

머신 러닝 모델 기법을 활용한
신용카드 사기 거래 여부
예측 및 성능 평가

목차

01. 주제 선정 배경

02. 데이터 설명 및 사전 코딩 작업

03. 모델 성능 평가 지표

04. 모델링 및 결과를 통한 성능 평가

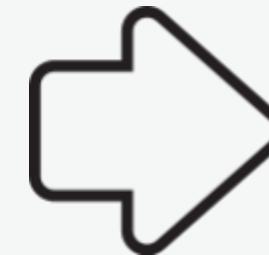
- Random Forest
- Artificial Neural network
- Autoencoder

주제 선정 배경

최 교수에 따르면 코로나19 사태가 시작하면서 개인 신분을 도용하는 사이버 범죄 발생 건수가 이전 시기에 비해 무려 300%나 증가했다. 정상적인 경제 활동이 셧다운되면서 온라인 사용이 급증한 까닭이다.

지난해 신용카드 사기 최대 피해자는 60대 이상으로 모두 6만8,013건의 신고가 접수됐고 피해액만도 8억3,500만달러에 달한다.

최근에는 경제활동이 활발한 30~39세 젊은층에서 신분도용에 의한 신용카드 범죄 신고가 늘고 있다는 게 최 교수의 설명이다.

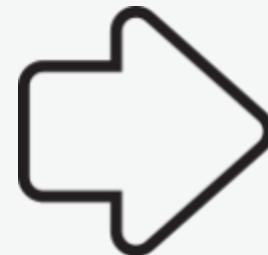


신분도용에 의한 신용카드 범죄,

- 코로나 전후 대비 **300%** 증가
- 최근 **30대에서도 크게 증가**

주제 선정 배경

‘다크웹(dark web)’에서 얻은 개인정보를 도용해 결제한 ‘카드 사기’ 규모가 2018년 280억달러(약 33조원)로 5년 전에 비해 2배 이상 늘어난 것으로 나타났다. 최근 각국에서 오픈뱅킹 도입과 전자
최근 각국에서 오픈뱅킹 도입과 전자상거래 확대, 핀테크 기업들의 지급결
제 분야 진출 등 지급수단이 다양해지면서 카드 사기 유형도 진화하고 있다



다크웹의 개인정보를 이용한 카드사기,
- 2013~2018, 5년간 2배 이상 증가

출처: <https://n.news.naver.com/article/032/0002992355?from=kakao>



늘어나는 신용카드 사기 문제

“신용카드 사기 거래
여부 예측 모형”

데이터 찾기

kaggle



신용카드 사기 거래 감지

2013년 9월 이틀 간 유럽 신용카드 사용자들의 실제 거래 기록

(284807, 31) 284,807건의 거래 내역, 31개의 변수



데이터 살펴보기

31개의 칼럼(열)

	A	B	C	D	E	F	Z	AA	AB	AC	AD	AE
1	Time	V1	V2	V3	V4	V25	V26	V27	V28	Amount	Class	
2	0	-1.35981	-0.07278	2.536347	1.378155	0.128539	-0.18911	0.133558	-0.02105	149.62	0	
3	0	1.191857	0.266151	0.16648	0.448154	0.16717	0.125895	-0.00898	0.014724	2.69	0	
4	1	-1.35835	-1.34016	1.773209	0.37978	-0.32764	-0.1391	-0.05535	-0.05975	378.66	0	
5	1	-0.96627	-0.18523	1.792993	-0.86329	0.647376	-0.22193	0.062723	0.061458	123.5	0	
6	2	-1.15823	0.877737	1.548718	0.403034	-0.20601	0.502292	0.219422	0.215153	69.99	0	
7	2	-0.42597	0.960523	1.141109	-0.16825	-0.23279	0.105915	0.253844	0.08108	3.67	0	
8	4	1.229658	0.141004	0.045371	1.202613	0.750137	-0.25724	0.034507	0.005168	4.99	0	
9	7	-0.64427	1.417964	1.07438	-0.4922	-0.41527	-0.05163	-1.20692	-1.08534	40.8	0	
10	7	-0.89429	0.286157	-0.11319	-0.27153	0.373205	-0.38416	0.011747	0.142404	93.2	0	
11	9	-0.33826	1.119593	1.044367	-0.22219	-0.06973	0.094199	0.246219	0.083076	3.68	0	
12	10	1.449044	-1.17634	0.91386	-1.37567	0.251367	-0.12948	0.04285	0.016253	7.8	0	
13	10	0.384978	0.616109	-0.8743	-0.09402	-0.76731	-0.49221	0.042472	-0.05434	9.99	0	

변수 설명



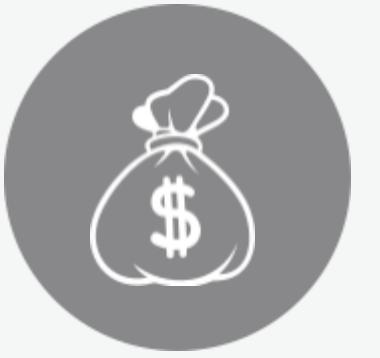
Time

시간



V1~V28

개인정보로



Amount

거래 금액



Class

사기여부

0 : 정상

1: 사기

공개되지

않은 값

결측값 확인

```
data.isnull().sum()
```

결측값 존재하지 않음

Time	0	V8	0	V16	0	V24	0
V1	0	V9	0	V17	0	V25	0
V2	0	V10	0	V18	0	V26	0
V3	0	V11	0	V19	0	V27	0
V4	0	V12	0	V20	0	V28	0
V5	0	V13	0	V21	0	Amount	0
V6	0	V14	0	V22	0	Class	0
V7	0	V15	0	V23	0	dtype:	int64

사기 거래 수 확인

```
tmp=data['Class'].value_counts().to_frame().reset_index()  
tmp['Percent(%)']=tmp["Class"].apply(lambda x : round(100*float(x) / len(data),2))  
tmp=tmp.rename(columns={"index":"Target", "Class":"Count"})
```

	Target	Count	Percent (%)
정상	0	284315	99.83
사기	1	492	0.17

매우 불균형
(High Unbalanced)

