Just Start up your Deep Learning for your Future !

슬로우캠퍼스 딥러닝 스쿨

– Kaggle
Credit Card Fraud Detection

한대희 @ 슬로우캠퍼스

딥러닝 교육/연구/컨설팅

daehee@slowcampus.com

http://medium.com/@slowcampus

http://slowcampus.com

슬로우캠퍼스

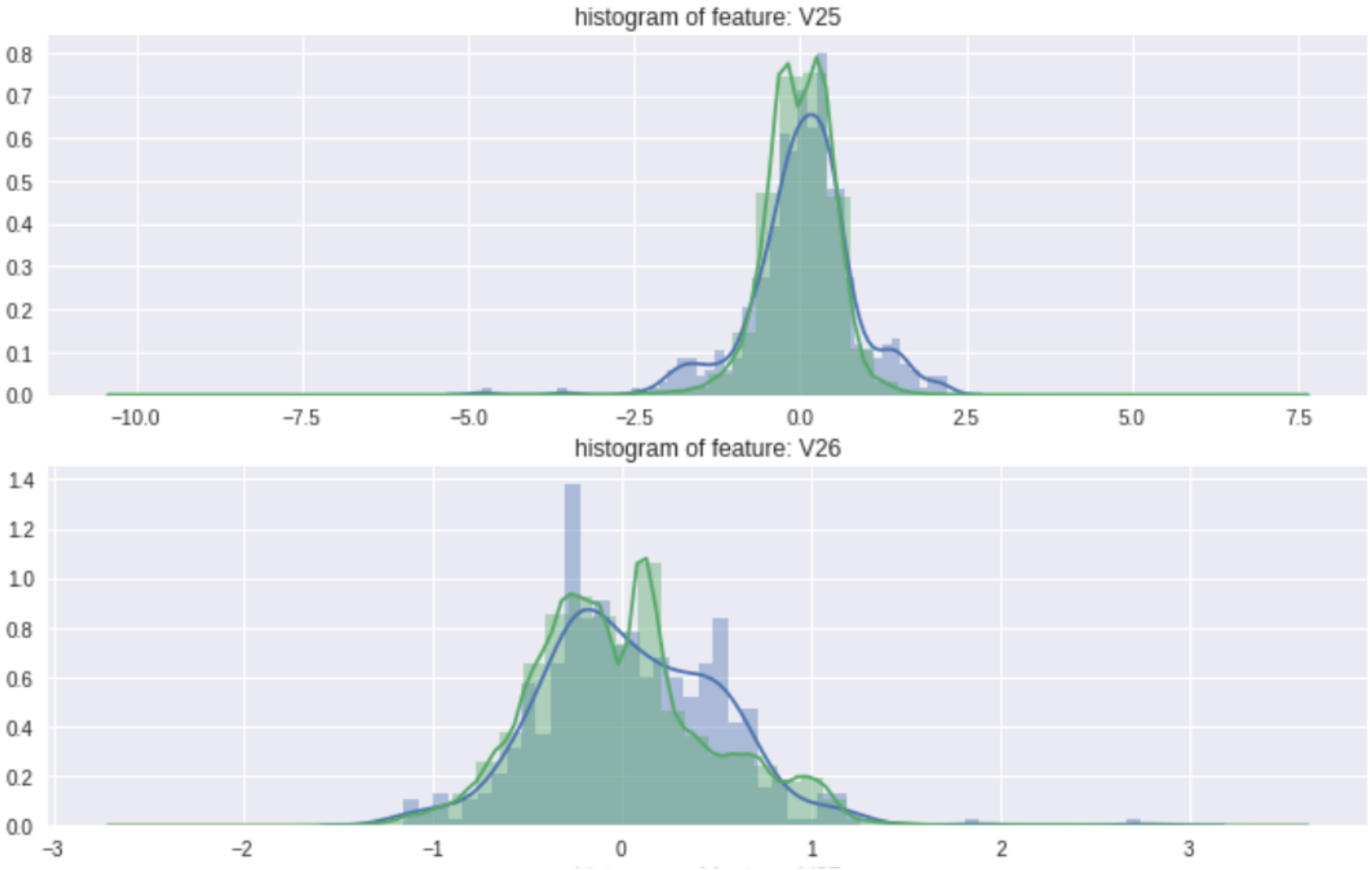
SLOW는 SW(소프트웨어) 입니다

슬로우는 학습과 성장을 의미합니다

```
#Drop all of the features that have very similar distributions between the two types of transactions.
df = df.drop(['V28','V27','V26','V25','V24','V23','V22','V20','V15','V13','V8'], axis =1)

#Based on the plots above, these features are created to identify values where fraudulent transaction are
# fraudulent transaction 이 더 잘나타는 방향으로 자료 정리 V1_~ V21_ 생성
df['V1_'] = df.V1.map(lambda x: 1 if x < -3 else 0)
df['V2_'] = df.V2.map(lambda x: 1 if x > 2.5 else 0)
df['V3_'] = df.V3.map(lambda x: 1 if x < -4 else 0)
df['V4_'] = df.V4.map(lambda x: 1 if x > 2.5 else 0)
df['V5_'] = df.V5.map(lambda x: 1 if x < -4.5 else 0)
df['V6_'] = df.V6.map(lambda x: 1 if x < -2.5 else 0)
df['V7_'] = df.V7.map(lambda x: 1 if x < -3 else 0)
df['V9_'] = df.V9.map(lambda x: 1 if x < -2 else 0)
df['V10_'] = df.V10.map(lambda x: 1 if x < -2.5 else 0)
df['V11_'] = df.V11.map(lambda x: 1 if x > 2 else 0)
df['V12_'] = df.V12.map(lambda x: 1 if x < -2 else 0)
```

Normal과 Fraud의 **구분**이 명확하지 않은 feature 제외



Normal과 Fraud의 **구분**이 명확한 feature

Normal 컬럼 생성
Class가 0 이면 1, Class가 1 이면 0

```
# Create a new feature for normal (non-fraudulent) transactions.
# Normal 컬럼 생성 Class가 0 이면 1, Class가 1이면 0
df.loc[df.Class == 0, 'Normal'] = 1
df.loc[df.Class == 1, 'Normal'] = 0
```

