2017-2 데이터마이닝실습

예금상품 가입 예측

3조

권상우, 양지성, 이은지, 장성훈, 홍성재

INDEX

1. 프로젝트 개요

1. 프로젝트 개요

데이터 소개

- 은행의 정기 예금상품 가입 여부를 예측
- 포르투갈 은행의 ’08년 5월 ~ ’10년 11월 데이터
- 총 41188개의 데이터 / 20개의 칼럼
- UCI Machine Learning Repository 데이터

1. 프로젝트 개요

변수 소개

- **개인 정보** — 나이, 직업, 결혼, 채무 불이행, 주택융자
- **마케팅 정보** — 마케팅 수단, 마지막 연락의 달, 마지막 연락의 요일
  마지막 연락과의 기간, 이번 시즌 연락 횟수
- **전 마케팅 정보** — 지난 시즌 연락 횟수, 지난 시즌 마지막 연락과의 기간
  지난 시즌 상품가입 여부, 이전 연락 횟수
- **경제적 수치** — 소비자 물가 지수, 소비자 신뢰 지수, 기준 금리, 총 고용인 수

2. 데이터 전처리

2. 데이터 전처리

- Tensorflow의 Transform 모듈 → 정보 적어서 X

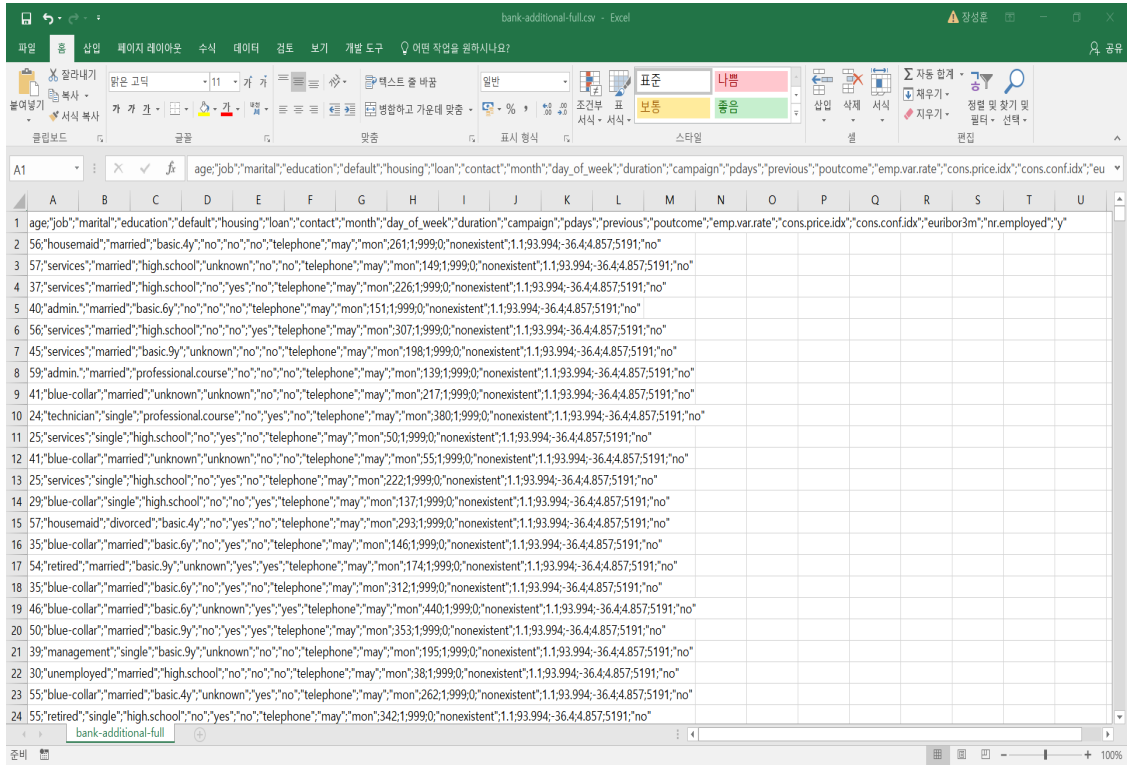
- 1. Pandas로 데이터 정리

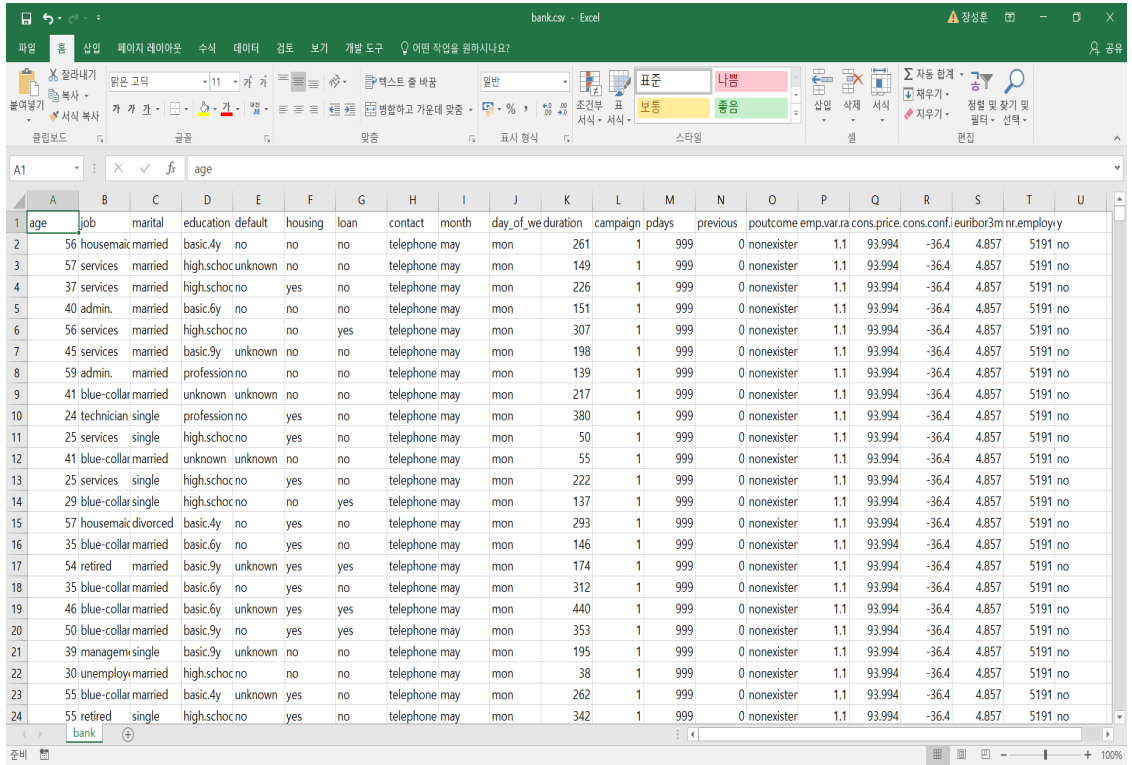
  2. R을 이용하여 분석에 사용할 열 선별

  3. Tensorflow로 불러오기

2. 데이터 전처리

- 엑셀 : 데이터 → 텍스트 나누기





2. 데이터 전처리

- CSV 파일 불러오기

```
df = pd.read_csv('C:/ProgramData/Anaconda3/ML script/bank-additional.csv',encoding = 'CP949’)
#df = df.dropna(axis=0)
```

- “y”열의 명목변수 → 0 or 1

```
cleanup_nums = {"y":{"yes": 1, "no": 0}}
df.replace(cleanup_nums, inplace=True)
```

2. 데이터 전처리

- data type이 object인 열 모두 더미변수 처리

```
conti_var=df.columns[df.dtypes!='object']
cate_var=df.columns[df.dtypes=='object'].difference(['y'])
dummy_var=pd.get_dummies(df[cate_var])
df_2=pd.concat([df[conti_var],dummy_var],axis=1)
df_2_y = df_2['y']
df_2_y = pd.DataFrame(data=df_2_y)
df_2 = df_2.drop('y', axis =1)
df_2 = pd.concat([df_2, df_2_y], axis=1)
df_2 = df_2.astype('float32')
df5 = pd.DataFrame(data=df_2, dtype=np.float32)
```

