**○pythonの基本**

・print関数で表示（print()は省略可能）

print(“hello” + ” ” + “world” )

→hello world

・#を前に書くことでコメント化

・変数の定義　「変数名 = 値」　（変数名は数字が最初はだめ。2語以上の時は\_でつなぐ）

例）name = “yuton”

number = 8

・”文字列”+”文字列”が可能

・型変換　数値型(int)を文字列型(str)に変換できてしまう(逆も)　str(),int()

price = 100

print(“リンゴ”+str(price))

・「input」でコンソールに文字を入力できるようになり、その入力された値を受け取れる

「変数 = input('コンソールに表示したい文字列')」でコンソールに入力された値が変数に代入される。（受け取った値は文字列(str)になるから数字として使うならint型変換する）

apple = input(“リンゴの数を入力しなさい→”)

・print()で改行

・f文字列「print(f” ”)」で文字と変数を同時に使える。変数は{ }で囲む

alist = [10,20,30,40,50]

for i in range(len(alist)):

print(f"{i}番目の要素は{alist[i]}です") # 0番目の要素は10です

・type関数に入れると型を教えてくれる type(123)→int

・「＼n」「\n」で改行

・「＼b」「\b」で一文字消しながら戻る（バックスペース）

・「名前.shape」で配列の形状（各個数）がわかる

・assert文→条件が間違ってるときにエラーを出す。正しいなら何もしない

assert 1 = 0

→エラー

・reshapeメソッドを使えばリストを思うような形にできる

例）64個のリストX[1]を8＊8の表示に

b = X[1]

b2 = b.reshape(8,8)

print(b2)

・比較演算子inを使うとリストの中に指定した要素が含まれているか調べられる

リストだけでなく文字列の中にも適応できる

3 in [1,3,5,7] #true

・べき乗は「\*\*」例10\*\*5 →100000

・pythonの割り算は「//」で整数の割り算（余りは切り捨て）、「/」で小数の割り算

・絶対値は「abs(値)」

**○＋α**

・「divmod(a, b)」を使うとaをbで割った商（a//b）と余り（a%b）を同時に返してくれる。返り値はタプルとなっている。

**○桁数を合わせる方法**

・整数の桁数の合わせるにはprintf関数の中で「変数:桁数d」とする。こうすると左に空欄ができる。「変数:0桁数d」とすると空欄が0で埋められる

例）1を001と表示する

n = 1

print(f”python{n:03d}シリーズ”) # python001シリーズ

・小数の桁数の指定はprintf関数の中で「変数:.桁数f」とする。「.」をつけるのを忘れないで

例）3.141592653589793を3.14と表記する

print(f"{pi:.2f}") # 3.14

**○pythonの豆知識**

・リスト、辞書、タプル、集合などをつくるときには最後の要素の後にコンマをつけてもよい。コンマをつけておくとコピーペーストがしやすくなるからベターかも

alist = [0, 1, 2, 3,]

・変数.upperで変数をすべて大文字にできる

・変数.lowerで変数をすべて小文字にできる

・print関数の中に「sep=””」と書くと自動的に挿入される空白を抑制できる

・print関数の中に「end=””」と書くと文末の改行を抑制できる

・3つ以上の何かを扱うときにはリストや辞書を使うと良いかも

・datetime モジュールを使うと，日付や時刻に関する計算を楽にできる

「import datetime」でモジュールを呼び出し、datetime.date(2020, 12, 5)やdatetime.timedelta(days = 数字)を使い日付に変換する

例）今日から100日後の日付

import datetime

datetime.date.today() + datatime.timedelta(days = 100)

・変数の交換（スワップ）をしたいときは「u,v = v,u」だけでOK

**○pythonの慣例**

・pythonのプログラムの中に数字を置くのは良くない（ハードコーディング）

→len(alist)のように何の数字かを書いた方が良い

**○効率重視の考え方**

・リストの中の要素数を集計したいときにそのまま調べると何回もリストを調べることになるから専用のメソッドcollections.Counter()を使うとよい。これは初見の要素については新しいカウンタを作り、既知の要素についてはカウンタに1プラスしていくというプロセスである。

from collections import Counter

counter = Counter(リスト名)

for v in counter.keys():

print(f”{v} の出現数は {counter[v]}”)