X, Y 데이터셋 분리

V1~V28로부터 사기여부(Class) 예측

X : V1~28

Y : Class

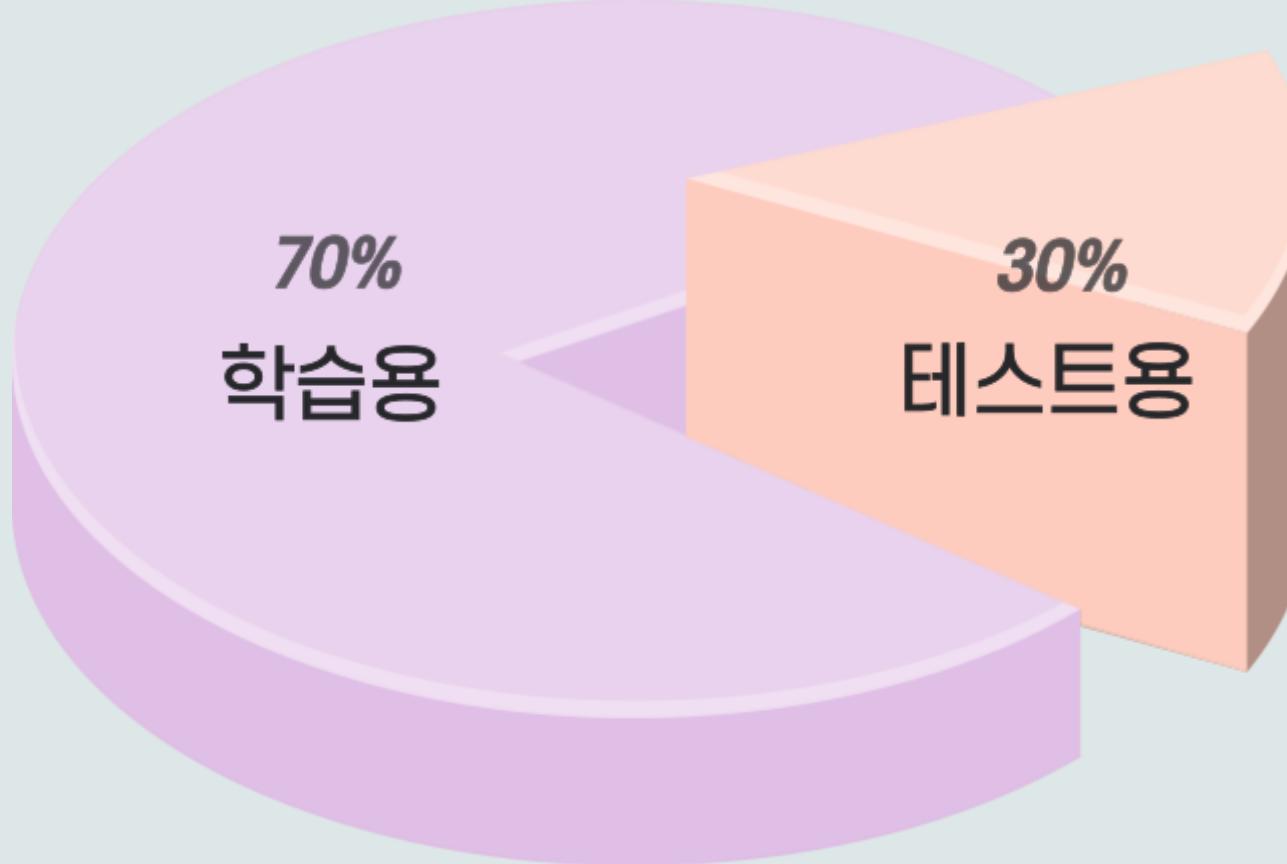
#Feature(x)과 Label(y) 분리

```
x_data=data.loc[:, 'V1':'V28'] #V1~V28
```

```
y_data=data.loc[:, 'Class'] #사기여부
```

학습용, 테스트용 데이터셋 분리

비율 7:3



랜덤으로 섞은 뒤 분리

```
shuffle_index=np.random.permutation(len(data))  
x_data=x_data.values[shuffle_index]  
y_data=y_data.values[shuffle_index]
```

```
n_train=int(len(x_data)*0.7)
```

```
x_train=x_data[:n_train]  
y_train=y_data[:n_train]  
x_test=x_data[n_train:]  
y_test=y_data[n_train:]
```

분류결과표(in FDS)

분류결과표

실제 클래스와 모형이 예측한 클래스가 일치하는지를 개수로 센 결과를 표로 나타낸 것

FDS(Fraud Detection System)

금융 거래에서 잘못된 거래나 사기 거래를 찾아내는 시스템

		사기라고 예측	정상이라고 예측
실제로 사기	실제로 사기	TP True Positive	FN False Negative
	실제로 정상	FP False Positive	TN True Negative

정확도(Accuracy)

=정분류율

전체 표본 중 맞게 예측한 표본수의 비율

$$\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP}$$

정밀도(Precision)

사기거래라고 예측한 거래 중 실제 사기 거래의 비율

$$\text{Precision} = \frac{TP}{TP + FP}$$

재현율(Recall)

=민감도(Sensitivity)

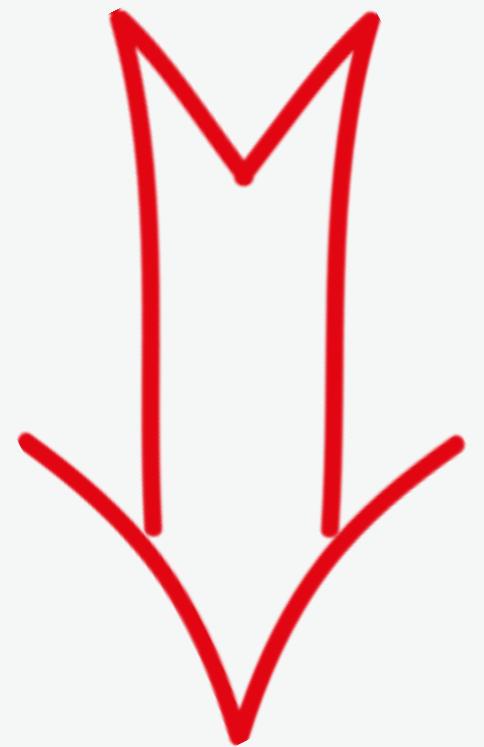
=TPR(True Positive Rate)

실제 사기 거래 중에서 사기라고 예측한 거래의 비율

$$\text{Recall} = \frac{TP}{TP + FN}$$

정밀도(Precision)

재현율(Recall)



F점수(F-score)

F점수(F-score)

정밀도와 재현율의 가중조화평균

0~1 사이의 값

1에 가까울수록 좋은 모형

베타 : 정밀도에 주어지는 가중치

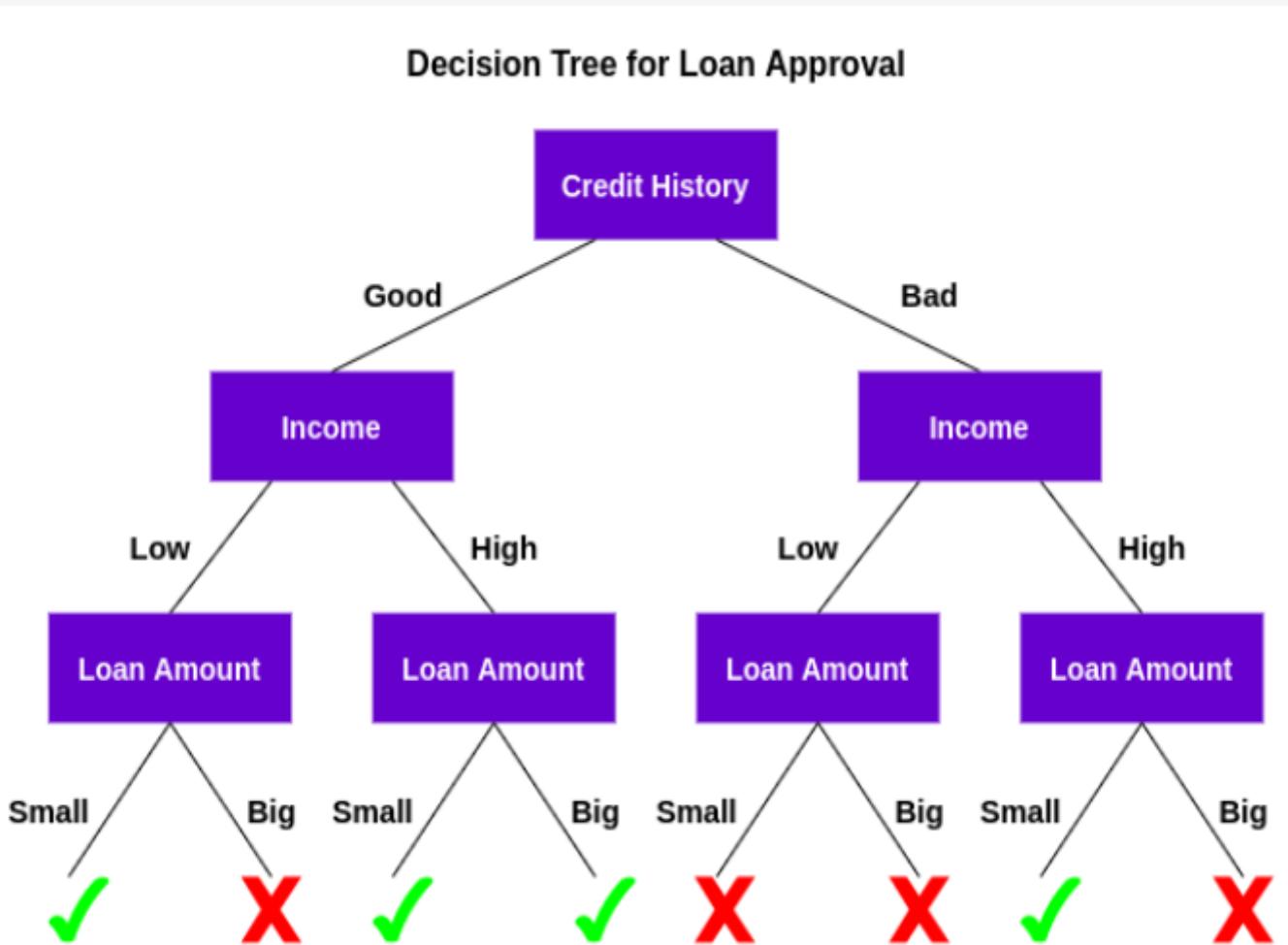
$$F_{\beta} = \frac{(1 + \beta^2) \text{Precision} \times \text{Recall}}{\beta^2 \text{Precision} + \text{Recall}}$$



