DATA SCIENCE WITH MACHINE LEARNING: CLASSIFICATION

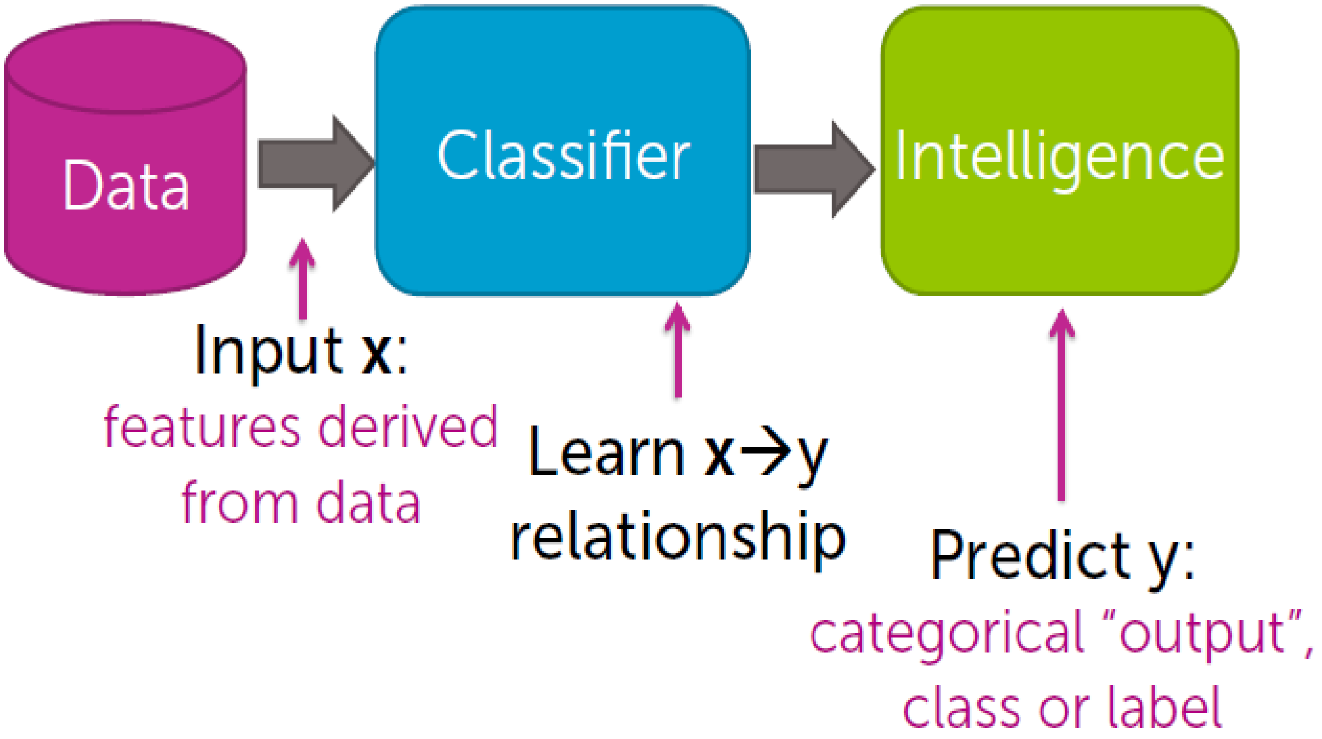
This lecture is
based on course by E. Fox and C. Guestrin, Univ of Washington

18/01/2024

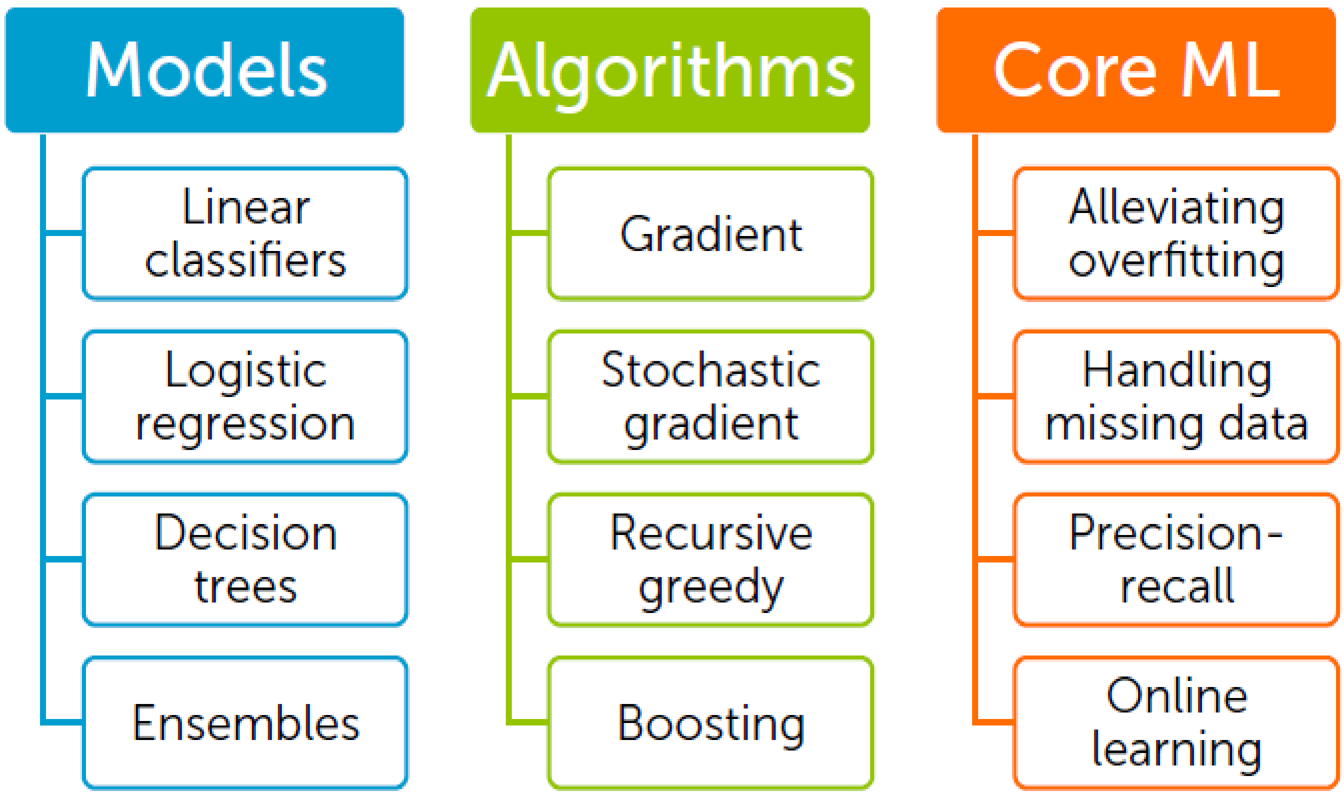
WFAiS UJ, Informatyka Stosowana
I stopień studiów

What is a classification?

From features to predictions



Overwiew of the content



Linear classifier

An inteligent restaurant review system

Positive reviews not positive about everything

Experience

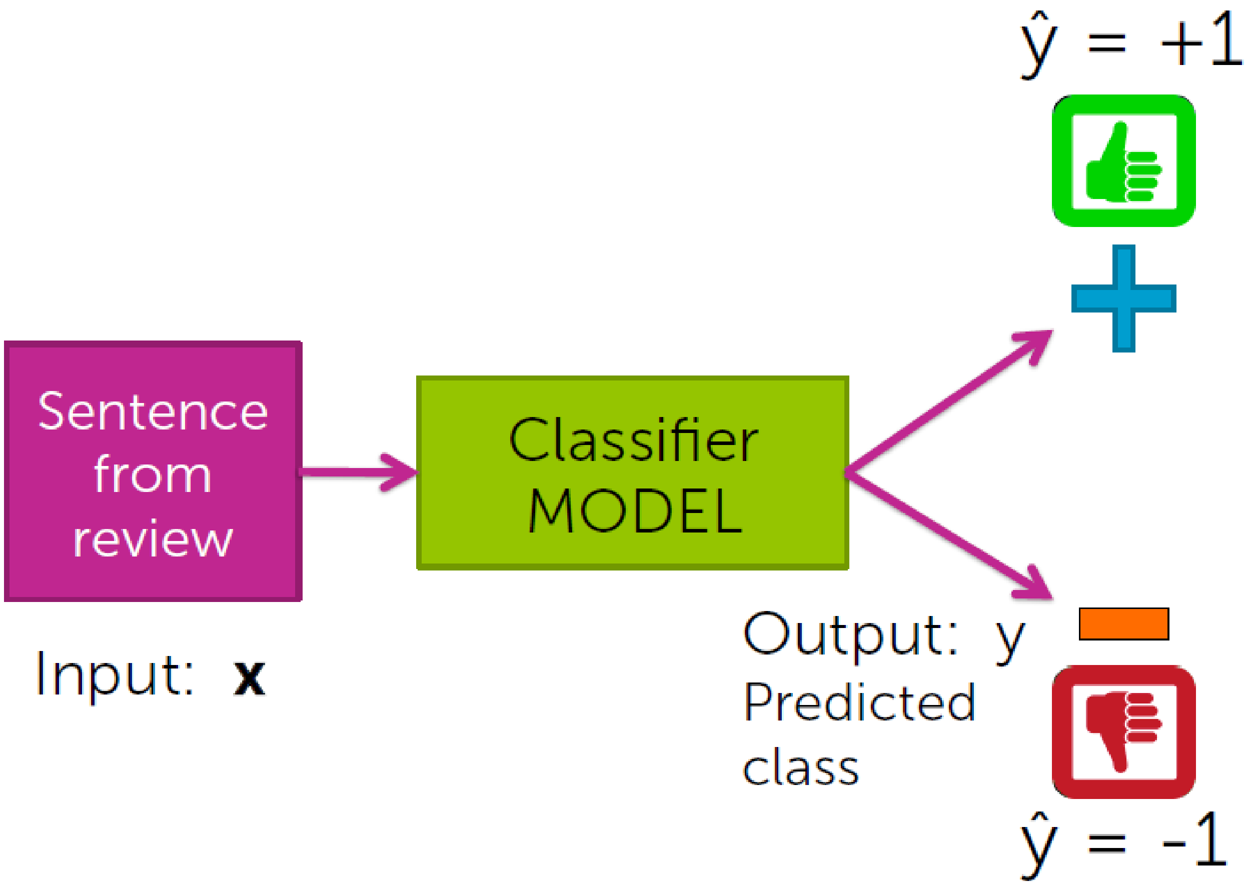
Sample review:

Watching the chefs create incredible edible art made the experience very unique.

My wife tried their ramen and it was pretty forgettable.

All the sushi was delicious! Easily best sushi in Seattle.

Classifying sentiment of review



Note: we'll start talking about 2 classes, and address multiclass later

A (linear) classifier: scoring a sentence

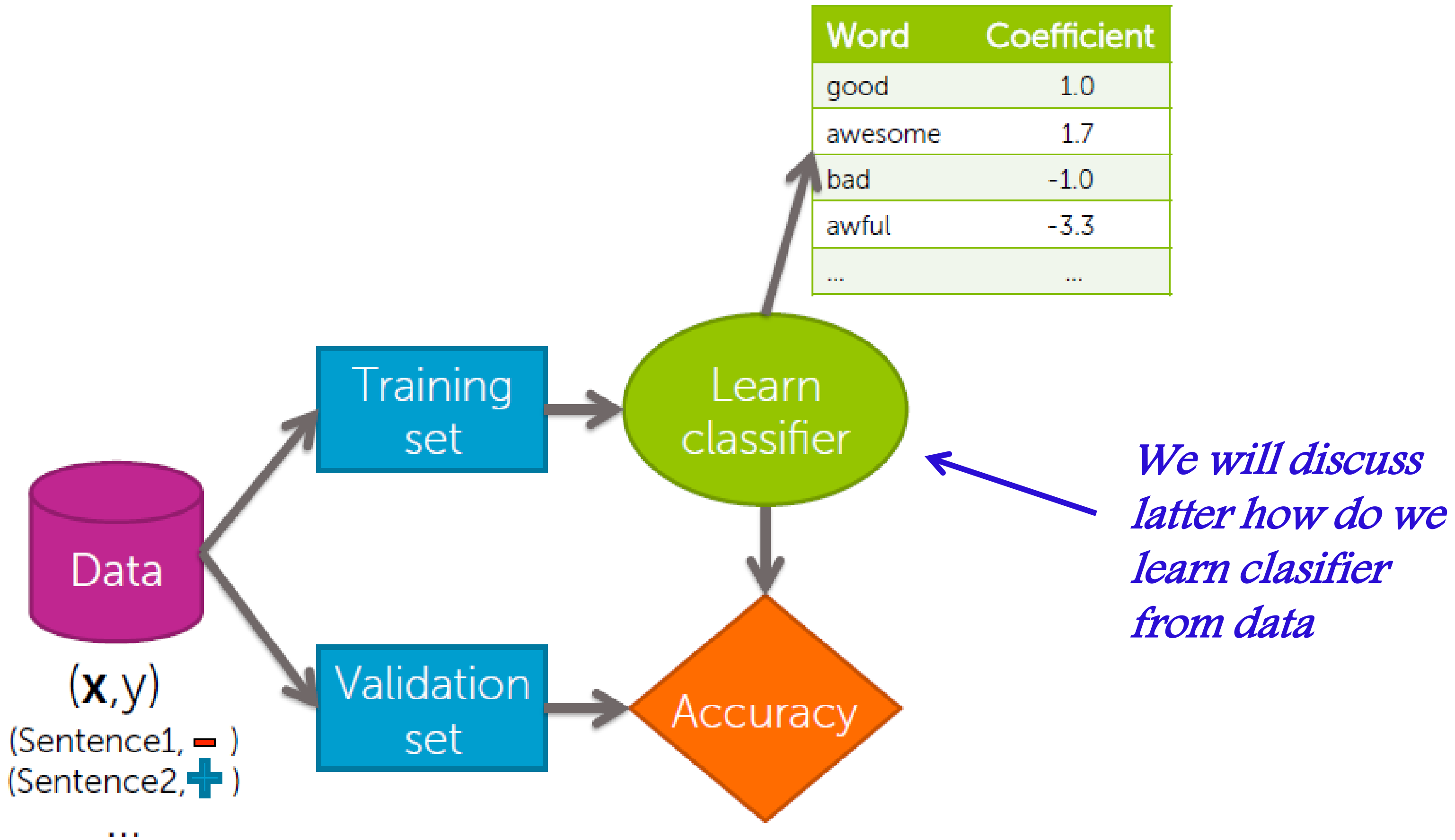
Word	Coefficient
good	1.0
great	1.2
awesome	1.7
bad	-1.0
terrible	-2.1
awful	-3.3
restaurant, the, we, where, …	0.0
…	…

Input $\mathbf{x}_i$:
Sushi was great,
the food was awesome,
but the service was terrible.

Score(xi) = 1.2+1.7 –2.1
= 0.8 >0
=> y = +1
positive review

Called a linear classifier, because output is weighted sum of input.

Training a classifier = Learning the coefficients

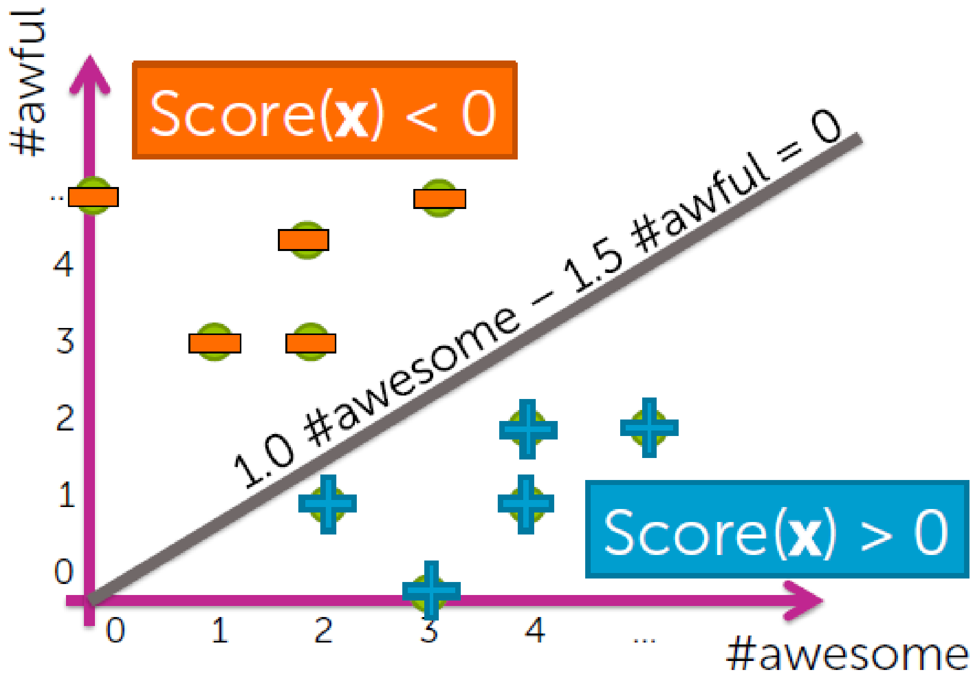


Word	Coefficient
good	1.0
awesome	1.7
bad	-1.0
awful	-3.3
...	...

Decision boundary example

Word	Coefficient
#awesome	1.0
#awful	-1.5

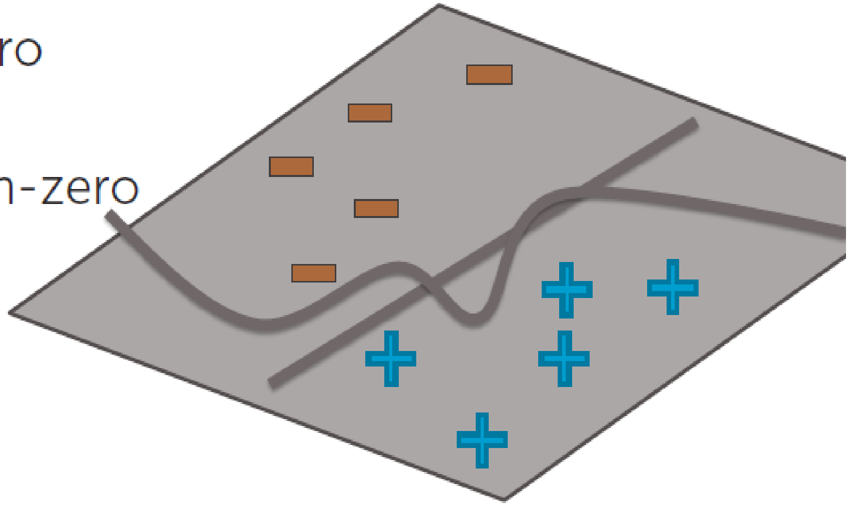
Score(x) = 1.0 #awesome – 1.5 #awful



Decision boundary

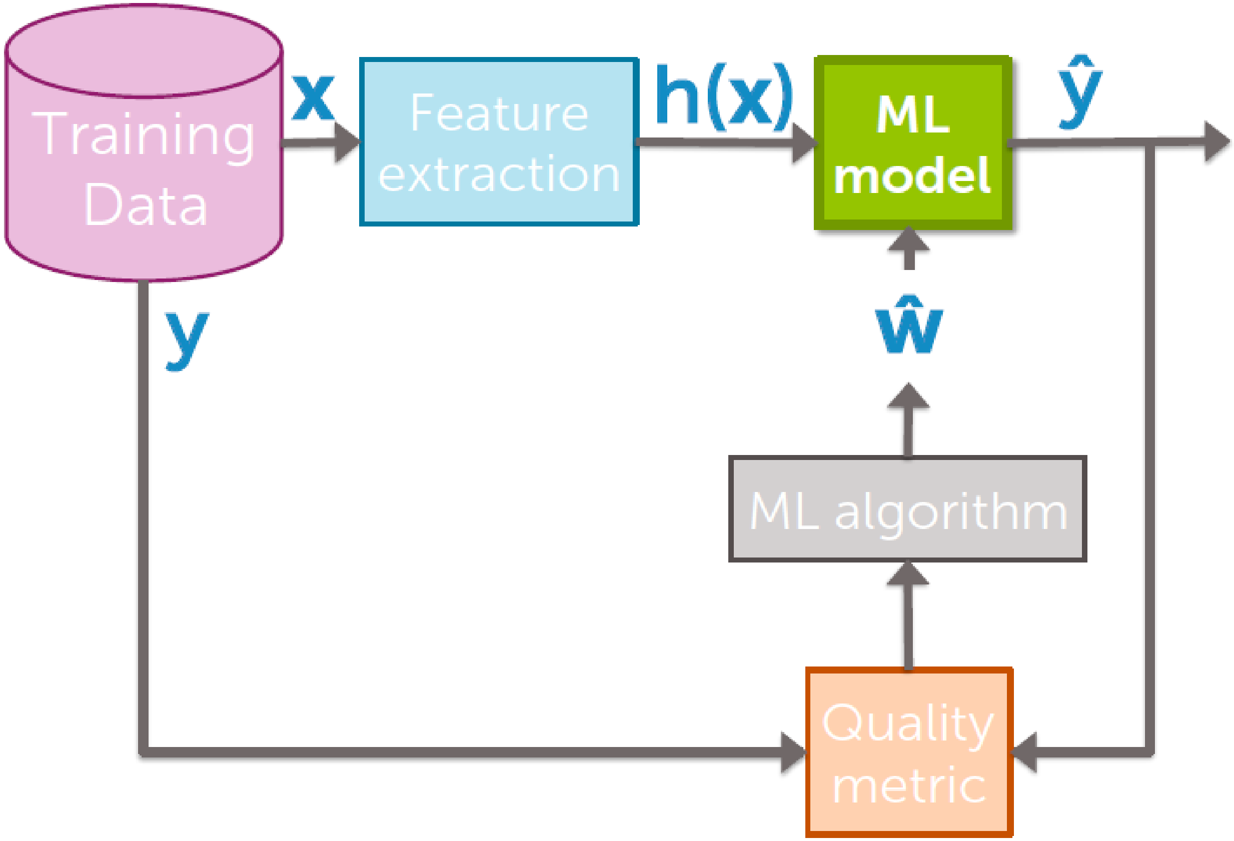
Decision boundary separates positive & negative predictions

- For linear classifiers:
  - When 2 coefficients are non-zero
    ➔ line
  - When 3 coefficients are non-zero
    ➔ plane
  - When many coefficients are non-zero
    ➔ hyperplane
- For more general classifiers
  ➔ more complicated shapes



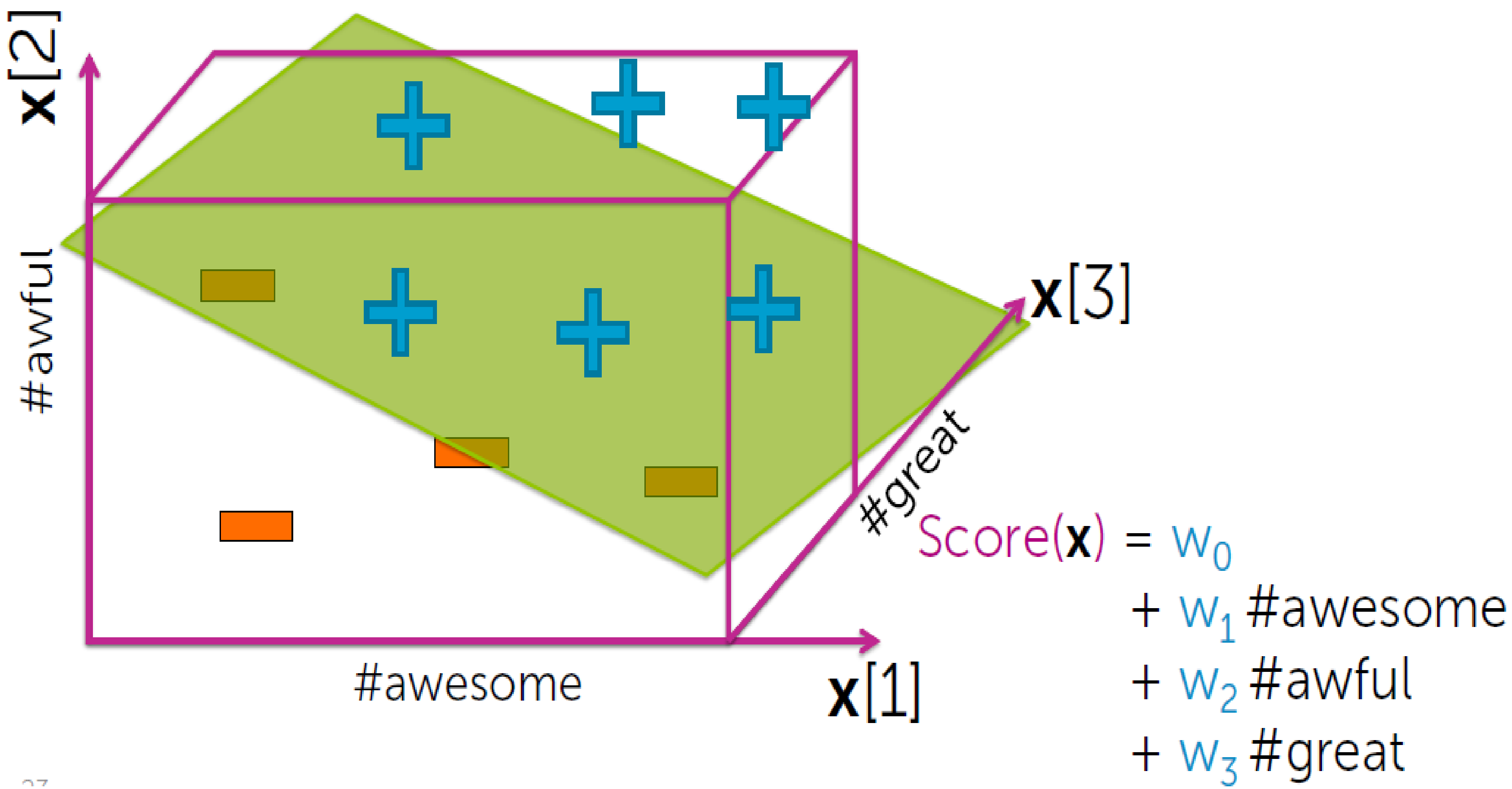
Flow chart: 

ML
model



Training
Data
x
Feature
extraction
h(x)
ML
model
ŷ
y
ŵ
ML algorithm
Quality
metric

Coefficients of classifier



General notation

Output: y ← $\{-1,+1\}$

Inputs: $\mathbf{x} = (\mathbf{x}[1], \mathbf{x}[2], \ldots, \mathbf{x}[d])$

↑ d-dim vector

Notational conventions:

- $\mathbf{x}[j]$ = j^{th} input (*scalar*)
- $h_j(\mathbf{x})$ = j^{th} feature (*scalar*)
- $\mathbf{x}_i$ = input of i^{th} data point (*vector*)
- $\mathbf{x}_i[j]$ = j^{th} input of i^{th} data point (*scalar*)

Simple hyperplane

Model: $\hat{y}_i = \text{sign}(\text{Score}(\mathbf{x}_i))$

$\text{Score}(\mathbf{x}_i) = w_0 + w_1\,\mathbf{x}_i[1] + \ldots + w_d\,\mathbf{x}_i[d] = \mathbf{w}^T\mathbf{x}_i$

feature 1 = 1

feature 2 = $\mathbf{x}[1]$ … e.g., #awesome

feature 3 = $\mathbf{x}[2]$ … e.g., #awful

…

feature d+1 = $\mathbf{x}[d]$ … e.g., #ramen

D-dimensional hyperplane

More generic features...

Model: $\hat{y}_i = \text{sign}(\text{Score}(\mathbf{x}_i))$

$$\text{Score}(\mathbf{x}_i) = w_0\, h_0(\mathbf{x}_i) + w_1\, h_1(\mathbf{x}_i) + \ldots + w_D\, h_D(\mathbf{x}_i)$$

$$= \sum_{j=0}^{D} w_j\, h_j(\mathbf{x}_i) = \mathbf{w}^T h(\mathbf{x}_i)$$

feature 1 = $h_0(\mathbf{x})$ ... e.g., 1

feature 2 = $h_1(\mathbf{x})$ ... e.g., $\mathbf{x}[1]$ = #awesome

feature 3 = $h_2(\mathbf{x})$ ... e.g., $\mathbf{x}[2]$ = #awful

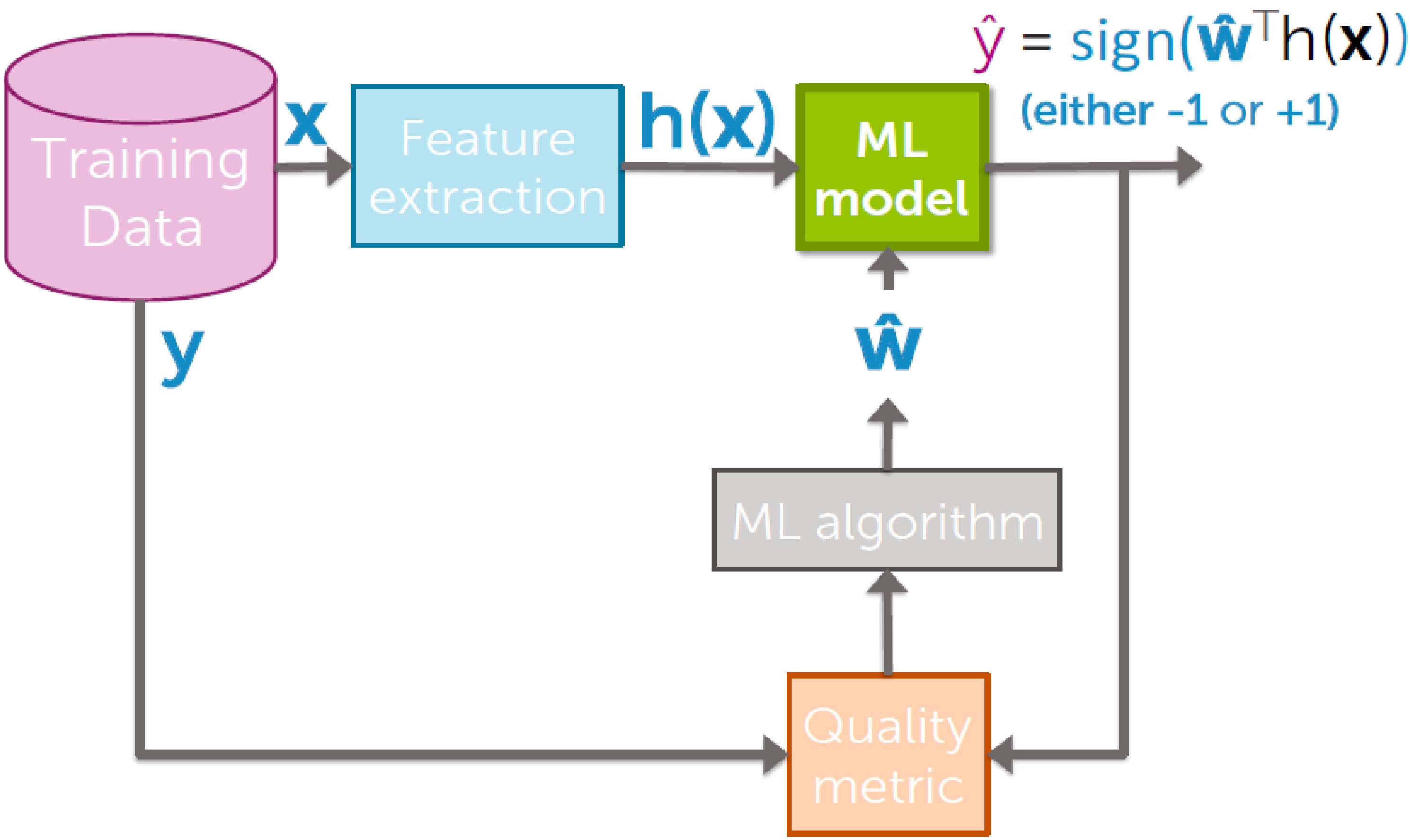
or, $\log(\mathbf{x}[7])\, \mathbf{x}[2]$ = log(#bad) x #awful

or, tf-idf("awful")

...

feature D+1 = $h_D(\mathbf{x})$ ... some other function of $\mathbf{x}[1], \ldots, \mathbf{x}[d]$

Flow chart: ML model

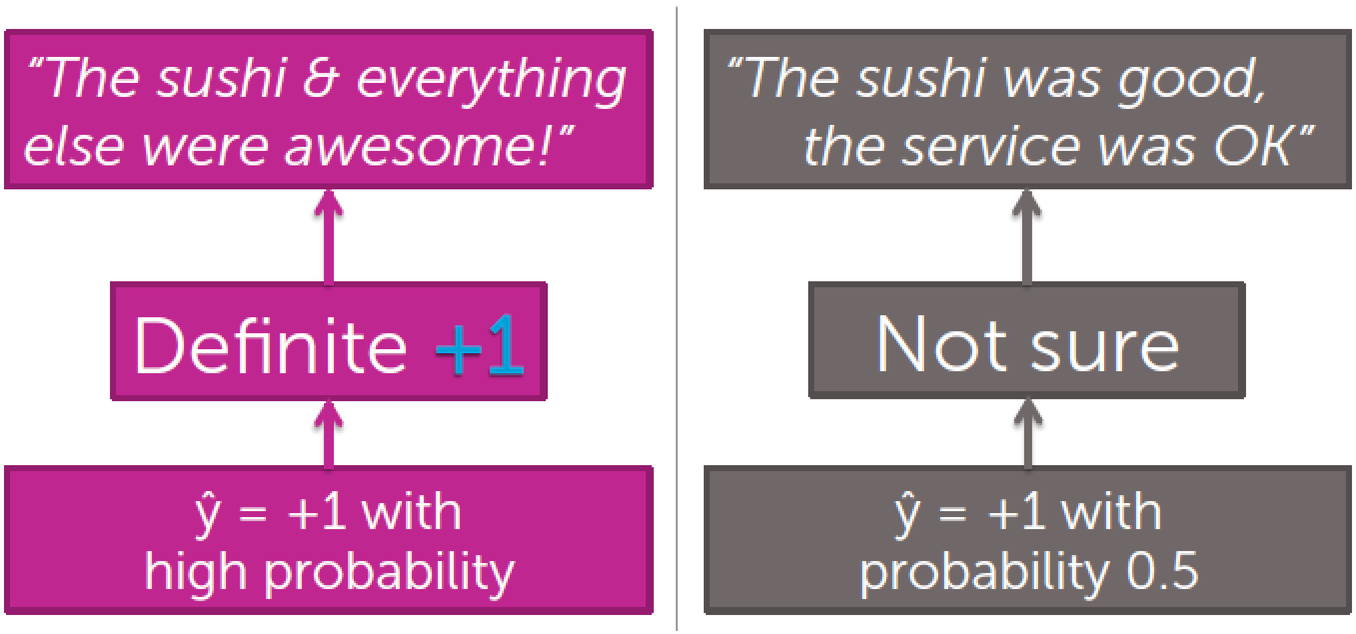


Linear classifier

- Class probability

How confident is your prediction?

