DM & ML - Final Assignment : การแก้ปัญหา Overfit ในการวิเคราะห์ข้อมูล

ในการแก้ไขปัญหา overfit ครั้งนี้ ผมจะทำการแก้ไขผ่านการเขียน Python จาก Google Colab โดยใช้ชุดข้อมูลที่ชื่อว่า Glass Classification จาก Kaggle ซึ่งประกอบไปด้วย 214 records และ 10 Attributes คือ

1. Id number : ตั้งแต่ 1 – 214
2. RI : Refractive Index หรือ ดัชนีการหักเหของแสง
3. Na : Sodium
4. Mg : Magnesium
5. Al : Aluminum
6. Si : Silicon
7. K : Potassium
8. Ca : Calcium
9. Ba : Barium
10. Fe : Iron
11. Type of glass (หรือ ชนิดของแก้ว) ในรูป Class Attribute โดยกำหนดตัวเลขดังต่อไปนี้

-- 1 building_windows_float_processed
-- 2 building_windows_non_float_processed
-- 3 vehicle_windows_float_processed
-- 4 vehicle_windows_non_float_processed (ไม่มีข้อมูลส่วนนี้ในชุดข้อมูลชุดนี้)
-- 5 containers
-- 6 tableware
-- 7 headlamps

(โดยให้ Attribute ที่ 3-10 อยู่ในหน่วย %wt หรือร้อยละต่อน้ำหนัก)

โดยวัตถุประสงค์ของการวิเคราะห์ข้อมูลชุดนี้ก็เพื่อที่จะสร้างโมเดลในการทำนายชนิดของแก้ว จากค่าพารามิเตอร์ต่าง ๆ ตามที่กำหนด

ในขั้นตอนแรกนั้น เราจะเริ่มจากการสำรวจข้อมูลเบื้องต้น (EDA) กันก่อน

```
[117]  1 glass_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   RI      214 non-null    float64
 1   Na      214 non-null    float64
 2   Mg      214 non-null    float64
 3   Al      214 non-null    float64
 4   Si      214 non-null    float64
 5   K       214 non-null    float64
 6   Ca      214 non-null    float64
 7   Ba      214 non-null    float64
 8   Fe      214 non-null    float64
 9   Type    214 non-null    int64
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
```

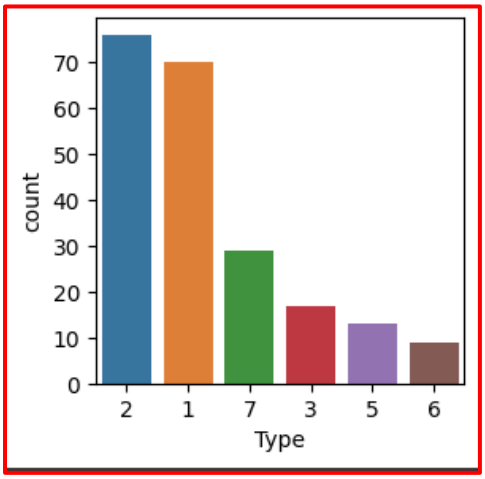
```
[114]  1 glass_df.shape

(214, 10)

       1 glass_df.isnull().sum()

RI      0
Na      0
Mg      0
Al      0
Si      0
K       0
Ca      0
Ba      0
Fe      0
Type    0
dtype: int64
```