Class	Normal
0	1
1	0
0	1
0	1
0	1

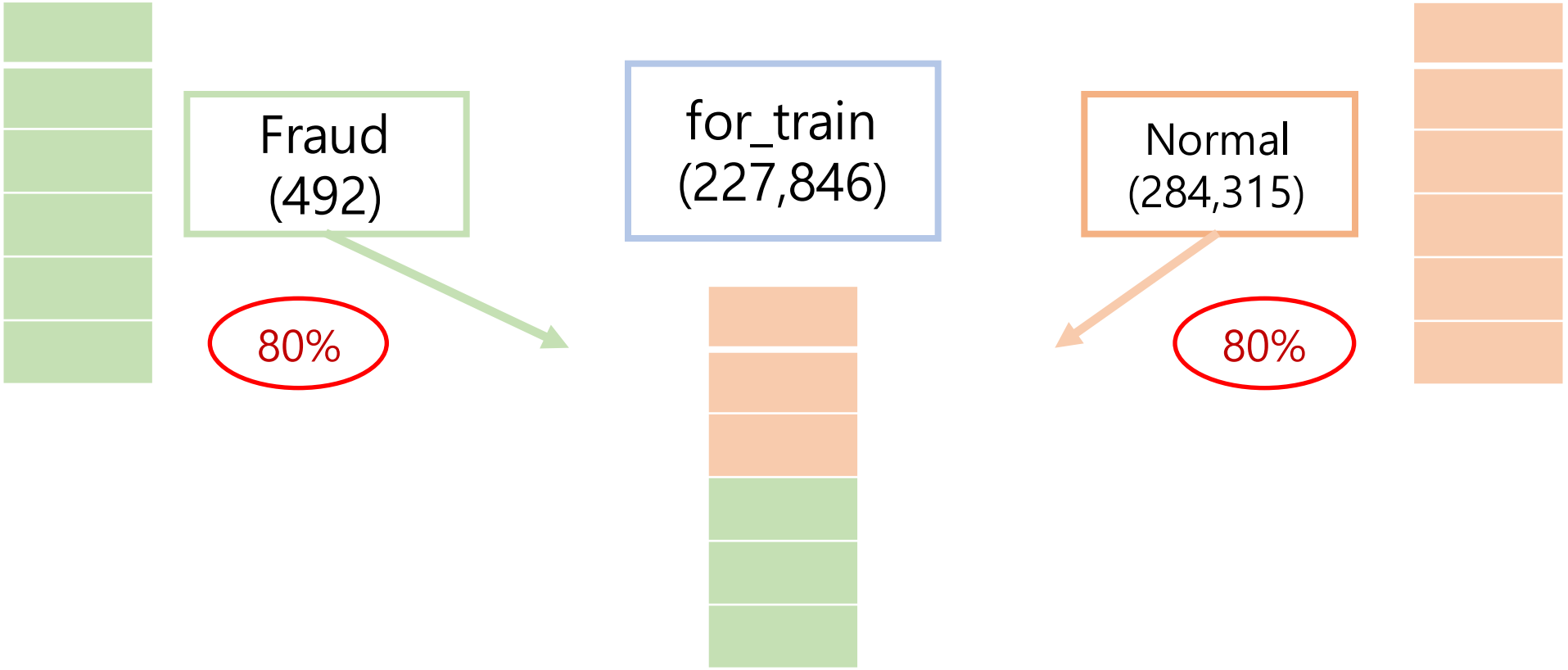
```
# 492 fraudulent transactions, 284,315 normal transactions.
# 0.172% of transactions were fraud.
print(df.Normal.value_counts())
print()
print(df.Fraud.value_counts())
```

Normal과 Fraud의 Count

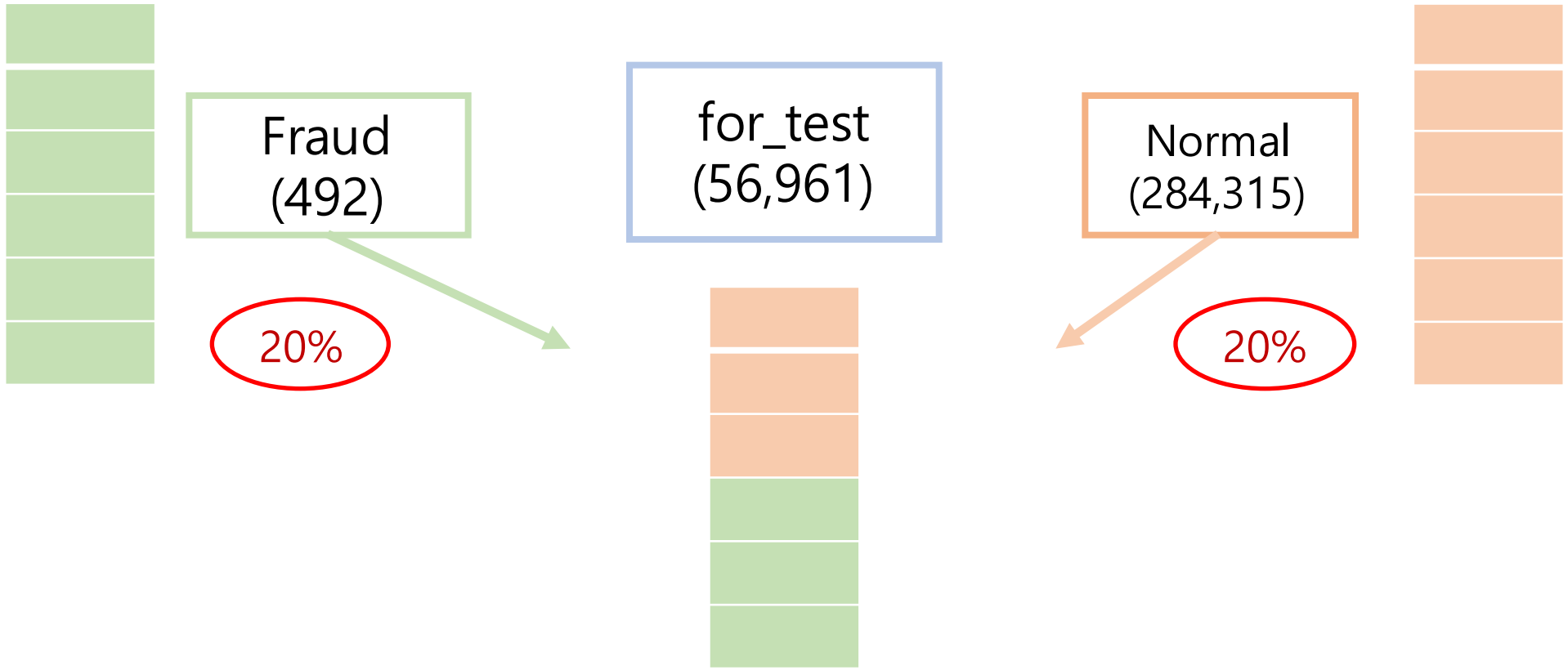
```
1.0    284315
0.0       492
Name: Normal, dtype: int64

0    284315
1       492
Name: Fraud, dtype: int64
```

```
# Set X_train equal to 80% of the fraudulent transactions.
FraudSample  = Fraud.sample(frac=0.8)
NormalSample = Normal.sample(frac=0.8)
count_Frauds = len(FraudSample)
# Add 80% of the normal transactions to X_train.
for_train = pd.concat([FraudSample, NormalSample], axis=0)
```