- Thus far, we've outputted a prediction +1 or -1
- But, how sure are you about the prediction?



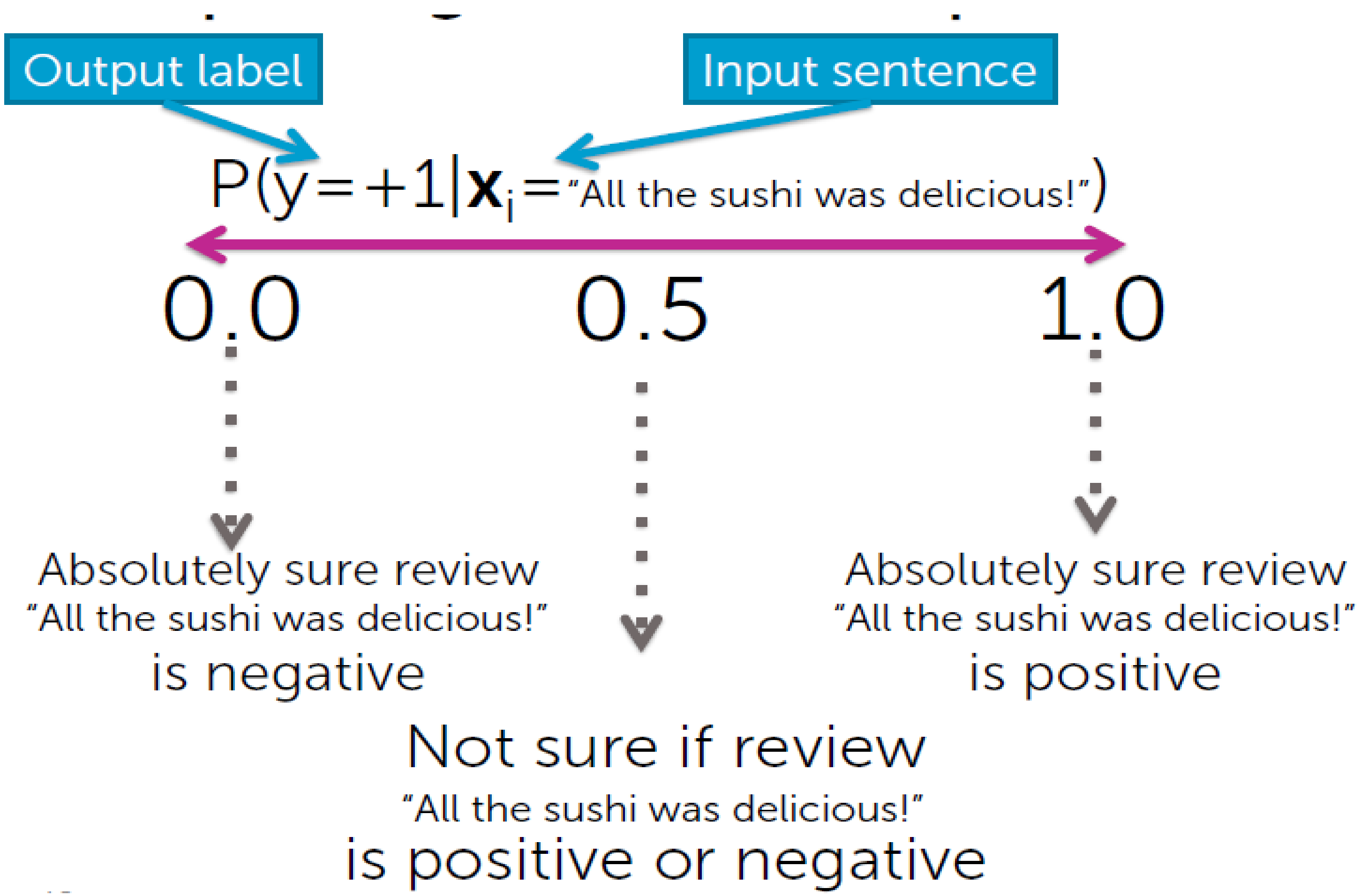
Conditional probability

Probability a review with
3 “awesome” and 1 “awful” is positive is 0.9

x = review text	y = sentiment
All the sushi was delicious! Easily best sushi in Seattle.	+1
Sushi was **awesome** & everything else was **awesome**! The service was **awful**, but overall **awesome** place!	+1
My wife tried their ramen, it was pretty forgettable.	-1
The sushi was good, the service was OK	+1
...	...
awesome ... awesome ... awful ... awesome	+1
...	...
awesome ... awesome ... awful ... awesome	-1
...	...
...	...
awesome ... awesome ... awful ... awesome	+1

I expect 90% of rows with reviews containing 3 “awesome” & 1 “awful” to have y = +1
(Exact number will vary for each specific dataset)

Interpreting conditional probabilities



How confident is your prediction?

"The sushi & everything else were awesome!"

Definite +1

$P(y=+1|\mathbf{x}=$ *"The sushi & everything else were awesome!"* $)$

$= 0.99$

"The sushi was good, the service was OK"

Not sure

$P(y=+1|\mathbf{x}=$ *"The sushi was good, the service was OK"* $)$

$= 0.55$

Many classifiers provide a degree of certainty:

Output label → $P(y|\mathbf{x})$ ← Input sentence

Extremely useful in practice

Learn conditional probabilities from data

Training data: N observations ($\mathbf{x}_i$,y_i)

x[1] = #awesome	x[2] = #awful	y = sentiment
2	1	+1
0	2	-1
3	3	-1
4	1	+1
...	...	...

Predicting class probabilities

Sentence from review

Input: **x**

Predict most likely class

$\hat{P}(y|x)$ = estimate of class probabilities

If $\hat{P}(y=+1|x) > 0.5$:

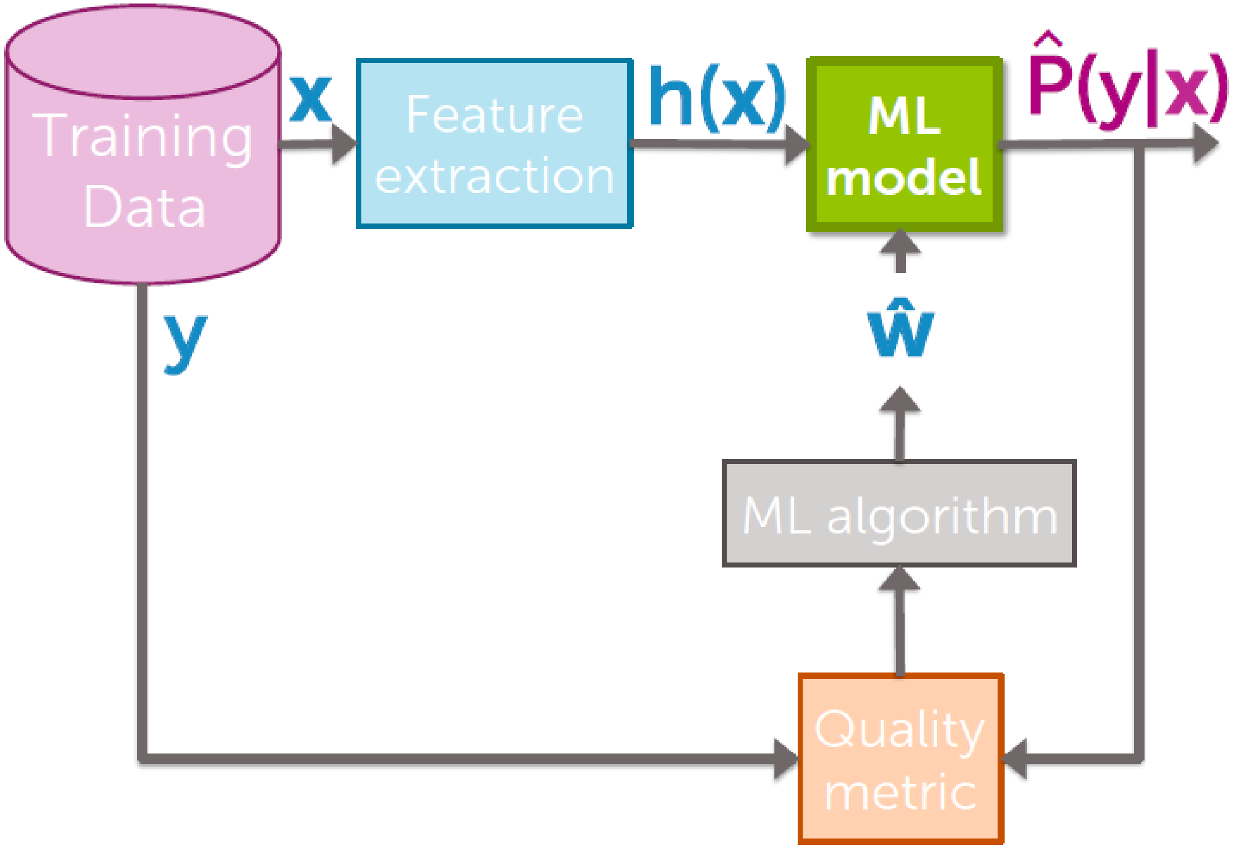
$\hat{y} = +1$

Else:

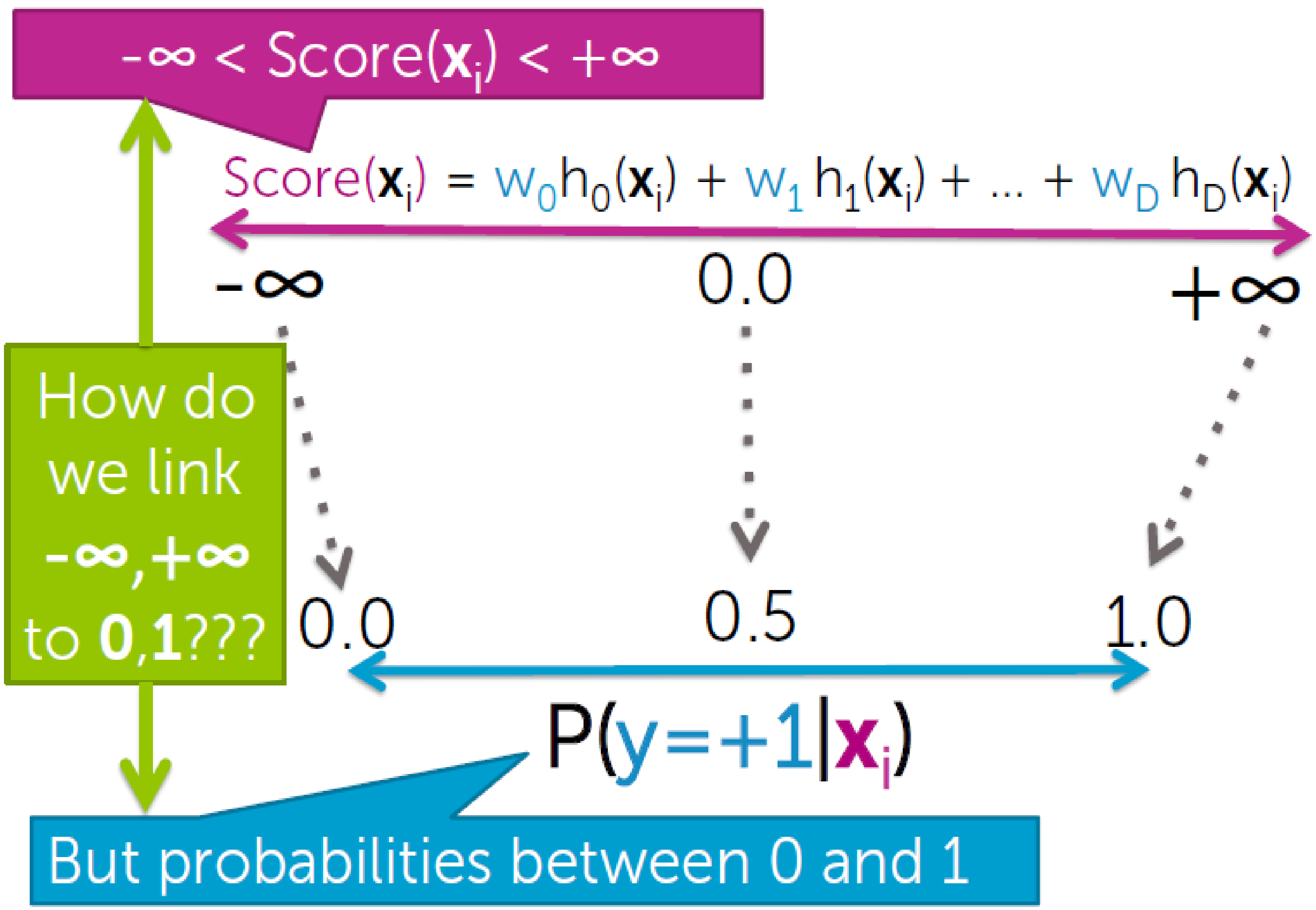
$\hat{y} = -1$

- Estimating $\hat{P}(y|x)$ improves **interpretability**:
  - Predict $\hat{y} = +1$ **and** tell me how sure you are

Flow chart: ML model

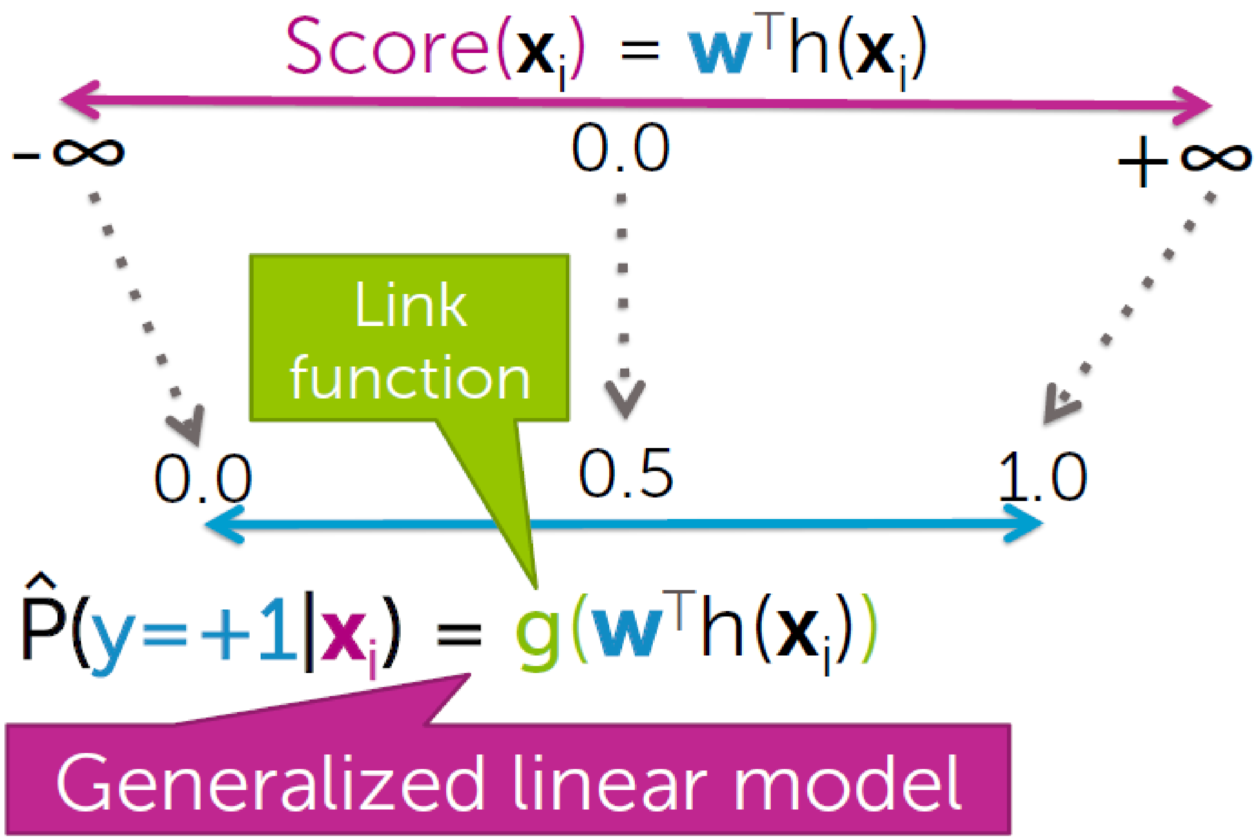


Why not just use regression to build classifier?



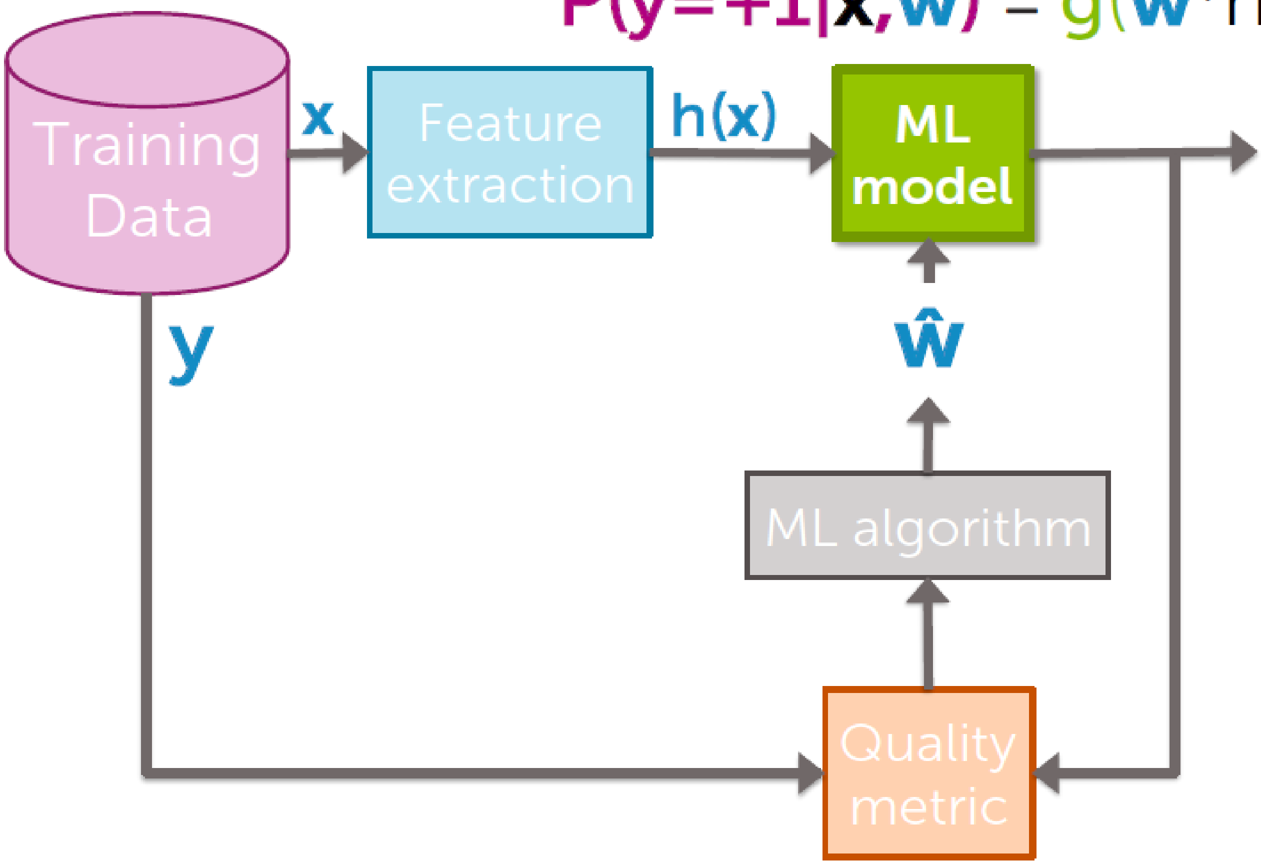
Link function

Link function: squeeze real line into [0,1]



Flow chart: ML model

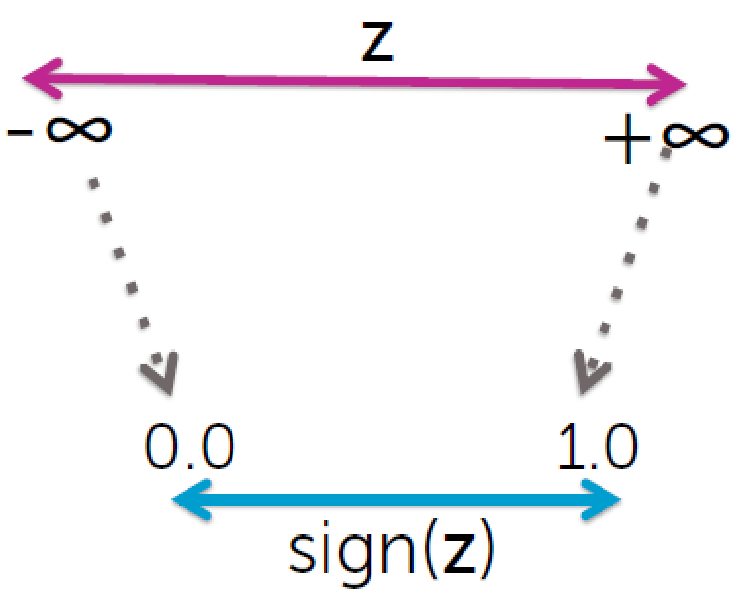
$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = g(\hat{\mathbf{w}}^T h(\mathbf{x}))$$

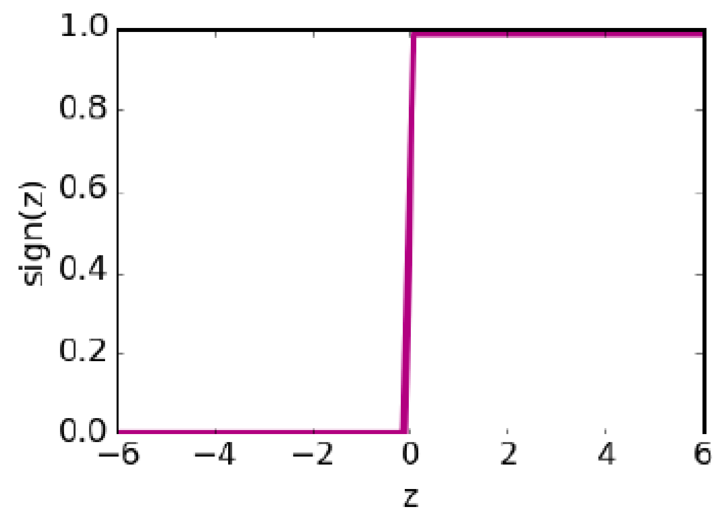


Logistic regression classifier:

- linear score with logistic link function

Simplest link function: sign(z)





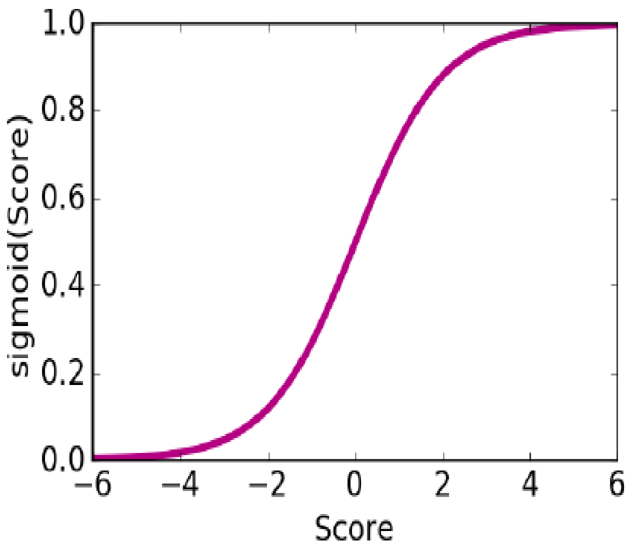
$$sign(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

But, sign(z) only outputs -1 or +1, no probabilities in between

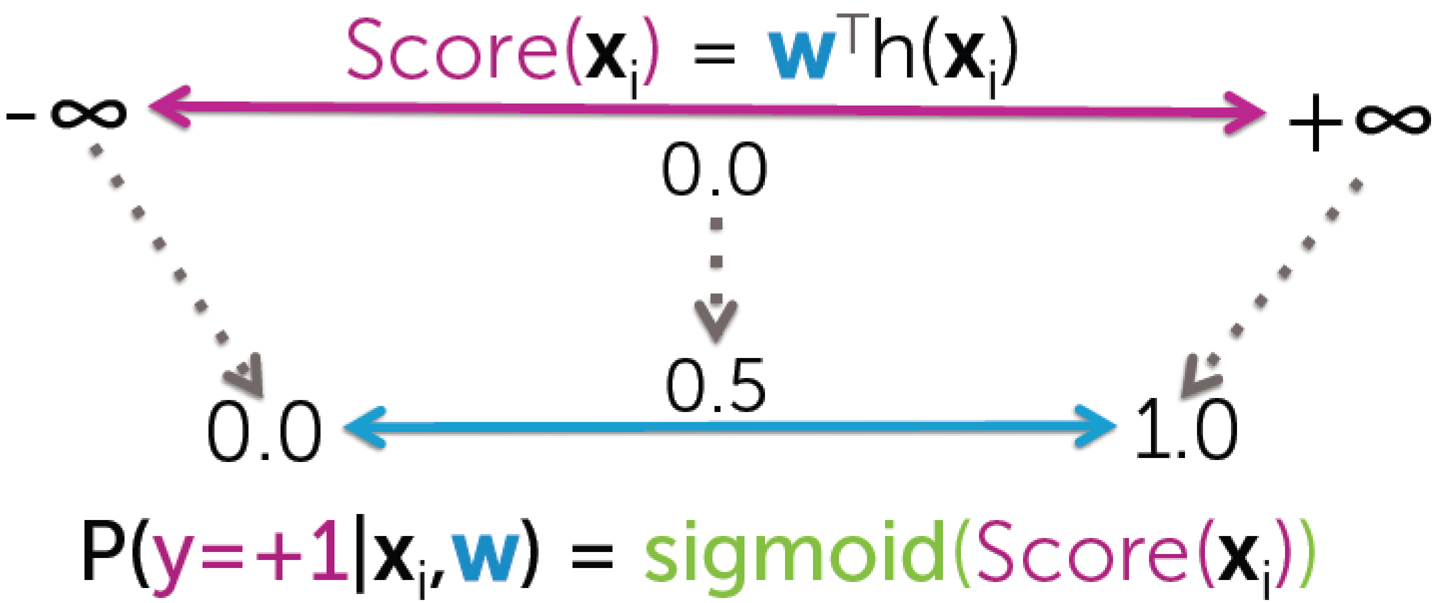
Logistic function (sigmoid, logit)

$$sigmoid(\text{Score}) = \frac{1}{1 + e^{-\text{Score}}}$$

Score	$-\infty$	-2	0.0	+2	$+\infty$
sigmoid(Score)	0.0	0.12	0.5	0.88	1.0



Logistic regression model



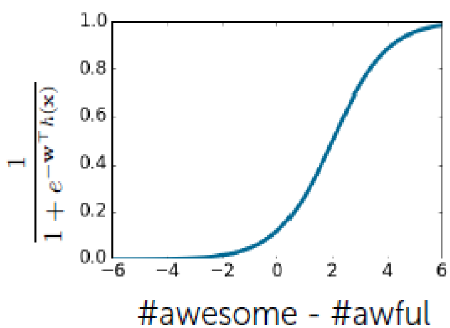
Effect of coefficients

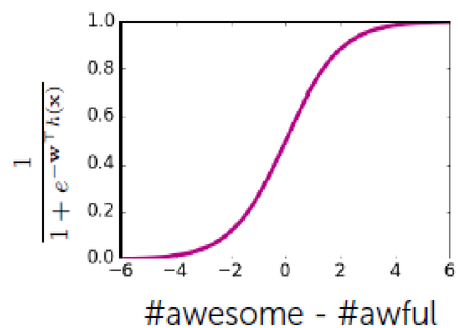
Effect of coefficients on logistic regression model

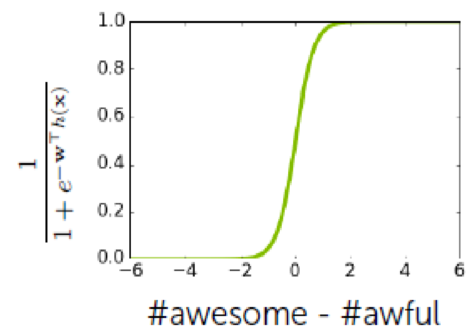
w_0	-2
$w_{\#awesome}$	+1
$w_{\#awful}$	-1

w_0	0
$w_{\#awesome}$	+1
$w_{\#awful}$	-1

w_0	0
$w_{\#awesome}$	+3
$w_{\#awful}$	-3

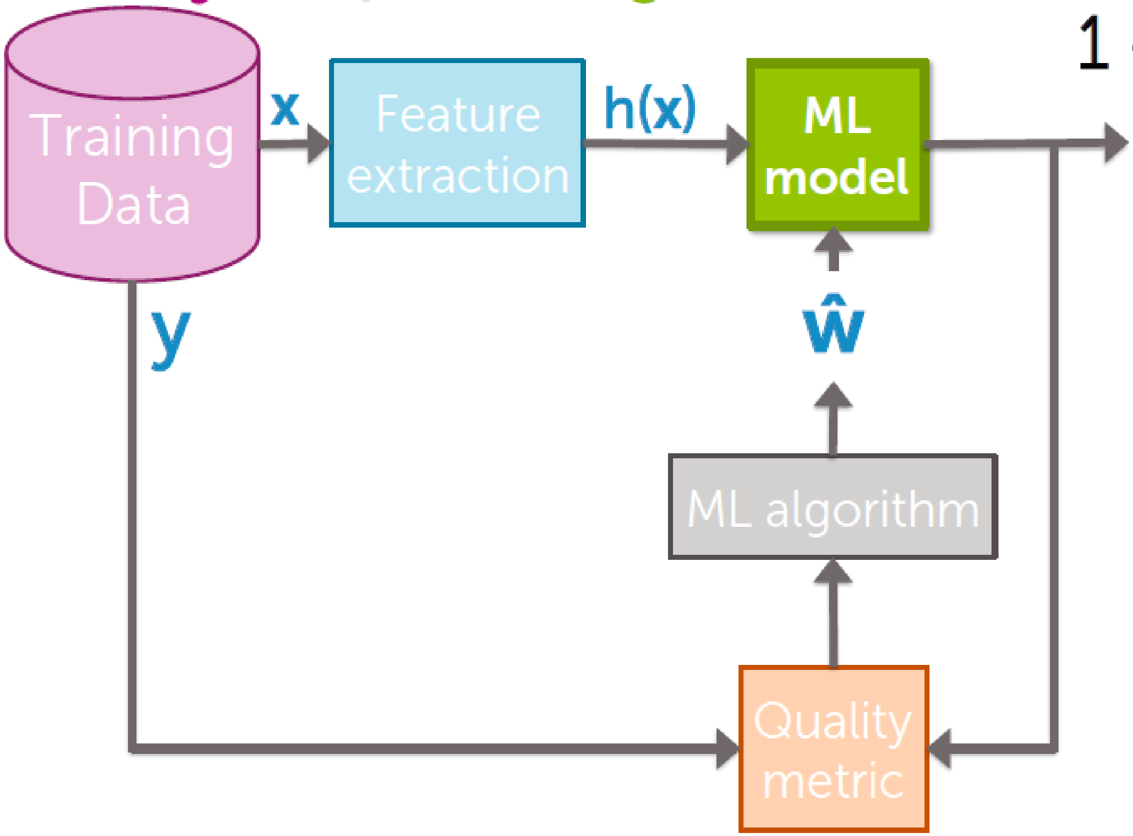






Flow chart:

$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = \text{sigmoid}(\hat{\mathbf{w}}^T h(\mathbf{x})) = \frac{1}{1+e^{-\hat{\mathbf{w}}^T h(\mathbf{x})}}$$

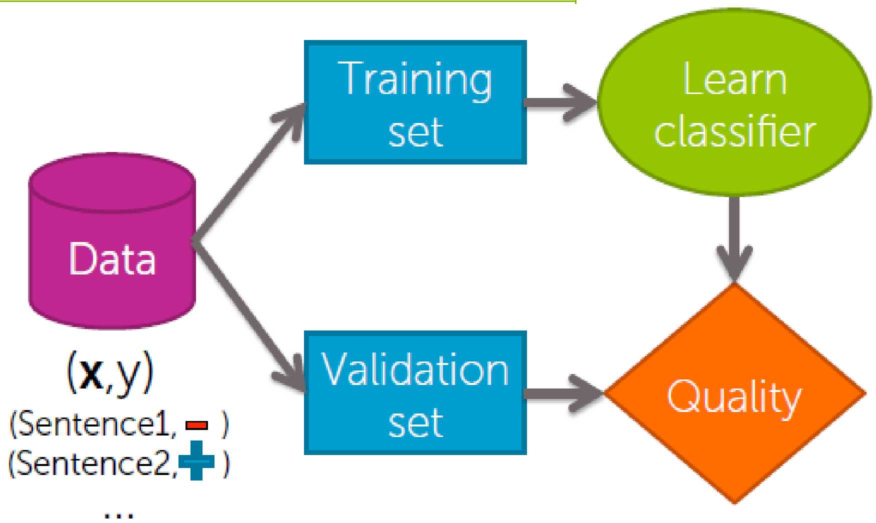


Learning logistic regression model

Training a classifier = Learning the coefficients

Word	Coefficient	Value
	$\hat{w}_0$	-2.0
good	$\hat{w}_1$	1.0
awesome	$\hat{w}_2$	1.7
bad	$\hat{w}_3$	-1.0
awful	$\hat{w}_4$	-3.3
...	...	...

