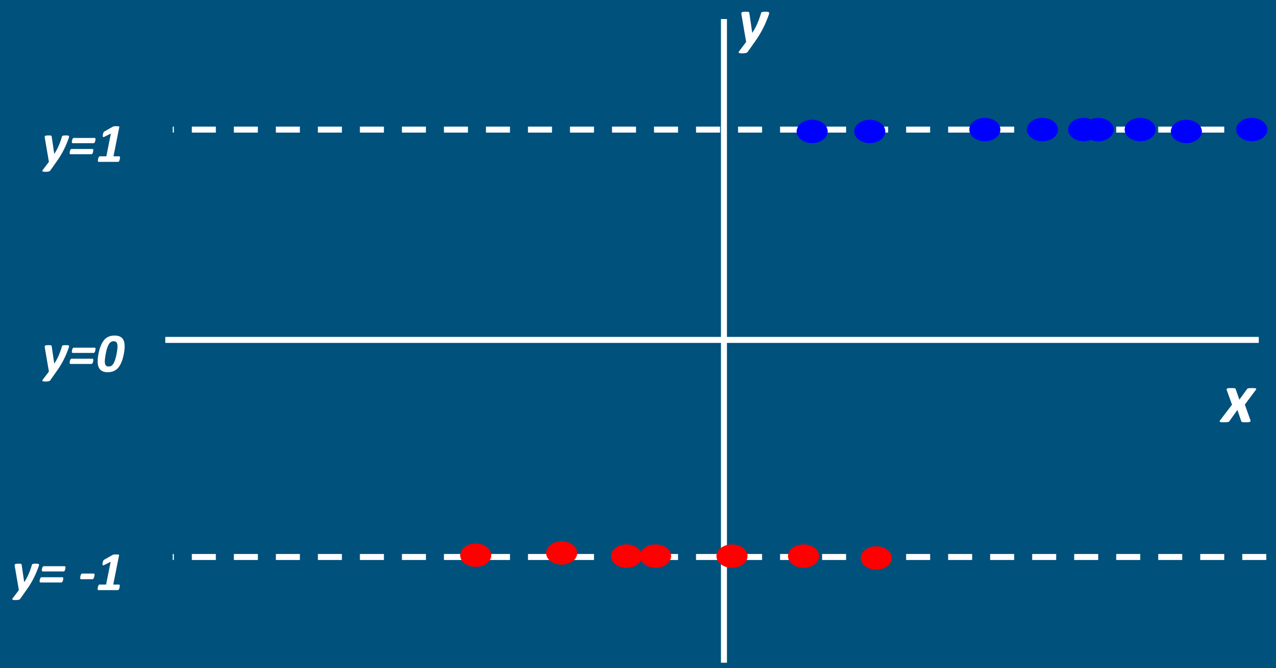
Linear Classifier

Dr. Dongchul Kim

Linear Classifier for Binary class

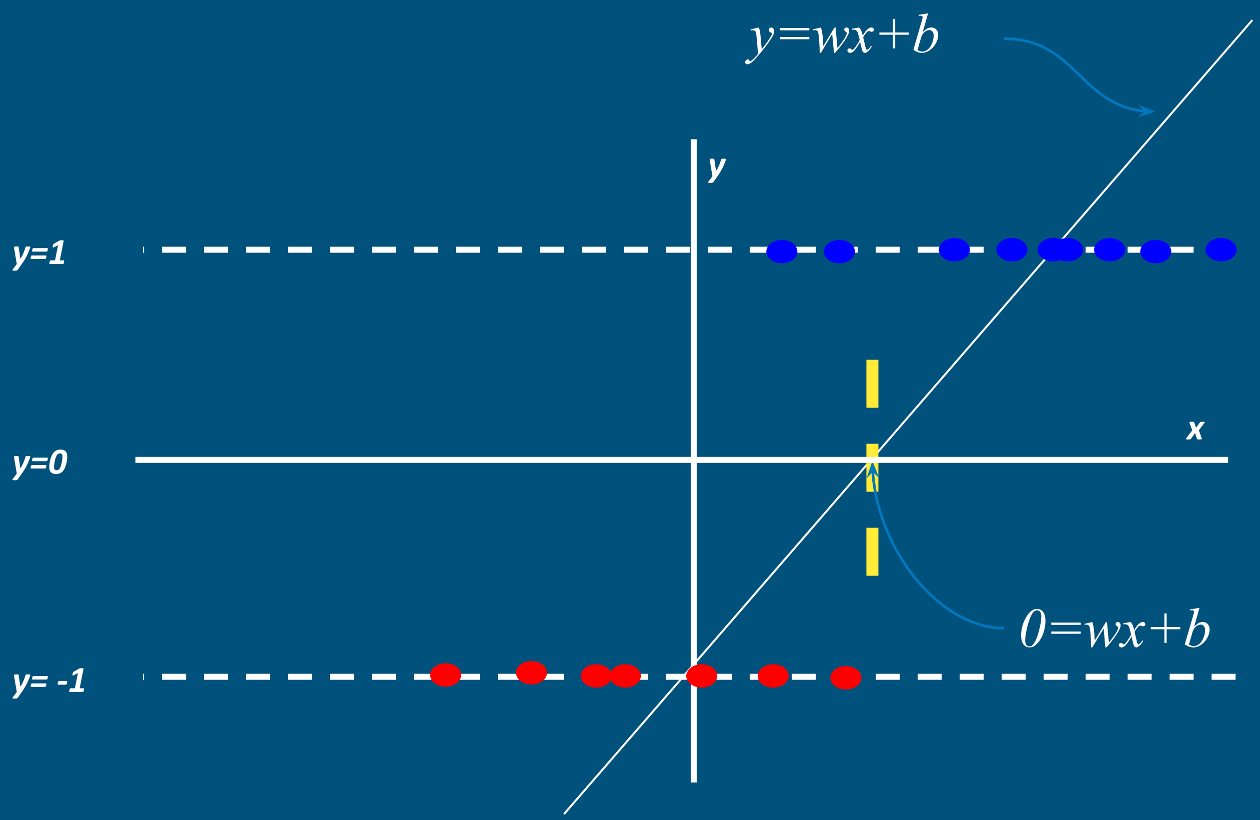
- Let's simplify the data by assuming:
  - x is a scalar. (a single feature)
  - There are only two classes, $y=(+1)$ and $y=(-1)$


