

# 데이터 사이언스 핵심 기말고사

글로벌소프트웨어학과

2014671038

김진수

## 목차

### 1. 데이터 소개 및 기계학습의 목표

- A. 데이터 소개
- B. 기계학습의 목표

### 2. 모델 구축 및 실험과정

- A. Data Scaling
- B. 데이터 분할
- C. ANN
- D. Random Forest
- E. Decision Tree

### 3. 최종 모델

### 4. 결론

# 1.데이터 소개 및 기계학습의 목표

## ※ 데이터 소개

- 위스콘신 유방암 진단 데이터이다.
- 컬럼 개수 : 32개 (ID, 진단결과, 30개의 데이터 값)
- 특질의 종류(설명)  
Radius (반경), texture(질감), perimeter(둘레), area(면적), smoothness(매끄러움),  
compactness(조그만 정도), concavity(오목함(윤곽의 오목한 부분의 정도)), symmetry(대칭),  
fractal dimension(차원) \_mean(각 특질의 평균 값을 의미한다).  
\_se(Standard Error로 표준오차를 의미한다)  
\_worst(각 세포별 구분들에서 제일 큰 3개의 값을 평균)
- 데이터 수의 개수 : 569명의 데이터
- 양성 비율 : 357 명
- 악성 비율 : 212 명

## ※ 기계학습의 목표

- 유방암 진단 사진으로부터 측정된 종양의 특징 값과 결과값을 사용하여 종양이 양성인지 악성인지를 정확히 판별하수 있도록 학습시킨다.
- 학습시킨 머신러닝을 Accuracy & ROC CURVE를 사용하여 평가하여 가장 좋은 머신러닝을 선택한다.

## 2. 모델 구축 및 실험과정

※ **Data Scaling**(\* ANN(10개의 hidden node가 있는 2개의 hidden layer 기준))

- Data scaling을 하지 않았을 때의 결과 값

: Predicted    1    All

True			
	0	1	All
0	212	212	
1	357	357	
All	569	569	

```
from sklearn.metrics import accuracy_score
print("정확도 %.2f" %accuracy_score(y,predictions))
```

정확도 0.63

Accuracy score : 0.63

### 1. Feature scaling - Standardization

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
a = sc.fit_transform(x)
print(a)
```

```
[[ 1.09706398 -2.07333501  1.26993369 ...  2.29607613  2.75062224
  1.93701461]
 [ 1.82982061 -0.35363241  1.68595471 ...  1.0870843  -0.24388967
  0.28118999]
 [ 1.57988811  0.45618695  1.56650313 ...  1.95500035  1.152255
  0.20139121]
```

- Standardization 후 Test 결과

Predicted    0    1    All

True			
	0	1	All
0	61	2	63
1	2	106	108
All	63	108	171

```
from sklearn.metrics import accuracy_score
print("정확도 %.2f" %accuracy_score(y,predictions))
```

정확도 0.99

(Confusion Matrix와 Accuracy score 값)

Standardization 후 데이터의 accuracy score 가 0.63 -> 0.99 까지 올랐다.

## 2. Feature scaling - MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import StandardScaler
ms = MinMaxScaler()

x = ms.fit_transform(x) # x_train data scaling
print(x)
```

- Min Max Scaler 후 Test 결과

Predicted	0	1	All
True			
0	206	6	212
1	5	352	357
All	211	358	569

```
from sklearn.metrics import accuracy_score
print("정확도 %.2f" % accuracy_score(y, predictions))
```

정확도 0.98

(Confusion Matrix와 Accuracy score 값)

confusion matrix와 accuracy score을 비교한 결과 많은 차이는 없었지만 조금더 높은 정확도  
여준 Standardization을 선택하여 모델을 구축하였다.

## ※ 데이터 분할(Training data, Test Data)

- train\_test\_split

1. Train Data : 60% / Test Data 40% 었을 때의 결과

```
from sklearn.model_selection import train_test_split

# Cross validation
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4, train_size = 0.6, random_state = 0)
#파라미터(data array, testsize(0.3 = 30%), train_size(0.7 = 70%), random_state(정수이면
# random하게 생성할때 사용되는 seed 숫자로 사용되며, None을 입력하면 np.random에서 제공하는 random number generator를 사용 )

Iteration 55, loss = 0.07878629
Iteration 56, loss = 0.07822851
Iteration 57, loss = 0.07761917
Iteration 58, loss = 0.07728140
```

Test Data 40% 었을 때 ANN learning 결과 값(0.07728140)