$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = \frac{1}{1+e^{-\hat{\mathbf{w}}^\top h(\mathbf{x})}}$$

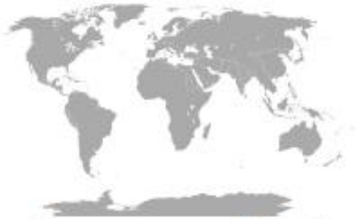


Categorical inputs

- Numeric inputs:
  - #awesome, age, salary,…
  - Intuitive when multiplied by coefficient
    - e.g., 1.5 #awesome
- Categorical inputs:

Numeric value, but should be interpreted as category
(98195 not about 9x larger than 10005)

Gender
(Male, Female,…)

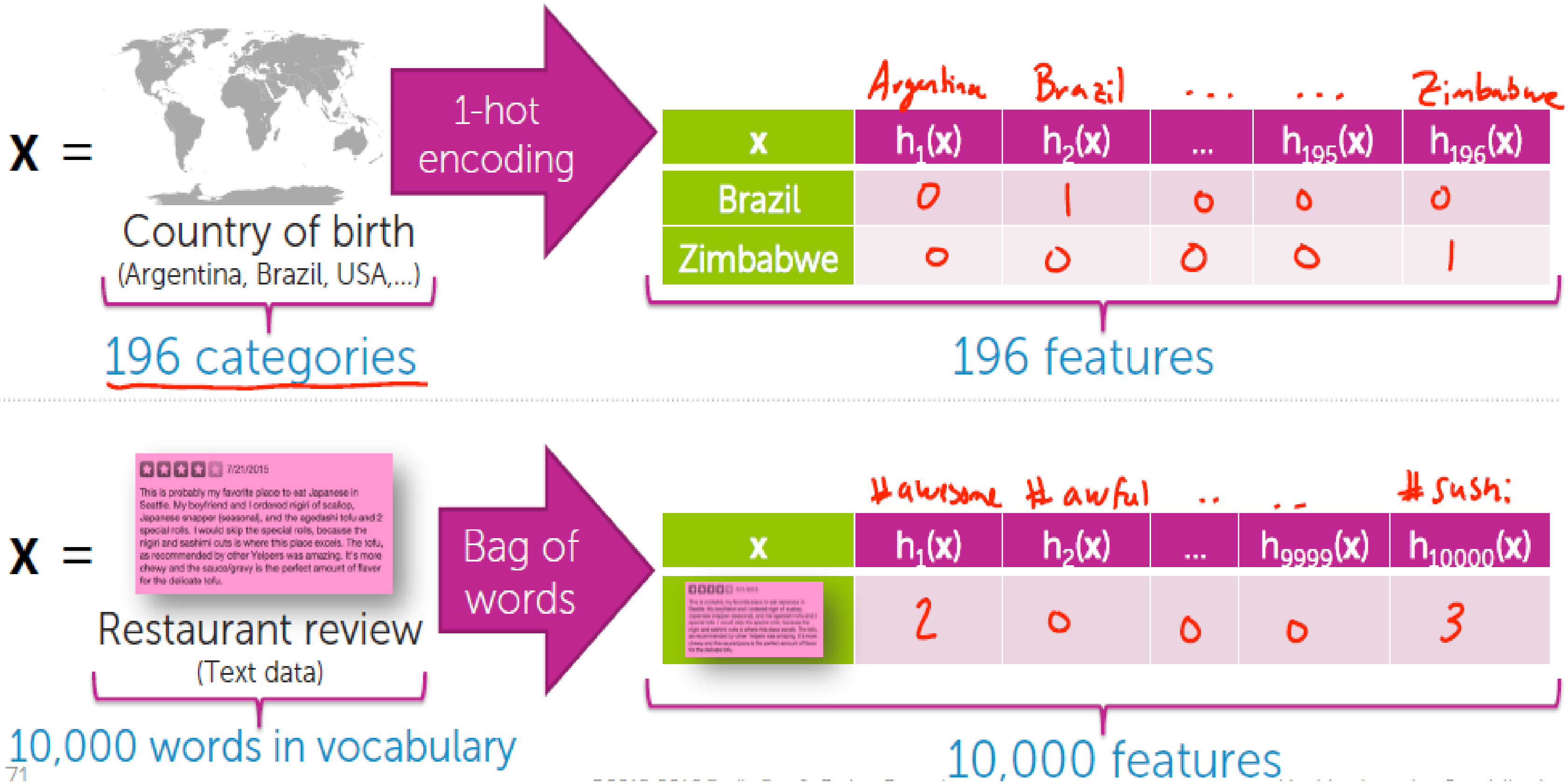


Country of birth
(Argentina, Brazil, USA,…)

Zipcode
(10005, 98195,…)

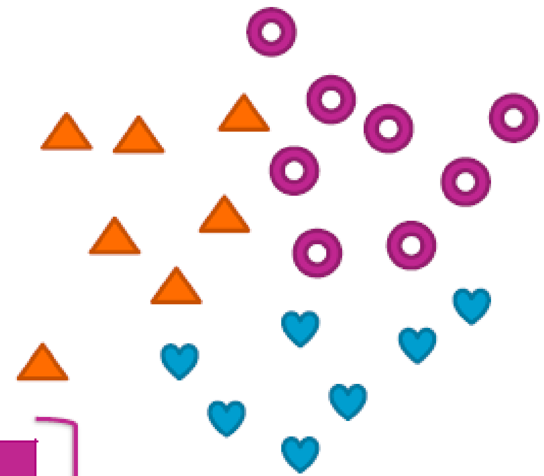
How do we multiply category by coefficient???
Must convert categorical inputs into numeric features

Encoding categories as numeric features



Multiclass classification

- C possible classes:
  - y can be 1, 2,..., C
- N datapoints:



Data point	x[1]	x[2]	y
$\mathbf{x}_1, y_1$	2	1	▲
$\mathbf{x}_2, y_2$	0	2	♥
$\mathbf{x}_3, y_3$	3	3	○
$\mathbf{x}_4, y_4$	4	1	○

Learn:

$\hat{P}(y=▲|\mathbf{x})$

$\hat{P}(y=♥|\mathbf{x})$

$\hat{P}(y=○|\mathbf{x})$

1 versus all

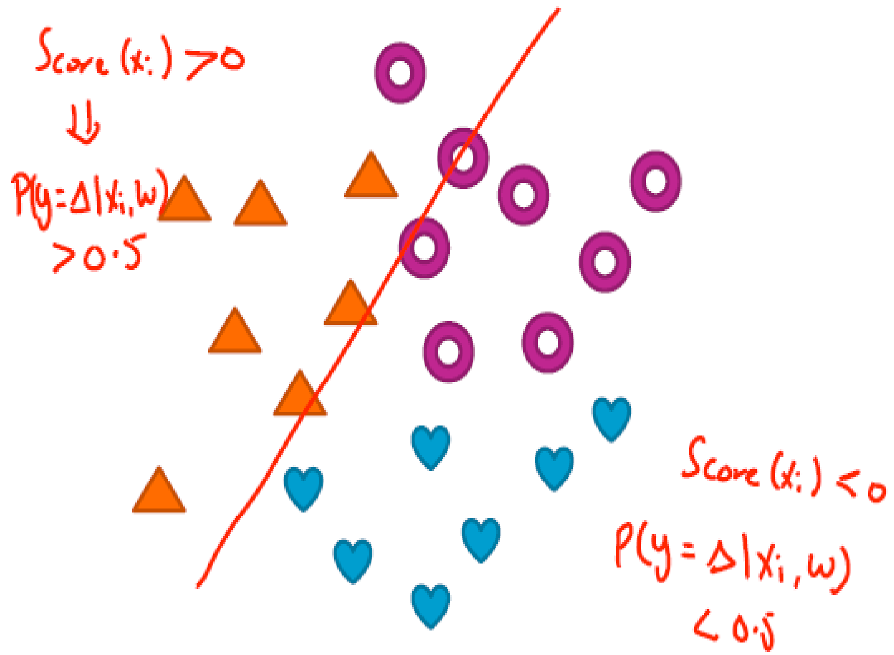
Estimate $\hat{P}(y=\blacktriangle|\mathbf{x})$ using 2-class model

+1 class: points with y_i= ▲

-1 class: points with y_i= ♥ OR ○

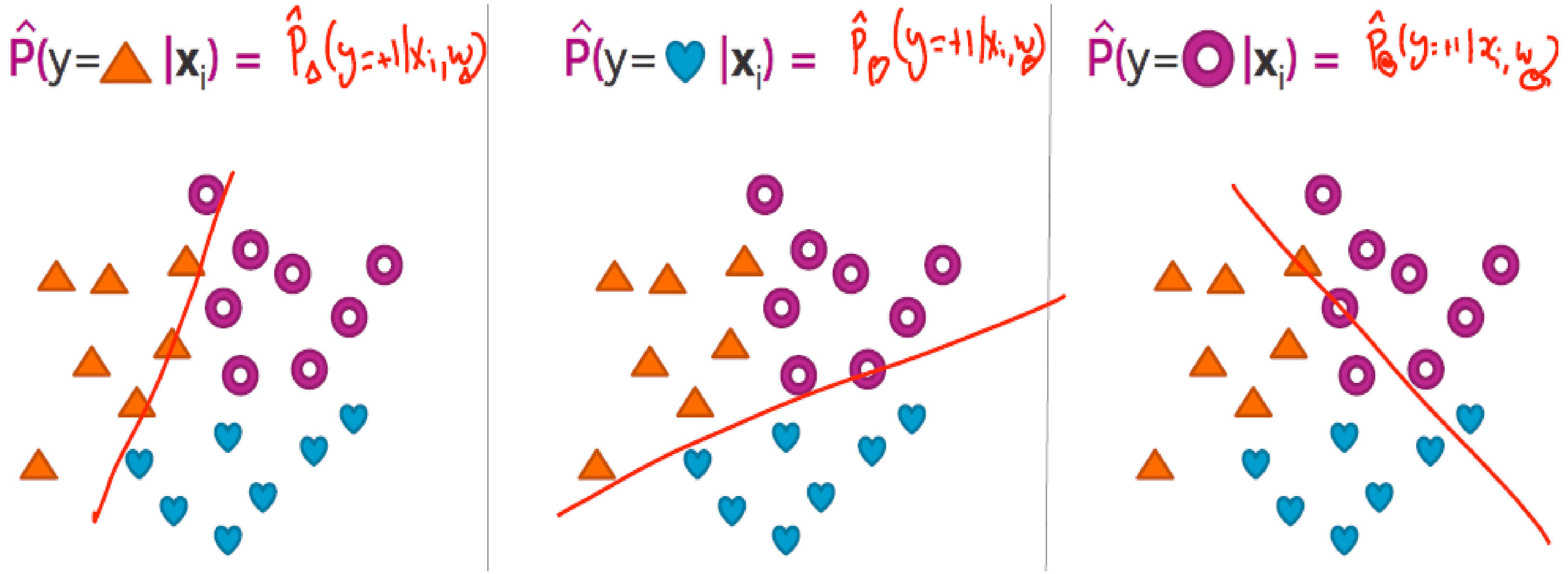
Train classifier: $\hat{P}_{\blacktriangle}(y=+1|\mathbf{x})$

Predict: $\hat{P}(y=\blacktriangle|\mathbf{x}_i) = \hat{P}_{\blacktriangle}(y=+1|\mathbf{x}_i)$

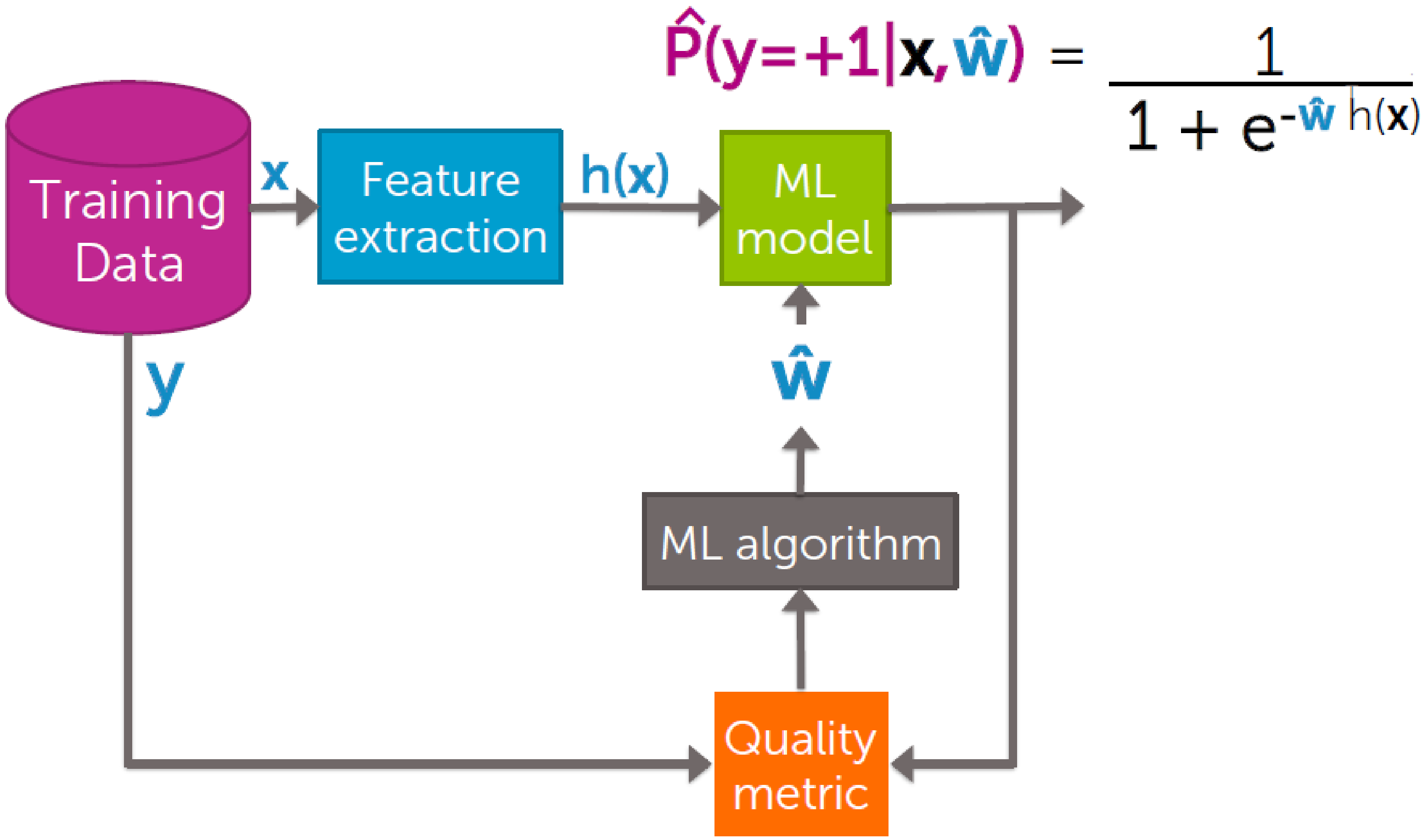


1 versus all

1 versus all: simple multiclass classification using C 2-class models



Summary: Logistic regression classifier



Linear classifier

- Parameters learning

Maximizing likelihood (probability of data)

Data point	x[1]	x[2]	y	Choose w to maximize
$\mathbf{x}_1, y_1$	2	1	+1	$P(y=+1 \mid x_1, w) = P(y=+1 \mid x[1]=2, x[2]=1, w)$
$\mathbf{x}_2, y_2$	0	2	-1	$P(y=-1 \mid x_2, w)$
$\mathbf{x}_3, y_3$	3	3	-1	$P(y=-1 \mid x_3, w)$
$\mathbf{x}_4, y_4$	4	1	+1	$P(y=+1 \mid x_4, w)$
$\mathbf{x}_5, y_5$	1	1	+1	
$\mathbf{x}_6, y_6$	2	4	-1	
$\mathbf{x}_7, y_7$	0	3	-1	
$\mathbf{x}_8, y_8$	0	1	-1	
$\mathbf{x}_9, y_9$	2	1	+1	

Must combine into single measure of quality ?

Multiply probabilities

$P(y=+1 \mid x_1, w)\, P(y=-1 \mid x_2, w)\, P(y=-1 \mid x_3, w) \ldots$

Maximum likelihood estimation (MLE)

Learn logistic regression model with MLE

Data point	x[1]	x[2]	y	Choose **w** to maximize
$\mathbf{x}_1, y_1$	2	1	y: +1	P(y=+1|**x**[1]=2, **x**[2]=1,**w**)
$\mathbf{x}_2, y_2$	0	2	-1	P(y=-1|**x**[1]=0, **x**[2]=2,**w**)
$\mathbf{x}_3, y_3$	3	3	-1	P(y=-1|**x**[1]=3, **x**[2]=3,**w**)
$\mathbf{x}_4, y_4$	4	1	+1	P(y=+1|**x**[1]=4, **x**[2]=1,**w**)

No $\hat{\mathbf{w}}$ achieves perfect predictions (usually)

Likelihood $\ell(\mathbf{w})$: Measures quality of fit for model with coefficients **w**

$$\ell(\mathbf{w}) = \underbrace{P(y=+1|\mathbf{x}[1]=2, \mathbf{x}[2]=1,\mathbf{w})}_{P(y_1|\mathbf{x}_1,\mathbf{w})}\ \underbrace{P(y=-1|\mathbf{x}[1]=0, \mathbf{x}[2]=2,\mathbf{w})}_{P(y_2|\mathbf{x}_2,\mathbf{w})}\ \underbrace{P(y=-1|\mathbf{x}[1]=3, \mathbf{x}[2]=3,\mathbf{w})}_{P(y_3|\mathbf{x}_3,\mathbf{w})}\ \underbrace{P(y=+1|\mathbf{x}[1]=4, \mathbf{x}[2]=1,\mathbf{w})}_{P(y_4|\mathbf{x}_4,\mathbf{w})}$$

Num. of data points → N

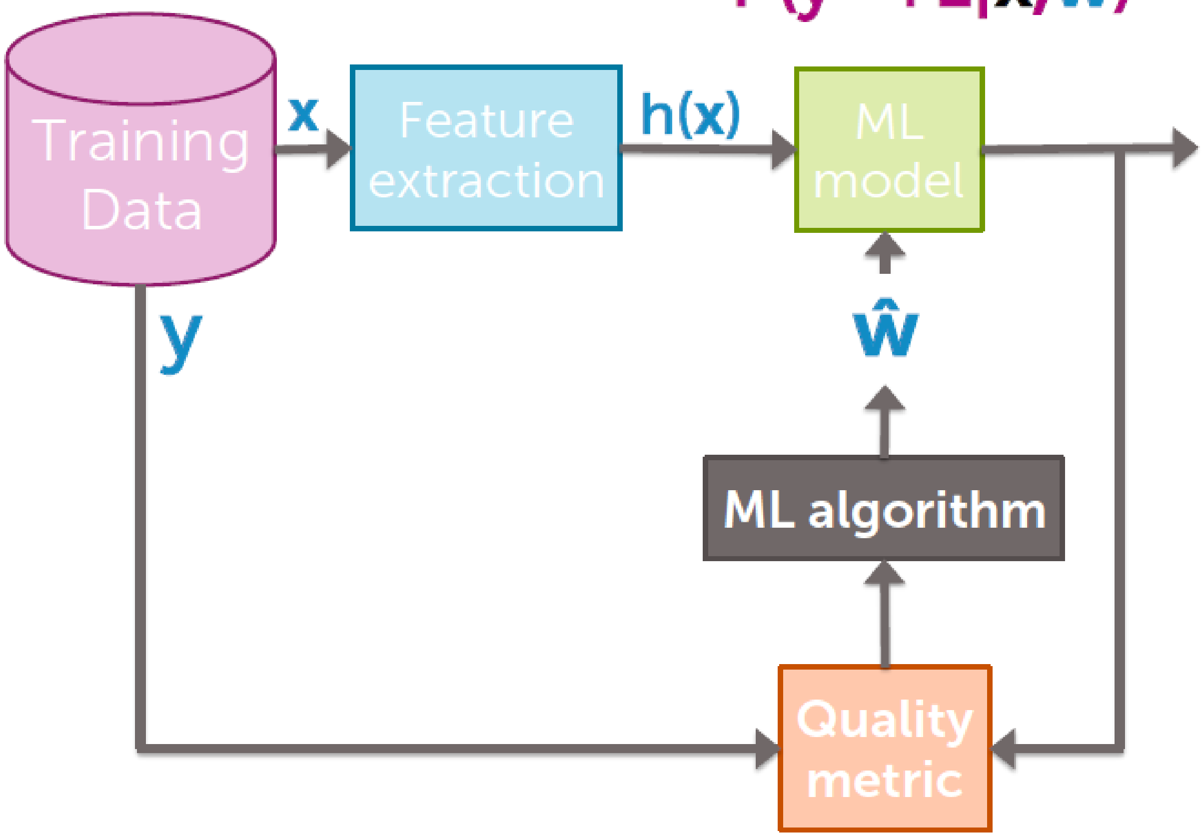
$$\ell(\mathbf{w}) = \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

← pick w to make this fn. as large as possible

17

Flow chart: ML algorithm

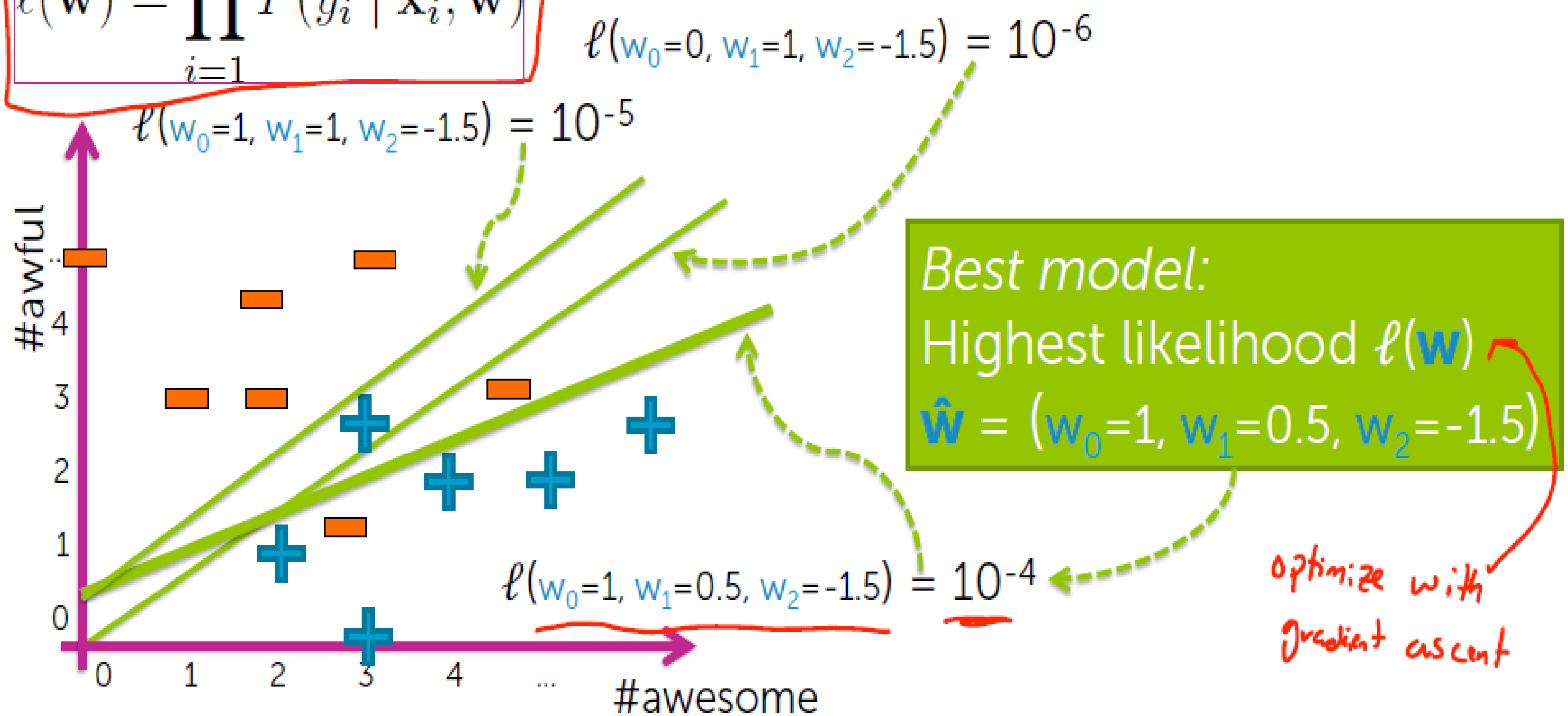
$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = \frac{1}{1+e^{-\hat{\mathbf{w}}^{\top} h(\mathbf{x})}}$$



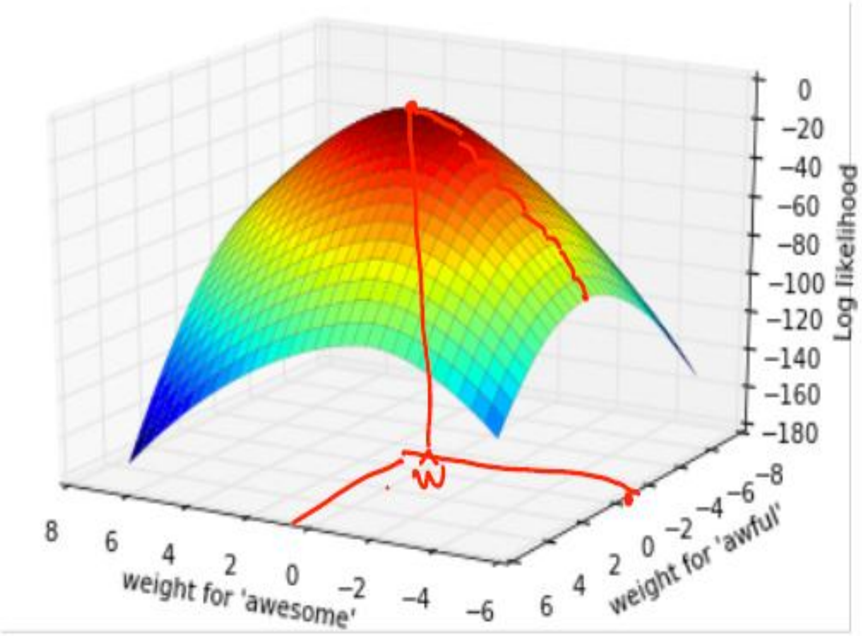
Find „best" classifier

Maximize likelihood over all possible w_0, w_1, w_2

$$\ell(\mathbf{w}) = \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$



Maximizing likelihood



Maximize function over all possible w_0, w_1, w_2

$$\max_{w_0, w_1, w_2} \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

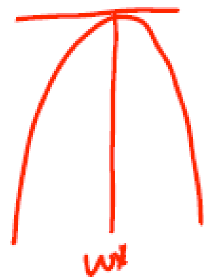
$\ell(w_0, w_1, w_2)$ is a function of 3 variables

No closed-form solution ➔ use gradient ascent

Gradient ascent

Convergence criteria

For convex functions, optimum occurs when



$$\frac{d\ell}{dw} = 0$$

In practice, stop when

$$\left.\frac{d\ell}{dw}\right|_{w^{(t)}} < \varepsilon$$

↑ tolerance

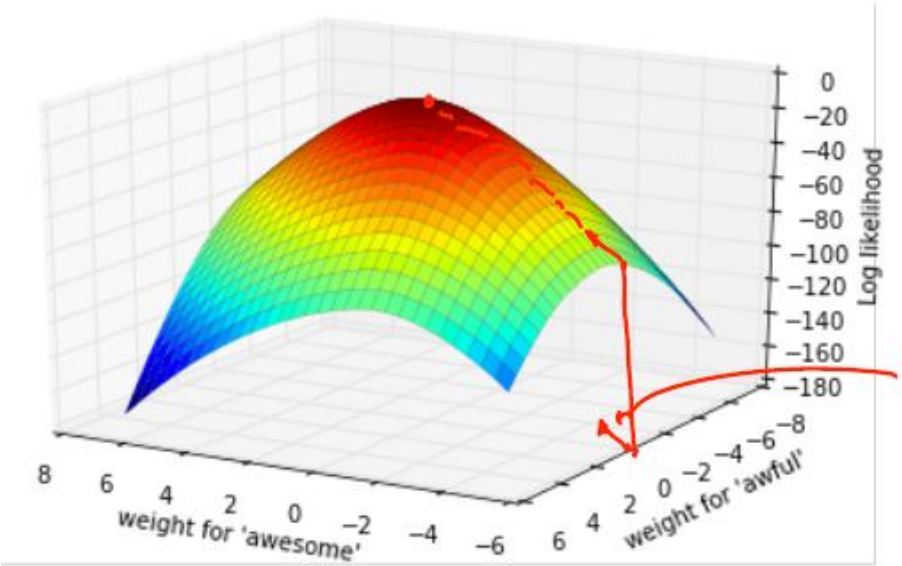
Algorithm:

while not converged

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \left.\frac{d\ell}{dw}\right|_{w^{(t)}}$$

Gradient ascent

Moving to multiple dimensions: Gradients



$$\nabla \ell(\mathbf{w}) = \begin{bmatrix} \frac{\partial \ell}{\partial w_0} \\ \frac{\partial \ell}{\partial w_1} \\ \vdots \\ \frac{\partial \ell}{\partial w_D} \end{bmatrix}$$

← D+1 dim vector

The log trick, often used in ML…

- Products become sums:

  $\ln a \cdot b = \ln a + \ln b$ | $\ln \frac{a}{b} = \ln a - \ln b$

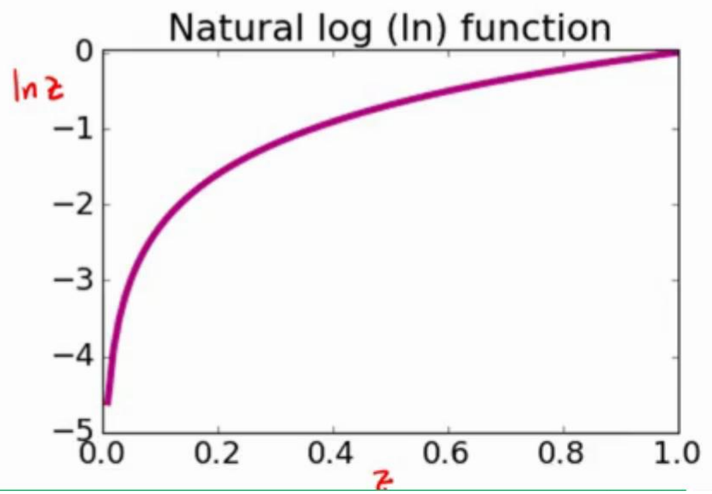
- Doesn't change maximum!
  - If $\hat{\mathbf{w}}$ maximizes f($\mathbf{w}$):

    $\hat{w} = \underset{w}{\arg\max}\, f(w)$ — the w that makes f(w) largest

  - Then $\hat{\mathbf{w}}_{\ln}$ maximizes ln(f($\mathbf{w}$)):

    $\hat{w}_{\ln} = \underset{w}{\arg\max}\, \ln(f(w))$

    $\hat{w} = \hat{w}_{\ln}$.