y
y=1
y=0
x
y= -1

Linear Classifier

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

$$sign(x) = \begin{cases} +1 & if \quad x \geq 0 \\ -1 & if \quad x < 0 \end{cases}$$



Linear regression model is a line (a single IV)
Classification boundary is a point (1D) but not line (2D)

Example

Binary class data

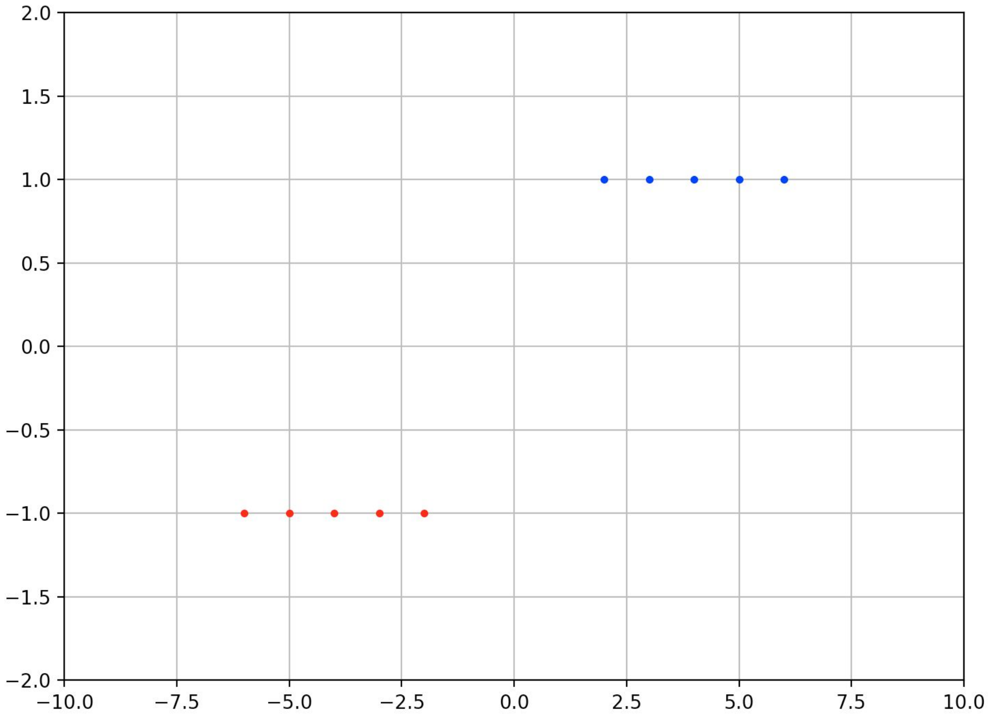
x	y
2	+1
3	+1
4	+1
5	+1
6	+1
-2	-1
-3	-1
-4	-1
-5	-1
-6	-1

plot

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

trainx = np.array([2, 3, 4, 5, 6, -2, -3, -4, -5, -6])
trainy = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1, -1])

plt.plot(trainx[0:5], trainy[0:5], 'bo', markersize=3)
plt.plot(trainx[5:], trainy[5:], 'ro', markersize=3)
plt.axis([-10, 10, -2, 2])
plt.grid(True)
plt.show()
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Estimate *w* and *b*

