智慧化企業整合

Final Project

鳶尾花預測

108034544 楊雨澄

目錄

01.

5W1H

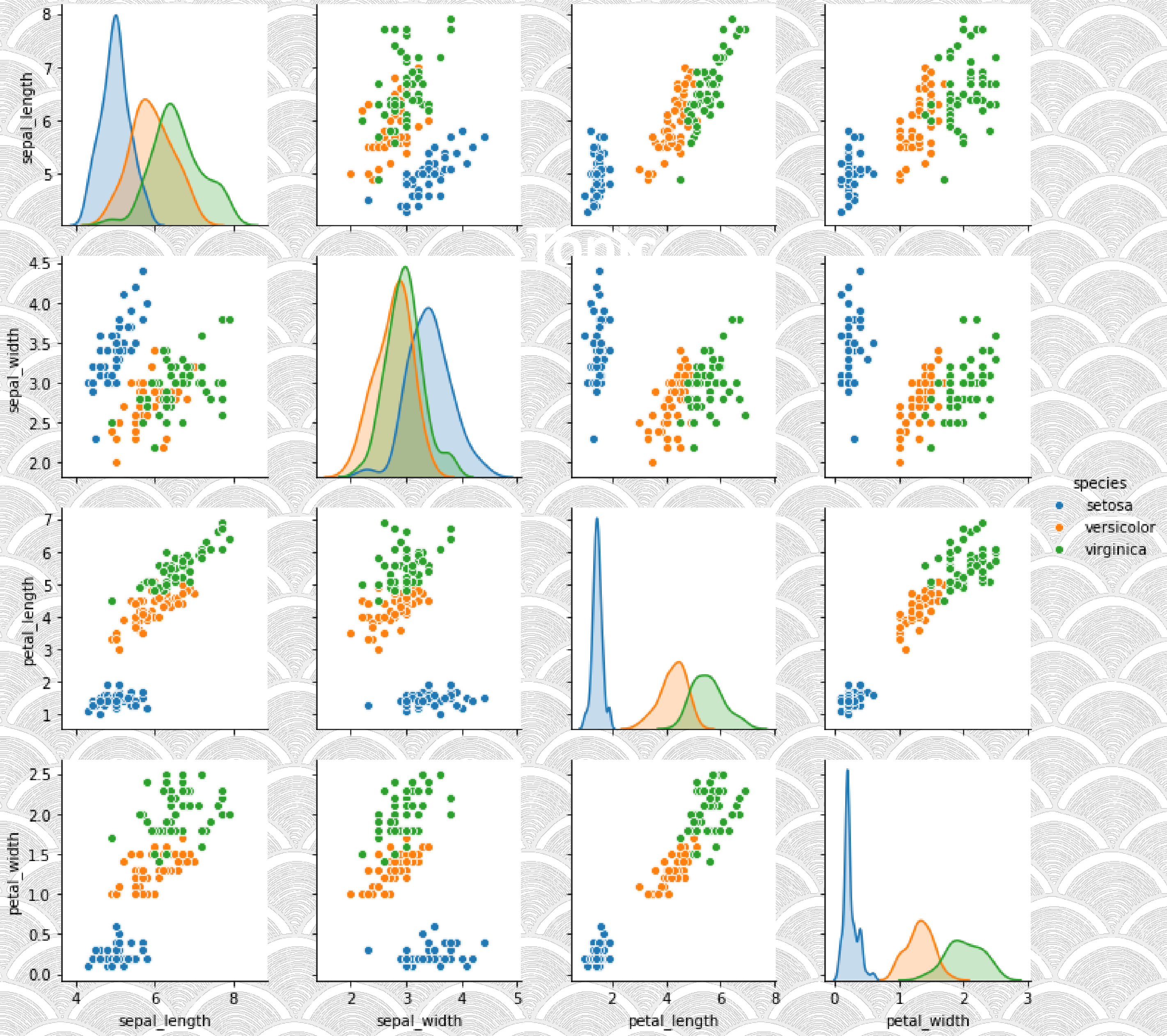
主題


5W1H

- What：提供遇到無法確認東西的人可能物種的方法
- Who：對動物植物有興趣的人
- Why：解決無法用肉眼及大致外觀特徵辨認的問題
- Where：任何地點
- When：遇到不確定的物種時
- How：利用分類器判別物種！

目標

- 透過建立CNN模型，利用四種特徵來辨識該花朵種類，並計算準確率
- Dataset使用：Iris
- 該資料及有150筆花朵資料，包含三種品種，每一筆資料皆有四種特徵


sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica

流程

資料前處理

建立model

訓練model

測試model

02.

