Applied Machine Learning

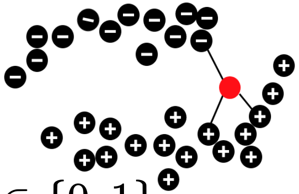
Linear classification

Reihaneh Rabbany

McGill School of Computer Science

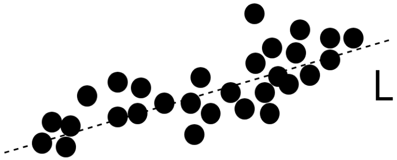
COMP 551 (winter 2020)

Sofar



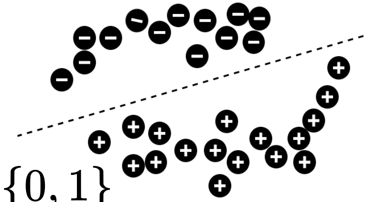
Nearest neighbour classifier

$y \in \{0, 1\}$



Linear regression

$y \in [0, 1]$



Today: logistic regression

A linear classifier!

$y \in \{0, 1\}$

Representing data

each instance: $x^{(n)} \in \mathbb{R}^D$ a (column) vector $\quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}$ a features

$y^{(n)} \in \{1, 2, \dots C\}$

$\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^N$

we assume N instances in the dataset

each instance has D features indexed by d

$$\begin{array}{c} \text{instances} \\ X = \begin{bmatrix} {x^{(1)}}^T \\ {x^{(2)}}^T \\ \vdots \\ {x^{(N)}}^T \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \dots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \dots, & x_D^{(N)} \end{bmatrix} \end{array} \text{features} \in \mathbb{R}^{N \times D}$$

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

Learning objectives

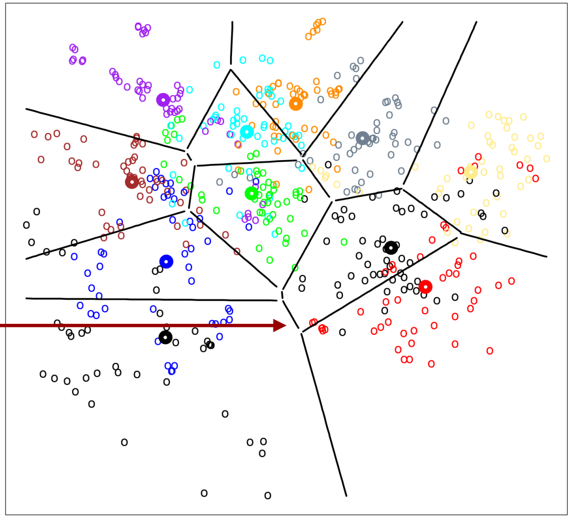
- logistic regression
  - model
  - loss function
  - optimization
- probabilistic classification
  - maximum-likelihood interpretation
  - **multi-class classification**

Classification problem

dataset of inputs $x^{(n)} \in \mathbb{R}^D$

and discrete targets $y^{(n)} \in \{0, \ldots, C\}$ multi-class

binary classification $y^{(n)} \in \{0, 1\}$



linear classification:

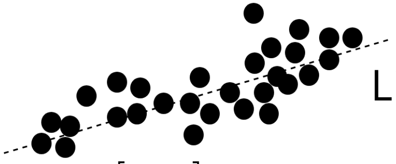
decision boundaries are linear

linear decision boundary $w^T x + b$

how do we find these boundaries?

different approaches give different linear classifiers

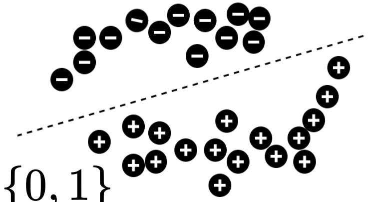
Linear to Logistic regression



$y \in [0, 1]$

Linear regression

Using linear regression to do classification



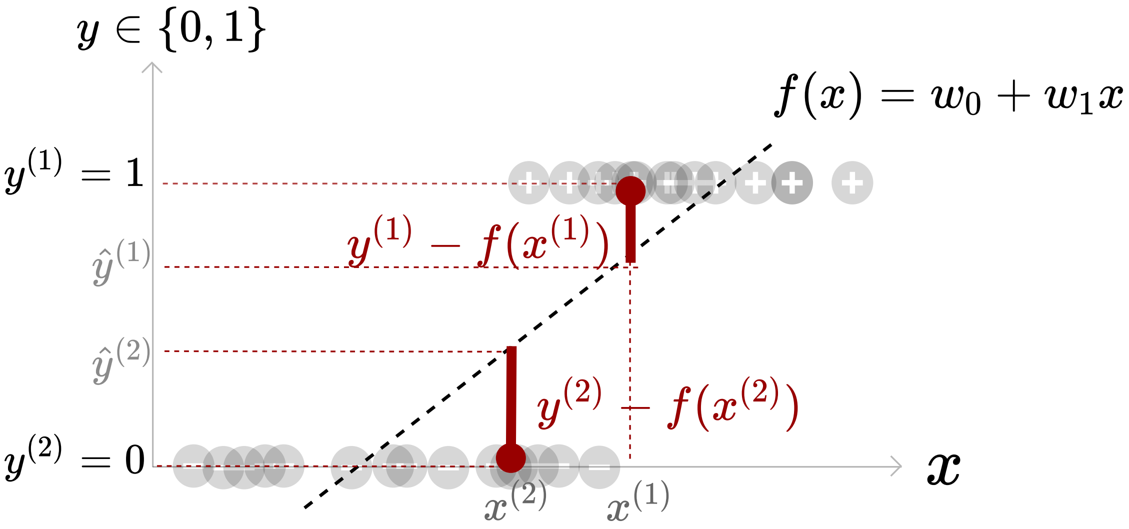
$y \in \{0, 1\}$

Today: logistic regression

A linear classifier!

