データサイエンス　最終レポート

T319059 谷本樹希

はじめに

本レポートでは、クラス分類と次元削減、深層学習を行った結果とその考察について報告する。

クラス分類

概要

クラスタリング用の等方性ガウス分布の塊を生成して、クラス分類器によってそれを分類する。特徴量は２次元で、塊の数は４つに設定した。つまり、２次元のデータを４クラスに分類する分類器を実装する課題である。また分類器の分類境界線を図により示す。

プログラム

以下にプログラムの主要部分を示す。

```
	N = 4

	from sklearn.datasets import make_blobs
	X, y = make_blobs(random_state=122, n_samples=450, n_features=2,
cluster_std=1.8, centers=N)
	from sklearn.model_selection import train_test_split
	X_train, _, y_train, _ = train_test_split(X, y, random_state=1)

	colors = ["deeppink", "lawngreen", "blue", "salmon"]
	if N > 4:
		colors = gene_colors(N=N)

	plt.grid(True)
	plt.title("Class classification boundaries")
	plt.xlabel("feature 1")
	plt.ylabel("feature 2")
	for i in range(N):
		plt.scatter(X[y == i][:, 0],X[y == i][:, 1], c=colors[i],
alpha=0.8)


	if True:
		from sklearn.svm import SVC
		estimator = SVC(C=4, gamma="auto")
	else:
		from sklearn.neural_network import MLPClassifier
		estimator = MLPClassifier()
```

```
        estimator.fit(X_train, y_train)
        print("socre :",estimator.score(X_train,y_train))
        from sklearn.metrics import confusion_matrix
        cm = confusion_matrix(y_train, estimator.predict(X_train))
        print(cm)
        x_min, x_max = X[:, 0].min()-1, X[:, 0].max()+1
        y_min, y_max = X[:, 1].min()-1, X[:, 1].max()+1
        resolution = 0.5
        x_mesh, y_mesh = np.meshgrid(np.arange(x_min, x_max,
resolution),np.arange(y_min, y_max, resolution))
        z = estimator.predict(np.array([x_mesh.ravel(), y_mesh.ravel()]).T)
        z = z.reshape(x_mesh.shape)
        from matplotlib.colors import ListedColormap
        cmap = ListedColormap(tuple(colors))
        plt.contourf(x_mesh, y_mesh, z, alpha=0.4, cmap=cmap)
        plt.xlim(x_mesh.min(), x_mesh.max())
        plt.ylim(y_mesh.min(), y_mesh.max())
        plt.show()
```

プログラム１　クラス分類を行うプログラム

プログラム１は、クラス分類のプログラムである。X,yには生成されたデータセットが格納される。そのデータを使って散布図としてプロットする。SVMを使って分類器を生成、学習を行った。

実行結果

標準出力には、分類器の正答率と分類結果である混同行列が出力される。またmatplotlibによって、散布図とクラスの分類境界線を示した図が表示される。

```
socre : 0.8991097922848664
[[82  3  7  0]
 [ 4 68  9  0]
 [ 3  8 74  0]
 [ 0  0  0 79]]
```