2. 데이터 전처리

- 총 41188개의 행 / 64개의 열

- 로지스틱 회귀분석(R)을 이용해서 불필요한 열 삭제

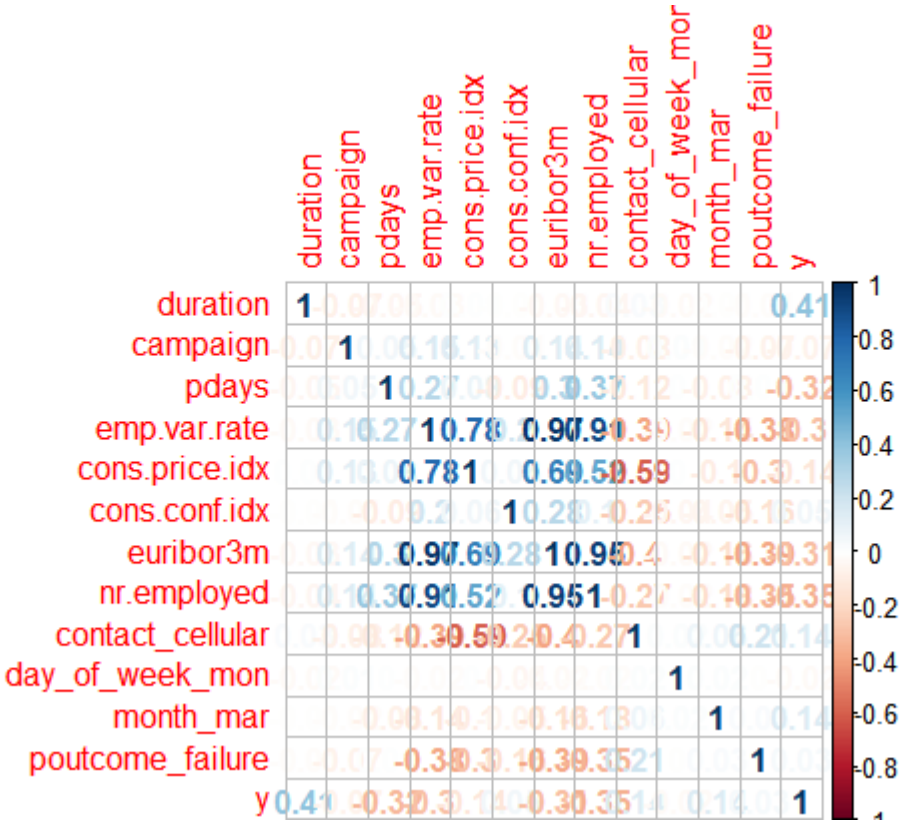
  1. 유의도가 낮게 나온 열 제거

  2. 같은 열에서 파생된 더미변수들은 가장 유의한 것만 남기고 나머지 제거

- 그 결과, 64개 열 → 13개 열

2. 데이터 전처리

- 상관분석 corrplot() → 최종적으로 9개의 독립변수, 1개의 종속변수
- duration / campaign / pdays / emp.var.rate / cons.conf.idx / contact_cellular /
  day_of_week_mon / month_mar / poutcome_failure / y



2. 데이터 전처리

```
xy = np.loadtxt('bank4-additional.csv',delimiter=',',dtype=np.float32)
xy_1 = xy[:,0:5]
xy_2 = xy[:,5:10]
```

- xy_1에서 각각의 열 정규분포 표준화

```
xy_standardized_skl = StandardScaler().fit_transform(xy_1)
xy = np.concatenate((xy_standardized_skl,xy_2), axis=1)
x_data = xy[:,:-1]
y_data = xy[:,[-1]]
```

2. 데이터 전처리

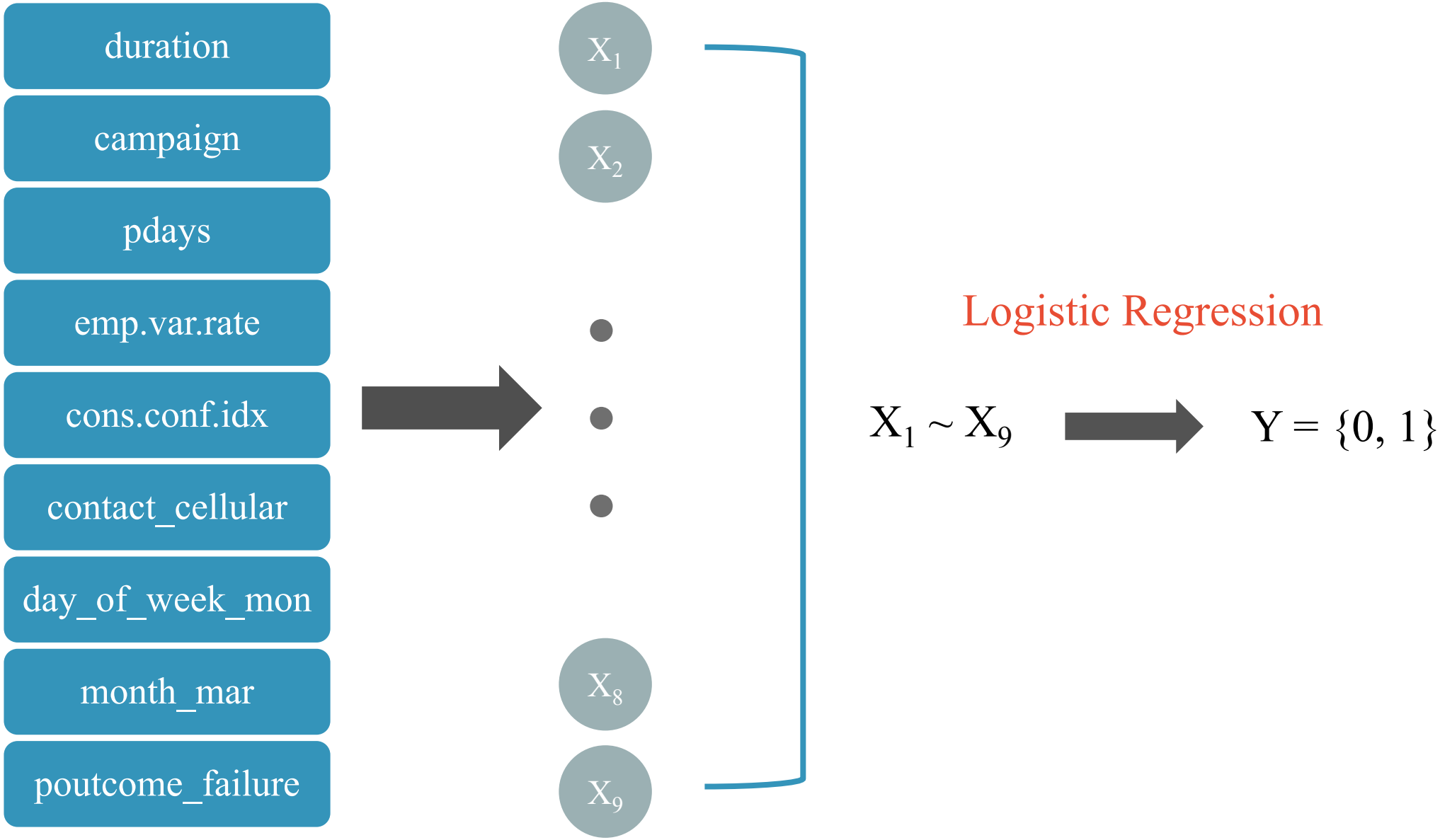
print(x_data)

