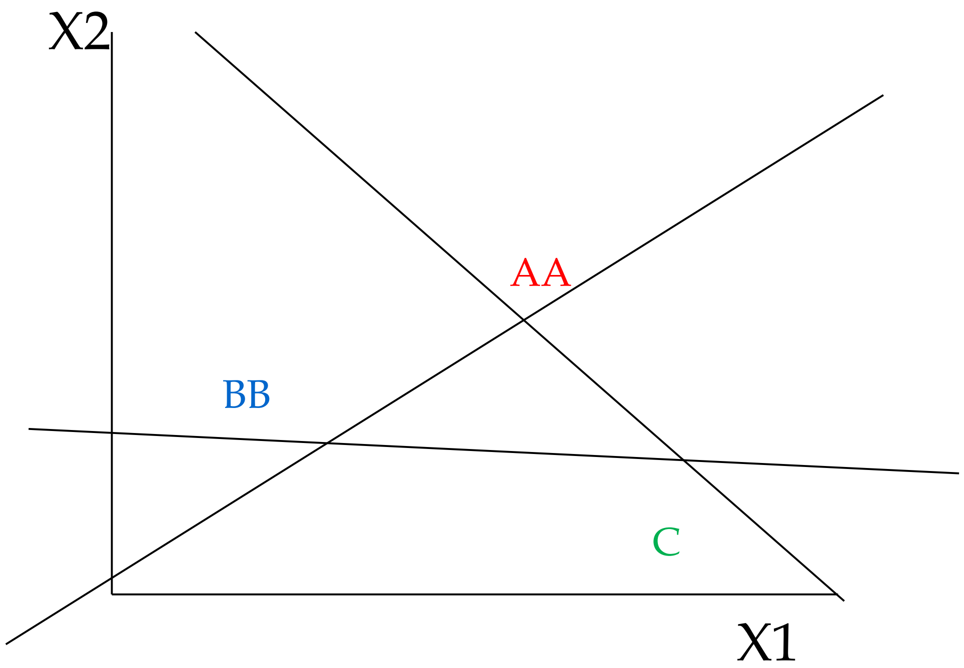
메디치소프트 기술연구소

딥러닝

Multinomial regression

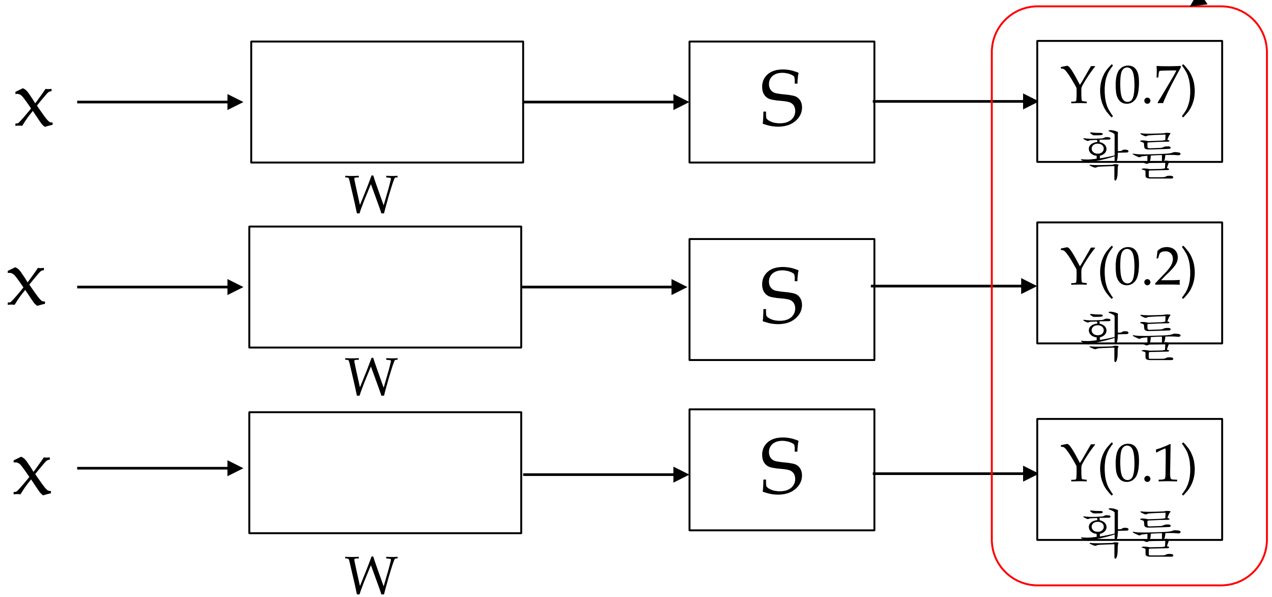
x1 (hours)	x2 (attendance)	y (grade)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C



Multinomial regression

$$\begin{bmatrix} W1 & W2 & W3 \\ W1 & W2 & W3 \\ W1 & W2 & W3 \end{bmatrix} \begin{pmatrix} X1 \\ X2 \\ X3 \end{pmatrix} = \begin{pmatrix} W1X1+W2X2+W3X3 \\ W1X1+W2X2+W3X3 \\ W1X1+W2X2+W3X3 \end{pmatrix} = \begin{matrix} Y1(2.0) \\ Y2(1.0) \\ Y3(0.1) \end{matrix}$$