Predicted	0	1	All
True			
0	79	4	83
1	2	143	145
All	81	147	228

Test Data 40% 였을 때 ANN test data에 대한 confusion matrix 값

Accuracy 96%

## 2. Train Data: 70% /test Data 30% 였을 때의 결과

```
from sklearn.model_selection import train_test_split

# Cross validation
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3 , train_size = 0.7, random_state = 0)
#파라미터(data array, testsize(0.3 = 30%),train_size(0.7 = 70%), random_stat(정수이면
# random하게 생성할때 사용되는 seed 숫자로 사용되며, None을 입력하면 np.random에서 제공하는 random number generator를 사용 )

Iteration 46, loss = 0.06566467
Iteration 47, loss = 0.06519567
Iteration 48, loss = 0.06479109
Iteration 49, loss = 0.06395951
Iteration 50, loss = 0.06356112
Iteration 51, loss = 0.06298191
```

- Test Data 40% 였을 때 ANN learning 결과 값(0.06298191)

- Test Data 30% 였을 때와 비교하면 약 0.01 차이가 난다.

Predicted	0	1	All
True			
0	61	2	63
1	2	106	108
All	63	108	171

- Test Data 30% 였을 때 ANN test data에 대한 confusion matrix 값

- Accuracy = 097%

## ※ 모델 설정

### ※ ANN

```
# ANN(Training)
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(50,50),# 은닉층 수 ex) (150,100) 유닛 150,100개짜리 hidden layer 2개
                    activation = 'relu', # activation function Relu로 설정
                    solver = 'sgd', # algorithm (lbfgs', 'sgd', 'adam)
                    tol = 1e-7, # 우리가 러닝을 했는데 더이상 낮아지지 않으면 공부 그만해 라는 뜻
                    learning_rate_init = .7, # 얼마나 움직일 것인가 learning latio
                    verbose = True) # 보고 할 것인가 안할 것인가. False로 바꾸면 Lossfunction과 안나옴

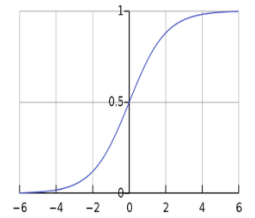
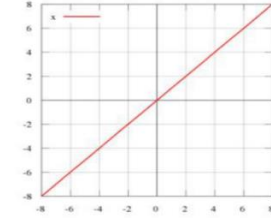
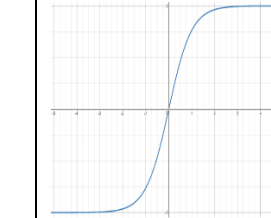
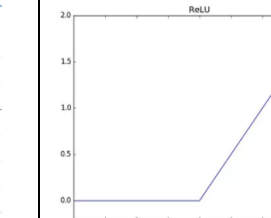
mlp.fit(x_train,y_train) # model 학습
```

## \* Hidden Layer 선정

Hidden Layer 2개 각 node(10,10)	Hidden Layer 2개 각 node(50,50)	Hidden Layer 2개 각 node(100,100)
Iteration 195, loss = 0.00124618	Iteration 195, loss = 0.00008009	Iteration 195, loss = 0.00008989
Iteration 196, loss = 0.00122398	Iteration 196, loss = 0.00008005	Iteration 196, loss = 0.00008976
Iteration 197, loss = 0.00120838	Iteration 197, loss = 0.00007983	Iteration 197, loss = 0.00008960
Iteration 198, loss = 0.00119826	Iteration 198, loss = 0.00007964	Iteration 198, loss = 0.00008938
Iteration 199, loss = 0.00118357	Iteration 199, loss = 0.00007947	Iteration 199, loss = 0.00008921
Iteration 200, loss = 0.00116829	Iteration 200, loss = 0.00007934	Iteration 200, loss = 0.00008909

Loss가 가장 적은 Hidden Layer 2개에 각 node(50,50)으로 설정

## \* Activation function 선정

			
Sigmoid function	Identity	Tanh	Relu function

## - Sigmoid

Sigmoid 시작	Sigmoid 마지막 loss값
Iteration 1, loss = 0.67358128 Iteration 2, loss = 0.66689756 Iteration 3, loss = 0.65971417 Iteration 4, loss = 0.65347850 Iteration 5, loss = 0.63323185 Iteration 6, loss = 0.60793620 Iteration 7, loss = 0.58093997 Iteration 8, loss = 0.52885594 Iteration 9, loss = 0.46403986 Iteration 10, loss = 0.38211128	Iteration 118, loss = 0.05131318 Iteration 119, loss = 0.05085291 Iteration 120, loss = 0.05117661 Iteration 121, loss = 0.05094352 Iteration 122, loss = 0.05107155 Iteration 123, loss = 0.05076282 Iteration 124, loss = 0.05097012 Iteration 125, loss = 0.05088392 Iteration 126, loss = 0.05124183 Iteration 127, loss = 0.05470168

에러가 낮아지는 속도가 느리다.

