実践統計解析【第10回】

深層学習，すぐできます.

松田　史生[1]*・川瀬　雅也[2]

Aさん：深層学習っていうのが，最近めっちゃ盛り上がってますよね．AI囲碁ソフトが人間に勝ったとか，医療画像の診断ができるとか，そのうちAIが人間を追い抜くシンギュラリティーがおきるとか…

B君：確かに，ちょっと流行りすぎかなとは思うけど，すごい技術だよね．この連載でも第4回（本誌第97巻第10号）でちょっとやってみたけど，深層学習ってPythonだとすぐできるんだよね.

チャレンジ深層学習！

B君：こないだ，この連載を執筆しているJ研究科のM先生に，深層学習を勉強するのに便利なデータセットを質問しに行ったら，生物工学会誌上で紹介できるような，シンプルなデータセットがあんまりないんだわ，ってぶつぶつ言ってた.

Aさん：じゃあどうするんですか？

B君：訓練用のデータもその場で作ってしまえってアドバイスをいただいた．たとえば，サイン関数と同じことができる（入力x: 0～4の実数，出力y: sin(xπ)），ニューラルネットワークを機械学習で作ってみよう.

＜リスト1＞

```
import numpy,math,random
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
import matplotlib.pyplot as plt
data = []
target =[]
size = 1000
#データ準備
for i in range(size):
    x = random.random() *4# 入力値をランダム生成
    y = math.sin(x* math.pi) #出力値を作成
    data.append([x]) #dataに追加
    target.append(y) #targetに追加
#テストデータと訓練データを7:3に分割
X_train, X_test, y_train, y_test = train_test_split(data,
target, test_size=0.3)
#ニューラルネット作成
reg =
MLPRegressor(hidden_layer_sizes=(10,10,10,10,),
max_iter=10000) #ニューロンが10個の隠れ層が4層のニューラルネットワークを生成
reg.fit(X_train, y_train) #訓練データでトレーニング
#テストデータでテスト(相関係数で評価)
print(reg.score(X_test,y_test))
#グラフに表示
plt.scatter([x[0] for x in X_test], reg.predict(X_test))
plt.show()
```

Aさん:まず，必要なモジュールをインポートした後，データを生成していますね．0～4の範囲でランダムにxを生成してサイン関数でyに変換した1000個（size = 1000）のデータセットを作成しています.

B君：次にsklearnモジュールのmodel_selection.train_test_splitという機能でデータを訓練用とテスト用に7：3に分割してるんだな．それから，sklearn.neural_network.MLPRegressorという機能を使って，ニューラルネットワークを作成している．hidden_layer_sizes = (10,10,10,10,)というのは，10個のニューロンからなる隠れ層が4つあるという意味だ．このニューロンと隠れ層が多いほど複雑なことができる．詳しい仕組みは自分で調べてね.

Aさん：多階層のニューラルネットワークを効率良く学習させるいい方法が見つかって，深層学習ブームが始まったんですよね.

B君：という理由で，今回は隠れ層を4層にしてみた．次の，reg.fit(X_train，y_train)で訓練データで機械学習を行っている．その次のprint(reg.score(X_test,y_test))では，テストデータで性能を評価している．予測値と実際の値の相関係数が出力される.

Aさん：データ作成以降はたった3行しかないですね．とりあえず，実行してみた結果がこれです（図1）．結果は実行するたびに微妙に異なります.

著者紹介　[1]大阪大学大学院情報科学研究科（教授）　E-mail: fmatsuda@ist.osaka-u.ac.jp
[2]長浜バイオ大学（教授）　E-mail: m_kawase@nagahama-i-bio.ac.jp

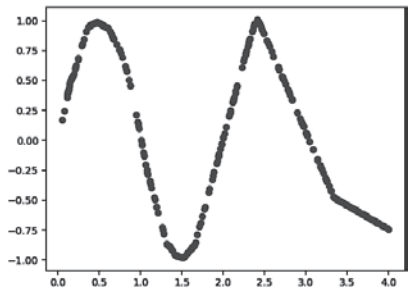

図1．テストデータの予測結果
size = 1000, hidden_layer_sizes=(10,10,10,10,)

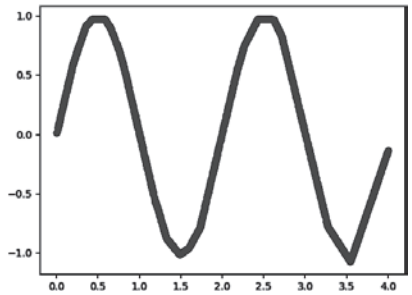

図2．テストデータの予測結果
size = 100000, hidden_layer_sizes=(10,10,10,10,)

