6 Feb 2017

Basic of ML : Regression and Classification

ISL lab Seminar

Han-Sol Kang

Contents

- Introduction
- Linear Regression
- Logistic Classification
- Softmax Classification
- Impleme ntation

Introduction

What is ML

- Limitations of explicit programming

- Spam filter : many rules

- Automatic driving : too many rules

- Machine learning : “Field of study that gives computers the ability to learn without being explicitly programmed” **- Arhur Samuel(1959)**

Introduction

Supervised/Unsupervised learning

- Supervised learning
  - Learning with labeled examples (training set)

- Unsupervised learning : un-labeled data
  - Google news grouping
  - Word clustering

Introduction

☆ Supervised learning

- Most common problem type in ML
  - **Image labeling :** learning from tagged images
  - **Email spam filter :** learning from labeled (spam or ham) email
  - **Predicting exam score :** learning from previous exam score and time spent

Introduction

- Supervised learning

- Type of supervised learning

  - Predicting final exam score based on time spent **– Regression**

  - Pass/non-pass based on time spent **– Binary classification**

  - Letter grade (A, B, C, D and F) based on time spent **– Multi-label classification**

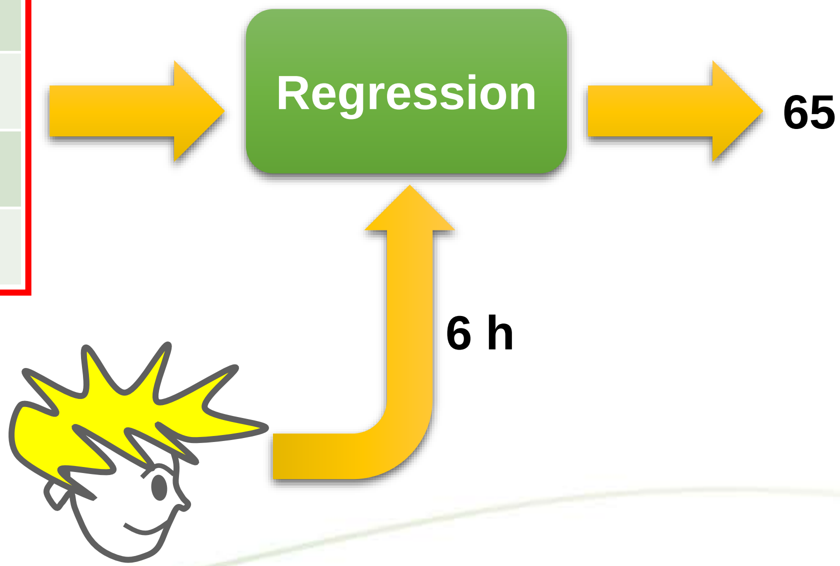
Linear regression

- Concept

Predicting final exam score based on time spent

X(hours)	Y(score)
10	90
9	80
3	50
2	30

Train

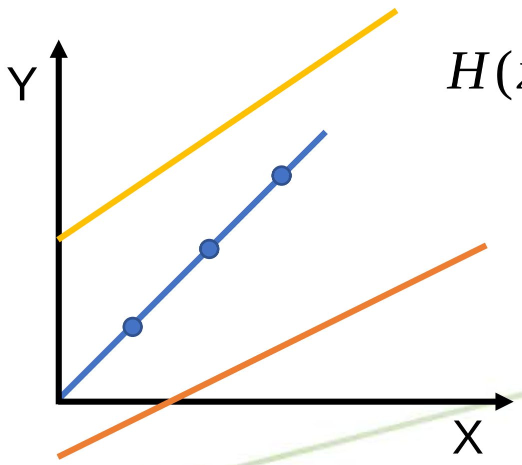


Linear regression

- Hypothesis and Cost

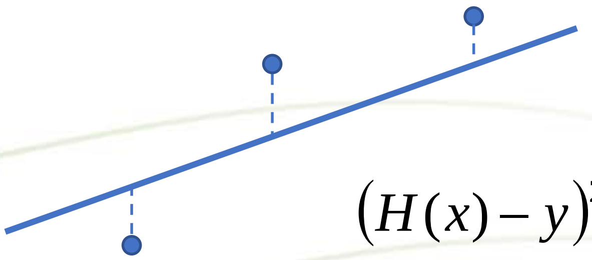
X	Y
1	1
2	2
3	3

(Linear)Hypothesis



$$H(x) = Wx + b$$

Cost



$$\left(H(x) - y\right)^2$$

Linear regression

- Cost function(Loss function)

X	Y
1	1
2	2
3	3

$$\frac{(H(x^{(1)})-y^{(1)})^2+(H(x^{(2)})-y^{(2)})^2+(H(x^{(3)})-y^{(3)})^2}{3}$$

$$\text{cost} = \frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)})-y^{(i)}\right)^2$$

