レポート提出票

科目名: 情報工学実験３

実験課題名: 課題２　パターン認識

実施日: 2024年 5月 30日

学籍番号: 4622045

氏名: 小澤　翼

共同実験者:

1　背景

今回扱うパターン認識は、特徴を抽出し数式にする段階と特徴に基づいてデータをあらかじめ定められたクラスに分類する段階があり、特徴をクラスに分類するために識別器を作成する必要がある。

2　目的

線形判別分析のプログラムを実装し、その性質を理解する。

3　課題

3.1　課題 1

One-vs-One 方式に基づく多クラス線形判別分析を実装し，正解率を求めよ。
以下は使用したソースコードである。

ソースコード 1: 課題 1

```
#------------- 0: setosa vs 1: versicolor ---------------#
# サンプル抽出
train_ind_01 = np.squeeze(np.any([np.array(y_train == 0), np.array(y_train
    == 1)], axis=0))
y_train_01 = y_train[train_ind_01]
X_train_01 = X_train[train_ind_01]

# 線形判別分析
lda_01 = LDA(n_components=1) #
lda_01.fit(X_train_01, y_train_01)
p_pred_01 = lda_01.predict_proba(X_test) # 確信度

#------------- 0: setosa vs 2: virginica ----------------#
# サンプル抽出
train_ind_02 = np.squeeze(np.any([np.array(y_train == 0), np.array(y_train
    == 2)], axis=0))
y_train_02 = y_train[train_ind_02]
X_train_02 = X_train[train_ind_02]

# 線形判別分析
lda_02 = LDA(n_components=1) #
lda_02.fit(X_train_02, y_train_02)
p_pred_02 = lda_02.predict_proba(X_test) # 確信度

#----------- 1: versicolor vs 2: virginica -------------#
# サンプル抽出
train_ind_12 = np.squeeze(np.any([np.array(y_train == 1), np.array(y_train
    == 2)], axis=0))
y_train_12 = y_train[train_ind_12]
X_train_12 = X_train[train_ind_12]

```

```
# 線形判別分析
lda_12 = LDA(n_components=1) #
lda_12.fit(X_train_12, y_train_12)
p_pred_12 = lda_12.predict_proba(X_test) # 確信度

#------------------------課題 1--------------------------#
# 3つの 2クラス分類器が求めた確信度から、各クラスの確信度を計算
# クラスのどちらを見るかに注意
p_pred_0 = p_pred_01[:, 0] * p_pred_02[:, 0] # 0: setosa
p_pred_1 = p_pred_01[:, 1] * p_pred_12[:, 0] # 1: versicolor
p_pred_2 = p_pred_02[:, 1] * p_pred_12[:, 1] # 2: versinica

# 最大の確信度を持つクラスに分類
y_pred_One = np.argmax(np.vstack([p_pred_0, p_pred_1, p_pred_2]).T, axis=1)

#------------------------ 描画 -------------------------#
fig = plt.figure(3, dpi=150)
ax = fig.add_subplot(1, 1, 1)

# クラス 0のサンプルの描画
ind_0 = np.squeeze(np.array(y_train == 0))
plt.plot(X_train[ind_0,0], X_train[ind_0,1], 'ro', label='class0')

# クラス 1のサンプルの描画
ind_1 = np.squeeze(np.array(y_train == 1))
plt.plot(X_train[ind_1,0], X_train[ind_1,1], 'bo', label='class1')

# クラス 2のサンプルの描画
ind_2 = np.squeeze(np.array(y_train == 2))
plt.plot(X_train[ind_2,0], X_train[ind_2,1], 'go', label='class2')

# 0 vs 1の識別境界の描画
x_ = np.array([-2,2.3])
y_ = -1*(lda_01.intercept_[0] + x_*lda_01.coef_[0][0]) / lda_01.coef_[0][1]
plt.plot(x_, y_, 'm-', label='0 vs 1')

# 0 vs 2の識別境界の描画
x_ = np.array([-2,2.3])
y_ = -1*(lda_02.intercept_[0] + x_*lda_02.coef_[0][0]) / lda_02.coef_[0][1]
plt.plot(x_, y_, color='orangered', label='0 vs 2')

# 1 vs 2の識別境界の描画
x_ = np.array([-2,2.3])
y_ = -1*(lda_12.intercept_[0] + x_*lda_12.coef_[0][0]) / lda_12.coef_[0][1]
plt.plot(x_, y_, 'c-', label='1 vs 2')

plt.xlim(-2.0,3.0)
plt.ylim(-3.0,3.5)

plt.legend()
plt.show()

# 識別精度の計算
```

```
82 print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_One))
```