Derivative for logistic regression

Derivative of (log-)likelihood

Sum over data points

Feature value

Difference between truth and prediction

$$\frac{\partial \ell(\mathbf{w})}{\partial \mathbf{w}_j} = \sum_{i=1}^{N} h_j(\mathbf{x}_i)\left(\mathbb{1}[y_i = +1] - P(y = +1 \mid \mathbf{x}_i, \mathbf{w})\right)$$

predict x_i is positive

Indicator function:

$$\mathbb{1}[y_i = +1] = \begin{cases} 1 & \text{if } y_i = +1 \\ 0 & \text{if } y_i = -1 \end{cases}$$

See slides at the end of this lecture
If you are interested how it is derived.

Derivative for logistic regression

Computing derivative

$$\frac{\partial \ell(\mathbf{w}^{(t)})}{\partial \mathbf{w}_j} = \sum_{i=1}^{N} h_j(\mathbf{x}_i)\left(\mathbb{1}[y_i = +1] - P(y = +1 \mid \mathbf{x}_i, \mathbf{w}^{(t)})\right)$$

$w^{(t)}$:

$w_0^{(t)}$	0
$w_1^{(t)}$	1
$w_2^{(t)}$	-2

$h_1(x)$ = # awesome

$\frac{\partial \ell}{\partial w_1}$

x[1]	x[2]	y	$P(y=+1 \mid x_i, w)$	Contribution to derivative for w_1
2	1	+1	0.5	$2(1-0.5) = 1$
0	2	-1	0.02	$0(0-0.02) = 0$
3	3	-1	0.05	$3(0-0.05) = -0.15$
4	1	+1	0.88	$4(1-0.88) = 0.48$

Total derivative:

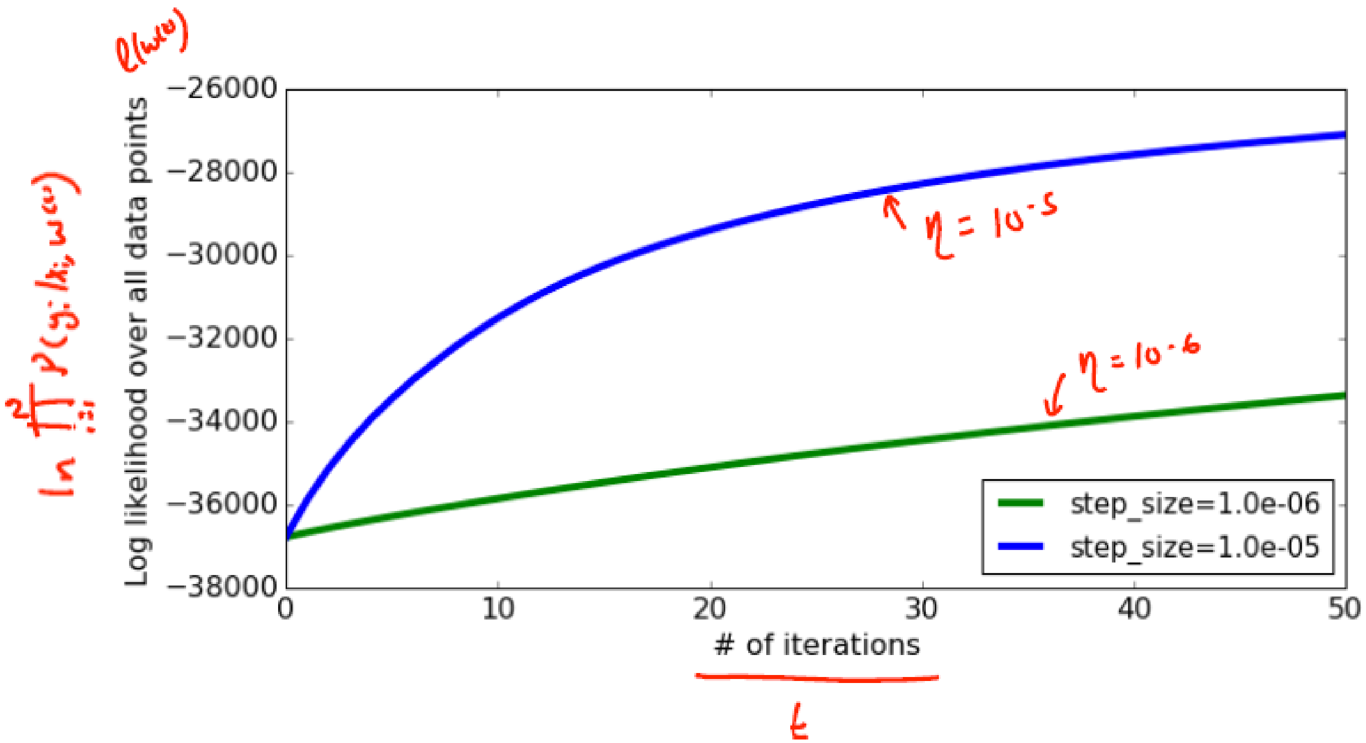
$$\frac{\partial \ell(w^{(t)})}{\partial w_1} = 1 + 0 - 0.15 + 0.48 = 1.33$$

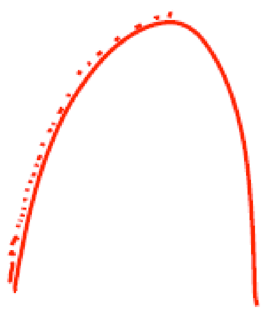
$$w_1^{(t+1)} = w_1^{(t)} + \eta \frac{\partial \ell(w^{(t)})}{\partial w_1} \quad \Big| \quad \eta = 0.1$$

$$= 1 + 0.1 \times 1.33 = 1.133$$

Choosing the step size

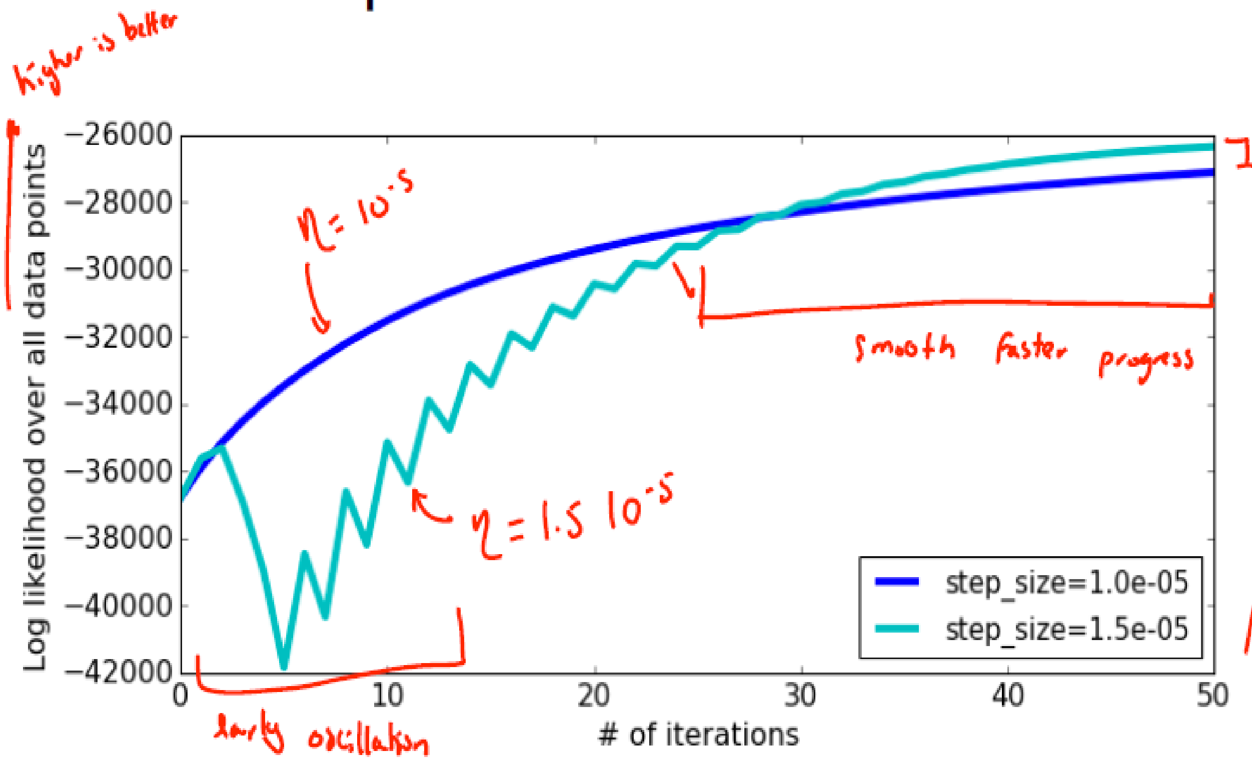
If step size is too small, can take a long time to converge

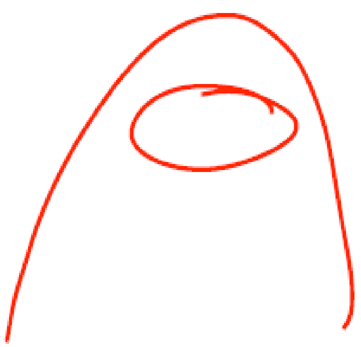




Choosing the step size

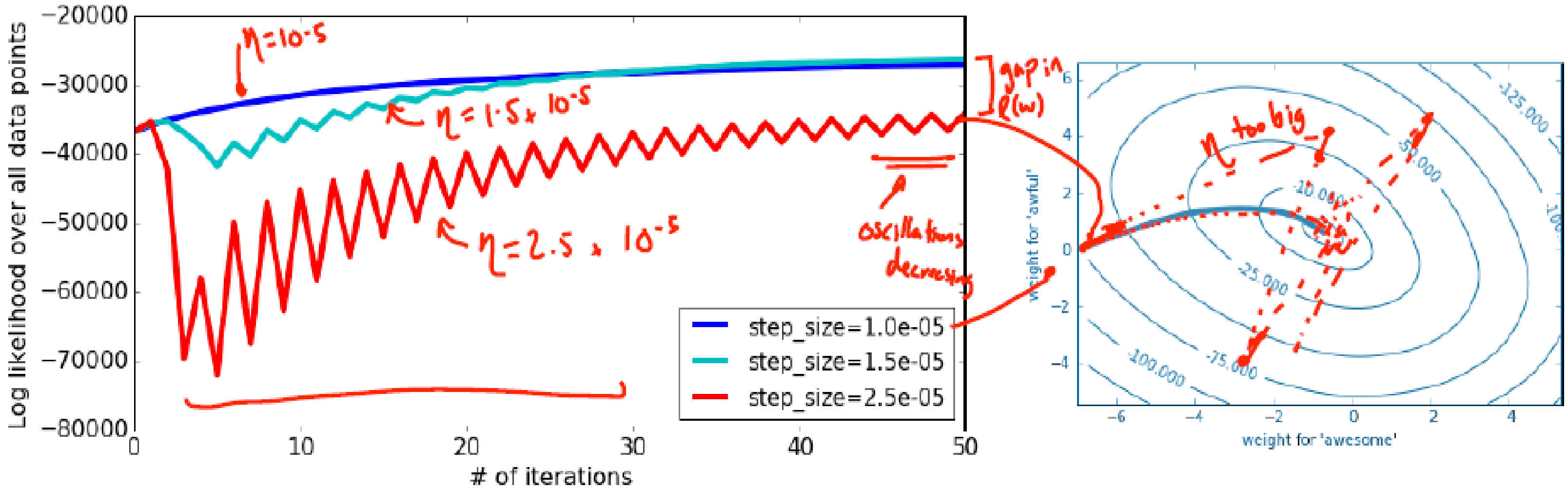
Compare converge with different step sizes





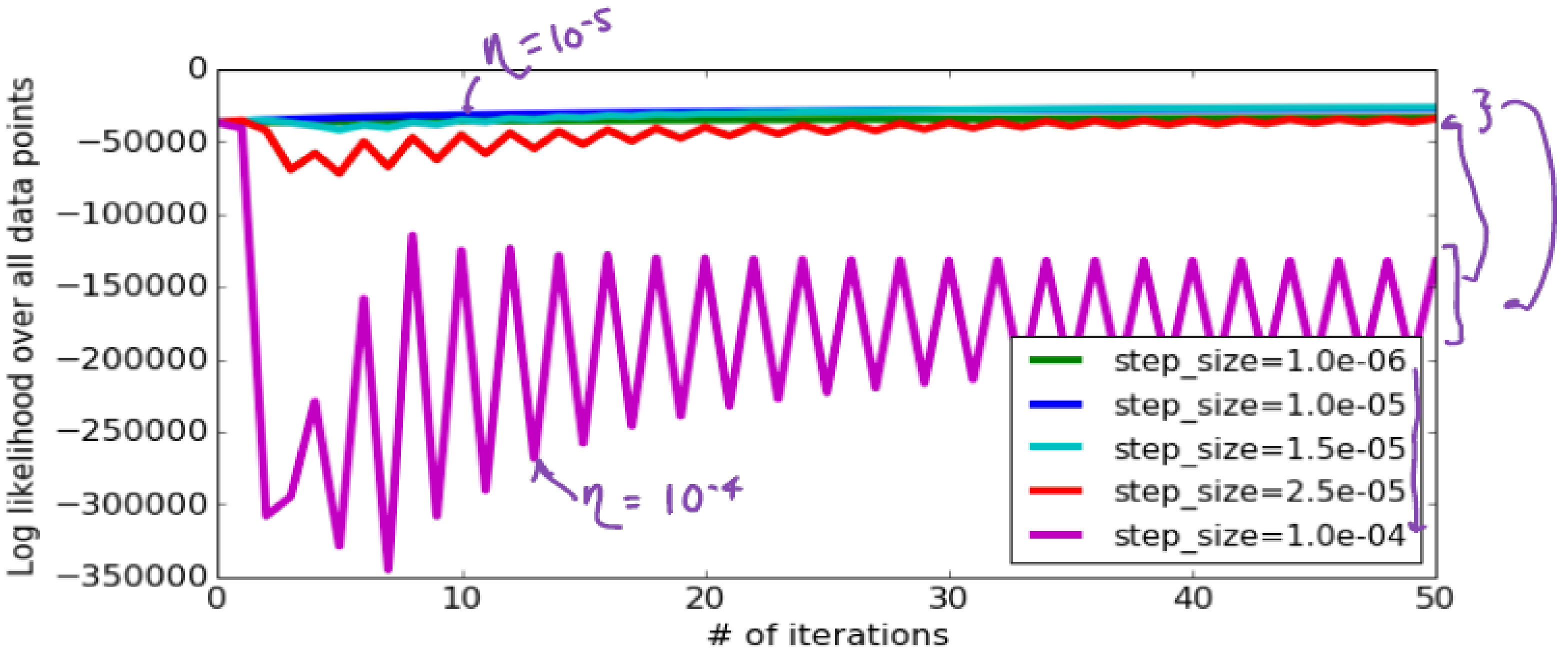
Choosing the step size

Careful with step sizes that are too large



Choosing the step size

Very large step sizes can even cause divergence or wild oscillations



Choosing the step size

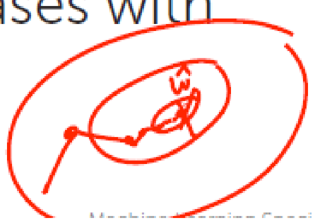
Simple rule of thumb for picking step size η

- Unfortunately, picking step size requires a lot of trial and error ☹
- Try a several values, exponentially spaced
  - **Goal**: plot learning curves to
    - find one η that is too small (smooth but moving too slowly)
    - find one η that is too large (oscillation or divergence)
- Try values in between to find "best" η

  ↳ exponentially space, pick one that leads best training data likelihood

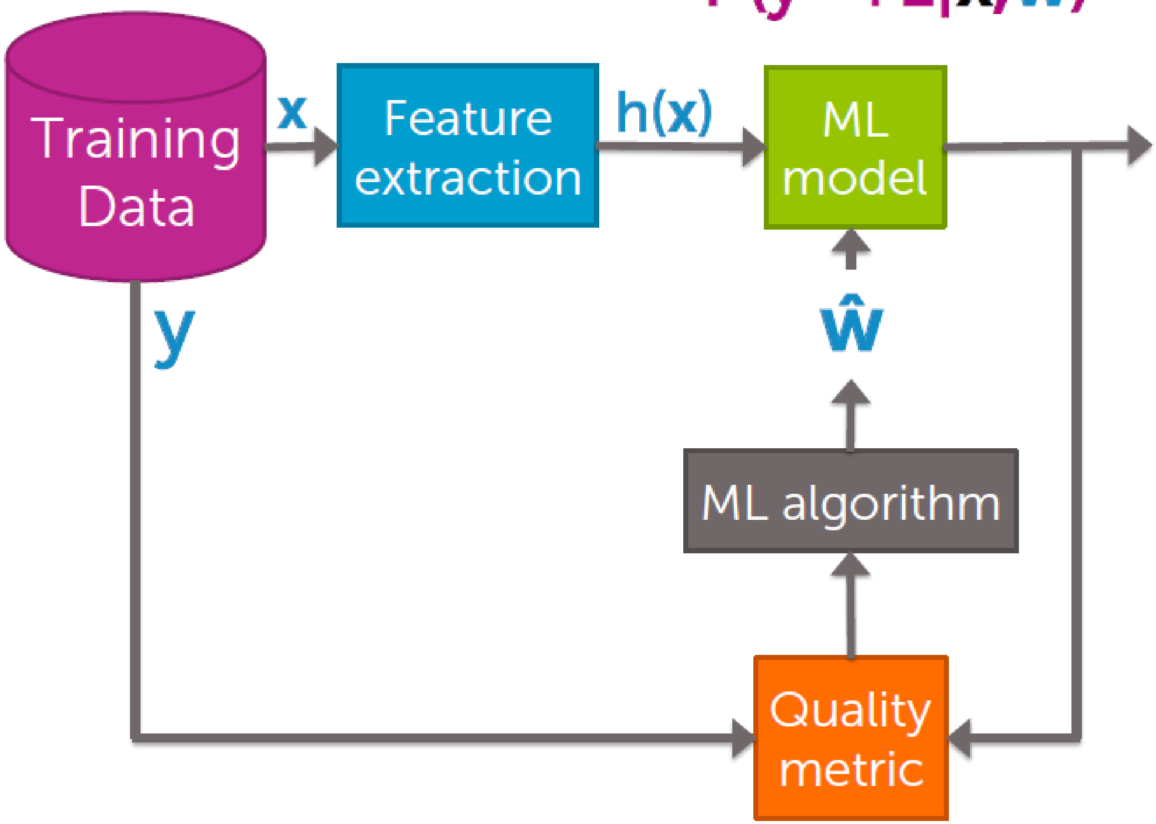
- *Advanced tip*: can also try step size that decreases with iterations, e.g.,

$$\eta_t = \frac{\eta_0}{t}$$



Flow chart: final look at it

$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = \frac{1}{1+e^{-\hat{\mathbf{w}}^\top h(\mathbf{x})}}$$

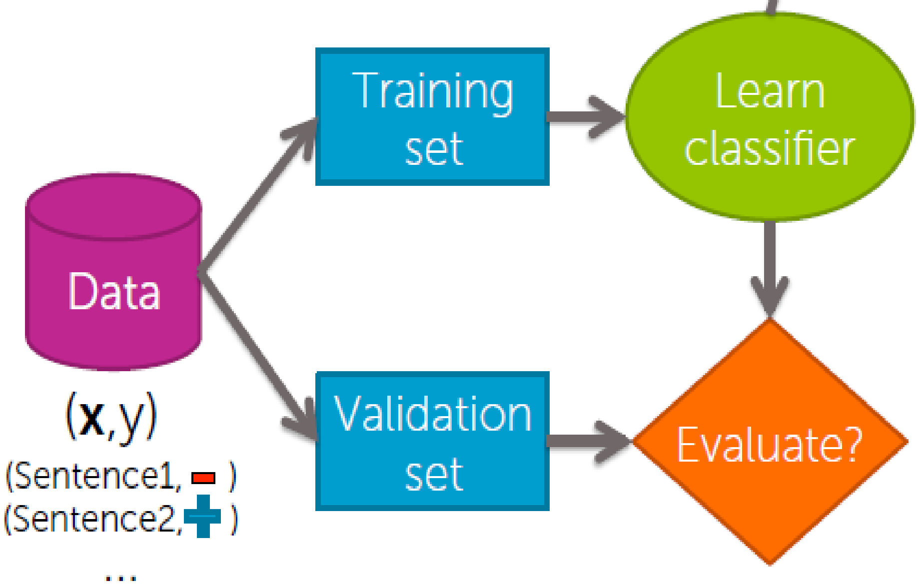


Linear classifier

- Overfitting & regularization

Training a classifier = Learning the coefficients

Word	Coefficient
good	1.0
awesome	1.7
bad	-1.0
awful	-3.3
...	...



Classification error & accuracy

- Error measures fraction of mistakes

$$\text{error} = \frac{\#\ \text{Mistakes}}{\text{Total number of data points}}$$

  - Best possible value is 0.0

- Often, measure **accuracy**
  - Fraction of correct predictions

$$\text{accuracy} = \frac{\#\ \text{Correct}}{\text{Total number of data points}}$$

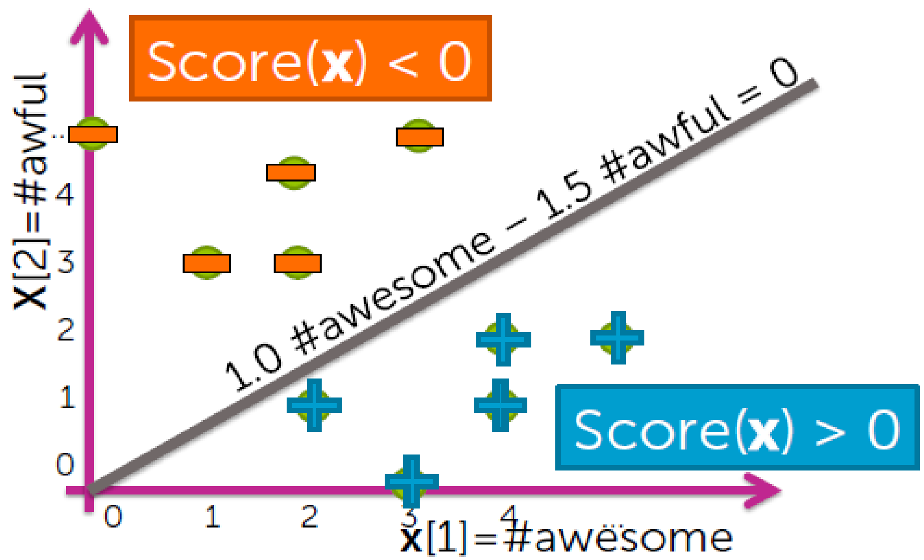
  - Best possible value is 1.0

Overfitting in classification

Decision boundary example

Word	Coefficient
#awesome	1.0
#awful	-1.5

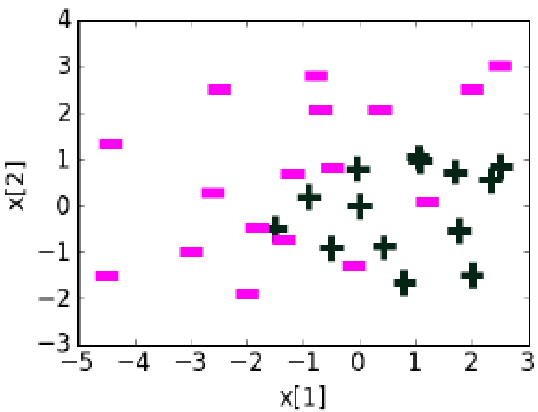
Score(x) = 1.0 #awesome – 1.5 #awful

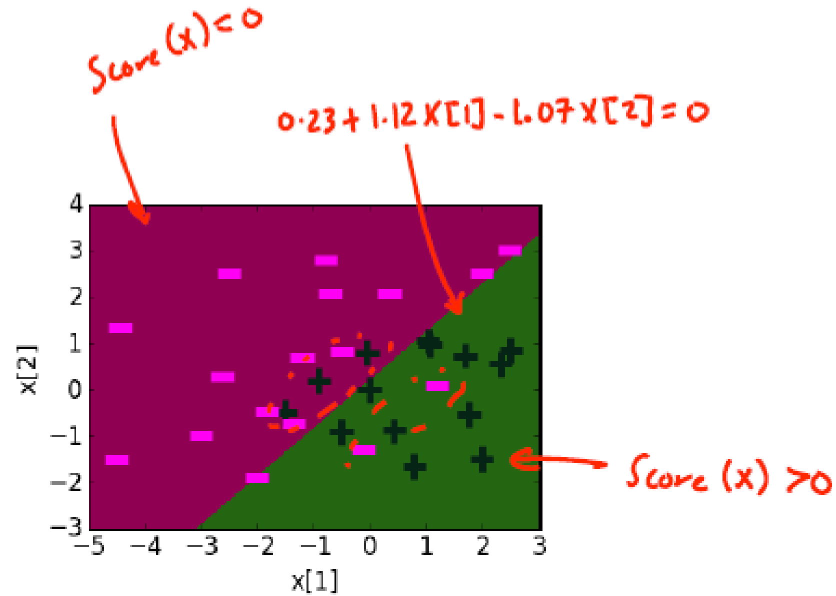


Overfitting in classification

Learned decision boundary

Feature	Value	Coefficient learned
$h_0(\mathbf{x})$	w_0 1	0.23
$h_1(\mathbf{x})$	w_1 x[1]	1.12
$h_2(\mathbf{x})$	w_2 x[2]	-1.07

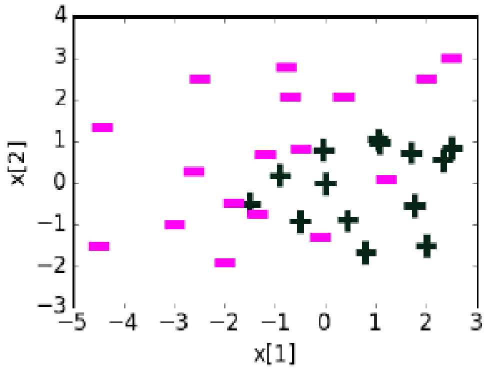


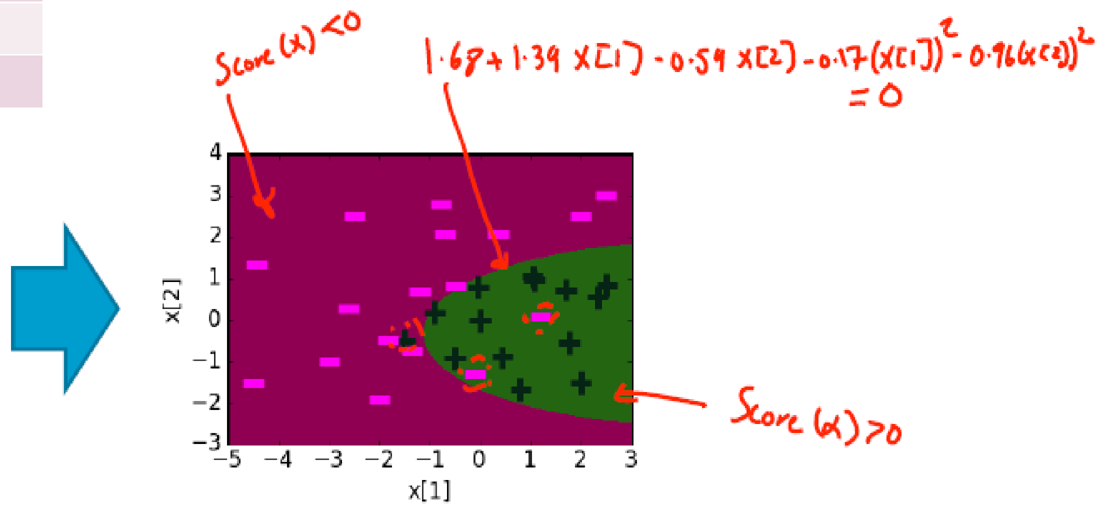


Overfitting in classification

Quadratic features (in 2d)

Feature	Value	Coefficient learned
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96