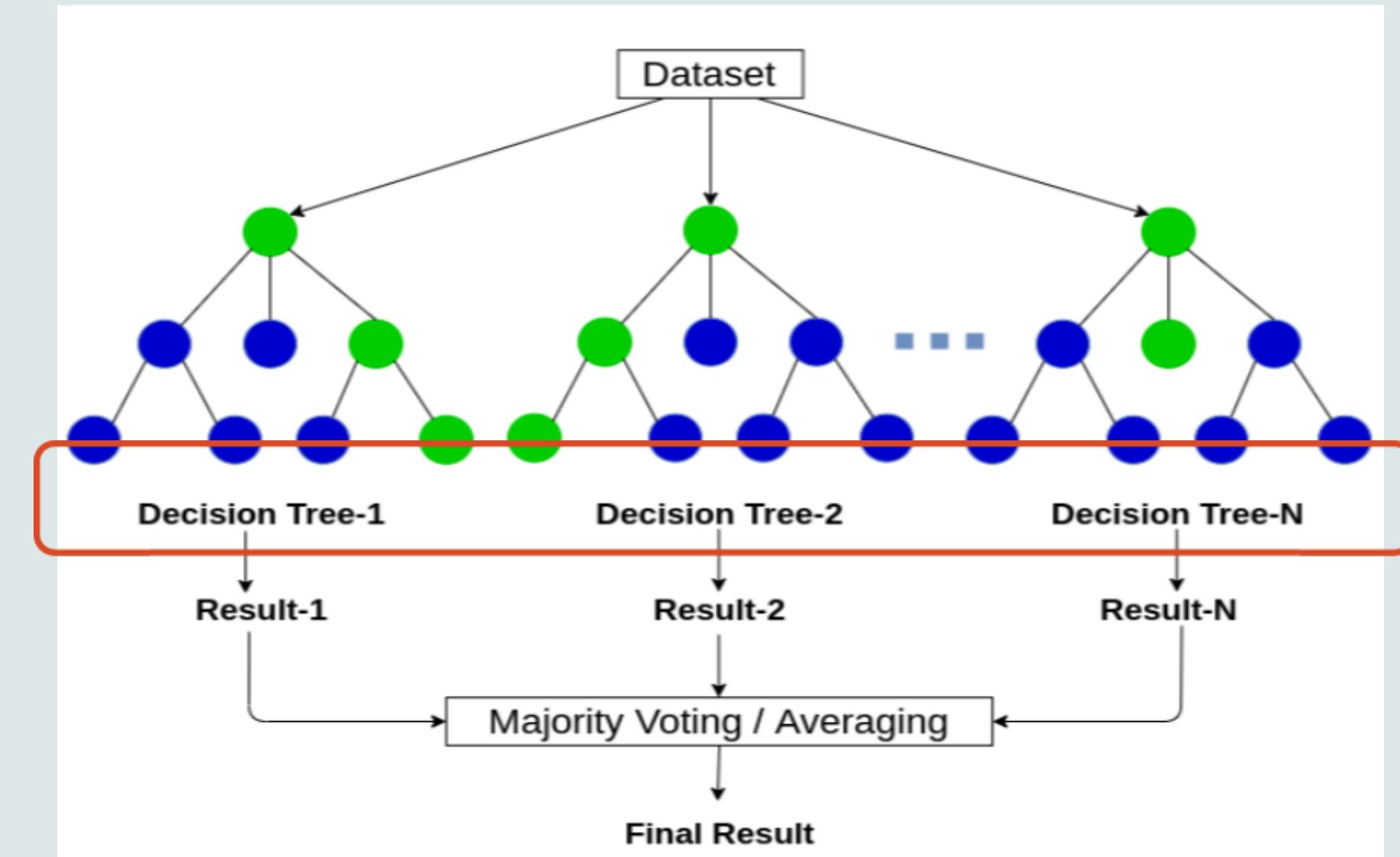
가중치를 1로 두어 계산

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Decision Tree



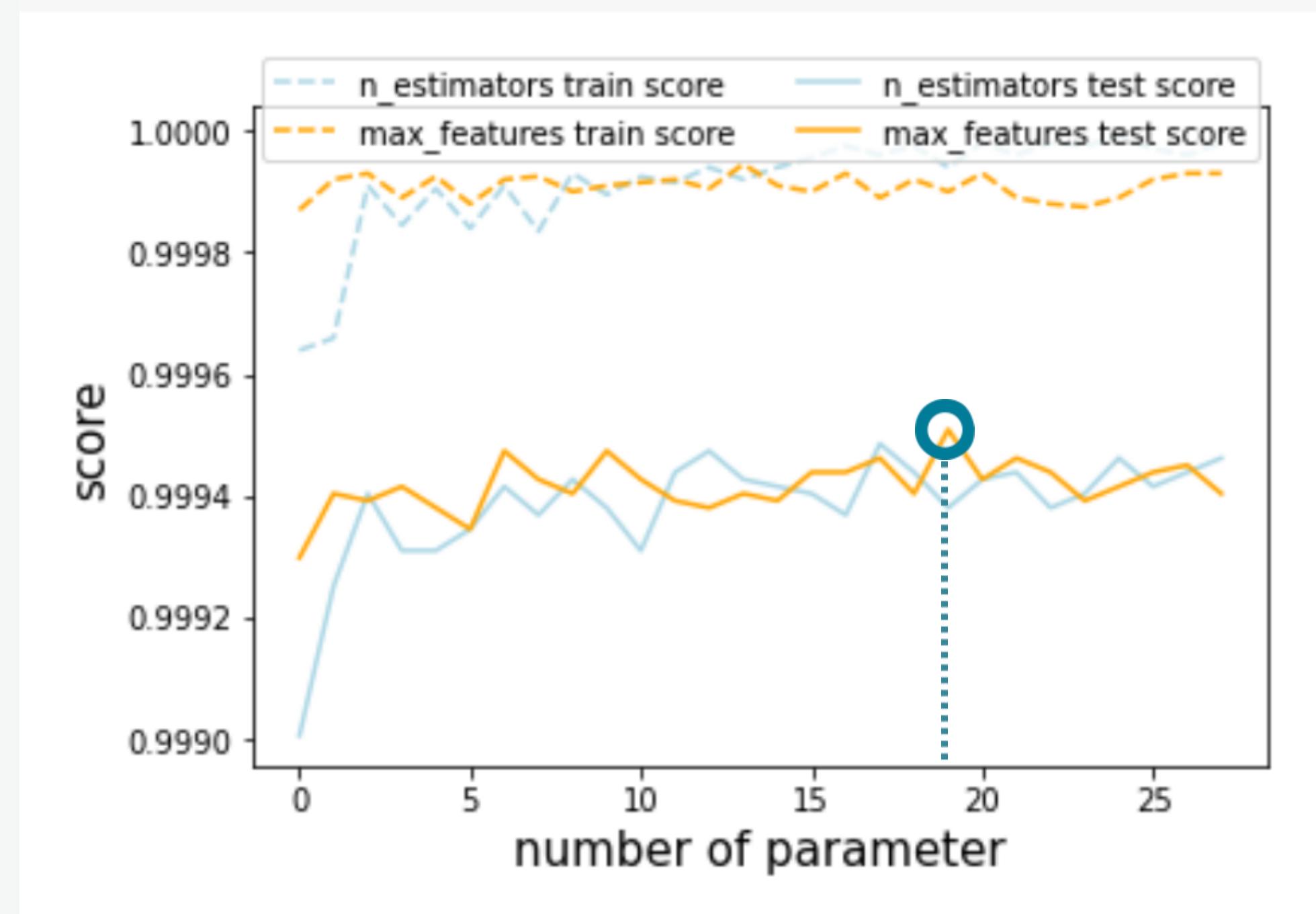
Random Forest



과적합 문제 (overfitting) = (high variance)

Modeling

```
from sklearn.ensemble import RandomForestClassifier  
model_rf=RandomForestClassifier(n_estimators=18,max_features=18, random_state=30)
```



Training

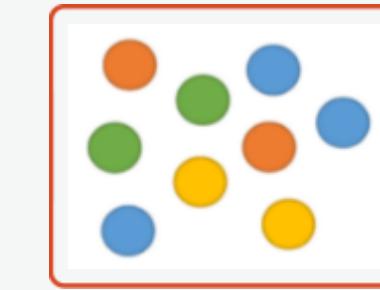
```
model_rf.fit(x_train,y_train)
```

Bagging - 랜덤하게 샘플 복원추출 반복



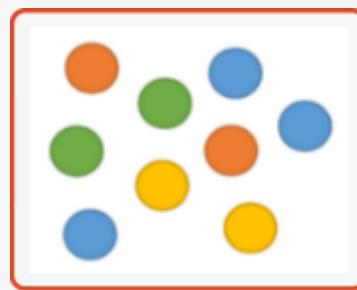
Training

Training Data

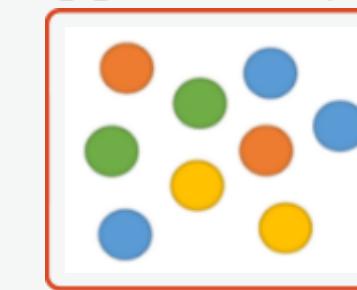


Sampling with Replacement
Bagging

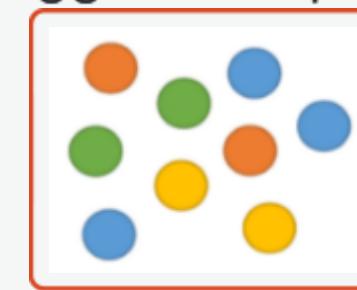
Bagged Sample 1



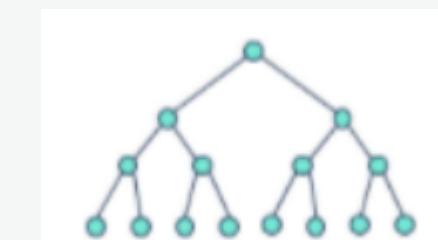
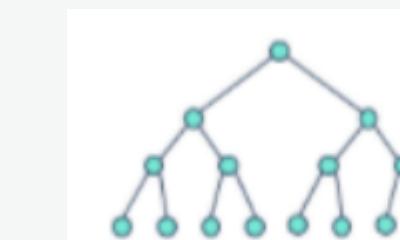
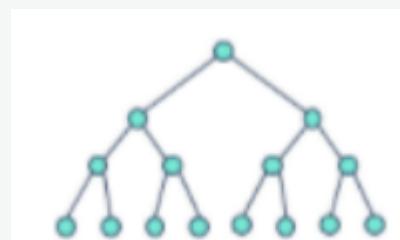
Bagged Sample 2



Bagged Sample N



Training of Decision Trees



예측 & 성능평가

```
y_pred=model_rf.predict(x_test)
```

```
y_real=y_test
```

```
from sklearn.metrics import classification_report
```

