PYTHON 機械学習プログラミング
補足資料

【目次】

１　コスト関数の最小化

（１）確率的降下法

ADALINE では、全てのトレーニングサンプルからコスト関数の勾配を求め、コスト関数が小さくなる方向に向かって重みを更新していました。その為エポック毎に重みが更新されることになります。
これに対し、1 トレーニングサンプル毎にコスト関数の勾配を求め、重みを更新してやる方法を確率的勾配降下法と言います。
この方法を用いると 1 つのトレーニングサンプルが得られた時に、リアルタイムで重みを更新することができます。
最新の結果をリアルタイムで反映することができるので、オンライン勾配降下法とも言います。
また、全トレーニングサンプルからコスト関数の重みを更新する従来の方法はバッチ勾配降下法と呼ばれています。
※以下の式は全て行列で表記しています。大文字で太字は行列、小文字で太字は縦ベクトルを表します。

バッチ勾配降下法（全トレーニングサンプルを使って重みを更新）

$$\Delta \mathbf{w}=-\eta(-\alpha \mathbf{X}^{\mathrm{T}} \mathbf{y}+\alpha^{2} \mathbf{X}^{\mathrm{T}} \mathbf{X} \mathbf{w})$$

確率的勾配降下法（1 トレーニングサンプルを使って重みを更新）

$$\Delta \mathbf{w}=-\eta(-\alpha x_i^T \mathbf{y}_{\mathrm{i}}+\alpha^{2} x_i^T \mathbf{x}_{\mathrm{i}} \mathbf{w})$$

$\mathbf{x}_{\mathrm{i}}$ は i 番目のトレーニングサンプルを表しており、
$\mathbf{x}_{\mathrm{i}}=[1 \quad \mathbf{x}_{\mathrm{i1}} \quad \mathbf{x}_{\mathrm{i2}} \quad \cdots \mathbf{x}_{\mathrm{in}} \quad]$

です。

（２）最小二乗法

ADALINE のコスト関数は、**w** に対して単純な 2 次関数になっており、コスト関数の最小値を与える **w** が 1 組だけ存在します。
最小値におけるコスト関数の勾配が 0 になることから、勾配降下法のような逐次計算を行わなくても、実は一発でコスト関数の最小値を与える **w** を求める事が出来ます。
この方法を最小二乗法と言います。

$$\frac{\partial J(w)}{\partial w}=-\alpha \mathbf{X}^{\mathrm{T}} \mathbf{y}+\alpha^{2} \mathbf{X}^{\mathrm{T}} \mathbf{X} \mathbf{w}=0$$

$$\therefore \quad \mathbf{w}=\frac{1}{\alpha}\left(\mathbf{X}^{\mathrm{T}} \mathbf{X}\right)^{-1} \mathbf{X}^{\mathrm{T}} \mathbf{y}$$

最小二乗法では一発でコスト関数の最小値を与える **w** が求まるので一番いい方法なのですが、もっと複雑な関数の場合は使えません。限られた場合にしか使えないことに注意してください。

