머신러닝

장경배

https://youtu.be/6390S8Eeu6E

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

감독학습(Supervised Learning)

- 지도학습 이라고도 부름
- 명시적인 정답이 주어진 상태에서 컴퓨터를 학습시키는 방법
- 입력 데이터와 출력 데이터를 나타내는 하나의 데이터 셋(Data Set) 을 이용
- 새로운 입력데이터에 대하여 출력데이터를 예측

감독학습(Supervised Learning)

Relationship

“Dog”

Relationship

“Cat”

Input

정답

감독학습(Supervised Learning)

- 그래프상으로 표현했을 때 **x축은 데이터**
  은 데이터와 맵핑되는 **y축은 레이블**
- 지도학습은 **분류**(Classification)와 **회귀**(Regression)로 나뉨

X(공부에 투자한 시간)	Y(시험점수)
10	PASS
9	FAIL
3	PASS
2	FAIL

분류

X(공부에 투자한 시간)	Y(시험점수)
10	90
9	80
3	50
2	30

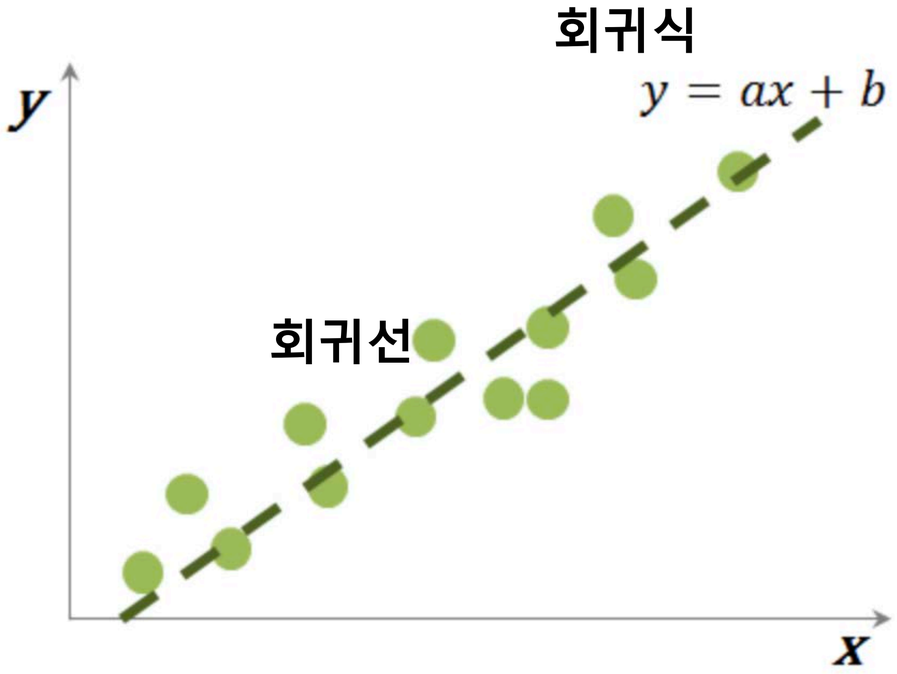
회귀

선형회귀(Linear Regression)

- 주어진 데이터를 표현하는 하나의 직선을 찾는 것
- 예제의 input이 되는 변수는 한개
- 회귀식을 변형해가며 관측값과 예측값의 차이를 최소로하는 과정 반복
  - (1,4)라는 점에 대하여 y=2x+1 회귀식은 (1,3)을 예측
  - 관측값 = 4, 예측값 = 3