Sigmoid 함수 최종 loss 값 = 0.05470168

## - Identity

Identity 시작	Identity 결과 값
Iteration 1, loss = 0.50445764 Iteration 2, loss = 0.08873442 Iteration 3, loss = 0.08850713 Iteration 4, loss = 0.08314871 Iteration 5, loss = 0.08029820 Iteration 6, loss = 0.08673859 Iteration 7, loss = 0.07162173 Iteration 8, loss = 0.07184191 Iteration 9, loss = 0.06214207 Iteration 10, loss = 0.05979554	Iteration 91, loss = 0.04725202 Iteration 92, loss = 0.04913953 Iteration 93, loss = 0.04372661 Iteration 94, loss = 0.04289204 Iteration 95, loss = 0.04286911 Iteration 96, loss = 0.04298035 Iteration 97, loss = 0.04170732 Iteration 98, loss = 0.04225283 Iteration 99, loss = 0.04247825

Iteration 1, loss = 0.67358128  
Iteration 2, loss = 0.66689756

Iteration 1, loss = 0.50445764  
Iteration 2, loss = 0.08873442

(Sigmoid)

(Identity)

Iteration 1의 loss 값과 Iteration2의 값을 보면 sigmoid function보다 loss값이 확 줄은 것을 확인할 수 있다. 그러나 최종 loss값의 결과는 약 0.01정도 차이가 났다.

Identity loss 값 = 0.04247825

#### - Tahn

Than 시작	Than 결과 값
Iteration 1, loss = 0.38959288 Iteration 2, loss = 0.09150667 Iteration 3, loss = 0.07351924 Iteration 4, loss = 0.06884792 Iteration 5, loss = 0.06643081 Iteration 6, loss = 0.06235078 Iteration 7, loss = 0.06049419 Iteration 8, loss = 0.05852376 Iteration 9, loss = 0.05736686 Iteration 10, loss = 0.05407331	Iteration 193, loss = 0.00091246 Iteration 194, loss = 0.00090877 Iteration 195, loss = 0.00089431 Iteration 196, loss = 0.00088443 Iteration 197, loss = 0.00087816 Iteration 198, loss = 0.00087053 Iteration 199, loss = 0.00087259 Iteration 200, loss = 0.00085828

Than값을 보면 loss 값이 Sigmoid, Identity의 loss 값 보다 확 줄은 것을 확인할 수 있다. Than loss 값 = 0.000858528

#### - Relu

Relu 시작 loss 값	Relu 결과 loss 값
Iteration 1, loss = 0.54516604 Iteration 2, loss = 0.19281710 Iteration 3, loss = 0.12290005 Iteration 4, loss = 0.08982186 Iteration 5, loss = 0.07098029 Iteration 6, loss = 0.06332192 Iteration 7, loss = 0.06189580 Iteration 8, loss = 0.05699473 Iteration 9, loss = 0.04984898 Iteration 10, loss = 0.04445891	Iteration 193, loss = 0.00050416 Iteration 194, loss = 0.00049968 Iteration 195, loss = 0.00049803 Iteration 196, loss = 0.00049476 Iteration 197, loss = 0.00049248 Iteration 198, loss = 0.00048906 Iteration 199, loss = 0.00048684 Iteration 200, loss = 0.00048264

Sigmoid 함수를 제외한 3개의 함수는 학습이 빠른 것을 확인할 수 있다.

**Relu loss 값 : 0.00048264**

**4개의 activation function에 대한 최종 loss 값(hidden layer 2개 (150,100)기준)**

Sigmoid	Identity	Than	Relu
0.05470168	0.04247825	0.000858528	0.00048264

\* Relu<Than<Identity<Sigmoid

Loss 값이 가장 적게 나온 Relu함수를 activation function으로 지정

## ※ Decison Tree

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(max_depth = 3 ,criterion = 'entropy') # 최대 깊이 3이다.
dtc1 = DecisionTreeClassifier(max_depth = 5 ,criterion = 'entropy') # 최대 깊이 5이다.
dtc2 = DecisionTreeClassifier(max_depth = 7 ,criterion = 'entropy') # 최대 깊이 7이다.
dtc3 = DecisionTreeClassifier(max_depth = 9 ,criterion = 'entropy') # 최대 깊이 9이다.
```

Decison Tree 최대 높이에 차이를 주어 학습을 해보았다.