print(y_data)

```
[[ 0.903952   -0.20922659  0.20103207 ...,  0.         0.         0.        ]
 [ 0.35030031  0.56962967  0.20103207 ...,  0.         0.         0.        ]
 [-0.11696601 -0.59865469  0.20103207 ...,  0.         0.         0.        ]
 ...,
 [-0.75700307 -0.20922659  0.20103207 ...,  1.         0.         1.        ]
 [ 1.06494296 -0.59865469  0.20103207 ...,  0.         0.         0.        ]
 [-0.32114962 -0.59865469  0.20103207 ...,  0.         0.         0.        ]]
[[ 0.]
 [ 0.]
 [ 0.]
 ...,
 [ 0.]
 [ 0.]
 [ 0.]]
```

3. 모델 설명

3. 모델 설명


duration
campaign
pdays
emp.var.rate
cons.conf.idx
contact_cellular
day_of_week_mon
month_mar
poutcome_failure
X_1
X_2
X_8
X_9
Logistic Regression
$X_1 \sim X_9$
$Y = \{0, 1\}$

3. 모델 설명

- **Training Set / Test Set** 구분: 일정한 비율로 무작위 추출

```
from sklearn.cross_validation import train_test_split
xy = np.loadtxt('bank4.csv',delimiter=',',dtype=np.float32)  ← 원본 데이터 불러오기
xy_train,xy_test = train_test_split(xy,test_size=0.2,random_state=777)
```

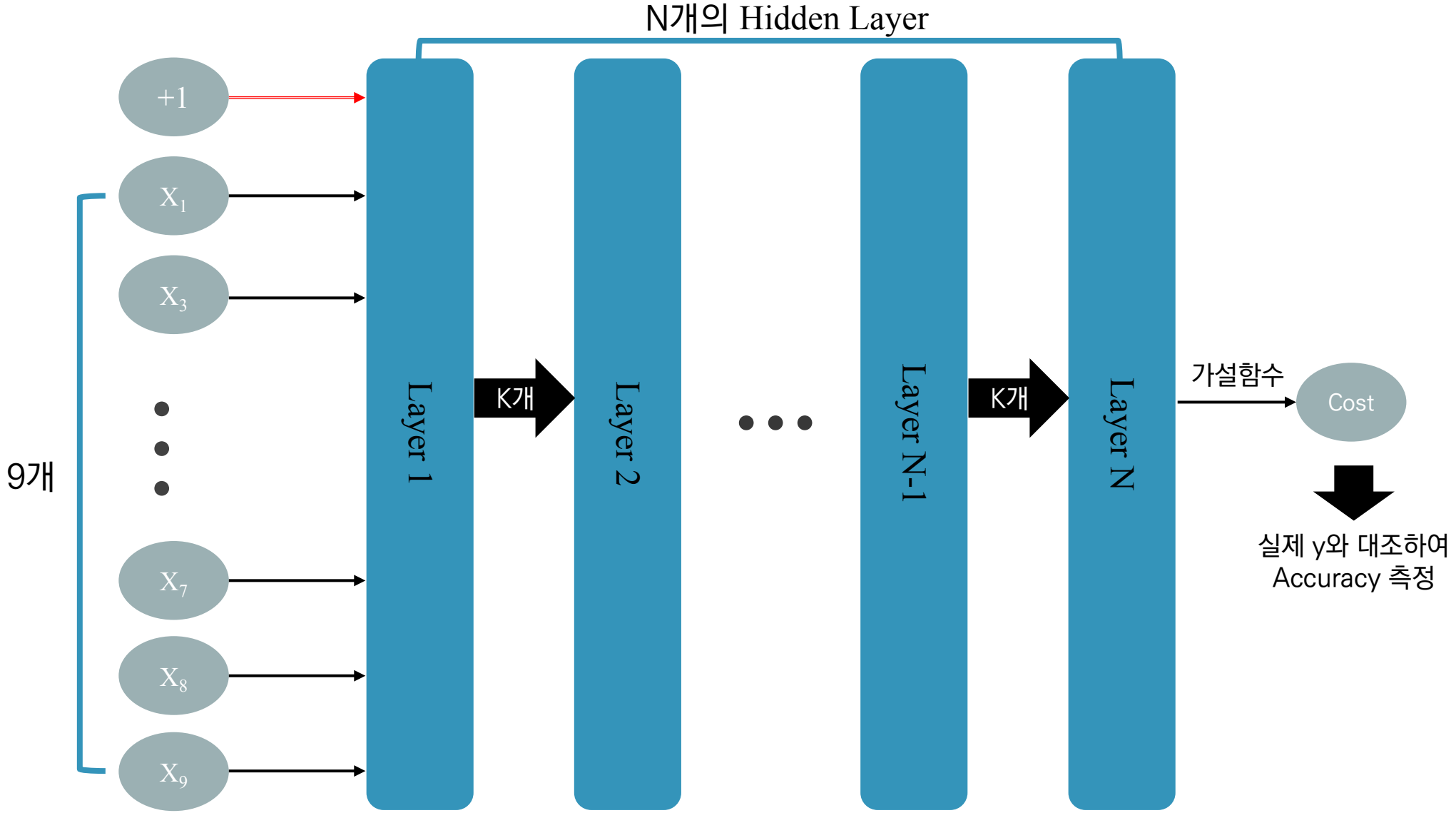
→ *전체 데이터셋의 무작위 20%를 테스트셋으로 설정*

(정규화 과정 생략)