前回の ADALINE のコードを少し変更して、確率的勾配降下法がどのような振る舞いをするのか見ていきたいと思います。
確率的勾配降下法では 1 トレーニングサンプル毎に重みが更新されるので、トレーニングサンプルの順番をランダムにシャッフルしてやった方が学習効率は上がります。この機能もつけようと思います。
また、最小二乗法で一発で求めた解とも比較してみます。

```
# -*- coding: utf-8 -*-
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def main():
    # ---アヤメデータの取得
    df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header=None)
    y = df.iloc[0:100, 4].values
    y = np.where(y == "Iris-setosa", -1, 1)
    X = df.iloc[0:100, [0, 2]].values

    # ---学習実施(学習率:0.01 最大エポック数:20 (特徴量を標準化,シャッフルオフ,バッチ勾配降下法))
    Xstd = np.copy(X)
    Xstd[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
    Xstd[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
    ada = Adaline(eta=0.01, numEpoch=20, shuffle=False)
    ada.fit(Xstd, y)
    plotResult(ada,Xstd,y,"Standardization","Off","GD")

    # ---学習実施(学習率:0.01 最大エポック数:20 (特徴量を標準化,シャッフルオフ,確率的勾配降下法))
    Xstd = np.copy(X)
    Xstd[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
    Xstd[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
    ada = Adaline(eta=0.01, numEpoch=20, shuffle=False)
    ada.fitSGD(Xstd, y)
    plotResult(ada, Xstd, y, "Standardization", "Off", "SGD")

    # ---学習実施(学習率:0.01 最大エポック数:20 (特徴量を標準化,シャッフルオン,確率的勾配降下法))
    Xstd = np.copy(X)
    Xstd[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
    Xstd[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
    ada = Adaline(eta=0.01, numEpoch=20, shuffle=True)
    ada.fitSGD(Xstd, y)
    plotResult(ada, Xstd, y, "Standardization", "On", "SGD")

    # ---学習実施(学習率:0.01 最大エポック数:20 (特徴量を標準化,シャッフルオン,最小二乗法))
```

```
    Xstd = np.copy(X)
    Xstd[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
    Xstd[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
    ada = Adaline(eta=0.01, numEpoch=20, shuffle=True)
    ada.fitLS(Xstd, y)
    plotResult(ada, Xstd, y, "Standardization", "On", "LS")


def plotResult(ada,X,y,trainSampleState,shuffeled,fitAlgorism):
    #---プロットのプロパティ
    markers = ('s', 'o', 'x', '^', 'v')
    colors = ('green', 'yellow','red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    labels = ('setosa', 'versicolor')

    fig = plt.figure(figsize=(12, 10))
    plt.clf()
    ax1 = fig.add_subplot(221)
    ax2 = fig.add_subplot(222)
    ax3 = fig.add_subplot(223)
    ax4 = fig.add_subplot(224)

    # ---エポック数と重みのプロット
    epochTimes = np.array(range(0, ada.numEpoch + 1))
    ada.W_ = np.array(ada.W_)
    ax1.plot(epochTimes, ada.W_[:, 0].T[0], color="blue", label="w0")
    ax1.plot(epochTimes, ada.W_[:, 1].T[0], color="red", label="w2")
    ax1.plot(epochTimes, ada.W_[:, 2].T[0], color="green", label="w1")
    ax1.legend(loc="upper left")
    ax1.set_xlabel('Epochs')
    ax1.set_ylabel('Weight')
    ax1.set_title("Epochs and weight")
    ax1.set_xlim(0, ada.numEpoch)
    # ---ax1.set_xticks(range(0,numEpoch+1,1))
    ax1.grid()

    # ---エポック数と正答率のプロット
    epochTimes = range(1, ada.numEpoch + 1)
    ax2.plot(epochTimes, np.array(ada.correctAnswerRateEachEpoch_) * 100, color="blue",
marker="o",
             label="Correct answer rate")
    # ---ax2.legend(loc="upper left")
    ax2.set_title("Epochs and correct answer rate")
    ax2.set_xlabel('Epochs')
    ax2.set_ylabel('Correct answer rate[%]')
    ax2.set_xlim(0, ada.numEpoch)
    ax2.grid()
    plt.xlim(0, ada.numEpoch)

    # ---エポック数とコスト関数のプロット
    epochTimes = range(1, ada.numEpoch + 1)
```

```
    ax3.plot(epochTimes, np.array(ada.J_)[:, 0][:, 0], color="blue", marker="o",
             label="Cost function values")
    # ---ax3.legend(loc="upper left")
    ax3.set_title("Epochs and cost function values")
    ax3.set_xlabel('Epochs')
    ax3.set_ylabel('Cost function values')
    ax3.set_xlim(0, ada.numEpoch)
    ax3.grid()
    # ---ax3.set_xticks(range(0,numEpoch+1,1))

    # ---決定領域のプロット
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    dx = 0.02
    X1 = np.arange(x1_min, x1_max, dx)
    X2 = np.arange(x2_min, x2_max, dx)
    X1, X2 = np.meshgrid(X1, X2)
    Z = ada.predict(np.array([X1.ravel(), X2.ravel()]).T)
    Z = Z.reshape(X1.shape)

    # ---決定領域
    ax4.contourf(X1, X2, Z, alpha=0.5, cmap=cmap)
    ax4.set_xlim(X1.min(), X1.max())
    ax4.set_ylim(X2.min(), X2.max())
    # ---アヤメデータ
    for idx, cl in enumerate(np.unique(y)):
        ax4.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=1.0, c=cmap(idx),
                    marker=markers[idx], label=labels[idx])
    ax4.set_title("Decision regions")
    ax4.set_xlabel("Sepal length [cm]")   # がく片の長さ
    ax4.set_ylabel("Petal length [cm]")   # 花びらの長さ
    ax4.legend(loc="upper left")
    ax4.grid()

    pngFileName=u"ADALINE 学習結果_トレーニングサンプル" +trainSampleState +u"_シャッフル"+shuffeled+u"_Fit アルゴリズム"+fitAlgorism+ u"_学習率"+str(ada.eta)+u"_最大エポック数"+str(ada.numEpoch)+ ".png"
    plt.savefig(pngFileName, dpi=300)


class Adaline(object):

    def __init__(self, alpha=1.0, eta=0.01, numEpoch=10, shuffle=True, random_state=None):
        self.alpha = alpha   # 活性化関数の定数
        self.eta = eta #学習率
        self.numEpoch = numEpoch #最大エポック数
        self.shuffle = shuffle #トレーニングサンプルをシャッフルする？
        if random_state:
            np.random.seed(random_state)
```

```
        self.W_=[] #各エポックごとに重みを保存するリスト
        self.J_=[] #各エポックごとにコスト関数を保存するリスト
        self.correctAnswerRateEachEpoch_=[] #各エポックごとに正答率を保存するリスト


    def __actFunc(self, z):
        """活性化関数(__でプライベート関数)"""
        return self.alpha*z

    def __quantizer(self, phi):
        """量子化器(__でプライベート関数)"""
        return np.where(np.array(phi) >= 0.0, 1, -1)

    def __shuffle(self, X, y):
        """トレーニングサンプルのシャッフル(__でプライベート関数)"""
        r = np.random.permutation(len(y))
        return X[r], y[r]

    def predict(self, X):
        """予測関数"""
        X = np.matrix(X)
        m, n = X.shape
        X = np.c_[np.matrix(np.ones((m, 1))), X]
        z=X*self.w_
        phi=self.__actFunc(z)
        return self.__quantizer(phi)

    def fit(self, X, y):
        """バッチ勾配降下法による学習の実施"""
        # ---トレーニングサンプルのシャッフル
        if self.shuffle:
            X, y = self.__shuffle(X, y)
        # ---特徴行列
        X = np.matrix(X)
        m, n = X.shape
        # ---しきい値用に一番左側の列に 1 を追加
        X = np.c_[np.matrix(np.ones((m, 1))), X]
        # ---教師データベクトル
        y = np.matrix(y).T

        # ---重みベクトル(初期化)
        self.w_ =np.matrix(np.zeros((n+1,1)))

        #  重みを学習回数ごとに保存
        self.W_.append(self.w_.tolist())

        for indexEpoch in range(1,self.numEpoch+1): #エポックループ
            correctAnswer=[] #各トレーニングセットに対する教師データと予測値の正誤表をイニシャライズ
```

```
                # ---特徴行列と重みベクトルの掛け合わせ
                z = X * self.w_
                # ---活性化関数出力ベクトル
                phi =self.__actFunc(z)
                # ---教師データと活性化関数出力の残差ベクトル
                e = y - self.alpha * X * self.w_
                # ---コスト関数
                J = e.T * e / 2
                # ---コスト関数の勾配
                gradJ = -self.alpha * X.T * y + self.alpha ** 2 * X.T * X * self.w_
                # ---重み更新
                dw = - self.eta * gradJ
                self.w_ = self.w_ + dw
                #  重みをエポックごとに保存
                self.W_.append(self.w_.tolist())
                #  コスト関数をエポックごとに保存
                self.J_.append(J.tolist())

                #--正答表
                correctAnswer = np.where(np.array(y == self.__quantizer(phi)) == True, 1, 0)
                #各エポックごとに正誤表から正答率算出し保存

self.correctAnswerRateEachEpoch_.append(float(sum(correctAnswer))/len(correctAnswer))


    def fitSGD(self, X, y):
        """確率的勾配降下法による学習の実施"""
        # ---トレーニングサンプルのシャッフル
        if self.shuffle:
            X, y = self.__shuffle(X, y)
        # ---特徴行列
        X = np.matrix(X)
        m, n = X.shape
        # ---しきい値用に一番左側の列に 1 を追加
        X = np.c_[np.matrix(np.ones((m, 1))), X]
        # ---教師データベクトル
        y = np.matrix(y).T

        # ---重みベクトル(初期化)
        self.w_ = np.matrix(np.zeros((n + 1, 1)))

        #  重みを学習回数ごとに保存
        self.W_.append(self.w_.tolist())

        for indexEpoch in range(1, self.numEpoch + 1):  # エポックループ
            correctAnswer = []  # 各トレーニングセットに対する教師データと予測値の正誤表
をイニシャライズ
            for xi, yi in zip(X, y):  # トレーニングセットループ
                # ---コスト関数の勾配
                gradJi = -self.alpha * xi.T * yi + self.alpha ** 2 * xi.T * xi * self.w_
```

```
                # ---重み更新
                dwi = - self.eta * gradJi
                self.w_ = self.w_ + dwi

            # ---特徴行列と重みベクトルの掛け合わせ
            z = X * self.w_
            # ---活性化関数
            phi = self.__actFunc(z)
            # ---教師データと活性化関数出力の残差ベクトル
            e = y - self.alpha * X * self.w_
            # ---コスト関数
            J = e.T * e / 2
            # ---コスト関数の勾配
            gradJ = -self.alpha * X.T * y + self.alpha ** 2 * X.T * X * self.w_
            # # ---重み更新
            # dw = - self.eta * gradJ
            # self.w_ = self.w_ + dw

            #  重みをエポックごとに保存
            self.W_.append(self.w_.tolist())
            #  コスト関数をエポックごとに保存
            self.J_.append(J.tolist())

            # --正答表
            correctAnswer = np.where(np.array(y == self.__quantizer(phi)) == True, 1, 0)
            #  各エポックごとに正誤表から正答率算出し保存
            self.correctAnswerRateEachEpoch_.append(float(sum(correctAnswer)) /
len(correctAnswer))


    def fitLS(self, X, y):
        """最小二乗法による学習の実施"""
        # ---トレーニングサンプルのシャッフル
        if self.shuffle:
            X, y = self.__shuffle(X, y)
        # ---特徴行列
        X = np.matrix(X)
        m, n = X.shape
        # ---しきい値用に一番左側の列に 1 を追加
        X = np.c_[np.matrix(np.ones((m, 1))), X]
        # ---教師データベクトル
        y = np.matrix(y).T

        # ---重みベクトル(初期化)
        self.w_ = np.matrix(np.zeros((n + 1, 1)))

        #  重みを学習回数ごとに保存
        self.W_.append(self.w_.tolist())

        for indexEpoch in range(1, self.numEpoch + 1):  # エポックループ
```

```
                correctAnswer = []  # 各トレーニングセットに対する教師データと予測値の正誤表をイニシャライズ

                # ---コスト関数を最小にする重みを一発で算出
                self.w_ = np.linalg.inv(X.T*X)*X.T*y /self.alpha

                # ---特徴行列と重みベクトルの掛け合わせ
                z = X * self.w_
                # ---活性化関数
                phi = self.__actFunc(z)
                # ---教師データと活性化関数出力の残差ベクトル
                e = y - self.alpha * X * self.w_
                # ---コスト関数
                J = e.T * e / 2

                #  重みをエポックごとに保存
                self.W_.append(self.w_.tolist())
                #  コスト関数をエポックごとに保存
                self.J_.append(J.tolist())

                # --正答表
                correctAnswer = np.where(np.array(y == self.__quantizer(phi)) == True, 1, 0)
                #  各エポックごとに正誤表から正答率算出し保存
                self.correctAnswerRateEachEpoch_.append(float(sum(correctAnswer)) / len(correctAnswer))


if __name__ == "__main__":
    main()
```

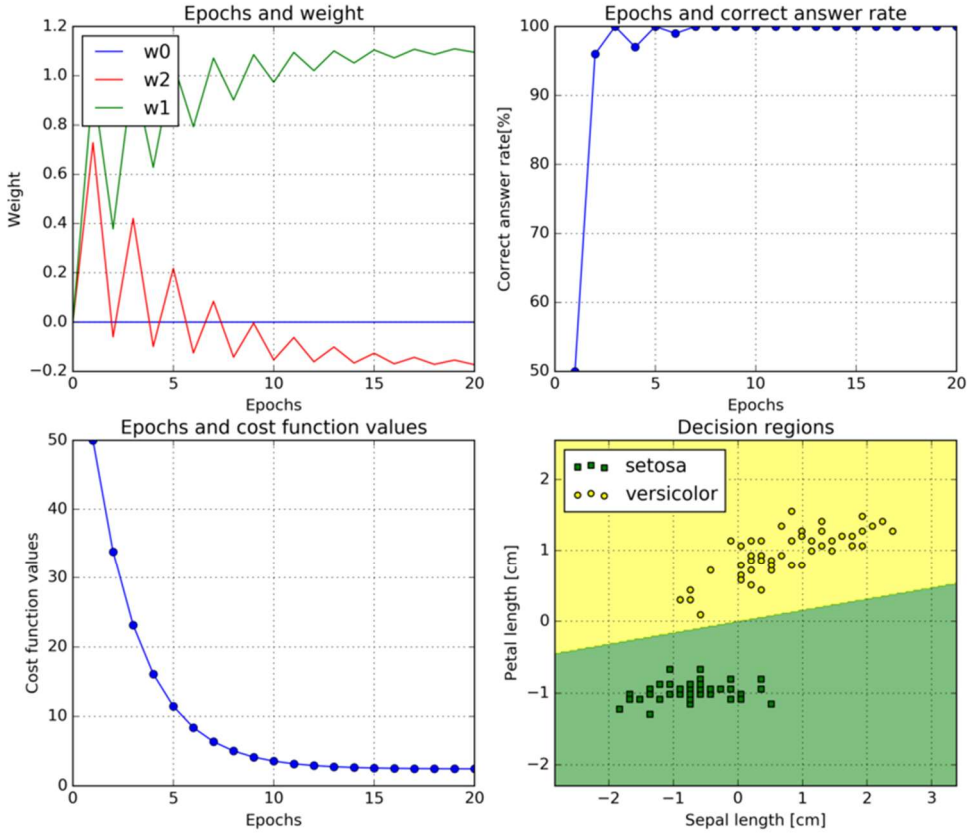
view raw StudyMachineLearning_AdalineSGD.py hosted with ❤ by GitHub

Adaline クラスにトレーニングサンプルをシャッフルするプライベートメソッド__shuffle を追加してあります。
また、確率的勾配降下法による学習実施メソッド fitSGD と、最小二乗法による学習メソッド fitLS を追加しました。
このスクリプトを実行すると以下 4 つの学習結果をグラフで出力します。

