 10 -5  # -5~5のランダム数値

print_vec("バイアス", b)

# 入力値
x = np.array([2, 3])
print_vec("入力", x)


# 総入力
u = np.dot(x, W) + b
print_vec("総入力", u)

# 中間層出力
z = functions.relu(u)
print_vec("中間層出力", z)
```

```
*** 重み ***
[[0.1]
 [0.2]]

*** バイアス ***
0.5

*** 入力 ***
[2 3]

*** 総入力 ***
[1.3]

*** 中間層出力 ***
[1.3]
```

コメント部分での実装演習

重みの変化：

```
*** 重み ***
[0. 0.]

*** バイアス ***
0.5

*** 入力 ***
[2 3]

*** 総入力 ***
0.5

*** 中間層出力 ***
0.5
```

```
*** 重み ***
[1. 1.]

*** バイアス ***
0.5

*** 入力 ***
[2 3]

*** 総入力 ***
5.5

*** 中間層出力 ***
5.5
```

```
*** 重み ***
[0.24135586 0.57921017]

*** バイアス ***
0.5

*** 入力 ***
[2 3]

*** 総入力 ***
2.720342216381039

*** 中間層出力 ***
2.720342216381039
```

```
*** 重み ***
[4 2]

*** バイアス ***
0.5

*** 入力 ***
[2 3]

*** 総入力 ***
14.5

*** 中間層出力 ***
14.5
```

バイアスの変化

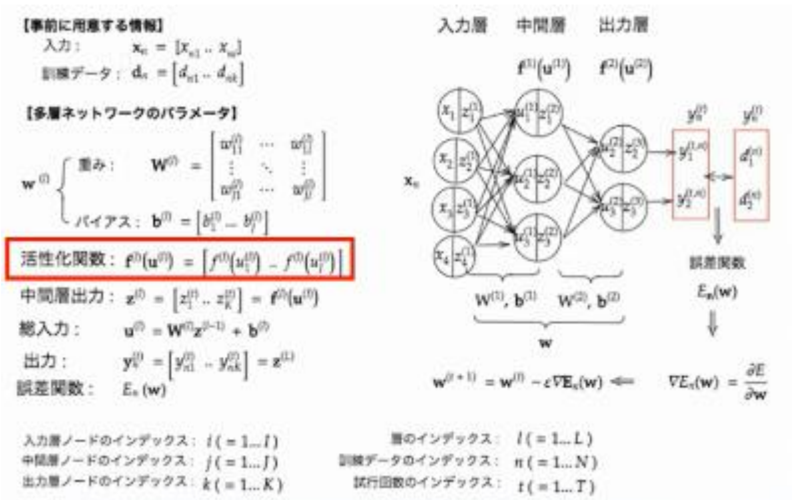
```
*** 重み ***
[[0.1]
 [0.2]]

*** バイアス ***
0.2713144791886195

*** 入力 ***
[2 3]

*** 総入力 ***
[1.07131448]

*** 中間層出力 ***
[1.07131448]
```

```
*** 重み ***
[[0.1]
 [0.2]]

*** バイアス ***
-3.1822816992574854

*** 入力 ***
[2 3]

*** 総入力 ***
[-2.3822817]

*** 中間層出力 ***
[0.]
```

3.2. Section2：**活性化関数**

NN で、次の層への出力の大きさを決める<u>非線形の関数</u>

：線形と非線形の図

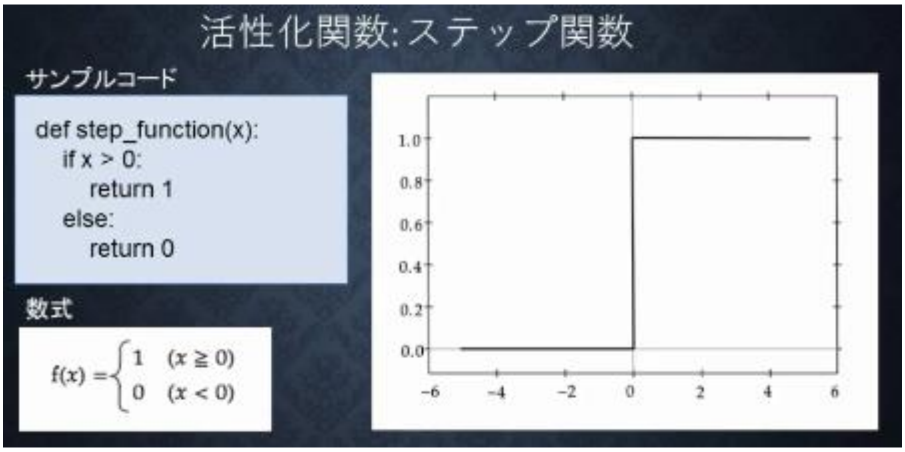


中間層用の活性化関数

- ReLU
- シグモイド（ロジスティック）
- ステップ

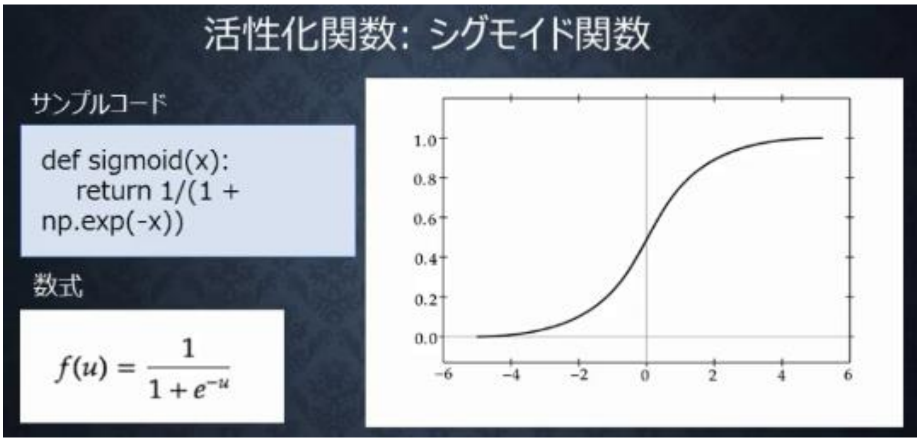
出力層用の活性化関数

- ソフトマックス
- 恒等写像
- シグモイド（ロジスティック）



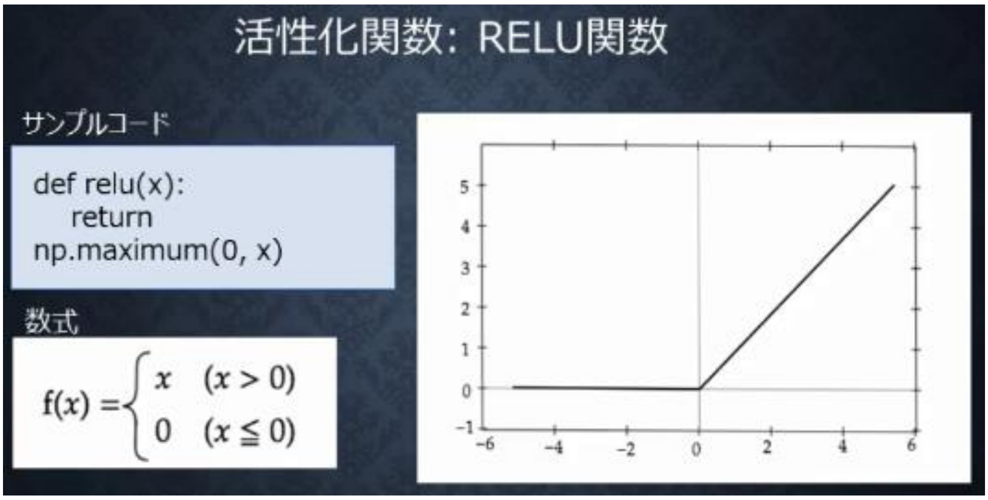
しきい値を超えたら発火する関数。出力は１か0。

デメリット：線形分離可能なものしか学習できなかった。

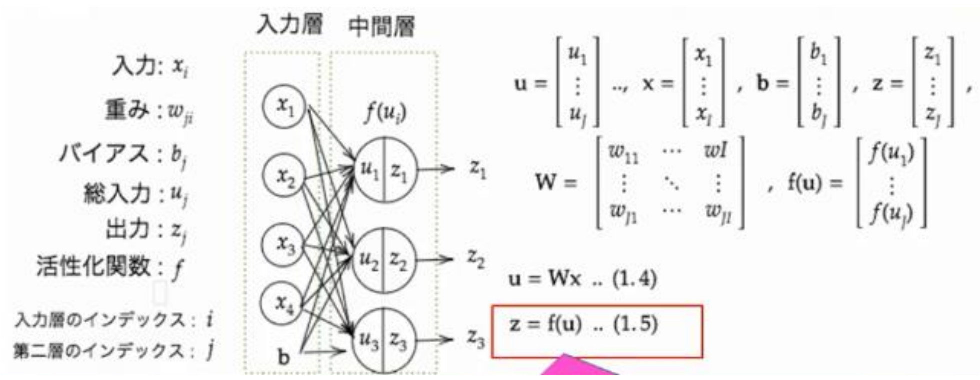


０～１の間を変化する関数で信号の強弱を伝えられる。

デメリット：大きな値では出力の変化が微小、勾配消失問題が起きる事がある



最も使われている活性化関数
勾配消失問題の回避、スパース化に貢献。



ハンズオン（実装演習）

jupyter 1_1_forward_propagation Last Checkpoint: 2018/11/24 (autosaved)

File Edit View Insert Cell Kernel Widgets Help   Trusted   Python

In [7]:

```
# 順伝播（単層・複数ユニット）

# 重み
W = np.array([
    [0.1, 0.2, 0.3],
    [0.2, 0.3, 0.4],
    [0.3, 0.4, 0.5],
    [0.4, 0.5, 0.6]
])

## 試してみよう_配列の初期化
#W = np.zeros((4,3))
#W = np.ones((4,3))
#W = np.random.rand(4,3)
#W = np.random.randint(5, size=(4,3))

print_vec("重み", W)

# バイアス
b = np.array([0.1, 0.2, 0.3])
print_vec("バイアス", b)

# 入力値
x = np.array([1.0, 5.0, 2.0, -1.0])
print_vec("入力", x)


#  総入力
u = np.dot(x, W) + b
print_vec("総入力", u)

# 中間層出力
z = functions.sigmoid(u)
print_vec("中間層出力", z)
```

```
*** 重み ***
[[0.1 0.2 0.3]
 [0.2 0.3 0.4]
 [0.3 0.4 0.5]
 [0.4 0.5 0.6]]

*** バイアス ***
[0.1 0.2 0.3]

*** 入力 ***
[ 1.  5.  2. -1.]

*** 総入力 ***
[1.4 2.2 3. ]

*** 中間層出力 ***
[0.80218389 0.90024951 0.95257413]
```

jupyter functions (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

```
In [ ]: import numpy as np
```

```
In [ ]: # 中間層の活性化関数
        # シグモイド関数（ロジスティック関数）
        def sigmoid(x):
            return 1/(1 + np.exp(-x))
```

```
In [ ]: # ReLU関数
        def relu(x):
            return np.maximum(0, x)
```

```
In [ ]: # ステップ関数（閾値0）
        def step_function(x):
            return np.where( x > 0, 1, 0)
```

```
In [ ]: # 出力層の活性化関数
        # ソフトマックス関数
        def softmax(x):
            if x.ndim == 2:
                x = x.T
                x = x - np.max(x, axis=0)
                y = np.exp(x) / np.sum(np.exp(x), axis=0)
                return y.T

            x = x - np.max(x) # オーバーフロー対策
            return np.exp(x) / np.sum(np.exp(x))
```

```
In [ ]: # ソフトマックスとクロスエントロピーの複合関数
        def softmax_with_loss(d, x):
            y = softmax(x)
            return cross_entropy_error(d, y)
```

```
# 中間層出力
#z = functions.sigmoid(u)
z = functions.relu(u)
print_vec("中間層出力", z)
```

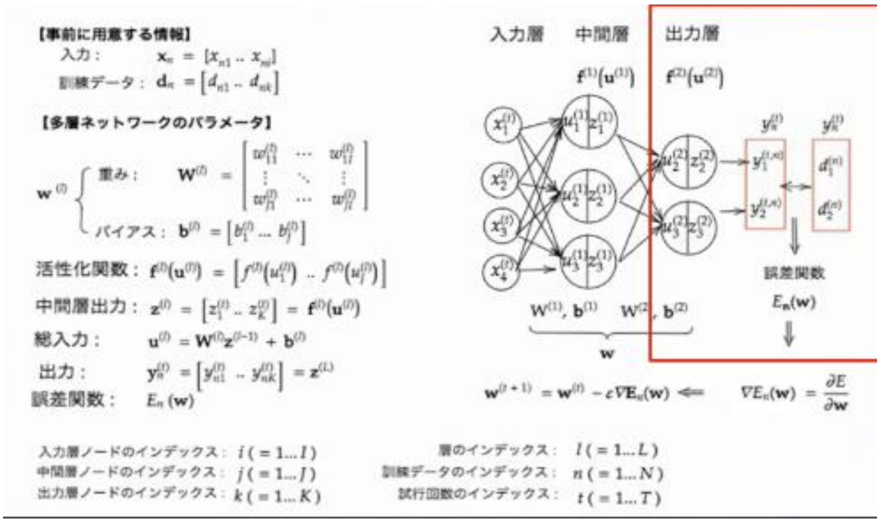
```
*** 重み ***
[[0.1 0.2 0.3]
 [0.2 0.3 0.4]
 [0.3 0.4 0.5]
 [0.4 0.5 0.6]]

*** バイアス ***
[0.1 0.2 0.3]

*** 入力 ***
[ 1.  5.  2. -1.]

*** 総入力 ***
[1.4 2.2 3. ]

*** 中間層出力 ***
[1.4 2.2 3. ]
```

```
# 中間層出力
#z = functions.sigmoid(u)
#z = functions.relu(u)
z = functions.step_function(u)
print_vec("中間層出力", z)
```

```
*** 重み ***
[[0.1 0.2 0.3]
 [0.2 0.3 0.4]
 [0.3 0.4 0.5]
 [0.4 0.5 0.6]]