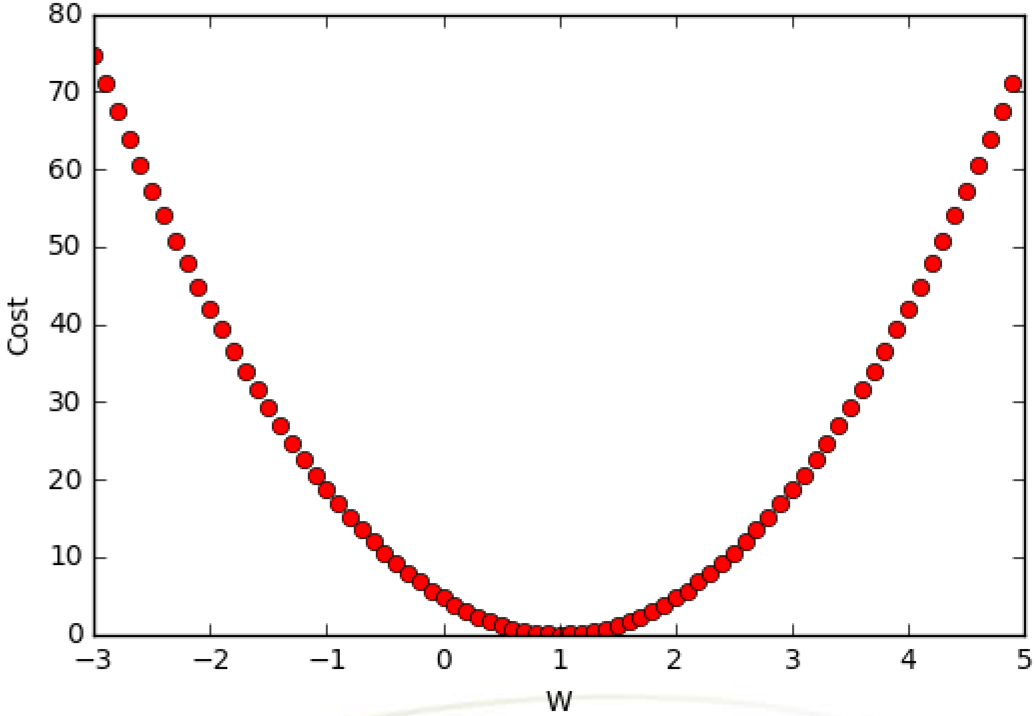
$$\text{cost}(\text{W},\text{b}) = \frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)})-y^{(i)}\right)^2$$

→ **Minimize**

Linear regression

- Gradient descent

X	Y
1	1
2	2
3	3



$$\text{cost}(W) = \frac{1}{m}\sum_{i=1}^{m}\left(Wx^{(i)} - y^{(i)}\right)^2$$

$$\text{cost}(W) = \frac{1}{2m}\sum_{i=1}^{m}\left(Wx^{(i)} - y^{(i)}\right)^2$$

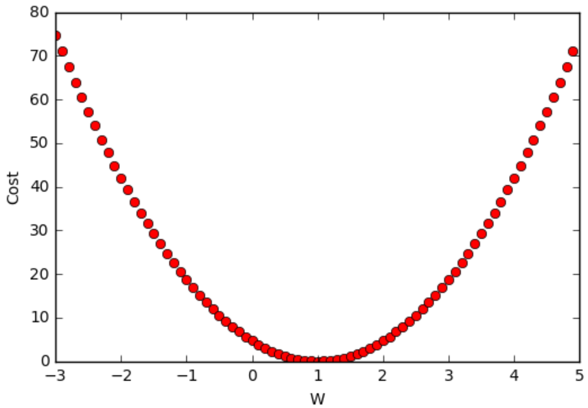
$$W := W - \alpha \frac{\partial}{\partial W}\text{cost}(W)$$

$$W := W - \alpha \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)}$$

Linear regression

- Gradient descent

X	Y
1	1
2	2
3	3



$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)}) x^{(i)}$$

$$W_0 = 0,\ \alpha = 0.1$$

$$W_1 = 0 - \frac{0.1}{3}(-1 - 4 - 9) = 0.433$$

$$W_2 = W_1 - \frac{0.1}{3}\{(0.433 - 1)1 + (0.866 - 2)2 + (1.299 - 3)3\} = 0.6976$$

$$W_3 = 0.83872$$

Linear regression

- Multi variable(feature)

X(hours)	Y(score)
10	90
9	80
3	50
2	60
11	40

X1(hours)	X2 (attendance)	Y(score)
10	5	90
9	5	80
3	2	50
2	4	60
11	1	40

$$H(x) = Wx + b$$

$$H(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

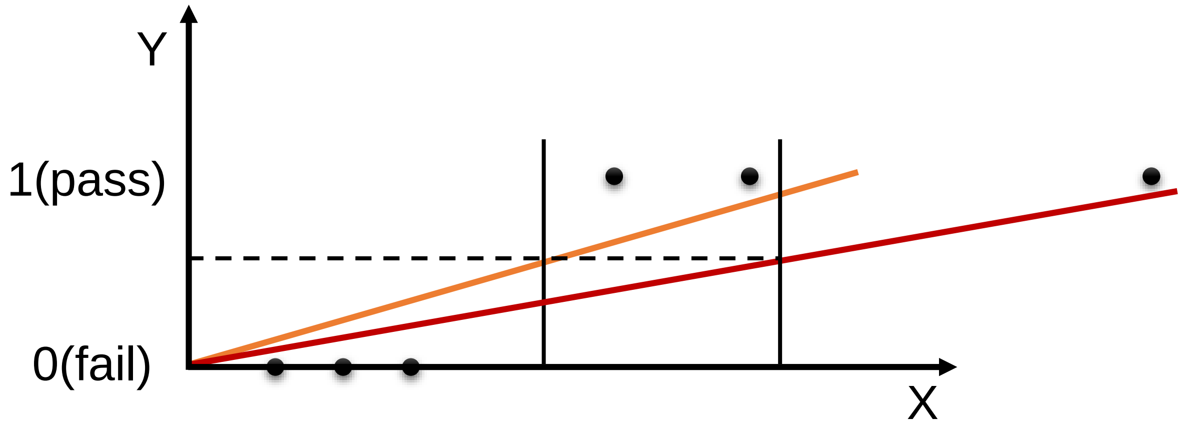
$$H(x_1, x_2, \cdots, x_n) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