出力１　分類器の正答率と混同行列

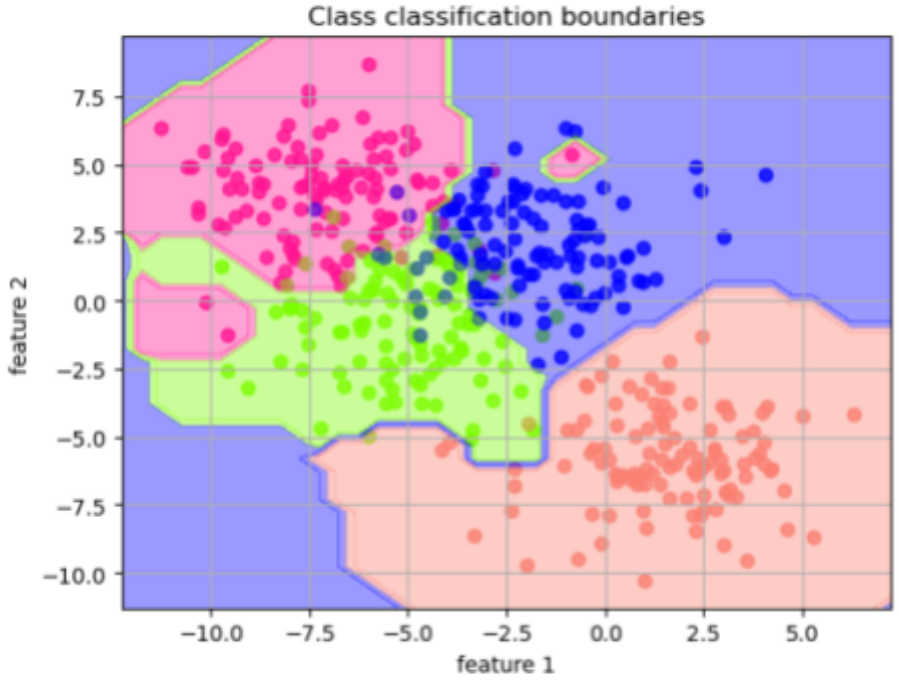

図1　散布図と分類器の分類境界線

上に示した出力１からもわかるように分類器の精度は89.9%だった。図１に散布図と分類境界線を示したグラフを示す。分類境界線は、全体的に青い部分が多くなるように引かれている。次にオレンジ色の面積が多い。

考察

ピンク色のクラスには飛び地があって、その飛び地もしっかりと同じクラスとして分類境界が引かれている。ピンク色と青色、緑色のクラスは境界で少し混じり合っている。しかしほぼ完全に一番多いクラスに含まれている。今回の分類はSVMを用いて、ガンマ値はsklearnのautoで行った。結果から

次元削減

概要

0 から９の手書き数字画像データセットである MNIST データから適当に 3000 個のデータを選択し、各数字の頻度を示す。選択したデータを T-SNE により２次元に次元削減を行う。その結果をプロットする。

プログラム

以下にプログラムの主要部分を示す。

```
def task2():
	from sklearn.datasets import fetch_openml
	X_org, y_org = fetch_openml('mnist_784', version=1, data_home=".",
return_X_y=True, as_frame=False)
	X = X_org[0:3000]
	y = y_org[0:3000]
	import collections
	c = collections.Counter(y)
	print(c)

	from sklearn.manifold import TSNE
	X = TSNE(n_components=2).fit_transform(X)

	colors = gene_colors(N=10)
	plt.clf()
	for i in range(10):
		plt.scatter(X[y == str(i)][:, 0],X[y == str(i)][:, 1], c=colors[i],
alpha=0.8, label=str(i))
	plt.legend(loc='upper left')
	plt.show()
```

プログラム２　次元削減を行うプログラム

プログラム２は手書き数字画像の次元削減を行うプログラムである。プログラム内では、はじめに手書き数字のデータセットであるMNISTをダウンロードし画像データと正解ラベルをそれぞれX_org, y_orgに格納した。格納したX_org, y_orgをそれぞれ初めから3,000個だけをスライスで切り出してX,yに格納して次元削減の用意が整った。scikit learn の用意TSNEにより次元削減を行った。結果はXに格納した。出力されたX,yを散布図としてプロットした。

実行結果

実行結果を以下に示す。標準出力には、各数字の画像枚数が表示される。また、matplotlibによって次元削減後の画像が散布図で表示される。各クラスには同じ色が付いている。

```
Counter({'1': 339, '7': 329, '4': 325, '6': 306, '2': 299, '3': 295, '9': 287,
'0': 285, '5': 274, '8': 261})
```