- **Training Set / Test Set** 분할

```
x_data_train = xy_train[:,:-1]
y_data_train = xy_train[:,[-1]]
x_data_test = xy_test[:,:-1]
y_data_test = xy_test[:,[-1]]
```

3. 모델 설명

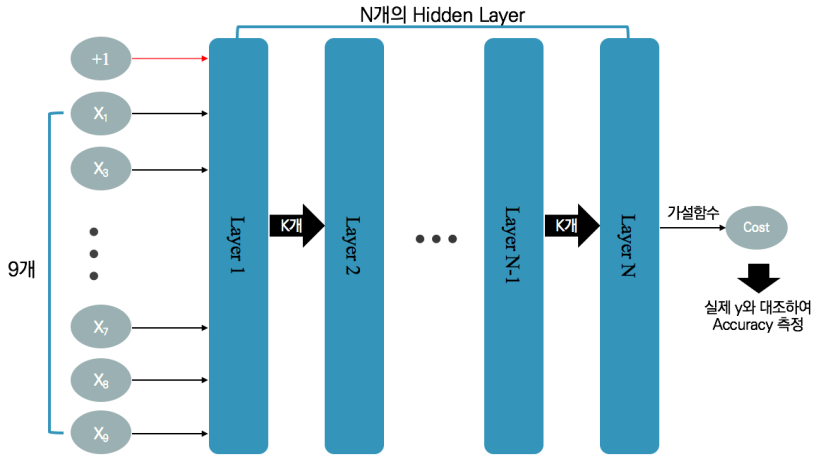

N개의 Hidden Layer
+1
X_1
X_3
X_7
X_8
X_9
9개
Layer 1
K개
Layer 2
Layer N-1
K개
Layer N
가설함수
Cost
실제 y와 대조하여
Accuracy 측정

3. 모델 설명

- **Placeholders**

X = tf.placeholder(tf.float32, [None, 9])
Y = tf.placeholder(tf.float32, [None, 1])

num_hidden = K ← Hidden Layer간 input의 개수



- **Layer 1**

W1 = tf.get_variable("weight1", shape=[9,K], dtype = tf.float32,
initializer=tf.contrib.layers.xavier_initializer()) ← ***Xavier 초기화 적용***

b1 = tf.get_variable("bias1", shape=[K], dtype = tf.float32,
initializer=tf.contrib.layers.xavier_initializer()) ← ***Xavier 초기화 적용***

_L1 = tf.nn.leaky_relu(tf.matmul(X, W1) + b1) ← ***Leaky ReLU/ReLU 적용***

L1 =tf.nn.dropout(_L1, keep_prob) ← ***Dropout 적용***

3. 모델 설명

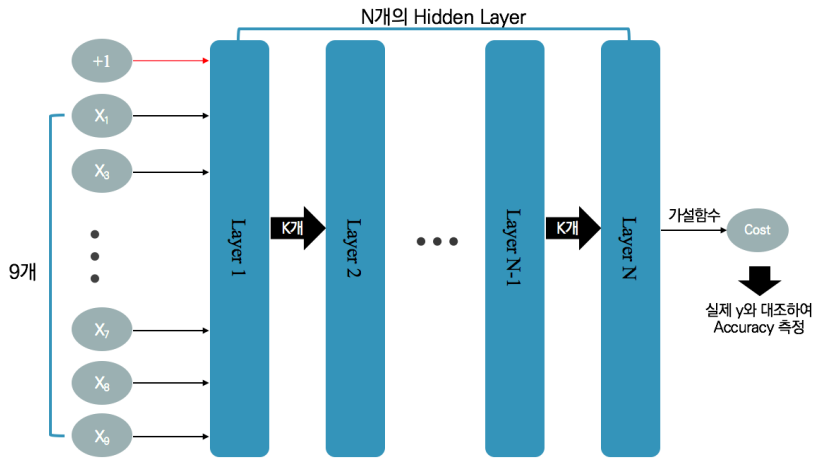
- **Layer N**

```
WN = tf.get_variable("weightN", shape=[K,1], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())

bN = tf.get_variable("biasN", shape=[1], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())

hypothesis = tf.sigmoid(tf.matmul(LN-1, WN) + bN)
```