**○ミスしやすいところ**

・リストのコピーを作るつもりで「新リスト＝旧リスト」として新リストのみ書き換えるつもりで「新リスト[0] = 100」とすると、旧リストも変わってしまう。これはコピーではなく参照

リストのコピーは「new\_list = alist.copy()」とする。

**○予備知識**

・円周率πを使うためにはmathライブラリからpiをつかう

import math

print(math.pi)

・もっとまで円周率πを使いたいときにはmpmathライブラリからpiを使う。使う前に「mpmath.mp.dps = 桁数」として桁数を決める

import mpmath

mpmath.mp.dps = 100

print(mpmath.pi)

・平方根をとりたいときはmathライブラリを importし、math.sqrt()とする。

例）平面座標上の距離(dxはｘ方向の長さでdyはｙ方向の長さ)

import math

math.sqrt(dx\*dx + dy\*dy)

**〇入力**

・入力はinputをつかう。Pythonの場合はC++と異なり、入力画面にも文字を表示できる

変数 = 使う型(input(“入力時に表示する文字列”))

例）足し算

A = int(input(“A=”)) # 入力画面に「A=」と表示

B = int(input(“B=”))

add = A+B

print(add)

**○ドキュメント文字列**

・ドキュメント文字列とはプログラムに何の影響もしないけど、他の人が見たときにどんな関数かを教えるもの

・関数の定義をした後のみ使える（defの下の行に””で囲んで書く）

ドキュメント文字列を複数行にわたって書きたいときは”””(トリプルクォート)で囲む

def hello7(name):

"入力されたnameの中に「伊達」という文字列が含まれていれば特別な挨拶をする"

if "伊達" in name:

print(f"お待ち申し上げておりました，{name} 殿！")

else:

print(f"こんにちは, {name}さん")

**○オプション引数**

・オプション引数とは、関数の引数に何も書かない場合に勝手に引数とする初期値である。

・関数の引数を書くところに「変数＝値」とかく

例）1を足す関数。引数を書かなかったら1を返す。

def add1(x = 0):

x += 1

return x

**〇リスト[ ]**

・大きなくくりの中にいろいろな要素を入れられる

・リストも1つの値である

・リスト名は慣用的に複数形にする

・番号は0から始まる

・空のリストを作るときは「aist = list()」でもよいし、「alist = []」でもよい

・[ ] の間に , で区切った要素を並べればリストが作られる

numbers = ['zero', 'one', 'two', 'three']

・要素へのアクセスは「名前[i番目]」

numbers[0]

→zero

・要素へのアクセスをするときに負の値を指定すると末尾から数えて〇番目になる

numbers[-1]

→two

・要素の数（要素のサイズ）はlen関数（lengthのこと）

len(numbers)

→４

・要素の追加　appendメソッドで末尾に新しい要素を追加　名前.append(“新要素”)

numbers.append(“four”)

・a[i:j:r]とかくとリストaのi番目からj-1番目まで公差rで取り出せる（スライス）

r=1,i=0,j=末項のときは省略できる

・「リスト名.count(値)」でリストの中のある要素の個数がわかる

・リストの中の最小値と最大値にアクセス→「min(alist)」「max(alist)」

**○ソート**

・リストのソートは2種類ある。「リスト名.sort()」でもとのリストを小さい順に並べる。

「新しいリスト名 = sorted(リスト名)」でソートした新しいリストを作成。元リストはそのまま

・「新リスト名 = sorted(リスト名, reverse = True)」で大きい順に並べられる

・sorted()のほうはオプション引数「key = 関数」を指定することでソートしたいリストを関数に通してその規則にのっとってソートすることができる。関数は自作でもよい。このときkey=関数の後ろには（）を書かない

例）大文字、小文字を気にしないでソート（そのままやると大文字が前に来ちゃう）

word\_list = [milk, Miyagi, Sendai, yuton, Nakano]

word\_list2 = sorted(word\_list, key = str.lower)

print(word\_list2) // [milk, Miyagi, Nakano, Sendai, yuton]

**○集合set { }**

・「集合名 = set(リスト名)」と書くとリストの中で重複、順番を区別しない集合を作れる

集合名 = {1,2,3,1,2,3,0,0}

例）出力のされ方

alist = [1,2,3,1,2,3,0,0]

aset = set(alist)

print(aset)