資料前處理

Data-Preprocessing

Data-preprocessing

```
IRIS = pd.read_csv('iris.csv')
IRIS.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris–setosa
1	4.9	3.0	1.4	0.2	Iris–setosa
2	4.7	3.2	1.3	0.2	Iris–setosa
3	4.6	3.1	1.5	0.2	Iris–setosa
4	5.0	3.6	1.4	0.2	Iris–setosa

Data-preprocessing

```
target = 'class'
features = list(iris.columns)
features.remove(target)
Class = iris[target].unique()
```

```
Class_dict = dict(zip(Class, range(len(Class))))
iris['target'] = IRIS[target].apply(lambda x: Class_dict[x])
lb = LabelBinarizer()
lb.fit(list(Class_dict.values()))
labels = lb.transform(iris['target'])
y_labels = []
for i in range(labels.shape[1]):
    y_labels.append('y' + str(i))
    iris['y' + str(i)] = labels[:, i]
```

Data-preprocessing

分割資料為測試集和訓練集

```
train_x, test_x, train_y, test_y = train_test_split(IRIS[features], IRIS[y_bin_labels], train_size=0.7, test_size=0.3, random_state=0)
return train_x, test_x, train_y, test_y, Class_dict
```

把三個種類的資料編號為0, 1, 2

```
[73] print (Class_dict)

{'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
```

03.

模型架構

Model Structure

Model Structure

```
[48] init = K.initializers.glorot_uniform(seed=1)
     adam = K.optimizers.Adam()
     adagrad = K.optimizers.Adagrad()
     model = K.models.Sequential()
     model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
     model.add(K.layers.Dense(units=6, kernel_initializer=init, activation='relu'))
     model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
     model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

CNN parameter	Value
Number of convolution layers	3
Activation function	Relu
Loss function	Categorical_crossentropy
Optimizer	adam
Classification function of the output layer	Softmax

04.

改善過程及訓練

Model Improvement & training

Model Improvement

增加Model卷積層數

```
[95]  #定義模型
      init = K.initializers.glorot_uniform(seed=1)
      adam = K.optimizers.Adam()
      adagrad = K.optimizers.Adagrad()
      model = K.models.Sequential()
      model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='sigmoid'))
      model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

Test Accuracy: 66.67%

```
[75]  #定義模型
      init = K.initializers.glorot_uniform(seed=1)
      adam = K.optimizers.Adam()
      adagrad = K.optimizers.Adagrad()
      model = K.models.Sequential()
      model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='sigmoid'))
      model.add(K.layers.Dense(units=15, input_dim=4, kernel_initializer=init, activation='sigmoid'))
      model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

Test Accuracy: 77.78%

Model Improvement

更改Activation Function → Relu

```
[75]  #定義模型
      init = K.initializers.glorot_uniform(seed=1)
      adam = K.optimizers.Adam()
      adagrad = K.optimizers.Adagrad()
      model = K.models.Sequential()
      model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='sigmoid'))
      model.add(K.layers.Dense(units=15, input_dim=4, kernel_initializer=init, activation='sigmoid'))
      model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

Test Accuracy: 77.78%

```
[101] #定義模型
      init = K.initializers.glorot_uniform(seed=1)
      adam = K.optimizers.Adam()
      adagrad = K.optimizers.Adagrad()
      model = K.models.Sequential()
      model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='relu'))
      model.add(K.layers.Dense(units=15, kernel_initializer=init, activation='relu'))
      model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

Test Accuracy: 93.33%

Model Improvement

減少卷積層Units → 10->5, 15->6

```
[101]
#定義模型
init = K.initializers.glorot_uniform(seed=1)
adam = K.optimizers.Adam()
adagrad = K.optimizers.Adagrad()
model = K.models.Sequential()
model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='relu'))
model.add(K.layers.Dense(units=15, kernel_initializer=init, activation='relu'))
model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

Test Accuracy: 93.33%

