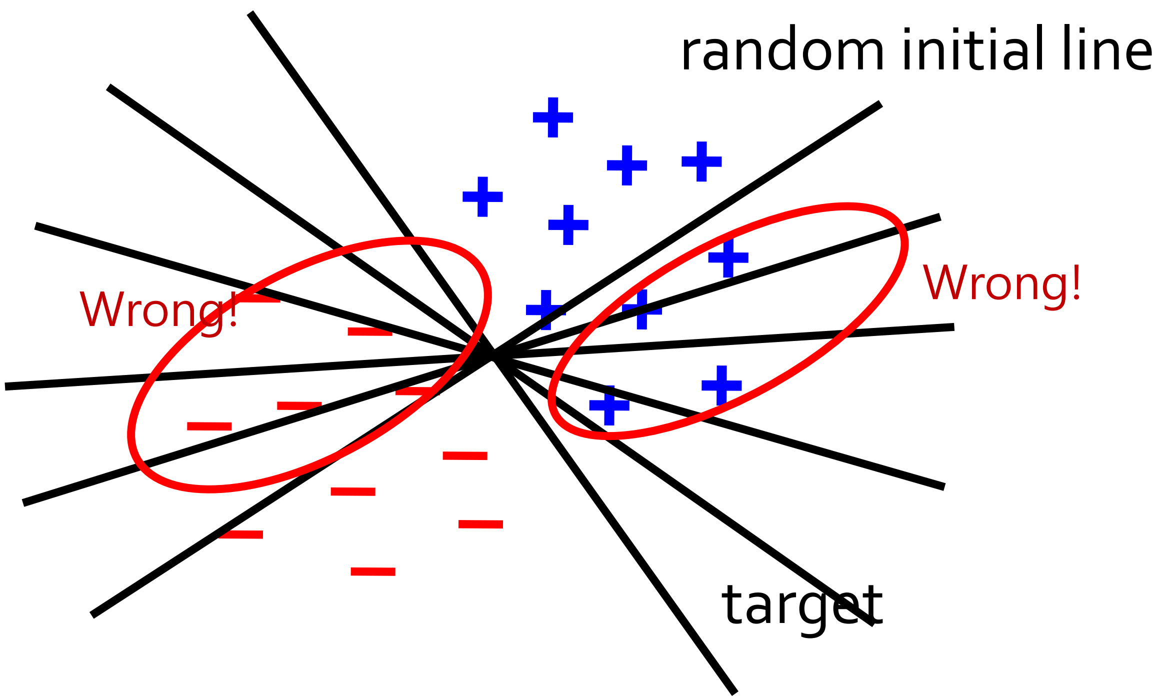
Logistic Regression

Logistic Regression

Goal: find the **best line separating** two sets of points



[Harold Liu with UC, Ber

Logistic Regression

Goal: find the **best line separating** two sets of points

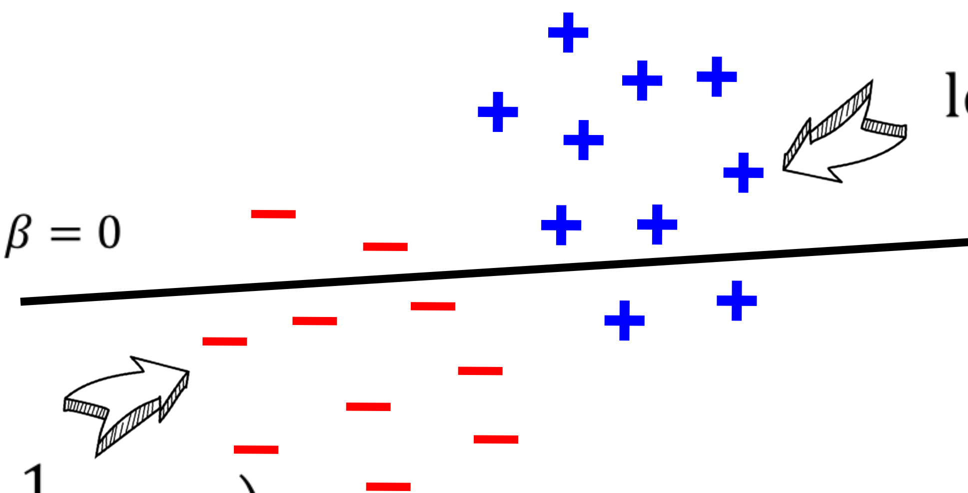
$y_i = 1$

Good as much as

$$\log \frac{1}{1 + e^{-(\beta_0 + x_i \cdot \beta)}}$$

Hyperplane:

$$\beta_0 + x_i \cdot \beta = 0$$



Good as much as

$$\log\left(1 - \frac{1}{1 + e^{-(\beta_0 + x_i \cdot \beta)}}\right)$$

$y_i = 0$

Optimization Problem

Maximize

$$\sum_{i=1}^{n} y_i \cdot \log\left(\frac{1}{1+e^{-(\beta_0+x_i\cdot\beta)}}\right) + \sum_{i=1}^{n}(1-y_i)\cdot\log\left(1-\frac{1}{1+e^{-(\beta_0+x_i\cdot\beta)}}\right)$$

Gradient descent method

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \alpha \sum_{i=1}^{n} \boxed{\left(y_i - \frac{1}{1+e^{-(\beta_0+x_i\cdot\beta)}}\right) x_i}$$

Sum of values calculated with each data points

Logistic Regression Code

RDD

```
▪ val data = spark.textFile(...).map(readPoint).cache()

▪ var w = Vector.random(D)

▪ for (i <- 1 to ITERATIONS) {
▪    val gradient = data.map(p =>
▪        (p.y - 1 / (1 + exp(-(w dot p.x)))) * p.x
▪    ).reduce(_ + _)
▪    w += gradient
▪ }

▪ println("Final w: " + w)
```

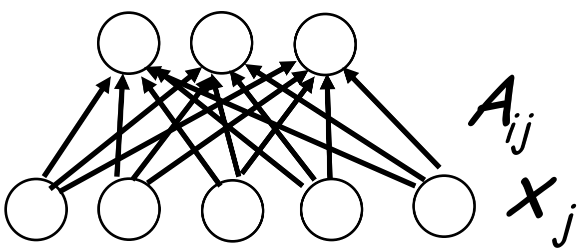
[Harold Liu with UC, Ber

Logistic Regression

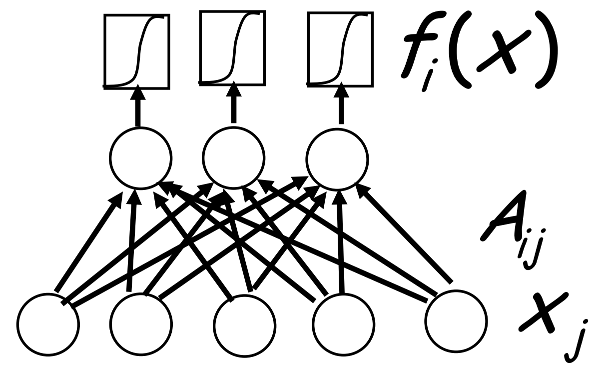
- This is also regression but with targets Y=(0,1). I.e. it is classification!
- We will fit a regression function on P(Y=1|X)