→｛1, 2, 3, 0｝と出力される

・空集合を作るときに注意が必要

「a = {}」と書くと辞書になってしまうから「a = set()」とかく

**○リストの内包表記**

・リストにどんどん要素を追加したいときには内包表記をするといいかも

・リスト内包表記→リストの中にリストに入れたい条件をそのまま書く

内包表記を使わないとからのリストを用意してから.appendしなきゃいけない

数学の集合の表記と似てる　S={i＾2　∣i∈{0,1,…9}}

例）0～100までの平方根のリスト

heihou[]

for i in range(100):

if i\*i <100

heihou.append(i\*i)

例）0～100までの平方数のリスト内包表記（なんと1行で書ける！！）

heihou = [i\*\*2 for i in range(100) if i\*\*2 <100]

**○タプル ( )**

・リストに似たやつ。()で囲む

・リストは要素を変更できるが、タプルは変更できない。

a = (0, 1, 2, 3, 4, 0, 6, 7, 8, 9)

a[5] = 5 　#変更できないのでエラーになる

・リストは辞書に登録できないが、タプルは登録できる

・要素数1のタプルを作るときに注意

「t = (3)」と書くとint型になっちゃう。「t = (3,)」と書けば良い

**〇辞書 { }**

・キー (key)と値 (value) のペアを覚え，キーを指定すれば対応する値を取り出すデータ型

・「名前={ }」 の間に，key: value のペアを ,で区切りながら並べる。または「名前＝dict()」

・空の辞書を作るときは「adict = dict()」または「adict = {}」とかく

【注意】

リストは[ ]だけど、辞書は{ }

b = { "one": 1, "two": 2, "zero": 0, "three": 3}

・要素へのアクセス

b[“two”]

→2

「b.get(キー、オプション引数)」と書くとキーにないものを書いてもNoneが返り、エラーにならないし、ない場合にはオプション引数を返すこともできるから優秀

・要素の追加（末尾に）

b[“four”] = 4

・辞書の keyの一覧の確認　名前.keys()

・辞書のvalueの一覧の確認　名前.values()

・辞書のkeyとvalueのペアの確認　名前.items()

・リストのように辞書も内包表記が可能

・逆引き辞書の作り方

e2j = { "one": "壱", "two": "弐", "zero": }

j2e = {v:k for k,v in e2j.items() }

・キーから値にアクセスするには　「名前[キー]」または「名前.get(キー,オプション引数)」

get()のほうがエラーを防げるし、オプション引数を指定もできる。キーだけでもよい。

・辞書の値が最大、最小になるキーを取得するには下記のようにする。値で大小比較をして、出力するのはキーにする

dict1 = {"apple": 3, "banana": 5, "orange":1, "strawberry": 2}

print("max = ", max(dict1, key=dict1.get)) # banana

keyを指定しない場合は辞書のキーで最大（辞書式順列）で取り出される。値は無視

print("max = ", max(dict1)) # strawberry

print("min = ", min(dict1)) # apple

・counter = defaultdict(int) と初期化しておくと，この辞書は，あたかも，すべてのキーkに対して，事前に { k : 0 } が登録されているかのように振る舞う．つまり，任意のキーに対するカウンタが準備されているとみなせるのです．新しいキーが出てきたらその都度勝手に新しく創ってくれるということ

**○if文**

・条件が正しいときのみ作用する

・インデントが重要。しないとおかしくなる。

if 条件:

処理

・elifは上の条件が偽で指定する条件が真の時のみ作用する

elif 条件:

処理

・上のすべての条件が正しくないときの処理はelse文を使える

else:

　　処理

例）7の倍数の時だけ改行して、違うときはその数字を出力

if i%7 == 0:

print()

else:

print(i)

【注意】

if文、else文、elif文の後のコロン「:」を忘れやすい

・条件式の連立

and→かつ、or→または、not→条件式の否定（if not 条件）

if a = 10 and b =0:

・if文の条件の中では論理演算子を使える。and→かつ、or→または

if 10 <= x and x < 100:

if x < 0 or 100 < x:

・論理演算子　&→かつ、|→または

**○for文**

・pythonでは、インデントで範囲が決まるので見た目が重要

for 要素の名前 in リストや辞書の名前:

alist = [10, 20, 30, 40, 50]

for v in alist:

print(v)