(3,3) (3,1) (3,1)



softmax는 점수로 나온 결과를 전체 합계가 1이 되
0과 1 사이의 값으로 변경

Softmax Classification& Cross entropy

$$y = softmax(Wx + b)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

. Softmax regression 모델 학습 평가 Evaluation function은 cross-entropy

$$H_{y'}(y) = -\sum_i y'_i log(y_i)$$

. Cross-entropy: 모델의 예측값(prediction)이 실제 참값(truth)설명하는데 얼마나 비효율적(inefficient)를 나타낸다.
. Coss-entropy가 최소화 => 경사하강법(gradient descent)방법 이용

Softmax Classification& Cross entropy

. Softmax

- Binary Classification 보다는 n개의 예측 사용

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

Logistic Classifier

$XW = y$

$$\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \quad f(\vec{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \quad \text{for } i = 1, \ldots, K \quad \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$$

Score(logits) Softmax Probabilities

. Cross Entropy

- 신경망 출력 확률 간주할 수 있는 경우 사용(손실함수)

```
# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

Softmax Classification& Cross entropy(예시)

```
x_data = [[1, 2, 1, 1], [2, 1, 3, 2], [3, 1, 3, 4], [4, 1, 5, 5], [1, 7, 5, 5],
                                                      [1, 2, 5, 6], [1, 6, 6, 6], [1, 7, 7, 7]]
y_data = [[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]]

X = tf.placeholder("float", [None, 4])
Y = tf.placeholder("float", [None, 3])
nb_classes = 3

W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Launch graph
with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())

   for step in range(2001):
       sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
       if step % 200 == 0:
           print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))
```

1.0
0.0
0.0
One-hot encoding

Test& one-hot encoding

```python
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

```python
# Testing & One-hot encoding
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
print(a, sess.run(tf.arg_max(a, 1)))
```

Argmax 가장 높은 값(첫번째 acces기준)

[[1.38904958e-03 9.98601854e-01 9.06129117e-06]] [1]

Softmax를 통과한 확률값(전체 합은 1)

```python
all = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9],
                                          [1, 3, 4, 3],
                                          [1, 1, 0, 1]]})
print(all, sess.run(tf.arg_max(all, 1)))
```

[[1.38904958e-03 9.98601854e-01 9.06129117e-06] [1]
[9.31192040e-01 6.29020557e-02 5.90589503e-03] [0]
[1.27327668e-08 3.34112905e-04 9.99665856e-01]] [2]

softmax_cross_entropy with_logits

```
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)
```

One-hot

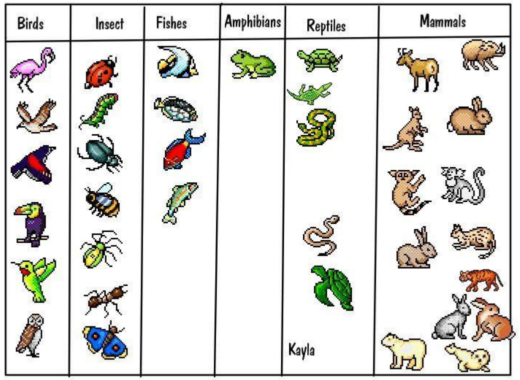
1

```
# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

2

```
# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                 labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
```

Animal classification(다리=> 동물예측)



1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

0~6 7가지

```python
# Predicting animal type based on various features
xy = np.loadtxt('zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

Tf.on_hot and reshape

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

[[1000000],[0001000]

0~6 한자리

```
Y = tf.placeholder(tf.int32, [None, 1])  # 0 ~ 6, shape=(?, 1)
Y_one_hot = tf.one_hot(Y, nb_classes)  # one hot shape=(?, 1, 7) 몇 개 클래스?
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes]) # shape=(-1, 7)
```

If the input indices is rank N, the output will have rank N+1. The new axis is created at dimension axis (default: the new axis is appended at the end.
(예 1차원=>2차원 변경)https://www.tensorflow.org/api_docs/python/tf/one_hot

MEDICISOFT

Tf.on_hot and reshape

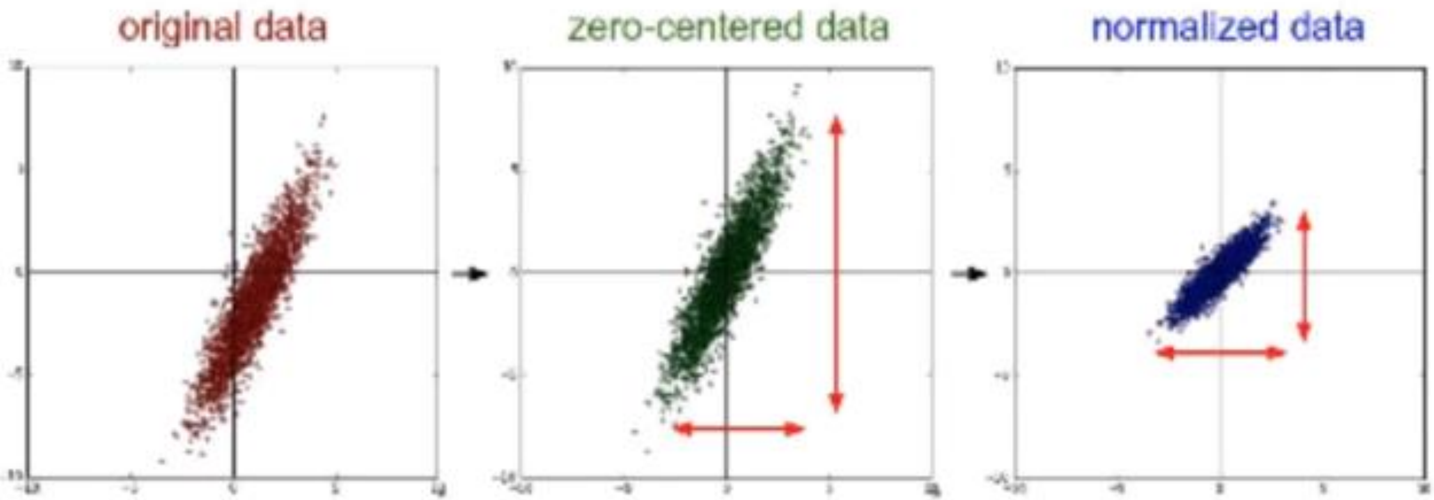
```
# Predicting animal type based on various features
xy = np.loadtxt('zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

nb_classes = 7  # 0 ~ 6

X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1])  # 0 ~ 6

Y_one_hot = tf.one_hot(Y, nb_classes)  # one hot
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])

W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                 labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

Tf.on_hot and reshape