このソースコードの結果は以下の通りである。

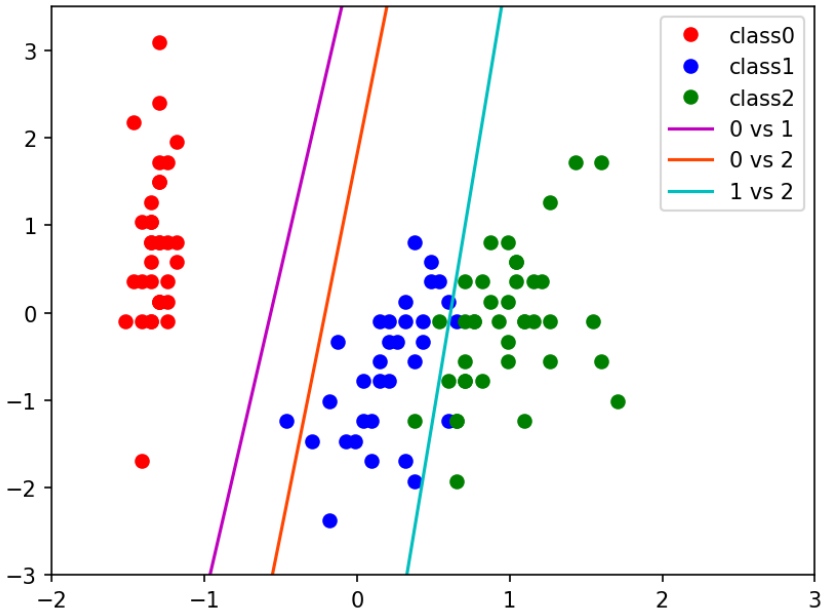

図 1: 課題 1 の結果

ソースコード 2: One vs One の正解率

```
1  Accuracy: 0.93
```

3.2 課題 2

One-vs-All 方式に基づく多クラス線形判別分析を実装し，正解率を求めよ。
以下は使用したソースコードである。

ソースコード 3: 課題 2

```
1  ------------- 0: setosa vs 1: versicolor & 2: virginica ---------------#
2  # 便宜的にクラス 2をクラス 1に置換
3  y_train_0 = y_train.copy() # 深いコピー
4  y_train_0[y_train_0 == 2] = 1
5
6  # 線形判別分析
7  lda_0 = LDA(n_components=1) #
8  lda_0.fit(X_train, y_train_0)
9  p_pred_0 = lda_0.predict_proba(X_test) # 確信度
10
11 #------------- 1: versicolor vs 0: setosa & 2: virginica ---------------#
12 # 便宜的にクラス 2をクラス 0に置換
13 y_train_1 = y_train.copy() # 深いコピー
14 y_train_1[y_train_1 == 2] = 0
15
16 # 線形判別分析
```

```
lda_1 = LDA(n_components=1) #
lda_1.fit(X_train, y_train_1)
p_pred_1 = lda_1.predict_proba(X_test) # 確信度

#------------- 2: virginica vs 0: setosa & 1: versicolor ---------------#
# 便宜的にクラス0をクラス1に置換
y_train_2 = y_train.copy() # 深いコピー
y_train_2[y_train_2 == 0] = 1

# 線形判別分析
lda_2 = LDA(n_components=1) #
lda_2.fit(X_train, y_train_2)
p_pred_2 = lda_2.predict_proba(X_test) # 確信度

#------------------------ 課題2 --------------------------#
# 各2クラス分類器が求めた確信度から、各クラスの確信度を計算 (np.
    argmax を使うと良い)
y_pred_All = np.argmax(np.vstack([p_pred_0[:, 0], p_pred_1[:, 0], p_pred_2
    [:, 0]]).T, axis=1)

#------------------------ 描画 -------------------------#
fig = plt.figure(4, dpi=150)
ax = fig.add_subplot(1, 1, 1)

# クラス0のサンプルの描画
ind_0 = np.squeeze(np.array(y_train == 0))
plt.plot(X_train[ind_0,0], X_train[ind_0,1], 'ro')

# クラス1のサンプルの描画
ind_1 = np.squeeze(np.array(y_train == 1))
plt.plot(X_train[ind_1,0], X_train[ind_1,1], 'bo')

# クラス2のサンプルの描画
ind_2 = np.squeeze(np.array(y_train == 2))
plt.plot(X_train[ind_2,0], X_train[ind_2,1], 'go')

# 0 vs 1,2の識別境界の描画
x_ = np.array([-2,2.3])
y_ = -1*(lda_0.intercept_[0] + x_*lda_0.coef_[0][0]) / lda_0.coef_[0][1]
plt.plot(x_, y_, 'r-', label='0 vs All')

# 1 vs 0,2の識別境界の描画
x_ = np.array([-2,2.3])
y_ = -1*(lda_1.intercept_[0] + x_*lda_1.coef_[0][0]) / lda_1.coef_[0][1]
plt.plot(x_, y_, 'b-', label='1 vs All')

# 2 vs 0,1の識別境界の描画
x_ = np.array([-2,2.3])
y_ = -1*(lda_2.intercept_[0] + x_*lda_2.coef_[0][0]) / lda_2.coef_[0][1]
plt.plot(x_, y_, 'g-', label='2 vs All')

plt.xlim(-2.0,2.3)
plt.ylim(-3.0,3.5)
```

```
68
69 plt.legend()
70 plt.show()
71
72 # 識別精度の計算
73 print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_All))
```