こうも書ける

for i in alist:

print(alist[i])

・while文もある

i=0

while i <= 100:

処理

i += 1

・pythonではforに対してelseが使える。ループ内の処理がすべて終わったときのみelseの中身を実行する。逆に言えば途中でbreakしたら実行しない

・enumerate()関数を使えば、リストなどの要素を前からiで順番をつけながら回せる。()の中にはリストや辞書などの名前を書く

alist = [5,4,3,2,1,0]

for i,v in enumerate(alist):

print(f"{i}番目の要素は{v}です") #　0番目の要素は5です

・zip()関数は2つ以上のリストに対して前から取り出し、回す関数。()の中にはリストをカンマで区切りながら、書く。inの前にはfor分の中で使う変数の名前を呼び出すリストの数分書く。リストの長さが異なるときは短い方に合わせられる。

for x1,x2,x3 in zp(alist1, alist2, alist3):

names = [yuton, nakano, miruku]

ages = [18, 49, 15]

for name, age in zip(names, ages):

print(f”名前：{name}、年齢：{age}”) #　名前：yuton、年齢：18

**○range()関数**

・範囲を指定できる(iからj-1まで,公差ｒの増分)

range(i, j, r)

・range(0,5,1)はrange(5)と省略できる(増分1なら省略可能)

・整数しか対応してない（小数はだめ）

**○モジュール**

・モジュールとはimportで読み込んでから使う関数のこと

・time.sleep()関数（timeモジュールの中の関数）

import time

time.sleep(秒数)

と書くと処理が停止する（fot文と組み合わせるとホラー演出ができる）

from time import sleep　という書き方もある

・モジュールから関数をimportするときにasを使うと関数に自分で名前をつけれる

from time import sleep as stop #sleep関数にstopという名前をつけた

import numpy as np #numpyモジュールをnpと省略した

・matplotlibライブラリに含まれるpyplotモジュールもよく使われるので，pltという名前で import することが多い（グラフを出力できる）

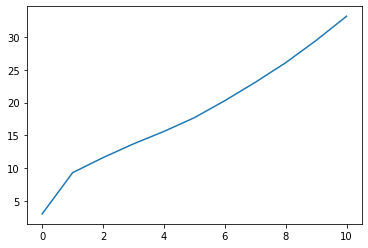
age = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

weight = [3.0, 9.3, 11.6, 13.7, 15.6, 17.7, 20.3, 23.1, 26.1, 29.5, 33.2]

import matplotlib.pyplot as plt

plt.plot(age, weight)

plt.show()



・関数の前に「%%writefile 名前.py」と書くことで、モジュールを自作できる

呼び出すときは「import 名前」

**○関数の自作**

・関数を自分で定義するときは「def 名前(引数名):」とし、処理はインデントする

・関数のすぐ下にドキュメント文字列をつけるようにする。単なるコメントではない…

def predict(x) :

“この関数の説明を簡単にここに書く”

例）"Hello World!"と出力する関数

def hello\_world ():　#関数の定義（引数なし）

print ("Hello World!")　#関数の処理

hello\_world ()　#関数の実行

例）挨拶をする関数

def hello(name):

print(f"こんにちは, {name}さん")

hello(“ゆうとん”)

→こんにちは、ゆうとんさん

・戻り値を持つ関数には「return 戻り値」とする

def add(x, y):

return x + y

add(1,2)

→3

・引数が複数あり、入力されないときに出力するデフォルト値を設定できる

「引数＝デフォルト値」

引数は最初の方から反映されるので1個飛ばして引数を設定したいときは引数＝””

def hello2(name, greeting="こんにちは"title="さん"):

print(f"{greeting}, {name}さん")

hello2(ゆうとん)#こんにちは、ゆうとんさん

hello2(“ゆうとん”, tytle = “様” ) #こんにちは、ゆうとん様

**○オプション引数**

・オプション引数とは関数の引数を指定しない場合に勝手に代入してくれる初期値のこと

・def 関数名 (オプション引数名 ＝ 値):