```python
cost = tf.reduce_mean(cost_i)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

prediction = tf.argmax(hypothesis, 1) # 0~6사이의 값(확률)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
# Launch graph
with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())

   for step in range(2000):
       sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
       if step % 100 == 0:
           loss, acc = sess.run([cost, accuracy], feed_dict={
                                X: x_data, Y: y_data})
           print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(
               step, loss, acc))

   # Let's see if we can predict
   pred = sess.run(prediction, feed_dict={X: x_data})
   # y_data: (N,1) = flatten => (N, ) matches pred.shape
   for p, y in zip(pred, y_data.flatten()):    #[[1],[ㅔ]] -> [1, 0]
       print("[{}] Prediction: {} True Y: {}".format(p == int(y), p, int(y)))
```

```
Step:  1100 Loss: 0.097 Acc: 99.01%
Step:  1200 Loss: 0.089 Acc: 100.00%
Step:  1300 Loss: 0.082 Acc: 100.00%
Step:  1400 Loss: 0.076 Acc: 100.00%
Step:  1500 Loss: 0.071 Acc: 100.00%
Step:  1600 Loss: 0.067 Acc: 100.00%
Step:  1700 Loss: 0.063 Acc: 100.00%
Step:  1800 Loss: 0.059 Acc: 100.00%
Step:  1900 Loss: 0.056 Acc: 100.00%
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 3 True Y: 3
```

Data(X)preprocessing for gradient descent



$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

Standardization

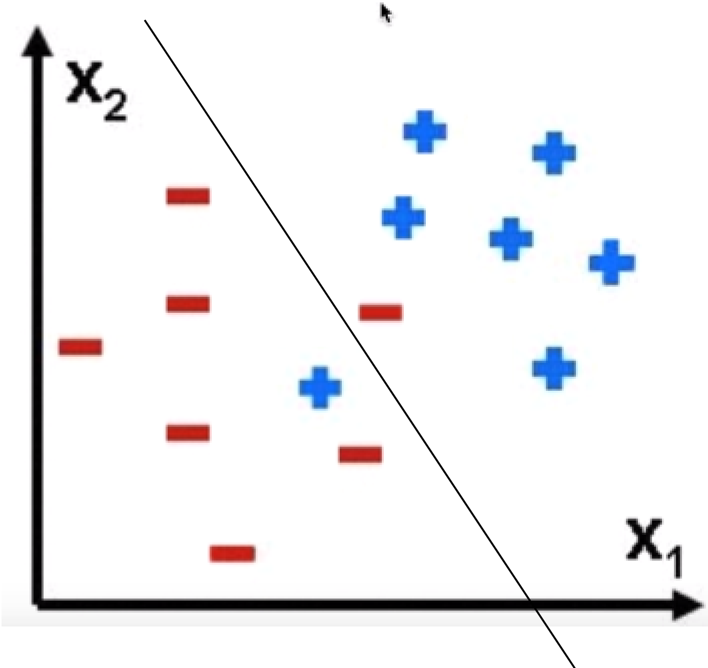
```
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
```

Overfitting

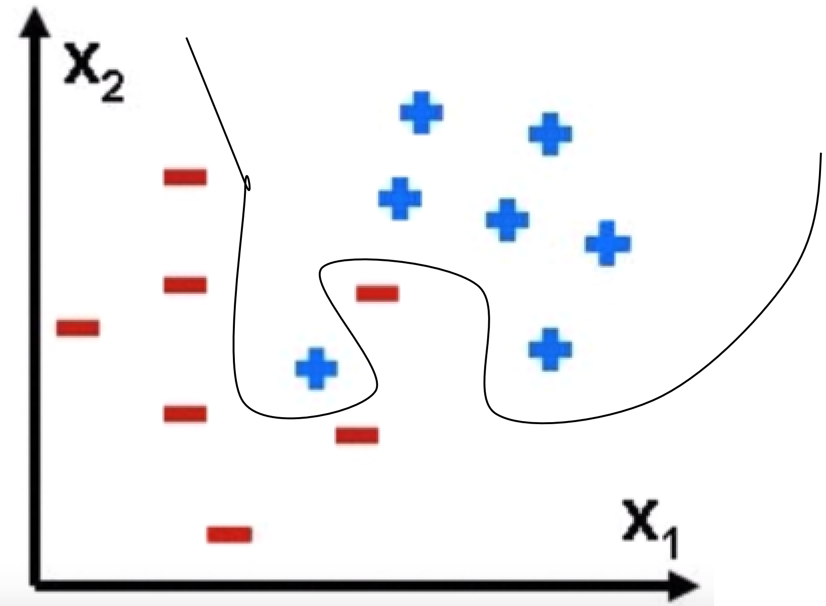
. Training data set에만 정확
. Test dataset이나 real data 부정확

→

. Training data 더 필요
. Features 개수 줄임
. Regularization 필요



Good



Regularization

. 정규화는 데이터를 구분하는 선이 지나치게 구불구불해지지 않게 막아주는 역할
. 최대한 직선에 가깝게 곡률을 줄여주는 역할을 하는 Factor를 Cost함수에 추가해서 그래프를 펴주는 역할

$$\mathscr{L} = \frac{1}{N} \sum_i D(S(wX_i + b), L_i) + \lambda \sum W^2$$

Loss(최소화)

Regularization Strength