```
dtc learning score 0.980
dtc test score 0.965
dtc1 learning score 0.992
dtc1 test score 0.965
dtc2 learning score 1.000
dtc2 test score 0.959
dtc3 learning score 1.000
dtc3 test score 0.959
```

높이 3/5/7/9에 해당하는 learning Data 결과 값, Test Data 결과 값이다.

최고 높이	3	5	7	9
Learning Data 결과	0.980	0.992	1.000	1.000
Test Data 결과	0.965	0.965	0.959	0.959

(위에 사진에 대한 결과 값)

학습데이터에서는 높으면 높아 질수록 좋은 결과 가 나왔다. 하지만 Test 결과 값으로 예측해본결과 0.959~0.965 까지 Learning Data 와 비교하면 좋지 않은 결과가 나왔다.

### - 4개의 Decision tree에 대한 Accuracy값

```
from sklearn.metrics import accuracy_score
print("dct 정확도 %.2f" %accuracy_score(y_test,dtc_pred))
print("dct 정확도 %.2f" %accuracy_score(y_test,dtc_pred1))
print("dct 정확도 %.2f" %accuracy_score(y_test,dtc_pred2))
print("dct 정확도 %.2f" %accuracy_score(y_test,dtc_pred3))
```

```
dct 정확도 0.96
dct 정확도 0.96
dct 정확도 0.96
dct 정확도 0.96
```



## ※ Random Forest

(\* 나무 = Decision Tree)

```
clf = RandomForestClassifier(n_estimators =1) # 나무의 개수 = 1
clf5 = RandomForestClassifier(n_estimators =5) # 나무의 개수 = 5
clf10 = RandomForestClassifier(n_estimators =10) # 나무의 개수 = 10
clf20 = RandomForestClassifier(n_estimators =20) # 나무의 개수 = 10
clf50 = RandomForestClassifier(n_estimators =50) # 나무의 개수 = 10
clf100 = RandomForestClassifier(n_estimators =100) # 나무의 개수 = 10
```

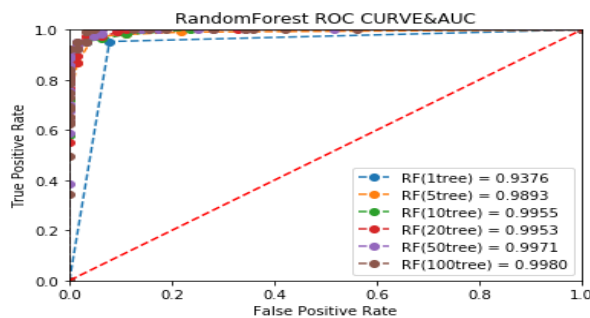
### - Random Forest 결과에 대한 Accuracy 결과

Case 1	Case 2	Case3
tree개수 1일 때 정확도 0.94	tree개수 1일 때 정확도 0.92	tree개수 1일 때 정확도 0.88
tree개수 5일 때 정확도 0.94	tree개수 5일 때 정확도 0.94	tree개수 5일 때 정확도 0.94
tree개수 10일 때 정확도 0.95	tree개수 10일 때 정확도 0.95	tree개수 10일 때 정확도 0.95
tree개수 20일 때 정확도 0.97	tree개수 20일 때 정확도 0.96	tree개수 20일 때 정확도 0.95
tree개수 50일 때 정확도 0.96	tree개수 50일 때 정확도 0.96	tree개수 50일 때 정확도 0.97
tree개수 100일 때 정확도 0.96	tree개수 100일 때 정확도 0.95	tree개수 100일 때 정확도 0.96

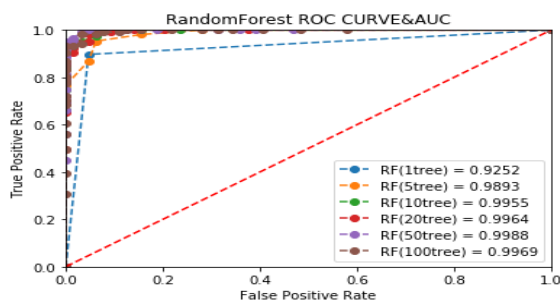
1. Tree의 개수가 1개인것보다는 여러 개 있는 것이 좋지만 tree가 많다고 해서 좋은 것은 아니다.
2. 결과가 계속해서 많은 차이가 난다.

### - Random Forest 결과에 대한 ROC CURVE & AUC 결과

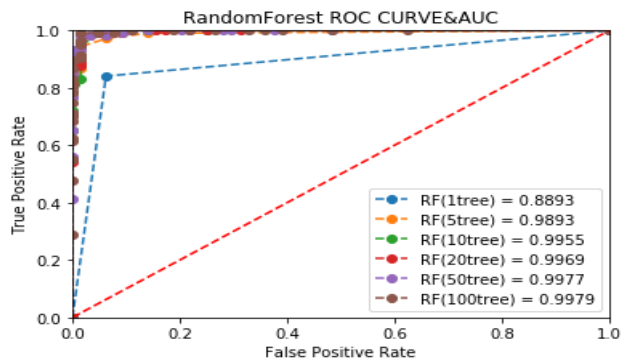
Case1 (ROC CURVE & AUC)



Case2 (ROC CURVE & AUC)



### Case3(ROC CURVE & AUC)



Random Forest는 tree의 개수의 관계없이 계속해서 accuracy값과 ROC CURVE & AUC 값이 바뀌는 것을 확인하였다.

## - Cross Validation (5-fold)

ANN 교차 검증 점수: [0.9375      0.9375      0.95      0.96202532 0.98734177]  
 ANN 교차 검증 평균 점수: 0.95  
 Random Forest 교차 검증 점수: [0.95      0.9625      0.925      0.91139241 1.      ]  
 Random Forest 교차 검증 평균 점수: 0.95  
 Decision Tree 교차 검증 점수: [0.925      0.9      0.9      0.92405063 0.96202532]  
 Decision Tree 교차 검증 평균 점수: 0.92

Cross Validation을 통해 ANN & Random Forest & Decision Tree에 대한 결과 값에 대한 평균 점수

## 3. 최종 모델

### - accuracy Confusion Matrix

ANN					Random Forest					Decision Tree				
Predicted					Predicted					Predicted				
True					True					True				
0	1	All			0	1	All			0	1	All		
63	1	64			63	1	64			62	2	64		
1	106	107			2	105	107			13	94	107		
All	64	107	171		All	65	106	171		All	75	96	171	

## - Random Forest & Decision Tree & ANN 의 ACCURACY 값