Using linear regression

Use 1 and 0 as the
target value directly
apply linear regression



square error **loss**
(a.k.a. **L2** loss)

$L(y, \hat{y}) \triangleq (y - \hat{y})^2$

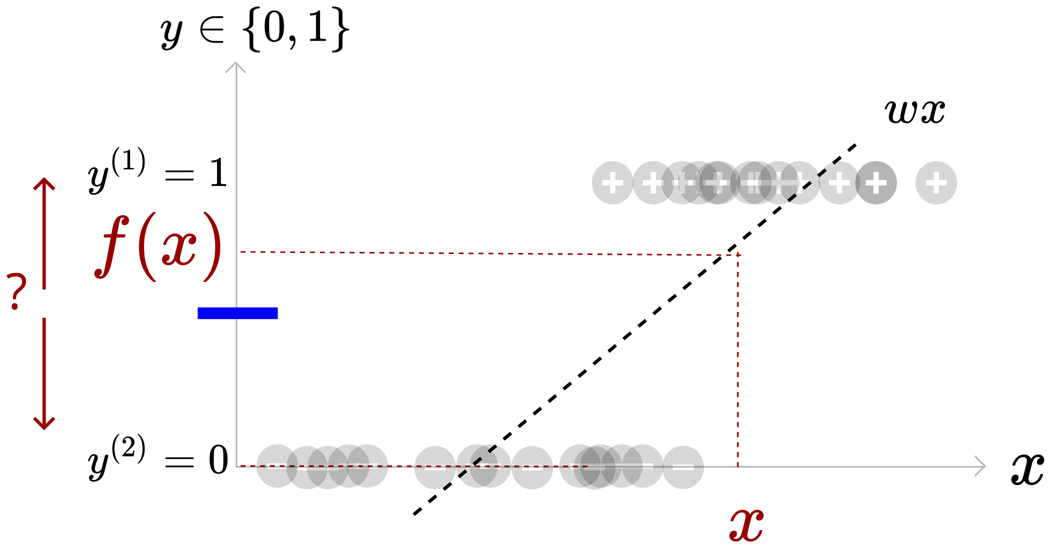
$w^* = \arg\min_w \frac{1}{2} \sum_{n=1}^{N} (w^T x^{(n)} - y^{(n)})^2$

Using linear regression

Use 1 and 0 as the
target value directly
apply linear regression

How to get a binary output?

- Threshold
- Interpret output as probability



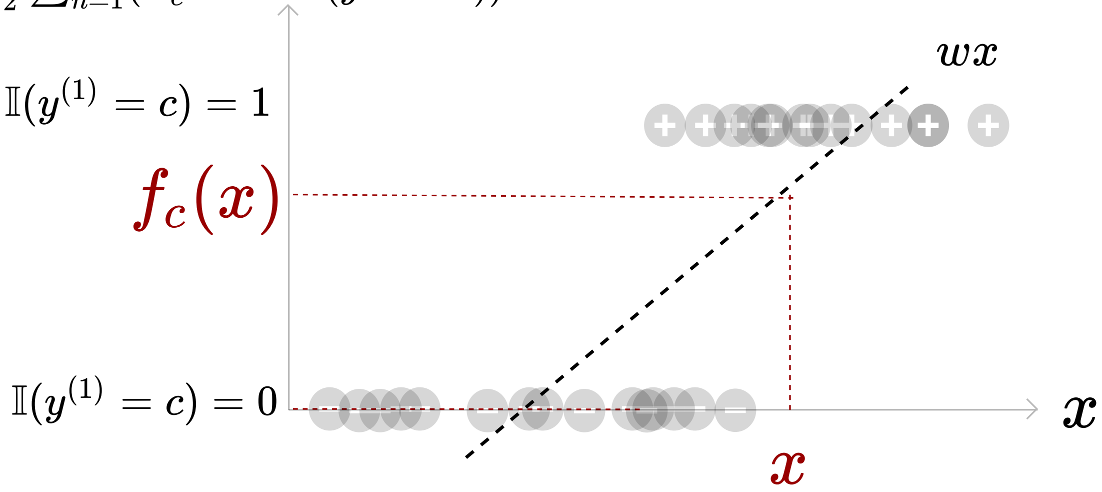
$$y = \mathbb{I}(f(x) > 0.5)$$

Using linear regression

more than one class? $y \in \{0, 1, \ldots, C\}$ fit a linear model to each class:

$w_c^* = \arg\min_{w_c} \frac{1}{2} \sum_{n=1}^{N} (w_c^T x^{(n)} - \mathbb{I}(y^{(n)} = c))^2$

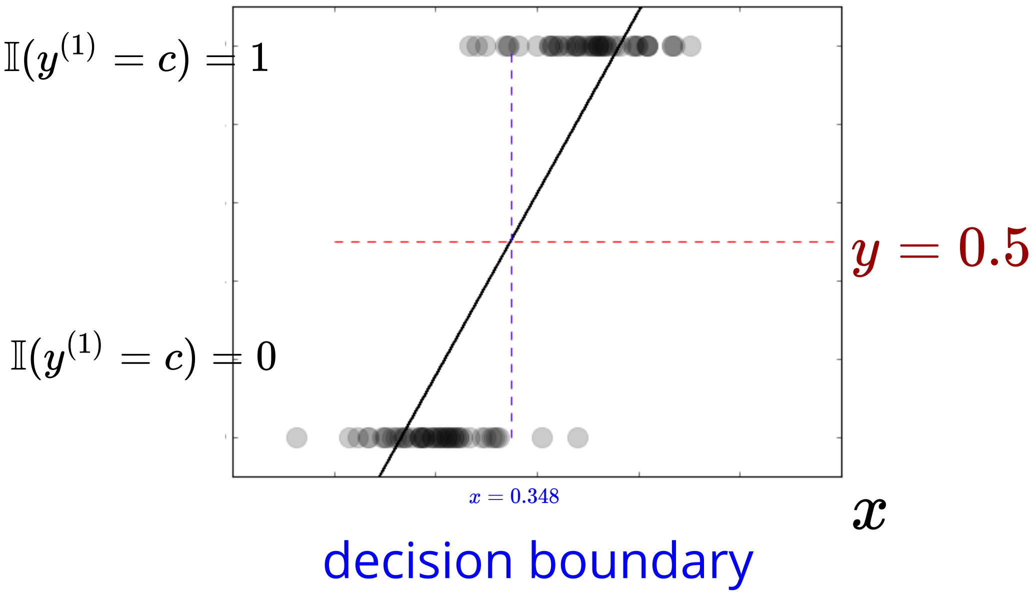
Use 1 and 0 as the target value directly apply linear regression, only one class maps to one, all other to zero