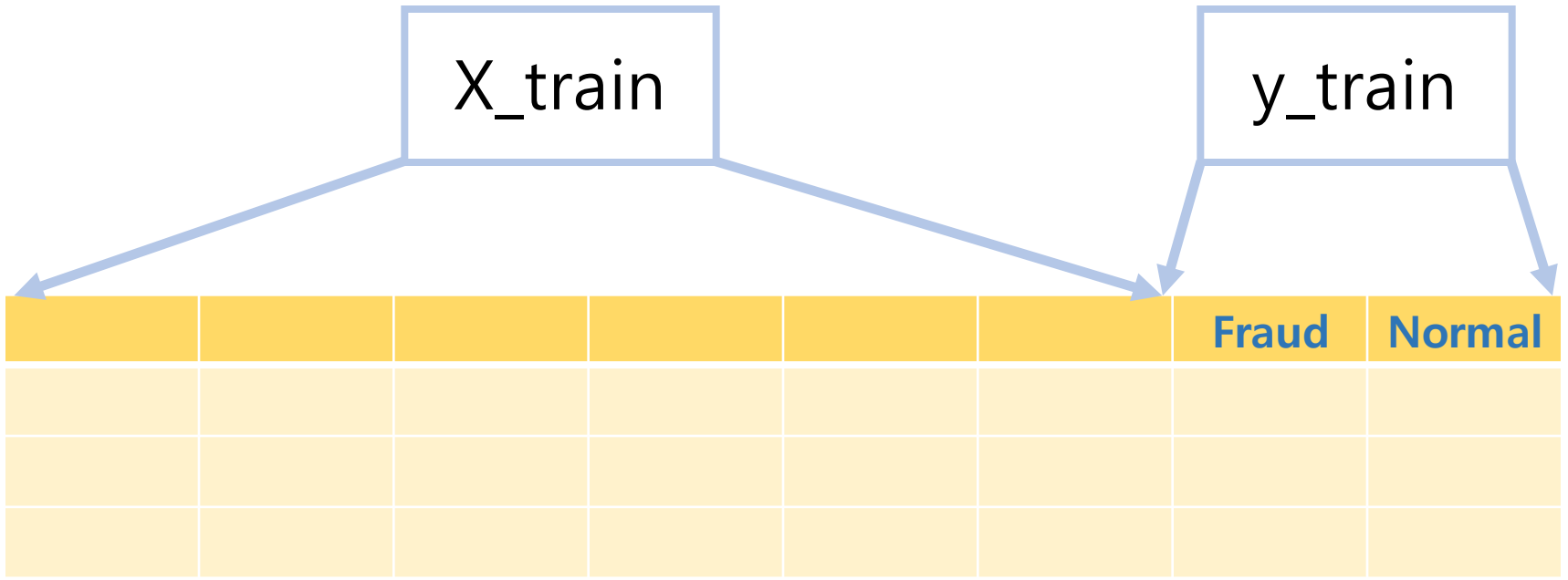


```
# X_test contains all the transaction not in X_train.
for_test = df.loc[~df.index.isin(for_train.index)]
```

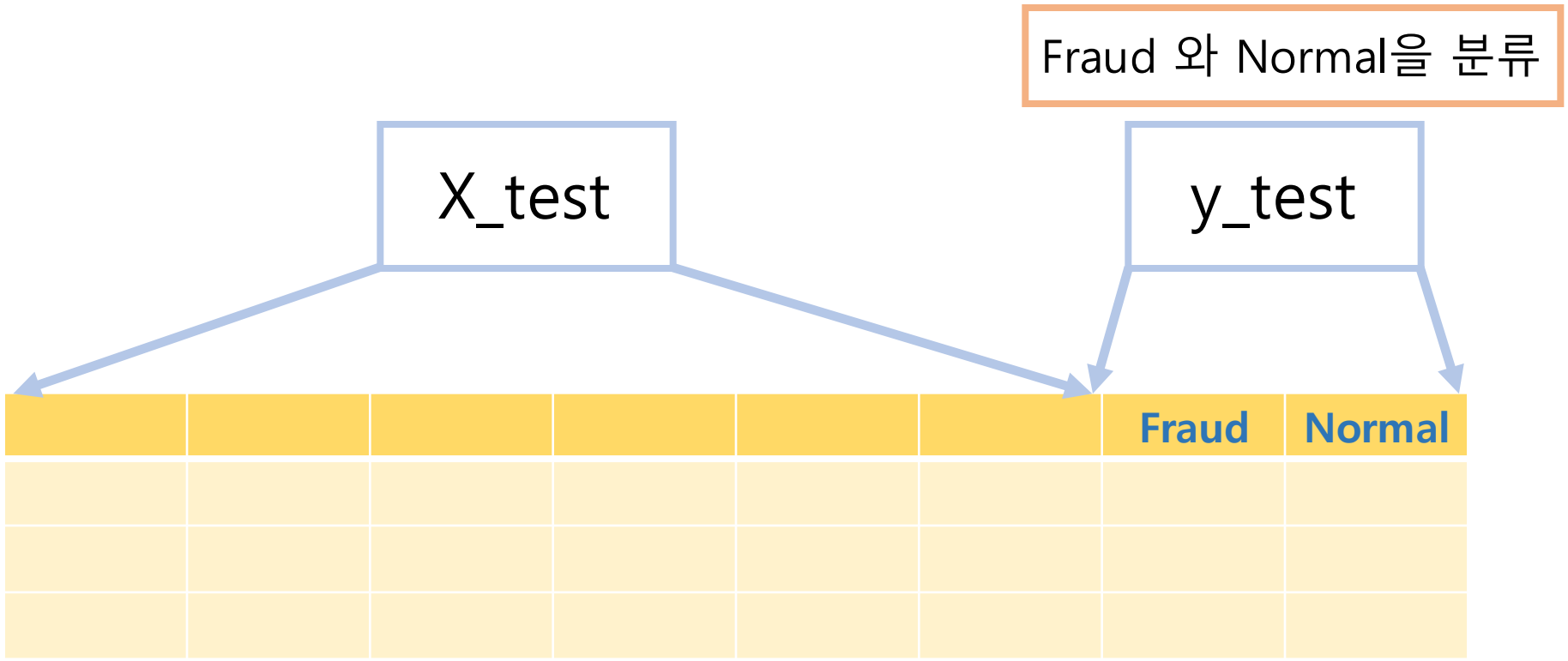


```
# Add our target features to y_train and y_test.
X_train = for_train.drop(['Fraud', 'Normal'], axis = 1)
y_train = for_train[['Fraud', 'Normal']]
```

Fraud 와 Normal을 분류



```
X_test = for_test.drop(['Fraud', 'Normal'], axis = 1)
y_test = for_test[['Fraud', 'Normal']]
```