（３）シミュレーション例

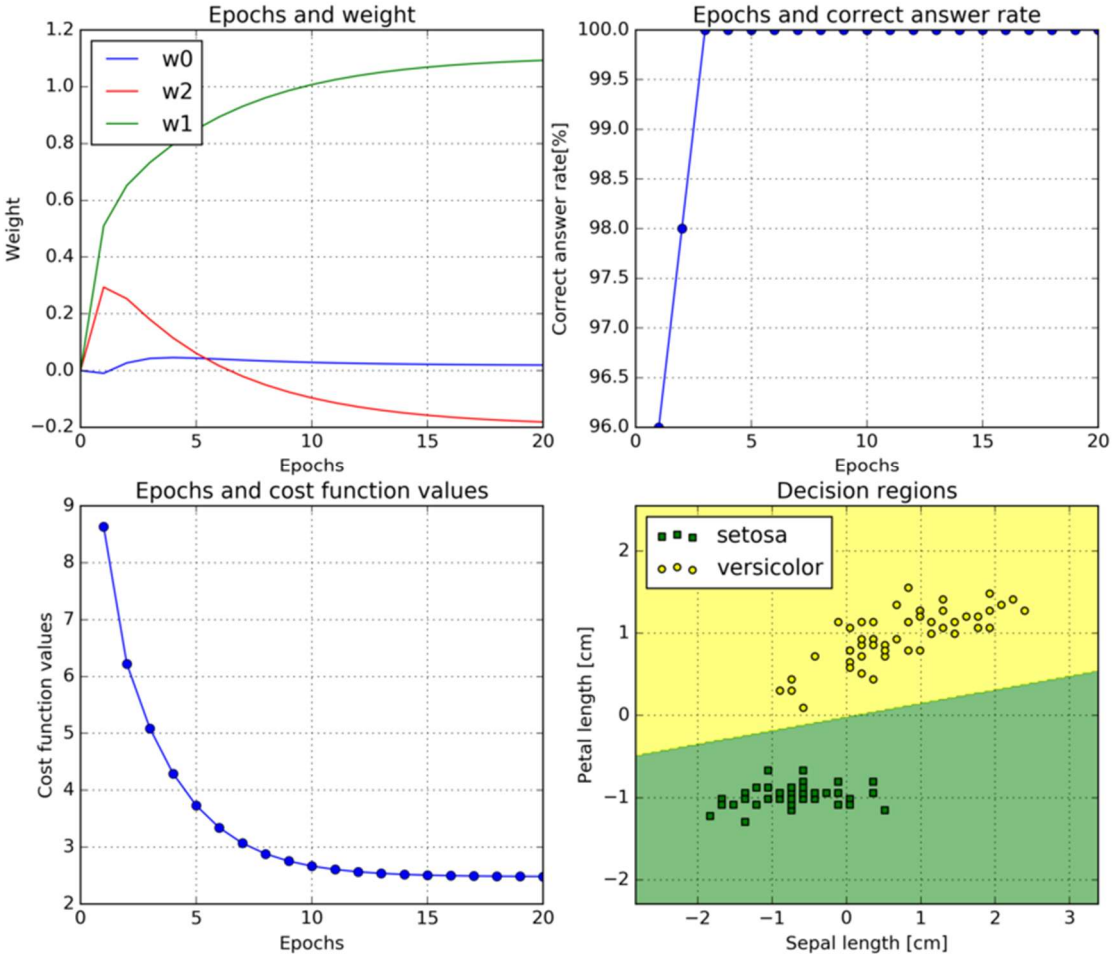
バッチ勾配効果法の結果(学習率:0.01 最大エポック数:20 特徴量を標準化:オン トレーニングサンプルシャッフル:オフ)

バッチ勾配降下法の結果です。比較用に出力しています。



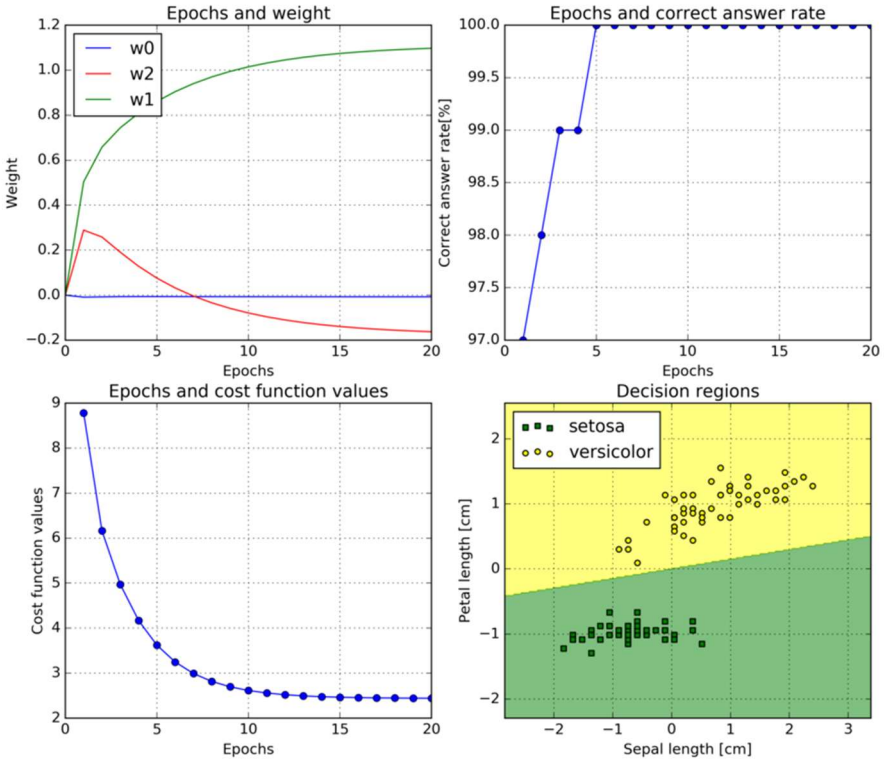
確率的勾配効果法の結果(学習率:0.01 最大エポック数:20 特徴量を標準化:オン トレーニングサンプルシャッフル:オフ)

バッチ勾配効果法に比べ、1 エポック目からコスト関数がずいぶん小さくなっているのがわかります。1 エポック目の正答率も 96%と高くなっています。



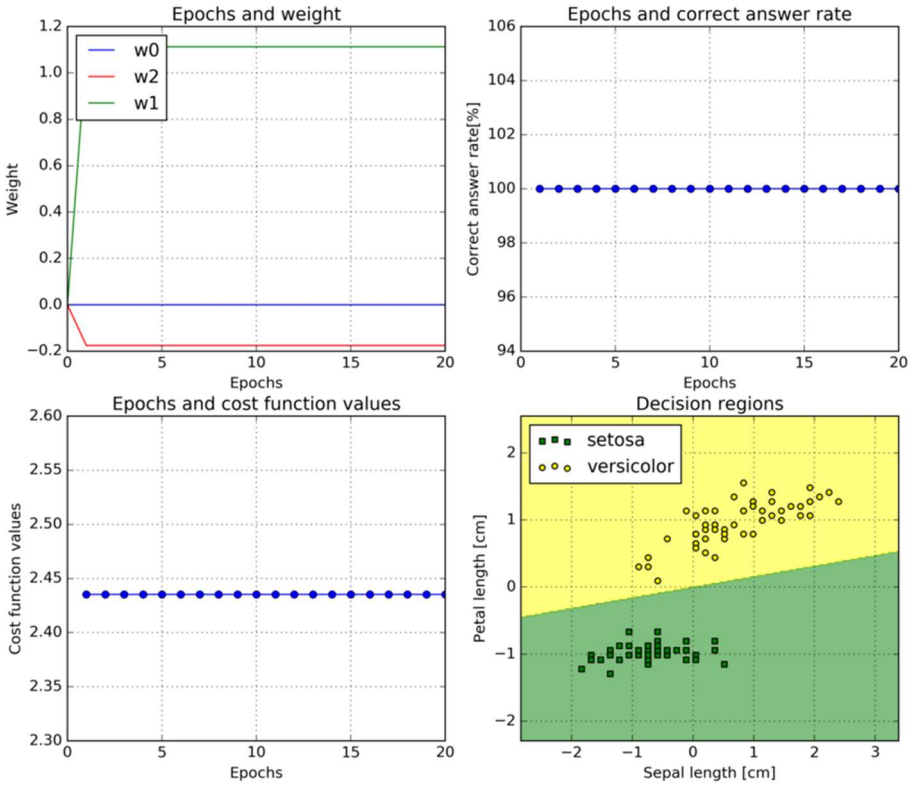
確率的勾配効果法の結果(学習率:0.01 最大エポック数:20 特徴量を標準化:オン トレーニングサンプルシャッフル:オン)

トレーニングサンプルをシャッフルする事でもっと学習効率が上がるかと思いましたが、そんなに大したことなかったですね。でも 1 エポック目の正答率が 97%とシャッフルしなかった時より高くなっているので少しは効果があります。



最小二乗法の結果(学習率:0.01 最大エポック数:20 特徴量を標準化:オン トレーニングサンプルシャッフル:オン)

さすが最小二乗法ですね。
一発で最小となるコスト関数が求まっています。最小二乗法では、学習率、エポック数、特徴量の標準化、トレーニングサンプルのシャッフルに関係なく、一発でコスト関数の最小値が求まります。
ちなみにコスト関数の最小値は 2.435 です。