How to get the output class? $\hat{y}^{(n)} = \arg\max_c f_c(x) = \arg\max_c w_c^T x^{(n)}$

Using linear regression

Example, D=1



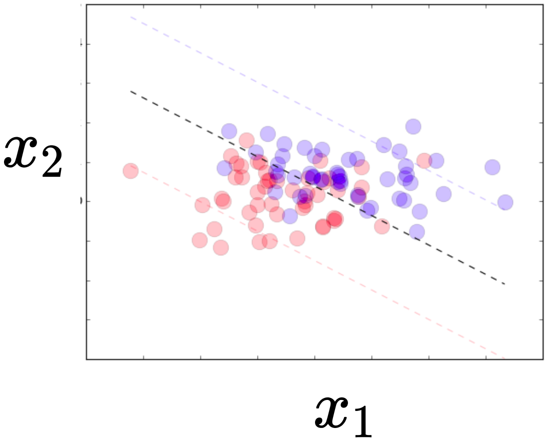
Decision boundary is a **D-1** hyperplane,

here a constant (x=0.348)

Using linear regression

Example, D=2

Decision boundary is a D-1 hyperplane, here a line

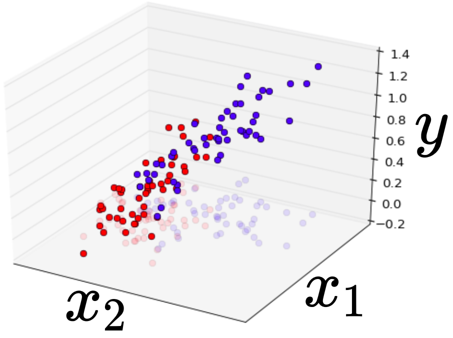


$$f_1(x) = f_0(x)$$

decision boundary

$$w_1^T x = w_2^T x$$

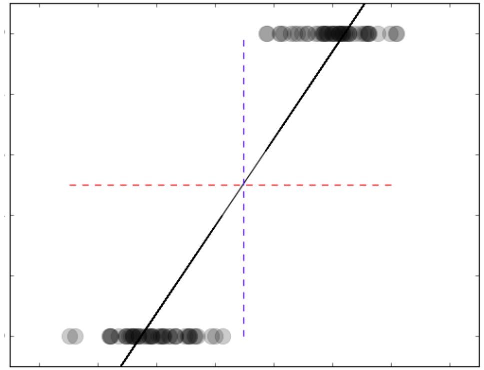
$$(w_1 - w_2)^T x = 0$$



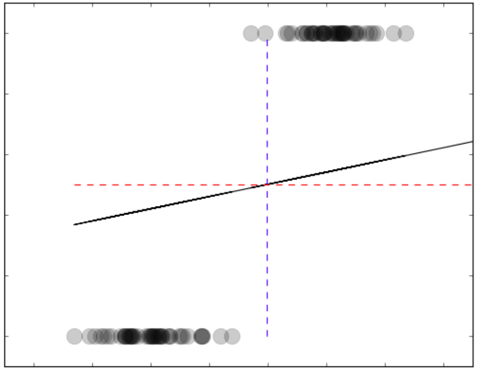
Sensitivity to outliers

Outliers can dominate the sum of least squares

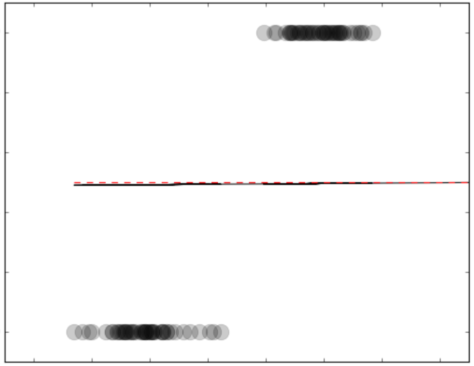
no outlier



one outlier at x10



one outlier at x100



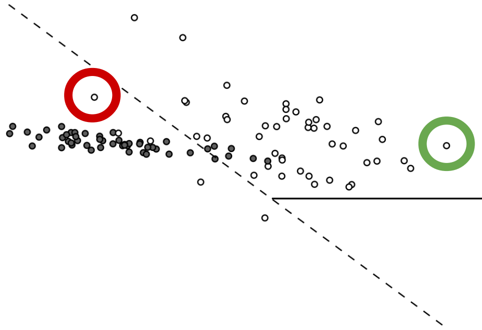
Penalizing correct predictions

Binary classification $y \in \{0, 1\}$

linear decision boundary

$$w_0^T x - w_1^T x = 0$$

$$(w_1 - w_0)^T x = 0$$



$$w^T x = 0 \begin{cases} y = 1 & w^T x > 0 \\ y = 0 & w^T x < 0 \end{cases}$$

so one weight vector is enough

L2 loss is ***the problem***: correct prediction can have higher loss than the incorrect one!

$$\begin{cases} w^T x^{(n)} = 100, y^{(n)} = 1 \\ w^T x^{(n)} = -2, y^{(n)} = 1 \end{cases}$$