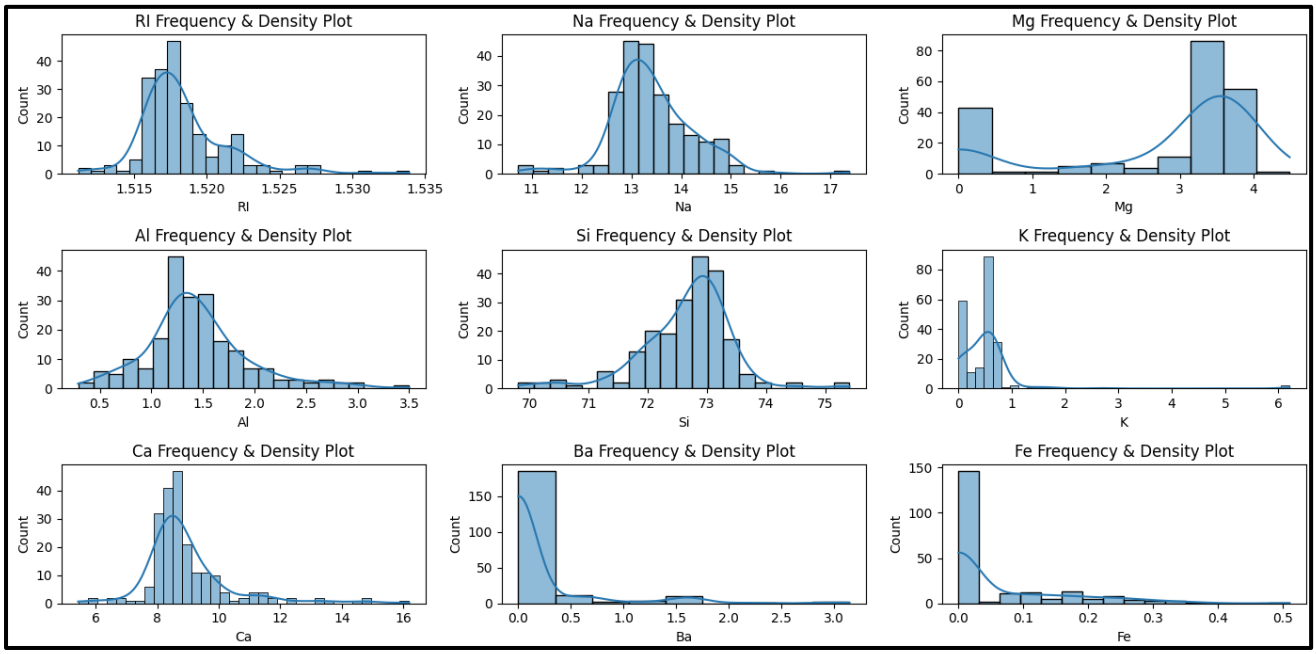
และเราพบว่า สัดส่วนของประเภทต่าง ๆ และการกระจายข้อมูล เป็นไปตามด้านล่าง

```
[116]  1 glass_df.head()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

```
Overall Data Distribution as below
------------------------------------------------------------
building_windows_non_float_processed      : 76  or 35.51%
building_windows_float_processed          : 70  or 32.71%
headlamps                                 : 29  or 13.55%
vehicle_windows_float_processed           : 17  or 7.94%
containers                                : 13  or 6.07%
tableware                                 :  9  or 4.21%
```

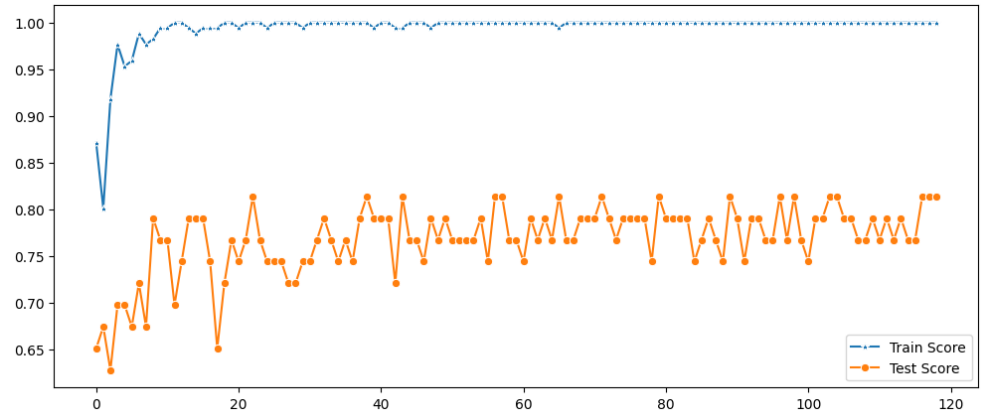




โดยหากเราต้องการให้ Model มีประสิทธิภาพดีขึ้นนั้น เราจำเป็นที่จะต้องทำการปรับค่า Hyperparameter ต่าง ๆ เพื่อให้ model สามารถทำนาย unseen data ได้แม่นยำขึ้น (ลดปัญหา Overfit) ซึ่งตัวอย่างวิธีการดังกล่าว เป็นไปดังต่อไปนี้

1) Splitting Data by Cross Validation Method

การใช้ cross validation number ที่จำนวนมากขึ้นจนถึงค่าหนึ่ง ก็จะทำให้ได้ค่า Model Cross Validation Accuracy ที่เพิ่มขึ้น ซึ่งนั่นก็หมายถึงโอกาสที่ข้อมูลจะถูกนำไปใช้ในการเรียนรู้ก็จะครอบคลุมขึ้น ซึ่งนั่นก็หมายถึงว่าโอกาสในการเกิด overfit ก็จะน้อยลง