```
from sklearn.metrics import accuracy_score
print("ANN accuracy %.2f" %accuracy_score(y_test,predictions))
print("Random Forest accuracy %.2f" %accuracy_score(y_test,random_forest_pred))
print("Decision Tree accuracy %.2f" %accuracy_score(y_test,Dct_pred))
```

ANN accuracy 0.99  
Random Forest accuracy 0.98  
Decision Tree accuracy 0.91

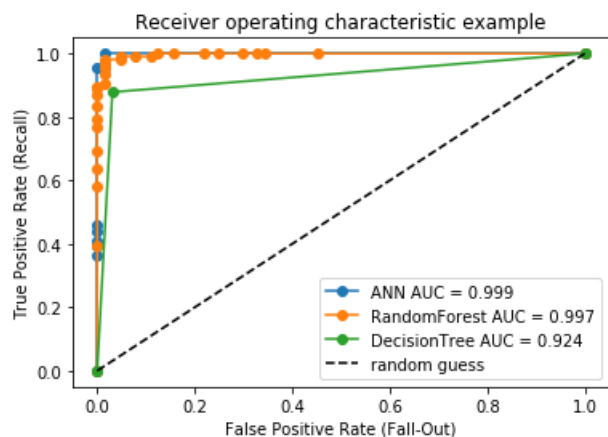
## - Random Forest & Decision Tree & ANN 의 F-1 Score