コスト関数が重み **w** で微分可能で、極小値がただ一つだけ存在する場合は最小二乗法を用いるのがいいです。
しかしながら、特微量行列の転置と特微量行列の積の逆行列$(\mathbf{X}^T\mathbf{X})^{-1}$ を求める必要があるのでトレーニングサンプルが非常に多い場合は、計算コストが増大しメモリ不足で逆行列が計算できなくなってしまう恐れがあります。
実際はトレーニングサンプルが非常に多くなる場合がほとんどで、勾配降下法で逐次的にコスト関数の最小値を求める方がはるかに計算コストが少なくなる為、一般的にはあまり使われていないみたいです。

２　ロジスティック回帰

ロジスティック回帰は ADALINE に確率的な解釈を与えたアルゴリズムです。
活性化関数とコスト関数に対し、ロジスティック関数と尤度を用いています。ADALINE との違いは、この活性化関数とコスト関数だけです。
ここでは、ロジスティック関数と尤度の概念を学んでいきたいと思います。

（１）ロジスティック関数

トレーニングサンプルと重みをかけ合わせた値($\mathbf{z}=\mathbf{X}\mathbf{w}$)を、ロジスティック関数に入力し活性化関数の値として出力します。

$$\phi(z)=\frac{1}{1+e^{-z}}$$

ロジスティック関数 $\phi(z)$ は、－∞～∞までの全実数入力 z に対し、0～1 までの値を出力します。
トレーニングサンプル $\mathbf{X}$ が与えられた場合に、教師データ $\mathbf{y}$ を与える条件付き確率として解釈されます。
確かに、ロジスティック関数の出力は 0～1 までの値になるので確率として解釈できそうですが、その概念は少々難解です。この概念について少し堀り下げてみようと思います。

ロジスティック関数はその形からシグモイド関数とも呼ばれています。－∞～＋∞までの実数入力を受けて、0～1 までの実数を出力する関数です。

今１組のトレーニングサンプル x_i が与えられた時、クラスラベル y_i が 1 となる条件付き確率を p とします。

$$p=p(y_i=1 \mid x_i)$$

この時オッズ比 r を以下のように定義します。オッズ比は[0,1]の定義域をとり、[0,∞)の値域を取ります。

$$r=\left(\frac{p}{1-p}\right)$$

オッズ比の対数を取ったものはロジット関数と呼ばれています。対数を取ることで、[0,1]の定義域に対し、値域が(-∞,+∞)に拡張されます。

$$z=\log\frac{p}{1-p}$$

このロジット関数の逆関数がロジスティック関数となります。ロジスティック関数の定義域は(-∞,+∞)で値域は[0,1]になります。

$$p=\frac{1}{1+e^{-z}}$$

ロジスティック関数の出力値が確率を表すことが、なんとなくわかりました。
ではなぜオッズ比とロジット関数を考える必要があるかというと、定義域が(-∞,+∞)であるロジスティック関数の入力 z を、トレーニングサンプルと重みの積の和 $\mathbf{x}_i\mathbf{w}$ に対応させるためです。

少々強引な気もしますが、$\mathbf{w}$ は、トレーニングサンプル $\mathbf{x}_i$ に対し、クラスラベル $\mathbf{y}_i$ が 1 となる条件付き確率分布関数を決定するパラメータになっていると解釈できるようになるわけです。

（２）ロジスティック関数の微分

コスト関数の勾配を求める際に、ロジスティック関数を微分してやる必要があるので、ここで確認しておきます。

ロジスティック関数 $\phi(z)$ を z で微分してみます。

$$\begin{aligned}\frac{d\phi(z)}{dz} &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{e^{-z}}{\left(1+e^{-z}\right)^2} \\ &= \frac{1}{1+e^{-z}}\left(1-\frac{1}{1+e^{-z}}\right) \\ &= \phi(z)\left(1-\phi(z)\right)\end{aligned}$$

面白い性質を持っていますね。

（３）ベルヌーイ試行

尤度を考える前にベルヌーイ試行というものを知っておいたほうがいいでしょう。
ベルヌーイ試行とはクラスラベルが 1 か 0 の試行で、コイン投げの試行等がそれに当たります。

クラスラベルが $y_i=1$ である確率が ϕ_i であるとき、その条件付き確率の確率分布関数は以下の式で表すことができます。

$p(y_i \mid \phi_i)=\phi_i^{y_i}\ (1-\phi_i)^{1-y_i}$

クラスラベルが 1 の時は、
$p(y_i=1 \mid \phi_i)=\phi_i$

クラスラベルが 0 の時は、
$p(y_i=0 \mid \phi_i)=1-\phi_i$

となり、一つの式で 1 と 0 のクラスラベルの確率分布を同時に表しています。
この式で定義される確率分布関数をベルヌーイ分布といいます。

ここで、ϕ_i という確率は、クラスラベルが $y_i=1$ である確率を表していますが、ϕ_i は確定した量ではなく、パラメータ $\mathbf{w}$ をもつ確率分布関数に従っていることに注意してください。

例えばコイン投げにしても本当に表が出る確率は 1/2 でしょうか？(ひん曲がったコインを投げた場合、表が出る確率が 1/2 かどうかを考えるといいと思います。)

つまり、表が出る確率はあるパラメータ $\mathbf{w}$ を持つ、確率分布関数に従って変動すると考えます。
言い換えると、ここでは表が出る確率が 1/2 である確率。つまり確率の確率を考えています。

パラメータを考慮した場合、ベルヌーイ分布を表す式に以下のようにパラメータを考慮しているということを明示的に書いてやります。

$p(y_i|\phi_i;\mathbf{w})=\phi_i^{y_i}(1-\phi_i)^{1-y_i}$

（４）尤度

尤度とはある事象が観測された時、それぞれの事象の確率の確率分布関数のパラメータがどれだけ尤もらしいかを測る量で、事象の同時確率として定義されます。

$$l(w)=p(y_1|\phi_1;\mathbf{w})p(y_2|\phi_2;\mathbf{w})\cdots\cdots p(y_m|\phi_m;\mathbf{w})=\prod_{i=1}^{m} p(y_i|\phi_i;\mathbf{w})$$

ひん曲がったコインを投げた時、表が出る確率が 1/2 になるかどうか分かりません。
表が出る確率を調べるためには、何回もコインを投げてどれくらいの頻度で表が出るかという事象を調べてやる必要があります。
表が出る確率がベルヌーイ分布に従っていると仮定し、観測された事象からパラメータ **w** を推定する事で、表が出る確率が分かるわけです。

尤度が最大値を与えるパラメータ **w** は、事象の確率が一番尤もらしい値となります。

このように尤度を最大化することで、一番尤もらしい事象の確率を求めてやることを最尤推定法と呼びます。

尤度の対数を取ったものは対数尤度と呼ばれており、尤度が最大になる時、対数尤度も最大値を取ります。

対数法則により掛け算が足し算になるので、対数尤度で評価した方が扱いが簡単になるという利点があります。

$$L(\mathbf{w})=-\sum_{i=1}^{m} \log[\,p(y_i|\phi_i;\mathbf{w})\,]$$

（５）まとめ

ベルヌーイ分布を使ってコスト関数を表してやるので、教師データのクラスラベルは１と０とします。

i 番目のトレーニングサンプルと教師データのセットが得られた時、活性化関数を以下の様に表します。活性化関数の出力は教師データのクラスラベルが 1 である確率を表しています。

$$\phi_i(z_i)=\frac{1}{1+e^{-z}}$$

$$\mathbf{z}_i=\mathbf{x}_i\mathbf{w}$$

コスト関数は対数尤度にマイナスを付けた値を用います。(コスト関数が最小値の時に、対数尤度は最大値となります。)

$$J(\mathbf{w})=-\sum_{i=1}^{m} [\,y_i\log\phi_i+(1-y_i)\log(1-\phi_i)\,]$$

コスト関数を j 番目の重み w_j で微分して、j 番目のコスト関数の勾配を求めてみます。

$$\frac{\partial J(w)}{\partial w_j} = \frac{\partial}{\partial w_j} - \sum_{i=1}^{m} \quad [\, y_i \log\phi_i + (1-y_i)\log(1-\phi_i)\,]$$

$$= -\sum_{i=1}^{m} \quad [\, y_i \quad \frac{\partial \log \phi_i}{\partial \phi_i} \frac{\partial \phi_i}{\partial z_i} \frac{\partial z_i}{\partial w_i} + (1-y_i) \quad \frac{\partial \log(1-\phi_i)}{\partial \phi_i} \frac{\partial \phi_i}{\partial z_i} \frac{\partial z_i}{\partial w_i}\,]$$

$$= -\sum_{i=1}^{m} \quad [\, y_i \quad \frac{1}{\phi_i} \phi_i (1-\phi_i) x_{ij} + (1-y_i) \frac{-1}{(1-\phi_i)} \phi_i (1-\phi_i) x_{ij}\,]$$

$$= -\sum_{i=1}^{m} \quad (y_i - \phi_i) x_{ij}$$

重みの更新は、従来通りコスト関数の勾配に学習率 η をかけ、マイナスをつけて足し込みます。

$$\Delta w_j = -\eta(-\sum_{i=1}^{m} \quad (y_i - \phi_i) x_{ij}) \quad w_j := w_j + \Delta w_j$$

まとめ(行列による表記)

numpy でコーディングしやすいように行列表記しておきます。

トレーニングサンプルと重みの積の和
$\mathbf{z} = \mathbf{X}\mathbf{w}$

活性化関数

$$\phi(z) = \frac{1}{1+e^{-z}}$$

コスト関数
$J(\mathbf{w}) = -\mathbf{y}^T \log\phi - (\mathbf{1}-\mathbf{y})^T \log(\mathbf{1}-\phi)$

