RECAP – Model Generalization

>Underfit (simple model) vs overfit (complex model)

-Use CV to diagnose generalization

-Sklearn - cross_val_score()

>Regularization L1 (Lasso), L2(Ridge)

-Use CV to set the regularization parameter

>Sklearn – LinearRegression(), Ridge(),Lasso()

-RidgeCV(),LassoCV()

Logistic Regression

(Intel week 5: Logistic Regression)

- From **regression** to **classification**
- **The basic unit of neural networks**
- **Multi-class**

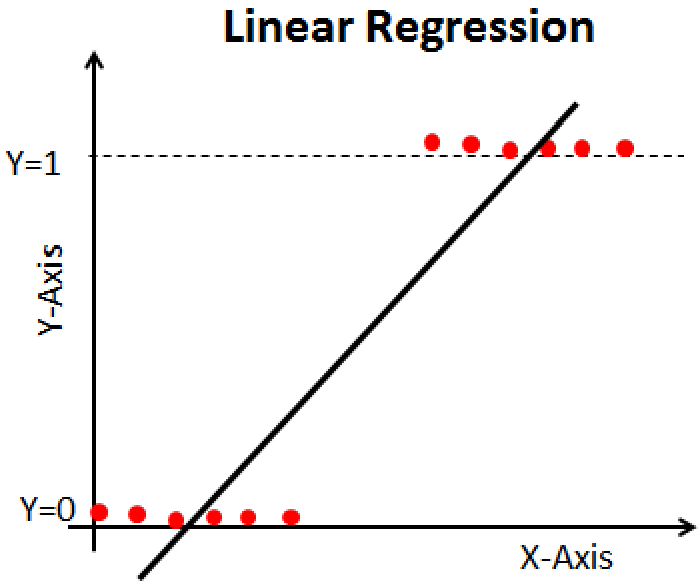
Generalized Linear Models - Extending linear regression

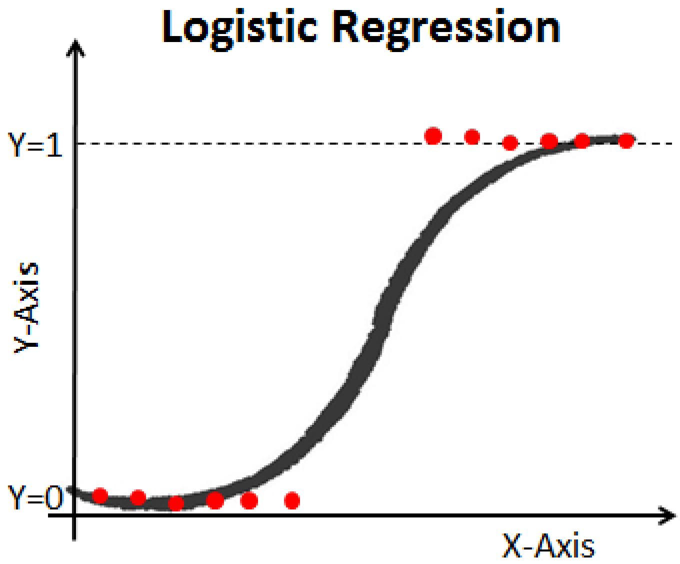
Generalized Linear Models are “plugins” that extend linear regression to other distributions

- Linear regression (continuous to continuous)
- Logistic regression (continuous to binary [True/False])
- Poisson regression (continuous to ordinal (0, 1, 2, 3, ...)

Classification (discrete values)
How can we use linear regression to solve this?

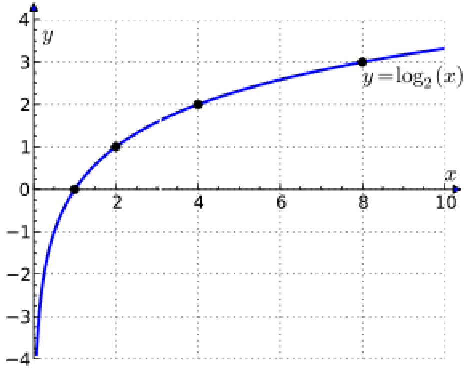
Direct plug-in of discrete values into regression (continuous) framework not efficient since y = {0,1} only





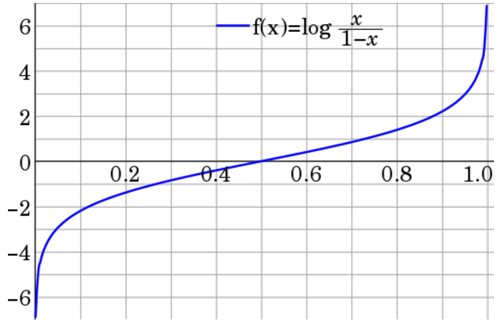
We wish to model p(x) as ax+b, but...
p(x) is bounded and ax+b is unbounded
Let’s find a function of p(x) which is unbounded