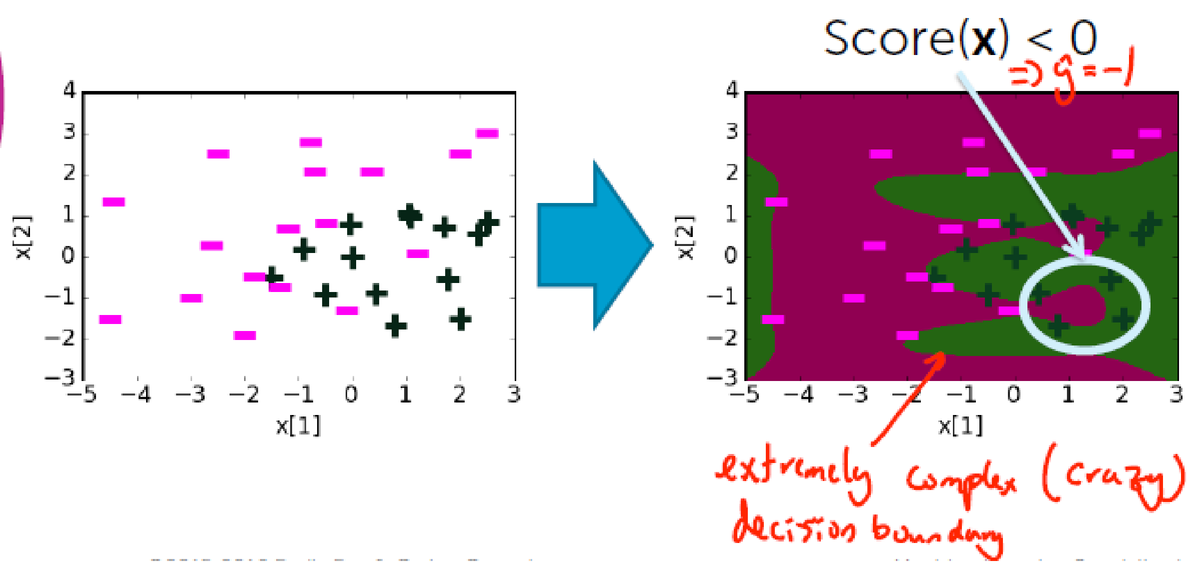


Overfitting in classification

Degree 6 features (in 2d)

Feature	Value	Coefficient learned
$h_0(\mathbf{x})$	1	21.6
$h_1(\mathbf{x})$	$\mathbf{x}[1]$	5.3
$h_2(\mathbf{x})$	$\mathbf{x}[2]$	-42.7
$h_3(\mathbf{x})$	$(\mathbf{x}[1])^2$	-15.9
$h_4(\mathbf{x})$	$(\mathbf{x}[2])^2$	-48.6
$h_5(\mathbf{x})$	$(\mathbf{x}[1])^3$	-11.0
$h_6(\mathbf{x})$	$(\mathbf{x}[2])^3$	67.0
$h_7(\mathbf{x})$	$(\mathbf{x}[1])^4$	1.5
$h_8(\mathbf{x})$	$(\mathbf{x}[2])^4$	48.0
$h_9(\mathbf{x})$	$(\mathbf{x}[1])^5$	4.4
$h_{10}(\mathbf{x})$	$(\mathbf{x}[2])^5$	-14.2
$h_{11}(\mathbf{x})$	$(\mathbf{x}[1])^6$	0.8
$h_{12}(\mathbf{x})$	$(\mathbf{x}[2])^6$	-8.6

Coefficient values getting large

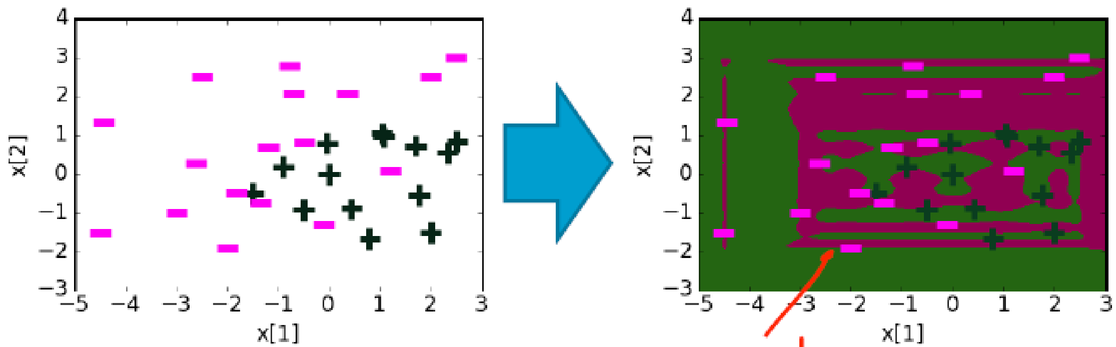


Overfitting in classification

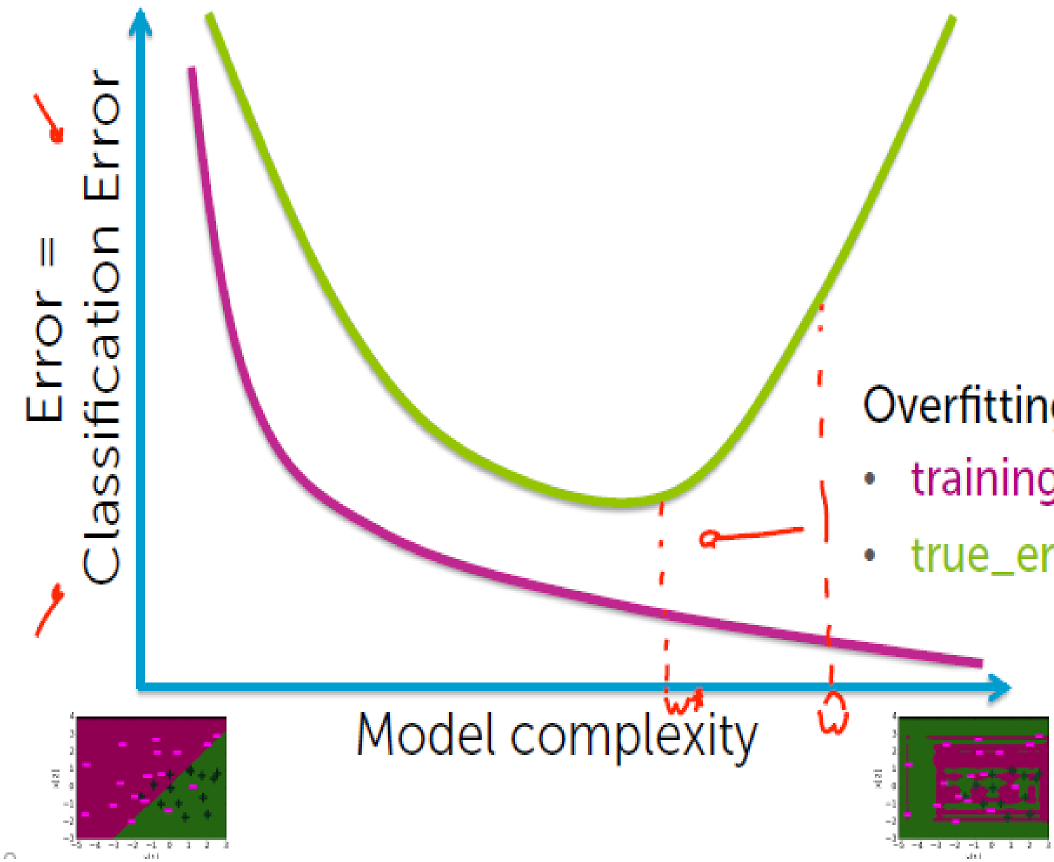
Degree 20 features (in 2d)

Feature	Value	Coefficient learned
$h_0(\mathbf{x})$	1	8.7
$h_1(\mathbf{x})$	$\mathbf{x}[1]$	5.1
$h_2(\mathbf{x})$	$\mathbf{x}[2]$	78.7
…	…	…
$h_{11}(\mathbf{x})$	$(\mathbf{x}[1])^6$	-7.5
$h_{12}(\mathbf{x})$	$(\mathbf{x}[2])^6$	3803
$h_{13}(\mathbf{x})$	$(\mathbf{x}[1])^7$	21.1
$h_{14}(\mathbf{x})$	$(\mathbf{x}[2])^7$	-2406
…	…	…
$h_{37}(\mathbf{x})$	$(\mathbf{x}[1])^{19}$	$-2{*}10^{-6}$
$h_{38}(\mathbf{x})$	$(\mathbf{x}[2])^{19}$	-0.15
$h_{39}(\mathbf{x})$	$(\mathbf{x}[1])^{20}$	$-2{*}10^{-8}$
$h_{40}(\mathbf{x})$	$(\mathbf{x}[2])^{20}$	0.03

Often, overfitting associated with very large estimated coefficients $\hat{\mathbf{w}}$



Overfitting in classification



Overfitting if there exists w*:

- training_error(w*) > training_error(ŵ)
- true_error(w*) < true_error(ŵ)

Overfitting in logistic regression

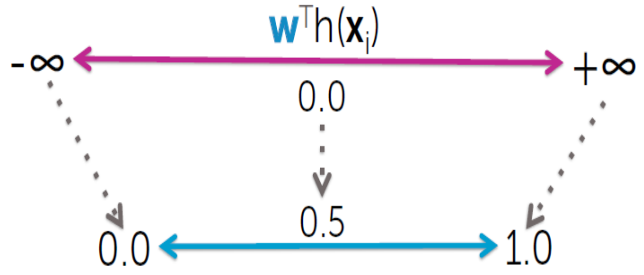
The subtle (negative) consequence of overfitting in logistic regression

Overfitting ➔ Large coefficient values

$\hat{\mathbf{w}}^T h(\mathbf{x}_i)$ is very positive (or very negative)
➔ **sigmoid**($\hat{\mathbf{w}}^T h(\mathbf{x}_i)$) goes to 1 (or to 0)

Model becomes extremely overconfident of predictions

Logistic regression model



$P(y=+1|\mathbf{x}_i,\mathbf{w}) = \text{sigmoid}(\mathbf{w}^T h(\mathbf{x}_i))$

Remember about this probability interpretation

Effect of coefficients on logistic regression model

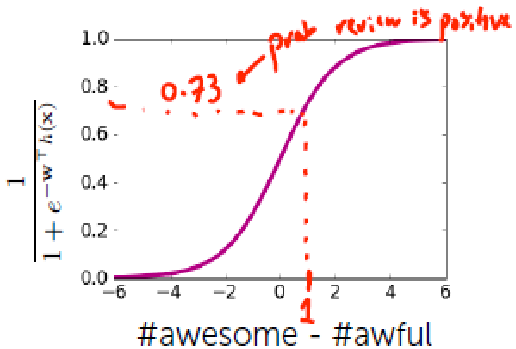
With increasing coefficients model becomes overconfident on predictions

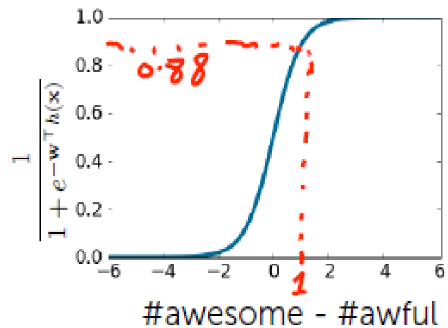
Input **x**: #awesome=2, #awful=1

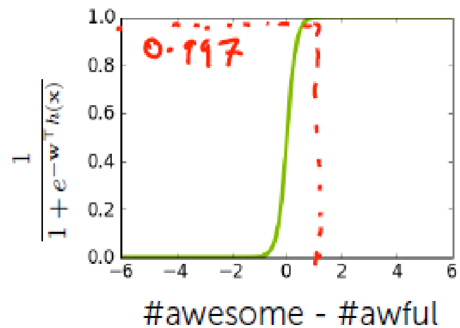
w_0	0
$w_{\#awesome}$	+1
$w_{\#awful}$	-1

w_0	0
$w_{\#awesome}$	+2
$w_{\#awful}$	-2

w_0	0
$w_{\#awesome}$	+6
$w_{\#awful}$	-6







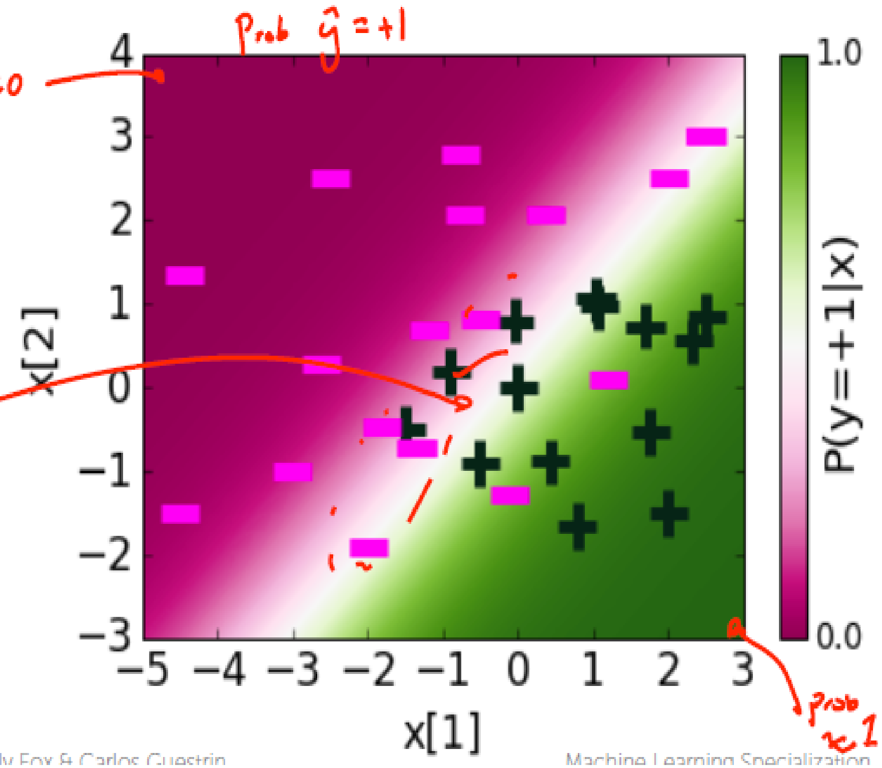
Learned probabilities

Feature	Value	Coefficient learned
$h_0(\mathbf{x})$	1	0.23
$h_1(\mathbf{x})$	x[1]	1.12
$h_2(\mathbf{x})$	x[2]	-1.07

$$P(y = +1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^\top h(\mathbf{x})}}$$

Make sense

wide region of uncertainty



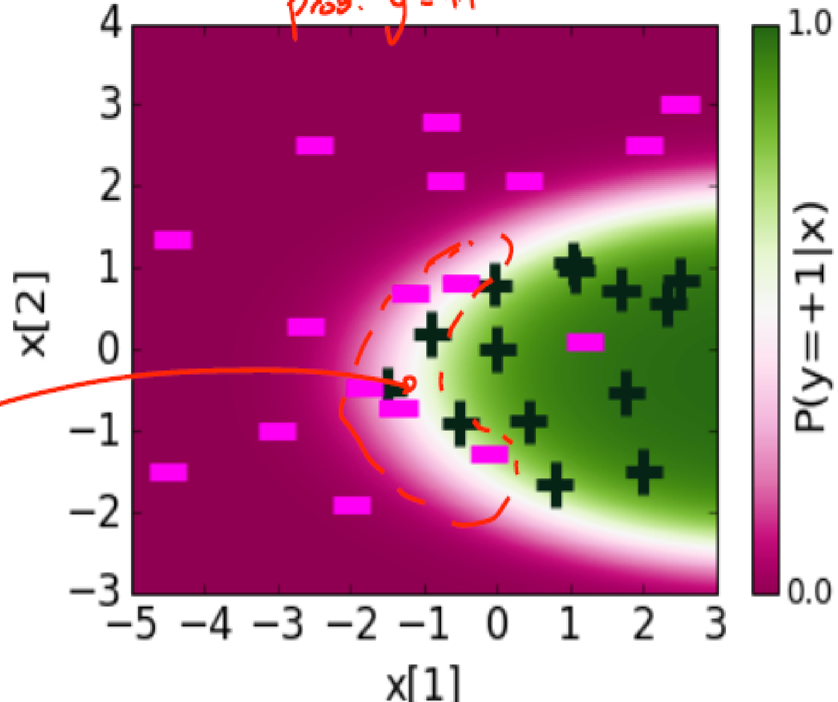
Quadratic features: learned probabilities

Feature	Value	Coefficient learned
$h_0(\mathbf{x})$	1	1.68
$h_1(\mathbf{x})$	$\mathbf{x}[1]$	1.39
$h_2(\mathbf{x})$	$\mathbf{x}[2]$	-0.58
$h_3(\mathbf{x})$	$(\mathbf{x}[1])^2$	-0.17
$h_4(\mathbf{x})$	$(\mathbf{x}[2])^2$	-0.96

better fit to data

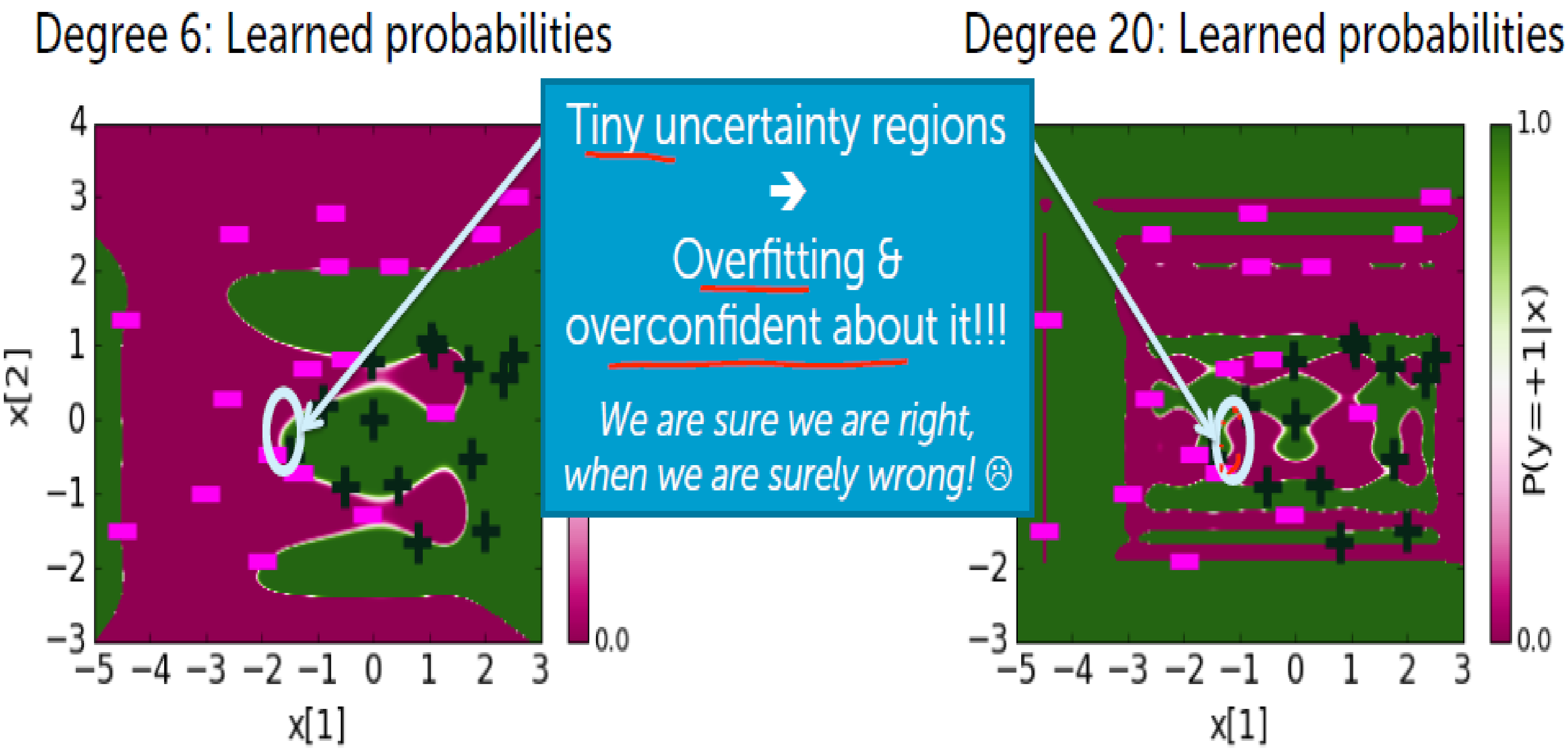
$$P(y=+1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^\top h(\mathbf{x})}}$$

prob. ŷ=+1



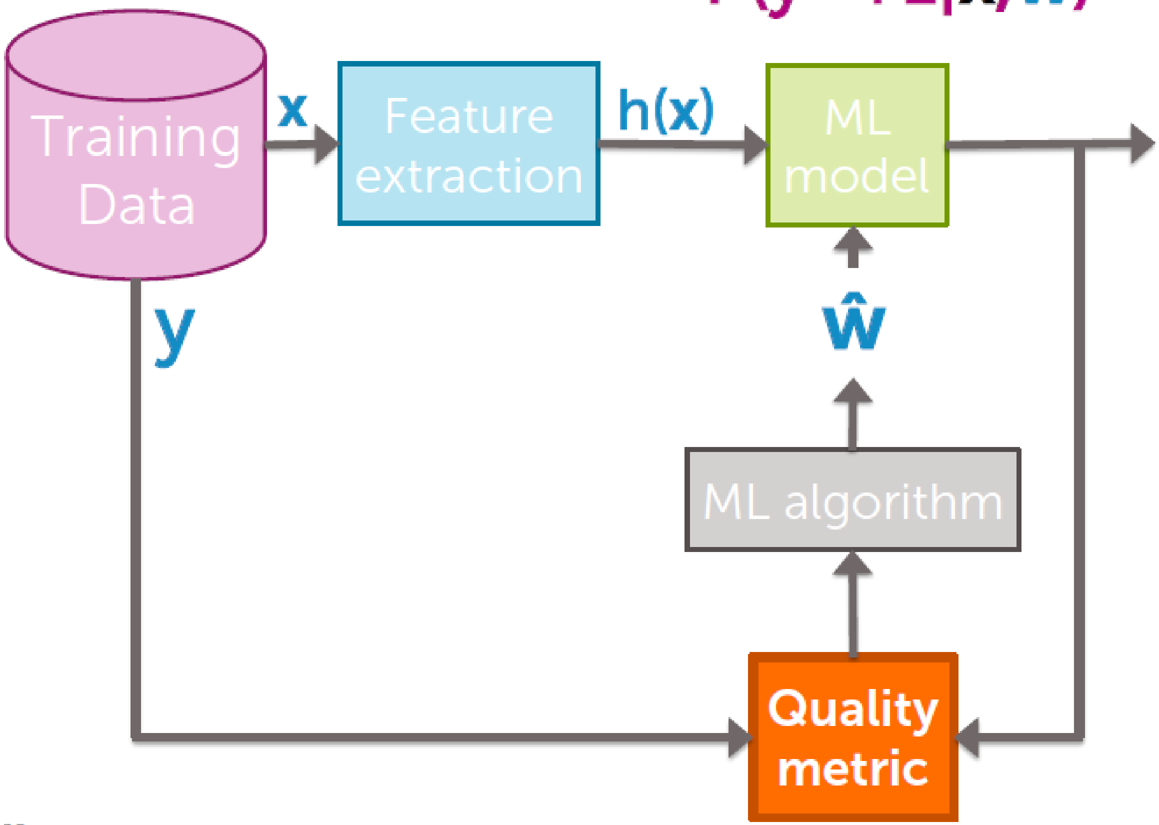
uncertainty region narrower

Overfitting → overconfident predictions



Quality metric → penelazing large coefficients

$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = \frac{1}{1+e^{-\hat{\mathbf{w}}^{\top}h(\mathbf{x})}}$$



Desired total cost format

Want to balance:

i. How well function fits data

ii. Magnitude of coefficients

want to balance

Total quality =

measure of fit - measure of magnitude of coefficients

(data likelihood)
large # = good fit to training data

large # = overfit

Measure of magnitude of logistic regression coefficients

What summary # is indicative of size of logistic regression coefficients?

- Sum of squares (L_2 norm)

$$\|w\|_2^2 = w_0^2 + w_1^2 + w_2^2 + \cdots + w_D^2$$

Penalize large Coefficients

- Sum of absolute value (L_1 norm)

$$\|w\|_1 = |w_0| + |w_1| + |w_2| + \cdots + |w_D|$$

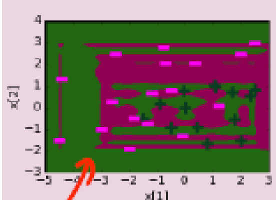
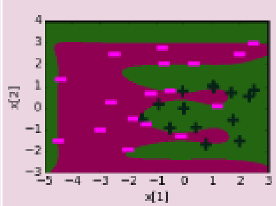
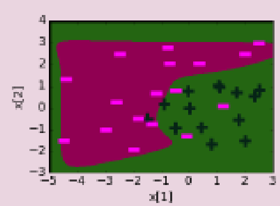
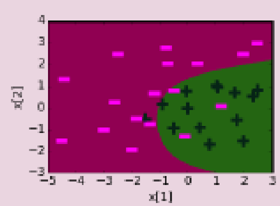
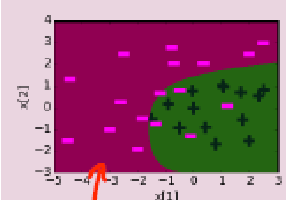
Sparse solution

Visualizing effect of regularisation

Degree 20 features,
effect of regularization penalty λ

very large

smaller coefficients

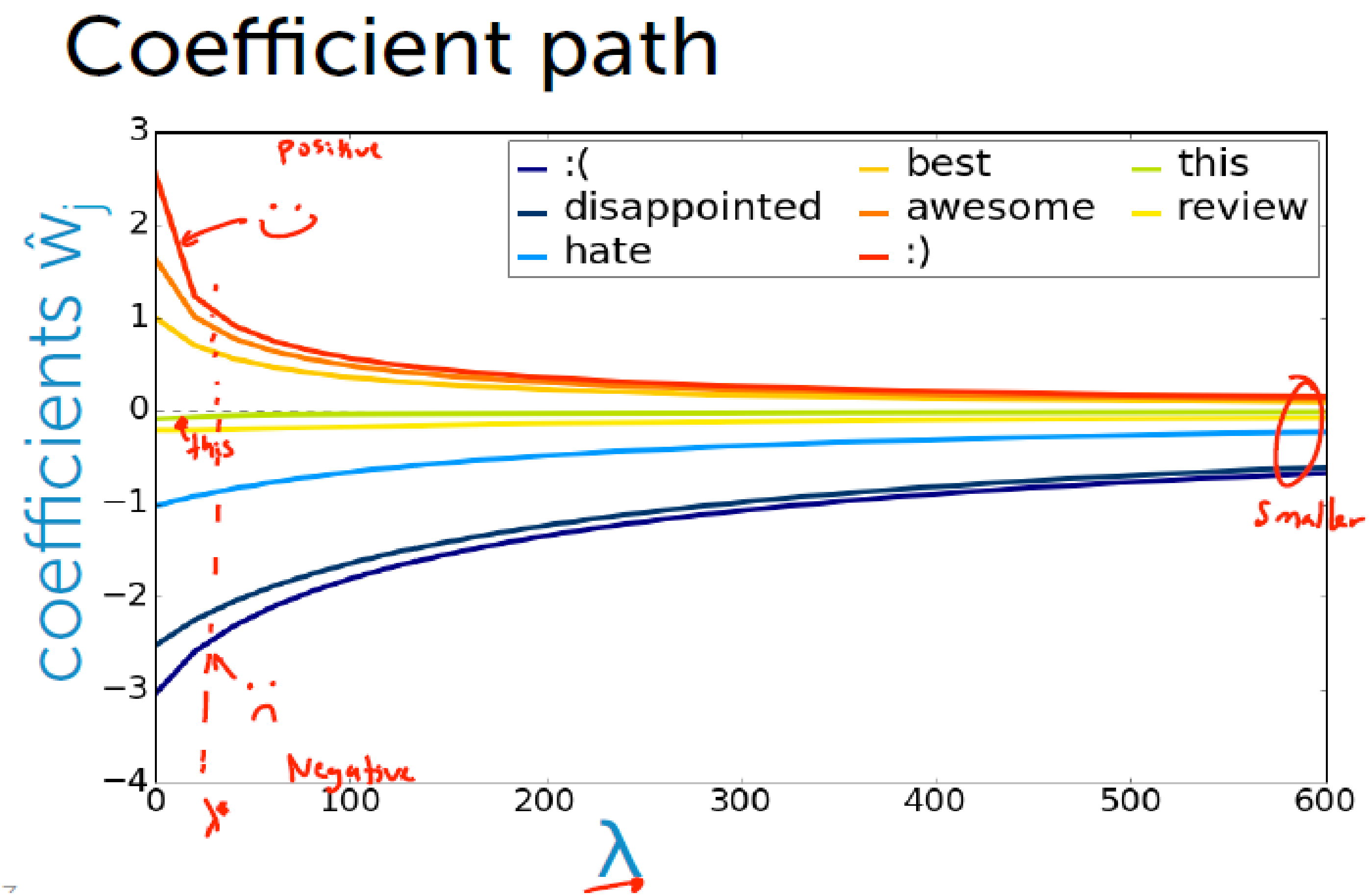
Regularization	λ = 0	λ = 0.00001	λ = 0.001	λ = 1	λ = 10
Range of coefficients	-3170 to 3803	-8.04 to 12.14	-0.70 to 1.25	-0.13 to 0.57	-0.05 to 0.22
Decision boundary					

crazy decision boundary

nicer smoother

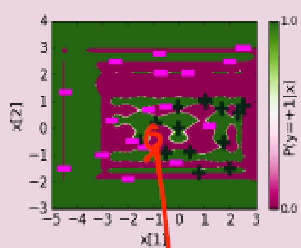
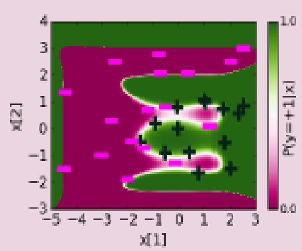
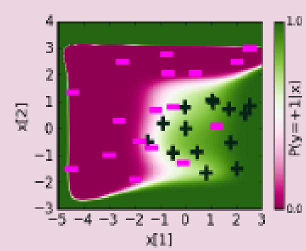
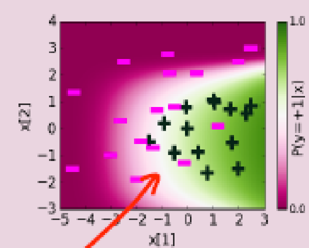
52

Effect of regularisation



Visualizing effect of regularisation

Degree 20 features:
regularization reduces "overconfidence"

