# Applied Machine Learning

Logistic Regression

**Siamak Ravanbakhsh**

COMP 551 (winter 2020)

# Learning objectives

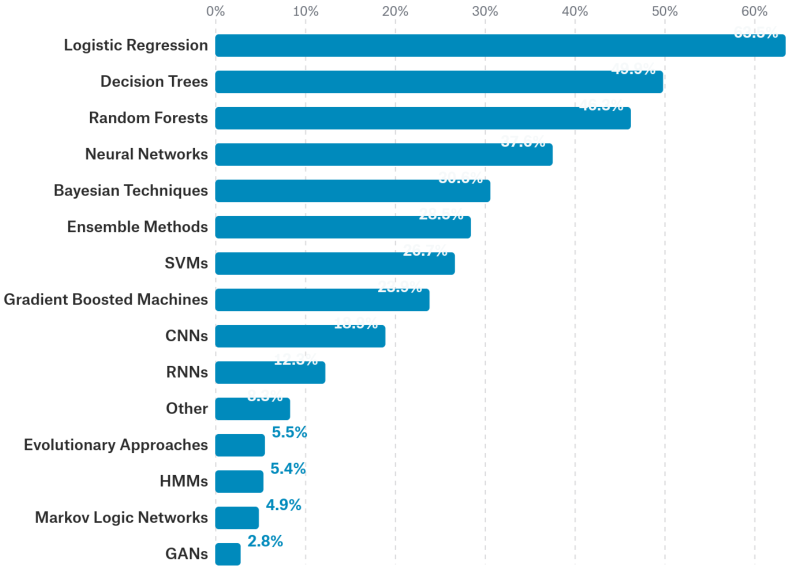
- what are linear classifiers
- logistic regression
    - model
    - loss function
- maximum likelihood view
- multi-class classification

# Motivation

we have seen KNN for classification

we see more classifiers today (linear classifiers)

Logistic Regression is **the** most commonly reported data science method used at work



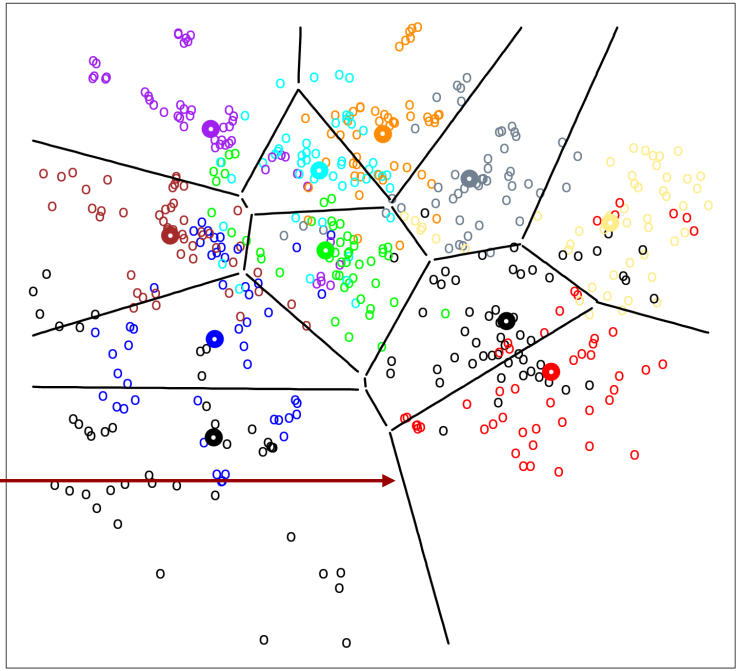

souce: 2017 Kaggle survey

# Classification problem

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

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

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



## linear classification:

decision boundaries are linear

linear decision boundary $w^\top x + b$

how do we find these boundaries?