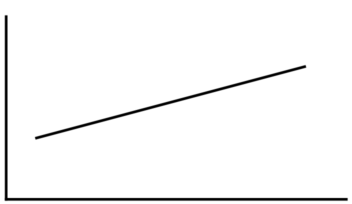
linear regression

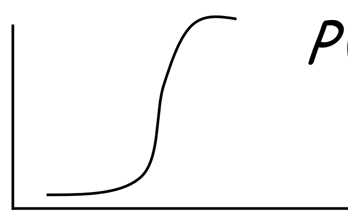
logistic regression



$$Y_n = AX_n + b$$



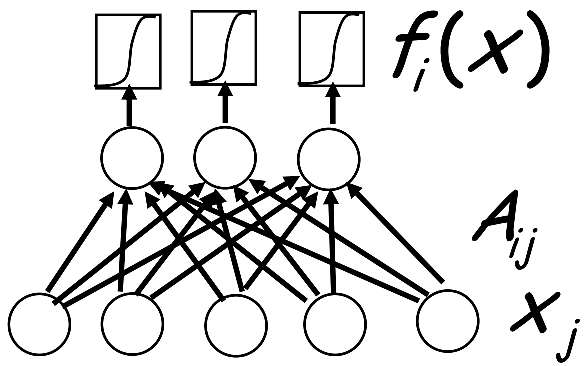




$$P(Y_n = 1 \mid X_n) = f(AX_n + b)$$

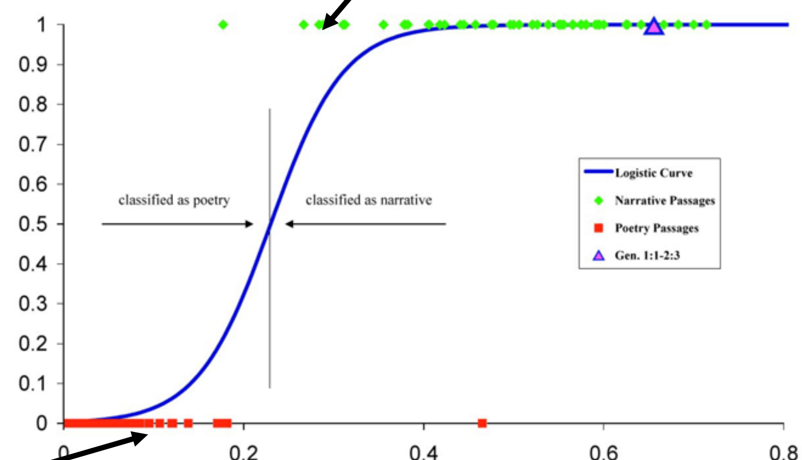
$$f(X) = \frac{1}{1 + \exp[-(AX + b)]}$$

Sigmoid function f(x)