Regularization	λ = 0	λ = 0.00001	λ = 0.001	λ = 1
Range of coefficients	-3170 to 3803	-8.04 to 12.14	-0.70 to 1.25	-0.13 to 0.57
Learned probabilities				

highly over confident

Very natural uncertainty region

Sparse logistic regression

Total quality =

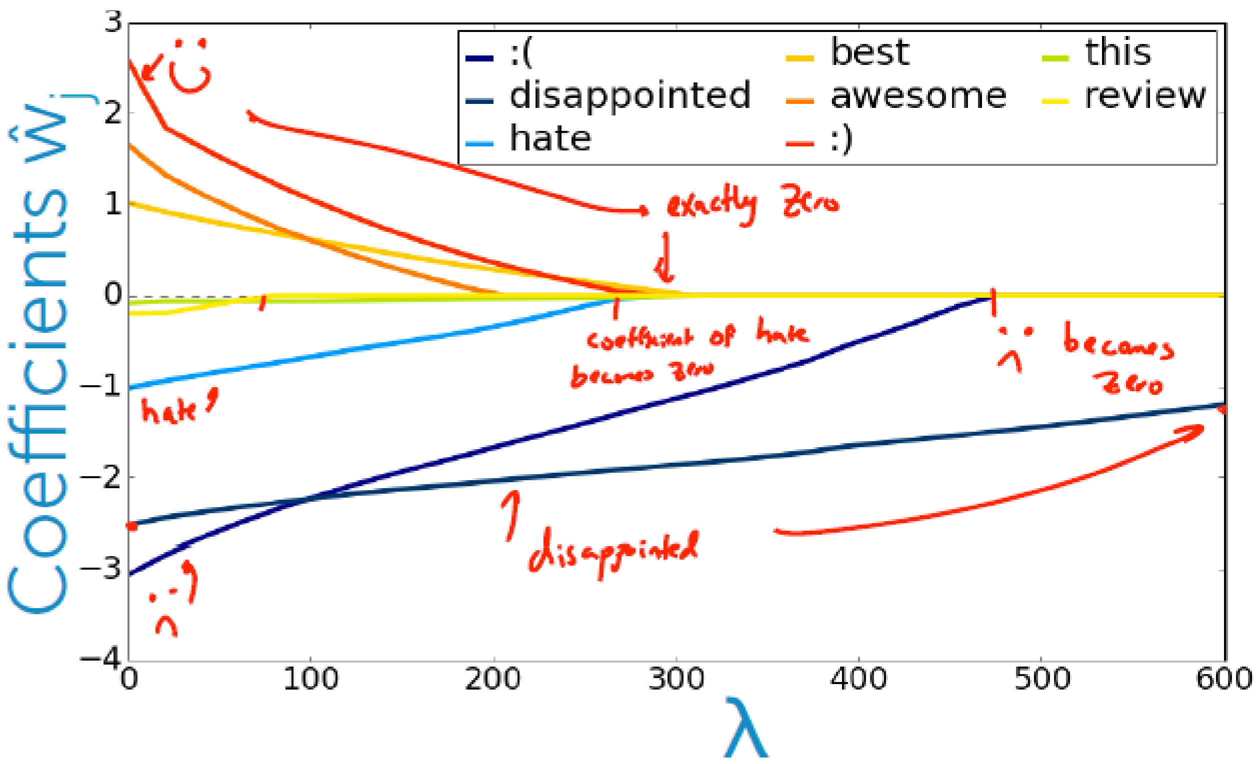
measure of fit - measure of magnitude of coefficients

$\ell(\mathbf{w})$ $\quad ||\mathbf{w}||_1 = |w_0| + \ldots + |w_D|$

L_1 regularized logistic regression

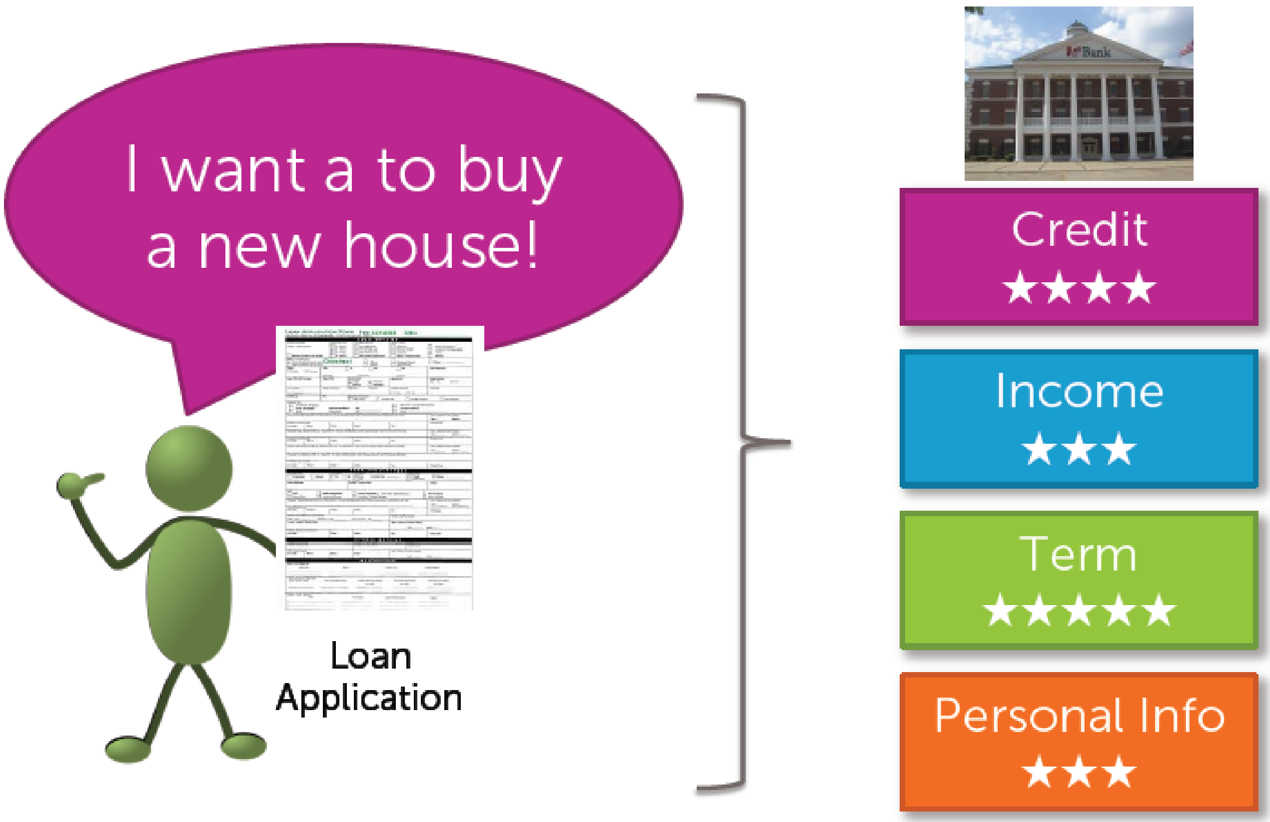
Leads to **sparse** solutions!

L1 regularised logistic regression

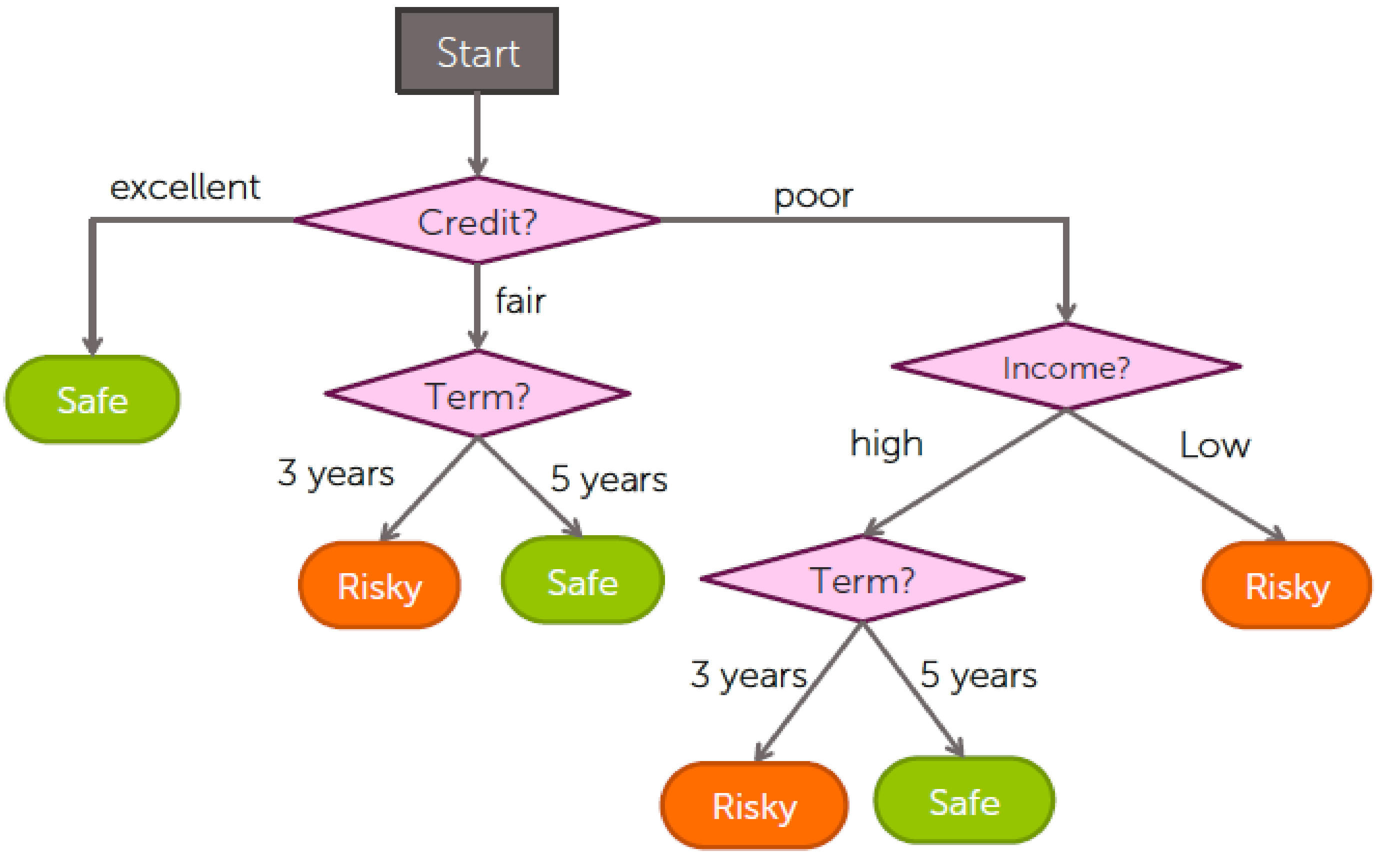


Decision trees

What makes a loan risky?



Classifier: decision trees



Quality metric: Classification error

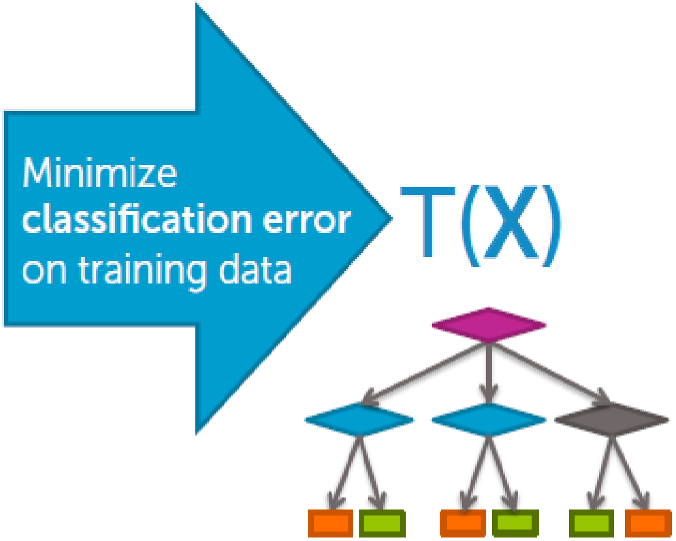
- Error measures fraction of mistakes

$$\text{Error} = \frac{\text{\# incorrect predictions}}{\text{\# examples}}$$

  - Best possible value : 0.0
  - Worst possible value: 1.0

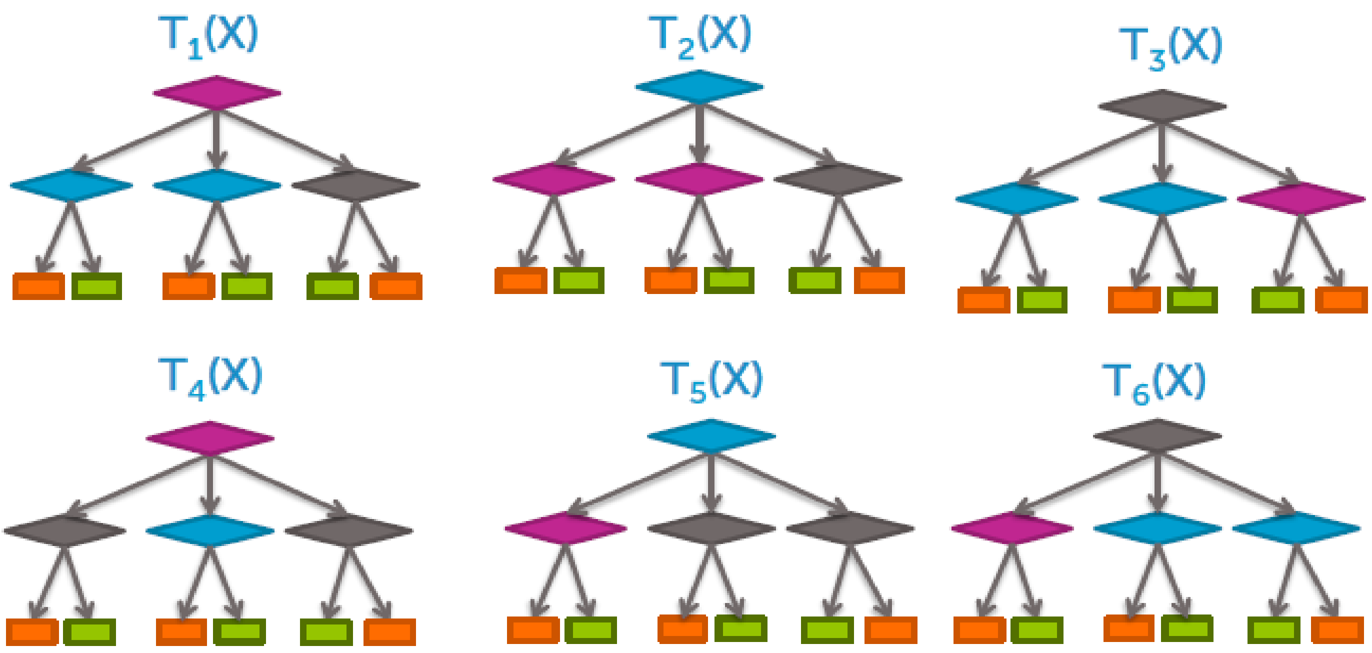
Find the tree with lowest classification error

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	5 yrs	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	low	safe
poor	3 yrs	high	risky
poor	5 yrs	low	safe
fair	3 yrs	high	safe



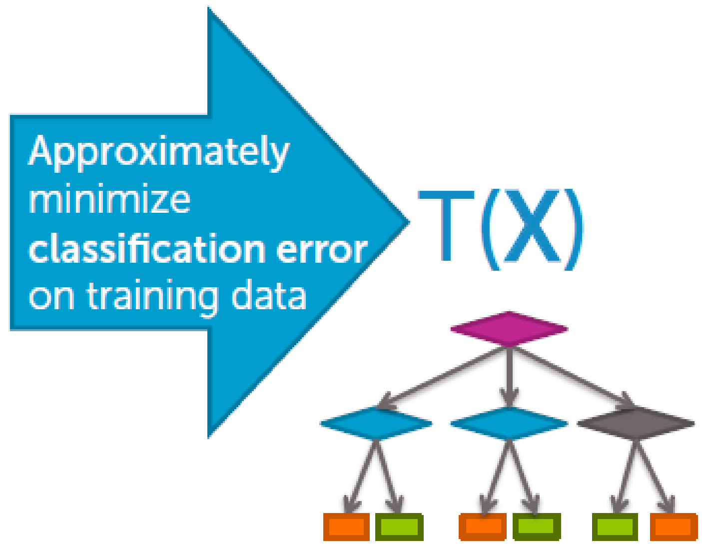
How do we find the best tree?

Exponentially large number of possible trees makes decision tree learning hard!
(NP-hard problem)



Simple (greedy) algorithm finds good tree

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	5 yrs	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	low	safe
poor	3 yrs	high	risky
poor	5 yrs	low	safe
fair	3 yrs	high	safe



Greedy decision tree learning

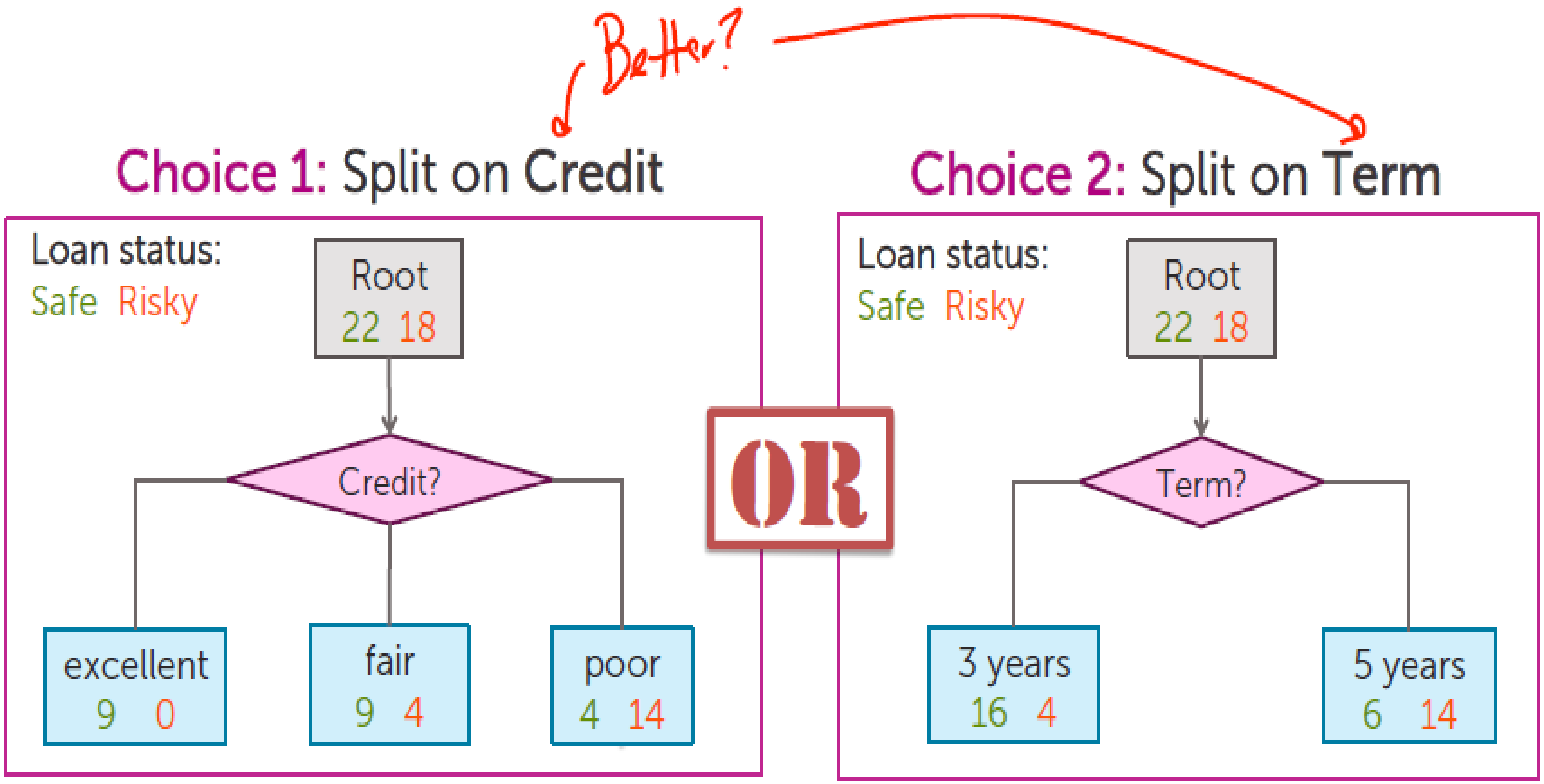
- Step 1: Start with an empty tree
- Step 2: Select a feature to split data
- For each split of the tree:
  - Step 3: If nothing more to, make predictions
  - Step 4: Otherwise, go to Step 2 & continue (recurse) on this split

Problem 1: Feature split selection

Problem 2: Stopping condition

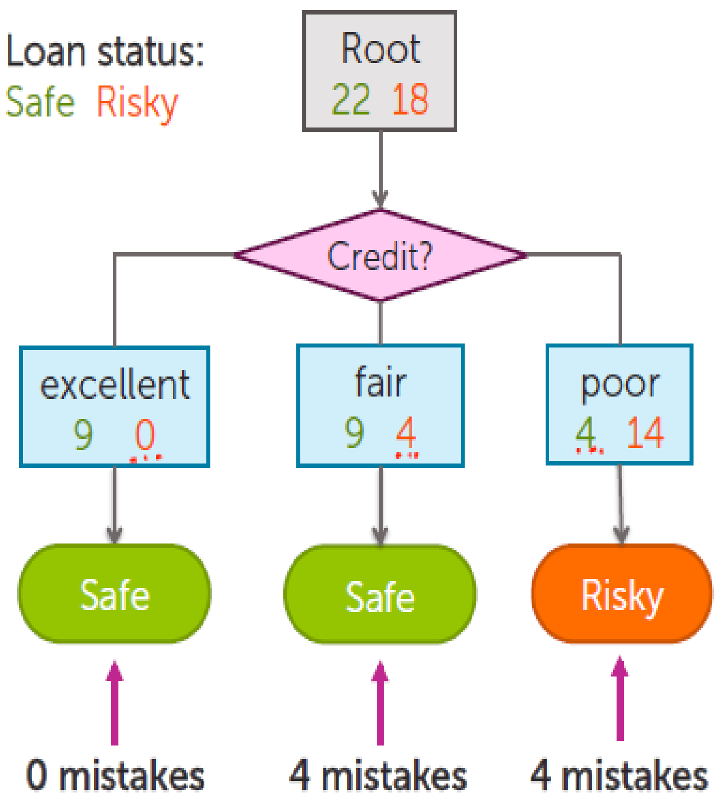
Recursion

How do we select the best feature to split on?



Classification error

Choice 1: Split on Credit



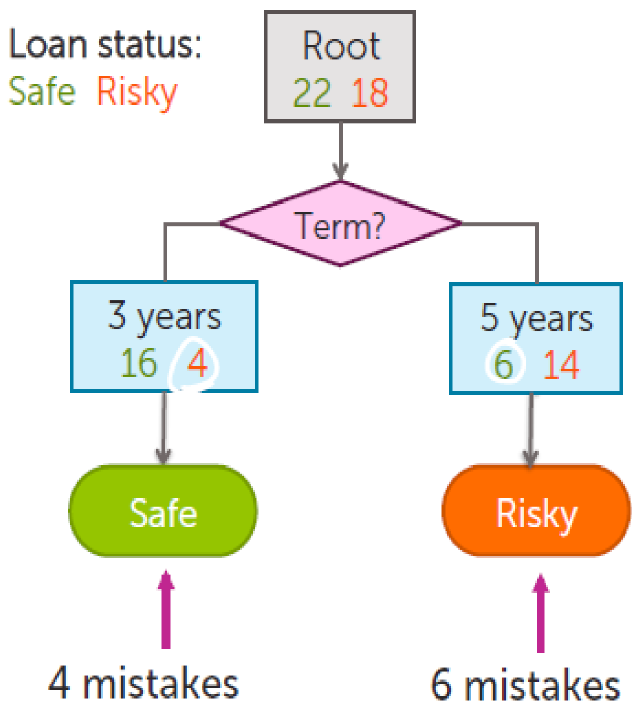
$$\text{Error} = \frac{4+4}{40}$$

$$= 0.20$$

Tree	Classification error
(root)	0.45
Split on credit	0.2

Classification error

Choice 2: Split on **Term**



$$\text{Error} = \frac{4+6}{40}$$

$$= 0.25$$

Tree	Classification error
(root)	0.45
Split on credit	0.2
Split on term	0.25

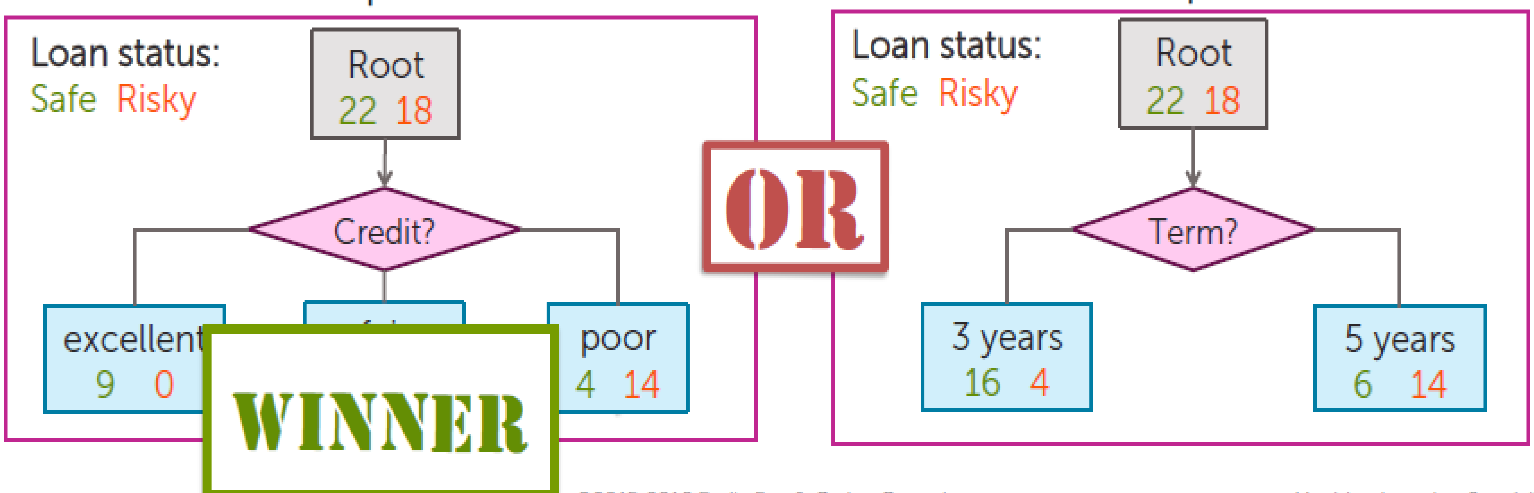
Choice 1 vs Choise 2

Tree	Classification error
(root)	0.45
split on **credit**	0.2
split on **loan term**	0.25

← First split!

Choice 1: Split on Credit

Choice 2: Split on Term



Greedy decision tree learning algorithm

- Step 1: Start with an empty tree
- Step 2: Select a feature to split data
- For each split of the tree:
  - Step 3: If nothing more to, make predictions
  - Step 4: Otherwise, go to Step 2 & continue (recurse) on this split

Pick feature split leading to lowest classification error

Greedy decision tree algorithm

- Step 1: Start with an empty tree
- Step 2: Select a feature to split data
- For each split of the tree:
  - Step 3: If nothing more to, make predictions
  - Step 4: Otherwise, go to Step 2 & continue (recurse) on this split

Pick feature split leading to lowest classification error

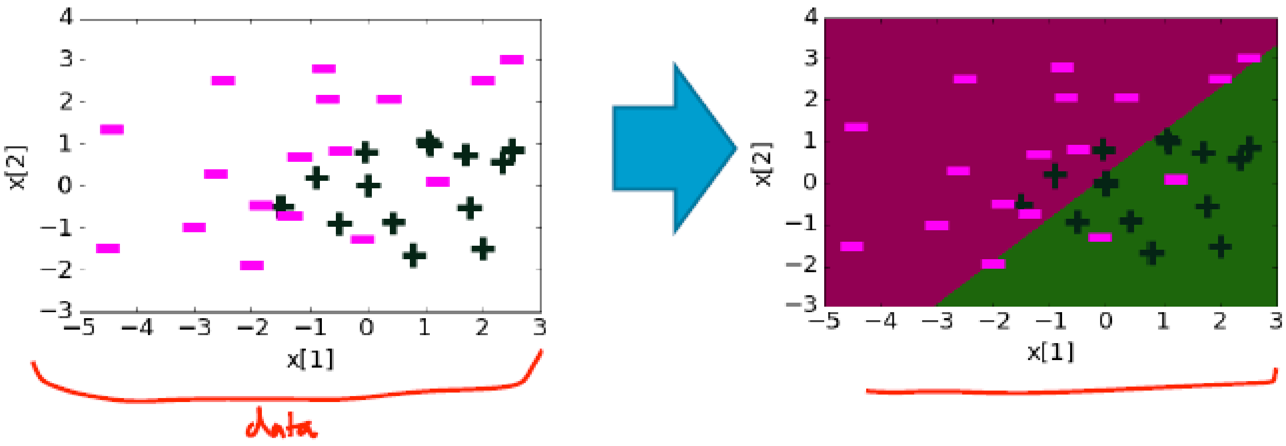
Stopping conditions 1 & 2

Recursion

Decision trees vs logistic regression

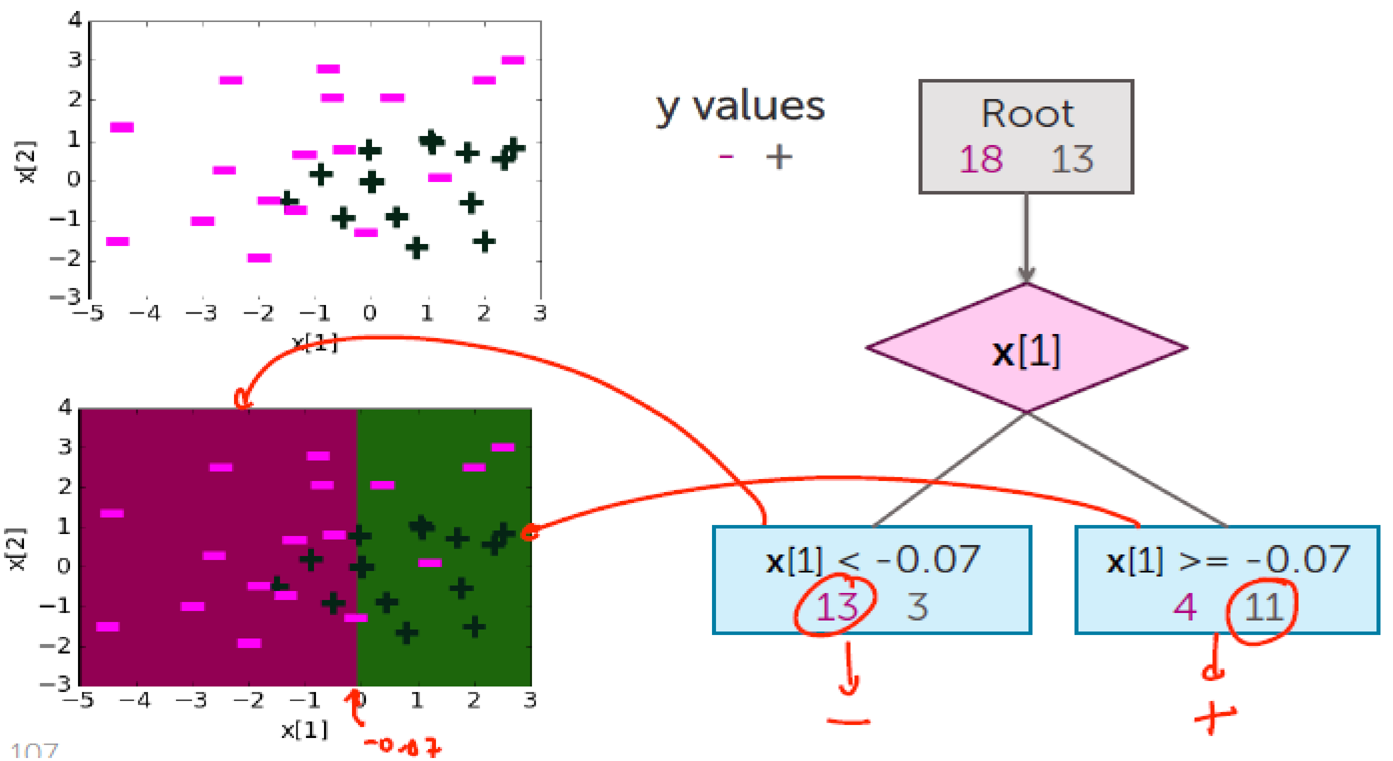
Logistic regression

Feature	Value	Weight Learned
$h_0(\mathbf{x})$	1	0.22
$h_1(\mathbf{x})$	**x**[1]	1.12
$h_2(\mathbf{x})$	**x**[2]	-1.07



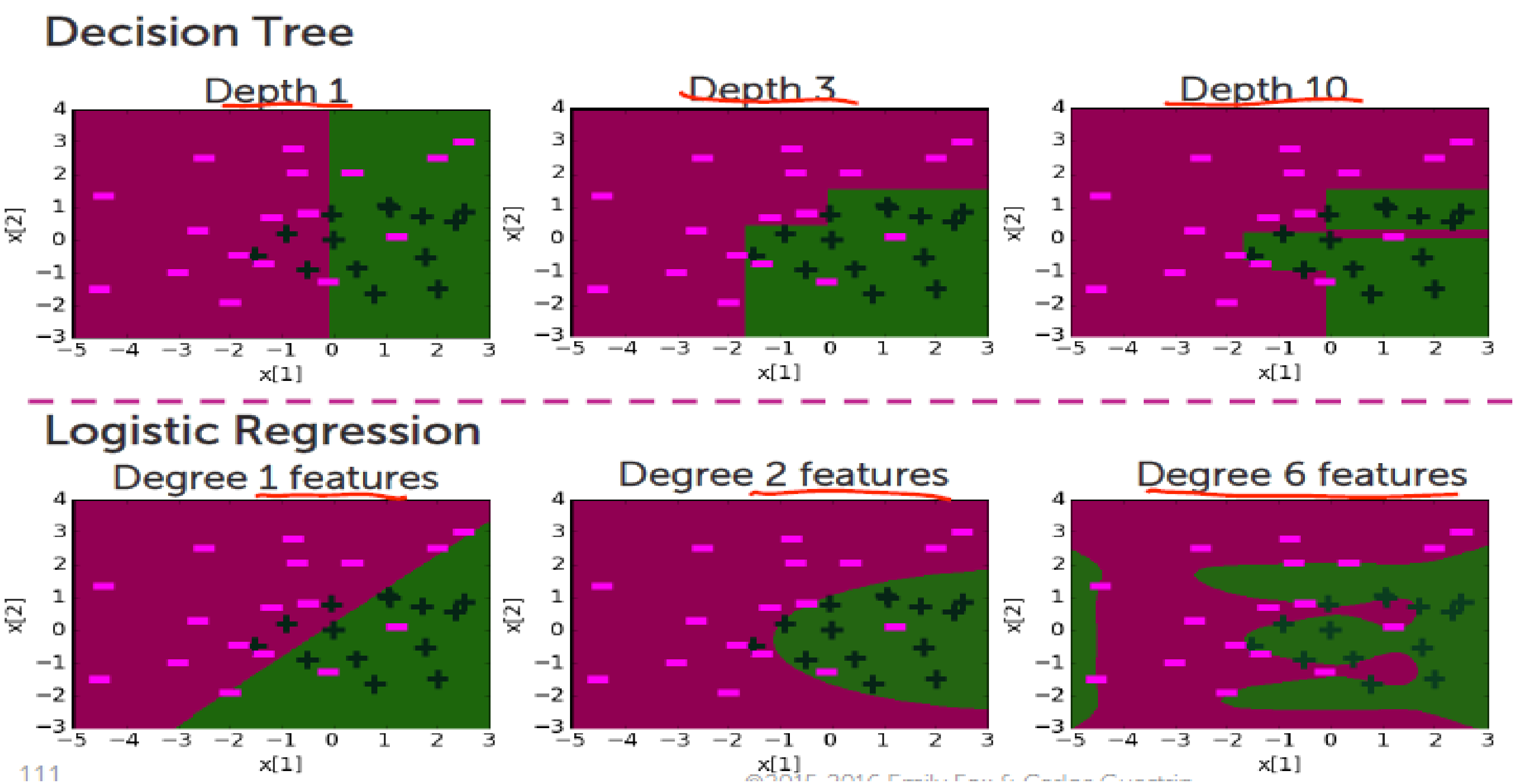
Decision trees vs logistic regression

Depth 1: Split on **x**[1]



Decision tree vs logistic regression

Comparing decision boundaries



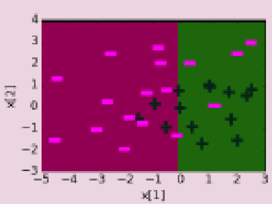
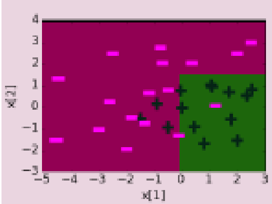
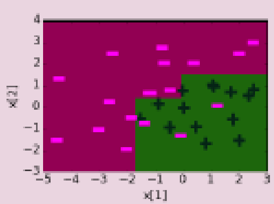
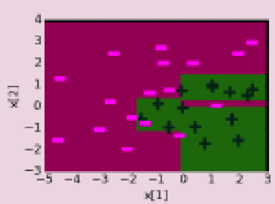
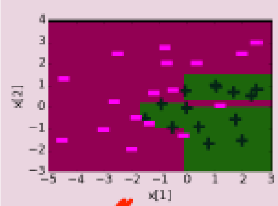
Overfitting in decision trees

Overfitting in decision tree

What happens when we increase depth?