→ 마지막 Hidden Layer에서는 *Sigmoid* 적용



3. 모델 설명

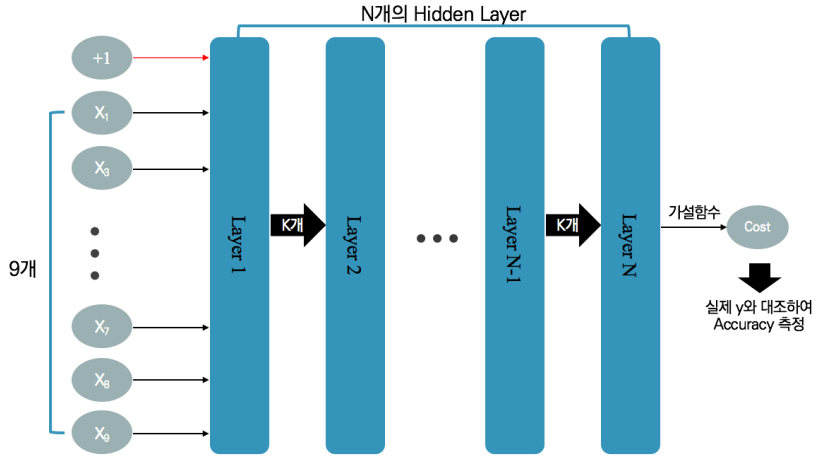
- **비용함수와 최적화**

cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
→ *로지스틱 회귀 cost function*

train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
→ *AdamOptimizer 사용*

- **예측치와 정확도 정의**

predicted = tf.cast(hypothesis > threshold, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))



4. 최적의 모델 찾기

4. 최적의 모델 찾기

Learning rate	0.003
Drop out	0.5~0.7
Threshold	0.65
Layer	5~7
Input	27~45
step	1000