```
l2reg = 0.001 * tf.reduce_sum(tf.square(W))
```


MEDICISOFT

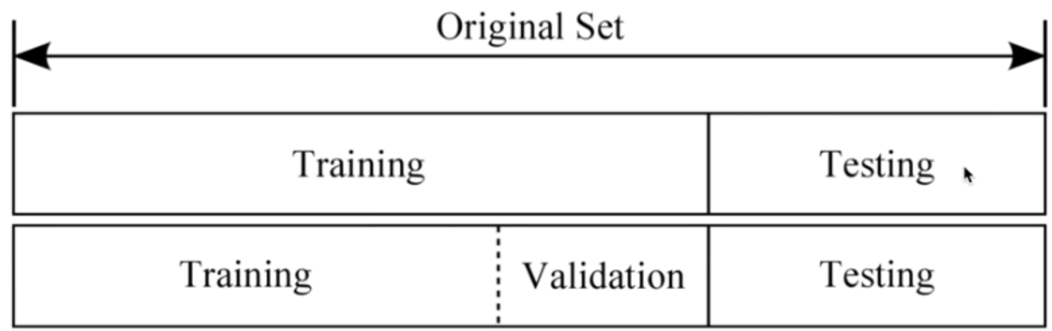
Evaluation using training set

Training set

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300

Test set

Size	Price
1534	315
1427	199
1380	212
1494	243



모의시험을
통한 튜닝

K-fold cross validation

.k개의 fold를 만들어서 진행하는 교차 검증이다
.데이터의 양이 충분치 않을 때, 분류기 성능 측정의 통계적 신뢰도를 높이기 위해서 쓰는 재 샘플링을 하는 기법 중 가장 대표적인 방법이다.
Machine learning을 할 때 사용자가 가지는 데이터는 오직 훈련집합과 테스트집합뿐이며, 집합 내에 샘플의 수가 무한정 제공 될 수 없기 때문에 사용하는 방법이다.

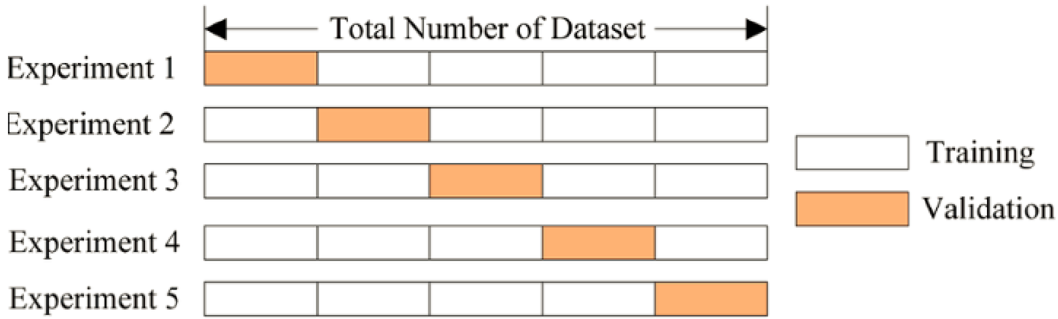


그림 5개의 fold로 나눴을 때의 검증

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i$$

크기로 나눈 다음에 하나의 부분씩을 test set으로 사용하여
나눈 다음에 하나의 부분씩을 test set으로 사용하여 k개의 test performance를 평균으로 내는 것을 의미한다 k개의 test performance를 평균으

K-fold cross validation

모든 기계 학습 분류 모델을 평가하여 데이터 세트를 테스트 데이터와 학습 데이터로 분분리하는 것 일반적이다. 100개의 데이터 세트가 있으면 20개의 행을 테스트 데이터로 사용하고 나머지 80개의 행은 학습 데이터로 사용한다. 다음 예제에서 사용하는 데이터 세트는 iris_data.csv파일을 사용한다. 꽃잎의 너비, 꽃잎이 길이, 꽃잎 길이와 꽃잎 너비를 사용하여 식물의 종류를 예측하려고 할 것이다. 일반적인 80:20 분할을 통해 데이터 집합을 분할하여 시작한다.

	A	B	C	D	E	F
99	6.2	2.9	4.3	1.3	Iris-versicolor	
100	5.1	2.5	3	1.1	Iris-versicolor	
101	5.7	2.8	4.1	1.3	Iris-versicolor	
102	6.3	3.3	6	2.5	Iris-virginica	
103	5.8	2.7	5.1	1.9	Iris-virginica	