Training error reduces with depth →

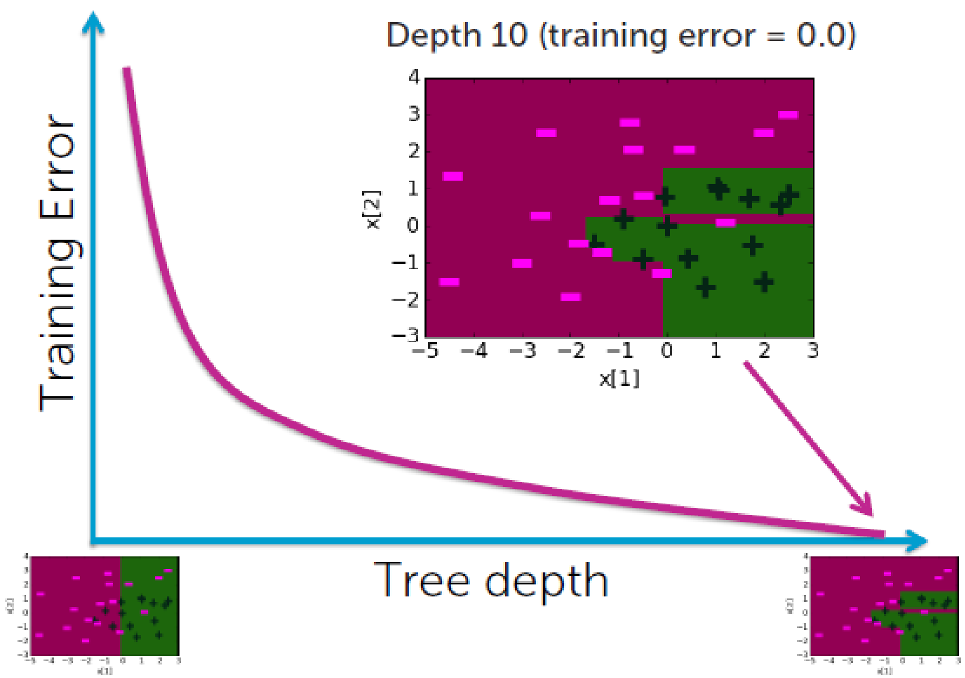
Big warning!!

Tree depth	depth = 1	depth = 2	depth = 3	depth = 5	depth = 10
Training error	0.22	0.13	0.10	0.03	0.00
Decision boundary					

complexity of decision boundary →

Overfitting in decision tree

Deeper trees → lower training error



Early stopping

1. Limit tree depth: Stop splitting after a certain depth

2. Classification error: Do not consider any split that does not cause a sufficient decrease in classification error

3. Minimum node "size": Do not split an intermediate node which contains too few data points

Greedy decision tree learning

- Step 1: Start with an empty tree
- Step 2: Select a feature to split data
- For each split of the tree:
  - Step 3: If nothing more to, make predictions ← Majority
  - Step 4: Otherwise, go to Step 2 & continue (recurse) on this split

Stopping conditions 1 & 2

or

Early stopping conditions 1, 2 & 3

Recursion

Strategies for handling missing data

Handling missing data

Missing value skipping: Ideas 1 & 2

Idea 1: Skip data points where any feature contains a missing value

- Make sure only a few data points are skipped

Idea 2: Skip an entire feature if it's missing for many data points

- Make sure only a few features are skipped

Handling missing data

Common (simple) rules for purification by imputation

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	?	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	high	safe
poor	3 yrs	high	risky
poor	?	low	safe
fair	?	high	safe

Impute each feature with missing values:

1. Categorical features use mode: Most popular value (mode) of non-missing x_i
2. Numerical features use average or median: Average or median value of non-missing x_i

Many advanced methods exist,
e.g., expectation-maximization (EM) algorithm

Handling missing data

Missing value imputation: Pros and Cons

Pros

- Easy to understand and implement
- Can be applied to any model
  (decision trees, logistic regression, linear regression,...)
- Can be used at prediction time: use same imputation rules

Cons

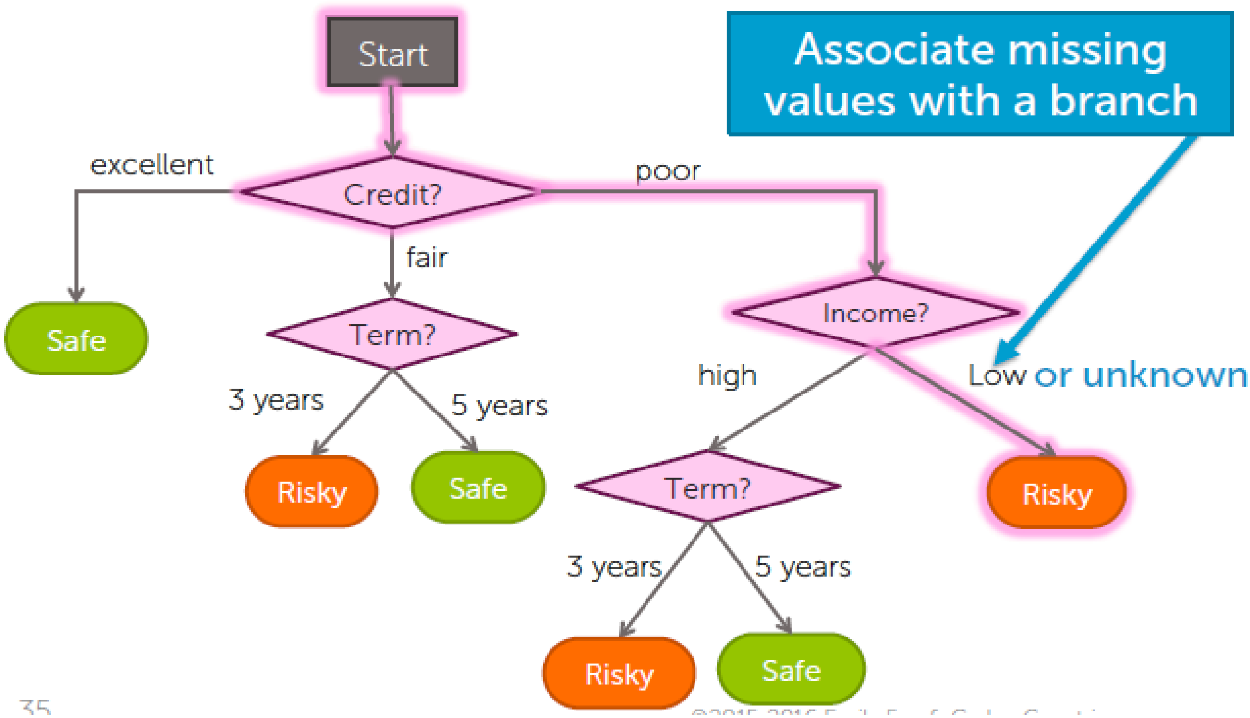
- May result in systematic errors

Example: Feature "age" missing in all banks in Washington by state law

Idea 3: addapt algorithm

Add missing values to the tree definition

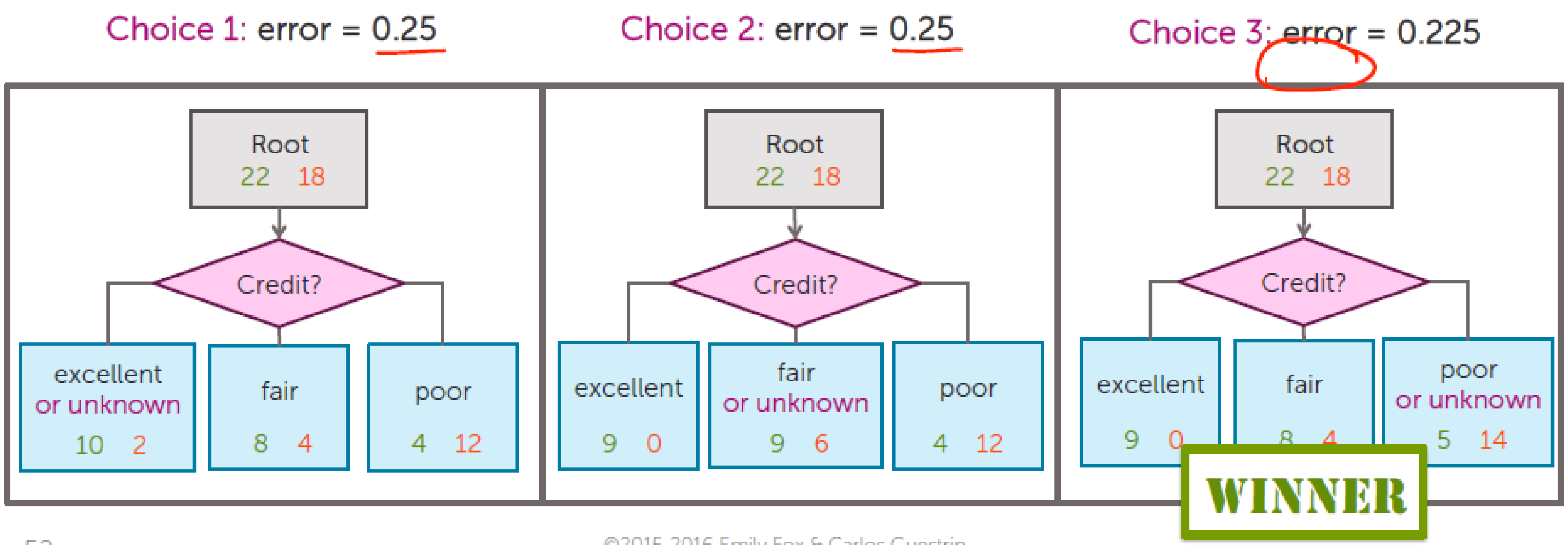
$\mathbf{x}_i$ = (Credit = poor, Income = ?, Term = 5 years)



Feature split selection with missing data

Use classification error to decide

Best choice → assign "unknown" to Credit = poor



Idea 3: addapt algorithm

Explicitly handling missing data by learning algorithm: Pros and Cons

Pros

- Addresses training and prediction time
- More accurate predictions

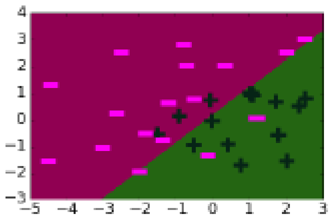
Cons

- Requires modification of learning algorithm
  - Very simple for decision trees

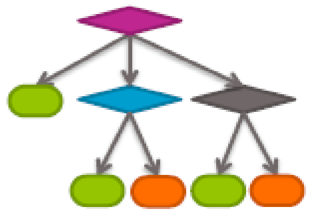
Ensemble classifiers and boosting

Simple classifiers

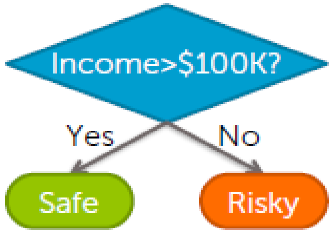
Simple (weak) classifiers are good!



Logistic regression w. simple features



Shallow decision trees



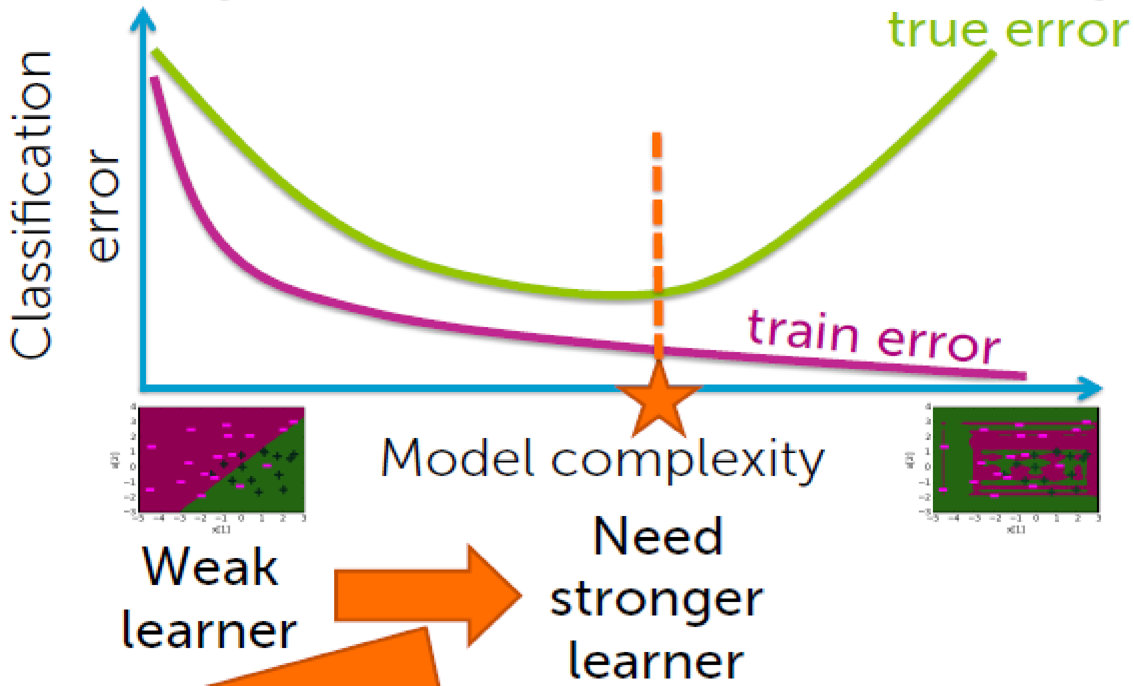
Decision stumps

Low variance. Learning is fast!

But high bias…

Simple classifiers

Finding a classifier that's just right



Option 1: add more features or depth
Option 2: ?????

Can they be combined?

Boosting question

"Can a set of weak learners be combined to create a stronger learner?" *Kearns and Valiant (1988)*

↓

Yes! *Schapire (1990)*

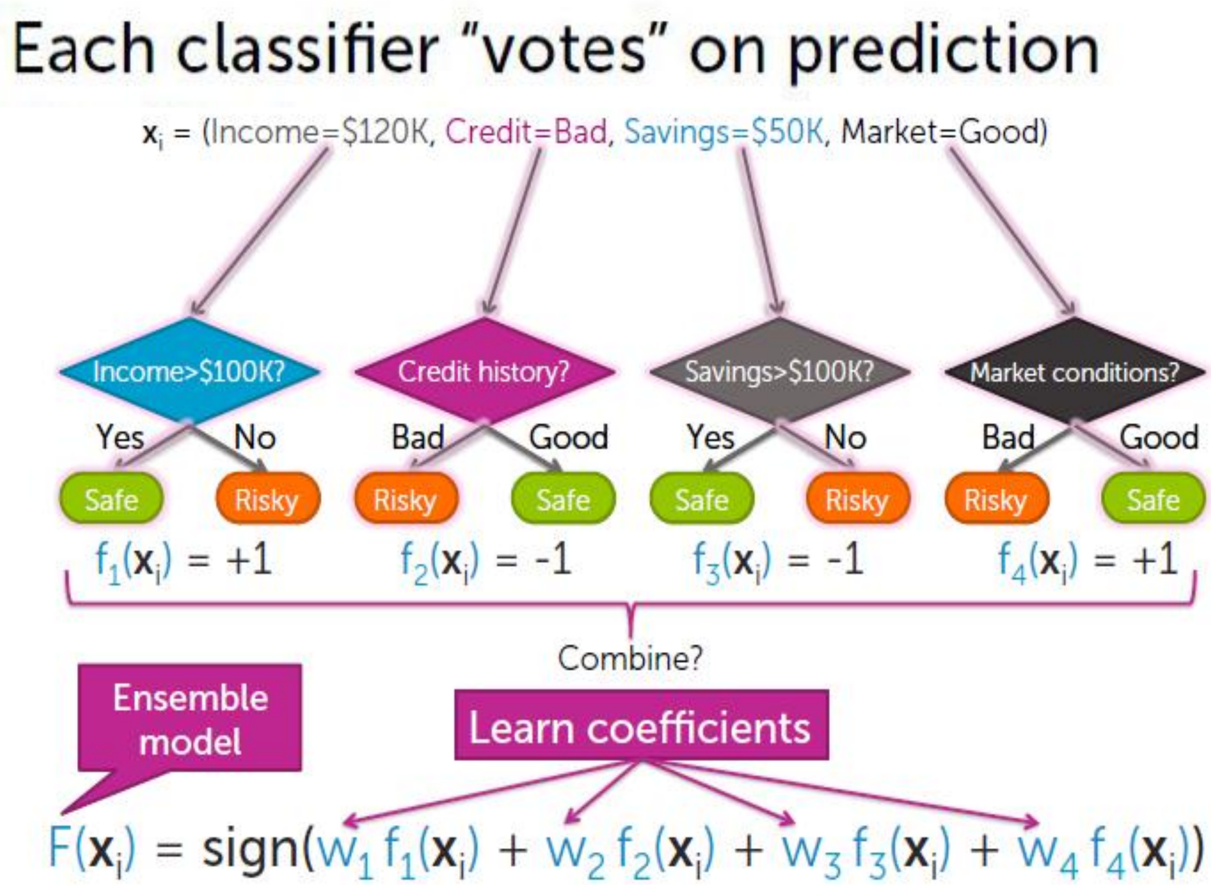
↓

Boosting

↓

Amazing impact: • simple approach • widely used in industry • wins most Kaggle competitions

Ensemble methods



w_1	2
w_2	1.5
w_3	1.5
w_4	0.5

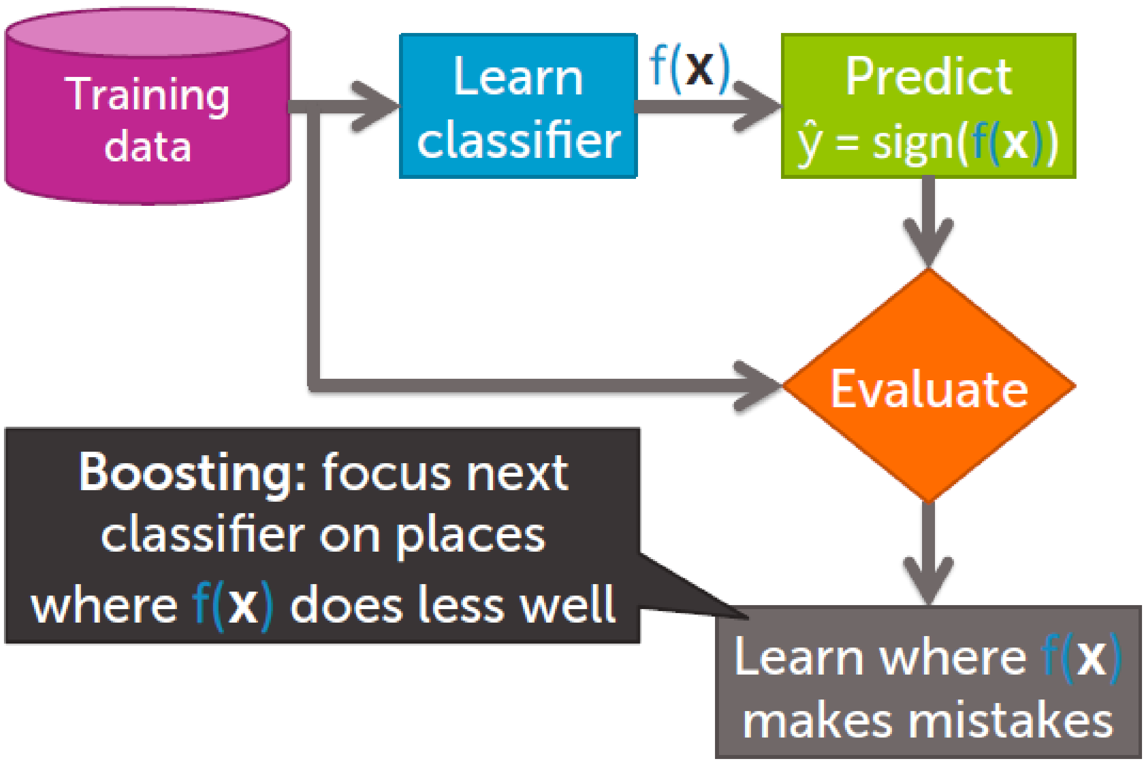
Ensemble classifier

- Goal:
  - Predict output y
    - Either +1 or -1
  - From input **x**
- Learn ensemble model:
  - Classifiers: $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_T(\mathbf{x})$
  - Coefficients: $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_T$
- Prediction:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathbf{w}}_t f_t(\mathbf{x})\right)$$

Boosting

Boosting = Focus learning on "hard" points



Weighted data

Learning on weighted data:
More weight on "hard" or more important points

- Weighted dataset:
  - Each $\mathbf{x}_i, y_i$ weighted by α_i
    - More important point = higher weight α_i

- Learning:
  - Data point j counts as α_i data points
    - E.g., $\alpha_i = 2$ ➔ count point twice

Weighted data

Learning from weighted data in general

- Usually, learning from weighted data
  - Data point i counts as α_i data points

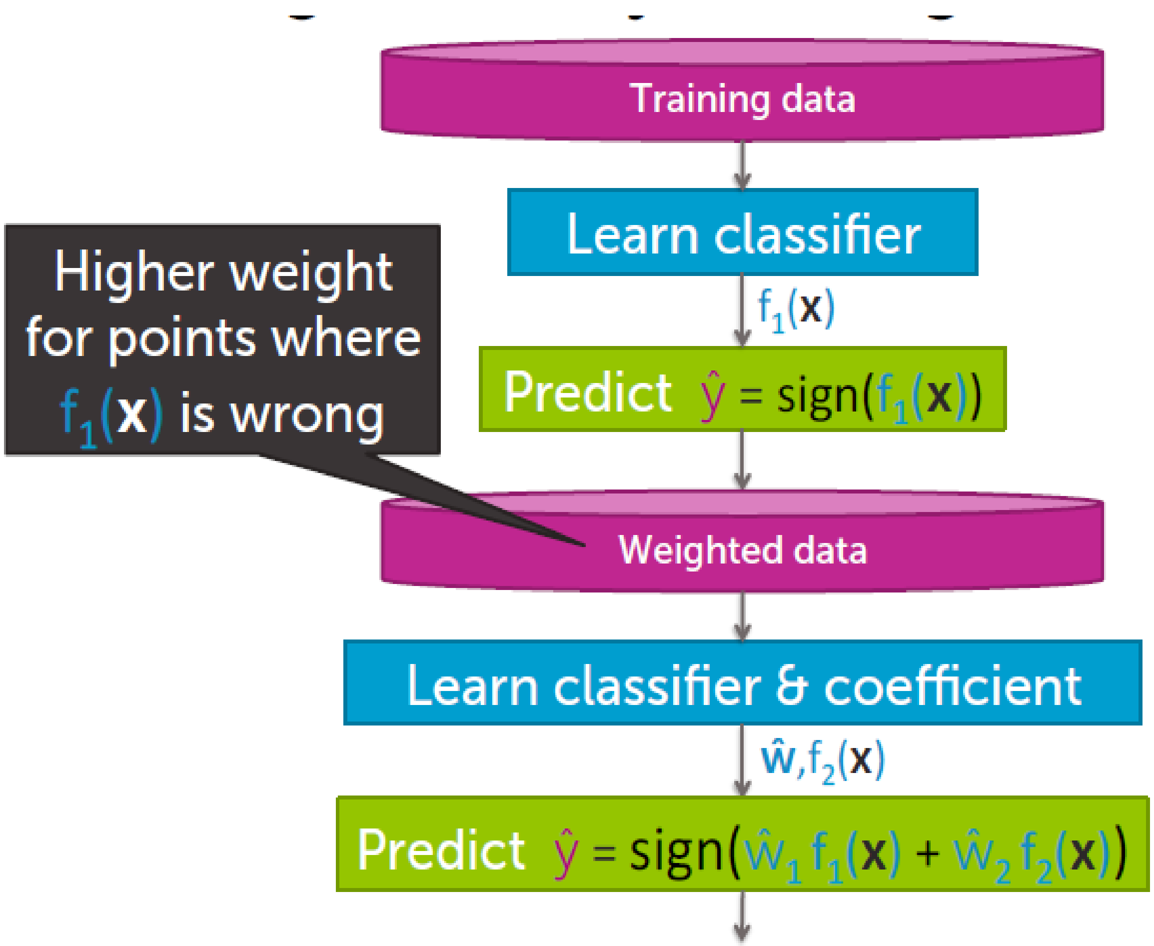
- E.g., gradient ascent for logistic regression:

Sum over data points

$$\mathbf{w}_j^{(t+1)} \leftarrow \mathbf{w}_j^{(t)} + \eta \sum_{i=1}^{N} \alpha_i h_j(\mathbf{x}_i)\Big(\mathbb{1}[y_i = +1] - P(y = +1 \mid \mathbf{x}_i, \mathbf{w}^{(t)})\Big)$$