*** バイアス ***
[0.1 0.2 0.3]

*** 入力 ***
[ 1.  5.  2. -1.]

*** 総入力 ***
[1.4 2.2 3. ]

*** 中間層出力 ***
[1 1 1]
```

3.3. Section3：出力層

３．３．１　出力層の役割



３．３．２　誤差関数

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{J}(y_j - d_j)^2 = \frac{1}{2}\|(\mathbf{y}-\mathbf{d})\|^2$$

・2 乗するのは値を正にするため。

・1/2 は微分の計算を簡単にするため。

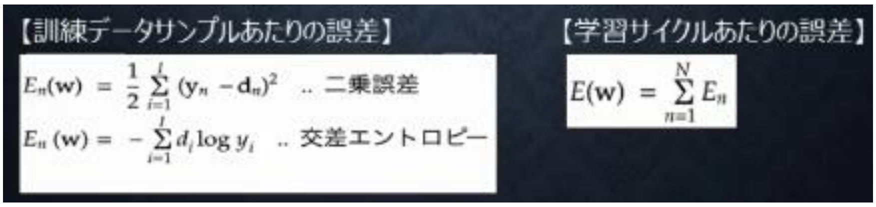
３．３．３　出力層の活性化関数

中間層の活性化関数との違い：値の強弱、多クラス分類では確率出力の総和を 1。

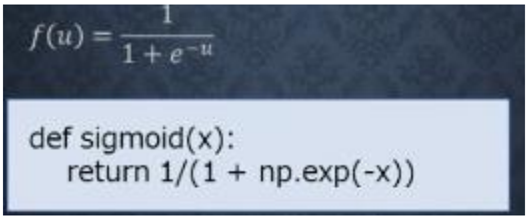
問題に対する、活性化関数と誤差関数の組み合わせ

計算結果で計算の相性が良いため。

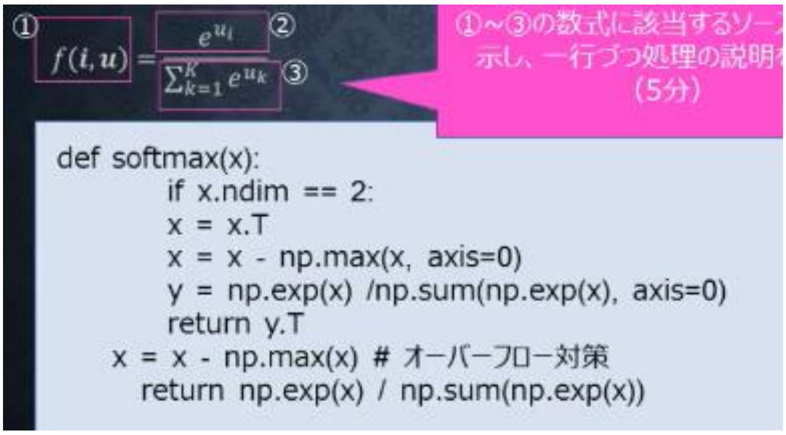
	回帰	二値分類	多クラス分類
活性化関数	恒等写像 $f(u) = u$	シグモイド関数 $f(u) = \frac{1}{1+e^{-u}}$	ソフトマックス関数 $f(i, u) = \frac{e^{u_i}}{\sum_{k=1}^{K} e^{u_k}}$
誤差関数	二乗誤差	交差エントロピー	



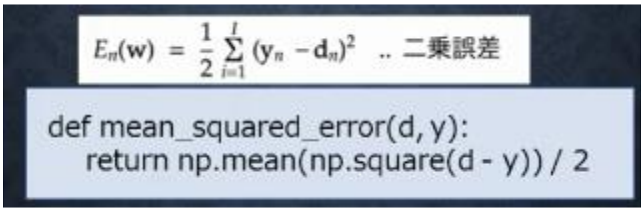
シグモイド関数



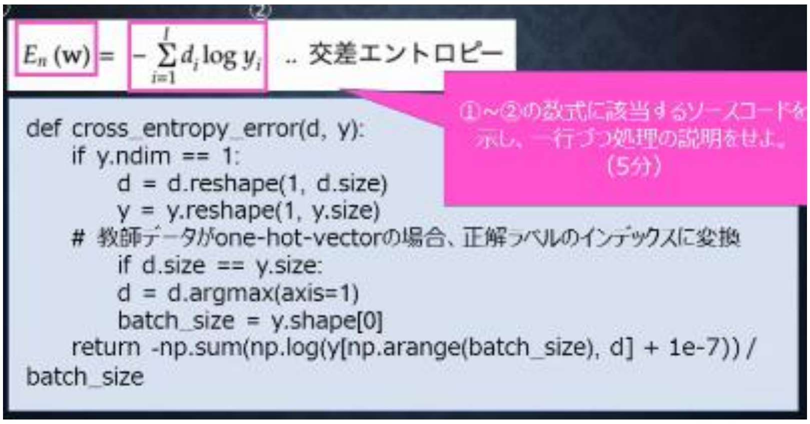
ソフトマックス関数



平均二乗誤差

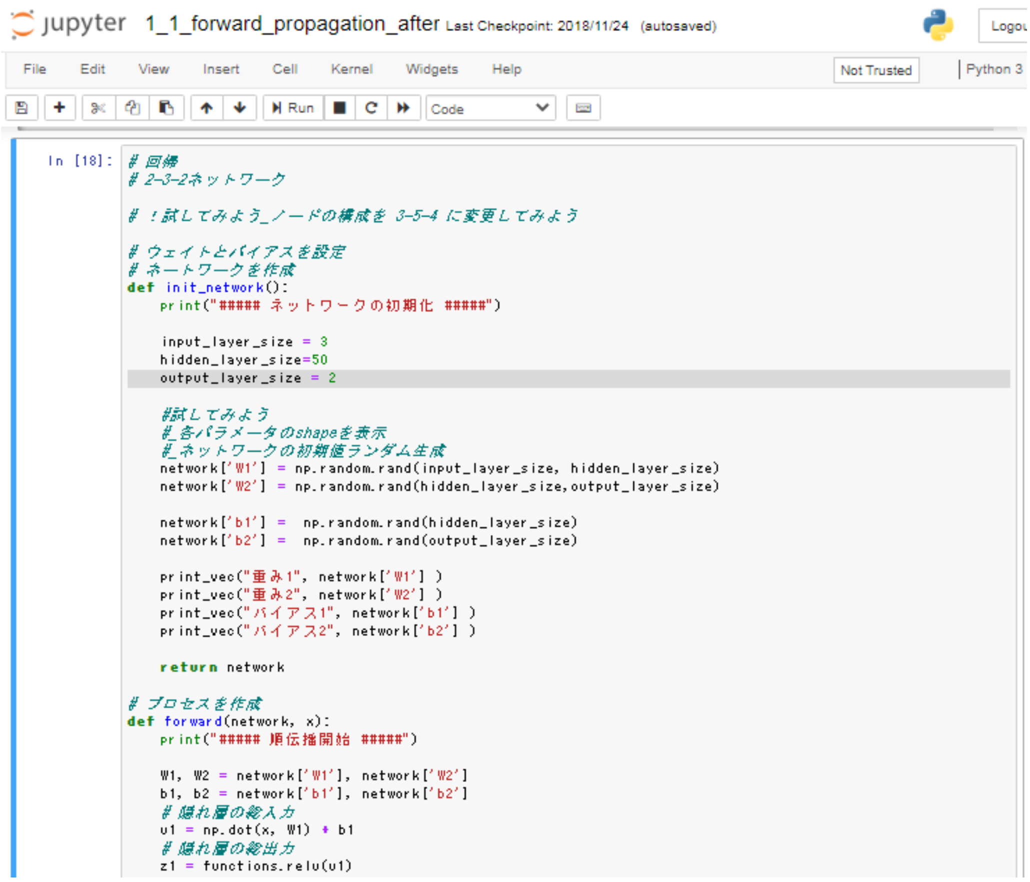


交差エントロピー



実装上の工夫で、微小な値を足すことでゼロにならないようにしている。

ハンズオン（実装演習）



jupyter 1_1_forward_propagation_after Last Checkpoint: 2018/11/24 (autosaved)

File Edit View Insert Cell Kernel Widgets Help   Not Trusted   Python 3

```
    b1, b2 = network['b1'], network['b2']
    # 隠れ層の総入力
    u1 = np.dot(x, W1) + b1
    # 隠れ層の総出力
    z1 = functions.relu(u1)
    # 出力層の総入力
    u2 = np.dot(z1, W2) + b2
    # 出力層の総出力
    y = u2

    print_vec("総入力1", u1)
    print_vec("中間層出力1", z1)
    print_vec("総入力2", u2)
    print_vec("出力1", y)
    print("出力合計: " + str(np.sum(y)))

    return y, z1

# 入力値
x = np.array([1., 2., 3.])
network =  init_network()
y, z1 = forward(network, x)
# 目標出力
d = np.array([2., 4.])
# 誤差
loss = functions.mean_squared_error(d, y)

## 表示
print("¥n##### 結果表示 #####")
print_vec("中間層出力", z1)
print_vec("出力", y)
print_vec("訓練データ", d)
print_vec("二乗誤差",  loss)
```