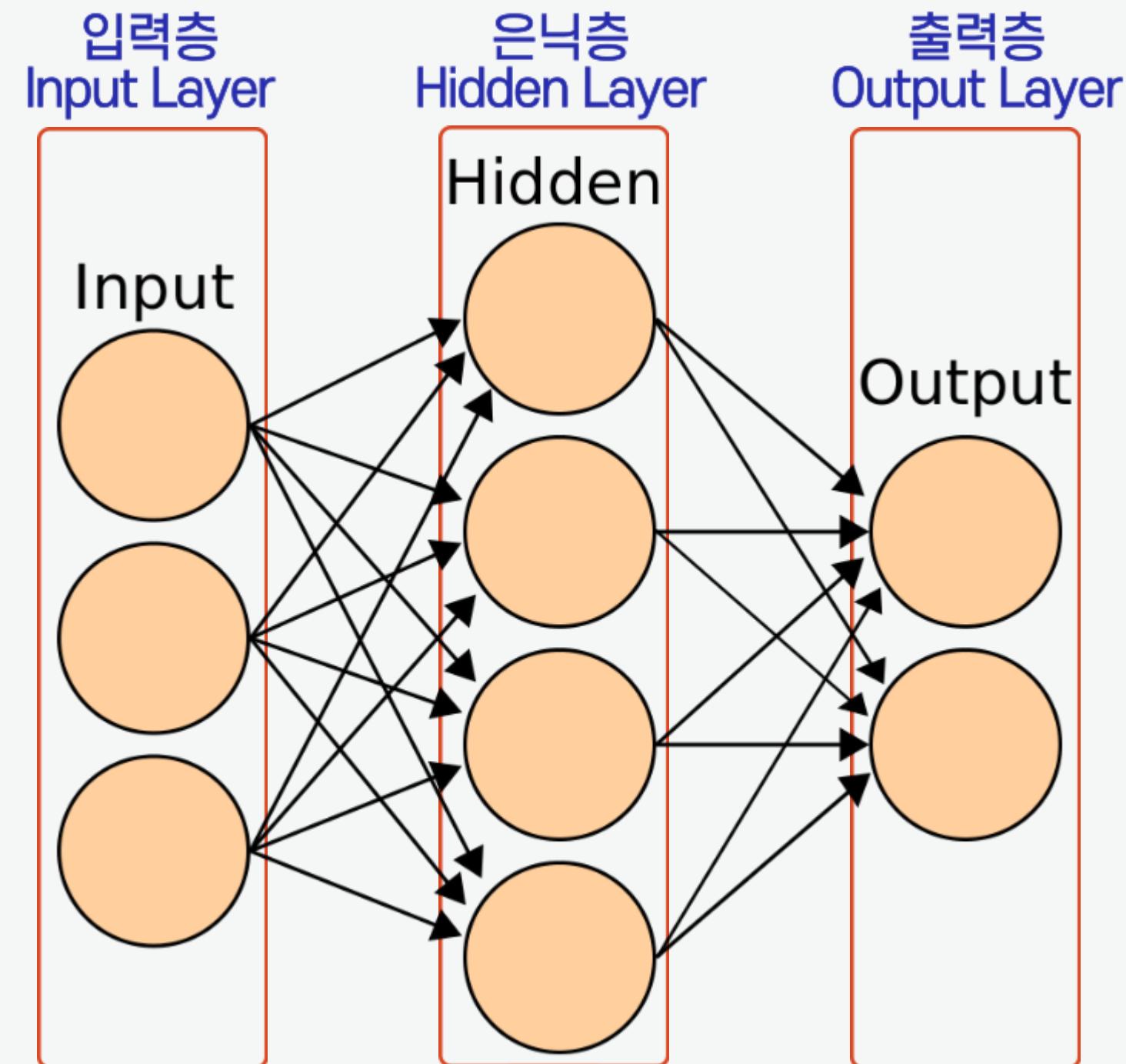
```
print(classification_report(y_real,y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85306
1	0.92	0.75	0.83	137
accuracy			1.00	85443
macro avg	0.96	0.88	0.91	85443
weighted avg	1.00	1.00	1.00	85443

	정확도	정밀도	재현율	F점수
	0.99	0.91	0.75	0.82

실제 사기 거래 건수 492건 중, 사기 거래라고 예측한 건수 372

인공 신경망 (Artificial Neural Network)



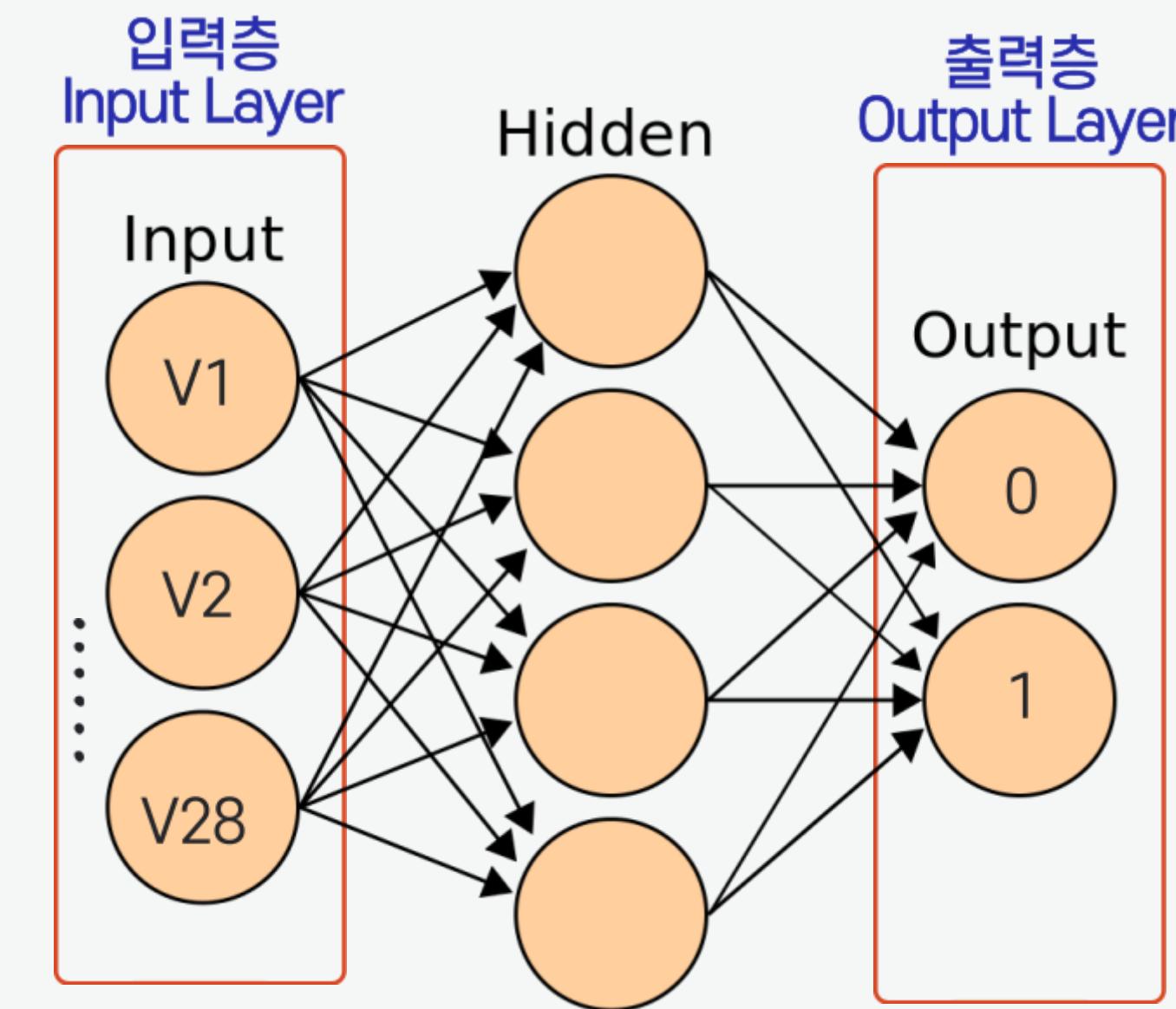
- * 각 동그라미는 하나의 뉴런 (Neuron)
- * 입력층 : 입력을 받기 때문에 Input Layer 라고 함.
- * 은닉층
 1. 활성 함수를 계산한 결과를 만듦
 2. 2개 이상의 층으로 구성 가능.
 3. 각 층의 뉴런 개수를 조절 하여 차원을 축소하거나 확장할 수 있음.
- * 출력층 : 은닉층의 Output을 종합하여 최종결과를 만들어 낸다.

Modeling

```
import tensorflow as tf  
import tensorflow.keras.layers as layers  
import tensorflow.keras.models as models
```