different approaches give different linear classifiers first idea

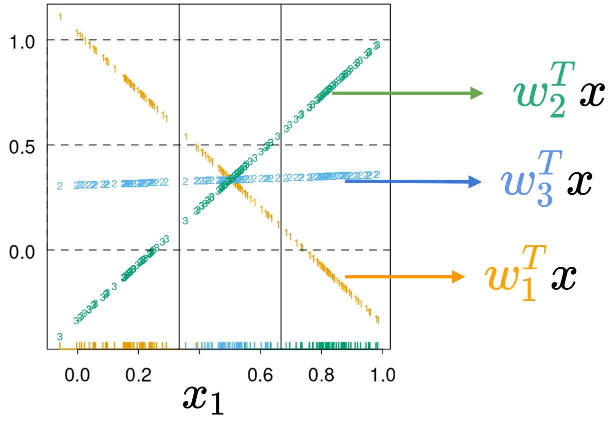
# Using linear regression

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

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

decision boundary between any two classes $w_c^\top x = w_{c'}^\top x$

example




recall

$x = [1, x_1]^\top$

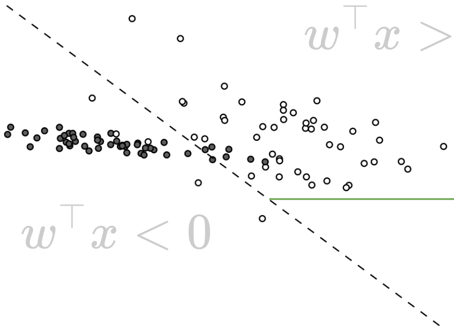
- where are the decision boundaries?
- but the instances are linearly separable
  - we should be able to find these boundaries
- where is the problem?

# first idea Using linear regression

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

$$a^\top x - b^\top x = 0$$ decision boundary is here

$$(a - b)^\top x = 0$$




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

so one weight vector is enough

# Using linear regression

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

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

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

incorrectly classified $w^\top x^{(n')} = -2 < 0$

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

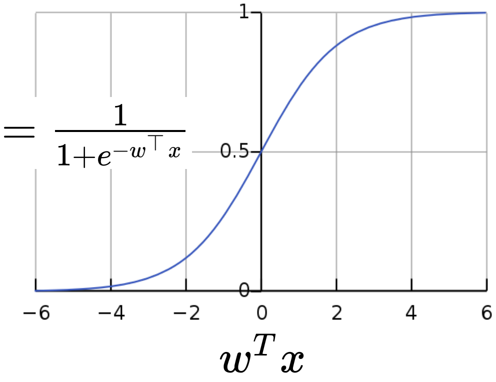
Idea: apply a squashing function to $w^\top x \to \sigma(w^\top x)$

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

- all $w^\top x > 0$ are squashed close together
- all $w^\top x < 0$ are squashed together

**logistic function** has these properties

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




the decision boundary is

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

still a linear decision boundary

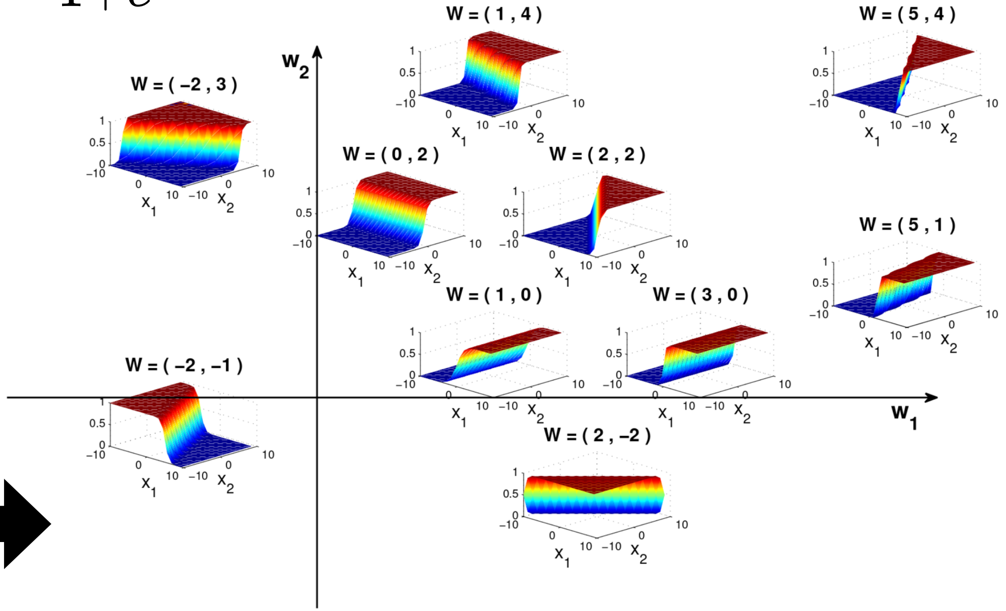
# Logistic regression: model

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

$z$
logit

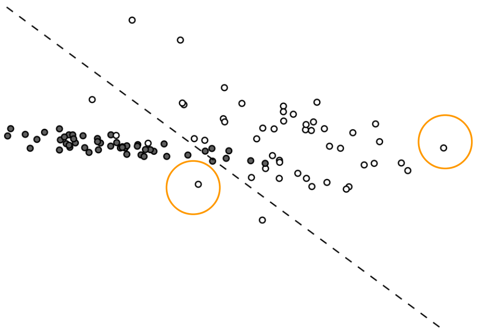
logistic function
squashing function
activation function