出力２　データセット内の各数字の枚数

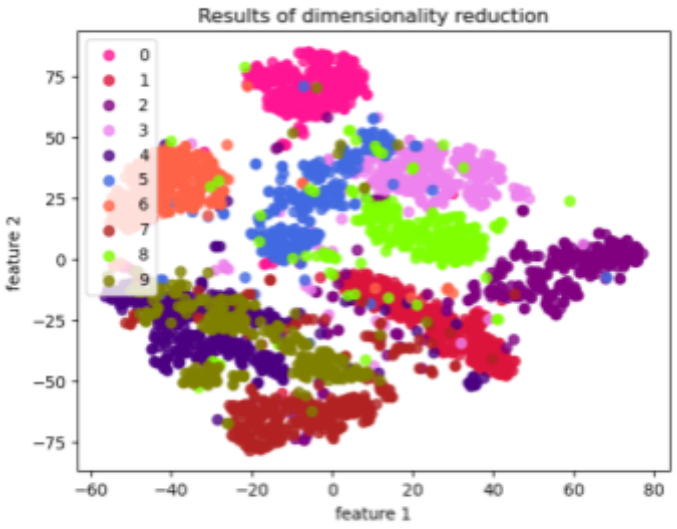

図２　次元削減の結果を示す散布図

図２には次元削減の結果を示す散布図を示す。０から９に違う色を割り振り、２次元まで削減したMNISTの各データを散布図でプロットした。概ねクラスごとに分布をかためることに成功した。しかし、まとまりすぎていて一部の数字ではまとまりがかぶってしまっている。