```
n_inputs=x_train.shape[1]
```

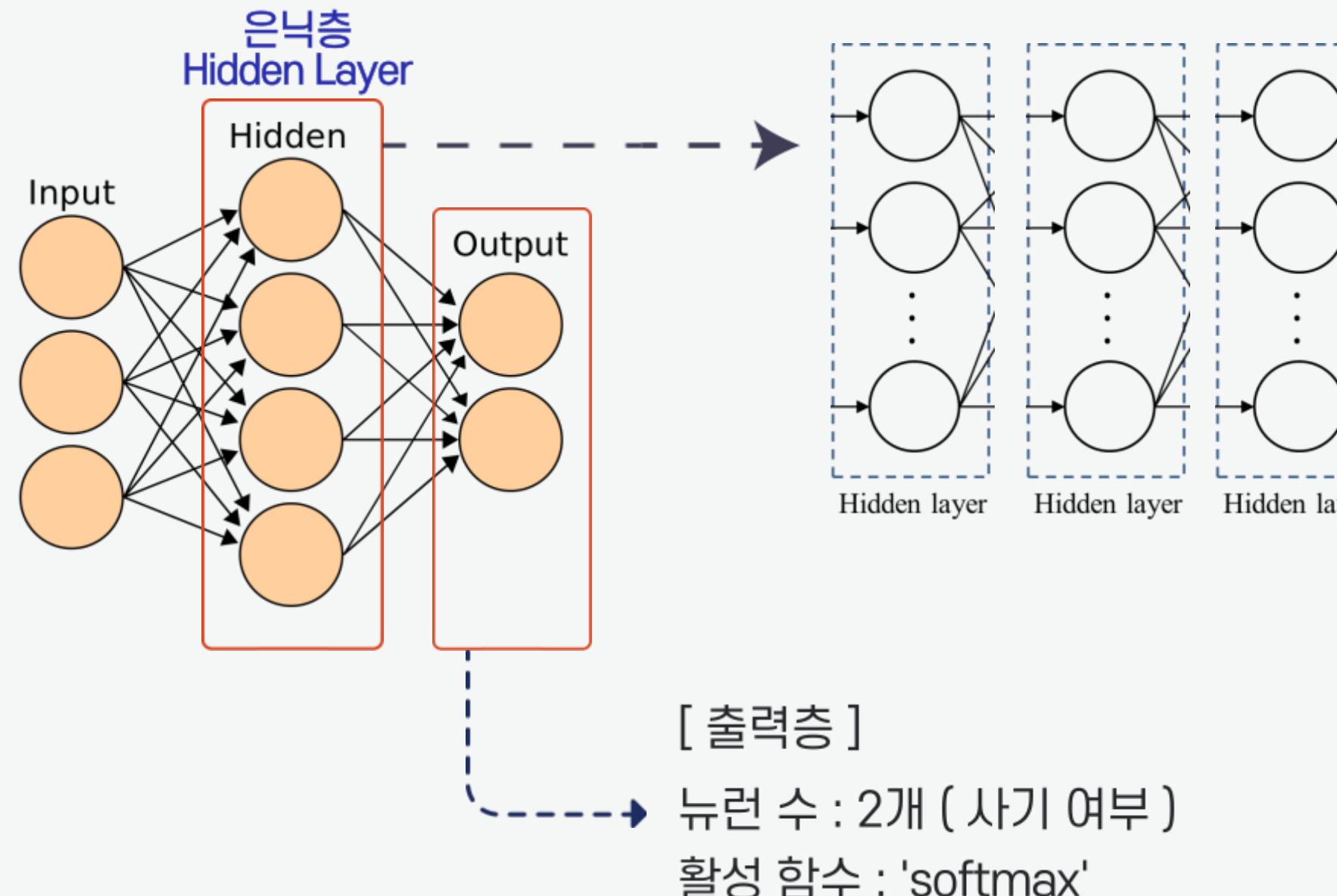
```
n_output=2
```



Modeling

```
model_nn=tf.keras.Sequential([  
    layers.Dense(64,input_shape=(n_inputs, ), activation='tanh'),  
    layers.Dense(32,activation='relu'),  
    layers.Dense(16,activation='relu'),  
    layers.Dense(n_output,activation='softmax'),  
])
```

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



[첫번째 은닉층]

뉴런 수 : 64개

활성 함수 : 'tanh'

[두번째 은닉층]

뉴런 수 : 32개

활성 함수 : 'relu'

[세번째 은닉층]

뉴런 수 : 16개

활성 함수 : 'relu'

Training

```
#train  
model_nn.fit(x_train,y_train,batch_size=100,epochs=10,validation_data=(x_test,y_test))
```

- 컴파일한 후 fit 함수를 이용해 모델 학습
- 에포크를 10으로 정의하여 **총 10회 학습**

```
Epoch 1/10  
1994/1994 [=====] - 7s 3ms/step - loss: 0.0470 - accuracy: 0.9815  
Epoch 2/10  
1994/1994 [=====] - 5s 2ms/step - loss: 0.0026 - accuracy: 0.9995  
Epoch 3/10  
1994/1994 [=====] - 5s 2ms/step - loss: 0.0026 - accuracy: 0.9994  
Epoch 4/10  
1994/1994 [=====] - 5s 3ms/step - loss: 0.0027 - accuracy: 0.9994  
Epoch 5/10  
1994/1994 [=====] - 5s 3ms/step - loss: 0.0023 - accuracy: 0.9994  
Epoch 6/10  
1994/1994 [=====] - 5s 2ms/step - loss: 0.0020 - accuracy: 0.9994  
Epoch 7/10  
1994/1994 [=====] - 5s 3ms/step - loss: 0.0017 - accuracy: 0.9995  
Epoch 8/10  
1994/1994 [=====] - 5s 2ms/step - loss: 0.0022 - accuracy: 0.9994  
Epoch 9/10  
1994/1994 [=====] - 5s 2ms/step - loss: 0.0016 - accuracy: 0.9996  
Epoch 10/10  
1994/1994 [=====] - 5s 3ms/step - loss: 0.0017 - accuracy: 0.9996
```

```
y_pred=model_nn.predict(x_test)
```

```
y_real=y_test
```

```
from sklearn.metrics import classification_report
```

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