X(공부에 투자한 시간)	Y(시험점수)
10	90
9	80
3	50
2	30

회귀

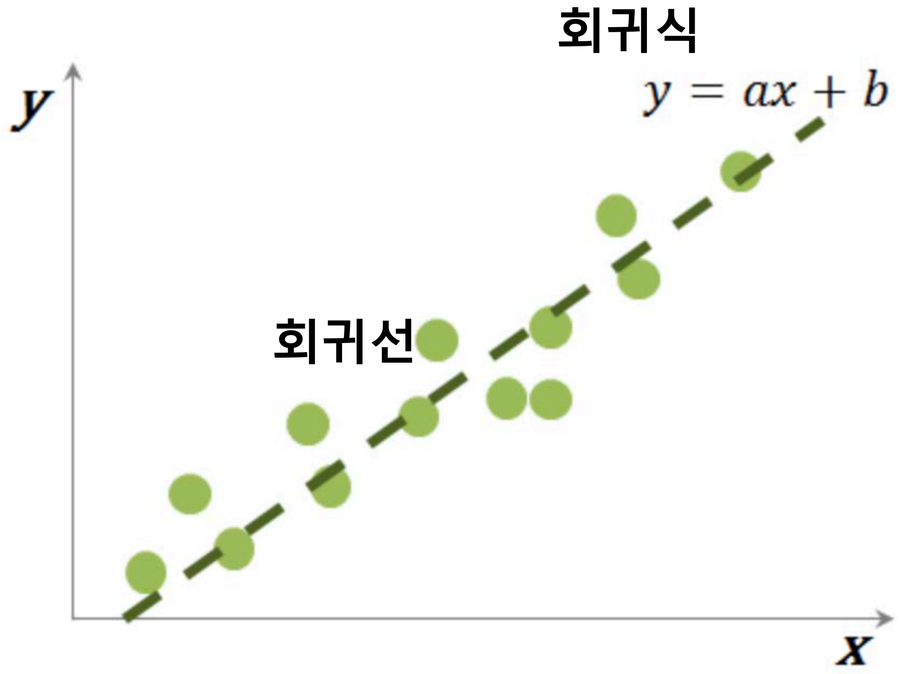


선형회귀(Linear Regression)

- a는 x가 y에 얼마나 영향을 주는지 그 크기와 방향을 알 수 있는 **가중치** 역할
- b는 회귀선을 위 아래로 평행 이동시키는 **절편**의 역할

X(공부에 투자한 시간)	Y(시험점수)
10	90
9	80
3	50
2	30

회귀

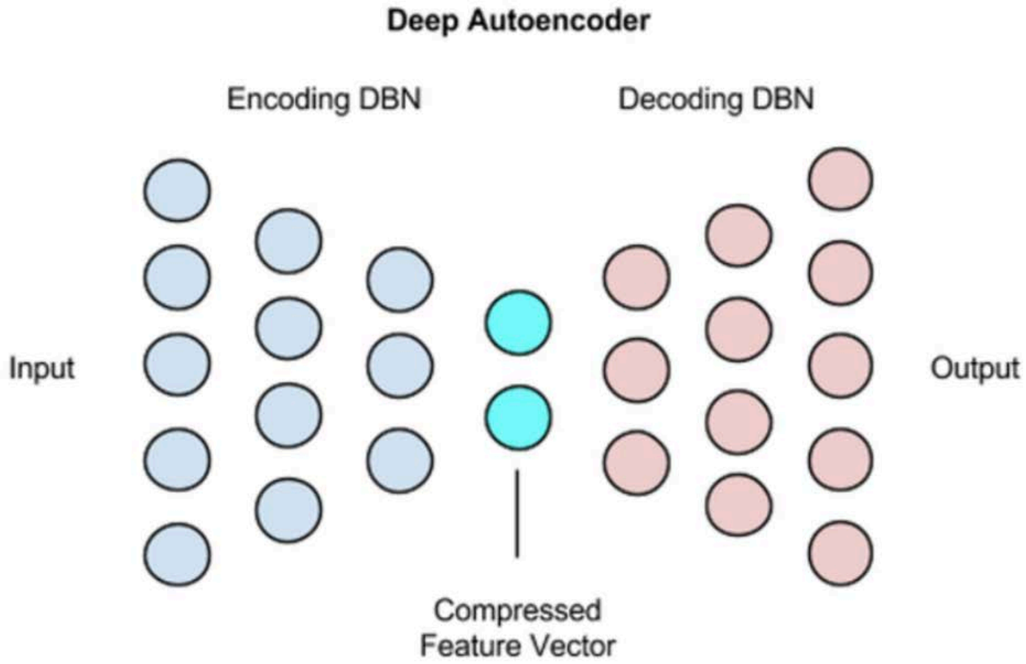


비감독학습(UnSupervised Learning)

- 비지도학습 이라고도 부름
- 명시적인 정답이 주어지지 않은 상태에서 컴퓨터를 학습시키는 방법
- 데이터와 레이블이 아닌 데이터만 사용하여 학습을 진행
- 데이터의 숨겨진 특징이나 구조를 발견하는데 사용