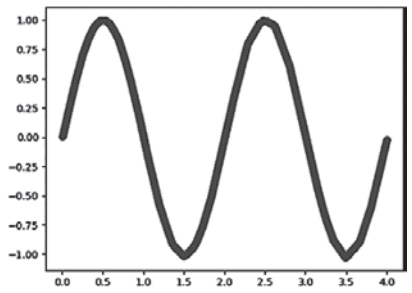

図3．テストデータの予測結果
size = 100000, hidden_layer_sizes=(20,20,20,20,)

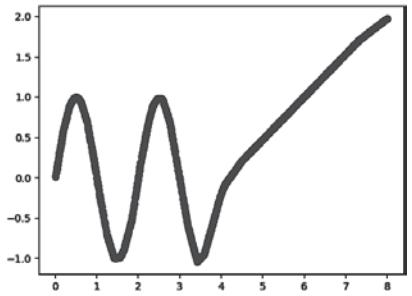

図4．学習していない範囲（4–8）を含む予測結果

B君：なんか，がたがただなぁ．いまいちなのでデータサイズをsize = 100000にしてみよう（図2）．

Aさん:だいぶましになりましたね. 次に隠れ層のニューロンの数を20にしてみました（図3）.

B君：まだちょっとかくかくしているけど，かなりサイン関数っぽくなってきたよね. やらせっぽいのは, 例ってことでお許しください.

Aさん：たしかにデータを大量につかって学習を行うだけで，サイン関数が作れちゃいましたね.

B君：この例から言えることは，訓練データが大量にあれば，入力から出力を生成する関数を自動で作れちゃうのが，すごいってことだね.

Aさん：遺伝子の発現プロファイルデータから有用物質生産量を予測するとか，化合物の構造から液体クロマトグラムの保持時間を予測するとか，盤面から最善の一手を予測するとか, いろんな応用がありえますよね.

B君：でも，学習した範囲でしか正確に予測できない. 入力xの範囲を0～4で訓練を行い，xの範囲を0～8に広げてテストすると, こういう残念な結果になる（図4）.

Aさん：学習した範囲以外はわかりません，っていうことですよね．サイン関数の本質をつかんだというよりは，教えた範囲でサイン関数をまねできるようになったということみたいですね.

B君：あと，どういう仕組みでサイン関数が再現できたのかはいまいちわかりにくい．重回帰分析なら各説明変数の係数から，重要な説明変数を同定したりできたけど，ニューラルネットワークでは，ちょっと工夫が必要になる.

判別機をつくってみる

B君：次に判別機を作ってみよう．0～9の数字からなる10桁の数列に1が含まれていたら1を，入っていなかったら0（ゼロ）を返す判別機をつくってみようか.

Aさん:（しばらくして）できました. 前例との違いはデータ生成部分と，sklearn.neural_network.MLPRegressorという回帰用の機能の代わりにsklearn.neural_network.MLPClassifierという判別機用の機能を利用している点です.

＜リスト2＞

```
import numpy,math,random
from sklearn.model_selection import train_test_split
```

```
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
data = []
target =[]
size = 10000
#データ準備
for i in range(size):
    z = 0
    temp = []
    for i in range(10):
        x = int(random.random() * 10)
        if x == 1:
            z = 1
        temp.append(x)
    data.append(temp)
    target.append(z)
#訓練データとテストデータに分割
X_train, X_test, y_train, y_test = train_test_split(data,
target, test_size=0.3)
clf = MLPClassifier(hidden_layer_sizes=(50,50,50,50,))
clf.fit(X_train, y_train) #訓練データでトレーニング
print(clf.score(X_test,y_test)) #テストデータでテスト
```

実行結果（毎回異なる）

➢ 0.6413333333333333

B君：データサイズが10000で正解率が約6割っていうのは悪くないな．つぎは100000個に増やしてやってみよう．さすがにこの学習には数分の時間がかかる．

実行結果（毎回異なる）

➢ 0.9975333333333334

Aさん：わ，スゴイ正解率ですね．

B君：数列の中にある特定のパターンを深層学習で検出できるようになったわけだよね．

Aさん：となると，画像データも数列なんだから，レントゲン画像データから病気があるもの，画像から出荷できない野菜を選別する判別機とか，いろんなものを作れそうですね．

画像を分類

B君：次は画像の分類をやってみようか．scikit-learnには手書き文字のデータセットdigitsがついてくる．これを呼び出してみよう．Spyderのコンソール上で入力してみて．

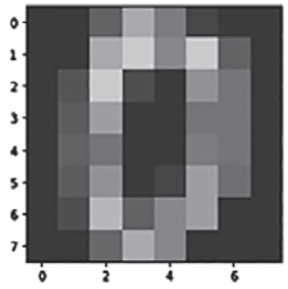

図5．手書きデータの例．学習にはこれを1次元に並べなおしたデータを用いる．

```
In [1]: from sklearn.datasets import load_digits #digitsデータセットの読み込みモジュール
In [2]: digits = load_digits()#digitsを読み込み
In [3]: import matplotlib.pyplot as plt
In [4]: plt.imshow(digits.images[0]) #pyplotで可視化
Out[4]:
```