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

# data
trainx = np.array([2, 3, 4, 5, 6, -2, -3, -4, -5, -6])
trainy = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1, -1])
# plot
plt.plot(trainx[0:5], trainy[0:5], 'bo', markersize=3)
plt.plot(trainx[5:], trainy[5:], 'ro', markersize=3)
xl = np.linspace(-10, 10, 100)
plt.axis([-10, 10, -2, 2])
# initialization
w, b = 0, 0
# learning rate
alpha = 0.05
# GD
for i in range(10):
    w = w - alpha * (1/len(trainx)) * sum((w * trainx + b - trainy) * trainx)
    b = b - alpha * (1/len(trainx)) * sum(w * trainx + b - trainy)
print("w = %f, b = %f" % (w, b))
# plot
plt.plot(xl, w * xl + b)
plt.grid(True)
plt.show()
```

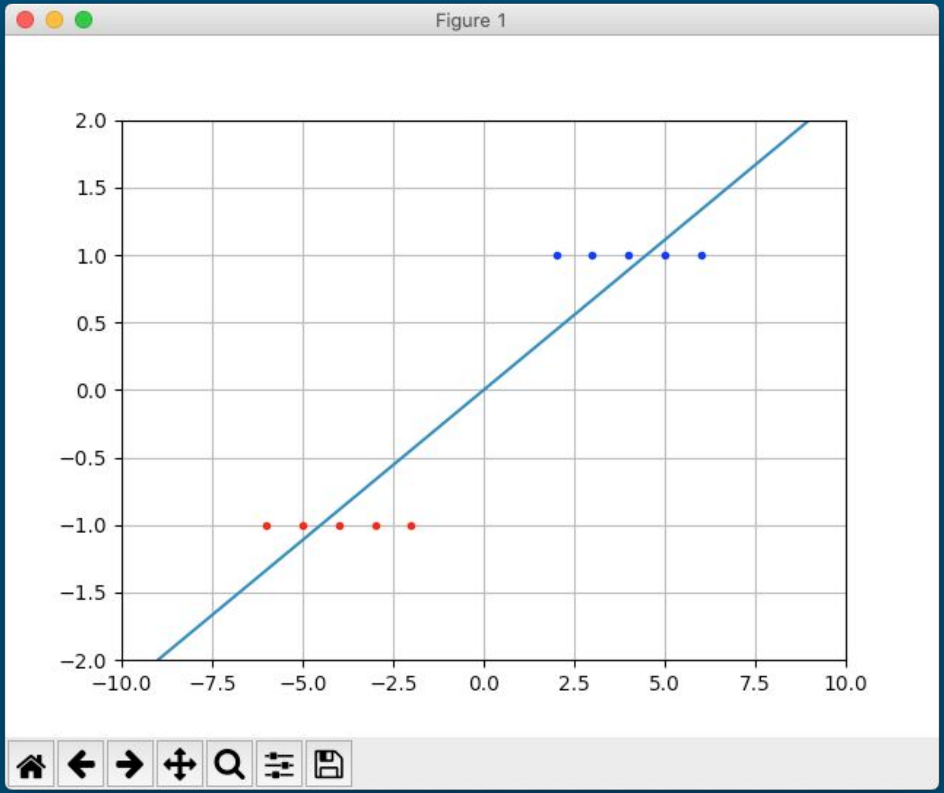
Estimate *w* and *b*

Run: linear_classifier_gd

```
/Users/dkim/PycharmProjects/CSCI4352/venv/bin/python /Users/dkim/PycharmProjects/CSCI4352/linear_classifier_gd.py
0: w = 0.200000, b = 0.000000
1: w = 0.220000, b = 0.000000
2: w = 0.222000, b = 0.000000
3: w = 0.222200, b = 0.000000
4: w = 0.222220, b = 0.000000
5: w = 0.222222, b = 0.000000
6: w = 0.222222, b = 0.000000
7: w = 0.222222, b = 0.000000
8: w = 0.222222, b = 0.000000
9: w = 0.222222, b = 0.000000
```

Plot

- w = 0.2222222
- b = 0



Test

```python
import numpy as np

def hypothisis(_w, _x, _b):
    return _w * _x + _b

# train data
trainx = np.array([2, 3, 4, 5, 6, -2, -3, -4, -5, -6])
trainy = np.array([1, 1, 1, 1, 1, -1, -1, -1, -1, -1])
# test data
testx = np.array([-4.4, 0.9, -2.5, -0.7, 2.4, 5.2])
testy = np.array([-1, 1, -1, -1, 1, 1])