$$P(Y_n = 1 \mid X_n) = f(AX_n + b)$$

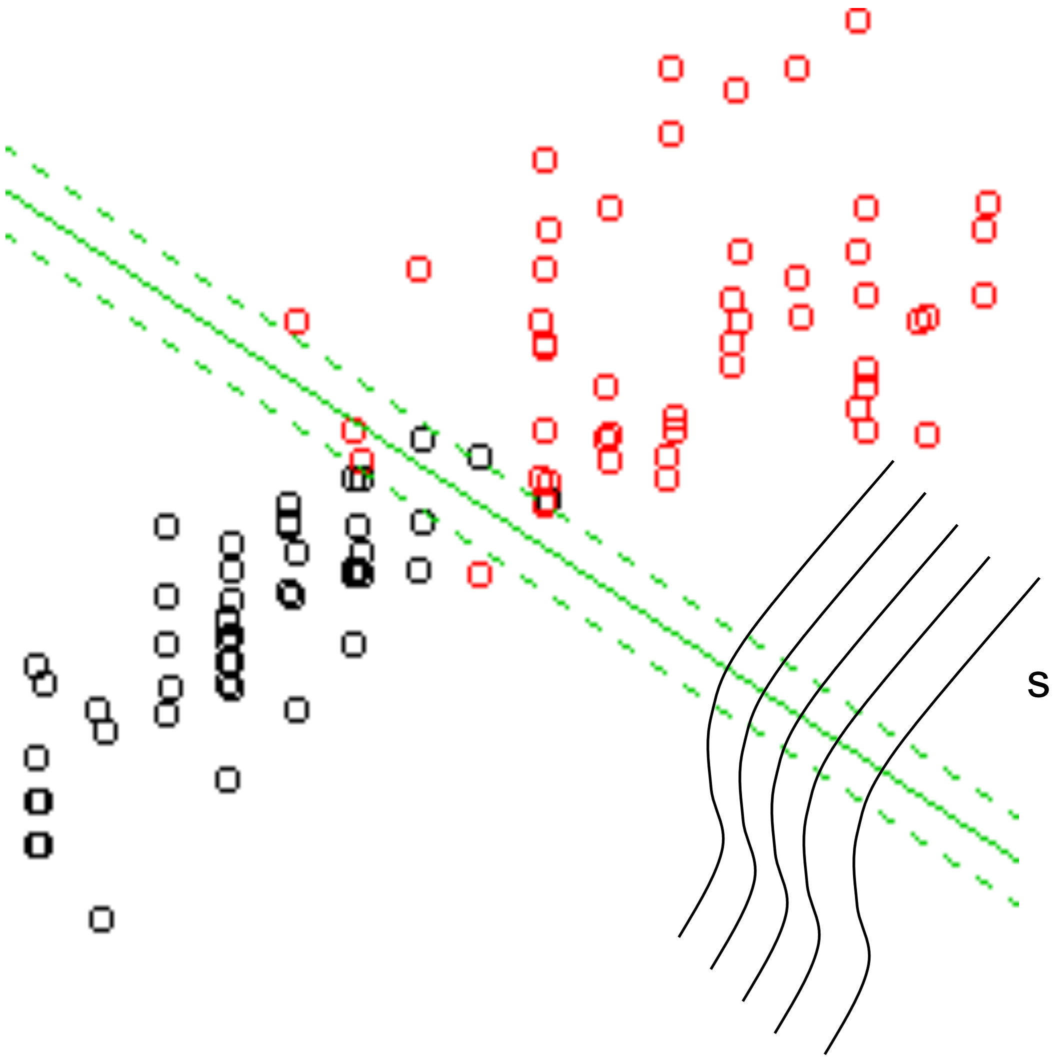
data-points with Y=1



data-points with Y=0

$$f(X) = \frac{1}{1 + \exp[-(AX + b)]}$$

In 2 Dimensions



sigmoid f(x)

A,b determine
1) orientation
2) thickness (margin)
3) offset
of decision surface

Cost Function

We want a different error measure that is better suited for 0/1 data.

This can be derived from maximizing the probability of the data again.

$$P(Y_n = 1 \mid X_n) = f(AX_n + b) \qquad P(Y_n = 0 \mid X_n) = 1 - f(AX_n + b)$$

$$P(Y_n \mid X_n) = f(X_n)^{Y_n}(1 - f(Y_n))^{1-Y_n}$$

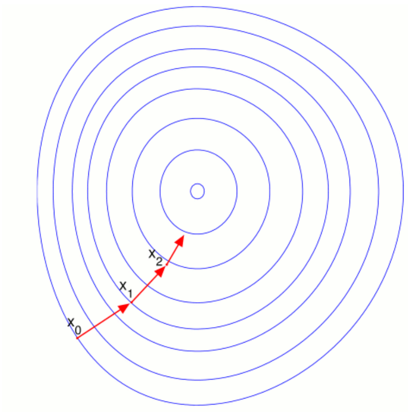
$$Error = -\sum_{n=1}^{N} Y_n \log f(X_n) + (1 - Y_n)\log(1 - f(X_n))$$

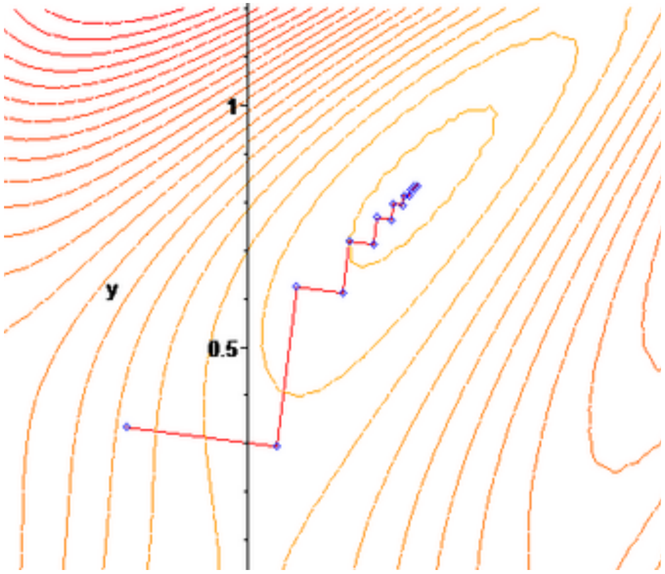
Learning A,b

Again, we take the derivatives of the Error w.r.t the parameters.