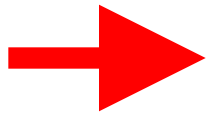
```
ratio : 578.2893401015228
```

```
ratio = len(X_train) / count_Frauds
print('ratio :', ratio)

y_train.Fraud *= ratio
y_test.Fraud *= ratio
```

Fraud 값에 ratio를 적용
(**weight** 부여)

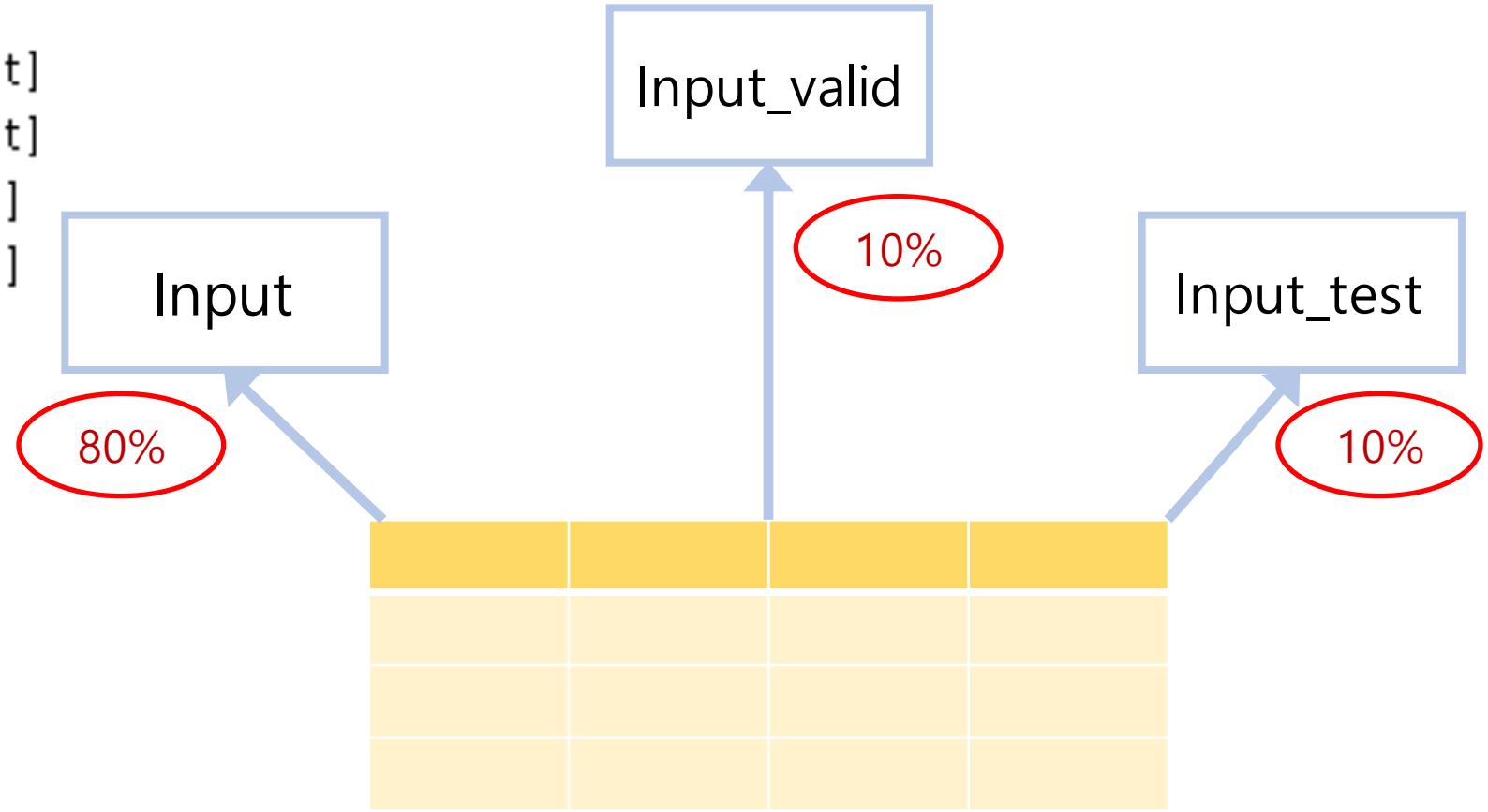
Fraud
0
1
0
1
0



Fraud
0
578.2893
0
578.2893
0

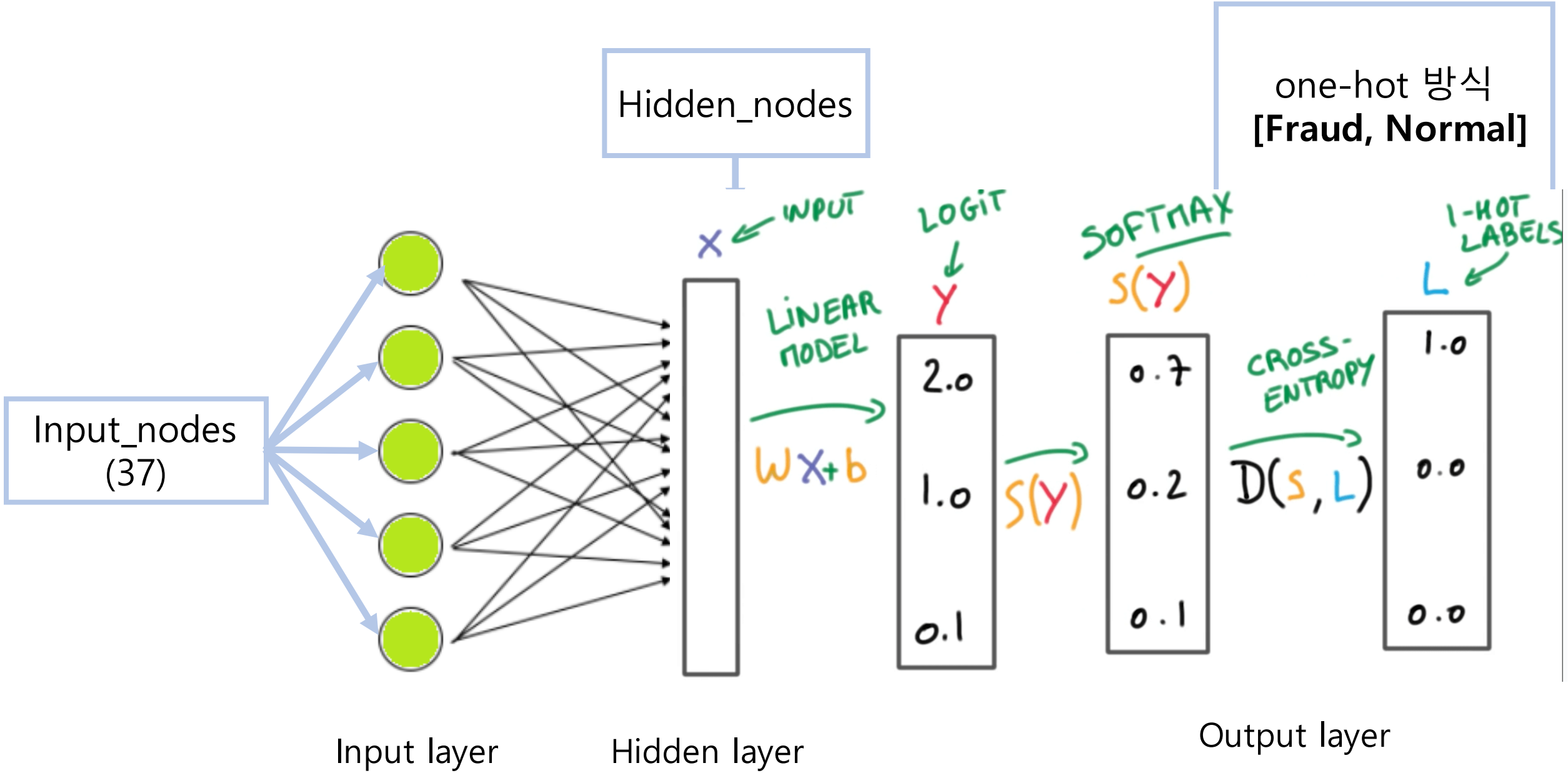
```
# Split the testing data into validation and testing sets
split = int(len(y_test)/2)
print('split : ', split)

inputX = X_train.as_matrix()
inputY = y_train.as_matrix()
inputX_valid = X_test.as_matrix()[:split]
inputY_valid = y_test.as_matrix()[:split]
inputX_test = X_test.as_matrix()[split:]
inputY_test = y_test.as_matrix()[split:]
```