例）乱数の種

def dice(random\_state = 88):

random.seed(random\_state)

return random.randint

・普段は乱数で、引数を入力時のみ乱数の種を使いたいときはrandom\_state = Noneとする

**○random関数**

・「import random」でrandomモジュールを呼ぶ

【random.random()】

0以上1未満の小数をランダムに生成する

・x = randam.random()をfor文で回してｘの値によって分ければ、ランダムに自分の思うような確率に近く分けることができる。

例)コイントスを作る(cointoss()関数)

import random

def cointoss():

x = random.random()

if x < 1/2:

rerurn 1

else:

return 0

【random.randint(a,b)】

a以上b以下の整数をランダムに返す。ｂも含む

例）サイコロを作る（dice()関数）

import random

def dice():

x = random.randint(1,6)

return x

【random.randrange(start,stop,step)】

startからstopまでの整数をstepごとにランダムに生成する。stopは含まない

例）サイコロ

def dice2():

　　return random.randrange(1,7,1)

【random.choice(リストの名前)】

・リストの中から等確率で返す

例）サイコロ関数

def dice():

me = [1,2,3,4,5,6]

return random.choice(me)

【random.choices(リストの名前, weights = [リスト], ｋ＝返したいリストの数)】

・重複を許してランダムに選んだk個からなるリストを返す

・weights = []の中のリストで出現確率を決める

例）トランプのマークを10個ランダムに出力

marks = ["♡", "♣", "♢", "♠"]

a = random.choices(marks, weights=[10, 1, 5, 4], k=10)

print(a)

→['♡', '♠', '♠', '♠', '♡', '♢', '♢', '♡', '♡', '♣']

【random.sample(リスト, k = 数字)】

choices()とは異なり重複を許さずにランダムにｋ個返す。

・リストの個数分回せば、リストをランダムに並び替えることができる

例）並び替え

a = random.sample(range(10),10)

【乱数の種】

・「random.seed(数字)」で乱数の種を固定できる。（デバッグに利用）

・関数の中で乱数の種を使うときはclassを使って\_\_init\_\_関数の中で埋め込むとよい

例）サイコロ関数

class dice :

def \_\_init\_\_ (self):

random.seed(88)

def throw(self):

return random.randint(1,6)

**○numpyの機能**

・import numpy as npでnumpyをnpと使うことができる

・np.arange(初項, 末項, 公比)

→range関数と同様に等比数列を作れるが小数にも対応している優れもの（末項は含まれない）

・x = linspace(start, stop, 何個か) でstartからstopまでの等比数列をｎ個作成(末項含む)

例）x = linspace(1, 3, 5) → 1, 1.5, 2, 2.5, 3

・y = np.sin(x) でsinを呼べる

・np.pi で円周率πを定数のように使える

・配列名 = np.column\_stack([リスト1、リスト2、リスト3…])でリストをくっつけた2次元配列を作れる

【グラフ】

・import matplotlib.pyplot as pltでmatplotlib.pyplotをpltと使うことができる

・plt.○○でグラフを使うことができる。このとき下の段にplt.show()とかく

・plt.plot([ｘ座標],[y座標],c=”red”)→折れ線グラフ（直線）

[ｘ座標]のところにリストの変数を入れても良い

・plt.plotの中では c = ‘red’ で色を赤にできる。他にもいろいろある

・plt.plotの中では marker = ‘o’でグラフ上の点の形を決められる。他には’+’がある

・plt.plotの中では label = ‘軸ラベル’ で関数の名前を表示できる。

その際には plt.legend(loc = ‘best’) と書き、良い感じの位置にしてもらうと良い

例）y=sinxのグラフを赤色、点をまる、関数の名前を「y=sinx」とする

plt.plot(x, np.sin(x), c = ‘red’, marker = ‘o’, label = ‘y=sinx’)

・plt.scatter()→ドット（点）のグラフ

・plt.hist()→ヒストグラム

・plt.bar()→棒グラフ

・plt.grid() でグラフにマス目を出せる

・plt.xticks([リスト]) または plt.yticks([リスト]) でメモリを描く値のリストを渡せる

・plt.xlim([リスト]) または plt.ylim([リスト]) でx、y座標の表示する範囲を決められる

・plt.xlabel(‘軸ラベル’) または plt.ylabel(‘軸ラベル’) でx、y座標の軸ラベルを表示できる

・plt.title(‘タイトル’) でグラフのタイトルを表示できる

・fig = plt.figure(dpi=100) で解像度100の写真を用意する

・fig.savefig(‘ファイル名.png’) で写真を保存できる。ファイル名.pdfでpdfも保存できる

【ファンシーインデックス】

・data = np.array([0,1,2,3,4,5]) とすることでリストの中から好きなものを抽出できる

・使うときには2重かっこにする[[ ]]

data[[0,4]]