104	7.1	3	5.9	2.1	Iris-virginica	
105	6.3	2.9	5.6	1.8	Iris-virginica	
106	6.5	3	5.8	2.2	Iris-virginica	
107	7.6	3	6.6	2.1	Iris-virginica	
108	4.9	2.5	4.5	1.7	Iris-virginica	
109	7.3	2.9	6.3	1.8	Iris-virginica	
110	6.7	2.5	5.8	1.8	Iris-virginica	
111	7.2	3.6	6.1	2.5	Iris-virginica	
112	6.5	3.2	5.1	2	Iris-virginica	
113	6.4	2.7	5.3	1.9	Iris-virginica	
114	6.8	3	5.5	2.1	Iris-virginica	
115	5.7	2.5	5	2	Iris-virginica	
116	5.8	2.8	5.1	2.4	Iris-virginica	
117	6.4	3.2	5.3	2.3	Iris-virginica	
118	6.5	3	5.5	1.8	Iris-virginica	
119	7.7	3.8	6.7	2.2	Iris-virginica	
120	7.7	2.6	6.9	2.3	Iris-virginica	
121	6	2.2	5	1.5	Iris-virginica	
122	6.9	3.2	5.7	2.3	Iris-virginica	
123	5.6	2.8	4.9	2	Iris-virginica	
124	7.7	2.8	6.7	2	Iris-virginica	
125	6.3	2.7	4.9	1.8	Iris-virginica	
126	6.7	3.3	5.7	2.1	Iris-virginica	
127	7.2	3.2	6	1.8	Iris-virginica	

Training Set

Partition

Testing Set

K-fold cross validation

. 장점:총 데이터의 개수가 적은 데이터 셋에 대하여 정확도를 향상시킬 수 있다.
이는 기존에 training/validation/test 세 개의 집단으로 분류하는 것보다,
Training과 test로만 분류할 때 학습 데이터 셋이 더 많기 때문이다

. 단점:일반적인 training set/test set을 통해 진행하는 학습법에 비해 시간 소요가 크다.

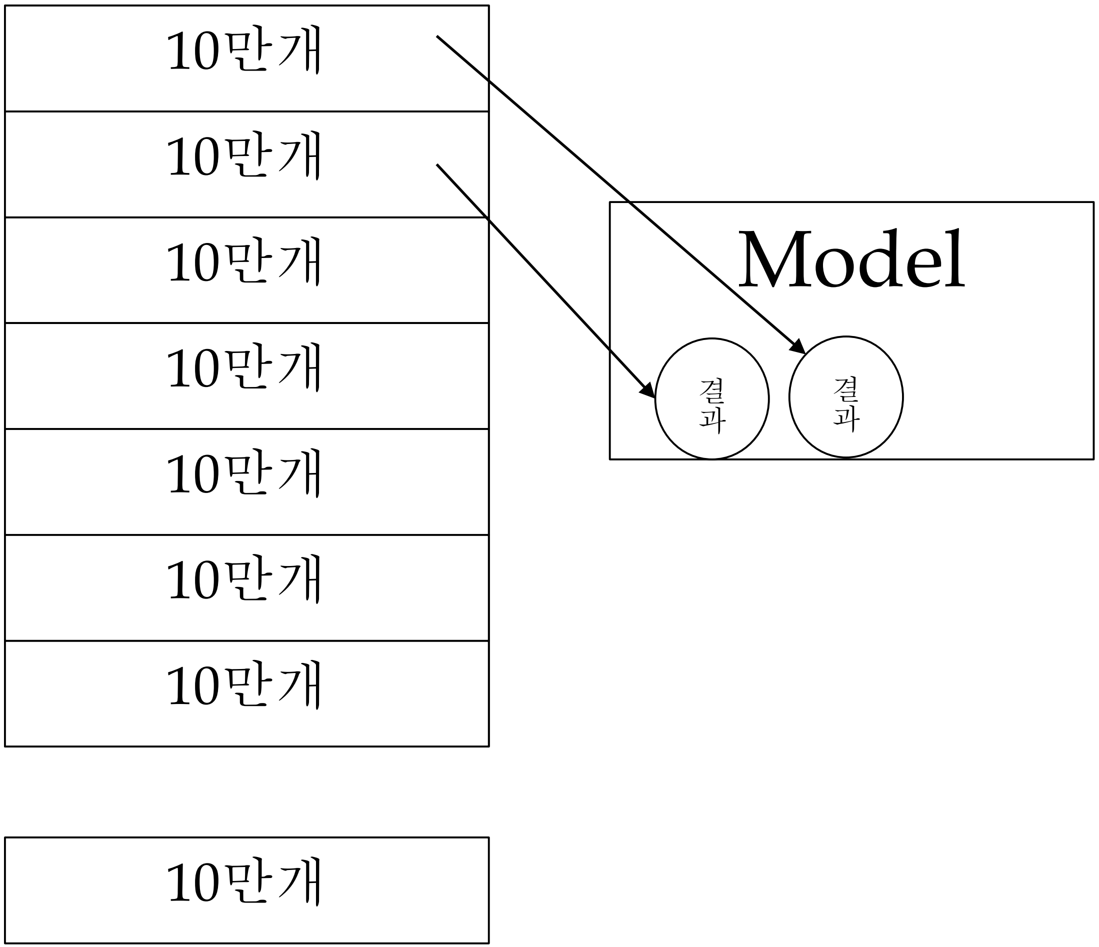
- 예제1(cross_validation_example.py)
  . 적절한 계층화 된 k-Folds cross validation