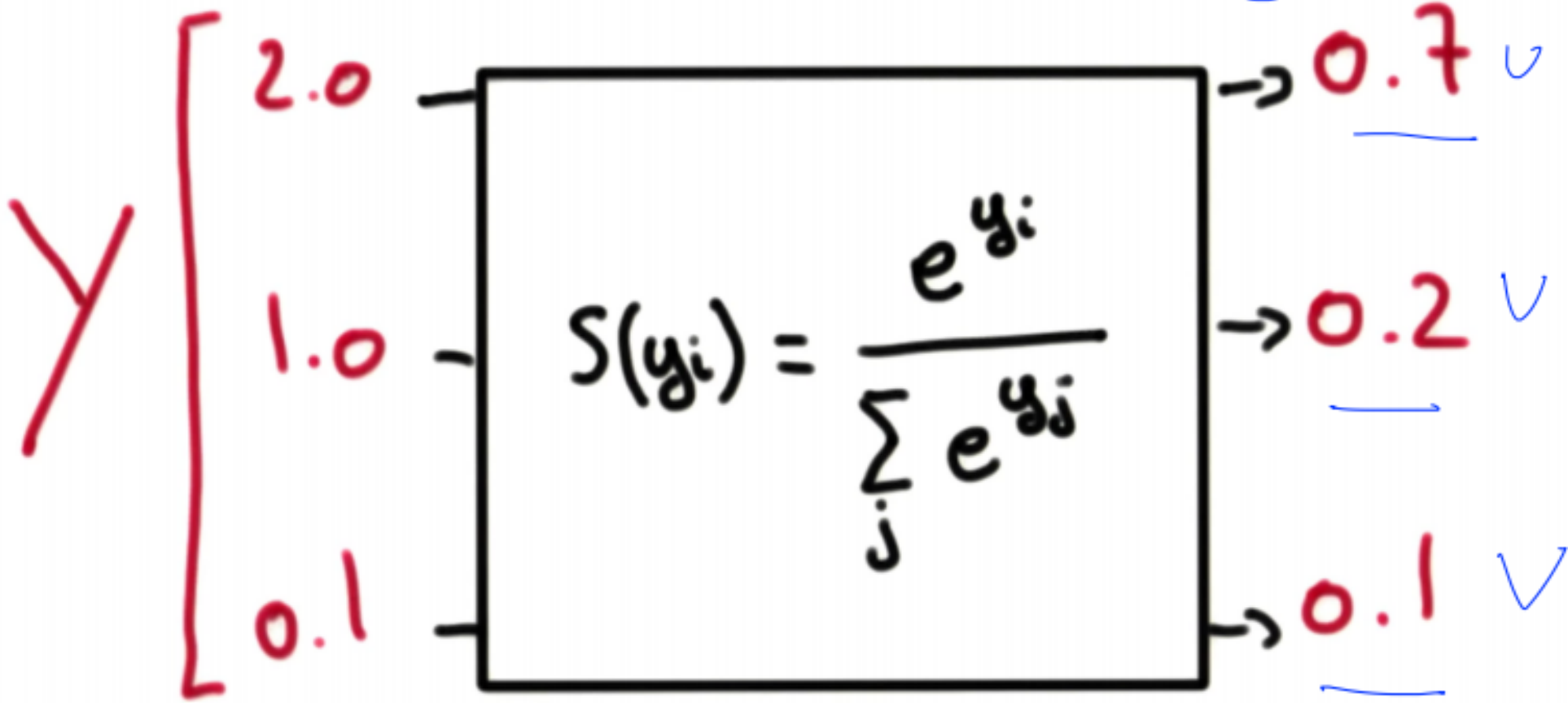
NN의 출력값(output y_):