note the linear decision boundary



# Logistic regression: the loss

first idea use the misclassification error



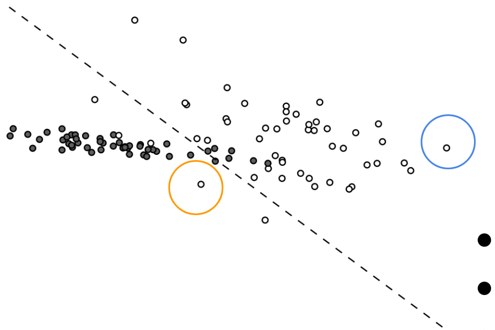
$$L_{0/1}(\hat{y}, y) = \mathbb{I}(y \neq \text{sign}(\hat{y} - \tfrac{1}{2}))$$

$\sigma(w^\top x)$

- not a continuous function (in w)
- hard to optimize

# Logistic regression: the loss

second idea use the L2 loss



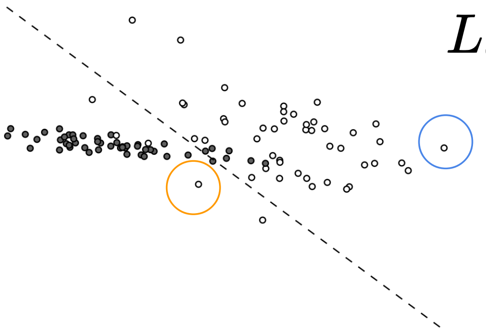
$$L_2(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$\sigma(w^\top x)$

- thanks to squashing, the previous problem is resolved
- loss is continuous
- still a problem: hard to optimize (non-convex in w)

# Logistic regression: the loss

third idea — use the cross-entropy loss



$$L_{CE}(\hat{y}, y) = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$$

$\sigma(w^\top x)$

- it is convex in w
- probabilistic interpretation (soon!)