```
n cross_validation_example.py
Accuracy: 0.933333333333
Accuracy: 0.933333333333
Accuracy: 1.0
Accuracy: 0.933333333333
Accuracy: 0.933333333333
Accuracy: 0.933333333333
Accuracy: 0.866666666667
Accuracy: 1.0
Accuracy: 1.0
Accuracy: 1.0
```

- 예제2(cross_validation_example2.py)
  . 모든 폴드의 결과를 final_red_y 및 expected_y 목록으로 결합하여 비교할 수 있는 분류 기준 정확도의 척도를 얻는다.

```
n cross_validation_example2.py
Accuracy: 0.953333333333
```

Online learning(대용량 데이터)



Training and Test datasets

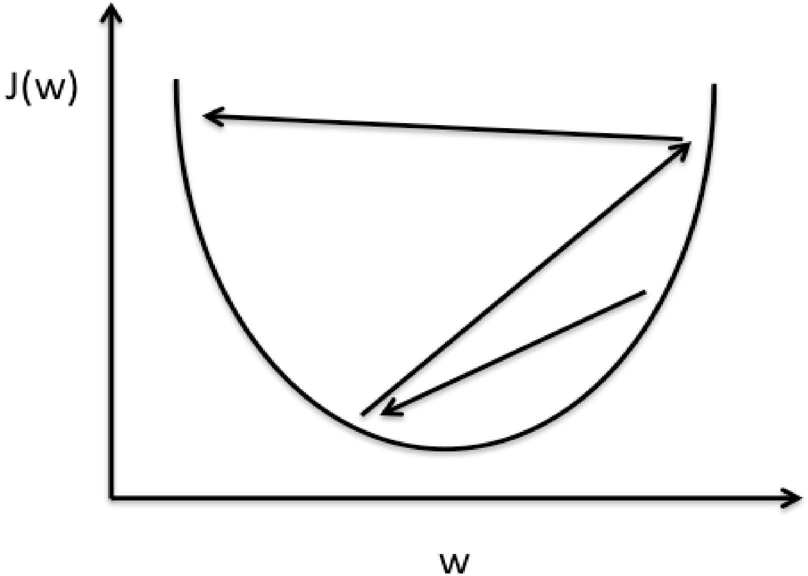
```python
x_data = [[1, 2, 1], [1, 3, 2], [1, 3, 4], [1, 5, 5], [1, 7, 5], [1, 2, 5], [1, 6, 6], [1, 7, 7]]
y_data = [[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]]

# Evaluation our model using this test dataset
x_test = [[2, 1, 1], [3, 1, 2], [3, 3, 4]]
y_test = [[0, 0, 1], [0, 0, 1], [0, 0, 1]]
```

Training and Test datasets

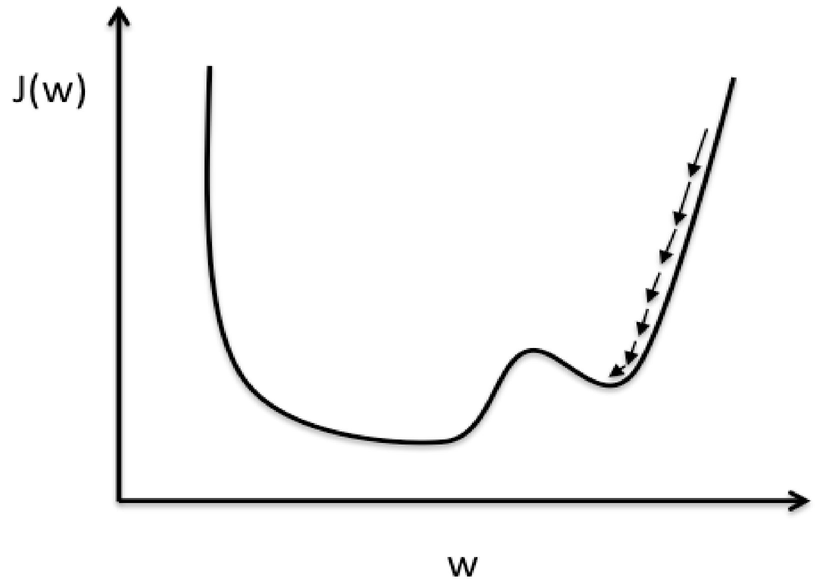
```
X = tf.placeholder("float", [None, 3])
Y = tf.placeholder("float", [None, 3])
W = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))

hypothesis = tf.nn.softmax(tf.matmul(X, W)+b)
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Correct prediction Test model
prediction = tf.arg_max(hypothesis, 1) #예측
Is_correct = tf.equal(prediction, tf.arg_max(Y, 1))
Accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# Launch graph
With tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())
   for step in range(201):
       cost_val, W_val, _ = sess.run([cost, W, optimizer],
                       feed dict={X: x data, Y: y_data}) #train
       print(step, cost_val, W_val)   #학습 끝

   # predict
   print( " Prediction: " , sess.run(prediction, feed_dict={X: x_test}))
   # Calculate the accuracy
   print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

```
199 0.672261 [[-1.15377033  0.28146935
1.13632679]
[ 0.37484586  0.18958236  0.33544877]
[-0.35609841 -0.43973011 -1.25604188]]
200 0.670909 [[-1.15885413  0.28058422
1.14229572]
[ 0.37609792  0.19073224  0.33304682]
[-0.35536593 -0.44033223 -1.2561723 ]]
Prediction: [2 2 2]
Accuracy:  1.0
```