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
predict_by_nn = tf.nn.softmax(y)
decision_by_nn = tf.argmax(y, 1)
actual = tf.argmax(y_, 1)
```

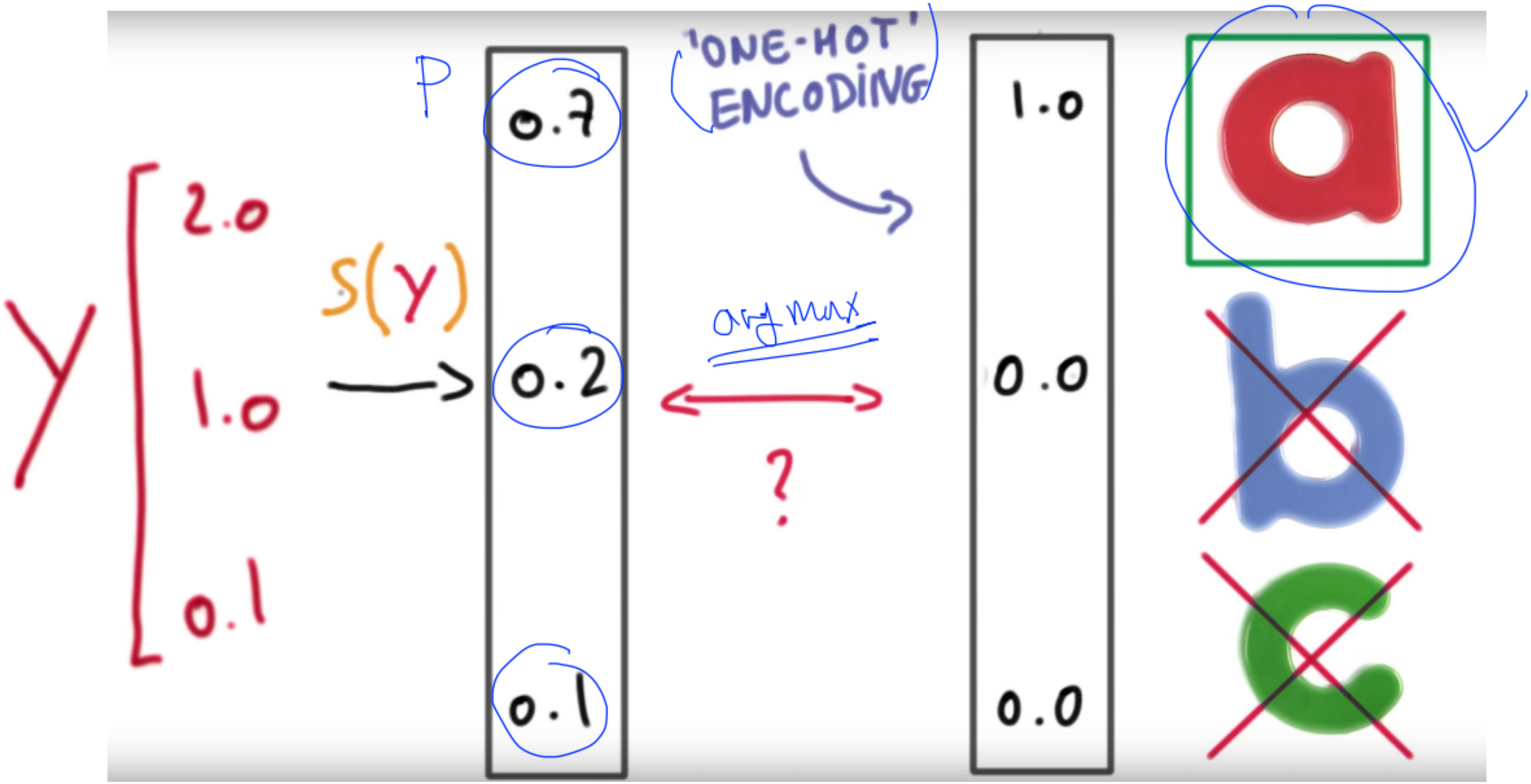


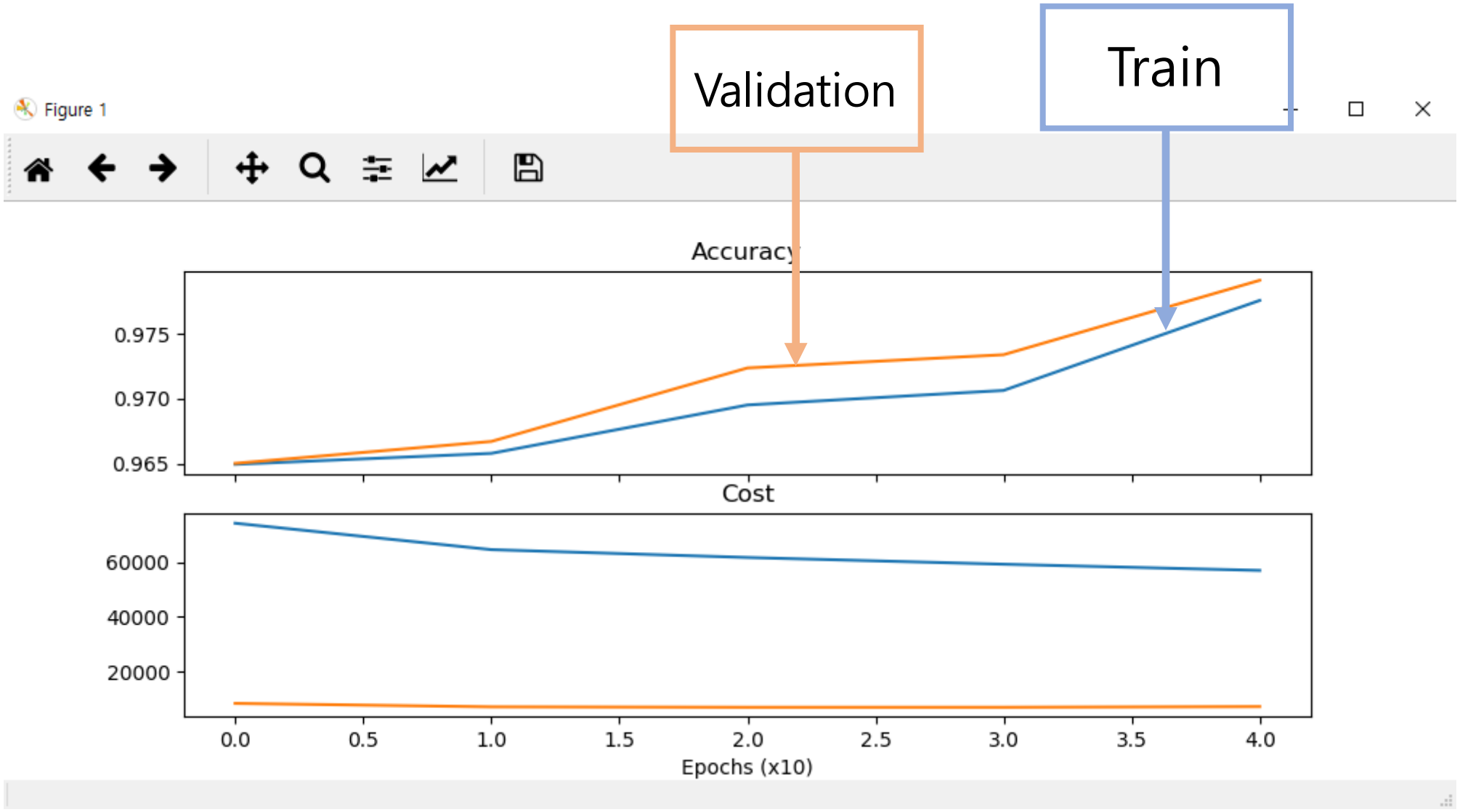
SOFTMAX

① 0~1
② 합 = 1



SCORES ⟶ PROBABILITIES

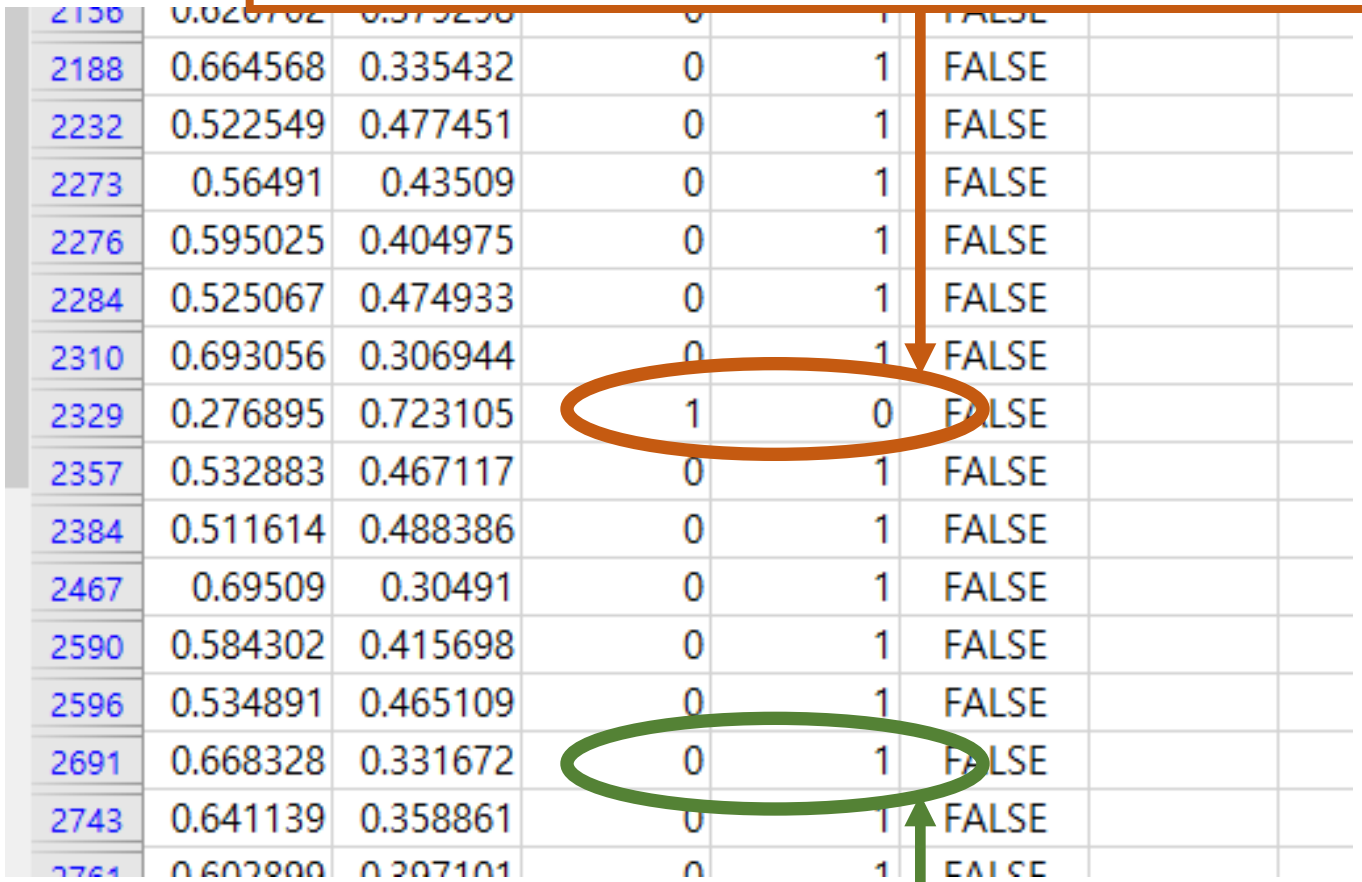