➡ correct prediction

➡ lower loss $(w^T x^{(n)} - y^{(n)})^2$

☹

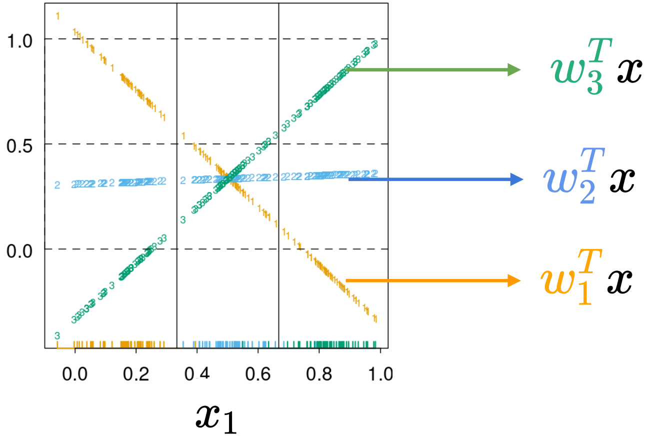
Using linear regression

fit a linear model to each class c: $w_c^* = \arg\min_{w_c} \frac{1}{2} \sum_{n=1}^{N} (w_c^T x^{(n)} - \mathbb{I}(y^{(n)} = c))^2$

class label for a new instance is then $\hat{y}^{(n)} = \arg\max_c w_c^T x^{(n)}$

decision boundary between any two classes $w_c^T x = w_{c'}^T x$

example



recall

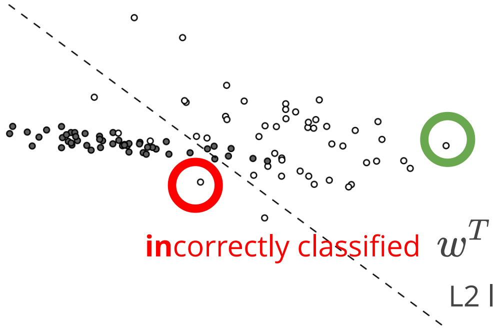
$x = [1, x_1]^T$

where are the decision boundaries?
but the instances are linearly separable
we should be able to find these boundaries

where is the problem?

first idea Using linear regression

Binary classification $y \in \{0, 1\}$ so we are fitting 2 linear models $a^T x, b^T x$



correctly classified $w^T x^{(n)} = 100 > 0$

L2 loss due to this instance: $(100 - 1)^2 = 99^2$

incorrectly classified $w^T x^{(n')} = -2 < 0$

L2 loss due to this instance: $(-2 - 1)^2 = 9$

correct prediction can have higher loss than the incorrect one! ☹

solution: we should try squashing all positive instance together and all the negative ones together

Logistic function

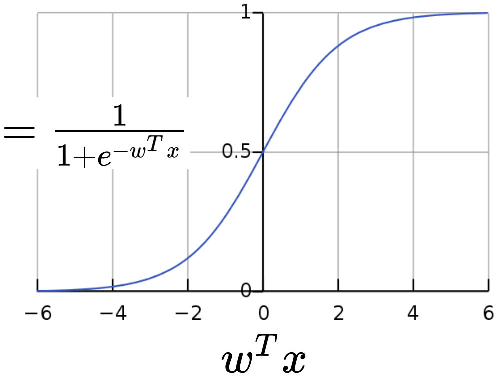
Idea: apply a squashing function to $w^T x \to \sigma(w^T x)$

desirable property of $\sigma : \mathbb{R} \to \mathbb{R}$

all $w^T x > 0$ are squashed close together
all $w^T x < 0$ are squashed together

logistic function

$$\sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$$



the boundary remains linear because

$$w^T x > 0 \Leftrightarrow \sigma(w^T x) > \frac{1}{2}$$

the decision boundary is

$$w^T x = 0 \Leftrightarrow \sigma(w^T x) = \frac{1}{2}$$

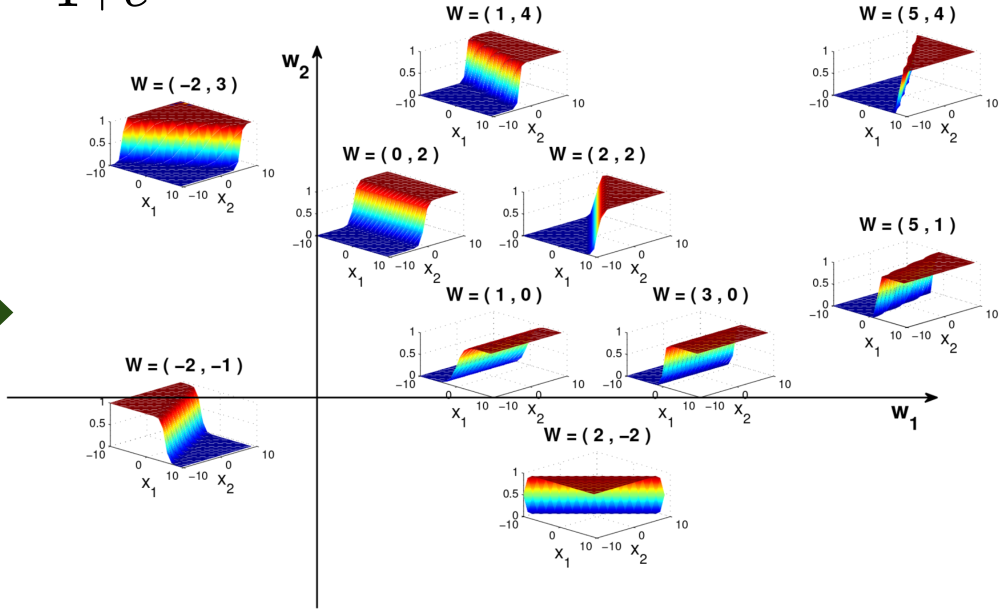
Logistic regression: model

$$f_w(x) = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$$

logistic function
squashing function
activation function

z
logit

note the linear decision boundary



Generally, $\sigma(w^T x)$ has a linear decision boundary for any monotonically increasing $\sigma : \mathbb{R} \to \mathbb{R}$

Loss functions for linear classification

Ideal loss function $L_{0-1}(y, w^T x) = \mathbb{I}(y = \text{sign}(w^T x))$

hard to optimize, why?

how about squared error?