```
##### ネットワークの初期化 #####
*** 重み1 ***
[[0.11495666 0.80192791 0.50124547 0.71612997 0.32522069 0.60879289
  0.18795707 0.11660115 0.53667371 0.68488913 0.03304506 0.19916327
  0.93297597 0.01047797 0.61461615 0.88127373 0.46285887 0.49167809
  0.1513227  0.80744985 0.46103037 0.23392669 0.39236901 0.11645287
  0.63410819 0.34969928 0.70049046 0.94088253 0.68942358 0.9491729
```

jupyter 1_1_forward_propagation_after Last Checkpoint: 2018/11/24 (autosaved)

File Edit View Insert Cell Kernel Widgets Help   Not Trusted   Python 3

```
 4.53816765 3.07476882 5.0541595  4.60875486 5.68562349 3.79411169
 5.35541201 2.55298344]
shape: (50,)

*** 総入力2 ***
[103.5496362   92.49459416]
shape: (2,)

*** 出力1 ***
[103.5496362   92.49459416]
shape: (2,)

出力合計: 196.04423036574318

##### 結果表示 #####
*** 中間層出力 ***
[2.55938067 5.25600364 3.20473261 2.58653406 3.60944828 2.71672146
 2.80774731 3.29740517 5.19625017 5.50140213 2.75813649 2.75533298
 4.87634243 3.03151824 4.89076814 4.4172441  3.48205477 2.77712663
 2.22306945 3.92154479 1.314171   2.76071347 3.18157423 3.76628928
```

jupyter 1_1_forward_propagation_after Last Checkpoint: 2018/11/24 (autosaved)

```
 2.22306945 3.92154479 1.314171   2.76071347 3.18157423 3.76628928
 2.88783274 3.26613117 3.38348884 3.86188332 3.12028162 3.16309066
 5.40448364 3.27172073 2.62475859 1.52012354 2.12875147 6.18061162
 4.55892713 1.75060036 4.64668576 3.50519156 2.76711346 4.99673109
 4.53816765 3.07476882 5.0541595  4.60875486 5.68562349 3.79411169
 5.35541201 2.55298344]
shape: (50,)

*** 出力 ***
[103.5496362   92.49459416]
shape: (2,)

*** 訓練データ ***
[2. 4.]
shape: (2,)

*** 二乗誤差 ***
4535.905452252771
shape: ()
```

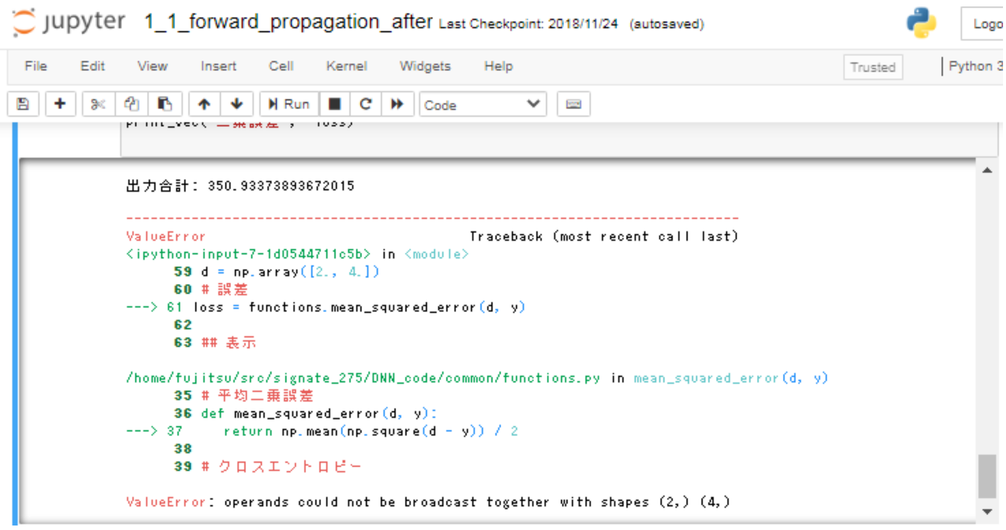
出力層を変更：

jupyter 1_1_forward_propagation_after Last Checkpoint: 2018/11/24 (unsaved changes)

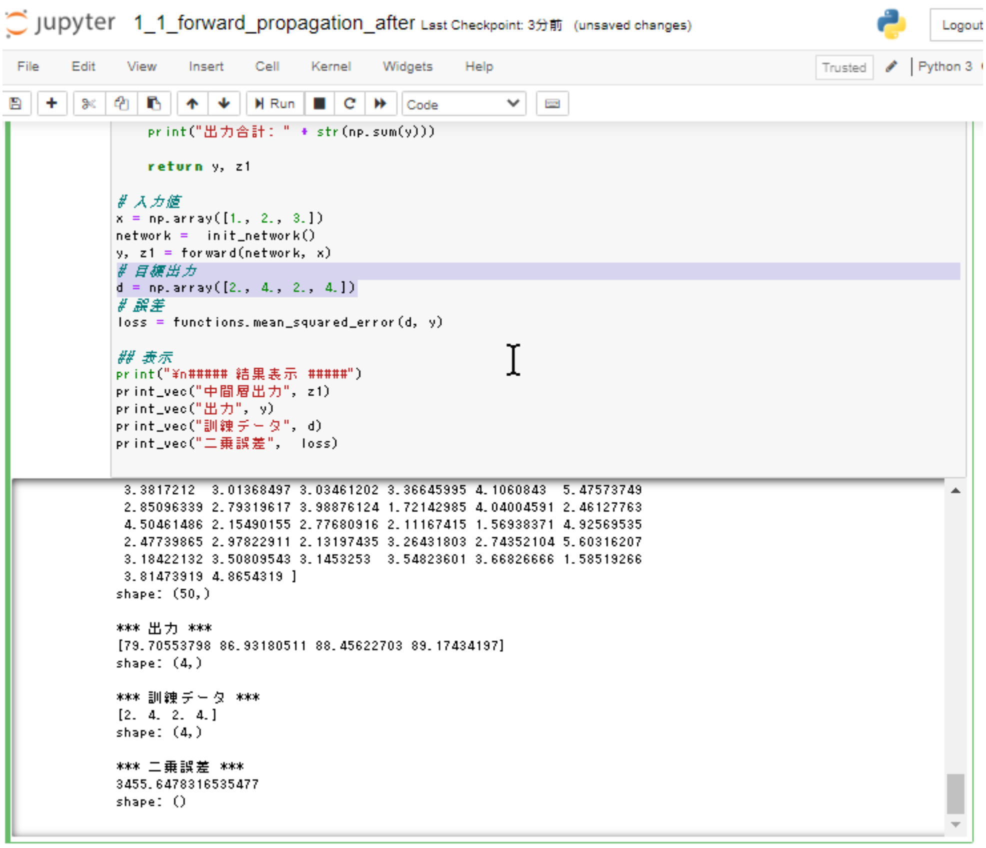
In [7]:

```python
# 回帰
# 2-3-2ネットワーク

# ！試してみよう_ノードの構成を 3-5-4 に変更してみよう

# ウェイトとバイアスを設定
# ネートワークを作成
def init_network():
    print("##### ネットワークの初期化 #####")

    input_layer_size = 3
    hidden_layer_size= 50
    output_layer_size = 4

    #試してみよう
    #_各パラメータのshapeを表示
    #_ネットワークの初期値ランダム生成
    network['W1'] = np.random.rand(input_layer_size, hidden_layer_size)
    network['W2'] = np.random.rand(hidden_layer_size,output_layer_size)

    network['b1'] =  np.random.rand(hidden_layer_size)
    network['b2'] =  np.random.rand(output_layer_size)

    print_vec("重み1", network['W1'] )
    print_vec("重み2", network['W2'] )
    print_vec("バイアス1", network['b1'] )
    print_vec("バイアス2", network['b2'] )