$$\begin{bmatrix} b & w_1 & \cdots & w_n \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad H(X) = W^T X$$

Logistic Classification

- Concept

X(hours)	Y(P/F)
2	F
3	F
4	F
7	P
9	P



50	P

Logistic Classification

- Hypothesis

$$H(x) = wx + b$$

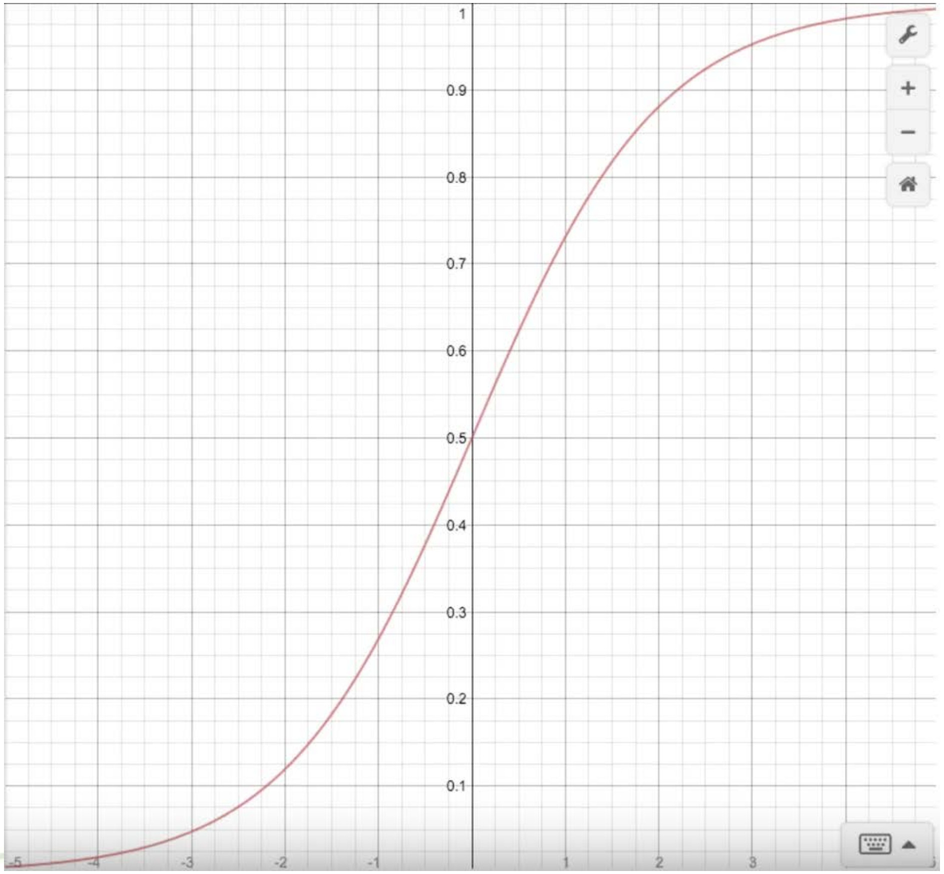
$$z = wx + b$$

$$g(z) = \frac{1}{(1+e^{-z})}$$

$$Z = W^T X$$

$$H(X) = g(Z)$$

$$H(X) = \frac{1}{1+e^{-(W^T X)}}$$



Sigmoid function
Logistic function

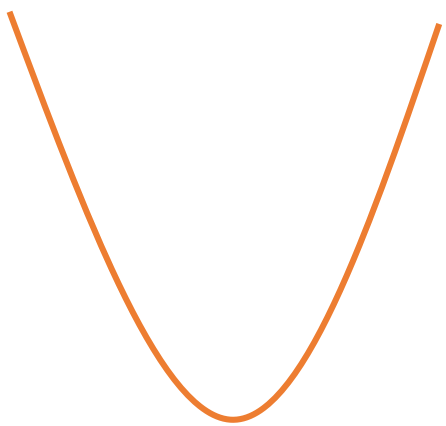
Logistic Classification

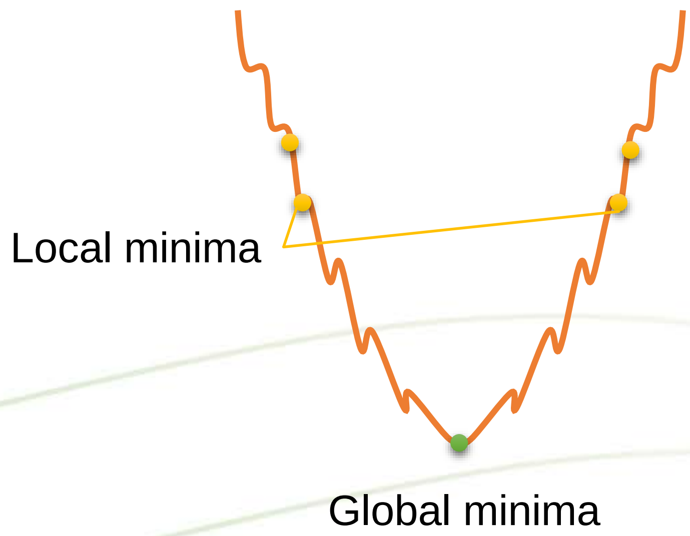
- Cost

$$\text{cost}(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)}) - y^{(i)}\right)^2$$