This time however, we can't solve them analytically, so we use gradient descent.

$$A \leftarrow A - \eta \frac{dError}{dA} \qquad\qquad b \leftarrow b - \eta \frac{dError}{db}$$





Gradients for Logistic Regression

After the math (on the white-board) we find:

$$\frac{\partial Error}{\partial A} = -\sum_n \left[y_n \left(1 - f(X_n)\right) + (1 - y_n) f(X_n) \right] X_n^T$$

$$\frac{\partial Error}{\partial b} = -\sum_n y_n \left(1 - f(X_n)\right) + (1 - y_n) f(X_n)$$

Note: first term in each eqn. (multiplied by Y) only sums over data with Y=1,
while second term (multiplied by (1-Y) only sums over data with Y=0.

Follow the gradient until the change in A,b falls below a small theshold (e.g. 1E-6).

Classification

Once we have found the optimal values for A, b we classify future data with:

$$Y_{new} = round(f(X_{new}))$$

- Least squares and Logistic regression are parametric methods since all the information in the data is stored in the parameters A,b, i.e. after learning you can toss out the data.

- Also, the decision surface is always linear, its complexity does not grow with the amount of data.

- We have imposed our prior knowledge that the decision surface should be linear.

A logistic regression learning using TensorFlow

```
import tensorflow as tf
import numpy as np
from sklearn import datasets


iris = datasets.load_iris()
iris_X = iris.data
iris_y = iris.target


np.random.seed(0)
indices = np.random.permutation(len(iris_X))
train_X = iris_X[indices[:-10]]
train_Y = np.asarray([ [ 1 if idx == 0 else 0 for idx in iris_y[indices[:-
10]] ] ]).T
test_X  = iris_X[indices[-10:]]
test_Y  = np.asarray([ [ 1 if idx == 0 else 0 for idx in iris_y[indices[-
10:]] ] ]).T
```

A logistic regression learning using TensorFlow

```
# data format is as usual:
# train_X and test_X have shape (num_instances, num_features)
# train_Y and test_Y have shape (num_instances, num_classes)
num_features = train_X.shape[1]
num_classes = train_Y.shape[1]

# Create variables
# X is a symbolic variable which will contain input data
# shape [None, num_features] suggests that we don't limit the number of 
instances in the model
# while the number of features is known in advance
X = tf.placeholder("float", [None, num_features])
# same with labels: number of classes is known, while number of instances is 
left undefined
Y = tf.placeholder("float",[None, num_classes])

# W - weights array
W = tf.Variable(tf.zeros([num_features,num_classes]))
# B - bias array
B = tf.Variable(tf.zeros([num_classes]))
```

A logistic regression learning using TensorFlow

```
# Define a model
# a simple linear model y=wx+b wrapped into softmax
pY = tf.nn.softmax(tf.matmul(X, W) + B)
# pY will contain predictions the model makes, while Y contains real data


# Define a cost function
cost_fn = -tf.reduce_sum(Y * tf.log(pY))


# Define an optimizer
opt = tf.train.AdamOptimizer(0.01).minimize(cost_fn)
```

A logistic regression learning using TensorFlow

```
# Create and initialize a session
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)


# run an optimization step with all train data
sess.run(opt, feed_dict={X:train_X, Y:train_Y})

# Now assess the model
# create a variable which reflects how good your predictions are
# here we just compare if the predicted label and the real label are the
same
accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(pY,1), tf.argmax(Y,1)),
"float"))
# and finally, run calculations with all test data
print(sess.run(accuracy, feed_dict={X:test_X, Y:test_Y}))
```

A logistic regression learning using Scikit-learn

```
from sklearn import linear_model, datasets

iris = datasets.load_iris()
iris_X = iris.data
iris_y = iris.target

np.random.seed(0)
indices = np.random.permutation(len(iris_X))
train_X = iris_X[indices[:-10]]
train_Y = [ 1 if idx == 0 else 0 for idx in iris_y[indices[:-10]] ]
test_X = iris_X[indices[-10:]]
test_Y = [ 1 if idx == 0 else 0 for idx in iris_y[indices[-10:]] ]

logistic = linear_model.LogisticRegression(C=1e5)
logistic.fit(train_X, train_Y)
print(np.mean([ 1 if a == b else 0 for a, b in zip(logistic.predict(test_X),
test_Y) ]))
```