    return network

# プロセスを作成
def forward(network, x):
    print("##### 順伝播開始 #####")

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']
    # 隠れ層の総入力
    u1 = np.dot(x, W1) + b1
    # 隠れ層の総出力
    z1 = functions.relu(u1)
    # 出力層の総入力
```

エラー：y のサイズが違う



目標出力(y)を修正。



3.4. Section4：勾配降下法

３．４．１　勾配降下法

誤差を最小にするネットワークを作る

誤差を最小化するパラメータを発見する

勾配降下法を利用してパラメータを最適化する

誤差関数の値を小さくする方向に重み W およびバイアス b を更新し、次のエポックに反映

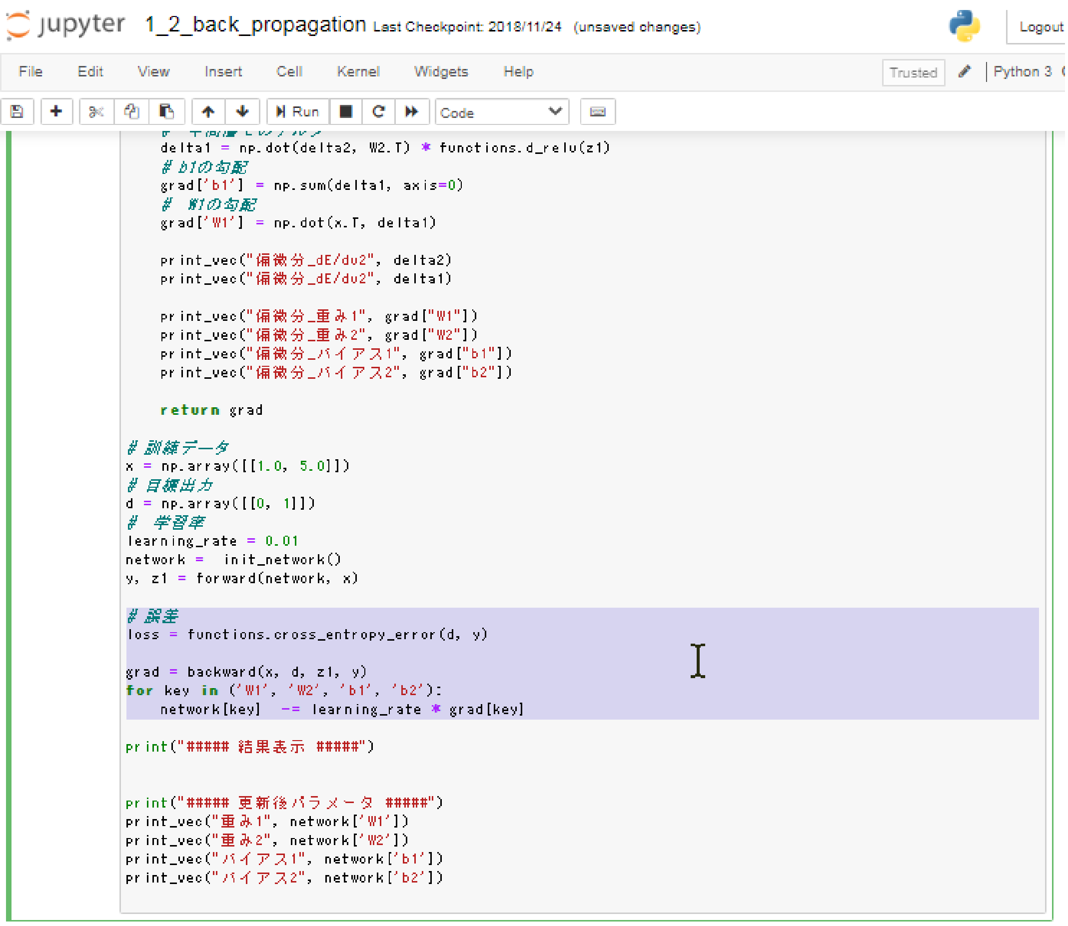
【勾配降下法】

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla E$$

$$\nabla E = \frac{\partial E}{\partial \mathbf{w}} = \left[\frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_M} \right]$$

∇E：誤差を w パラメータで微分した値

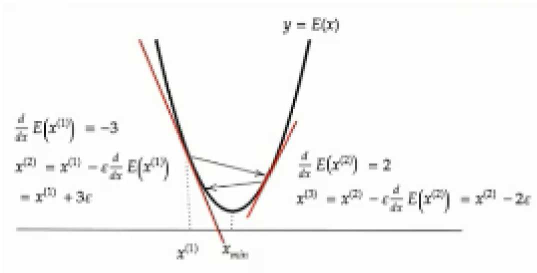
ハンズオン（実装演習）




```
    delta1 = np.dot(delta2, W2.T) * functions.d_relu(z1)
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    #  W1の勾配
    grad['W1'] = np.dot(x.T, delta1)

    print_vec("偏微分_dE/du2", delta2)
    print_vec("偏微分_dE/du2", delta1)

    print_vec("偏微分_重み1", grad["W1"])
    print_vec("偏微分_重み2", grad["W2"])
    print_vec("偏微分_バイアス1", grad["b1"])
    print_vec("偏微分_バイアス2", grad["b2"])

    return grad

# 訓練データ
x = np.array([[1.0, 5.0]])
# 目標出力
d = np.array([[0, 1]])
#  学習率
learning_rate = 0.01
network =  init_network()
y, z1 = forward(network, x)

# 誤差
loss = functions.cross_entropy_error(d, y)

grad = backward(x, d, z1, y)
for key in ('W1', 'W2', 'b1', 'b2'):
    network[key]  -= learning_rate * grad[key]

print("##### 結果表示 #####")