# initialization
w, b = 0, 0
# learning rate
alpha = 0.05
# GD
for i in range(10):
    w = w - alpha * (1/len(trainx)) * sum((w * trainx + b - trainy) * trainx)
    b = b - alpha * (1/len(trainx)) * sum(w * trainx + b - trainy)
print("w = %f, b = %f" % (w, b))

# test
print("Accuracy:", sum(np.sign(hypothisis(w, testx, b)) == testy)/len(testx))
```

Accuracy

```
w = 0.222222, b = 0.000000
Accuracy: 1.0
```

Multiple Linear Classifier for binary class data

Without using matrix

```python
import numpy as np

def hypothesis(w1, x1, w2, x2, w3, x3, b):
    return w1 * x1 + w2 * x2 + w3 * x3 + b

# data
trainX = np.array([[1.5, 2.7, 1.3],
                   [2.4, 1.7, 2.1],
                   [2.5, 1.3, 2.2],
                   [8.5, 5.3, 4.8],
                   [4.9, 6.4, 5.7],
                   [7.2, 7.1, 7.4]])
trainy = np.array([1, 1, 1, -1, -1, -1])
testX = np.array([[2.4, 2.5, 0.7],
                  [5.9, 4.4, 5.2]])
testy = np.array([1, -1])
# initialization
w1, w2, w3, b = 0, 0, 0, 0
# learning rate
alpha = 0.05
# GD
for i in range(100):
    w1 = w1 - alpha * (1 / len(trainX)) * sum((trainX[:, 0] * w1 + trainX[:, 1] * w2 + trainX[:, 2] * w3 + b - trainy) * trainX[:, 0])
    w2 = w2 - alpha * (1 / len(trainX)) * sum((trainX[:, 0] * w1 + trainX[:, 1] * w2 + trainX[:, 2] * w3 + b - trainy) * trainX[:, 1])
    w3 = w3 - alpha * (1 / len(trainX)) * sum((trainX[:, 0] * w1 + trainX[:, 1] * w2 + trainX[:, 2] * w3 + b - trainy) * trainX[:, 2])
    b = b - alpha * (1 / len(trainX)) * sum(trainX[:, 0] * w1 + trainX[:, 1] * w2 + trainX[:, 2] * w3 + b - trainy)
print("w1 = %f, w2 = %f, w3 = %f, b = %f" % (w1, w2, w3, b))
# test (accuracy)
print(sum(np.sign(hypothesis(w1, testX[:, 0], w2, testX[:, 1], w3, testX[:, 2], b)) == testy)/len(testX))
```

```
w1 = -0.123876, w2 = -0.244516, w3 = 0.046912, b = 1.244646
1.0
```

Using matrix

Linear Classifier (Multi-dimension)

```python
import numpy as np


def hypothesis(X, w, b):
    return np.dot(X, w)+b

# data
trainX = np.array([[1.5, 2.7, 1.3],
                   [2.4, 1.7, 2.1],
                   [2.5, 1.3, 2.2],
                   [8.5, 5.3, 4.8],
                   [4.9, 6.4, 5.7],
                   [7.2, 7.1, 7.4]])
trainy = np.array([1, 1, 1, -1, -1, -1])
testX = np.array([[2.4, 2.5, 0.7],
                  [5.9, 4.4, 5.2],
                  [0.2, 0.5, 0.6],
                  [4.3, 4.5, 5.5]])
testy = np.array([1, -1, 1, -1])
# initialization
w = np.zeros(np.size(trainX, 1))
b = 0
# learning rate
alpha = 0.01
# GD
for i in range(2000):
    w = w - alpha * (1 / len(trainX)) * np.dot(np.transpose(np.dot(trainX, w)+b - trainy), trainX)
    b = b - alpha * (1 / len(trainX)) * sum(np.dot(trainX, w)+b - trainy)
print(w, b)
# test (accuracy)
print(sum(np.sign(hypothesis(testX, w, b)) == testy)/len(testX))
```

Linear Classifier (Multi-dimension)

```
/Users/dkim/PycharmProjects/CSCI4352/venv/bin/python /Users/dkim/PycharmProjects/CSCI4352/linear_classifier_gd_test_data_multivariable_matrix.py
[-0.15005405 -0.28097513  0.00386677] 1.8027770609918465
1.0

Process finished with exit code 0
```

Lab

- Implement a linear classifier with **Iris** data. (https://archive.ics.uci.edu/ml/datasets/iris)
- Step 1: Use **only the first 100 samples** (**only two classes**) to make it binary class data.
- Step 2: Shuffle and split the data into train and test (test size is 0.2)
- Step 3: Repeat Step 2 for 100 times, then calculate accuracy on average.