$$H(x) = Wx + b$$

$$H(X) = \frac{1}{1 + e^{W^T X}}$$





Logistic Classification

- Cost

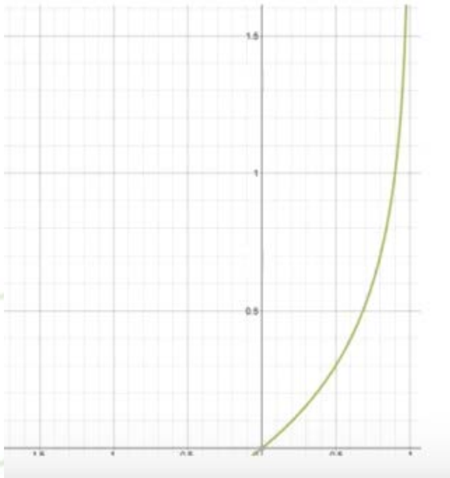
$$\text{cost}(W) = \frac{1}{m}\sum_{i=1}^{m} C\big(H(x), y\big)$$

$$C(H(x), y) = \begin{cases} -\log(H(x)) & (y=1) \\ -\log(1-H(x)) & (y=0) \end{cases}$$

$y=1$	$H(x)=1$	$\text{cost}=0$
	$H(x)=0$	$\text{cost}=\infty$

$y=0$	$H(x)=0$	$\text{cost}=0$
	$H(x)=1$	$\text{cost}=\infty$

$-\log(x)$



$-\log(1-x)$

$$C(H(x), y) = -y\log(H(x)) - (1-y)\log(1-H(x))$$

Logistic Classification

- Minimize cost

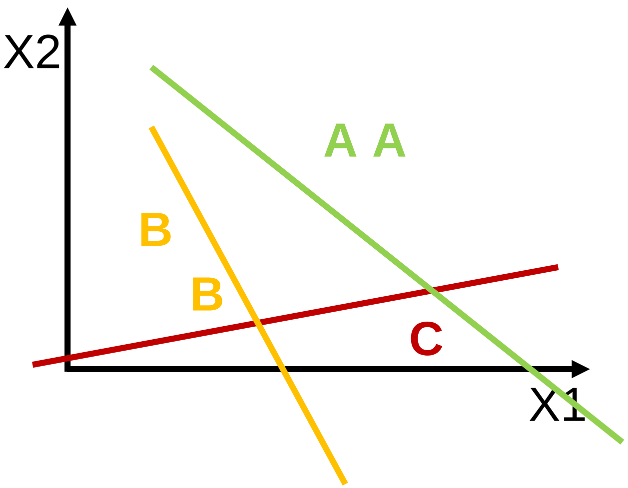
$$\text{cost}(W) = -\frac{1}{m}\sum_{i=1}^{m} y\log(H(x)) + (1-y)\log(1-H(x))$$

$$W := W - \alpha\frac{\partial}{\partial W}\text{cost}(W)$$

Softmax Classification

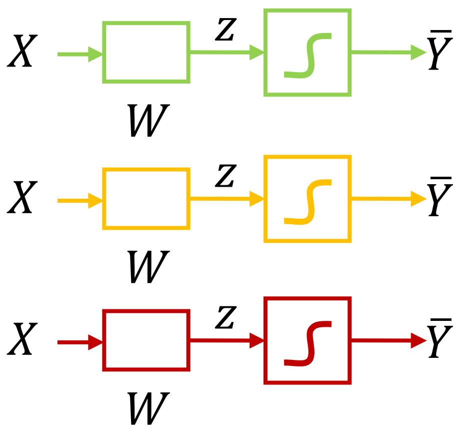
- Concept

X1(hours)	X2 (attendance)	Y(grade)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C



Softmax Classification

- Concept



$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B2} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix}$$

Softmax Classification

- Softmax

$$= \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

p=0.7 → A

p=0.2 → ~~B~~

p=0.1 → ~~C~~

1) 0~1

2) $\sum = 1$

Score

$$\begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \rightarrow \boxed{s(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}} \rightarrow$$

Probabilities

Probabilities	One-hot encoding		
0.7	→	1	A
0.2		0	~~B~~
0.1		0	~~C~~

Softmax Classification

- Cost (Cross entropy)

$S(y)=\bar{y}$	$L=y$
0.7	**1**
0.2	**0**
0.1	**0**

$$D(S,L) = -\sum_i L_i \log(S_i)$$

$$= -\sum_i L_i \log(\bar{y}_i)$$

$$= \sum_i (L_i) \times -\log(\bar{y}_i)$$

Softmax Classification

- Cost (Cross entropy)

$$= \sum_i (L_i) \times -\log(\bar{y}_i)$$

$$L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

B

$\bar{Y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ **B** **Cost** ↓ $\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes -\log \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ ➡ $0 + 0 = 0$