```
[83]
#定義模型
init = K.initializers.glorot_uniform(seed=1)
adam = K.optimizers.Adam()
adagrad = K.optimizers.Adagrad()
model = K.models.Sequential()
model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
model.add(K.layers.Dense(units=6, input_dim=4, kernel_initializer=init, activation='relu'))
model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

Test Accuracy: 95.56%

Model Improvement

更改優化器 Adagrad → Adam

```
[83]    #定義模型
        init = K.initializers.glorot_uniform(seed=1)
        adam = K.optimizers.Adam()
        adagrad = K.optimizers.Adagrad()
        model = K.models.Sequential()
        model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=6, input_dim=4, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

Test Accuracy: 95.56%

```
[ ]     #定義模型
        init = K.initializers.glorot_uniform(seed=1)
        adam = K.optimizers.Adam()
        model = K.models.Sequential()
        model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=6, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

Test Accuracy: 97.78%

Improve Steps

Steps	Accuracy
原始 **model**	**66.67%**
原始 **model +**層數	66.67% -> **77.78%**（↑11.11）
改 **Activation**	77.78% -> **93.33%**（↑15.55）
改 **Units**	93.33% -> **95.56%**（↑2.23）
改 **Optimizer**	95.56% -> **97.78%**（↑2.22）

Model Training

Epochs = 100
batch = 1
Training Time = 11.573 sec

```
import time
start_time = time.time()
b_size = 1
epochs = 100
print("Starting training")
history = model.fit(train_x, train_y, batch_size=b_size, epochs=epochs, shuffle=True, verbose=1)
print("Training finished \n")
print("--- %s seconds ---" % (time.time()-start_time))
```

```
Epoch 96/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0811 - acc: 0.9524
Epoch 97/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0310 - acc: 0.9810
Epoch 98/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0447 - acc: 0.9905
Epoch 99/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0285 - acc: 0.9905
Epoch 100/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0412 - acc: 0.9714
Training finished

--- 11.572912693023682 seconds ---
```

05.

結果&討論

Result & Discussion

Result

```
[26]  #評估模型
      eval = model.evaluate(test_x, test_y, verbose=0)
      print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" % (eval[0], eval[1] * 100) )

Evaluation on test data: loss = 0.110360 accuracy = 97.78%
```

Test Accuracy: 97.78%

Testing Result

(6.1, 3.1, 5.1, 1.1)

```
np.set_printoptions(precision=4)
new = np.array([[6.1, 3.1, 5.1, 1.1]], dtype=np.float32)
predicted = model.predict(new)
print("用模型預測四個特徵: ")
print(new)
print("\n預測的softmax向量為: ")
print(predicted)
new_dict = {v:k for k,v in Class_dict.items()}
print("\n預測的種類為: ")
print(new_dict[np.argmax(predicted)])
```

```
用模型預測四個特徵:
[[6.1 3.1 5.1 1.1]]

預測的softmax向量為:
[[0.0063 0.9676 0.0261]]

預測的種類為:
Iris-versicolor
```

Versicolor

(2.2, 3.2, 3.1, 5.2)

```
new = np.array([[2.2, 3.2, 3.1, 5.2]], dtype=np.float32)
predicted = model.predict(new)
print("\n用模型預測四個特徵: ")
print(new)
print("\n預測的softmax向量為: ")
print(predicted)
new_dict = {v:k for k,v in Class_dict.items()}
print("\n預測的種類為: ")
print(new_dict[np.argmax(predicted)])
```

```
用模型預測四個特徵:
[[2.2 3.2 3.1 5.2]]

預測的softmax向量為:
[[0.0027 0.0378 0.9595]]

預測的種類為:
Iris-virginica
```

Virginica

06.

結論 & 未來

Conclusion & Future work

Conclusion & Future Work

Conclusion

從這個研究可以得知，用特徵資料來判斷並且建立深度學習模型為一個可行的方式。但是其資料集較為稀少，可能造成其準確率較無法提升。

Future Work

未來研究方向希望增加資料集複雜度。另外透過圖像預測，並搭配相關特徵，來做混合預測。

Thank you for your attention