$$L_{SE}(y, w^T x) = \frac{1}{2}(y - w^T x)^2$$

we saw that it penalizes even correct predictions

squared error+ logistic

$$L_{SE+\text{logistic}}(y, w^T x) = \frac{1}{2}(y - \sigma(w^T x))^2$$

non-convex: tricky to optimize

cross entropy loss

$$L_{CE}(y, w^T x) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

$\hat{y} = \sigma(w^T x)$

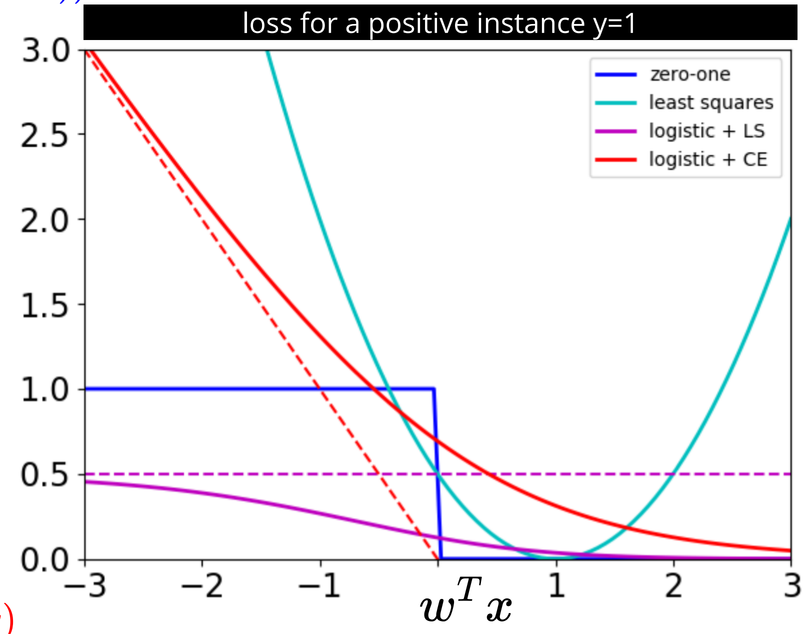


image: Gross, Farahmand, Carrasquilla

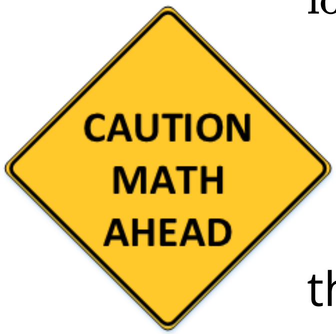
simplifying the **Cost function**

$$L_{CE}(y,\hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \qquad \hat{y} = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$$

$$J(w) = \sum_{n=1}^{N} -y^{(n)} \log(\sigma(w^T x^{(n)})) - (1-y^{(n)}) \log(1-\sigma(w^T x^{(n)}))$$

$$\log\left(\frac{1}{1+e^{-w^T x}}\right) = -\log\left(1+e^{-w^T x}\right)$$

$$\log\left(1-\frac{1}{1+e^{-w^T x}}\right) = \log\left(\frac{1}{1+e^{w^T x}}\right) = -\log\left(1+e^{w^T x}\right)$$



therefore

$$J(w) = \sum_{n=1}^{N} y^{(n)} \log\left(1+e^{-w^T x}\right) + (1-y^{(n)}) \log\left(1+e^{w^T x}\right)$$

image: https://am2.co/2016/04/convert-float-using-scientific-notation/

implementing the **Cost function**

$$J(w) = \sum_{n=1}^{N} y^{(n)} \log\left(1 + e^{-w^T x}\right) + (1 - y^{(n)}) \log\left(1 + e^{w^T x}\right)$$

```
def cost(w, # N
         X, # N x D
         y  # N
        ):
  z = np.dot(X,w) #N x 1
  J = np.mean( y * np.log1p(np.exp(-z)) + (1-y) * np.log1p(np.exp(z)) )
  return J
```

why not `np.log(1 + np.exp(-z)) ?`

```
In [3]: np.log(1+1e-100)
Out[3]: 0.0
In [4]: np.log1p(1e-100)
Out[4]: 1e-100
```

for small ϵ, $\log(1+\epsilon)$ suffers from floating point inaccuracies

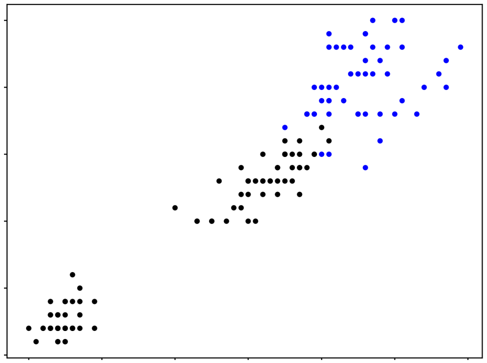
$$\log(1+\epsilon) = \epsilon - \frac{x^2}{2} + \frac{x^3}{3} - \ldots$$

Example: binary classification

classification on **Iris flowers dataset**:

(a classic dataset originally used by Fisher)

$N_c = 50$ samples with D=4 features, for each of C=3 species of Iris flower



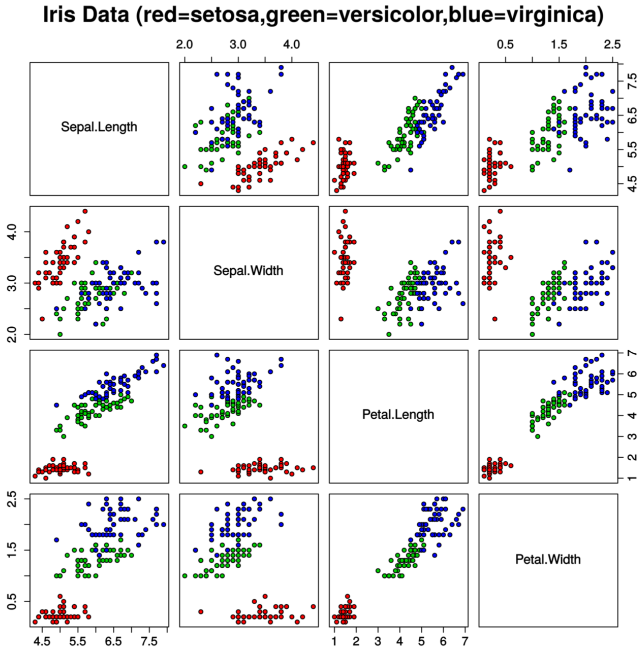
our setting

2 classes

(blue vs others)