$\bar{Y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ **A** **Cost** ↑ $\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes -\log \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ \infty \end{bmatrix}$ ➡ $0 + \infty = \infty$

$$L = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

A

$\bar{Y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ **A** **Cost** ↓ $\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes -\log \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ ➡ $0 + 0 = 0$

$\bar{Y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ **B** **Cost** ↑ $\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes -\log \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} \infty \\ 0 \end{bmatrix}$ ➡ $0 + \infty = \infty$

Softmax Classification

- Cost(Loss) function & minimization

$$L = \frac{1}{N}\sum_i D(S(wx_i + b), L_i)$$

Loss

Minimization->Gradient descent

Implementation

- Docker

docker | join us April 17-20 for dockercon17 | Register

DOCS | EVENTS | COMMUNITY | SUPPORT | TRAINING | PARTNERS | BLOG | LOG-IN | SIGN-UP

What is Docker? | Solutions | Get Docker | Pricing | Open Source | Company

Get started with Docker for Windows

Estimated reading time: 26 minutes

Welcome to Docker for Windows!

Docker is a full development platform for creating containerized apps, and Docker for Windows is the best way to get started with Docker on Windows systems.

> **Already have Docker for Windows?** If you already have Docker for Windows installed, and are ready to get started, skip down to Step 3. Check versions of Docker Engine, Compose, and Machine to work through the rest of the Docker for Windows tour, or jump over to getting started tutorials at Learn Docker.

Download Docker for Windows

If you have not already done so, please install Docker for Windows. You can download installers from the stable or beta channel. For more about stable and beta channels, see the FAQs.

Stable channel

This installer is fully baked and tested, and comes with the latest GA version of Docker Engine along with experimental features in Docker Engine, which are enabled by default and configurable on Docker Daemon settings for experimental mode.

This is the best channel to use if you want a reliable platform to work with. (Be sure to disable experimental features for apps in production.)

These releases follow a version schedule with a longer lead time than the betas, synched with Docker Engine releases and hotfixes.

On the stable channel, you can select whether to send usage statistics and other data.

Get Docker for Windows (stable)

Beta channel

This installer provides the latest Beta release of Docker for Windows, offers cutting edge features along with experimental features in Docker Engine, which are enabled by default and configurable on Docker Daemon settings for experimental mode.

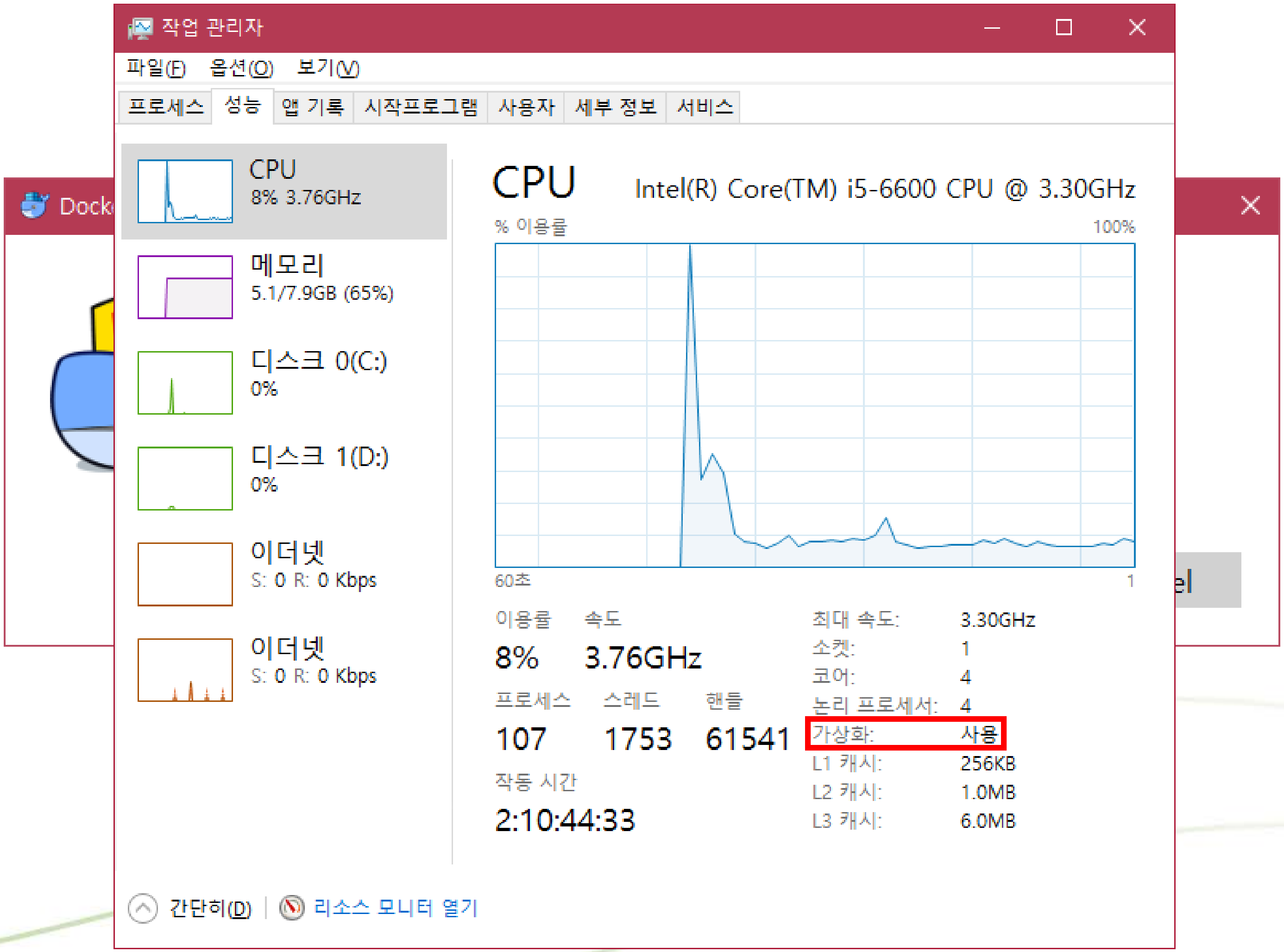
This is the best channel to use if you want to experiment with features under development, and can weather some instability and bugs. This channel is a continuation of the beta program, where you can provide feedback as the apps evolve. Releases are typically more frequent than for stable, often one or more per month.