Learning rate (0.1/0.01)



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

Learning rate (0.1/0.01)

```
X = tf.placeholder("float", [None, 3])
Y = tf.placeholder("float", [None, 3])
W = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))
hypothesis = tf.nn.softmax(tf.matmul(X, W)+b)
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer
                (learning_rate=1.5).minimize(cost)
```

Big learning rate(1.5) : cost값이 변동 확인

Small learning rate(1e-10): cost값 동일 확인

```
# Correct prediction Test model
prediction = tf.arg_max(hypothesis, 1)
is_correct = tf.equal(prediction, tf.arg_max(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# Launch graph
with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())
   for step in range(201):
       cost_val, W_val, _ = sess.run([cost, W, optimizer],
                        feed_dict={X: x_data, Y: y_data})
       print(step, cost_val, W_val)

   # predict
   print("Prediction:", sess.run(prediction, feed_dict={X: x_test}))
   # Calculate the accuracy
   print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

Non-normalized inputs

```
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```

Non-normalized inputs

```
xy=...
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([4, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = tf.matmul(X, W) + b
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

sess = tf.Session()
sess.run(tf.global_variables_initializer())
for step in range(2001):
   cost_val, hy_val, _ = sess.run(
       [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
   print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

```
xy = MinMaxScaler(xy)
print(xy)
```

Non-normalized inputs(min-max scale)

```python
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```

```python
xy = MinMaxScaler(xy)
print(xy)
```

```
[[ 0.99999999  0.99999999  0.          1.          1.        ]
 [ 0.70548491  0.70439552  1.          0.71881782  0.83755791]
 [ 0.54412549  0.50274824  0.57608696  0.606468    0.6606331 ]
 [ 0.33890353  0.31368023  0.10869565  0.45989134  0.43800918]
 [ 0.51436     0.42582389  0.30434783  0.58504805  0.42624401]
 [ 0.49556179  0.42582389  0.31521739  0.48131134  0.49276137]
 [ 0.11436064  0.          0.20652174  0.22007776  0.18597238]
 [ 0.          0.07747099  0.5326087   0.          0.        ]]
```

Non-normalized inputs

```
xy=...
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([4, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = tf.matmul(X, W) + b
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

sess = tf.Session()
sess.run(tf.global_variables_initializer())
for step in range(2001):
   cost_val, hy_val, _ = sess.run(
       [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
   print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

```
xy = MinMaxScaler(xy)
print(xy)
```


MEDICISOFT

Non-normalized inputs

```
def MinMaxScaler(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    # noise term prevents the zero division
    return numerator / (denominator + 1e-7)


xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])


  # very important. It does not work without it.
  xy = MinMaxScaler(xy)
  print(xy)
```

```
xy = MinMaxScaler(xy)
print(xy)
```


MEDICISOFT

MNIST Dataset

. 손으로 필기한 숫자 각각의 이미지가 정수로 표시되는 데이터베이스
머신러닝 알고리즘의 성능을 벤치마킹하는 데 사용되며 99.7% 이상의 정확도 달성
55000개의 학습 데이터와 10000개의 테스트 데이터, 5000개의 검증 데이터
학습 데이터는 28 X 28 사이즈에 총 784개의 픽셀로 이루어진 흑백 이미지
컴퓨터는 흑색 이미지를 표현할 때 픽셀의 숫자가 0에 가까울수록 검정색으로
255에 가까울수록 하얀색으로 표현

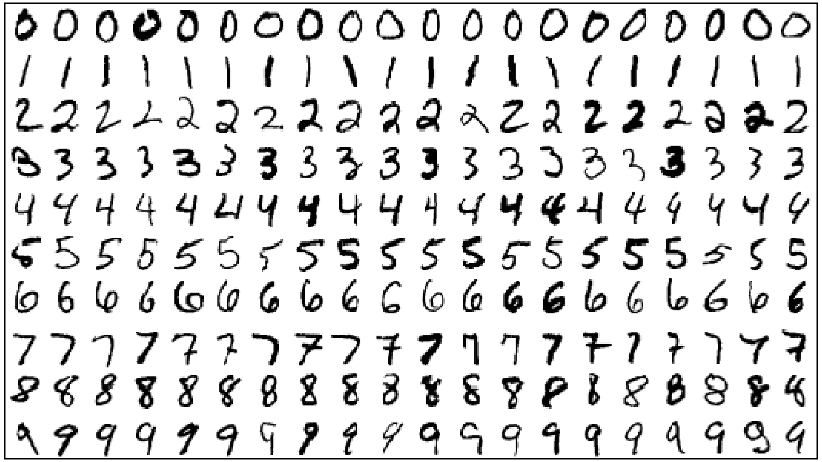
MNIST Dataset