```
from sklearn.metrics import f1_score

print("ANN accuracy %.2f" %f1_score(y_test,predictions)) # ANN Accuracy 과 출력
print("Random Forest accuracy %.2f" %f1_score(y_test,random_forest_pred))# Random Forest 과 출력
print("Decision Tree accuracy %.2f" %f1_score(y_test,Dct_pred))# Decision Tree 과 출력
```

ANN accuracy 0.99  
Random Forest accuracy 0.99  
Decision Tree accuracy 0.93

## - Random Forest & Decision Tree & ANN 의 ROC CURVE

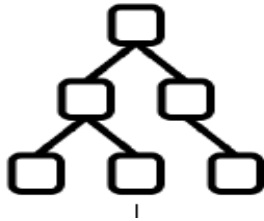


	ANN	Random Forest	Decision Tree
Accuracy	0.99	0.98	0.91
AUC	0.999	0.997	0.924
F-1	0.99	0.99	0.93

위의 표에 Accuracy 와 ROC&AUC가 가장 높은 ANN을 최종 모델로 선택하였다.

## 4. 결론

### - Decision Tree

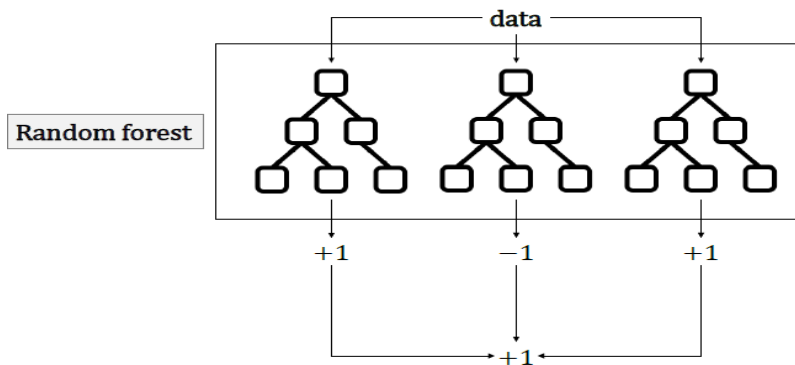


Decision Tree의 깊이를 깊게 하여도 결과에는 차이가 많이 나지 않았다.

가장 먼저 Test Data 를 30% 한뒤 마지막에 K-fold로 하여 Test 해보았지만 Accuracy값은 거의 동일하였다.

3분류기 중에 가장 Accuracy,Roc&AUC 값이 가장 낮았음으로 좋은 분류기가 아니었다.

### - Random Forest



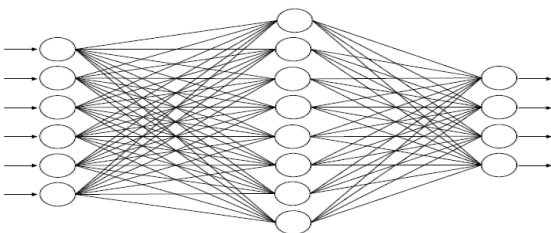
Decision Tree의 단점을 보완하기 위해 Decision Tree를 여러 개 하여 나온 결과 값 중에 가장 많이 나온 결과 값을 최종 결과로 한 Random Forest를 사용해 보았다.

Tree의 개수를 많게 하면 Decision tree보다 좋은 accuracy를 얻을 수 있었다.

그러나 Tree가 개수가 많으면 좋았지만 Tree의 개수가 많다고 해서 꼭 좋은 결과는 아니었다.

(결과값이 계속해서 바뀌었다)

### - ANN(Artificial Neural Network)



Hidden Layer(및 Hidden node) & Activation function등 여러가지 기준에 따라 학습속도, 학습결과가 다르다는 것을 알 수 있었다.

#### - Random Forest & Decision Tree & ANN 의 ROC CURVE

	ANN	Random Forest	Decision Tree
Accuracy	0.99	0.98	0.91
AUC	0.999	0.997	0.924
F-1	0.99	0.99	0.93

위의 표에서 볼 수 있듯이 ANN 모델이 모든 평가 지표에서 가장높은 점수를 얻었다.

#### - 최종결론

각 머신러닝을 Python library를 통해 구현 할 수 있었고 구현 완성 후 Accuracy, ROC&AUC를 통해 어떤 모델이 가장 좋은지 확인 할 수 있었다. ANN & Decision Tree, Random Forest 대부분의 머신러닝이 있는 데이터 값으로 학습 시킨 후 Test결과 값의 정확률이 94%~98%(=0.94)로 좋은 결과 값을 주었다. 그러나 정확도가 94%이상 이라 하여 좋은 분류기구나! 해서 사용하여도 유방암 환자를 정상이라고 1명이라도 판단하게 된다면 정상이라고 판단된 유방암 환자는 생명에 위협을 느낄 것이다. 그렇기 데이터가 정말 많고 인간이 하기에는 시간이 정말 오래 걸리는 문제에 대해서 머신러닝을 사용하는 것은 좋다고 생각하지만 암처럼 생명에 영향을 주는 문제들 에는 머신러닝을 사용하여 판단하는 것은 아직 좋지 않다고 생각하였다.