We collect all usage data on betas across the board.

Get Docker for Windows (beta)

Implementation

- Docker



Implementation

✯ Docker

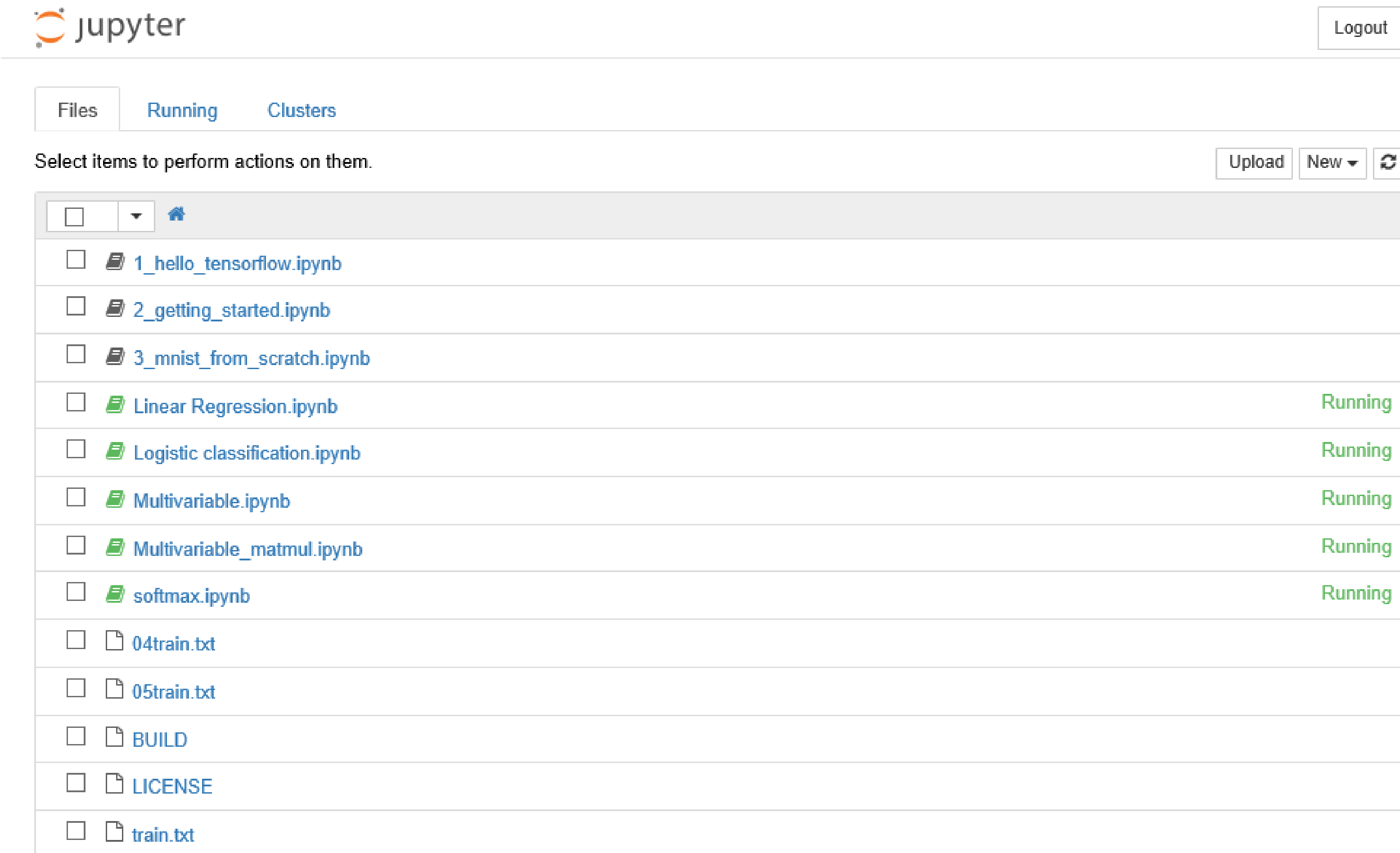
```
명령 프롬프트
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:₩Users₩River>docker --version
Docker version 1.13.0, build 49bf474

C:₩Users₩River>docker images
REPOSITORY                 TAG      IMAGE ID       CREATED        SIZE
gcr.io/tensorflow/tensorflow   latest   e3f7f02f1c66   5 weeks ago    981 MB
d4w/nsenter                latest   9e4f13a0901e   4 months ago   83.8 kB

C:₩Users₩River>_
```