```
test_predict_by_nn, test_decision_by_nn, test_actual = sess.run([predict_by_nn,
decision_by_nn, actual], feed_dict={x: inputX_test, y_: inputY_test, pkeep: 1})

res = [(a[0], a[1], b,c) for a,b,c in zip(test_predict_by_nn, test_decision_by_nn, test_a
resdf = pd.DataFrame(data=res, columns=['Fr', 'Nm', 'NN', 'ACTUAL'])
)
print('FN', resdf[(resdf.NN == 1) & (resdf.ACTUAL == 0)].values.shape[0])
print('FP', resdf[(resdf.NN == 0) & (resdf.ACTUAL == 1)].values.shape[0])
print('TN', resdf[(resdf.NN == 1) & (resdf.ACTUAL == 1)].values.shape[0])
print('TP', resdf[(resdf.NN == 0) & (resdf.ACTUAL == 0)].values.shape[0])
```

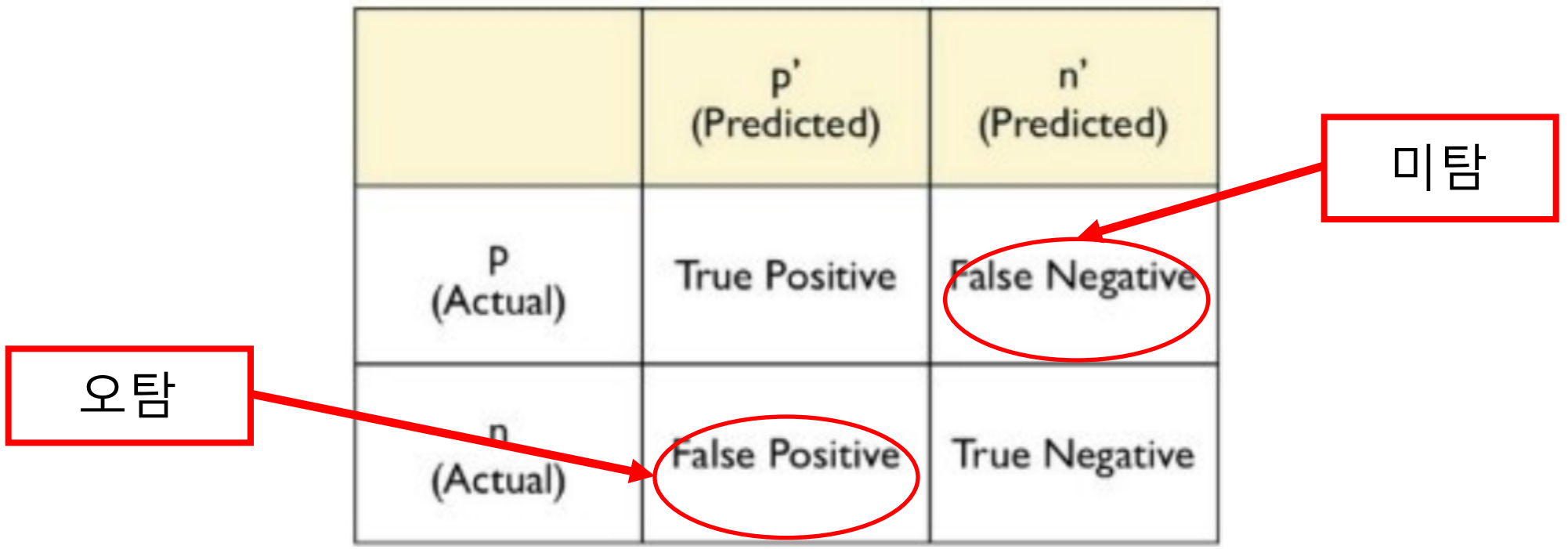
Fraud 데이터를 Normal로 분류 오류
FN(False Negatives)



2188	0.664568	0.335432	0	1	FALSE
2232	0.522549	0.477451	0	1	FALSE
2273	0.56491	0.43509	0	1	FALSE
2276	0.595025	0.404975	0	1	FALSE
2284	0.525067	0.474933	0	1	FALSE
2310	0.693056	0.306944	0	1	FALSE
2329	0.276895	0.723105	1	0	FALSE
2357	0.532883	0.467117	0	1	FALSE
2384	0.511614	0.488386	0	1	FALSE
2467	0.69509	0.30491	0	1	FALSE
2590	0.584302	0.415698	0	1	FALSE
2596	0.534891	0.465109	0	1	FALSE
2691	0.668328	0.331672	0	1	FALSE
2743	0.641139	0.358861	0	1	FALSE

Normal 데이터를 Fraud로 분류 오류
FP(True Positives)

Confusion Matrix, 혼동행렬



TP(True Positives) : 실제값과 예측치 모두 True인 빈도
TN(True Negatives) : 실제값과 예측치 모두 False인 빈도
FP(True Positives) : 실제값은 False이나 True로 예측한 빈도
FN(False Negatives) : 실제값은 True이나 False로 예측한 빈도

Confusion Matrix, 혼동행렬

	p' (Predicted)	n' (Predicted)
p (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

Accuracy(정분류율) $= \frac{TP+TN}{P+N}$

Precision(정확도) $= \frac{TP}{TP+FP}$

Recall(재현율) $= \frac{TP}{TP+FN}$

Test Data의 deep-learning 결과

Confusion Matrix value

FN 4
FP 1140
TN 27288
TP 49

Accuracy(정분류율) $= \frac{TP+TN}{P+N}$

Precision(정확도) $= \frac{TP}{TP+FP}$

Recall(재현율) $= \frac{TP}{TP+FN}$

Acc - 0.92

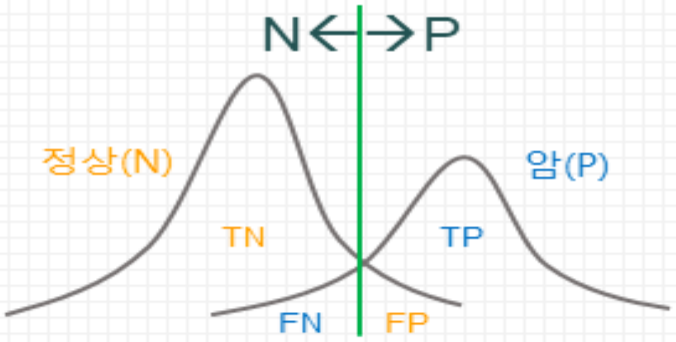
Precision - 0.018

Recall - 0.932

30 ROC curve & AUC

- TP / TN / FP / FN

1000	정상판정	암판정
정상환자	988 TN	2 FP
암환자	1 FN	9 TP



Precision : 내가 내린 양성 판단 중에...

Recall : 실제 있는 양성 중에...

$$Acc = \frac{TP + TN}{All} \quad P = \frac{TP}{TP + FP} \quad R = \frac{TP}{TP + FN}$$

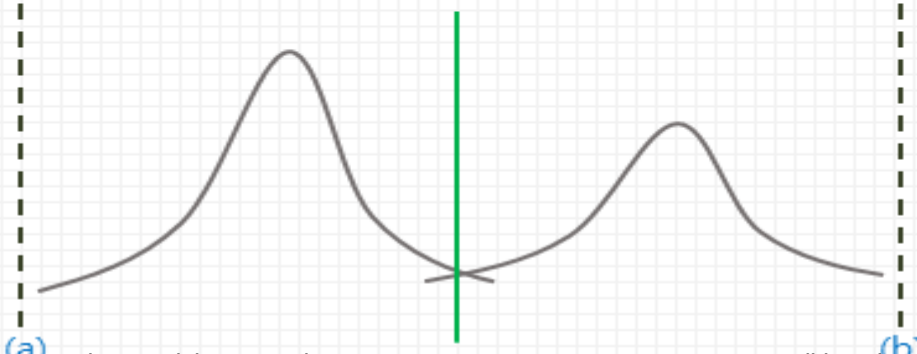
True positive rates :
(= recall, sensitivity)

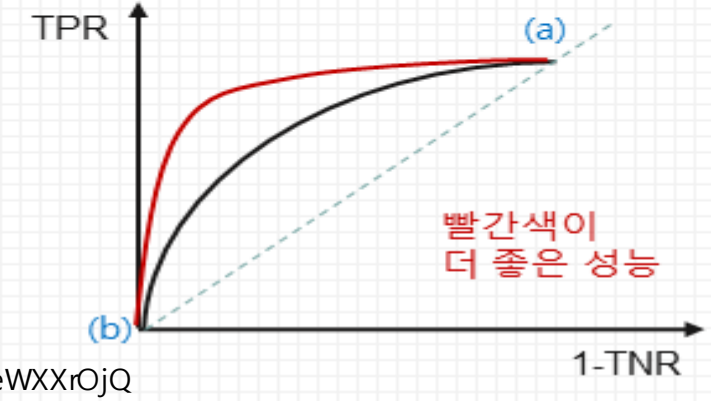
$$TPR = R = \frac{TP}{TP + FN}$$

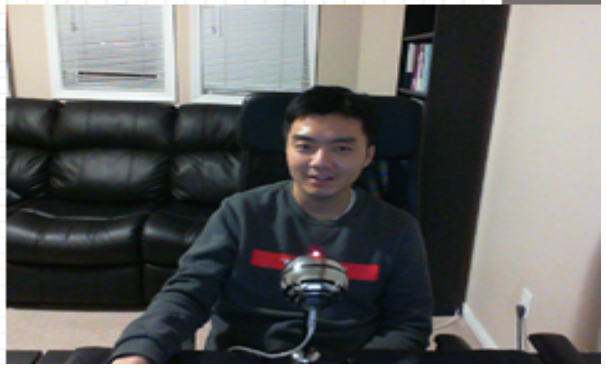
True negative rates :
(= specificity)

$$TNR = \frac{TN}{TN + FP}$$

- ROC curve & AUC







http://t-robotics.blogspot.kr/2017/10/20-roc-auc-precision-recall.html#.WjoeWXXrOjQ

ROC Curve