1 features

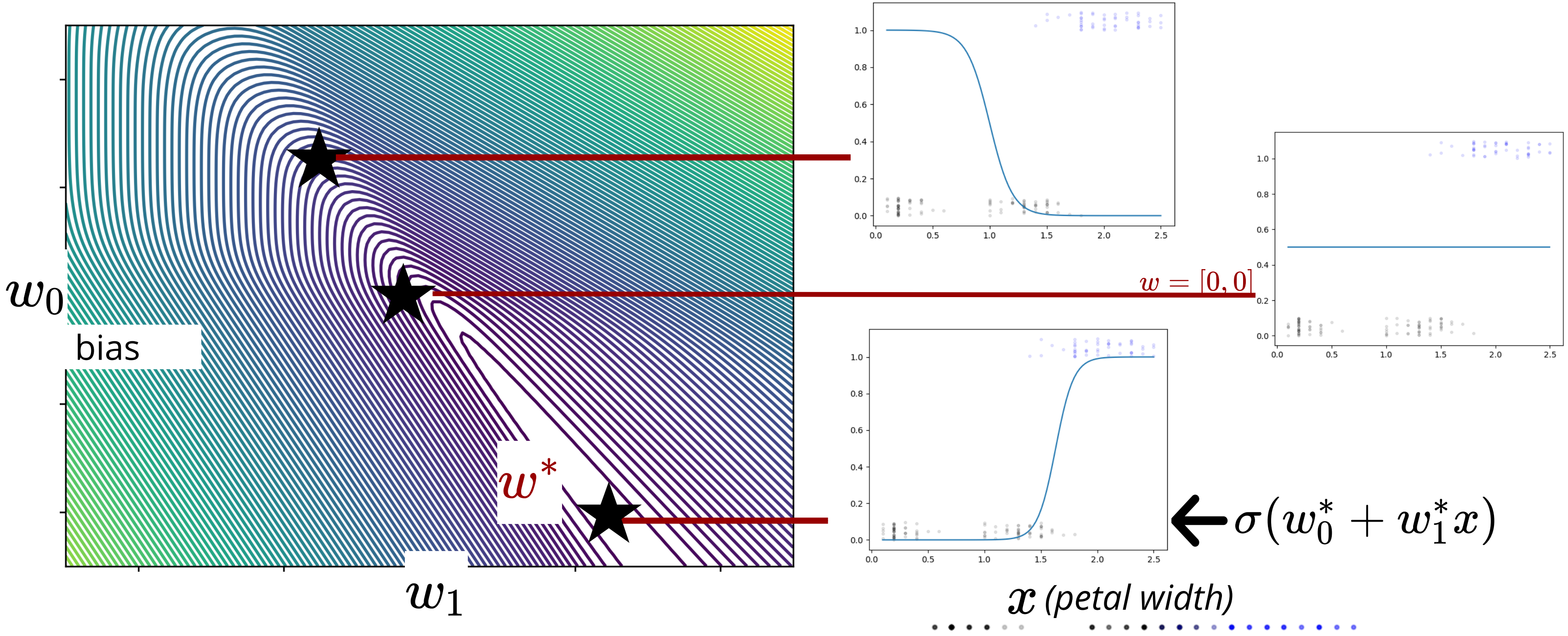
(petal width + bias)



Example: binary classification

we have two weights associated with bias + petal width

$J(w)$ as a function of these weights



Gradient

how did we find the optimal weights?

(in contrast to linear regression, no closed form solution)

cost: $J(w) = \sum_{n=1}^{N} y^{(n)} \log\left(1 + e^{-w^T x^{(n)}}\right) + (1 - y^{(n)}) \log\left(1 + e^{w^T x^{(n)}}\right)$

taking partial derivative $\frac{\partial}{\partial w_d} J(w) = \sum_n -y^{(n)} x_d^{(n)} \frac{e^{-w^T x^{(n)}}}{1+e^{-w^T x^{(n)}}} + x_d^{(n)} (1 - y^{(n)}) \frac{e^{w^T x^{(n)}}}{1+e^{w^T x^{(n)}}}$

$$= \sum_n -x_d^{(n)} y^{(n)} (1 - \hat{y}) + x_d^{(n)} (1 - y^{(n)}) \hat{y} = x_d^{(n)} (y - \hat{y})$$

gradient $\nabla J(w) = \sum_n x^{(n)} (y^{(n)} - \boxed{\hat{y}}) \longrightarrow \sigma(w^T x^{(n)})$

$w^T x^{(n)}$

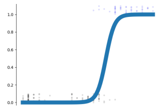
compare to gradient for linear regression $\nabla J(w) = \sum_n x^{(n)} (y^{(n)} - \boxed{\hat{y}})$

Probabilistic view of logistic regression

probabilistic interpretation of logistic regression $\hat{y} = p_w(y=1 \mid x) = \frac{1}{1+e^{-w^T x}} = \sigma(w^T x)$

logit function is the inverse of logistic $\log \frac{\hat{y}}{1-\hat{y}} = w^T x$

the log-ratio of class probabilities is linear



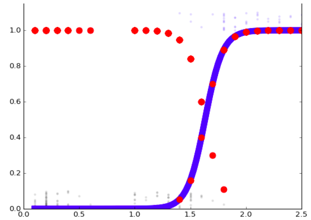
likelihood probability of data as a function of model parameters

$$L^{(n)}(w) = p_w(y^{(n)} \mid x^{(n)}) = \text{Bernoulli}(y^{(n)}; \sigma(w^T x^{(n)})) = {\hat{y}^{(n)}}^{y^{(n)}} (1-\hat{y}^{(n)})^{1-y^{(n)}}$$

is a function of w

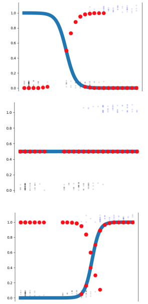
not a probability distribution function

$\hat{y}^{(n)}$ is the probability of $y^{(n)} = 1$



likelihood of the dataset $L(w) = \prod_{n=1}^{N} p_w(y^{(n)} \mid x^{(n)}) = \prod_{n=1}^{N} {\hat{y}^{(n)}}^{y^{(n)}} (1-\hat{y}^{(n)})^{1-y^{(n)}}$