We provide 4 Docker images:

- `gcr.io/tensorflow/tensorflow`: TensorFlow CPU binary image.
- `gcr.io/tensorflow/tensorflow:latest-devel`: CPU Binary image plus source code.
- `gcr.io/tensorflow/tensorflow:latest-gpu`: TensorFlow GPU binary image.
- `gcr.io/tensorflow/tensorflow:latest-devel-gpu`: GPU Binary image plus source code.

Implementation

- Docker



Here is the complete set of keyboard shortc

Shortcut	Action
Shift-Enter	run cell
Ctrl-Enter	run cell in-place
Alt-Enter	run cell, insert below
Ctrl-m x	cut cell
Ctrl-m c	copy cell
Ctrl-m v	paste cell
Ctrl-m d	delete cell
Ctrl-m z	undo last cell deletion
Ctrl-m -	split cell
Ctrl-m a	insert cell above
Ctrl-m b	insert cell below
Ctrl-m o	toggle output
Ctrl-m O	toggle output scroll
Ctrl-m l	toggle line numbers
Ctrl-m s	save notebook
Ctrl-m j	move cell down
Ctrl-m k	move cell up
Ctrl-m y	code cell
Ctrl-m m	markdown cell
Ctrl-m t	raw cell
Ctrl-m 1-6	heading 1-6 cell
Ctrl-m p	select previous
Ctrl-m n	select next
Ctrl-m i	interrupt kernel
Ctrl-m .	restart kernel
Ctrl-m h	show keyboard shortcuts

Implementation

- Docker

```
In [2]: import tensorflow as tf

        hello = tf.constant('Hellow, TensorFlow!')
        print hello
        sess = tf.Session()
        print sess.run(hello)

        Tensor("Const_1:0", shape=(), dtype=string)
        Hellow, TensorFlow!

In [3]: a = tf.constant(2)
        b = tf.constant(3)

        c = a+b

        print c

        print sess.run(c)

        Tensor("add:0", shape=(), dtype=int32)
        5

In [ ]:
```

Implementation

- Linear Regression

```
In [9]: import tensorflow as tf

        #Train data, w=1, b=0
        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))

        #Hypothesis
        hypothesis = w*x_data +b
        #cost
        cost = tf.reduce_mean(tf.square(hypothesis - y_data))
        #minimize
        a = tf.Variable(0.1) #Learning rate, alpha
        optimizer = tf.train.GradientDescentOptimizer(a)
        train = optimizer.minimize(cost)
        #initialize tf.initialize_all_variables()->tf.global_variables_initializer
        #will be removed after 2017-03-02.
        init = tf.global_variables_initializer()
        #Launch the graph
        sess = tf.Session()
        sess.run(init)
        #Fit the line
        for step in xrange(2001):
            sess.run(train)
            if step % 20 ==0:
                print step, sess.run(cost), sess.run(w), sess.run(b)
```

```
540 3.2685e-13 [ 0.99999934] [  1.43538921e-06]
560 9.4739e-14 [ 0.99999958] [  8.71132386e-07]
580 6.15804e-14 [ 0.99999964] [  6.72450426e-07]
600 6.15804e-14 [ 0.99999976] [  5.45293915e-07]
620 2.36848e-14 [ 0.99999988] [  2.75086109e-07]
640 1.89478e-14 [ 0.99999994] [  1.16140292e-07]
660 0.0 [ 1.] [  5.25620081e-08]
680 0.0 [ 1.] [  5.25620081e-08]
700 0.0 [ 1.] [  5.25620081e-08]
720 0.0 [ 1.] [  5.25620081e-08]
740 0.0 [ 1.] [  5.25620081e-08]
760 0.0 [ 1.] [  5.25620081e-08]
780 0.0 [ 1.] [  5.25620081e-08]
800 0.0 [ 1.] [  5.25620081e-08]
820 0.0 [ 1.] [  5.25620081e-08]
840 0.0 [ 1.] [  5.25620081e-08]
```

Implementation

- Linear Regression with placeholder

In [5]:
```
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)
Y = tf.placeholder(tf.float32)

#Hypothesis
hypothesis = w*X +b
#cost
cost = tf.reduce_mean(tf.square(hypothesis - Y))
#minimize
a = tf.Variable(0.1) #Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
#initialize tf.initialize_all_variables()->tf.global_variables_initializer
#will be removed after 2017-03-02.
init = tf.global_variables_initializer()
#Launch the graph
sess = tf.Session()
sess.run(init)
#Fit the line
for step in xrange(2001):
    sess.run(train, feed_dict={X:x_data, Y:y_data})
    if step % 20 ==0:
        print step, sess.run(cost, feed_dict={X:x_data, Y:y_data}), sess.rur
#Learn best fit is w=[1], b=[0]
print sess.run(hypothesis, feed_dict={X:5.})
print sess.run(hypothesis, feed_dict={X:2.5})
```

```
1900 0.0 [ 1.] [  5.95162248e-08]
1920 0.0 [ 1.] [  5.95162248e-08]
1940 0.0 [ 1.] [  5.95162248e-08]
1960 0.0 [ 1.] [  5.95162248e-08]
1980 0.0 [ 1.] [  5.95162248e-08]
2000 0.0 [ 1.] [  5.95162248e-08]
[ 5.]
[ 2.5]
```