print("##### 更新後パラメータ #####")
print_vec("重み1", network['W1'])
print_vec("重み2", network['W2'])
print_vec("バイアス1", network['b1'])
print_vec("バイアス2", network['b2'])
```

ε 学習率：学習率の値によって学習の効率が大きく異なる

大きくした場合：最小値にいつまでもたどり着かず発散



学習率の決定方法、収束性向上のためのアルゴリズム

・Momentum、AdaGrad、Adadelta、Adam

３．４．２　確率的勾配降下法（SGD）

勾配降下法のデメリットの解決

勾配降下法は全サンプルの平均誤差、

確率的勾配降下法はランダムに抽出したサンプルの誤差

・データが冗長な場合には、計算コストの低減

・望まない局所極小解に収束するリスク低減

・オンライン学習が可能（あとから追加）

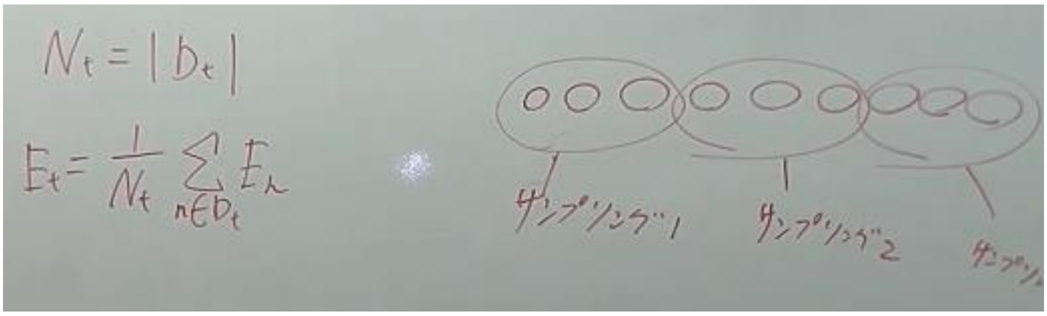
３．４．３　ミニバッチ勾配降下法

確率的勾配降下法と同様にデータに対するアプローチが変わる

ランダムに分割したデータの集合（ミニバッチ）Dt に属するサンプルの平均誤差。

確率的勾配降下法のメリットを損なわずに、計算資源を有効活用可能。

（CPU のスレッド並列化や GPU の SIMD 並列化）



誤差勾配の計算、数値微分

$$\nabla E = \frac{\partial E}{\partial \mathbf{w}} = \left[\frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_M} \right]$$

数値微分を行う、プログラムで微小な数値を生成し、疑似的に微分計算

デメリット : 各パラメータｗｍそれぞれについて、順伝播の計算を繰り返すため負荷が大きい　→誤差逆伝播法を利用する

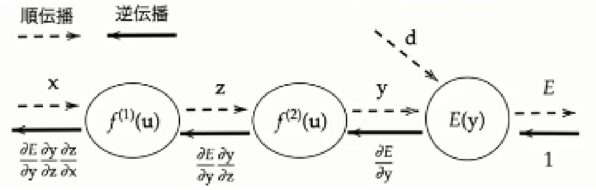
$$\frac{\partial E}{\partial w_m} \approx \frac{E(w_m + h) - E(w_m - h)}{2h}$$

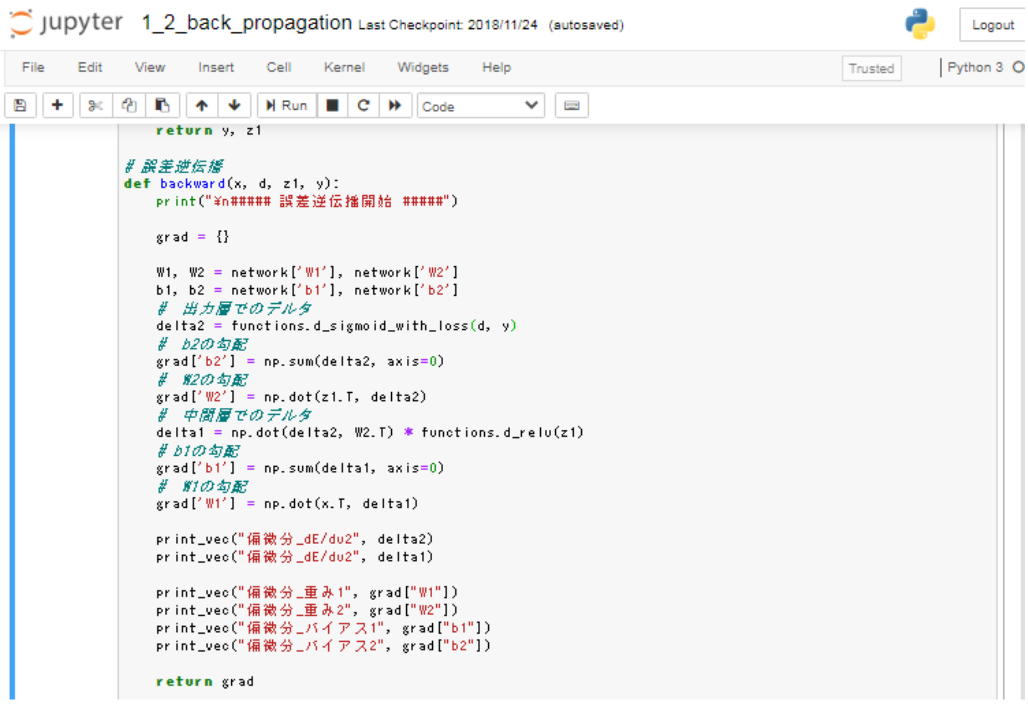
3.5. Section5：誤差逆伝播法

数値微分でも算出できるが、負荷が大きい。計算負荷を減らすために。

算出された誤差を、出力層側から順に微分して、前の層へと伝播。

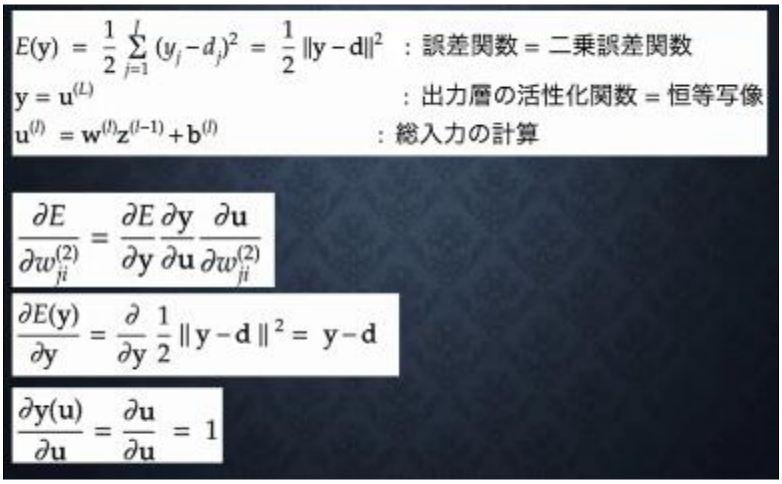
最小限の計算で各パラメータでの微分値を解析的に計算できる。





jupyter 1_2_back_propagation Last Checkpoint: 2018/11/24 (autosaved)

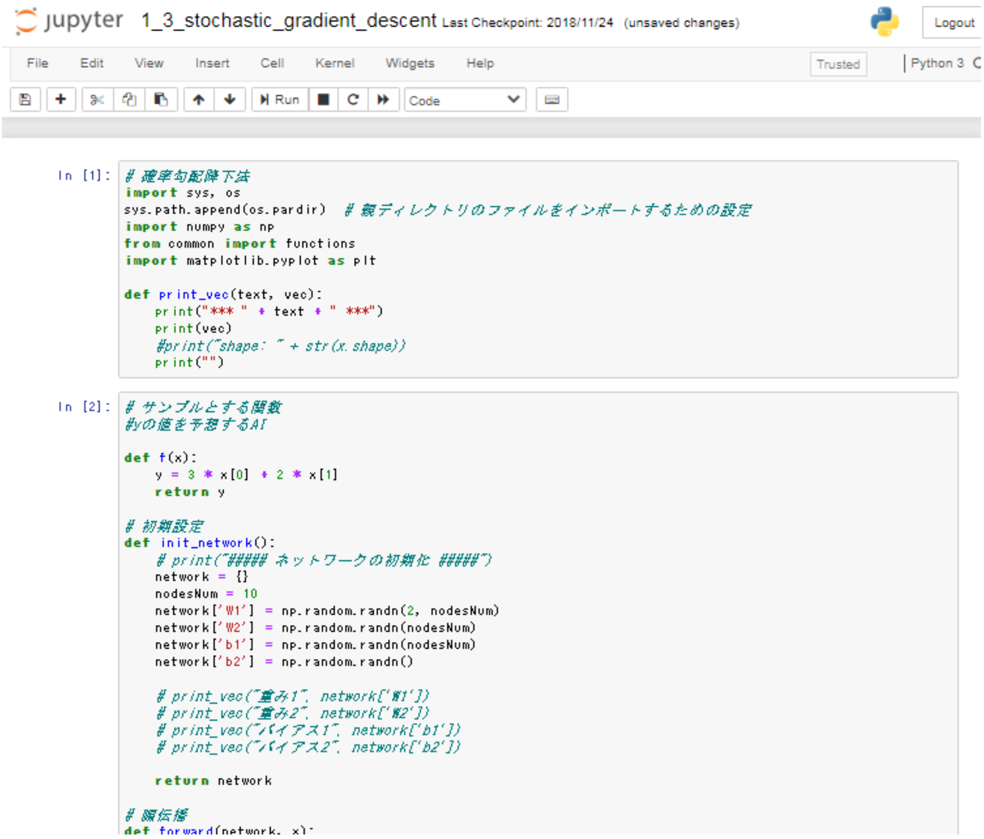
```
    return y, z1