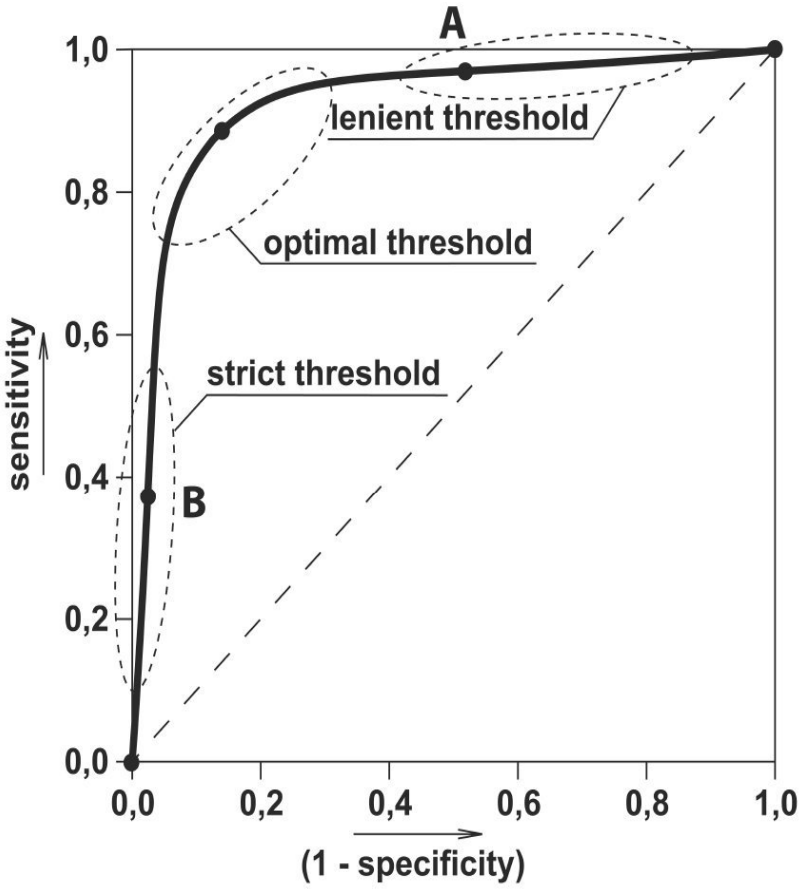
4. 최적의 모델 찾기

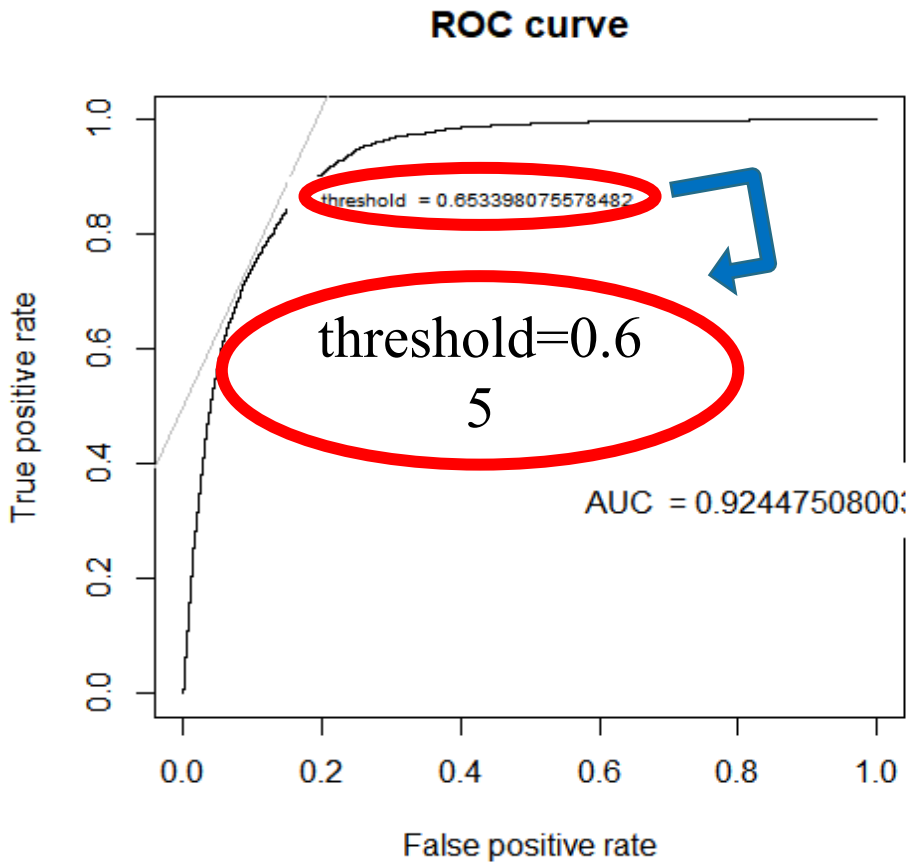
Learning rate / drop out / threshold

```
# Learning Rate
tf.reset_default_graph()
learning_rate = 0.003
keep_prob = tf.placeholder_with_default(0.6, shape=())
threshold = tf.placeholder_with_default(0.65, shape=())
```

4. 최적의 모델 찾기

Learning rate / drop out / threshold





4. 최적의 모델 찾기

Layer & Input

```
# Layer 4
W4 = tf.get_variable("weight4", shape=[num_hidden,num_hidden], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.get_variable("bias4", shape=[num_hidden], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())
_L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L4 = tf.nn.dropout(_L4, keep_prob)

# Layer 5
W5 = tf.get_variable("weight5", shape=[num_hidden,num_out], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.get_variable("bias5", shape=[num_out], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())
hypothesis = tf.sigmoid(tf.matmul(L4, W5) + b5)
hypothesis = tf.clip_by_value(hypothesis,1e-1,1-(1e-1))
```

```
# In[7]:

num_in = 9
num_hidden = 50
num_out = 1

# Layer 1
W1 = tf.get_variable("weight1", shape=[num_in,num_hidden], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.get_variable("bias1", shape=[num_hidden], dtype = tf.float32,
                     initializer=tf.contrib.layers.xavier_initializer())
_L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 =tf.nn.dropout(_L1, keep_prob)
```

4. 최적의 모델 찾기

Step

```
# In[9]:
# Launch Graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(1001):
        sess.run(train, feed_dict={X: x_data_train, Y: y_data_train})