```
accuracy=round(sum(y_pred==y_real) / len(y_pred),4)
```

```
print('Accuracy :',accuracy)
```

```
print(classification_report(y_real,y_pred))
```

```
Accuracy : 0.9994  
precision recall f1-score support
```

0	1.00	1.00	1.00	85298
1	0.82	0.80	0.81	145

accuracy			1.00	85443
macro avg	0.91	0.90	0.91	85443
weighted avg	1.00	1.00	1.00	85443

정확도	정밀도	재현율	F점수
0.99	0.82	0.80	0.81

실제 사기 거래 건수 492건 중, 사기 거래라고 예측한 건수 396

지도 학습 [Supervised Learning]

- 정답이 있는 데이터를 활용해 데이터를 학습
- 입력 값(X data)이 주어지면 입력값에 대한 Label(Y data)를 주어 학습



랜덤 포레스트

인공 신경망

비지도 학습 [Unsupervised Learning]

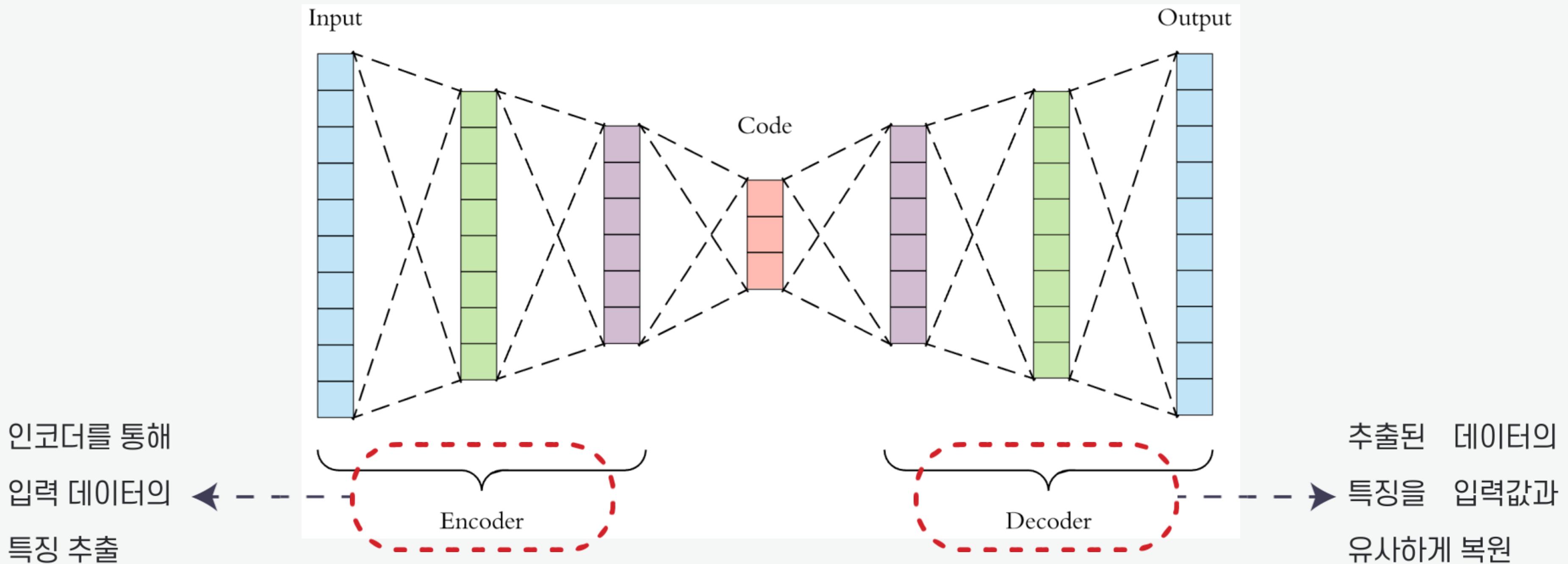
- 데이터에 대한 레이블(Label)
명시적인 정답이 주어지지 않은 상태에서 컴퓨터를 학습
- Input에 대하여 올바른 정답이 없는 데이터 집합이 주어지는 경우의 학습
- 데이터안에서 어떤 패턴을 찾아내는데 목적이 있음



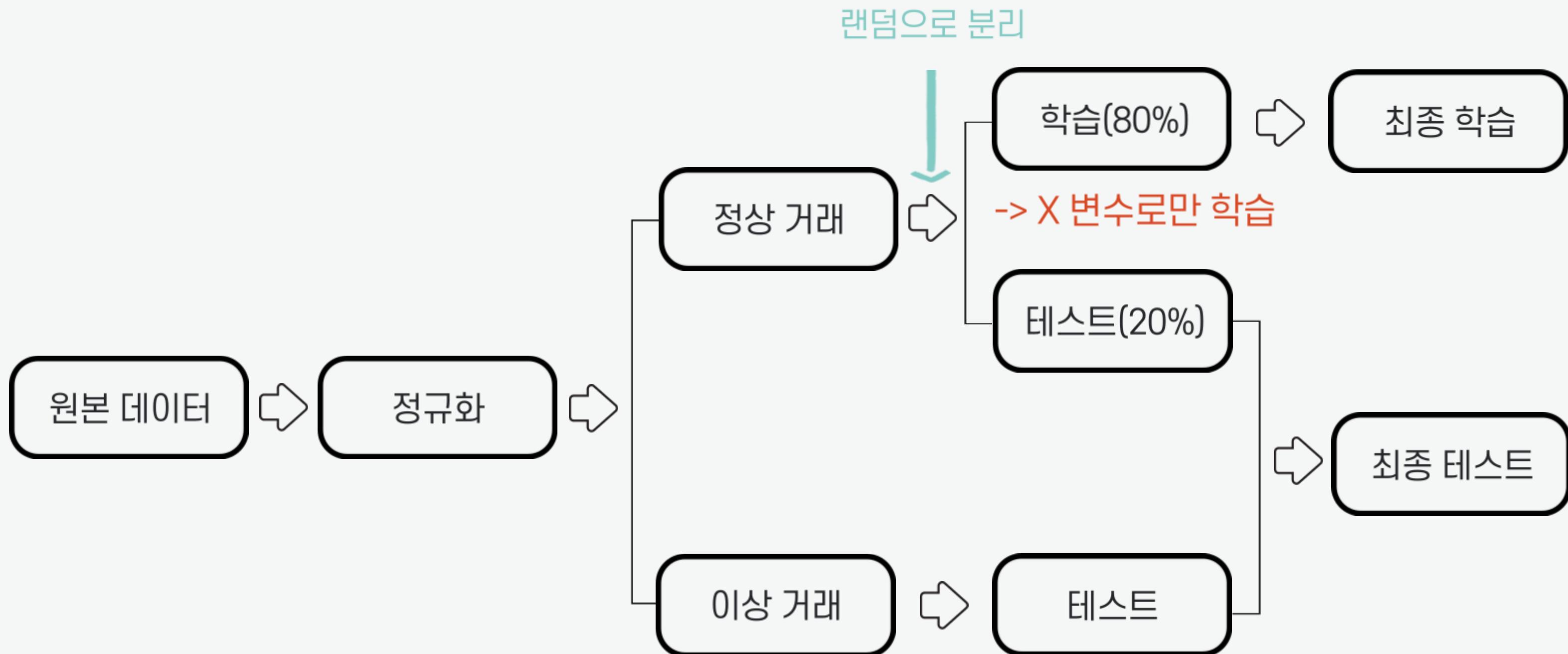
오토인코더

오토인코더 (Auto Encoder)

차원 축소, 이상치 감지에 효과적인 비지도 학습 모델



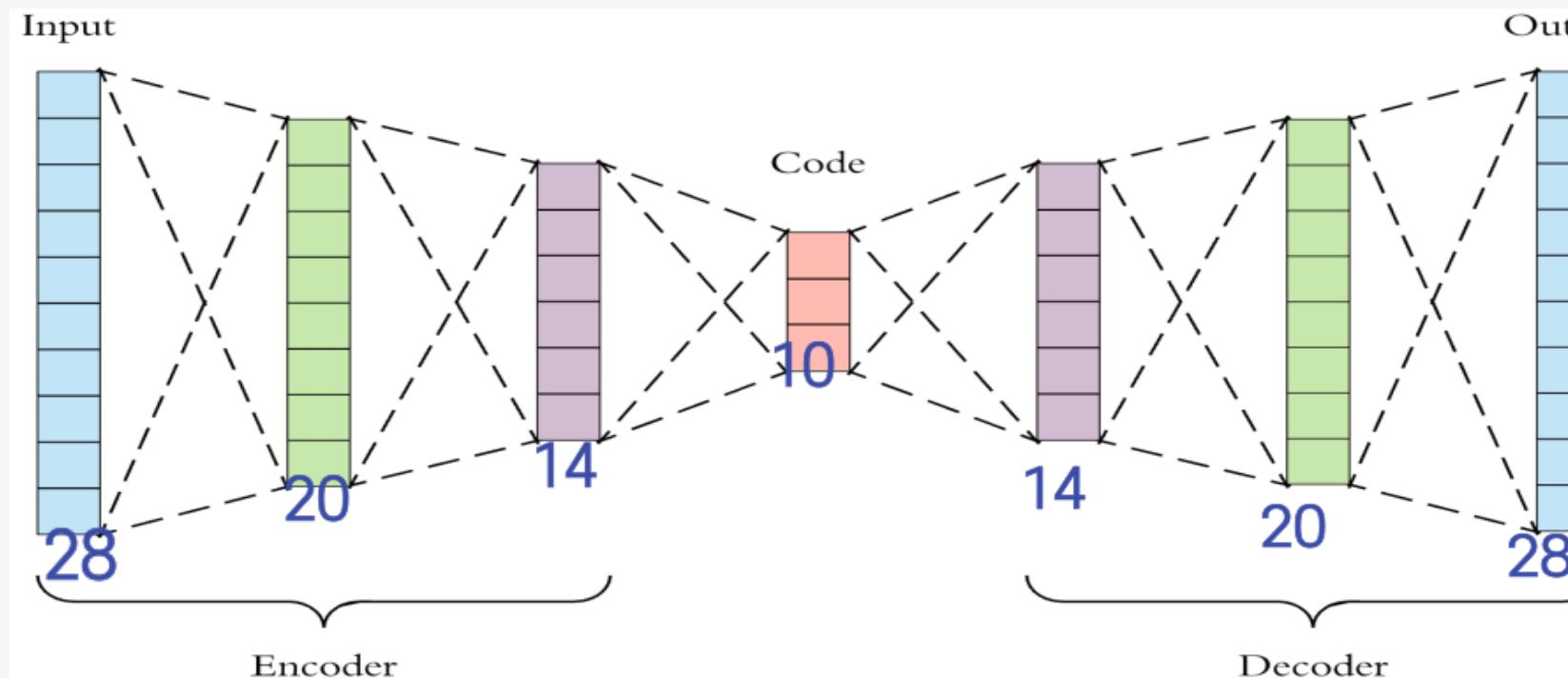
데이터 처리 및 분리



데이터 처리 및 분리