```
[71]  1 X_train , X_test , y_train , y_test = train_test_split(X,y ,test_size = 0.3 , random_state = 1994 , stratify = y )
```

```
[72]  1
      2 rfc = RandomForestClassifier(n_estimators = 250, criterion = 'entropy')
      3 rfc.fit(X_train,y_train)
      4
      5 y_pred = rfc.predict(X_test)
      6 y_pred
```

```
array([1, 1, 7, 7, 1, 1, 1, 2, 1, 2, 2, 1, 2, 7, 1, 7, 1, 7, 1, 6, 2, 2,
       2, 1, 2, 1, 2, 2, 1, 5, 1, 6, 7, 1, 2, 2, 7, 2, 7, 1, 1, 1, 2, 1,
       1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 3, 2, 1, 5, 1, 3, 1, 6, 2, 1, 1])
```

```
[77]  1 print("using CV = 2",cross_val_score(rfc, X, y, cv=2)*100)
      2 print("using CV = 5",cross_val_score(rfc, X, y, cv=5)*100)
      3 print("using CV = 10",cross_val_score(rfc, X, y, cv=10)*100)
      4
```

```
using CV = 2 [59.81308411 56.07476636]
using CV = 5 [74.41860465 72.09302326 65.11627907 69.76744186 76.19047619]
using CV = 10 [63.63636364 72.72727273 81.81818182 72.72727273 61.9047619  85.71428571
 90.47619048 57.14285714 80.95238095 85.71428571]
```

```
for cv_i in (2,5,7,10,15,20,30,50):
    accuracies_i_percentage = cross_val_score(rfc, X_train, y_train, cv = cv_i)
    print(f'Using CV = {cv_i} Accuracy (mean):{accuracies_i_percentage.mean()*100:.2f}')
```

```
Using CV = 2 Accuracy (mean):73.80
Using CV = 5 Accuracy (mean):76.51
Using CV = 7 Accuracy (mean):77.12
Using CV = 10 Accuracy (mean):78.48
Using CV = 15 Accuracy (mean):77.11
Using CV = 20 Accuracy (mean):77.59
Using CV = 30 Accuracy (mean):77.17
Using CV = 50 Accuracy (mean):79.33
```

```
Random Forest F1 Score: 70.77
Accuracy (mean):77.81
std: % 6.86
```

จากภาพ เป็นการเปรียบเทียบผลลัพธ์การใช้ Cross Validation ที่ 2,5,7,10,15,20,30 และ 50-Fold CV ซึ่งจะเห็นได้ชัดว่า Cross Validation Score (Validation Accuracy) มีแนวโน้มดีขึ้นหากใช้ CV ที่เพิ่มขึ้นถึงค่าหนึ่ง หลังจากนั้นอาจคงที่หรือด้อยลง ซึ่งจากผลการรันตามภาพนั้น ค่าที่เหมาะที่สุดคือ cv ในช่วงตั้งแต่ 5-10 เนื่องจาก เป็นการแบ่งที่ได้ score ที่ยอมรับได้ และไม่ได้ใช้เวลาทำงานมากจนเกินไป

2) Removing Layers / Changing number of unit per layers

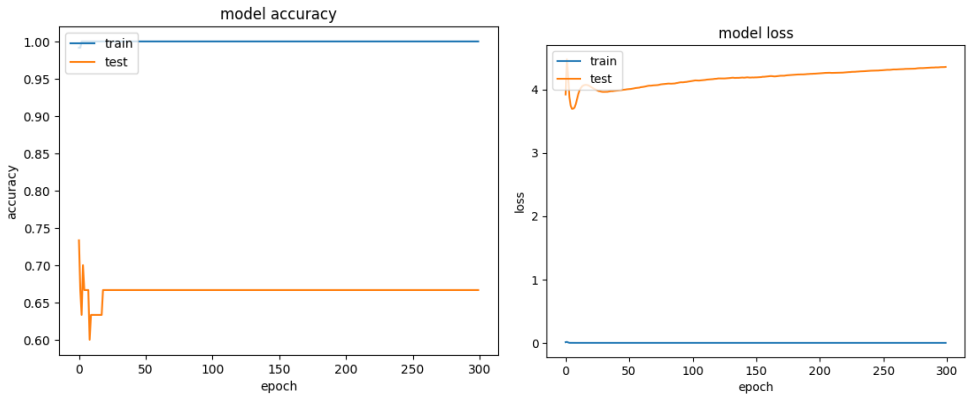
สำหรับการสร้างโมเดลพยากรณ์จาก Algorithm Neural Network .ในกรณีนี้ ได้กำหนดพารามิเตอร์ตั้งต้นไว้ตามภาพ