コスト関数の勾配
$\partial J(\mathbf{w})/\partial \mathbf{w} = -\mathbf{X}^T(\mathbf{y}-\phi)$

重みの更新
$\Delta\mathbf{w} = -\eta(-\mathbf{X}^T(\mathbf{y}-\phi))$
$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}$

コスト関数の最小値を与える重み

コスト関数の勾配を 0 と置いて式をいじってみたところ、式の上ではコスト関数が最小となる重みが一発で求まりそうだったのでこちらも試してみようかと思います。
$-\mathbf{X}^T(\mathbf{y}-\phi) = 0$ が常に成り立つためには、$\mathbf{y} = \phi$ です。

ϕ を与える $\mathbf{z}$ は、ロジット関数より求まるので、

$$z = \log\frac{\phi}{1-\phi}$$

式を展開します。
$\mathbf{Xw}=\log\phi-\log(\mathbf{1}-\phi)$

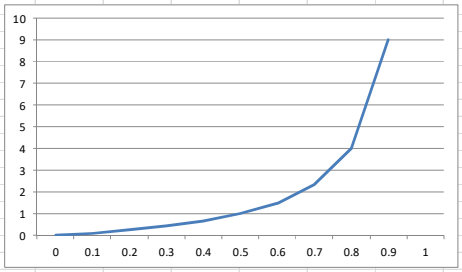
両辺に $\mathbf{X}^{\mathrm{T}}$ をかけ、$\phi=\mathbf{y}$ より、ϕ を消去します。

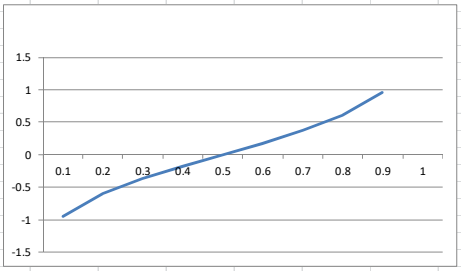
$\mathbf{X}^{\mathrm{T}}\mathbf{Xw}=\mathbf{X}^{\mathrm{T}}(\log\mathbf{y}-\log(\mathbf{1}-\mathbf{y}))$

両辺に$(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ をかけてやると、
$\mathbf{w}=(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}(\log\mathbf{y}-\log(1-\mathbf{y}))$
ADALINE の最小二乗法の時と似てますね。

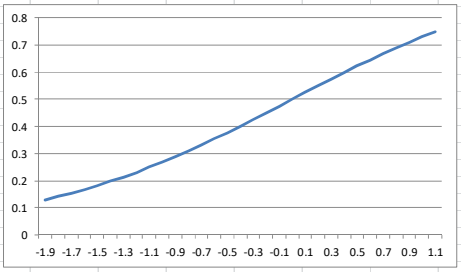
３　ロジスティック関数

p	1−p	p/(1−p)	log(p/(1−p))
0	1	0	
0.1	0.9	0.111111	−0.95424
0.2	0.8	0.25	−0.60206
0.3	0.7	0.428571	−0.36798
0.4	0.6	0.666667	−0.17609
0.5	0.5	1	0
0.6	0.4	1.5	0.176091
0.7	0.3	2.333333	0.367977
0.8	0.2	4	0.60206
0.9	0.1	9	0.954243
1	0		



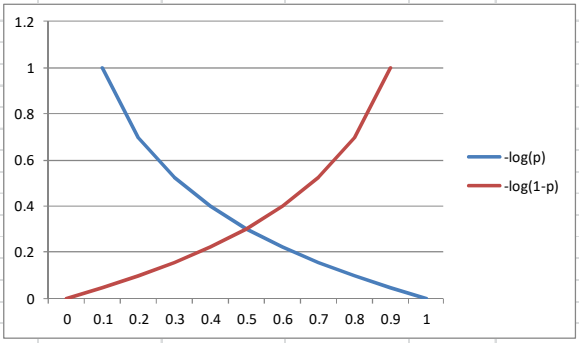


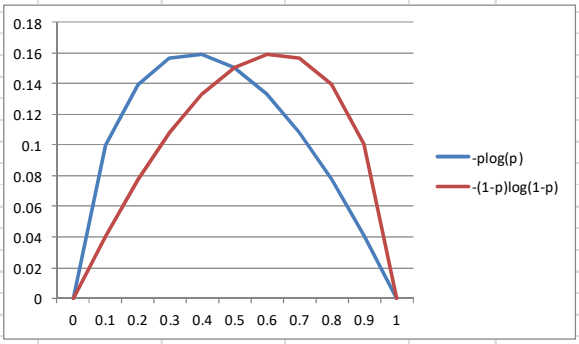
−1.9	0.130108
−1.8	0.141851
−1.7	0.154465
−1.6	0.167982
−1.5	0.182426
−1.4	0.197816
−1.3	0.214165
−1.2	0.231475
−1.1	0.24974
−1	0.268941
−0.9	0.28905
−0.8	0.310026
−0.7	0.331812
−0.6	0.354344
−0.5	0.377541
−0.4	0.401312
−0.3	0.425557
−0.2	0.450166
−0.1	0.475021
0	0.5
0.1	0.524979
0.2	0.549834
0.3	0.574443
0.4	0.598688
0.5	0.622459
0.6	0.645656
0.7	0.668188
0.8	0.689974
0.9	0.71095
1	0.731059
1.1	0.75026

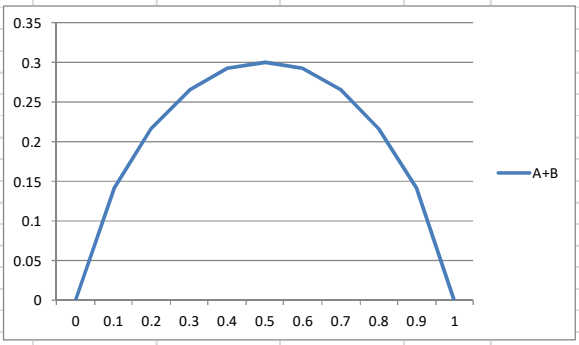


４　交差エントロピー

p	1−p	−log(p)	−log(1−p)	−plog(p)	−(1−p)log(1−p)	A+B
0	1		0	0	0	0
0.1	0.9	1	0.045757491	0.1	0.041182	0.141181742
0.2	0.8	0.698970004	0.096910013	0.139794001	0.077528	0.217322011
0.3	0.7	0.522878745	0.15490196	0.156863624	0.108431	0.265294996
0.4	0.6	0.397940009	0.22184875	0.159176003	0.133109	0.292285253
0.5	0.5	0.301029996	0.301029996	0.150514998	0.150515	0.301029996
0.6	0.4	0.22184875	0.397940009	0.13310925	0.159176	0.292285253
0.7	0.3	0.15490196	0.522878745	0.108431372	0.156864	0.265294996
0.8	0.2	0.096910013	0.698970004	0.07752801	0.139794	0.217322011
0.9	0.1	0.045757491	1	0.041181742	0.1	0.141181742
1	0	4.82164E−17		4.82164E−17	0	4.82164E−17







1.3.1.1. Numpy や numpy array って何¶

1.3.1.1.1. numpy array オブジェクト¶

Python objects:
- 高レベルの数オブジェクト: 整数、浮動小数点
- コンテナ: リスト(コストレスな挿入と追加)、辞書(高速な検索)

Numpy provides:
- 多次元 array 用の Python 拡張パッケージ
- ハードウェア寄り(効率)
- 科学技術計算のために設計(便利さ)
- **配列志向コンピューティング** としても知られています

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

例えば、配列には以下のようなものを入れます: 離散時間ステップでの実験/シミュレーションの値 ...

ちなみに

例えば、配列には以下のようなものを入れます:
- 離散時間ステップでの実験/シミュレーションの値
- 測定装置に記録された信号、例 音波
- 画像のピクセル、グレーレベルや色
- 異なる X-Y-Z 位置で測定された 3 次元データ、例 MRI スキャン
- ...

どうして便利なの: メモリ効率のよいコンテナが高速な数値操作を提供している。

```
In [1]: L = range(1000)

In [2]: %timeit [i**2 for i in L]
1000 loops, best of 3: 403 us per loop

In [3]: a = np.arange(1000)

In [4]: %timeit a**2
100000 loops, best of 3: 12.7 us per loop
```

1.3.1.1.2. Numpy の参考ドキュメント¶

- web では http://docs.scipy.org/
- 対話的なヘルプ:

```
In [5]: np.array?
String Form:<built-in function array>
Docstring:
array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0, ...
```

- 何か探す:

```
>>> np.lookfor('create array')
Search results for 'create array'
--------------------------------
numpy.array
    Create an array.
numpy.memmap
    Create a memory-map to an array stored in a *binary* file on disk.
In [6]: np.con*?
```

```
np.concatenate
np.conj
np.conjugate
np.convolve
```

1.3.1.1.3. インポートするときの決まり¶

numpy を import するときには次の決まりですることを推奨します:

```
>>> import numpy as np
```

1.3.1.2. 配列の作成¶

1.3.1.2.1. 配列の手動作成¶

- 1 次元:

```
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
>>> a.ndim
1
>>> a.shape
(4,)
>>> len(a)
4
```

- 2 次元、3 次元、...:

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]])      # 2 x 3 array
>>> b
array([[0, 1, 2],
       [3, 4, 5]])
>>> b.ndim
2
>>> b.shape
(2, 3)
>>> len(b)       # returns the size of the first dimension
2