Instead of **p(x) = ax+b** **(linear regression)** which is unbounded (but $0<p<1$) use:



log(p(x)) = ax+b , but log is unbounded in one direction

Let’s try:



log(p(x)/(1-p(x))) = ax+b **(logistic regression)**, which is unbounded in both directions

Solving for p(x) ...

log(p(x)/(1-p(x))) = ax+b ===>

p(x) = 1/(1+exp(-[ax+b]))

Therefore:

Predict Y = 1 when p(x)>0.5 i.e. when ax+b>0

(this is a linear classifier: There's no interaction between the weight parameter values, nothing like $w_1x_1 * w_2x_2$)

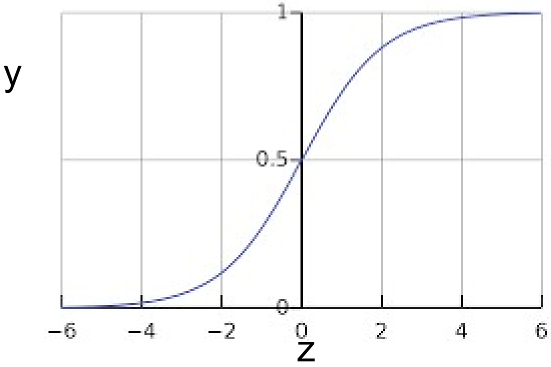
Logistic Regression

Use logistic function (or sigmoid) function to map between continuous to “probabilities”

$$z = b + w_1 x_1 + w_2 x_2 + \ldots \longrightarrow y = \frac{1}{1+e^{-z}}$$

Our old linear model

predicted (“probability”)



$$z = \log\left(\frac{y}{1-y}\right)$$

Z = log odds or logit

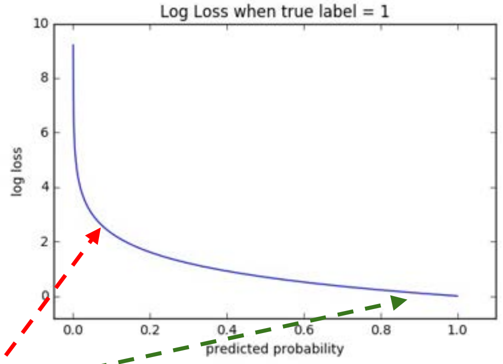
Loss Function (it's easier to minimize the log of the loss)

$$\log Loss = \sum_{n} \left[-y \log(y') - (1-y) \log(1-y') \right]$$

y' = predicted ("probability")
y = label

Intuition: **y** $\neq$ **y'** $\Rightarrow$ Big Loss

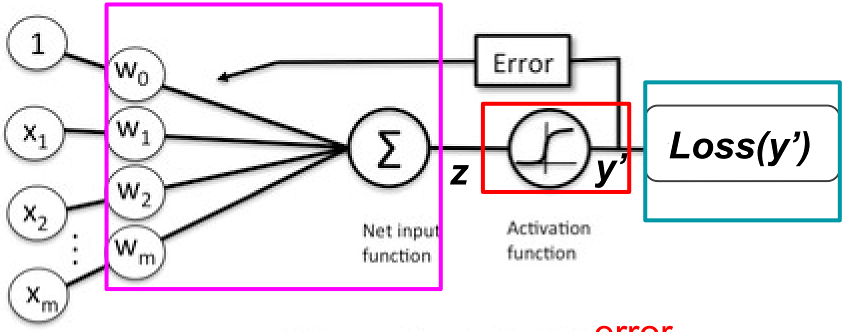
y	y'	logLoss
0	0.9	2.3
0	0.1	0.1
1	0.9	0.1
1	0.1	2.3