```
model = Sequential()
model.add(Dense(45, activation="relu", input_dim=9))
model.add(Dense(30, activation="relu"))
model.add(Dense(25, activation="relu"))
model.add(Dense(22, activation="relu"))
model.add(Dense(18, activation="relu"))
model.add(Dense(15, activation="relu"))
model.add(Dense(10, activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(6, activation="softmax"))
```

```
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy" ,
              metrics=['accuracy'])
```

```
history = model.fit(X_train_ss,y_train,
                    epochs=300,
                    batch_size=30 ,
                    validation_split=0.3,
                    verbose=1)
```

```
Epoch 272/300
Epoch 120/300
4/4 [==============================] - 0s 11ms/step - loss: 3.1287e-05 - accuracy: 1.0000 - val_loss: 7.5533 - val_accuracy: 0.6889
Epoch 121/300
4/4 [==============================] - 0s 8ms/step - loss: 3.1162e-05 - accuracy: 1.0000 - val_loss: 7.5539 - val_accuracy: 0.6889
Epoch 122/300
4/4 [==============================] - 0s 10ms/step - loss: 3.0978e-05 - accuracy: 1.0000 - val_loss: 7.5547 - val_accuracy: 0.6667
Epoch 123/300
4/4 [==============================] - 0s 9ms/step - loss: 3.0805e-05 - accuracy: 1.0000 - val_loss: 7.5557 - val_accuracy: 0.6667
Epoch 124/300
4/4 [==============================] - 0s 9ms/step - loss: 3.0662e-05 - accuracy: 1.0000 - val_loss: 7.5568 - val_accuracy: 0.6667
```



โดยจากการันจนครบ 300 epochs พบว่า training accuracy อยู่ที่ประมาณ 66.67 %

และเมื่อนำมาทำนายผลลัพธ์ และจะได้ค่าจากการทำนายคือ

```
y_pred = y_pred.argmax(axis=1)
y_pred
```

```
array([1, 0, 0, 1, 1, 5, 0, 0, 3, 2, 0, 0, 1, 1, 0, 0, 0, 5, 0, 5, 0, 2,
       3, 0, 1, 0, 3, 5, 0, 3, 1, 0, 0, 1, 5, 1, 5, 1, 0, 0, 1, 0, 5, 4,
       0, 4, 0, 0, 2, 0, 1, 0, 0, 4, 2, 0, 1, 0, 1, 1, 2, 0, 0, 5, 0])
```

ซึ่งผลลัพธ์จากการทำนายจะได้ความแม่นยำของโมเดลประมาณ 75.38% (difference 8.71%)

```
1 from sklearn.metrics import classification_report
2 print(classification_report(y_test,y_pred))
3 print("accuracy          (%) : ",round(accuracy_score(y_test,y_pred)*100,2)   )
4
5
```

```
              precision    recall  f1-score   support

           0       0.67      0.86      0.75        21
           1       0.73      0.70      0.71        23
           2       1.00      0.20      0.33         5
           3       0.80      1.00      0.89         4
           4       1.00      0.67      0.80         3
           5       1.00      0.89      0.94         9

    accuracy                           0.75        65
   macro avg       0.87      0.72      0.74        65
weighted avg       0.78      0.75      0.74        65

accuracy          (%) :  75.38
```

แต่ถ้าเราลองปรับ Hidden layer ก็จะได้ผลลัพธ์ดังต่อไปนี้

2.1) Remove 5 Hidden layers

```
1 # code ด้านล่างที่ comment สีเขียว คือ hidden layer ที่ถูกลบไป
2
3 model = Sequential()
4 model.add(Dense(45, activation="relu", input_dim=9))
5 #model.add(Dense(30, activation="relu"))
6 #model.add(Dense(25, activation="relu"))
7 #model.add(Dense(22, activation="relu"))
8 #model.add(Dense(18, activation="relu"))
9 #model.add(Dense(15, activation="relu"))
10 model.add(Dense(10, activation="relu"))
11 model.add(Dense(8, activation="relu"))
12 model.add(Dense(6, activation="softmax"))
```