        if step % 100 == 0:
            print("step ",step, "cost ", sess.run(cost, feed_dict={
                  X: x_data_train, Y: y_data_train}))
```

4. 최적의 모델 찾기

Result

```
학습을 시작합니다.
step  0 cost  1.3825
step  100 cost  0.348505
step  200 cost  0.348507
step  300 cost  0.348507
step  400 cost  0.348507
step  500 cost  0.348495
step  600 cost  0.267739
step  700 cost  0.253482
step  800 cost  0.252512
step  900 cost  0.249519
step  1000 cost  0.248062
```

```
Hypothesis:  [[ 0.1       ]
 [ 0.1       ]
 [ 0.1       ]
 ...,
 [ 0.47589734]
 [ 0.1       ]
 [ 0.1       ]]
Correct:  [[ 0.]
 [ 0.]
 [ 0.]
 ...,
 [ 0.]
 [ 0.]
 [ 0.]]
Accuracy:  0.905008
```

정확도 90% 이상! 성공입니다.

4. 최적의 모델 찾기

최대 : accuracy	레이어 수			
Dropout rate 및 input 개수	5	6	7	최대값
0.5	**0.9045**	**0.9066**	**0.9071**	**0.9071**
27	0.8937	0.9066	0.8894	0.9066
32	0.9003	0.9017	0.8894	0.9017
36	0.9045	0.9056	0.8894	0.9056
41	0.9022	0.8968	0.9071	0.9071
45	0.904	0.9036	0.8894	0.904
0.6	**0.9073**	**0.9114**	**0.9061**	**0.9114**
27	0.9007	0.9026	0.8999	0.9026
32	0.9018	0.9027	0.9061	0.9061
36	0.9037	0.904	0.8894	0.904
41	0.903	0.9028	0.8894	0.903
45	0.9073	0.9114	0.8997	0.9114
0.7	**0.909**	**0.9077**	**0.9084**	**0.909**
27	0.9028	0.9077	0.9019	0.9077
32	0.9034	0.9049	0.9038	0.9049
36	0.9072	0.9039	0.9033	0.9072
41	0.909	0.9024	0.9077	0.909
45	0.908	0.905	0.9084	0.9084
최대값	**0.909**	**0.9114**	**0.9084**	**0.9114**

정확도 : 0.88 ~ 0.91

최대 정확도 : 91.14%

Layer: 6 / Input: 45 / Dropout: 0.6

5. 정리 및 결론

5. 정리 및 결론

Processing

Logistic Regression

Neural Network

MODEL

(Layer: 6 / Input: 45 / dropout: 0.6)