Maximum likelihood & logistic regression



likelihood $L(w) = \prod_{n=1}^{N} p_w(y^{(n)} \mid x^{(n)}) = \prod_{n=1}^{N} \hat{y}^{(n)\, y^{(n)}} (1 - \hat{y}^{(n)})^{1-y^{(n)}}$

maximum likelihood use the model that maximizes the likelihood of observations

$$w^* = \arg\max_w L(w)$$

likelihood value blows up for large N, work with log-likelihood instead (same maximum)

log likelihood $\max_w \sum_{n=1}^{N} \log p_w(y^{(n)} \mid x^{(n)})$

$$= \max_w \sum_{n=1}^{N} y^{(n)} \log(\hat{y}^{(n)}) + (1 - y^{(n)}) \log(1 - \hat{y}^{(n)})$$

$= \min_w J(w)$ the cross entropy cost function!

so using cross-entropy loss in logistic regression is maximizing conditional likelihood

Maximum likelihood & linear regression

squared error loss also has max-likelihood interpretation

cond. probability $p_w(y \mid x) = \mathcal{N}(y \mid w^T x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-w^T x)^2}{2\sigma^2}}$

mean μ

σ^2 variance

σ standard deviation

(don't confuse with logistic function)

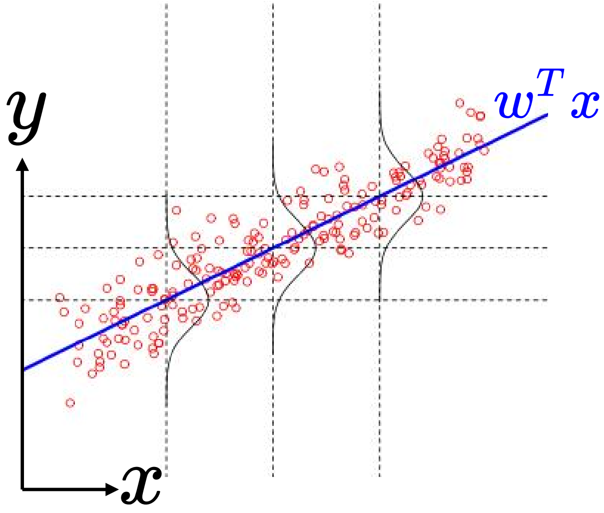
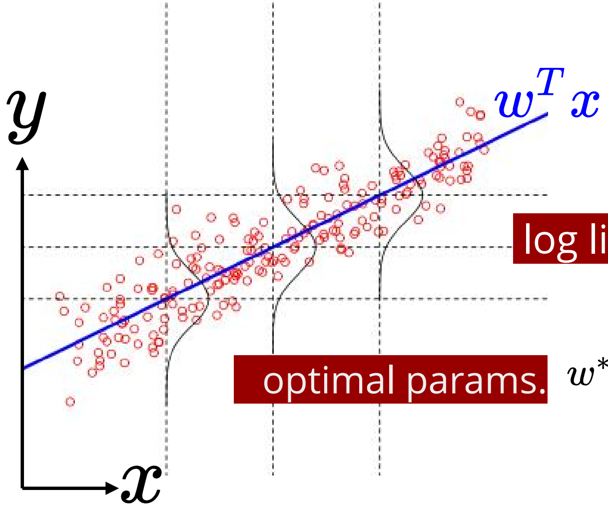


image: http://blog.nguyenvq.com/blog/2009/05/12/linear-regression-plot-with-normal-curves-for-error-sideways/

Maximum likelihood & linear regression

squared error loss also has max-likelihood interpretation

cond. probability $p_w(y \mid x) = \mathcal{N}(y \mid w^T x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-w^T x)^2}{2\sigma^2}}$



likelihood $L(w) = \prod_{n=1}^{N} p_w(y^{(n)} \mid x^{(n)})$

log likelihood $\ell(w) = \sum_n -\frac{1}{2\sigma^2}(y^{(n)} - w^T x^{(n)})^2 + \text{constants}$

optimal params. $w^* = \arg\max_w \ell(w) = \arg\min_w \frac{1}{2} \sum_n (y^{(n)} - w^T x^{(n)})^2$

linear least squares!

image: http://blog.nguyenvq.com/blog/2009/05/12/linear-regression-plot-with-normal-curves-for-error-sideways/

Multiclass classification

binary classification: Bernoulli likelihood:

$$\text{Bernoulli}(y \mid \hat{y}) = \hat{y}^y (1-\hat{y})^{1-y} \xrightarrow{\text{subject to}} \hat{y} \in [0,1]$$

$$\begin{cases} \hat{y} & y = 1 \\ 1-\hat{y} & y = 0 \end{cases}$$

using logistic function to ensure this $\hat{y} = \sigma(z) = \sigma(w^T x)$

$$\begin{cases} \hat{y}_1 & y = 1 \\ \hat{y}_2 & y = 2 \\ \dots & \\ \hat{y}_C & y = C \end{cases}$$

C classes: categorical likelihood

$$\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{I}(y=c)} \xrightarrow{\text{subject to}} \sum_c \hat{y}_c = 1$$

how to enforce it?

achieved using softmax function

Softmax

generalization of logistic to > 2 classes:

- **logistic**: $\sigma : \mathbb{R} \to (0,1)$ produces a single probability
  - probability of the second class is $(1 - \sigma(z))$

- **softmax:** $\mathbb{R}^C \to \Delta_C$ probability simplex $p \in \Delta_c \to \sum_{c=1}^{C} p_c = 1$

$$\hat{y}_c = \mathrm{softmax}(z)_c = \frac{e^{z_c}}{\sum_{c'=1}^{C} e^{z_{c'}}} \quad \text{so} \quad \sum_c \hat{y} = 1$$

if input values are large, softmax becomes similar to argmax

example $\mathrm{softmax}([10, 100, -1]) \approx [0, 1, 0]$

so similar to logistic this is also a squashing function

numerical stability

```
def softmax(
  z # C x ... array
  ):
  z0 = z - np.max(z,0)
  yh = np.exp(z)
  yh /= np.sum(yh, 0)
  return yh
```