☺

# Cost function

we need to optimize the cost wrt. parameters

**first:** simplify

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

substitute logistic function

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

substitute logistic function

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

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

## implementing the **Cost function**

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

```python
def cost(w, # D
         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)) ?`

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

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

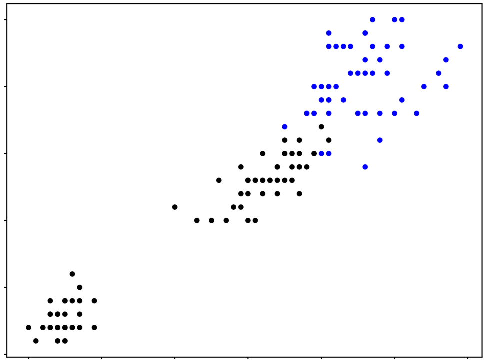
$$\log(1+\epsilon) = \epsilon - \frac{x^2}{2} + \frac{x^3}{3} - \dots$$

# 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



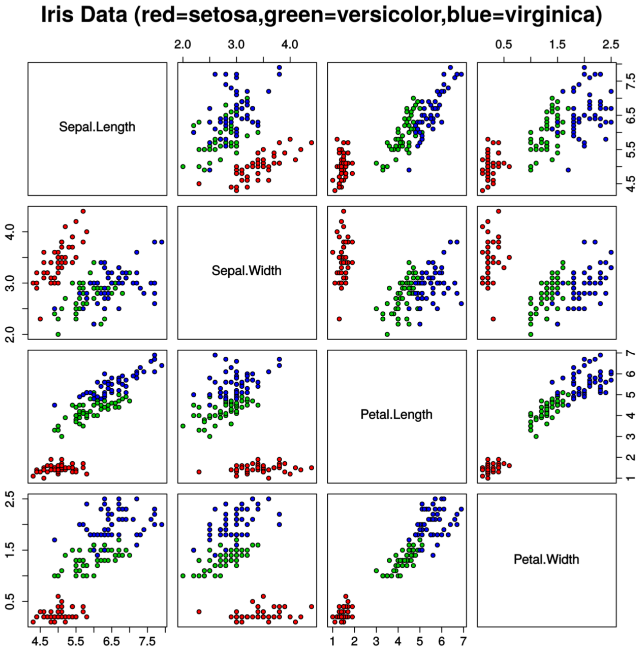
our setting

2 classes

(blue vs others)

1 features

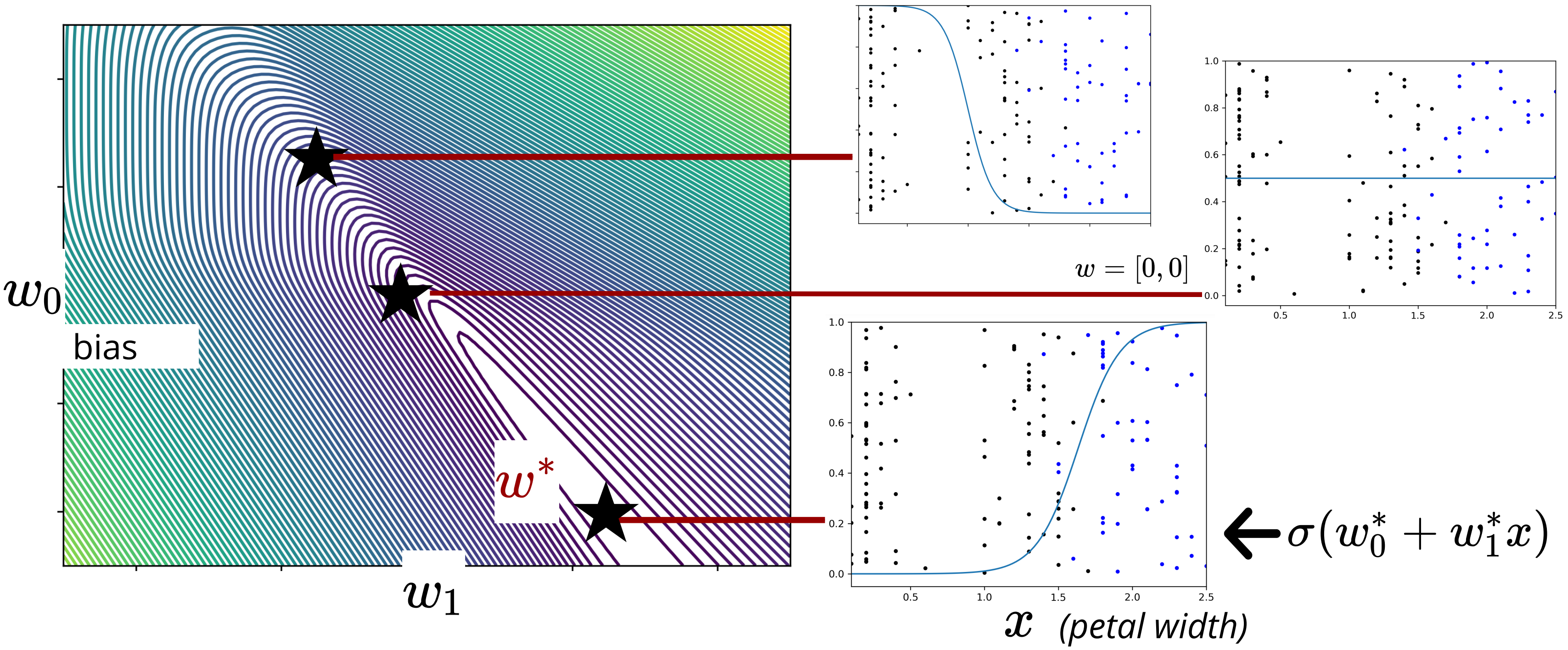
(petal width + bias)



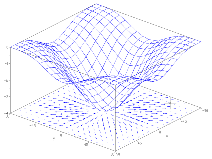
# **Example**: binary classification

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

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

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

gradient $\nabla J(w) = \sum_n x^{(n)} (\underbrace{\hat{y}^{(n)}}_{\sigma(w^\top x^{(n)})} - y^{(n)})$

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

# Probabilistic view of logistic regression

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

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

the log-ratio of class probabilities is linear

**likelihood** probability of data as a function of model parameters

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

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

mean $\mu$

$\sigma^2$ variance

$\sigma$ standard deviation

(don't confuse with logistic function)

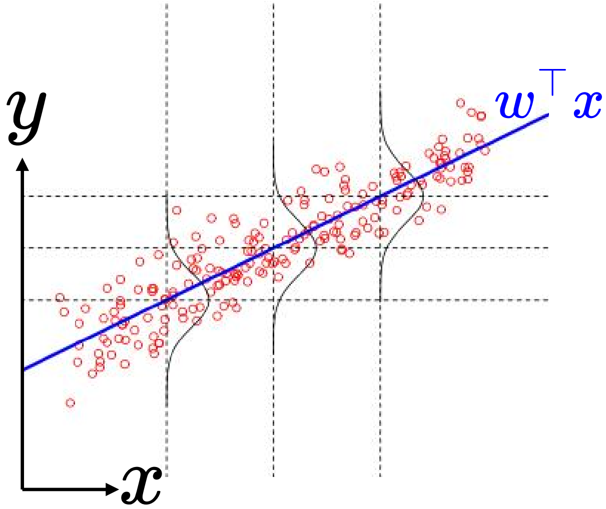
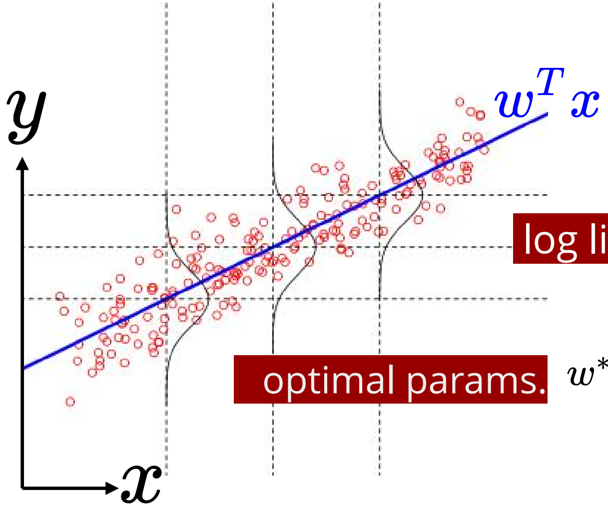



image: http://blog.nguyenvq.com/blog/2009/05/12/linear-regression-plot-with-normal-curves-for-error-sideways/

# Maximum likelihood & linear regression

squared error loss also has max-likelihood interpretation

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




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

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

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

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:

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

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

**C classes**: categorical likelihood

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

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