Implementation

- Logistic classification

In [5]:
```python
import tensorflow as tf
import numpy as np

xy = np.loadtxt('04train.txt', unpack=True, dtype='float32')
x_data = xy[0:-1]
y_data = xy[-1]

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

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

h = tf.matmul(W, X)
hypothesis = tf.div(1., 1. + tf.exp(-h))

cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))

a = tf.Variable(0.1)  # learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)  # goal is minimize cost

init = tf.global_variables_initializer()

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

for step in xrange(2001):
    sess.run(train, feed_dict={X: x_data, Y: y_data})
    if step % 20 == 0:
        print step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W)

print '----------------------------------------'
print sess.run(hypothesis, feed_dict={X: [[1], [2], [2]]}) > 0.5
print sess.run(hypothesis, feed_dict={X: [[1], [5], [5]]}) > 0.5
print sess.run(hypothesis, feed_dict={X: [[1, 1], [4, 3], [2, 5]]}) > 0.5
```

```
#x0 x1 x2 y
1   2   1   0
1   3   2   0
1   3   5   0
1   5   5   1
1   7   5   1
1   2   5   1

```

```
1840 0.341649 [[-5.7943778   0.46956521  1.00736821]]
1860 0.341405 [[-5.81615019  0.47058851  1.0111444 ]]
1880 0.341166 [[-5.83767319  0.47159278  1.01488173]]
1900 0.340932 [[-5.85895205  0.47257826  1.01858103]]
1920 0.340704 [[-5.87999249  0.47354579  1.02224302]]
1940 0.34048 [[-5.90079832  0.47449559  1.0258683 ]]
1960 0.340262 [[-5.92137575  0.47542834  1.02945769]]
1980 0.340048 [[-5.94172859  0.47634441  1.03301191]]
2000 0.339839 [[-5.96186209  0.4772442   1.03653145]]
----------------------------------------
[[False]]
[[ True]]
[[False  True]]
```

Implementation

Softmax classification

In [3]:
```
import tensorflow as tf
import numpy as np

xy = np.loadtxt('05train.txt', unpack=True, dtype='float32')

x_data = np.transpose(xy[0:3])
y_data = np.transpose(xy[3:])

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

W = tf.Variable(tf.zeros([3, 3]))

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

learning_rate = 0.01

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), reduction_indices=1))

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


init = tf.global_variables_initializer()

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

    for step in xrange(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}), sess.run(W)

    a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7]]})
    print "a :", a, sess.run(tf.arg_max(a, 1))

    b = sess.run(hypothesis, feed_dict={X: [[1, 3, 4]]})
    print "b :", b, sess.run(tf.arg_max(b, 1))

    c = sess.run(hypothesis, feed_dict={X: [[1, 1, 0]]})
    print "c :", c, sess.run(tf.arg_max(c, 1))

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

$$L = \frac{1}{N}\sum_i D(S(wx_i + b), L_i)$$

```
#x0 x1 x2 y[A   B    C]
1   2   1   0   0    1
1   3   2   0   0    1
1   3   4   0   0    1
1   5   5   0   1    0
1   7   5   0   1    0
1   2   5   0   1    0
1   6   6   1   0    0
1   7   7   1   0    0
```

```
1200 0.780959 [[-1.06231129 -0.26727253  1.32958329]
 [ 0.06808005 -0.11823834  0.05015875]
 [ 0.17550454  0.23514733 -0.41065112]]
1400 0.756943 [[-1.19854808 -0.29670808  1.49525583]
 [ 0.07591439 -0.11214777  0.03623381]
 [ 0.19498996  0.237331   -0.43232018]]
1600 0.735893 [[-1.32743537 -0.32218221  1.64961684]
 [ 0.08333746 -0.10557999  0.022243  ]
 [ 0.21336642  0.23823628 -0.45160189]]
1800 0.717269 [[-1.44994974 -0.34407791  1.79402602]
 [ 0.09020081 -0.09902246  0.00882213]
 [ 0.23099625  0.23841871 -0.46941414]]
2000 0.700649 [[-1.56689739 -0.36275655  1.92965221]
 [ 0.09643649 -0.09271803 -0.00371792]
 [ 0.24811605  0.23818412 -0.48629922]]
a :  [[ 0.68849677  0.26731515  0.04418808]] [0]
b :  [[ 0.24322268  0.44183081  0.3149465 ]] [1]
c :  [[ 0.02974809  0.08208466  0.8881672 ]] [2]
all :  [[ 0.68849677  0.26731515  0.04418808]
 [ 0.24322268  0.44183081  0.3149465 ]
 [ 0.02974809  0.08208466  0.8881672 ]] [0 1 2]
```

Q & A

Thank You!!!