# 誤差逆伝播
def backward(x, d, z1, y):
    print("¥n##### 誤差逆伝播開始 #####")

    grad = {}

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']
    #  出力層でのデルタ
    delta2 = functions.d_sigmoid_with_loss(d, y)
    #  b2の勾配
    grad['b2'] = np.sum(delta2, axis=0)
    #  W2の勾配
    grad['W2'] = np.dot(z1.T, delta2)
    #  中間層でのデルタ
    delta1 = np.dot(delta2, W2.T) * functions.d_relu(z1)
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    #  W1の勾配
    grad['W1'] = np.dot(x.T, delta1)

    print_vec("偏微分_dE/du2", delta2)
    print_vec("偏微分_dE/du2", delta1)

    print_vec("偏微分_重み1", grad["W1"])
    print_vec("偏微分_重み2", grad["W2"])
    print_vec("偏微分_バイアス1", grad["b1"])
    print_vec("偏微分_バイアス2", grad["b2"])

    return grad
```



$$\frac{\partial \mathrm{u}(\mathrm{w})}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left(\mathrm{w}^{(l)}\mathrm{z}^{(l-1)} + \mathrm{b}^{(l)}\right) = \frac{\partial}{\partial w_{ji}}\left(\begin{bmatrix} w_{11}z_1 + \dots + w_{1i}z_i + \dots w_{1I}z_I \\ \vdots \\ w_{j1}z_1 + \dots + w_{ji}z_i + \dots w_{jI}z_I \\ \vdots \\ w_{J1}z_1 + \dots + w_{Ji}z_i + \dots w_{JI}z_I \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_j \\ \vdots \\ b_J \end{bmatrix}\right) = \begin{bmatrix} 0 \\ \vdots \\ z_i \\ \vdots \\ 0 \end{bmatrix}$$

$$\frac{\partial E}{\partial \mathrm{y}}\frac{\partial \mathrm{y}}{\partial \mathrm{u}}\frac{\partial \mathrm{u}}{\partial w_{ji}^{(2)}} = (\mathrm{y} - \mathrm{d}) \cdot \begin{bmatrix} 0 \\ \vdots \\ z_i \\ \vdots \\ 0 \end{bmatrix} = (y_j - d_j)z_i$$

ハンズオン（実装演習）



jupyter 1_3_stochastic_gradient_descent Last Checkpoint: 2018/11/24 (unsaved changes)

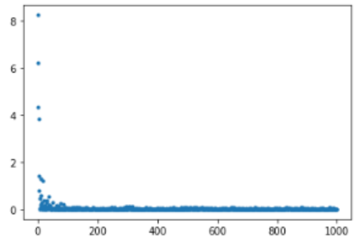
```
In [1]:
# 確率勾配降下法
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import numpy as np
from common import functions
import matplotlib.pyplot as plt

def print_vec(text, vec):
    print("*** " + text + " ***")
    print(vec)
    #print("shape: " + str(x.shape))
    print("")
```

```
In [2]:
# サンプルとする関数
#yの値を予想するAI

def f(x):
    y = 3 * x[0] + 2 * x[1]
    return y

# 初期設定
def init_network():
    # print("##### ネットワークの初期化 #####")
    network = {}
    nodesNum = 10
    network['W1'] = np.random.randn(2, nodesNum)
    network['W2'] = np.random.randn(nodesNum)
    network['b1'] = np.random.randn(nodesNum)
    network['b2'] = np.random.randn()

    # print_vec("重み1", network['W1'])
    # print_vec("重み2", network['W2'])
    # print_vec("バイアス1", network['b1'])
    # print_vec("バイアス2", network['b2'])

    return network

# 順伝播
def forward(network, x):
```

：

```
print("##### 結果表示 #####")
lists = range(epoch)


plt.plot(lists, losses, '.')
# グラフの表示
plt.show()
```

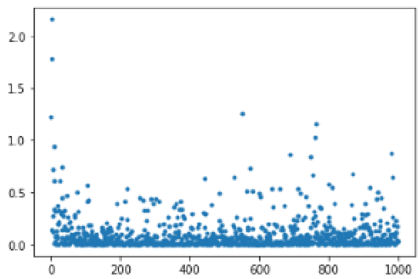
結果表示

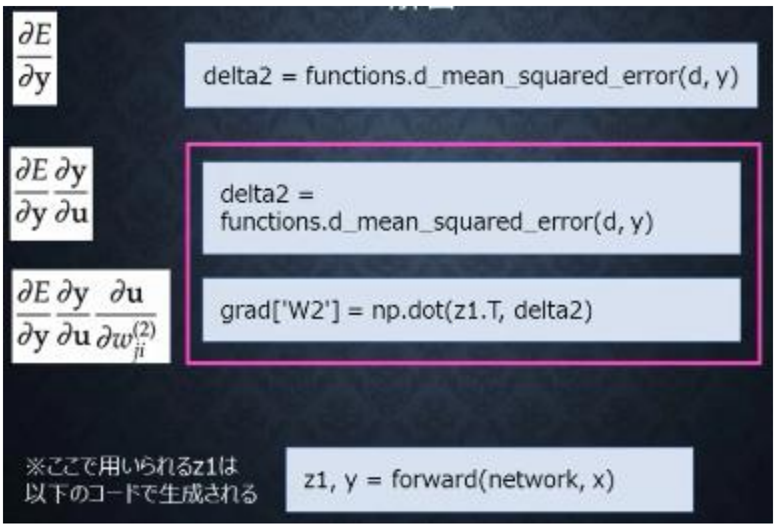


Relu→sigmoid

```
print("##### 結果表示 #####")
lists = range(epoch)


plt.plot(lists, losses, '.')
# グラフの表示
plt.show()
```

結果表示





4.　修了テスト

4.1.　修了課題

iris データを使用して、回帰または分類。
Jupyter notebook 上でソースが動く
組み立てた NN(deeplearning)の構造図、可視化図
入力層、出力層、誤差関数の明記

4.2.　修了課題の確認について

Q1. 課題の目的とは？　どのような工夫ができそうか

A1. 4 個の計測値データから深層学習を用いて、セトナ，バーシクル，　バージニカの 3 種類のアヤメに分類する。これにより深層学習の構造を理解する。

Q2. 課題を分類タスクで解く場合の意味は何か

A2. 新たな未分類データで、分類することができる。

Q3. iris データとは何か 2 行で述べよ

A3. アヤメの花のデータセット。セトナ，バーシクル，　バージニカの 3 種類のアヤメで 4 個の計測値を持つ。イギリスのロナルド・フィッシャーが 1936 年に論文で紹介した。

4.3.　実装

iris データを使用して、回帰で予測する。

jupyter 修了課題 Last Checkpoint: 10分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

In [1]:
```python
import sys, os
import numpy as np

sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
from common import functions # 演習で利用した関数を利用
```

In [2]:
```python
# irisデータを取得
from sklearn import datasets
iris = datasets.load_iris()
iris.data.shape # 行数、列数
```

Out[2]:
```
(150, 4)
```

In [3]:
```python
def print_vec(text, vec):
    print("*** " + text + " ***")
    print(vec)
    #print("shape: " + str(x.shape))
    print("")
```

In [4]:
```python
# 初期設定
def init_network():
    print("##### ネットワークの初期化 #####")
    network = {}
    nodesNum = 10
    network['W1'] = np.random.randn(4, nodesNum)
    network['W2'] = np.random.randn(nodesNum)
    network['b1'] = np.random.randn(nodesNum)
    network['b2'] = np.random.randn()

    print_vec("重み1", network['W1'])
    print_vec("重み2", network['W2'])
    print_vec("バイアス1", network['b1'])
    print_vec("バイアス2", network['b2'])
    return network
```

jupyter 修了課題 Last Checkpoint: 11分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

In [5]:
```python
# 順伝播
def forward(network, x):
    print("##### 順伝播開始 #####")