Aさん：これは数字の0ですね（図5）．

B君：このデータセットには8*8ピクセルの手書きの数字画像とその正解情報が1797セットある．これを用いて，手書き画像を0–9に分類する判別機を作ってみよう．

Aさん：データを読み込んだあと，最後の3行はまったく同じになりますね．

＜リスト3＞

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
digits = load_digits()#digitsを読み込み
data = digits.data #画像データ
target = digits.target #答え
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size=0.3) #訓練データとテストデータに分割
clf = MLPClassifier(hidden_layer_sizes=(20,20,))
clf.fit(X_train, y_train) #訓練データでトレーニング
print(clf.score(X_test,y_test)) #テストデータでテスト
出力結果（正解率）
0.9666666666666667
```

B君：ざっと97 %の正解率の判別機ができたってこと

だね．超簡単だろ．

Aさん：なんかあっけないですね．ネットにたくさんある深層学習の記事は何故にあんなに盛り上がっているんですか？

B君：今回hidden_layer_sizes=(10,10,10,10,)と記述したけど，シンプルなニューラルネットワークでは性能に限界があるんだよ．そこで，隠れ層の数を増やしたり，ネットワークの構造を工夫したりすることで，用途ごとの性能を上げる研究が盛んに行われている．というか，今回の例と高度な手法には，ニューラルネットワークの構造くらいしか違いがない．

Aさん：訓練データで学習してるだけですもんね．

B君：この連載の読者の皆さんなら，たくさんある参考書やネットの記事を参考にちょっと勉強して，最新手法のスクリプトを改造すれば，自分のデータですぐ試せるよ．むしろ，生物工学分野の課題は十分な量のデータをどうやってそろえるかだと思うなぁ．

Aさん：ほかに注意点はないんですか？

B君：ある．Anacondaに入っているscikit-learnでは高度なニューラルネットワークを扱えない．でも，最新の深層学習モジュールもPython上で動くので，心配いらない．

衝撃の次元圧縮法

C君：先輩！聞いてください！

Aさん：C君しばらく見なかったけど何してたの？

C君：いろいろあって下宿に引きこもってたんですが，すごいの見つけちゃいました！この手書き文字のデータの主成分分析をするじゃないですか．

＜リスト4＞

```
from matplotlib import pyplot # pyplotをインポート
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
digits = load_digits()
pca = PCA(n_components=4)#主成分分析準備
score = pca.fit_transform(digits.data) #主成分スコアの計算とグラフの表示
pyplot.scatter(score[:, 0], score[:, 1], c=digits.target)
pyplot.colorbar()
pyplot.show()
```

```
#tSNE法を実施
from sklearn.manifold import TSNE#モジュールインポート
tSNE = TSNE(n_components=2)# tSNE準備
score = tSNE.fit_transform(digits.data) #スコアの計算とグラフの表示
pyplot.scatter(score[:, 0], score[:, 1], c=digits.target)
pyplot.colorbar()
pyplot.show()
```

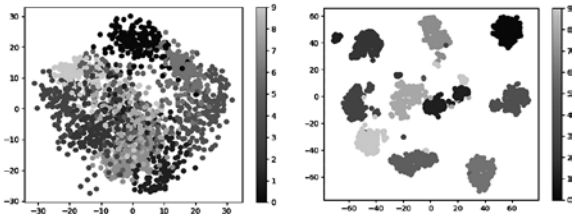

図6.（左）digitデータセットの主成分分析結果，（右）t-SNEでの次元圧縮結果．

C君：これを実行すると，まず主成分分析（左）とt-SNE（t-distributed Stochastic Neighbor Embedding）という最新の方法（右）で行った次元圧縮結果が出ます（図6）．

Aさん：うわ，主成分分析ではぼんやりした分類が，t-SNEではちゃんと0–9の10個のクラスターに，はっきり分かれてますね．

C君：最近はオミクスデータの可視化の論文で，よく使われているみたいですね．

B君：なんでこんなことができるの？

C君：いま，勉強中なんですけど，元の次元（この場合64次元）で距離が近い点が，2次元でも近くなる確率分布パラメータを探す，ということをしているみたいですね．

Aさん：深層学習もすごいけど，次元圧縮やいろんな技術がどんどん進歩しているのね．うまく使いこなして生物工学会で発表しなくっちゃ．

参考文献

1) van der Maaten, L. *et al.*: *J. Mac. Learn. Res.*, **9**, 2579 (2008).