t-SNEのデフォルトパラメーターからperplexityを変更して実行した結果を図３に示す。

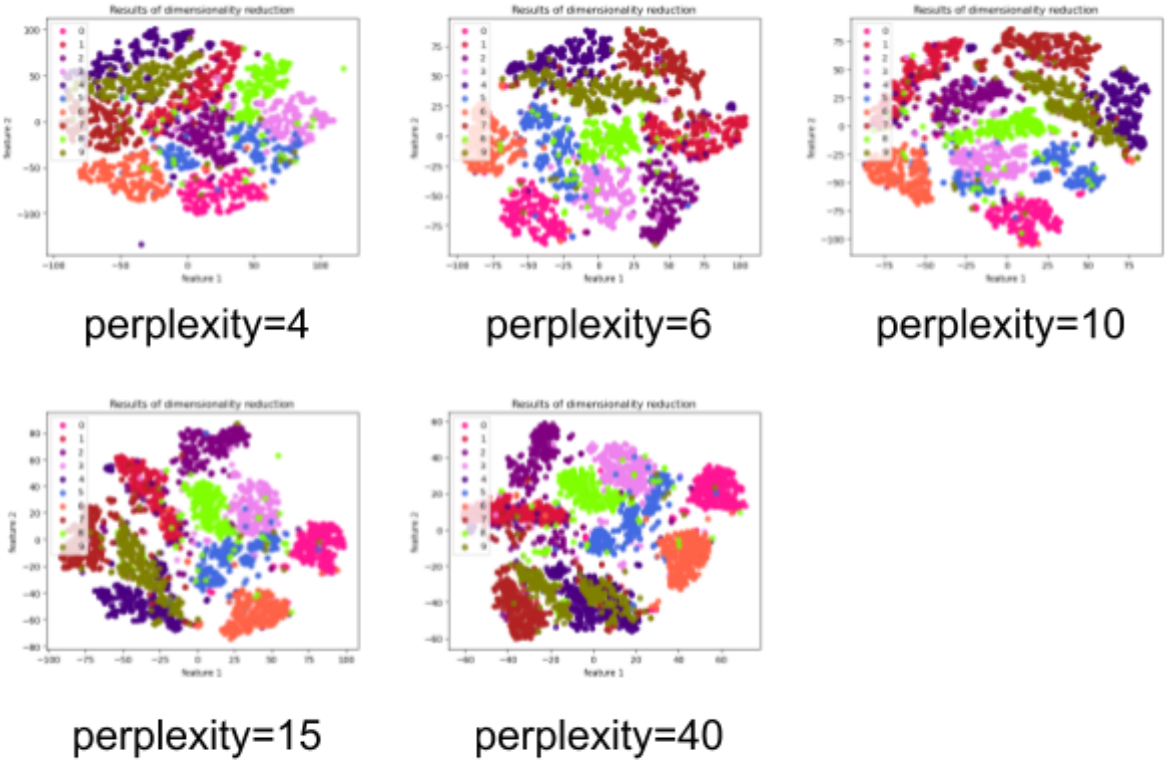

図3　perplexityを変更したときの散布図の変化

考察

700次元以上ある画像を2次元に次元削減しているので、混じっている部分もあった。特に似ている数字ではそれが顕著に現れた。例えば、９と４は輪から下方向に棒が飛び出ている形状で似ているので、範囲がそれぞれの塊が混じっている。５と３も塊の一部分が混じっていて、数字の下の部分が円のようになっていてそれが原因と考えられる。５と３の近くには、８が多く分布している塊がある。これは、３が８に内包されるような形状をしていて、画素の分布も同様に８の中に３が含まれるような状態になっているので、８と３の分布が近くなっていると考えられる。

パラメーターのperplexityを変更する実験では、perplexity=6のときが一番クラス間の混じりが少なく想定していた散布図が出力できた。また、どの場合でも３、５、８などの似ている数字は近傍に塊が配置されている。

t-SNEを使用するときには、パラメーターを変更しながら一番良いものを選ぶと良いことがわかった。

深層学習

概要

Fashion-MNIST に対して、深層学習による分類を行う。学習データに対する損失関数の値と正解率の推移、テストデータに対する正解率と混同行列をしめし、結果を考察する。

プログラム

以下のプログラム３、４、５にそれぞれニューラルネットワークの定義部分と、学習部分、テスト部分を示す。今回使うFashionMNISTのデータセットでは、28*28の正規化された配列データとして保存されている。プログラム３では、28*28の画像データを10クラスに変換するクラスに変換するニューラルネット実装した。定義部分では、MyCNNというpytorchのnn.Moduleを継承したクラスを定義した。コンストラクタ内で、一層ごとに畳み込み層と正規化の層、活性化関数、プーリング層を定義した。

特徴抽出部分を一層ごとに説明する。まず、畳み込み層で入力の28*28の行列を畳み込んで32*32の大きさに変えてバッチ正規化のあとに活性化関数のReLUを適用している。その後、maxプーリングで10*10に次元を削減している。同様の操作を行い、更に4*4まで次元を削減する。

次に全結合層の説明をする。全結合層では、線形変換と活性化関数を用いて畳み込み層で抽出された特徴量から最終的なクラス分類を行う。全結合層の一層目では、特徴量を線形変換によって256次元に変換し、活性化関数のReLUを適用している。その後、256次元のベクトルを線形変換によって数字のクラス数（10次元）に変換する。その結果をLogSoftmaxで正規化を行う。

MyCNNの実行部であるforward関数では、入力画像を畳み込み層によって特徴抽出を行い、出力のテンソルの形状を変形、全結合層に入力し出力を返している。

```
import torch.nn as nn

class MyCNN(nn.Module):
	"""Some Information about MyCNN"""
	def __init__(self, num_classes):
		super().__init__()

		self.convs = nn.Sequential(
			# 28x28
			nn.Conv2d(1, 32, 3, 1, 1),
			nn.BatchNorm2d(32),
			nn.LeakyReLU(),
			nn.MaxPool2d(3, padding=1),

			# 10x10
			nn.Conv2d(32, 64, 3, 1, 1),
			nn.BatchNorm2d(64),
			nn.LeakyReLU(),
			nn.MaxPool2d(3, padding=1),
```

```
                # 4x4
        )

        self.fc = nn.Sequential(
                nn.Linear(64 * 4 * 4, 256),
                nn.LeakyReLU(),

                nn.Linear(256, num_classes),
                nn.LogSoftmax(dim=1)

        )
    def forward(self, x):
        h = self.convs(x)
        h = h.view(-1, 64 * 4 * 4)
        return self.fc(h)
```