このソースコードの結果は以下の通りである。

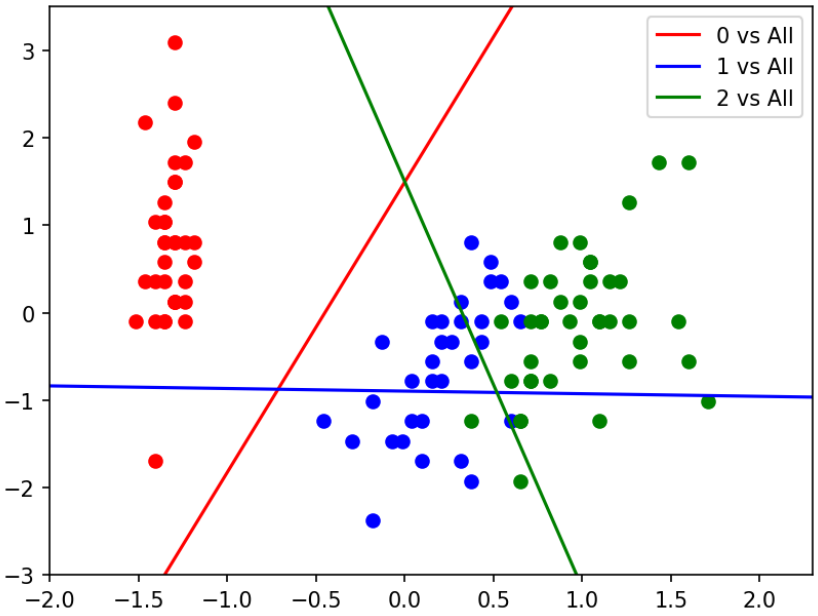

図 2: 課題 2 の結果

ソースコード 4: One vs All の正解率

```
1  Accuracy: 0.62
```

3.3 課題 3

One vs One の方の図では、現れた線でほとんどの class を区別することが出来ているので、かなり優れているといえると考える。一方、One vs All の方の図では、ある程度区別する事出来ているが、青色の class が One vs One よりも区別することが出来ていないように見える。また、測定した正解率からも One vs One の方が優れていることが分かる。このようになった原因としては、データセットの class0,1,2 がまとまって左から横並びになっているからだと思われる。例えば、x=-2 0,y=-3 0 の範囲に class0 の集合、x=-0.5 1.5,y=0 3 の範囲に class1 の集合、x=1 3,y=-3 0 の範囲に class2 の集合のデータセットがあった時、One vs All の方が優れている可能性があると思われる。

したがって、今回のデータセットでは One vs One の方が精度が高いと思われる。ただ、どっちが優れているかはデータセットによって変わって来るので一概にどちらの方が精度が高いかは言えない。

4　まとめ

線形判別分析のプログラムを実装し、One vs One と One vs All の結果を見比べることで、どのような図が算出されれば正確といえるかを理解することが出来た。

参考文献

[1] Qiita:　「機械学習のアルゴリズム（2 クラス分類から多クラス分類へ）」, https://qiita.com/hiro88hyo/items/683d7d9feab0f33d69ea