>>> c = np.array([[[1], [2]], [[3], [4]]])
>>> c
array([[[1],
        [2]],

       [[3],
        [4]]])
>>> c.shape
(2, 2, 1)
```

練習問題: 単純な配列

- 単純な二次元配列を作りましょう。まず、上の例を再現しましょう。そして次に自分で考えて作ってみましょう: 最初の列に奇数を逆向きに、次の列に偶数を入れてみたものだとどうなりますか。
- len() や numpy.shape() をそれらの配列に対して使ってみましょう。それらの関数はお互いどう関係しているでしょうか。そして配列の ndim 属性についてはどうでしょうか。

1.3.1.2.2. 配列を作るための関数¶

ちなみに

実際には、項目一つ一つを入れることは少ない...

- 1 つとばしの値:

```
>>> a = np.arange(10) # 0 .. n-1  (!)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b = np.arange(1, 9, 2) # start, end (exclusive), step
>>> b
array([1, 3, 5, 7])
```

- または点の数を指定:

```
>>> c = np.linspace(0, 1, 6)   # start, end, num-points
>>> c
array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ])
>>> d = np.linspace(0, 1, 5, endpoint=False)
>>> d
array([ 0. ,  0.2,  0.4,  0.6,  0.8])
```

- よく使う配列:

```
>>> a = np.ones((3, 3))  # reminder: (3, 3) is a tuple
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> b = np.zeros((2, 2))
>>> b
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> c = np.eye(3)
>>> c
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> d = np.diag(np.array([1, 2, 3, 4]))
>>> d
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

- **np.random**: 乱数 (Mersenne Twister 疑似乱数生成器):

```
>>> a = np.random.rand(4)          # uniform in [0, 1]
>>> a
array([ 0.95799151,  0.14222247,  0.08777354,  0.51887998])

>>> b = np.random.randn(4)         # Gaussian
>>> b
array([ 0.37544699, -0.11425369, -0.47616538,  1.79664113])
```

```
>>> np.random.seed(1234)          # Setting the random seed
```

練習問題: 関数を使って配列を作る

- arange, linspace, ones, zeros, eye そして diag を使って実験します。
- 乱数を使っていろいろな種類の配列を作りましょう。
- 乱数を使って配列を作る前に乱数の seed を設定しましょう。
- np.empty 関数を使ってみてみましょう。この関数は何をしますか。どういうときに便利でしょうか。

1.3.1.3. 基本的なデータ型¶

気づいているかもしれませんが、いくつかの例で配列の要素にドットがついて表示されているものがあります(例 2. vs 2)。これはデータ型の違いによるものです:

```
>>> a = np.array([1, 2, 3])
>>> a.dtype
dtype('int64')

>>> b = np.array([1., 2., 3.])
>>> b.dtype
dtype('float64')
```

ちなみに異なるデータ型を使うことでデータをメモリ上にコンパクトに蓄えることができますが、ほとんどの場合はただ単に浮動小数点数を扱いたいだけです。上の例にあるように NumPy は入力から自動的にデータ型を判別します。

明示的にデータ型を指定したい場合はこうします:

```
>>> c = np.array([1, 2, 3], dtype=float)
>>> c.dtype
dtype('float64')
```

デフォルト のデータ型は浮動小数点数です:

```
>>> a = np.ones((3, 3))
>>> a.dtype
dtype('float64')
```

別の型もあります:

Complex:

```
>>> d = np.array([1+2j, 3+4j, 5+6*1j])
>>> d.dtype
dtype('complex128')
```

Bool:

```
>>> e = np.array([True, False, False, True])
>>> e.dtype
dtype('bool')
```

Strings:

```
>>> f = np.array(['Bonjour', 'Hello', 'Hallo',])
>>> f.dtype     # <--- strings containing max. 7 letters
dtype('S7')
```

Much more:

- int32
- int64
- uint32
- uint64

■[Python]Python 基礎文法最速マスター★★★★★30★

↓に便乗して Python 版も書いてみました。

- Perl 基礎文法最速マスター - Perl 入門～サンプルコードによる Perl 入門～
- Ruby 基礎文法最速マスター - Route 477
- PHP 基礎文法最速マスター - Shin x blog

ほとんど上記の記事と同じような内容で書いたので Python 入門記事としては色々抜けていたりしますがご了承ください。

Python は現在 3.x 系がリリースされていますが本記事では基本的に Python2.6 について書きます。
参考文献:

- 初めての Python (asin:4873113938)
- Python Documentation Index http://www.python.org/doc/
- Python 和訳 Document http://docs.python.jp/2/

0. 対話環境として使う

対話環境

python はそのまま実行すると対話環境として働きます。

```
% python
(中略)
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello'
hello
>>> 1 + 2
3
```

help()に関数やその他のオブジェクトを渡すことでヘルプが(あれば)見られます。q をタイプするとヘルプを抜けます。

```
>>> help(1)
Help on int object:

class int(object)
 |  int(x[, base]) -> integer
(以下略)
```

dir()にオブジェクトを渡すことでオブジェクトの属性名一覧が返されます。

```
>>> dir(1)
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__', '__delattr__',
'__div__', '__divmod__', '__doc__', '__float__', '__floordiv__', '__format__',
'__getattribute__', '__getnewargs__', '__hash__', '__hex__', '__index__', '__init__',
'__int__', '__invert__', '__long__', '__lshift__', '__mod__', '__mul__', '__neg__',
'__new__', '__nonzero__', '__oct__', '__or__', '__pos__', '__pow__', '__radd__',
```

'__rand__', '__rdiv__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__',
'__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__rpow__',
'__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__',
'__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__',
'__xor__', 'conjugate', 'denominator', 'imag', 'numerator', 'real']
どんな関数があるかを調べるのに使ったりしますが、標準のままではだいぶ見辛いので外部モジュールの see の導入をおすすめします。

-
- ライブラリ:see - Life with Python
- see()

type()でオブジェクトの型を調べます。
```
>>> type('hoge')
<type 'str'>
```
暗黙の変数 _ に一つ前に入力した式の結果が代入されます。
```
>>> 1 + 2
3
>>> _ + 10
13
>>> result = _ + 100
>>> _
13 # 代入文では _ の値は更新されない
```
exit()を呼び出すと終了します。Ctrl-D でもいいですが端末によるかもしれません。
より高機能な python 対話環境として IPython や Jupyter がありますがここでは説明を省略します。

0.5. プログラムの文字コード

最近の環境では UTF-8 あたりに統一した方がよさそうです。
ファイルの 1 行目か 2 行目に
```
# -*- coding: UTF-8 -*-
```
とすることでプログラムの文字コードを指定します。これが無いと日本語のコメントや文字列リテラルを使った際にエラーが出ます。
もちろん、必要があれば UTF-8 以外を指定することもできます。

1. 基礎

ライブラリのインポート

```
import math

#math モジュールの要素を使う
print math.pi # 3.141592653589793
```
import 文で標準ライブラリや外部のライブラリを読み込みます。
指定した名前のライブラリが無い場合は ImportError 例外が発生します。

from 形式を用いてローカルスコープにコピーすることもできます。同時に別名を付けることもできます。

```
from math import e
print e # 2.718281828459045
from math import e as napiers_constant
print napiers_constant # 同様
from math import * # 全部コピー(作法としてはよろしくないかも)
```

print 文

print 文です。自動的に末尾に改行が付加されます。
カンマで区切って複数いっぺんに出力もできます。この場合、間に半角空白が挿入されます。

```
print 'Hello world'
print 'hoge', 'moge', 'fuga'
print >>sys.stderr, 'nurupo' #標準エラー出力
```

※Python3 系では、print は文(statement)ではなく組み込み関数になりました。

```
# Python3
print('Hello world')
print('hoge', 'moge', 'fuga')
print('nurupo', file=sys.stderr) #標準エラー出力
```

入力

```
raw_input()
input()
```

raw_input()は入力された文字列をそのまま返します。
input()は eval(raw_input())と等価であり、入力文字列を Python の式として解釈した結果を返すため、入力文字列が信頼できない状況では使ってはいけません。(参考→Wikipedia:ja:eval#セキュリティ上のリスク)
※Python3 系では input()が 2 系の raw_input()相当となり、raw_input()は廃止されました。

コメント

一行コメントです。#から行末までがコメントアウトされます。

```
# コメント
```

複数行コメントのかわりとして 3 重クオート文字列を使う場合があります。後述。

```
"""
複数行
コメント
"""
```

変数の宣言

Python には変数の宣言はありません。変数に代入する型も制限されません。

```
a = 1
a = 'hoge' # a を上書き
```

スクリプトの実行

```
% python hoge.py
```

ワンライナー実行

```
% python -c "print 'Hello'"
```

バージョンの確認

```
% python -V # あるいは --version オプション
```

2. 数値

整数や小数、複素数が使えます。末尾に j を付与すると虚数を表します。

```
num = 1
num = 3.14
num = 1 + 1j
```

各種演算

```
num = 1 + 1    # 2
num = 1 - 1    # 0
num = 1 * 2    # 2
num = 5 / 2    # 2（整数どうしの除算は整数に切り捨てられる）
num = 5.0 / 2  # 2.5（どちらかが小数なら結果も小数）
num = 5 % 2    # 1（剰余をとる）
num = 2 ** 3   # 8（累乗）
```

python3 では整数/整数の演算も小数が帰ってくるようになりました。
従来の整数に切り捨てる演算がしたいときは//演算子を使いましょう。(//演算子は python2.6～でも使えます)

```
# python3
5 / 2 # 2.5
5 // 2 # 2
```