오토인코더(AutoEncoder)

- 딥러닝분야에서 사용되는 비지도학습 방법
- **인코더**와 **디코더**의 개념을 사용
- 주어진 데이터(Input)에 대하여 인코더를 사용하여 데이터를 압축
- 인코딩된 데이터에 디코더를 사용하여 입력과 같은 사이즈의 데이터를 출력



오토인코더(AutoEncoder)

- 입력값과 출력값이 최대한 같아지도록 튜닝함으로써 좋은 Feature를 잘 추출할 수 있게하는 것이 원리
- 자신의 설정한 오차(임계값)를 만족하면 모델 생성완료
- 만들어진 모델에 기존데이터와 다른 데이터가 들어오면 큰 오차가 발생
  - 이상치 탐지에 많은 활용이 됨

실습

실습

```
(base) Bigbossui-MacBookPro:~ kb$ ipython
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

터미널에서 작업대신

`kb$ jupyter notebook` 토큰을 생성하여 코드를 브라우저에서 실행 가능하게 해줌

선형회귀 분석(Linear Regression Analysis)

jupyter Untitled Last Checkpoint: 2시간 전 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [9]:

```
import tensorflow as tf

x_data = [1, 2, 3]
y_data = [1, 2, 3]

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))


X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")
print(X)
print(Y)

hypothesis = W * X + b

cost = tf.reduce_mean(tf.square(hypothesis - Y))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)

train_op = optimizer.minimize(cost)


with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())


    for step in range(100):

        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})

        print(step, cost_val, sess.run(W), sess.run(b))


    print("\n=== Test ===")
    print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
    print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X: 2.5}))
```

선형회귀 분석

```
Tensor("X_9:0", dtype=float32)
Tensor("Y_9:0", dtype=float32)
0 0.047728285 [0.96339947] [0.10794754]
1 0.0021003813 [0.9543809] [0.10099825]
2 0.0014826622 [0.9565595] [0.09904623]
3 0.00140604 [0.9574855] [0.0966132]
4 0.0013391833 [0.9585204] [0.09429635]
5 0.0012755672 [0.9595161] [0.0920289]
6 0.0012149793 [0.9604895] [0.08981667]
7 0.001157268 [0.9614393] [0.08765754]
8 0.0011022972 [0.9623663] [0.08555032]
9 0.0010499391 [0.96327096] [0.08349373]
10 0.0010000636 [0.9641539] [0.08148659]
11 0.00095255533 [0.9650156] [0.0795277]
12 0.0009073135 [0.9658567] [0.07761593]
13 0.0008642121 [0.96667737] [0.07575006]
14 0.00082316203 [0.96747845] [0.07392911]
15 0.0007840646 [0.9682603] [0.07215191]
16 0.000746818 [0.9690233] [0.07041741]
```

학습 과정

```
83 0.0009242461 [0.96553946] [0.07833686]
84 0.0008803411 [0.96636784] [0.0764537]
85 0.0008385239 [0.9671764] [0.07461582]
86 0.00079869694 [0.9679655] [0.07282212]
87 0.0007607581 [0.9687356] [0.07107151]
88 0.0007246205 [0.9694871] [0.06936298]
89 0.00069020333 [0.9702207] [0.06769557]
90 0.0006574139 [0.9709364] [0.06606817]
91 0.0006261858 [0.97163516] [0.06447995]
92 0.00059644174 [0.972317] [0.06292988]
93 0.00056811125 [0.9729825] [0.06141712]
94 0.00054112397 [0.973632] [0.05994068]
95 0.00051542145 [0.9742659] [0.05849975]
96 0.0004909371 [0.97488445] [0.05709345]
97 0.00046762067 [0.97548825] [0.05572098]
98 0.00044540525 [0.9760775] [0.05438149]
99 0.00042425058 [0.9766526] [0.0530742]
```

가중치 절편

테스트

```
=== Test ===
X: 5, Y: [4.9363375]
X: 2.5, Y: [2.4947057]
```

선형회귀 분석

```python
import tensorflow as tf

x_data = [10,9,3,2]
y_data = [90,80,50,30]

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))


X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")
print(X)
print(Y)

hypothesis = W * X + b

cost = tf.reduce_mean(tf.square(hypothesis - Y))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.02)

train_op = optimizer.minimize(cost)


with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())


    for step in range(1000):