```python
import matplotlib.pyplot as plt
import numpy as np

# Training Data 로드
data_file = open('./MNIST_data/mnist_train.csv', 'r')

# Training Data 파일의 내용을 한줄씩 불러와서 문자열 리스트로 반환
training_data = data_file.readlines()

# Training Data 의 두번째 데이터 확인
print(training_data[1])

# Training Data 의 두번째 데이터를 ','로 분리
training_data_array = np.asfarray(training_data[1].split(","))

# 일렬로 늘어진 784 개의 픽셀 정보를 28X28 행렬로 변환
matrix = training_data_array[1:].reshape(28,28)

# 회색으로 Training Data 의 두번째 데이터 숫자 확인
plt.imshow(matrix, cmap='gray')
plt.show()
```


MEDICISOFT

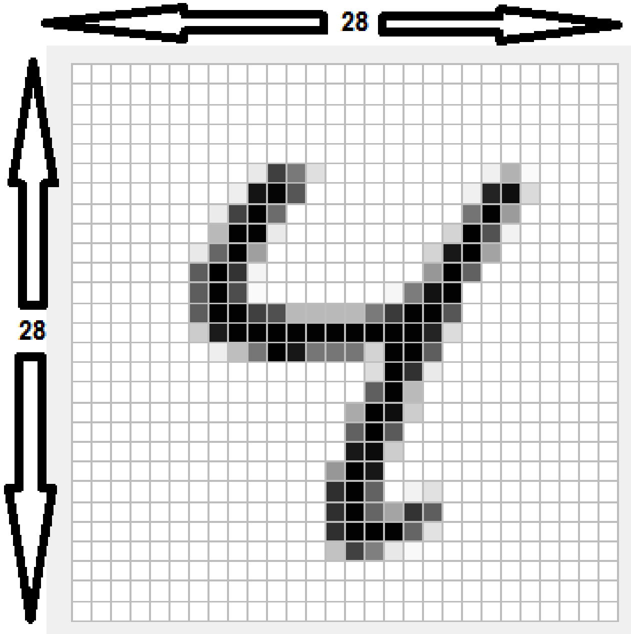
MNIST Dataset

. 데이터 준비: 머신러닝의 고전적인 문제: 필기 숫자들의 그레이 스케일 28*28 픽셀 이미지를 보고, 0부터 9시까지의 모든 숫자들에 대해 이미지가 어떤 숫자를 나타내는지 판별



파일	목적
train-images-idx3-ubyte.gz	학습 셋 이미지 - 55000개의 트레이닝 이미지, 5000개의 검증 이미지
train-labels-idx1-ubyte.gz	이미지와 매칭되는 학습 셋 레이블
t10k-images-idx3-ubyte.gz	테스트 셋 이미지 - 10000개의 이미지
t10k-labels-idx1-ubyte.gz	이미지와 매칭되는 테스트 셋 레이블

MINIST Dataset



28x28x1 image

```
# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])
```

MINIST Dataset

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
…
batch_xs, batch_ys = mnist.train.next_batch(100) #100개씩
…
print("Accuracy: ", accuracy.eval(session=sess,
           feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

Reading data and set variables

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))
```

Reading data and set variables

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))
```

MINIST Dataset

```
# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

Training epoch/batch

```
# parameters
training_epochs = 15
batch_size = 100

with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())
   # Training cycle
   for epoch in range(training_epochs):
       avg_cost = 0
       total_batch = int(mnist.train.num_examples / batch_size) #전체사이즈(10000)/100 =100번 돌면 1번

       for i in range(total_batch):
           batch_xs, batch_ys = mnist.train.next_batch(batch_size)#100개씩 돌면서 학습
           c, _ = sess.run([cost, optimizer], feed_dict={X: batch_xs, Y: batch_ys})
           avg_cost += c / total_batch

       print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
```

Training epoch/batch

In the neural network terminology:

- one **epoch** = one forward pass and one backward pass of *all* the training examples(전체)
- **Batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you ' ll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

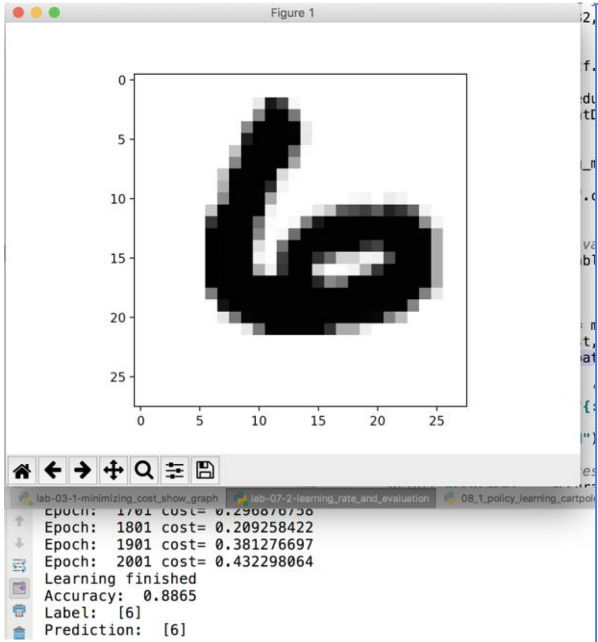
Report results on test dataset

```
# Test the model using test sets#sess.run()

print("Accuracy: ", accuracy.eval(session=sess,
 feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

MINIST Dataset

```
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
training_epochs = 15
batch_size = 100

with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())
   # Training cycle
   for epoch in range(training_epochs):
       avg_cost = 0
       total_batch = int(mnist.train.num_examples / batch_size)

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

       print('Epoch:', '%04d' % (epoch + 1),
                               'cost =', '{:.9f}'.format(avg_cost))
```

MINIST Dataset




```python
import matplotlib.pyplot as plt
import random

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label:", sess.run(tf.argmax(mnist.test.labels[r:r+1], 1)))
print("Prediction:", sess.run(tf.argmax(hypothesis, 1),
                              feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(mnist.test.images[r:r + 1].
          reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```