プログラム３　ニューラルネットワークの定義部分

以下のプログラム４では、プログラム３で定義したクラスを生成し学習データによりニューラルネットワークの重みを調整している。最適化関数には、Adamを使用し学習率は0.01に設定した。epoch数は100に設定した。for文の中では、実際に画像をニューラルネットワークに入力し出力と解答の誤差を計算する。誤差逆伝播法によってニューラルネットワークの重みを更新している。学習が終わると、学習時に保存していた各エポックの正答率と誤差のグラフが出力される。グラフを図４と図５に示す。

```
    net = models.MyCNN(10)
    net.to(device)

    optimizer = torch.optim.Adam(net.parameters(), lr=1e-2)
    criterion = torch.nn.NLLLoss()

    print(net)

    num_epoch = 100
    since = time.time()

    _loss = []
    _acc = []
    for epoch in range(num_epoch):
        running_loss = 0.0
        running_acc = 0.0
        total = 0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
```

```
                    loss = criterion(outputs, labels)
                    loss.backward()
                    optimizer.step()

                    ##ここで精度計算
                    results = outputs.cpu().detach().numpy().argmax(axis=1)
                    running_acc += accuracy_score(labels.cpu().numpy(), results)
* len(inputs)
                    total += len(inputs)

                    running_loss += loss.item()

             running_acc /= total
             print('Loss: ',running_loss, 'ACC :', running_acc, epoch)
             _acc.append(running_acc)
             _loss.append(running_loss)

      print('Finished Training')
```

プログラム４　ニューラルネットの学習実行部分

以下のプログラム５では、学習した重みが適切であるかをテストするプログラムである。for内ではテストデータを読み込み、定義したニューラルネットワークと学習した重みを用いて実際にクラス分類を行う。正答率と誤差を計算する。クラス分類が終わったら、混同行列と全体の正答率と誤差を出力している。混同行列と全体の正答率と誤差を出力３に示す。

```
      net = models.MyCNN(10)

net.load_state_dict(torch.load('./save_models/fashion_mnist_classification.pth')
)
      net.to(device)
      net.eval()

      test_loss = 0
      target_all=[]
      result_all=[]

      with torch.no_grad():
            for data, target in testloader:
                  data, target = data.to(device), target.to(device)
                  output = net(data)
                  pred_cpu = output.cpu().detach().numpy().argmax(axis=1)
                  test_loss += F.nll_loss(output, target,
reduction='sum').item() # sum up batch loss
                  # すべてのテストデータの正解ラベルと推定されたラベルをnumpy配列に
格納
                  target_all = np.append(target_all, target.cpu().numpy())
                  result_all = np.append(result_all, pred_cpu)

      test_loss /= len(testloader.dataset)
      acc = accuracy_score(target_all, result_all)
```

```
      print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{}
({:.0f}%)\n'.format(
            test_loss, int(acc*len(testloader.dataset)),
len(testloader.dataset),
            100. * acc))

      c_mat= confusion_matrix(target_all, result_all)
      print(c_mat)
```

プログラム５　ニューラルネットの精度計算部分

実行結果

各エポックの正答率と誤差のグラフを以下の図４と図５に示す。出力３は混同行列と全体の正答率と誤差である。正答率は91%という結果になった。混同行列を見ると０と６を間違えることが多かった。０を６と間違えたものとその逆が88枚と114枚で合計202枚と多く間違えていた。

```
Test set: Average loss: 1.1226, Accuracy: 9073/10000 (91%)

[[840   0  20  17   5   1 114   0   3   0]
 [  3 982   2  12   0   0   0   0   1   0]
 [ 13   0 896   7  41   0  42   0   1   0]
 [ 20   4  15 877  31   0  50   0   3   0]
 [  3   1  78  15 868   0  35   0   0   0]
 [  0   0   0   0   0 983   0  14   0   3]
 [ 88   2  71  16  95   0 716   0  12   0]
 [  0   0   0   0   0   7   0 956   0  37]
 [  2   0   0   4   4   2   2   1 985   0]
 [  1   0   0   0   0   5   1  23   0 970]]
```