```python
def softmax(
  z # C x ... array
  ):
  z = 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

$$\hat{y}_c = \text{softmax}([{w_{[1]}}^\top x, \dots, {w_{[C]}}^\top x])_c = \frac{e^{{w_{[c]}}^\top x}}{\sum_{c'} e^{{w_{[c']}}^\top x}}$$

so we have on parameter vector for each class

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

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

# Likelihood

**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

$$\hat{y}_c = \text{softmax}([z_1, \dots, z_C])_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}} \quad \text{where} \quad z_c = {w_{[c]}}^\top x$$

substituting softmax in Categorical likeihood:

likelihood

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

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

# 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

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

```
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** $\ell(\{w_c\}) = \sum_{n=1}^{N} {y^{(n)}}^{\top} z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}}$

# One-hot encoding

side note

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

**one-hot encoding** for input features

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

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

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

# Implementing the cost function

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

similar to the binary case

$$J(\{w_c\}) = -\left( \sum_{n=1}^{N} {y^{(n)}}^\top z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}} \right)$$ where $z_c = {w_{[c]}}^\top x$

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.sum(Z * Y, 1) - logsumexp(Z))
  return nll
```

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

# Optimization

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

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

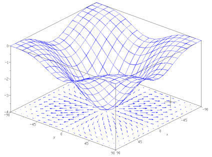
by minimizing the cost (maximizing the likelihood of $\mathcal{D}$)

$$J(\{w_c\}) = -\sum_{n=1}^{N} {y^{(n)}}^\top z^{(n)} + \log \sum_{c'} e^{z_{c'}^{(n)}} \quad \text{where} \quad z_c = {w_{[c]}}^\top x$$

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

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

# Gradient



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

$$J(\{w_c\}) = -\sum_{n=1}^{N} {y^{(n)}}^\top z^{(n)} + \log \sum_{c'} e^{z_{c'}^{(n)}} \quad \text{where} \quad z_c = {w_{[c]}}^\top x$$

using chain rule

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

this looks familiar!

$$\frac{\partial z_c^{(n)}}{\partial w_{[c],d}} \rightarrow x_d^{(n)}$$

$$\frac{\partial J}{\partial z_c^{(n)}} \rightarrow -y_c^{(n)} + \frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z^{(n)}{}_{c'}}}$$

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

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 | $\Leftrightarrow$ | cross-entropy loss |

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