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']
    u1 = np.dot(x, W1) + b1
    z1 = functions.relu(u1)
    #z1 = functions.sigmoid(u1)

    u2 = np.dot(z1, W2) + b2
    y = u2

    print_vec("順伝播 総入力1", u1)
    print_vec("順伝播 中間層出力1", z1)
    print_vec("順伝播 総入力2", u2)
    print_vec("順伝播 出力1", y)
    print("順伝播 出力合計: " + str(np.sum(y)))
    return z1, y
```

In [6]:
```python
# 誤差逆伝播
def backward(x, d, z1, y):
    print("¥n##### 誤差逆伝播開始 #####")

    grad = {}

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    # 出力層でのデルタ
    delta2 = functions.d_mean_squared_error(d, y)
    # b2の勾配
    grad['b2'] = np.sum(delta2, axis=0)
    # W2の勾配
    grad['W2'] = np.dot(z1.T, delta2)
    # 中間層でのデルタ
    delta1 = np.dot(delta2, W2.T) * functions.d_relu(z1)
    #delta1 = np.dot(delta2, W2.T) * functions.d_sigmoid(z1)

    delta1 = delta1[np.newaxis, :]
```

jupyter 修了課題 Last Checkpoint: 11分前 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```python
    delta1 = delta1[np.newaxis, :]
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    x = x[np.newaxis, :]
    # W1の勾配
    grad['W1'] = np.dot(x.T, delta1)

    print_vec("誤差逆伝播 偏微分 重み1", grad["W1"])
    print_vec("誤差逆伝播 偏微分 重み2", grad["W2"])
    print_vec("誤差逆伝播 偏微分 バイアス1", grad["b1"])
    print_vec("誤差逆伝播 偏微分 バイアス2", grad["b2"])
    return grad
```

In [7]:

```python
# データ設定
data_sets_size = iris.data.shape[0]
data_sets = [0 for i in range(data_sets_size)]

for i in range(data_sets_size):
    data_sets[i] = {}
    # irisの計測値を設定
    data_sets[i]['x'] = iris.data[i]

    # irisのtargetを設定
    data_sets[i]['d'] = iris.target[i]
```

In [8]:

```python
# 学習率
learning_rate = 0.07

# 抽出数
epoch = 500

# パラメータの初期化
network = init_network()
losses = []
# データのランダム抽出
np.random.seed(seed=epoch)
random_datasets = np.random.choice(data_sets, epoch)

# 勾配降下の繰り返し
```

jupyter 修了課題 Last Checkpoint: 12分前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```python
# パラメータの初期化
network = init_network()
losses = []
# データのランダム抽出
np.random.seed(seed=epoch)
random_datasets = np.random.choice(data_sets, epoch)

# 勾配降下の繰り返し
i = 0
for dataset in random_datasets:
    x, d = dataset['x'], dataset['d']
    z1, y = forward(network, x)
    grad = backward(x, d, z1, y)
    # パラメータに勾配適用
    for key in ('W1', 'W2', 'b1', 'b2'):
        network[key]  -= learning_rate * grad[key]

    # 誤差
    loss = functions.mean_squared_error(d, y)
    losses.append(loss)
    i += 1
    print("誤差 epoch={} loss={}¥n¥n".format(i,loss))
```

```
##### 誤差逆伝播開始 #####
*** 誤差逆伝播 偏微分 重み1 ***
[[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. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

*** 誤差逆伝播 偏微分 重み2 ***
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

*** 誤差逆伝播 偏微分 バイアス1 ***
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

*** 誤差逆伝播 偏微分 バイアス2 ***
-0.07583179360287462
```

jupyter 修了課題 Last Checkpoint: 13分前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

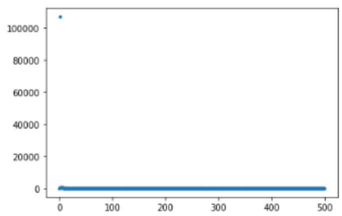
Trusted | Python 3

```
*** 誤差逆伝播 偏微分 重み2 ***
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

*** 誤差逆伝播 偏微分 バイアス1 ***
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

*** 誤差逆伝播 偏微分 バイアス2 ***
-0.07583179360287462

誤差 epoch=500 loss=0.002875230460514488
```

In [9]:

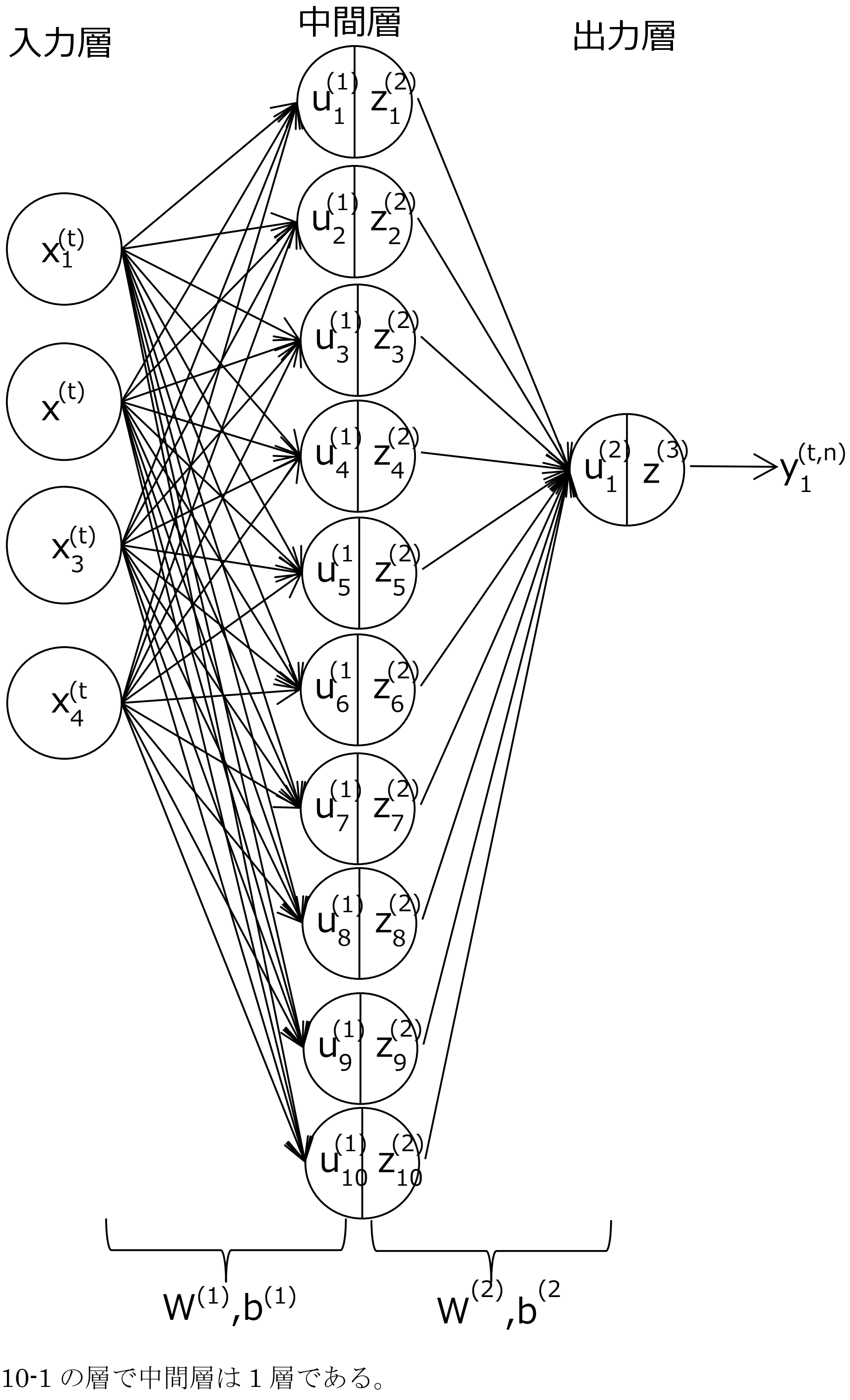
```python
print("##### 結果表示 #####")
import matplotlib.pyplot as plt
lists = range(epoch)

plt.plot(lists, losses, '.')
# グラフの表示
plt.show()
```

```
##### 結果表示 #####
```



早い段階で誤差がほぼゼロになっている。

4.4.　NN の構造図について



・4-10-1 の層で中間層は 1 層である。

・中間層の活性化関数：ReLU 関数

・出力層の活性化関数：恒等写像

・誤差関数：平均二乗誤差
・誤差逆伝播を行っている。

以上