出力３　混同行列と全体の正答率と誤差

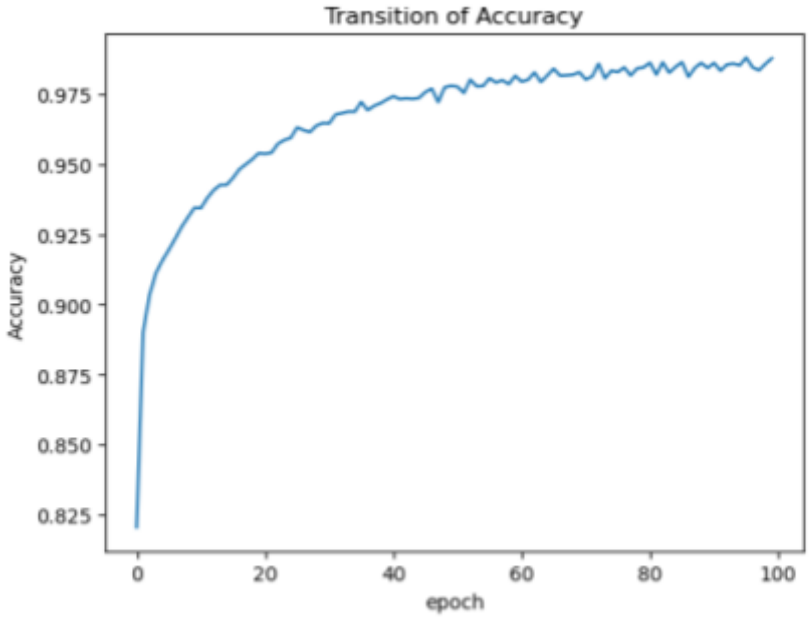

図4　学習データの正答率の推移

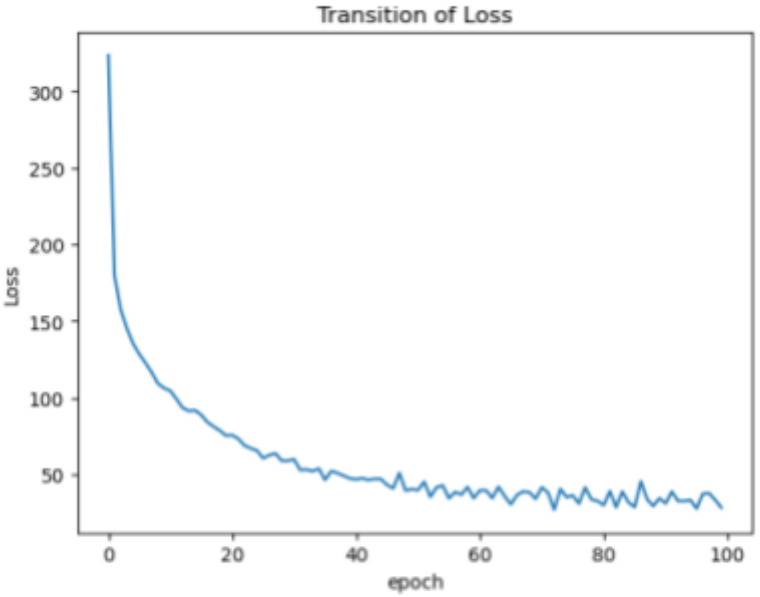

図5　学習データの誤差の推移

正答率の推移と誤差の推移を見るとちょうど上下で鏡合わせを居たようなグラフになった。最終的に、学習データの正答率は97%を超えており、誤差は初め300を超えていた数値が50を下回る様になった。

考察

混同行列から0と６はよく間違えることがわかった。これは０と６が図形的に似ていることに起因するものだと考えられる。特にMNISTでは、６が潰れていたり丸が大きかったりと人間が判断するのも難しいような画像もふくまれているので間違えやすいと考えられる。正答率の推移を見ると20 epoch目を超える時点で95%を超えていて誤差の推移を見ても40 epoch目以降はあまり誤差が減っていない。このことからも今回の学習パラメータでは60 epochを超えない範囲で学習させるのが効率的だとわかった。

まとめ

本レポートでは、クラス分類器と画像の次元削減、深層学習を使った画像分類の実験について報告した。どの実験でもパラメータの設定が重要であるとわかった。特にT-SNEと深層学習の実験では、精度だけでなく計算時間も関わってくるのでとても重要である。