```
              precision    recall  f1-score   support

           0       0.63      0.81      0.71        21
           1       0.71      0.65      0.68        23
           2       0.50      0.20      0.29         5
           3       1.00      0.75      0.86         4
           4       0.75      1.00      0.86         3
           5       1.00      0.89      0.94         9

    accuracy                           0.72        65
   macro avg       0.77      0.72      0.72        65
weighted avg       0.73      0.72      0.71        65

accuracy          (%) :  72.31
```

จะได้ training accuracy อยู่ที่ 66.67% เท่ากับพารามิเตอร์ตั้งต้น / testing Accuracy อยู่ที่ 72.31% (difference 5.64%)

2.2) ลดจำนวณ Neuron ในแต่ละ layer ลง 50%

```
1 model = Sequential()
2 model.add(Dense(45, activation="relu", input_dim=9))
3 model.add(Dense(15, activation="relu")) #  before adjust - dense = 45
4 model.add(Dense(13, activation="relu")) #  before adjust - dense = 30
5 model.add(Dense(11, activation="relu")) #  before adjust - dense = 25
6 model.add(Dense(9, activation="relu")) #  before adjust - dense = 22
7 model.add(Dense(8, activation="relu")) #  before adjust - dense = 18
8 model.add(Dense(5, activation="relu")) #  before adjust - dense = 15
9 model.add(Dense(4, activation="relu")) #  before adjust - dense = 8
10 model.add(Dense(6, activation="softmax"))
```

```
1 print(classification_report(y_test,y_pred))
2 print("accuracy(%) : ",round(accuracy_score(y_test,y_pred)*100,2)   )
```

```
              precision    recall  f1-score   support

           0       0.67      0.86      0.75        21
           1       0.75      0.39      0.51        23
           2       0.18      0.40      0.25         5
           3       0.50      0.25      0.33         4
           4       0.75      1.00      0.86         3
           5       0.89      0.89      0.89         9

    accuracy                           0.63        65
   macro avg       0.62      0.63      0.60        65
weighted avg       0.68      0.63      0.63        65

accuracy(%) :  63.08
```

จะได้ training accuracy อยู่ที่ 71.11% / testing Accuracy อยู่ที่ 63.08% (difference 8.03%)

ซึ่งจะเห็นว่า ทั้งวิธีการที่ 2.1 และ 2.2 ต่างทำให้ค่า Difference ระหว่าง training และ testing accuracy ลดลงจากวิธีการตั้งต้น หรืออาจกล่าวได้ว่า การปรับลด node ใน hidden layer หรือแม้กระทั่งการปรับลดจำนวน hidden layer ก็จะทำให้ overfitting ลดลงนั่นเอง

3) Dropout

จากข้อ 2 ทำการแก้ไขพารามิเตอร์เพิ่มเติม โดยเพิ่ม dropout บริเวณแต่ละ hidden layer แต่รายละเอียดอื่น ๆ คงเดิม ดังต่อไปนี้

```
model = Sequential()
model.add(Dense(45, activation="relu", input_dim=9))
model.add(Dense(30, activation="relu"))
model.add(Dense(25, activation="relu"))
model.add(Dense(22, activation="relu"))
model.add(Dense(18, activation="relu"))
model.add(Dense(15, activation="relu"))
model.add(Dense(10, activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(6, activation="softmax"))
```

```
model = Sequential()
model.add(Dense(45, activation="relu", input_dim=9))
model.add(Dense(30, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(25, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(22, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(18, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(15, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(10, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(8, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(6, activation="softmax"))
```

```
4/4 [==============================] - 0s 10ms/step - loss: 0.6175 - accuracy: 0.7788 - val_loss: 1.5921 - val_accuracy: 0.6222
<keras.callbacks.History at 0x7f66ce453820>
```

	precision	recall	f1-score	support
0	0.65	0.81	0.72	21
1	0.65	0.65	0.65	23
2	0.00	0.00	0.00	5
3	0.17	0.25	0.20	4
4	0.00	0.00	0.00	3
5	0.80	0.89	0.84	9
accuracy			0.63	65
macro avg	0.38	0.43	0.40	65
weighted avg	0.56	0.63	0.59	65

accuracy(%) : 63.08

จะได้ training accuracy อยู่ที่ 62.22% / testing Accuracy อยู่ที่ 63.08% (difference 0.86 %)

ซึ่งจะเห็นว่า วิธีการดังกล่าวนี้ทำให้ค่า Difference ระหว่าง training และ testing accuracy ลดลงจากวิธีการตั้งต้น หรืออาจกล่าวได้ว่า การ dropout หรือการสุ่มตัดนิวรอลบางตัวออกไประหว่างการเรียนรู้ ก็จะทำให้ overfitting ลดลงเช่นเดียวกัน