```
# 데이터 정규화  
df_norm = (df - df.min0) / (df.max0 - df.min0)  
  
# 정상/ 이상 거래 분류  
fraud=df_norm[df_norm.Class==1.0]  
normal=df_norm[df_norm.Class==0.0]  
  
# train/ test 데이터 분류  
train=normal.sample(frac=0.8)          ➤ train  
test=normal.loc[~normal.index.isin(train.index)] -> train 데이터 80% 제외  
testX=test.sample(frac=1)                } test  
fraudX=fraud
```

Modeling



input 설정

```
X=tf.placeholder(tf.float32,[None,28])
```

```
dense1=tf.layers.dense(inputs=X,units=20,activation=tf.nn.sigmoid)  
dense2=tf.layers.dense(inputs=dense1,units=14,activation=tf.nn.sigmoid)  
encoder = tf.layers.dense(inputs=dense2,units=10,activation=tf.nn.sigmoid)
```

→ encoder

```
dense4 = tf.layers.dense(inputs=encoder,units=14,activation=tf.nn.sigmoid)  
dense5 = tf.layers.dense(inputs=dense4,units=20,activation=tf.nn.sigmoid)  
decoder = tf.layers.dense(inputs=dense5,units=28,activation=tf.nn.sigmoid)
```

→ decoder

cost 설정

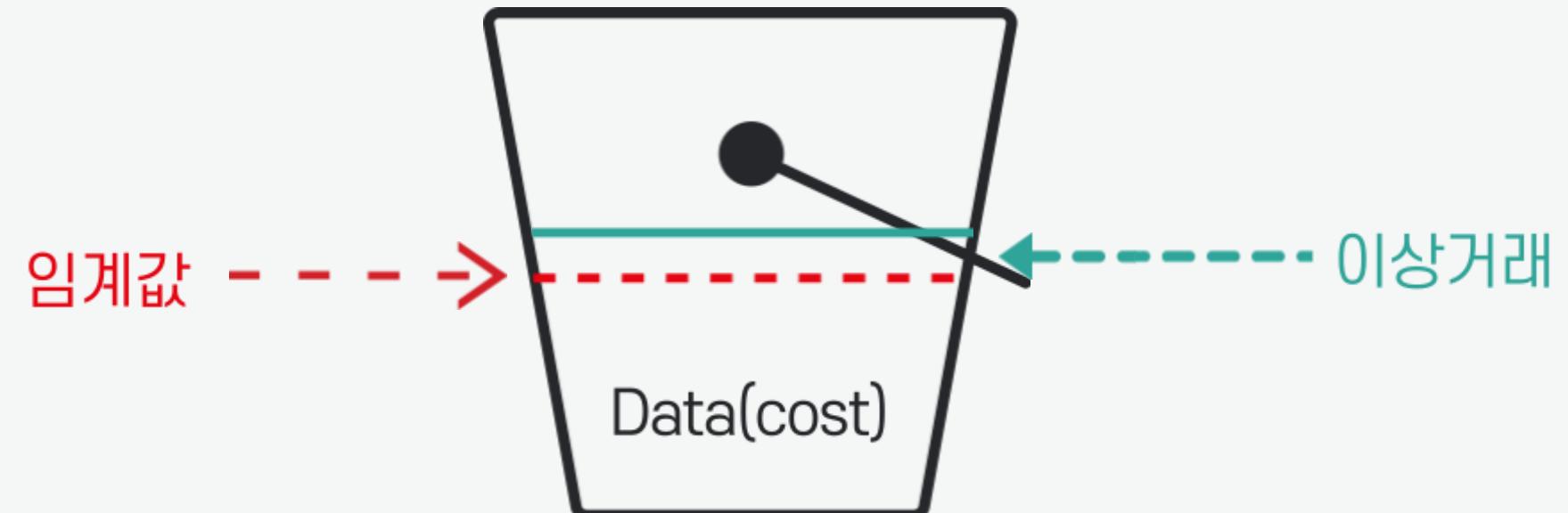
```
cost = tf.reduce_sum(tf.square(tf.subtract(decoder,X)))  
optimizer = tf.train.AdamOptimizer().minimize(cost) ↓
```

Training

```
sess.run(tf.global_variables_initializer())
batch_size=500
    total_train_batch=int(trainX.shape[0]/batch_size)
    for epoch in range(3):
        total_cost=0
        for i in range(total_train_batch):
            batch_xs = next_batch(trainX,i,batch_size)
            _, cost_val = sess.run([optimizer,cost],feed_dict={X:batch_xs})
            total_cost += cost_val
```

- 배치 사이즈를 500으로 정의
- 에포크를 3으로 정의하여 **총 3회 학습**
- **임계값이 넘는** 데이터를 **이상거래**로 판단할 수 있게 학습

```
training
epoch: 1 cost: 75.08
epoch: 2 cost: 24.43
epoch: 3 cost: 24.13
train complete!
```



예측 & 성능평가

```
limit = 0.05 # 임계값 0.05  
for i in range(len(fraud)):  
    a=sess.run(cost,feed_dict={X:fraudX[i].reshape(1,28)})  
    if a >= limit:  
        fraud_count +=1  
        a_list.append(fraudX[i])  
  
for i in range(len(testX[0].shape)):  
    b=sess.run(cost,feed_dict={X:testX[i].reshape(1,28)})  
    if b >= limit:  
        normal_count +=1  
        b_list.append(b)  
  
print('fraud : ', fraud_count,'/',len(fraud), 'normal : ', normal_count,'/',len(testX))
```

→ 테스트 용 데이터 중
사기 거래 데이터 (fraud) : 492 개

→ 테스트 용 데이터 중
정상 거래 데이터의 20% : 56863개

```
testing..  
임계값 : 0.05  
(fraud : 462 / 492)normal : 0 / 56863
```

3가지 머신 러닝 모델

지도 학습 모델
비지도 학습 모델

	정확도	정밀도	재현율	F점수
랜덤 포레스트	0.99	0.91	0.75	0.82
인공 신경망	0.99	0.82	0.80	0.81
오토인코더	0.99	1	0.93	0.96

결론 & 느낀점

임계값

오토인코더의 알고리즘 원리

학습용, 테스트용 데이터의 비율