5. 정리 및 결론

은행 정기예금 상품에 대한 가입 여부를 예측

은행의 수익 = 대출이자 – **예금이자**

↓ 최적의 모델로 가입예측

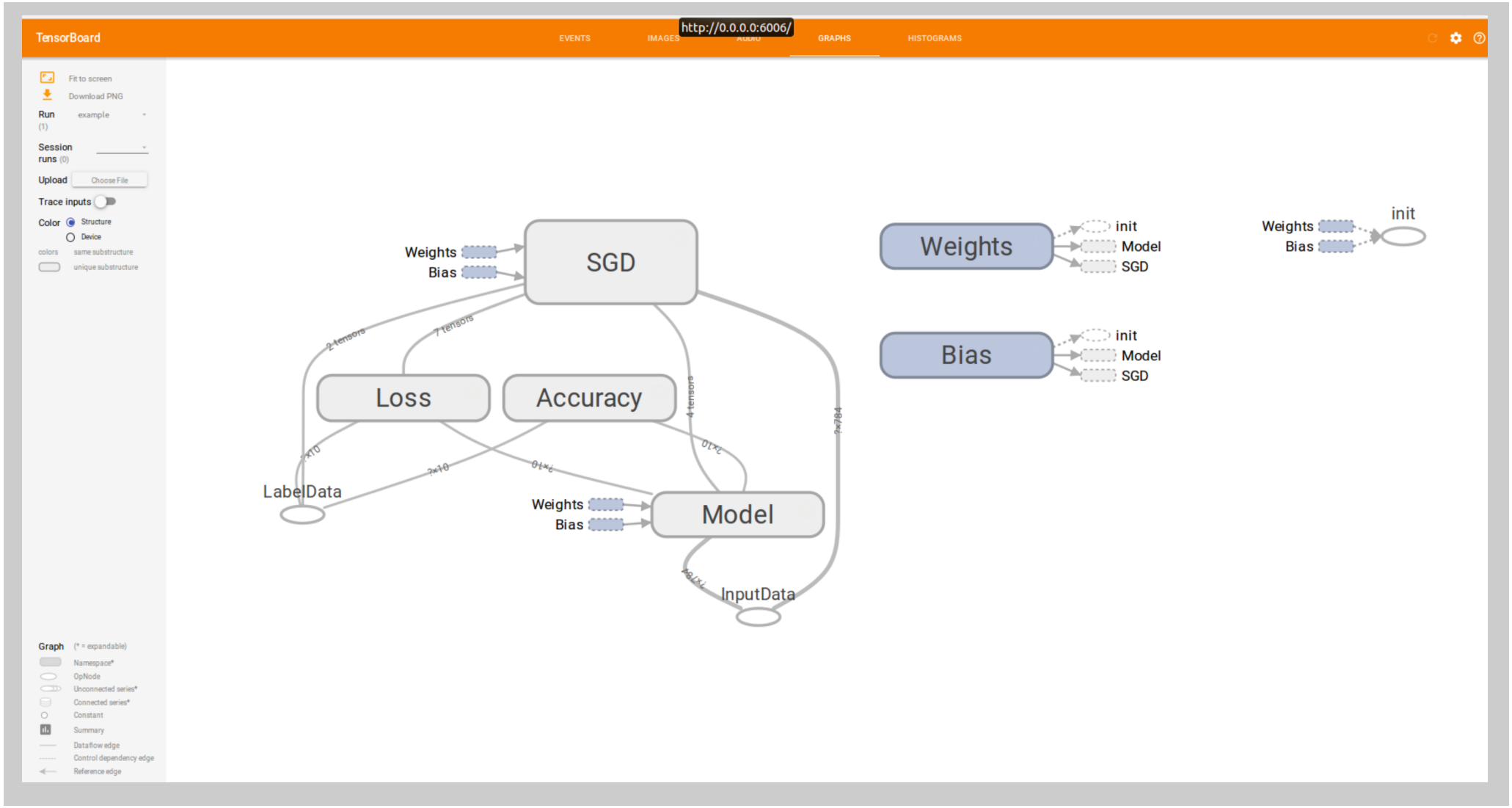
맞춤형 예금서비스 제공, 꾸준하면서도 충분한 고객 확보 가능

5. 정리 및 결론

보완할 점

- Epoch / Mini batch – cost값을 효과적으로 줄이고 정확도를 높여 일반화 가능성 ↑
- Tensorboard – visualization

5. 정리 및 결론


TensorBoard
EVENTS
IMAGES
AUDIO
GRAPHS
HISTOGRAMS
http://0.0.0.0:6006/
Fit to screen
Download PNG
Run (1)
example
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
colors
same substructure
unique substructure
Graph
(* = expandable)
Namespace*
OpNode
Unconnected series*
Connected series*
Constant
Summary
Dataflow edge
Control dependency edge
Reference edge
Weights
Bias
SGD
2 tensors
7 tensors
Loss
Accuracy
4 tensors
LabelData
Weights
Bias
Model
InputData
Weights
init
Model
SGD
Bias
init
Model
SGD
Weights
Bias
init

THANK YOU

Q & A