        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})

        print(step, cost_val, sess.run(W), sess.run(b))


    print("\n=== Test ===")
    print("X: 10, Y:", sess.run(hypothesis, feed_dict={X: 10}))
    print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
    print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X: 2.5}))
```

X(공부에 투자한 시간)	Y(시험점수)
10	90
9	80
3	50
2	30

```
996 24.249992 [6.600106?] [22.89914]
997 24.25 [6.6001067] [22.899149]
998 24.249992 [6.600104] [22.899157]
999 24.25 [6.600105] [22.899166]

=== Test ===
X: 10, Y: [88.90021]
X: 5, Y: [55.89969]
X: 2.5, Y: [39.39943]
```

5시간공부하면 점수가 얼마나 나올까?

오토인코더(AutoEncoder)

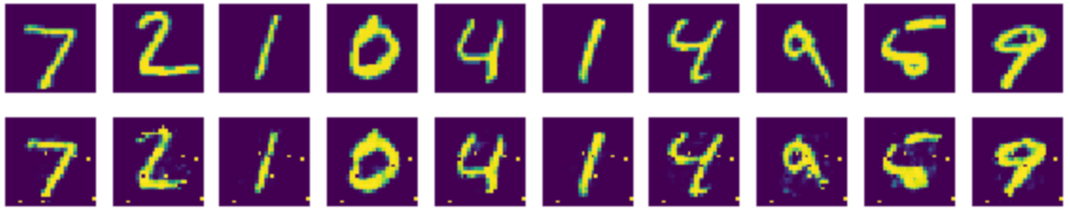
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

learning_rate = 0.01
training_epoch = 20
batch_size = 100

n_hidden = 256  # 히든 레이어의 뉴런 갯수
n_input = 28*28   # 입력값 크기 - 이미지 픽셀수

#########

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

W_encode = tf.Variable(tf.random_normal([n_input, n_hidden]))
b_encode = tf.Variable(tf.random_normal([n_hidden]))

encoder = tf.nn.sigmoid(
                tf.add(tf.matmul(X, W_encode), b_encode))

W_decode = tf.Variable(tf.random_normal([n_hidden, n_input]))
b_decode = tf.Variable(tf.random_normal([n_input]))

decoder = tf.nn.sigmoid(
                tf.add(tf.matmul(encoder, W_decode), b_decode))

cost = tf.reduce_mean(tf.pow(X - decoder, 2))
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

```
total_batch = int(mnist.train.num_examples/batch_size)

for epoch in range(training_epoch):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.4f}'.format(total_cost / total_batch))

print('최적화 완료')
```

```
sample_size = 10

samples = sess.run(decoder,
                   feed_dict={X: mnist.test.images[:sample_size]})

fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))

for i in range(sample_size):
    ax[0][i].set_axis_off()
    ax[1][i].set_axis_off()
    ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
    ax[1][i].imshow(np.reshape(samples[i], (28, 28)))

plt.show()
```

오토인코더

```
Extracting ./mnist/data/train-images-idx3-ubyte.gz
Extracting ./mnist/data/train-labels-idx1-ubyte.gz
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
Epoch: 0001 Avg. cost = 0.1921
Epoch: 0002 Avg. cost = 0.0555
Epoch: 0003 Avg. cost = 0.0443
Epoch: 0004 Avg. cost = 0.0402
Epoch: 0005 Avg. cost = 0.0378
Epoch: 0006 Avg. cost = 0.0364
Epoch: 0007 Avg. cost = 0.0341
Epoch: 0008 Avg. cost = 0.0325
Epoch: 0009 Avg. cost = 0.0310
Epoch: 0010 Avg. cost = 0.0286
Epoch: 0011 Avg. cost = 0.0270
Epoch: 0012 Avg. cost = 0.0265
Epoch: 0013 Avg. cost = 0.0252
Epoch: 0014 Avg. cost = 0.0246
Epoch: 0015 Avg. cost = 0.0242
Epoch: 0016 Avg. cost = 0.0233
Epoch: 0017 Avg. cost = 0.0227
Epoch: 0018 Avg. cost = 0.0224
Epoch: 0019 Avg. cost = 0.0222
Epoch: 0020 Avg. cost = 0.0220
최적화 완료
```



결과화면

감사합니다