インクリメントとデクリメント

Python には++演算子がないので、+=や-=を使います。

```
i += 1
i -= 1
```

論理演算

Python の論理演算は以下のとおりです。or や and は True/False を返すわけではないことに注意。

```
x or y  # x が偽なら y 、そうでなければ x
x and y # x が偽なら x 、そうでなければ y
not x   # x が偽なら True 、そうでなければ False
```

3. 文字列

文字列の表現

通常の文字列はシングルクォートかダブルクォートで囲みます。
また、'''～'''、"""～"""で複数行にわたる文字列を構成できます。クォートに囲まれている範囲では改行がそのまま扱われます。

```
str1 = 'abc'
str2 = "def"
str3 = '''ghi
jkl
mno'''
```

3 重クォート文字列を変数に代入せずに記述し、複数行コメントのようにして使うこともあります。

```
    """
    複数行
    コメント
    """
```

C 言語でいうところの sprintf 的な操作は%演算子、あるいは format()関数で行います。

```
str4 = '%s def' % str1              # 'abc def'
str5 = '%s %s' % (str1, str2)       # 'abc def'
str6 = '{1} {0}'.format(str1, str2) # 'def abc'
```

文字列操作

各種文字操作です。

```
# 結合
join1 = 'hoge' + 'moge' # 'hogemoge'
join2 = ','.join(['aaa', 'bbb', 'ccc']) # 'aaa,bbb,ccc'

# 分割
record1 = 'aaa bbb  ccc'.split() # ['aaa', 'bbb', 'ccc'] デフォルトでは空白で分割
record2 = 'aaa,bbb,ccc'.split(',') # ['aaa', 'bbb', 'ccc']

# 長さ(文字数)
length = len('Supercalifragilisticexpialidocious') # 34

# 切り出し
# [start:stop]の形で指定して切り出す
substr1 = '0123456789'[:3]  # '012'  0～stop-1 を切り出す
substr2 = '0123456789'[3:6] # '345'  start～stop-1 を切り出す
substr3 = '0123456789'[6:]  # '6789' start～末尾を切り出す

# 検索
result = 'abcd'.find('cd') # 見つかった場合はその位置、見つからなかった場合は-1 が返る
```

4. リスト、タプル

リスト
: 他言語でいう配列のような型です。要素としてどんな型のものでも含むことができます。

タプル
: 「オブジェクトの組」という点ではリストに似ていますが、要素の追加や削除はできません。関数から複数の値を返したいとき等に多用されます。

```
# 空リストの作成
list1 = []
list2 = list()
# 複数要素からなるリストを作成
list3 = [0, 1, 2]
# 他のリストの(浅い)コピーを作成
list4 = list(list3)
# 連番のリストを作成
list5 = range(3) # [0, 1, 2] range(stop)は[0, 1, ..., stop-1]を返す
list6 = range(3, 6) # [3, 4, 5] range(start, stop)は[start, start+1, ..., stop-1]を返す
list7 = range(1, 8, 2) # [1, 3, 5, 7] 3番目の引数で値の増減量を指定できる

# 空タプルの作成
tuple1 = ()
tuple2 = tuple()
# 要素が1つのタプルの作成
tuple3 = (1,)
# 複数要素からなるタプルを作成
tuple4 = (0, '1', 2, [3], -4)
```

要素の参照と代入

リスト、タプルは0から始まる添字で要素を参照できます。また、タプルは要素を新たに代入できません。

```
# 要素の参照
list2[1] # 1
list2[-1] # 2 インデックスに負数を指定することで後ろから辿れる
tuple4[2] # 2

# 要素の代入
list2[0] = 100
tuple4[0] = 1 # エラー
```

タプル/リストの要素数

```
len(list3) # 5
len(tuple4) # 5
```

リストの操作

```
array = [1, 2, 3, 4]

# 任意の要素を取り出す
first = array.pop(0) # first == 1, array == [2, 3, 4]
# 任意の要素を追加する
array.insert(0, 5) # array == [5, 2, 3]
# 末尾を取り出す
last = array.pop() # last == 3, array = [5, 2]
# 末尾に追加
```

```
array.append(9) # array == [5, 2, 9]
# 末尾にリストを追加
array.extend([0, 1]) # array == [5, 2, 9, 0, 1]
```

5. ディクショナリ

他の言語でハッシュや連想配列、マップ等と呼ばれているものと似たようなものです。
ディクショナリのキーには、数値、文字列、タプルをはじめとした不変オブジェクトが指定できます。

```
dic = {} # 空ディクショナリ
dic = dict() # 空ディクショナリその 2
dic = {'a': 1, 'b': 2} # key:value, ... の形式で初期化
dic = dict(a=1, b=2) # キーワード引数による初期化
```

ディクショナリの要素の参照と代入

```
# 要素の参照
dic['a'] # 1

# 要素の代入
dic['a'] = 100
dic['c'] = 42
```

ディクショナリに関する関数

```
# キーの取得
dic.keys() # ['a', 'c', 'b'] (辞書式順や追加順に並ぶとは限らない)
# 値の取得
dic.values()
# キーの存在確認
'a' in dic # True
# ハッシュのキーの削除
del dic['a']
```

6. 制御文

Python は一般的なプログラミング言語と異なり、タブやスペースによるインデント(字下げ)が意味を持ちます。
同じ深さのインデントは同じブロックに属するものとみなされます。

if 文

```
if 条件:
    print 'true'
```

if～else 文

```
if 条件:
```

```
    print 'true'
else:
    print 'false'
```

if〜elif 文

他の言語のように else if ではなく elif です。

```
if 条件 1:
    print 'if1'
elif 条件 2:
    print 'if2'
```

while 文

```
i = 0
while i < 5:
    # 処理
    i += 1
```

いわゆる無限ループには while 1:が主に使われます。

for 文

Java の foreach のような動作です。

```
for i in [0, 1, 2, 3, 4]:
    # in の右に渡されたリスト等の要素が i に順次代入されて処理される
    print i
```

今何番目の要素かを取っておきたい場合は、組み込みの enumerate()と合わせて使うとよいでしょう。

```
for i, j in enumerate(['zero', 'one', 'two', 'three']):
    print i, j
```

while-else, for-else

while ブロックや for ブロックの後に else ブロックを付加できます。このブロック内の演算は、break 以外でループを抜けた時に実行されます。

```
for i in ['hoge', 'moge', 'fuga']:
    print i
else:
    print 'else' # 出力される

i=1
while i <= 0:
    i += 1
else:
    print 'else' # 出力される

i=1
while i:
    i += 1
```

```
    if i>10:
        break
else:
    print 'else' # 出力されない
```

複数回繰り返し

```
for i in xrange(1000000):
    print 'SPAM ' # 1000000 回繰り返し
```

xrange()は range()と似ていますが、最初からリスト全体を生成するわけではないのでこういったループ目的に多用されます。(※Python3 系では range が xrange 相当になっています。配列を最初から生成する range はなくなりました)

7. 関数

関数は基本以下のようにして記述します。返り値は return 文で返します。(return 文がない場合は None が返ります)

```
def add1(num1, num2):
    return num1 + num2

add1(1, 2) # 3
```

引数に*や**を添えることで可変長引数を実現できます。
収まりきらなかった引数はタプルとして*付き引数に、キーワード引数はディクショナリとして**付き引数に格納されます。
*、**付き引数の名前はなんでも使えますが*args, **kwargs が主流のようです。

```
def f(arg, *args, **kwargs):
    print arg
    print args
    print kwargs

f(1, 2, 3, four=4, five=5,six=6)
1
(2, 3)
{'four': 4, 'six': 6, 'five': 5}
```

lambda 式で無名関数を作成できます。lambda (引数): (python 式)の形で表現します。

```
add2 = lambda x, y: x + y
add2(1, 2) # 3
```

lambda による無名関数は、小規模な関数を作成して使用したり他の関数の引数とするのに向いています。
無理に lambda で関数を記述しようとすると可読性も落ちるので、ある程度以上規模の大きい関数は通常の形式で記述したほうがよいです。

7.5. 関数の呼び出し方・補足

引数にデフォルト値を指定できます。デフォルト値が指定された引数は省略できます。
ただし、デフォルト値がある引数の後にデフォルト値のない引数を記述すると文法エラーとなります。

```
def emphasis(text, decoration='*'):
    return decoration + text + decoration
```

```
emphasis('hello') # *hello*

def f(a, b=1, c):
    return a + b + c
```

SyntaxError: non-default argument follows default argument
よくある落とし穴として、デフォルト値は都度初期化されるのではなく 1 回のみ初期化されて使いまわされるため、デフォルト値に可変オブジェクトを置いてしまうというミスがあります。

```
def my_append(a, array=[]):
    array.append(a)
    return array

my_append(4, [1, 2, 3]) # [1, 2, 3, 4]
my_append(1) # [1]
my_append(2) # [2] を期待したが [1, 2] が返される
```

関数を呼び出す際に、仮引数の名前を用いて呼び出す方法があります。

```
emphasis('hello', decoration='***') # ***hello***
```

関数に渡す引数にリテラルが多くてわかりにくい時に便利です。また、デフォルト値付き引数が複数ある時に途中の値のみ値を指定したいときにも使えます。
よく使うところではリストの sort メソッドにキー関数を指定する場合など。

```
help([].sort)
sort(...)
    L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
    cmp(x, y) -> -1, 0, 1

L = ['1', '8', '4', '6', '5', '9', '7', '2', '3', '10']
L.sort() # L == ['1', '10', '2', '3', '4', '5', '6', '7', '8', '9']
L.sort(key=int) # L == ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
```