Multiclass classification

C classes: categorical likelihood

$\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{I}(y=c)}$ using softmax to enforce sum-to-one constraint

$$\hat{y}_c = \text{softmax}([w_{[1]}{}^T x, \ldots, w_{[C]}{}^T x])_c = \frac{e^{w_{[c]}{}^T x}}{\sum_{c'} e^{w_{[c']}{}^T x}}$$

so we have on parameter vector for each class

to simplify equations we write $z_c = w_{[c]}{}^T x$

$$\hat{y}_c = \text{softmax}([z_1, \ldots, z_C])_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}}$$

Likelihood

C classes: categorical likelihood

$\mathrm{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{I}(y=c)}$ using softmax to enforce sum-to-one constraint

$$\hat{y}_c = \mathrm{softmax}([z_1, \ldots, z_C])_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}}$$ where $z_c = {w_{[c]}}^T x$

substituting softmax in Categorical likeihood:

likelihood $L(\{w_c\}) = \prod_{n=1}^{N} \prod_{c=1}^{C} \mathrm{softmax}([z_1^{(n)}, \ldots, z_C^{(n)}])_c^{\mathbb{I}(y^{(n)}=c)}$

$$= \prod_{n=1}^{N} \prod_{c=1}^{C} \left(\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}} \right)^{\mathbb{I}(y^{(n)}=c)}$$

One-hot encoding

likelihood $L(\{w_c\}) = \prod_{n=1}^{N} \prod_{c=1}^{C} \left(\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}} \right)^{\mathbb{I}(y^{(n)}=c)}$

log-likelihood $\ell(\{w_c\}) = \sum_{n=1}^{N} \sum_{c=1}^{C} \mathbb{I}(y^{(n)} = c) z_c^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}}$

one-hot encoding for labels

$$y^{(n)} \rightarrow [\mathbb{I}(y^{(n)} = 1), \ldots, \mathbb{I}(y^{(n)} = C)]$$

```python
def one_hot(
  y, #vector of size N class-labels [1,...,C]
):
  N, C = y.shape[0], np.max(y)
  y_hot = np.zeros(N, C)
  y_hot[np.arange(N), y-1] = 1
  return y_hot
```

using this encoding from now on

log-likelihood $\ell(\{w_c\}) = \sum_{n=1}^{N} {y^{(n)}}^{T} z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}}$

One-hot encoding

side note

we can also use this encoding for categorical **inputs** features

one-hot encoding for input features

$$x_d^{(n)} \to [\mathbb{I}(x_d^{(n)} = 1), \ldots, \mathbb{I}(x_d^{(n)} = C)]$$

problem

these features are **not** linearly independent, why?

might become an issue for *linear regression.* why?

solution

remove one of the one-hot encoding features

$$x_d^{(n)} \to [\mathbb{I}(x_d^{(n)} = 1), \ldots, \mathbb{I}(x_d^{(n)} = C - 1)]$$

Implementing the cost function

softmax cross entropy cost function is the negative of the log-likelihood

similar to the binary case

$$J(\{w_c\}) = -\sum_{n=1}^{N} {y^{(n)}}^T z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}} \quad \text{where} \quad z_c = {w_{[c]}}^T x$$

naive implementation of log-sum-exp causes over/underflow

prevent this using the following trick:

$$\log \sum_c e^{z_c} = \bar{z} + \log \sum_c e^{z_c - \bar{z}}$$

$$\bar{z} \leftarrow \max_c z_c$$

```
def cost(X, # NxD design matrix
         y, # N labels in {1,...,C}
         W # C x D: one weight vector per class
        ):

  Z = np.dot(X, W.T) # N x C
  Y = onehot(y) # N x C
  nll = - np.sum( np.dot(Z, Y.T) - logsumexp(Z))
  return nll
```

```
def logsumexp(
  Z# C x N
):
  Zmax = np.max(Z,0)[None,:]
  lse = Zmax + np.log(np.sum(np.exp(Z - Zmax)))
  return lse
```

Optimization

given the training data $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_n$

find the best model parameters $\{w_{[c]}\}_c$

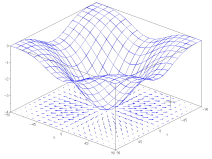
by minimizing the cost (maximizing the likelihood of $\mathcal{D}$)

$$J(\{w_c\}) = -\sum_{n=1}^{N} {y^{(n)}}^T z^{(n)} + \log \sum_{c'} e^{z_{c'}^{(n)}} \quad \text{where} \quad z_c = {w_{[c]}}^T x$$

need to use gradient descent (for now calculate the gradient)

$$\nabla J(w) = \underbrace{[\frac{\partial}{\partial w_{[1],1}} J, \ldots \frac{\partial}{\partial w_{[1],D}} J, \ldots, \frac{\partial}{\partial w_{[C],D}} J]^T}_{\text{length } C \times D}$$

Gradient



need to use gradient descent (for now calculate the gradient)

$$J(\{w_c\}) = -\sum_{n=1}^{N} {y^{(n)}}^T z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}} \quad \text{where} \quad z_c = {w_{[c]}}^T x$$

using chain rule

$$\frac{\partial}{\partial w_{[c],d}} J = \sum_{n=1}^{N} \underbrace{\frac{\partial J}{\partial z_c^{(n)}}}_{-y_c^{(n)} + \frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z^{(n)}_{c'}}}} \underbrace{\frac{\partial z_c^{(n)}}{\partial w_{[c],d}}}_{x_d^{(n)}} = -\sum_n (y_c^{(n)} - \hat{y}_c^{(n)}) x_d^{(n)}$$

this looks familiar!

(note the negative sign)

$$\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z^{(n)}_{c'}}} \rightarrow \hat{y}_c^{(n)}$$

so the derivative of log-sum-exp is softmax

Summary

- logistic regression: logistic activation function + cross-entropy loss
  - cost function
  - probabilistic interpretation
    - using maximum likelihood to derive the cost function

Gaussian likelihood	$\Leftrightarrow$	L2 loss
Bernoulli likelihood		cross-entropy loss

- multi-class classification: softmax + cross-entropy
  - cost function
  - one-hot encoding
  - gradient calculation (will use later!)