Update Rule for log loss minimization

Apply the chain rule

$$\frac{\partial Loss}{\partial W} = \boxed{\frac{\partial Loss}{\partial y'}} \boxed{\frac{\partial y'}{\partial z}} \boxed{\frac{\partial z}{\partial w}}$$



$$\boxed{\frac{\partial Loss}{\partial y'}} = \frac{-y}{y'} + \frac{1-y}{1-y'}$$

$$\boxed{\frac{\partial y'}{\partial z}} = y'(1-y')$$

$$\boxed{\frac{\partial z}{\partial w_i}} = x_i$$

$$\frac{\partial Loss}{\partial W_i} = (y-y')x_i$$

error

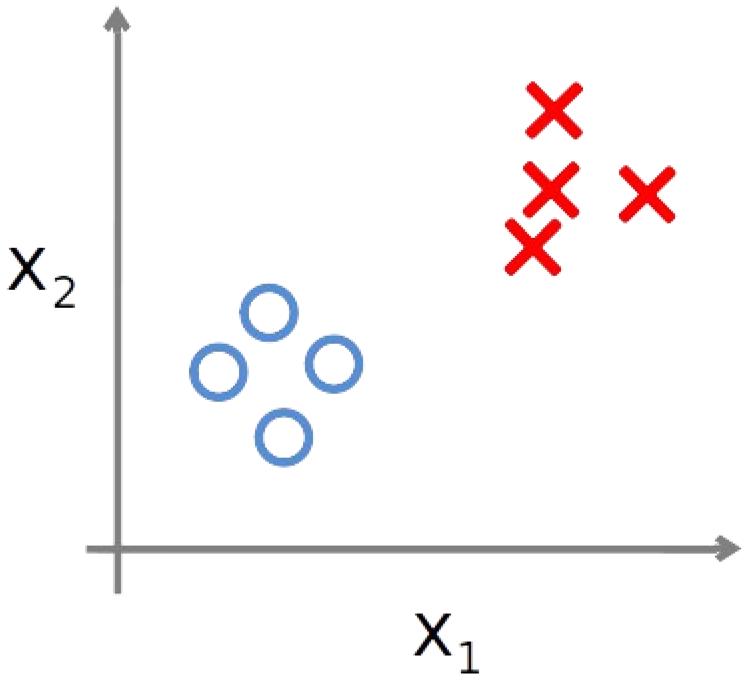
Adaptation step

$$w_i \leftarrow w_i - \alpha \frac{\partial Loss}{\partial W_i} = w_i - \alpha(y'-y)x_i$$

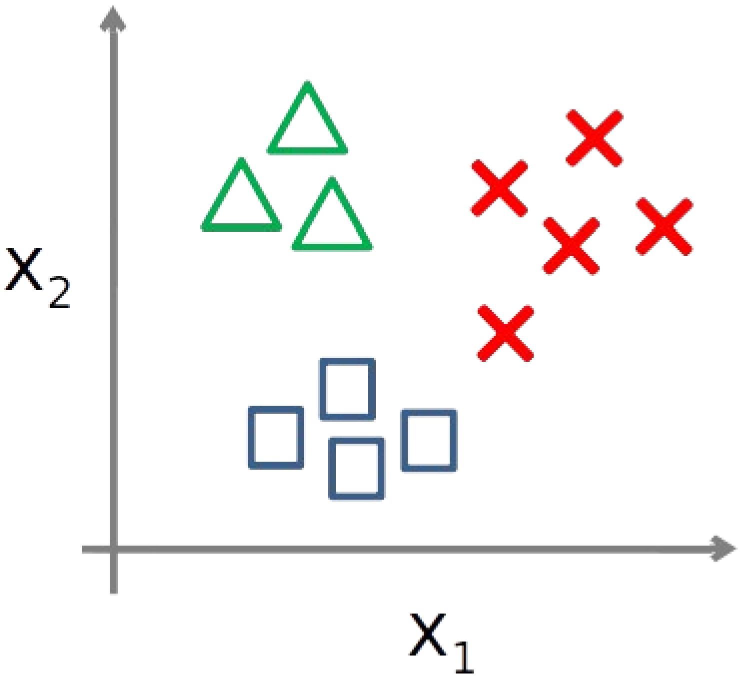
Algorithm looks identical to linear regression!

Extension to multi-class / nonlinear problems

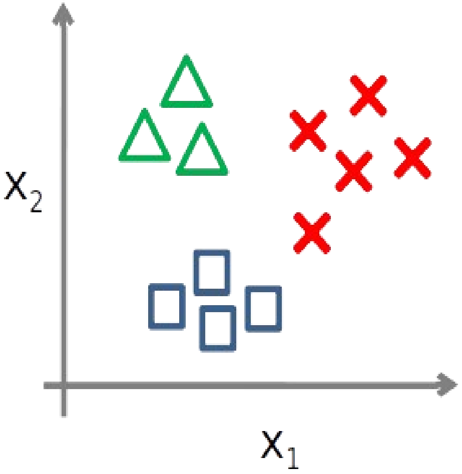
Binary classification:

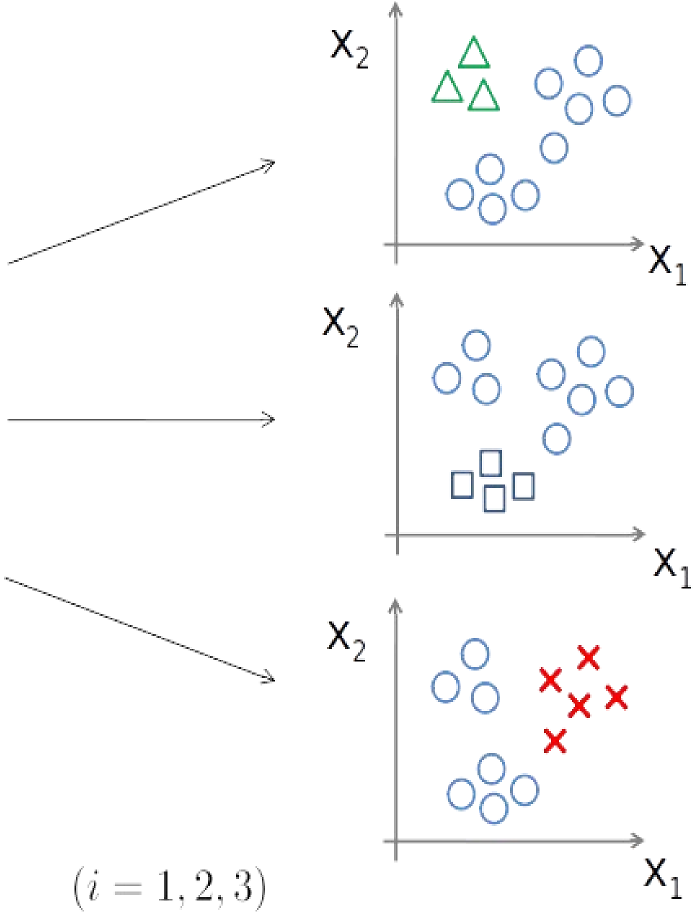


Multi-class classification:



One vs. all or Softmax (for multiclass)

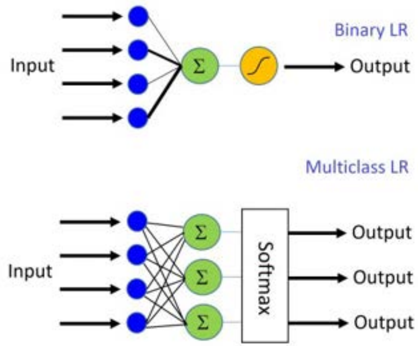




Class 1: △
Class 2: □
Class 3: ×

$$h_\theta^{(i)}(x) = P(y = i | x; \theta) \qquad (i = 1, 2, 3)$$

We could also use the cross-entropy cost (instead of the log loss) for more than two classes



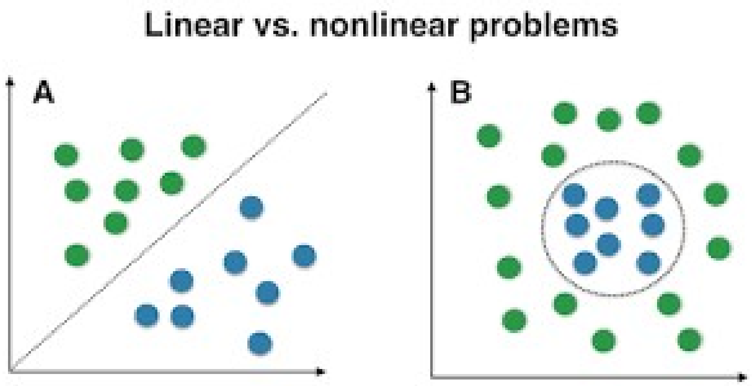
We'll see later the softmax layer, the algebraic simplification of N logistic classifiers.

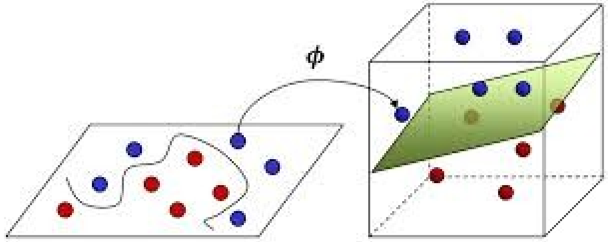
Feature transformations (for nonlinear problems)

How to linearly separate a nonlinear region

1. Map all data points x to higher dimension by transformation function **ϕ**,

   x -> **ϕ**(x),
2. Calculate the hyperplane in the new linear attribute space, i.e. hyperplane a***ϕ**(x)+b.
3. Solve the linear problem y=w(**a*ϕ(x)+b**)





[redacted]