関数にリストやタプル、ディクショナリを用いて引数を渡す方法もあります。引数の形式に合わないと TypeError 例外になります。

```
args = ['hello', '*']
emphasis(*args) # *hello*

kwargs = {'text': 'bye', 'decoration': '!!!'}
emphasis(**kwargs) # !!!bye!!!
```

8. ファイル入出力

ファイルコピーの(ナイーブな)例を示します。*1

```
# ファイルを読み込みモードでオープン
orig = open('orig.txt')
# ファイルを書き込みモードでオープン
copy = open('copy.txt', 'w')
for line in orig:
    # line は改行文字を含む
    copy.write(line)

orig.close()
```

```
copy.close()
```

file()組み込み関数によるファイルのオープンは python 2.5 から非推奨です。3.x 系にはもはや存在しません。

9. 雑多な tips

Python の真偽値

python において、None、False、ゼロとして解釈できる数値(0, 0.0, 0j)、空文字列(''')、空リスト、空タプル、空ディクショナリは偽と評価されます。
また、XXX.__len__()が 0 を返すようなオブジェクト、XXX.__nonzero__()が False を返すようなオブジェクトも偽と評価されます。
その他は真と評価されます。

切り捨て

```
num = int(1.5) # 1
```

小数点以下 n 桁目で丸める

```
num = round(3.14159, 3) # 3.142
num = round(1250, -2)   # 1300.0  桁数に負数も指定できる
```

文字→数値

```
# 10 進表記で変換
int('100') # 100
# 基数を指定して変換
int('FFFF', 16) # 65535
# 小数
float('1.0') # 1.0
```

数値→文字

```
str(1234) # '1234'
```

文字列→配列

```
list('hoge') # ['h', 'o', 'g', 'e']
```

文字列をそのまま出力したい

print 文を使うと空白や改行が付加されて出力されますが、それを望まない場合。
sys.stdout.write('hogehoge')

```
print('hogehoge', end='') #Python3
```

コマンドライン引数

sys.argv にリスト形式で格納されています。sys.argv[0]はスクリプト自身を指します。

```
print sys.argv
```

後置の if～else

以下のように 3 項演算子のような書き方ができます。

```
result = 'true_value' if 条件 else 'false_value'
```

クラス定義

クラスは class 名前:で定義します。

```
class Person():
    def __init__(self, name, age):
        self.name, self.age = name, age
        self.address = None

jhon = Person('Jhon', 15)
print jhon.name
jhon.address = 'USA, Earth'
```

map

map を使ってリスト等の各要素を変換できます。リスト内包表記も使えます。

```
map1 = map(str, range(5)) # ['0', '1', '2', '3', '4']
map2 = [str(x) for x in range(5)] # ['0', '1', '2', '3', '4']
```

filter

filter を使ってリスト等の各要素から条件に合うものを抽出したリストを生成できます。リスト内包表記も使えます。

```
filter1 = filter(lambda x: 'cat' in x, ['cat', 'dog', 'catalog']) # ['cat', 'catalog']
filter2 = [x for x in ['cat', 'dog', 'catalog'] if 'cat' in x]
```

タプル代入、リスト代入

タプル代入、リスト代入と呼ばれる代入方法があります。

```
(a, b) = (1, 2) # a=1, b=2
[c, d] = [a, b] # c=1, d=2
a, b = b, a # a, b の値を交換
```

タプルやリストとして評価されるような式を右辺にして用いると便利です。

```
year, month, day = '2010/01/26'.split('/')
```

例外処理

raise で例外を投げ、try～except(～finally)文で捕捉します。

```
try:
    #何かの処理
    if 条件:
        raise Exception();
except Exception, e:
    # 捕捉した Exception 型の例外は e に代入される
    (略)
finally:
    # 例外の有無に関わらず実行されるブロック(必須ではない)
    (略)
else:
    # 例外が発生しなかった場合のみ実行されるブロック(必須ではない)
```

except 節は 2.6 系から新しい表記ができるようになりました。

```
except Exception as e:
    # 捕捉した Exception 型の例外は e に代入される
```

as を用いた方が「この変数に代入する」という意図が明確に表現されているように思えるのでこちらを使うべきでしょう。Python3 では as を用いない形式は文法エラーになります。

[Python] クラスに入門（宣言、インスタンス変数/メソッド、クラス変数/メソッド、スタティックメソッド、継承）
https://flic.kr/p/cYKmtC

目次

クラスの基礎

Python は他の多くの言語と同じく、クラスを定義して使うことができます。具体的には以下のように定義します。

```
# クラスの定義
class MyClass(object):
    pass

# クラスからインスタンスを生成
me = MyClass()
print(me) # => <__main__.MyClass object at 0x100799b00>
```

これで何も要素を持たないクラスを定義できました。ちなみにこの「何も要素を持たない」クラスは以外と便利なのです。

```
# 何も要素を持たないクラス
class EmptyClass(object):
    pass

# 要素を好きなように定義できるので、複数の変数をまとめた入れ物として扱える
holder = EmptyClass()
holder.name = "Yohei"
holder.age = 30
print(holder.name, holder.age) # => Yohei 30
```

ただ、普通はクラスを何か決まった用途に使うので、後述の方法で利用します。

インスタンス変数とインスタンスメソッド

Python では、インスタンス変数とインスタンスメソッドを以下のように定義します。

```
class MyClass(object):
    # クラス直下に定義した変数はクラス変数
    # インスタンス変数ではなく、Java をやっていた身としては混乱しやすい
    # some_field = "aaa"
```

```
    # インスタンス生成時に、値を渡すことができる
    def __init__(self, name, age):
        # 「self.xxx」でインスタンス変数の参照/代入ができる
        self.name = name
        self.age = age

    # インスタンスメソッドの定義
    def introduce(self):
        print("My name is %s, %d years old." % (self.name, self.age))

# インスタンス化
me = MyClass('Yohei', 30)
me.introduce() # => My name is Yohei, 30 years old.

# 直接値を代入することもできる
me.name = "Taro"
me.age = 25
me.introduce() # => My name is Taro, 25 years old.
```

と、こんな感じでインスタンス変数とインスタンスメソッドを定義することができます。インスタンス変数は通常、インスタンス毎に異なる値を持たせたい場合に利用します。例えば、データベースの検索結果を 1 行ずつインスタンスとして保持したい場合などです。

クラス変数とクラスメソッドとスタティックメソッド

Python ではクラス変数とクラスメソッドも利用でき、以下のように定義を行います。

```
class MyClass(object):
    # クラス直下に定義すると、クラス変数になる
    primary_key = "id"

    # クラスメソッドは「@classmethod」を付与して定義する
    @classmethod
    def show_primary_key(cls):
        print("PrimayKey is %s" % cls.primary_key)

# クラス変数やクラスメソッドへのアクセスは、インスタンス化する必要ない
print(MyClass.primary_key) # => id
MyClass.show_primary_key() # => PrimayKey is id
```

また、クラスメソッドに似たものとしてスタティックメソッドがあります。これは引数に `cls（自分のクラスを指す変数）` を受け取りません。

```
class MyClass(object):
    primary_key = "id"
```

```
    @classmethod
    def show1(cls):
        print(cls.primary_key)

    # スタティックメソッドは「@staticmethod」を付与して関数を定義します
    @staticmethod
    def show2():
        print(MyClass.primary_key)

MyClass.show1() # => id
MyClass.show2() # => id
```

クラスメソッドとスタティックメソッドでは、継承した場合に挙動が変わります。

```
class MySubClass(MyClass):
    primary_key = "subid"

MySubClass.show1() # => subid
MySubClass.show2() # => id
```

`show1` メソッドは引数に `cls` を受け取ることで `MySubClass.primary_key` を取得しています。一方、`show2` メソッドは関数定義内部で明示的に `MyClass.primary_key` を指定しているので、継承先の `MySubClass` から呼び出しても `MyClass.primary_key` の値を参照します。継承を行う場合に振る舞いをどうしたいのかによって、クラスメソッドとスタティックメソッドは使い分ける必要がありそうです（といっても、どっちを使ってもそんなに困ることはあまりないですが）。

クラスの継承

上で少しだけ出てきましたが、クラスを継承して新しいクラスを作成することができます。クラスを継承した先は、親クラスの内容を受け継いでそれらを利用することができます。

```
# 人を表すクラス
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def sayHi(self):
        print("Hi, my name is %s, %d years old." % (self.name, self.age))

# 仕事人を表すクラス
class Worker(Person):
    def __init__(self, name, age, skills):
        # 親クラスの関数を使う場合には、super()を使う
        super().__init__(name, age)
        self.skills = skills
    def show_skills(self):
        # 親クラスの変数も参照できる
        print("%s's skills are %s" % (self.name, self.skills))
```

```
# インスタンス化
w = Worker("Yohei", 30, ["html","js","css"])
# 親クラスのメソッドも呼び出せる
w.sayHi()
w.show_skills()
```

親クラスの変数や関数を継承することができ、また`super()`（Python2 系の場合は`super(self.__class__, self)`）で親クラスにもアクセスすることができます。

最後に

今日は Python のクラスを利用する方法をブログに書きました。インスタンス変数、インスタンスメソッド、クラス変数、クラスメソッド、スタティックメソッド、継承と、色々と使い分けられると便利ですね！

最後になりますが本ブログでは、フロントエンド・Python・機械学習など雑多に情報発信をしていきます。自分の第２の脳にすべく、情報をブログに貯めています。気になった方は、本ブログの RSS や Twitter をフォローして頂けると幸いです ＾＾。

最後までご覧頂きましてありがとうございました！