→array([0,4])　と取り出せる（0、4番目を抽出できる）

・二次元のリストからファンシーインデックスするときは注意が必要

data[[0,3,5]]　　　　　 　→0,3,5行目を抽出

data[ : ,[2,4,6]]　　　　 　→2,4,6列目を抽出

data[[0,1,3]][ : , [0,2,4]]　 →0,1,3行目の0,2,4列を抽出

**○scikit-learn分類器**

・基本的な使い方は下記の通り。ただし、3，4行目は人間の確認用

model = ○○\_Classifier( )

model.fit(X, y)

print("予測:", model.predict(X))

print("正解:", y)

print("正解率は", model.score(X, y))

・クラスを継承することによって前のクラスの上書きができる。

class DIYClassifier02( DIYClassifier01 ):

　　def kannsuu():

→01を継承して02には01＋02に書いた関数が乗る

・関数の中で「self.関数名」とすることで関数の外でも「 class名.関数名」として使える。このように紐付けることでclassの中の異なる関数で同じ変数を使うことができる

・予測器のclassの中の関数は下記の3つからなる

1. \_\_init\_\_関数

値の紐付け。self.を使って行う

1. fit関数

学習させる

1. predict関数

予想結果をリストにして返す

【k近傍法】

それぞれのデータに対して、その点に近いｋ個のデータを参照して、その点を分類する。わかりやすいけど、データが増えると計算量が莫大に増えてしまうのが欠点。

【決定木（decision tree）】

一番効率の良い特徴量を1つ取り出し、2つに分ける。完全に分類できたなら葉となりその枝の分類は完了。完全に分類できていないのなら、再び分類しやすい特徴量を1つ取り出し、2つに分ける。これを再帰的に繰り返す。

特徴量の選び方は、それぞれから不純度を割り出し、そのときの分けた後の複数の不純度の加重平均をとり、元の不純度との差をとる。これを情報利得(information gain)という。情報利得が最も大きい特徴量を選ぶという手順からなる。

【線形サポートベクトルマシン（線形SVC）】

1本の直線（3次元なら平面）を良い感じの位置に引いて、分類する。2次元の場合、良い感じの分離直線の位置は直線に一番近い点との距離が最大になるようにする。この方法で完全に分離できる状態を「線形分離ができる」という。最も近い訓練例のことを「サポートベクトル」といい、サポートベクトルと直線の距離を「マージン」という。

Cというペナルティ項を導入して、「マージンが広がると報酬を得る」「マージンの内側に侵入したり誤分類するとペナルティを課す」Cは後者に付く係数でありペナルティの割合を決める。Cが大きいほどペナルティが重くなり、ペナルティを避けようとするので過分類しやすくなる。Cが小さいとペナルティが軽んじられ、ただマージンを広げようとするあほな分類器になる。

【非線形SVN：多項式カーネル】

直線での分類には限界がある。そこで特徴量を2乗したり、かけてみたりすることによって新しいと特徴量を作る。この過程を「特徴量エンジニアリング」という。「カーネルトリック」によって、この中からいい特徴量を選んでいる。

【非線形SVM：ガウスRBFカーネル】

特徴量エンジニアリングをする際にある場所に原点を作ってしまい、その点への近さを新たな特徴量として線形分離できるようにする方法。それぞれのデータで新たな特徴量を作っている。これがSVMの中では最も精度が良い。

「SVC(kernel=”rbf”, gamma=値)」とする。gammaはそれぞれのデータが作る新たな特徴量の主張の強さを示す。gammaが大きすぎると主張が強すぎてデータにないものの分類ができなくなってしまう（過学習）

【アンサンブル法】

いくつかの予測器を組み合わせて多数決をして1つの予測器よりも精度を高くする方法。複数の異なる予測器でも使えるし、同じ予測器で何回も行うこともできる。自信がある予測器の発言権を大きくするソフト投票のほうが精度がより高くなる。