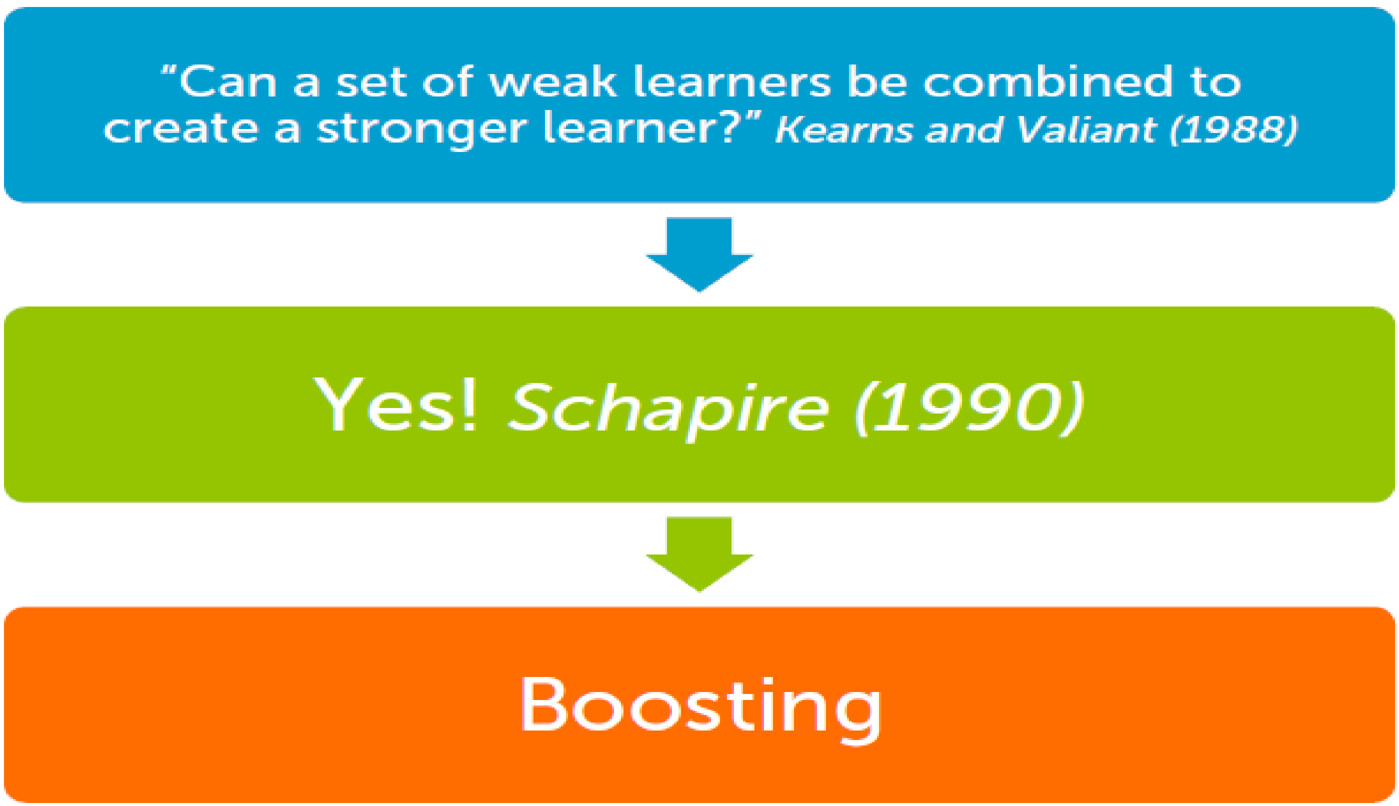
Weigh each point by α_i

Boosting = greedy learning ensembles from data



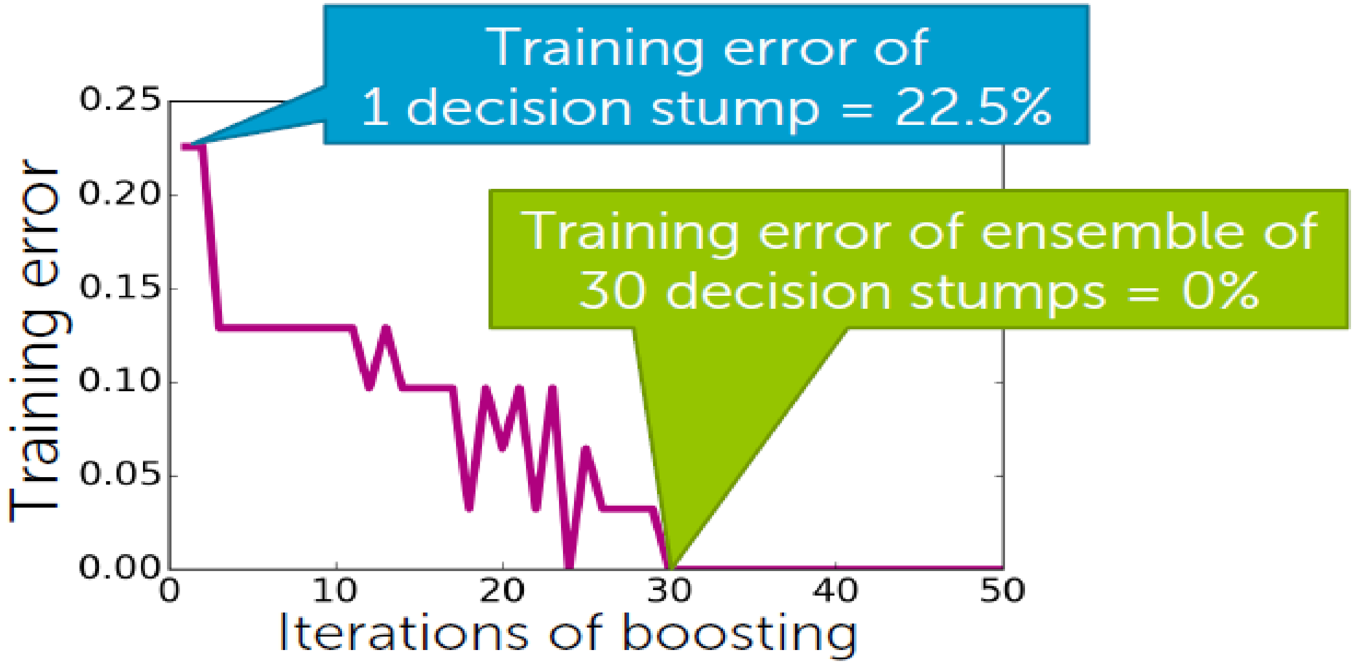
Boosting convergence & overfitting

Boosting question revisited



Boosting convergence & overfitting

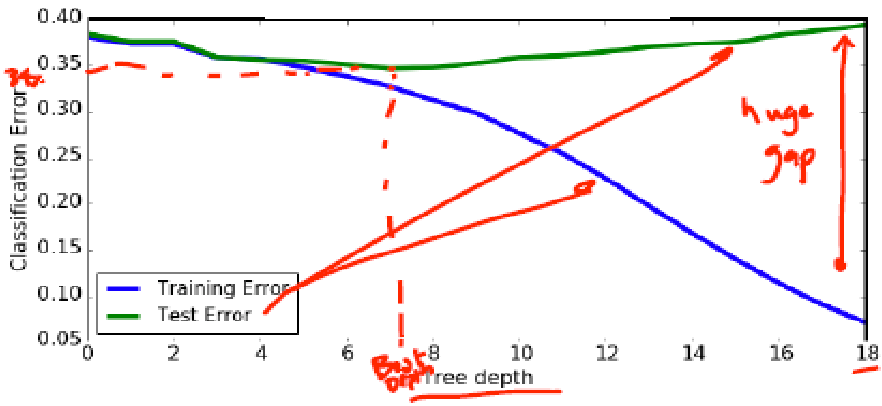
After some iterations,
training error of boosting goes to zero!!!



Boosted decision stumps on toy dataset

Example

Decision trees on loan data

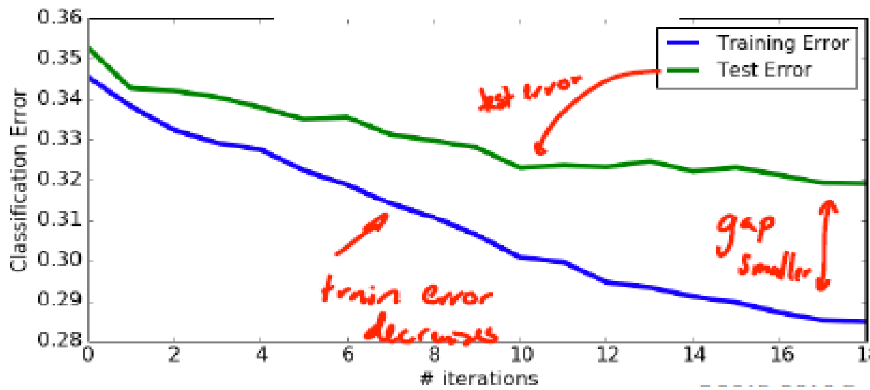


39% test error

Overfitting

8% training error

Boosted decision stumps on loan data



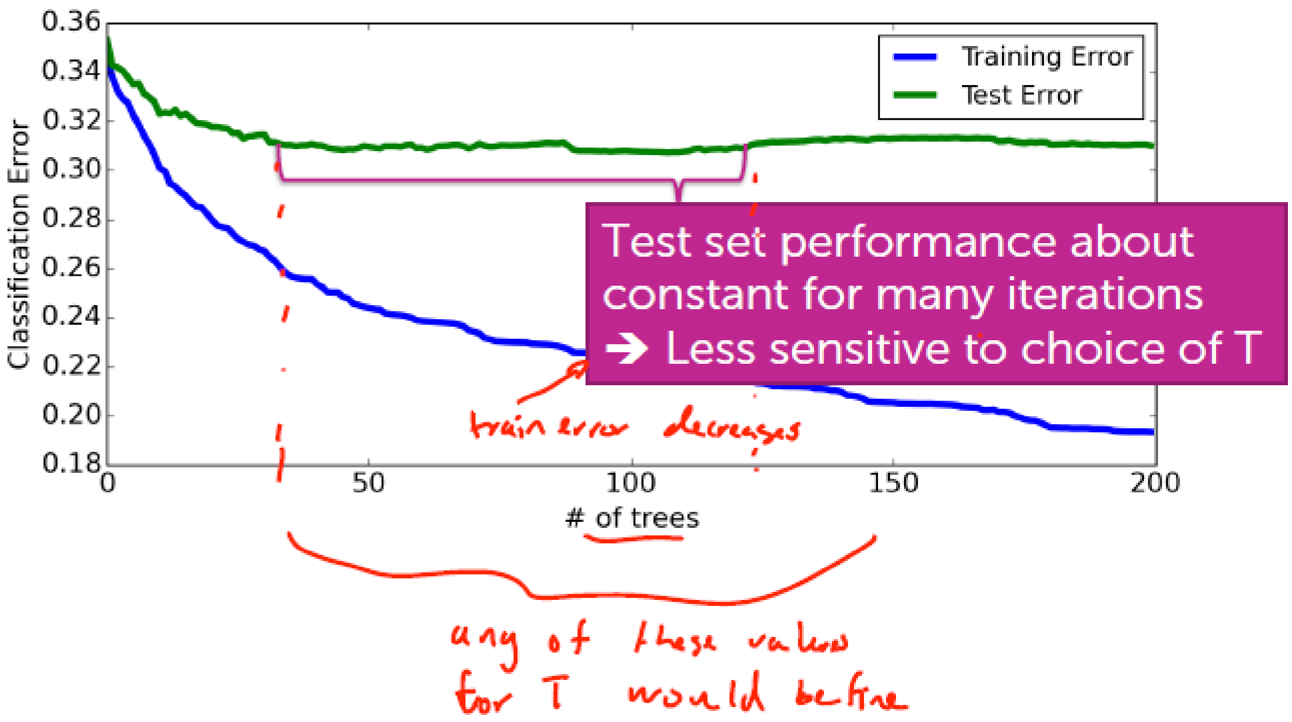
32% test error

Better fit & lower test error

28.5% training error

Example

Boosting tends to be robust to overfitting



Boosting: summary

Variants of boosting and related algorithms

There are hundreds of variants of boosting, most important:

Gradient boosting

- Like AdaBoost, but useful beyond basic classification

Many other approaches to learn ensembles, most important:

Random forests

- Bagging: Pick random subsets of the data
  - Learn a tree in each subset
  - Average predictions
- Simpler than boosting & easier to parallelize
- Typically higher error than boosting for same number of trees (# iterations T)

Boosting: summary

Impact of boosting
(spoiler alert… HUGE IMPACT)

Amongst most useful ML methods ever created

Extremely useful in computer vision

- Standard approach for face detection, for example

Used by most winners of ML competitions (Kaggle, KDD Cup,…)

- Malware classification, credit fraud detection, ads click through rate estimation, sales forecasting, ranking webpages for search, Higgs boson detection,…

Most deployed ML systems use model ensembles

- Coefficients chosen manually, with boosting, with bagging, or others

Classification: summary

- **Models**
  - Linear classifiers
  - Logistic regression
  - Decision trees
  - Ensembles
- **Algorithms**
  - Gradient
  - Stochastic gradient
  - Recursive greedy
  - Boosting
- **Core ML**
  - ~~Alleviating overfitting~~
  - Handling missing data
  - ~~Precision-recall~~
  - ~~Online learning~~

Details

- Derivative of likelihood for logistic regression

The log trick, often used in ML…

- Products become sums:

  $\ln a \cdot b = \ln a + \ln b$ | $\ln \frac{a}{b} = \ln a - \ln b$

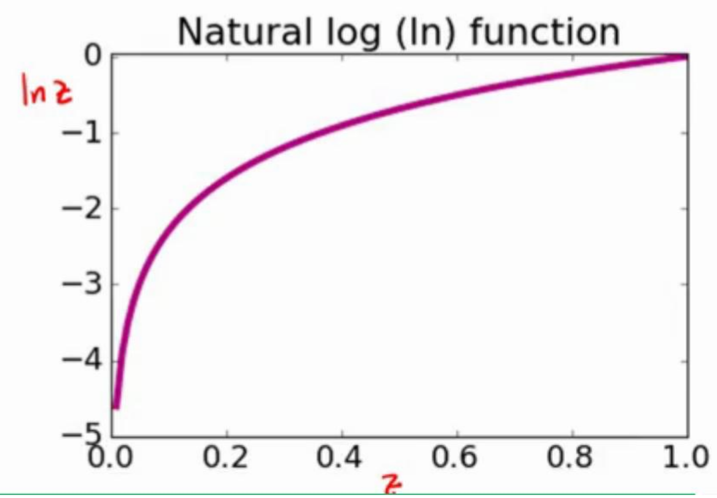
- Doesn't change maximum!
  - If $\hat{\mathbf{w}}$ maximizes f(**w**):

    $\hat{w} = \underbrace{\arg\max_{w}}_{\text{the } w \text{ that makes } f(w) \text{ largest}} f(w)$

  - Then $\hat{\mathbf{w}}_{ln}$ maximizes ln(f(**w**)):

    $\hat{w}_{ln} = \arg\max_{w} \ln(f(w))$

    $\hat{w} = \hat{w}_{ln}$.



Log-likelihood function

- Goal: choose coefficients **w** maximizing likelihood:

$$\ell(\mathbf{w}) = \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

- Math simplified by using log-likelihood – taking (natural) log:

$$\ell\ell(\mathbf{w}) = \ln \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

natural log

Log-likelihood function

Using log to turn products into sums

$$\ln \prod_{i=1}^{N} f_i = \sum_{i=1}^{N} \ln f_i$$

- The log of the product of likelihoods becomes the sum of the logs:

$$\ell\ell(\mathbf{w}) = \ln \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

$$= \sum_{i=1}^{N} \ln P(y_i \mid x_i, w)$$

Rewritting log-likelihood

- For simpler math, we'll rewrite likelihood with indicators:

$$\ell\ell(\mathbf{w}) = \sum_{i=1}^{N} \ln P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

$$= \sum_{i=1}^{N} \left[\mathbb{1}[y_i = +1] \ln P(y = +1 \mid \mathbf{x}_i, \mathbf{w}) + \mathbb{1}[y_i = -1] \ln P(y = -1 \mid \mathbf{x}_i, \mathbf{w}) \right]$$

Indicator function

if $y_i = +1$ ✓ 0

if $y_i = -1$ 0 ✓

Logistic regression

Logistic regression model: P(y=-1|x,w)

- Probability model predicts y=+1:

$$P(y=+1|\mathbf{x},\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^\top h(\mathbf{x})}}$$

- Probability model predicts y=-1:

$$P(y=-1|x,w) = 1 - P(y=+1|x,w) = 1 - \frac{1}{1+e^{-w^T h(x)}}$$

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

Logistic regression

Plugging in logistic function for 1 data point

$$P(y=+1 \mid \mathbf{x},\mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^\top h(\mathbf{x})}} \qquad P(y=-1 \mid \mathbf{x},\mathbf{w}) = \frac{e^{-\mathbf{w}^\top h(\mathbf{x})}}{1+e^{-\mathbf{w}^\top h(\mathbf{x})}}$$

$$\ell\ell(\mathbf{w}) = \mathbb{1}[y_i=+1]\ln P(y=+1 \mid \mathbf{x}_i,\mathbf{w}) + \mathbb{1}[y_i=-1]\ln P(y=-1 \mid \mathbf{x}_i,\mathbf{w})$$

$$= \mathbb{1}[y_i=+1]\ln\frac{1}{1+e^{-w^T h(x_i)}} + (1-\mathbb{1}[y_i=+1])\ln\frac{e^{-w^T h(x_i)}}{1+e^{-w^T h(x_i)}}$$

$$= -\mathbb{1}[y_i=+1]\ln(1+e^{-w^T h(x_i)}) + (1-\mathbb{1}[y_i=+1])[-w^T h(x_i) - \ln(1+e^{-w^T h(x_i)})]$$

$$= -(1-\mathbb{1}[y_i=+1])\,w^T h(x_i) - \ln(1+e^{-w^T h(x_i)})$$

Simpler form

$$\ln e^a = a$$

$$\mathbb{1}[y_i=-1] = 1-\mathbb{1}[y_i=+1]$$

$$\ln\frac{1}{1+e^{-w^T h(x_i)}} = -\ln(1+e^{-w^T h(x_i)})$$

$$\ln\frac{e^{-w^T h(x_i)}}{1+e^{-w^T h(x_i)}} = \ln e^{-w^T h(x_i)} - \ln(1+e^{-w^T h(x_i)})$$

$$\ln e^{-w^T h(x_i)} = -w^T h(x_i)$$

Logistic regression

Gradient for 1 data point

$$\ell\ell(\mathbf{w}) = -(1 - \mathbb{1}[y_i = +1])\mathbf{w}^\top h(\mathbf{x}_i) - \ln\left(1 + e^{-\mathbf{w}^\top h(\mathbf{x}_i)}\right)$$

$$\frac{\partial \ell\ell}{\partial w_j} = -(1 - \mathbb{1}[y_i = +1]) \frac{\partial}{\partial w_j} w^T h(x_i) - \frac{\partial}{\partial w_j} \ln\left(1 + e^{-w^T h(x_i)}\right)$$

$$= -(1 - \mathbb{1}[y_i = +1]) h_j(x_i) + h_j(x_i) P(y = -1 \mid x_i, w)$$

$$= h_j(x_i)\left[\mathbb{1}[y_i = +1] - P(y = +1 \mid x_i, w)\right]$$

$$\frac{\partial}{\partial w_j} w^T h(x_i) = h_j(x_i)$$

$$\frac{\partial}{\partial w_j} \ln\left(1 + e^{-w^T h(x_i)}\right)$$

$$= -h_j(x_i) \underbrace{\frac{e^{-w^T h(x_i)}}{1 + e^{-w^T h(x_i)}}}_{P(y = -1 \mid x_i, w)}$$

Logistic regression

Finally, gradient for all data points

- Gradient for one data point:

$$h_j(\mathbf{x}_i)\Big(\mathbb{1}[y_i = +1] - P(y = +1 \mid \mathbf{x}_i, \mathbf{w})\Big)$$

- Adding over data points:

$$\frac{\partial \ell\ell}{\partial w_j} = \sum_{i=1}^{N} h_j(x_i)\left(\mathbb{1}[y_i = +1] - P(y = +1 \mid x_i, w)\right)$$

Details

- ADA boosting

AdaBoost: learning ensemble

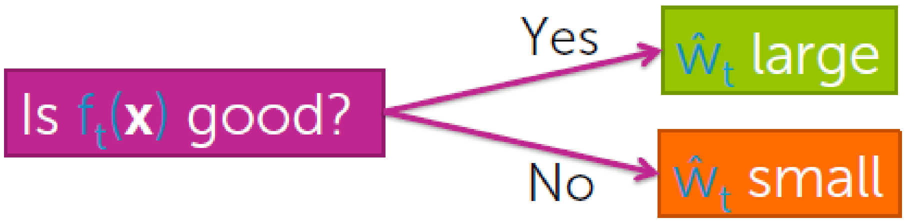
[Freund & Schapire 1999]

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$ ← Problem 1: How much do I trust f_t?
  - Recompute weights α_i ← Problem 2: weigh mistakes more?
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathrm{w}}_t f_t(\mathbf{x})\right)$$

(coefficients)

AdaBoost: Computing coefficients w_t



- $f_t(\mathbf{x})$ is good ➔ f_t has low training error
- Measuring error in weighted data?
  – Just weighted # of misclassified points

Weighted classification error

- Total weight of mistakes:

$$= \sum_{i=1}^{N} \alpha_i \underbrace{\mathbb{1}(\hat{y}_i \neq y_i)}_{\text{Mistake?}}$$

- Total weight of all points:

$$= \sum_{i=1}^{N} \alpha_i$$

- Weighted error measures fraction of weight of mistakes:

$$\text{weighted\_error} = \frac{\text{Total weight of mistakes}}{\text{Total weight of all data points}}.$$

  - Best possible value is 0.0 → worst 1.0 → Random classifier = 0.5

AdaBoost formula

AdaBoost: Formula for computing coefficient $\hat{w}_t$ of classifier $f_t(\mathbf{x})$

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

Is $f_t(\mathbf{x})$ good?

Yes / No

weighted_error(f_t) on training data	$\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}$	$\hat{w}_t$
0.01	$\frac{1-0.01}{0.01} = 99$	$\frac{1}{2} \ln 99 = 2.3$
0.5	$\frac{1-0.5}{0.5} = 1$	0
0.99	$\frac{1-0.99}{0.99} = 0.01$	-2.3

Terrible classifier, but $1-f_t$ is awesome!!

AdaBoost: learning ensemble

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i

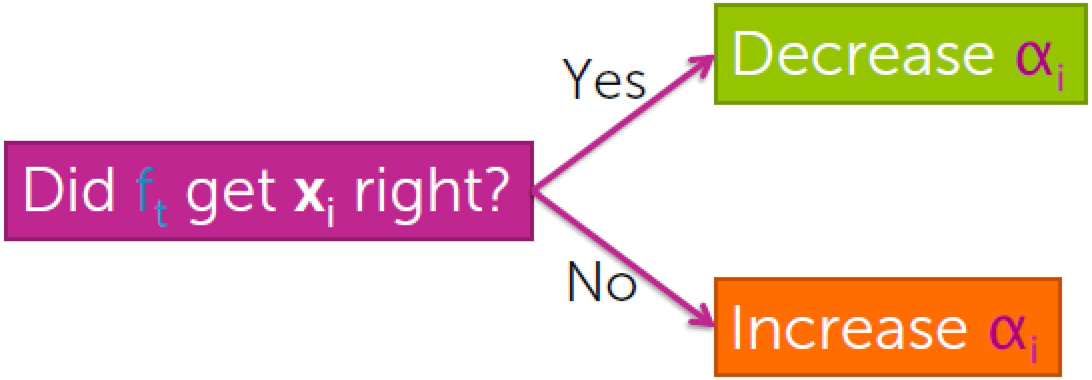
$$\hat{\mathrm{w}}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathrm{w}}_t f_t(\mathbf{x}) \right)$$

AdaBoost: updating weights α_i

Updating weights α_i based on where classifier $f_t(\mathbf{x})$ makes mistakes



AdaBoost: updating weights α_i

AdaBoost: Formula for updating weights α_i

$$
\alpha_i \leftarrow \begin{cases} \alpha_i \, e^{-\hat{W}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \quad \leftarrow \text{correct} \\ \alpha_i \, e^{\hat{W}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \quad \leftarrow \text{mistake} \end{cases}
$$

Did f_t get $\mathbf{x}_i$ right? Yes / No

$f_t(\mathbf{x}_i)=y_i$?	$\hat{W}_t$	Multiply α_i by	Implication
Correct	2.3	$e^{-2.3} = 0.1$	Decrease importance of x_i, y_i
Correct	0	$e^{-0} = 1$	Keep importance the same
Mistake	2.3	$e^{2.3} = 9.98$	Increasing importance of x_i, y_i
Mistake	0	$e^{0} = 1$	Keep importance the same

AdaBoost: learning ensemble

- Start same weight for all points: $\alpha_i = 1/N$

- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i \, e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i \, e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x}) \right)$$

AdaBoost: normlizing weights α_i

$\mathbf{x}_i$

If $\mathbf{x}_i$ often mistake, weight $\boldsymbol{\alpha}_i$ gets very **large**

If $\mathbf{x}_i$ often correct, weight $\boldsymbol{\alpha}_i$ gets very **small**

Can cause numerical instability after many iterations

Normalize weights to add up to 1 after every iteration

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

AdaBoost: learning ensemble

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})\right)$$

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

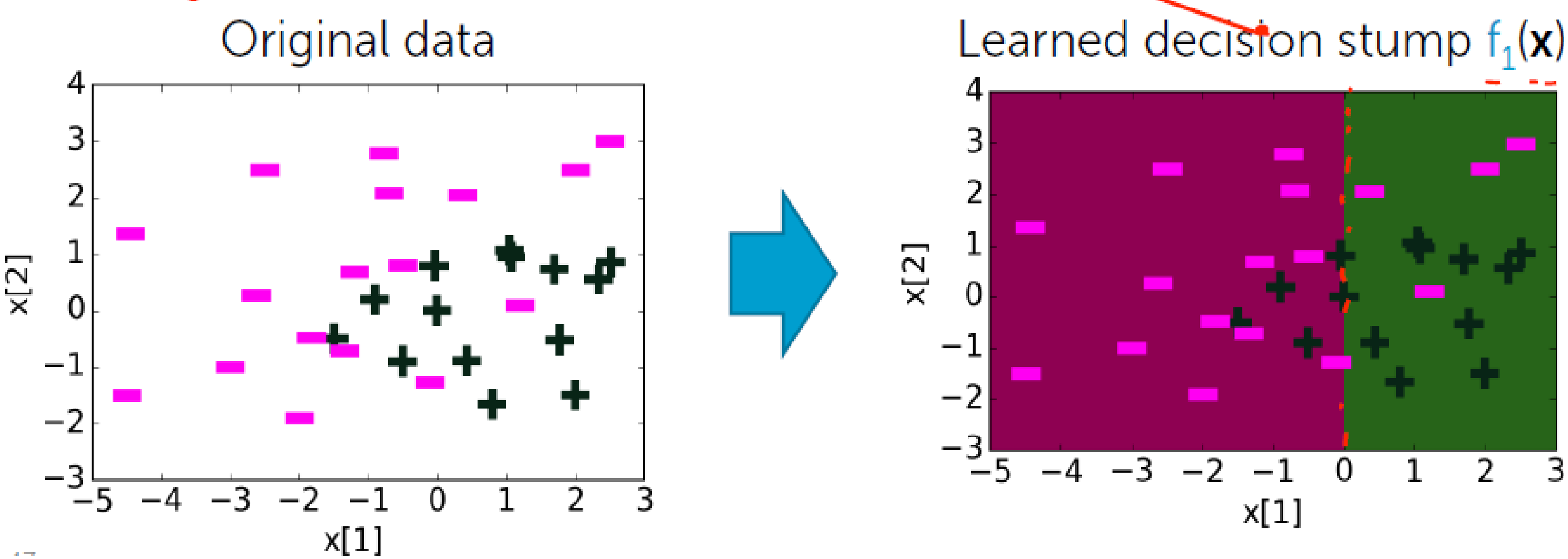
$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

AdaBoost: example

t=1: Just learn a classifier on original data

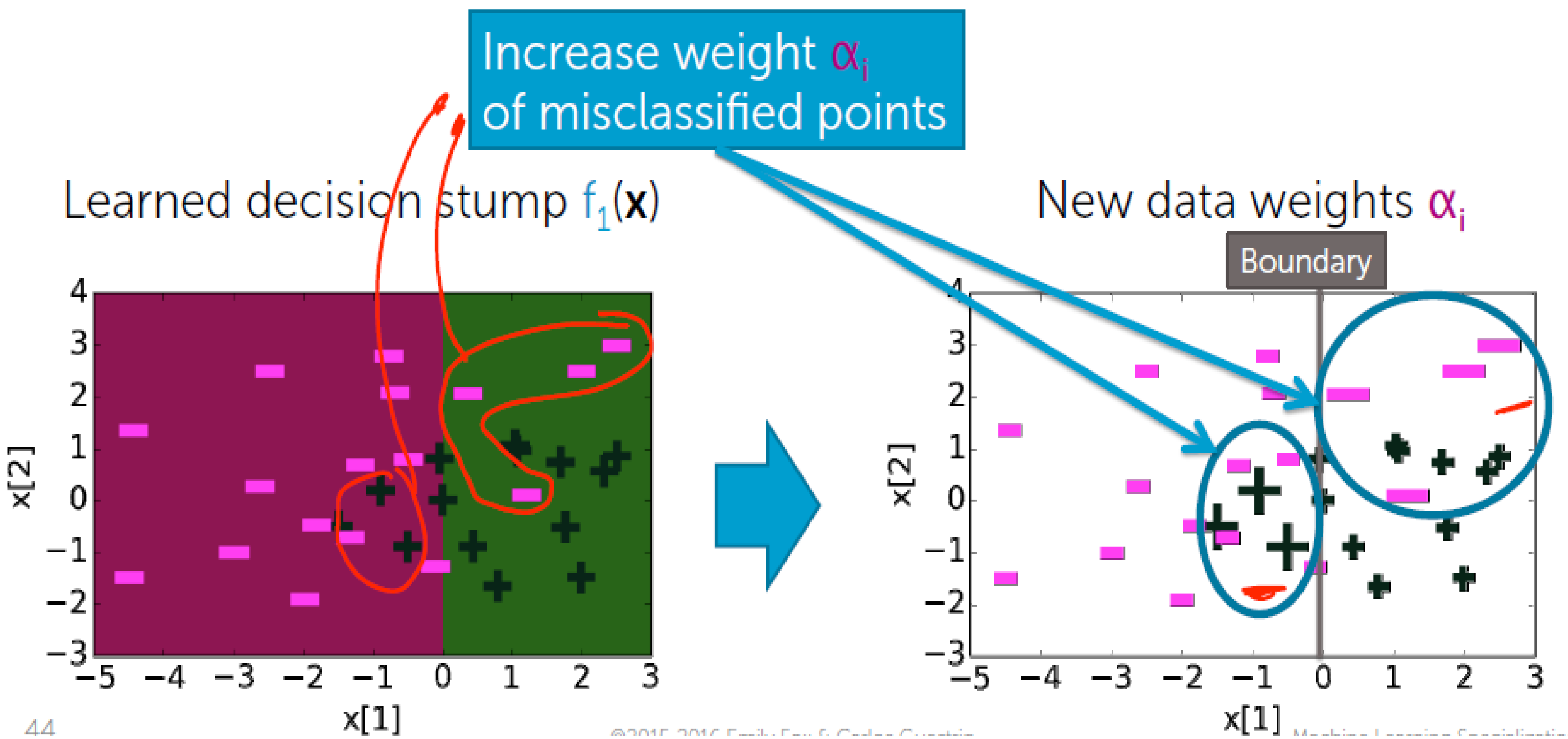
all points have same weight

standard learning



AdaBoost: example

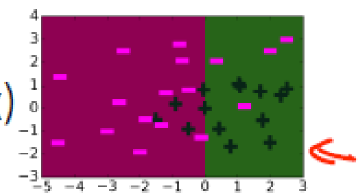
Updating weights α_i



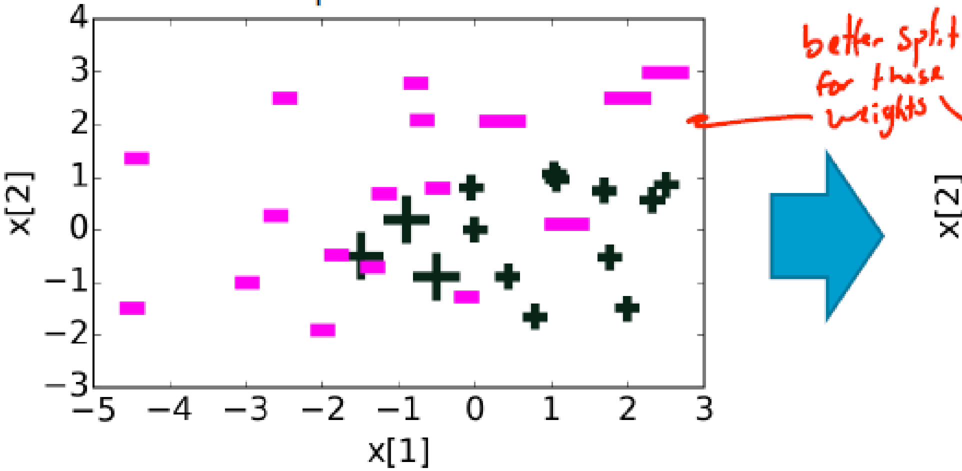
AdaBoost: example

t=2: Learn classifier on weighted data

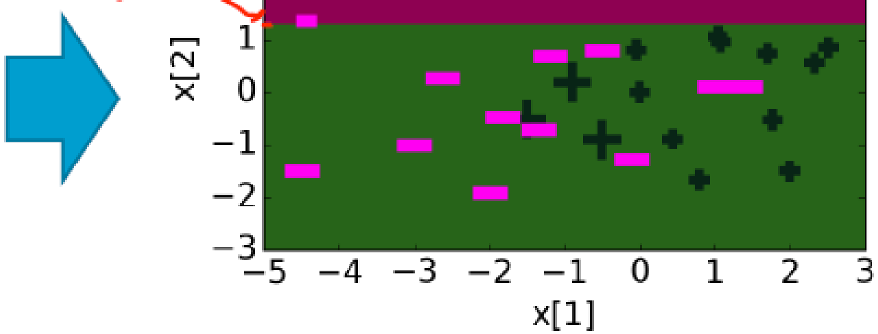
$f_1(\mathbf{x})$



Weighted data: using α_i chosen in previous iteration

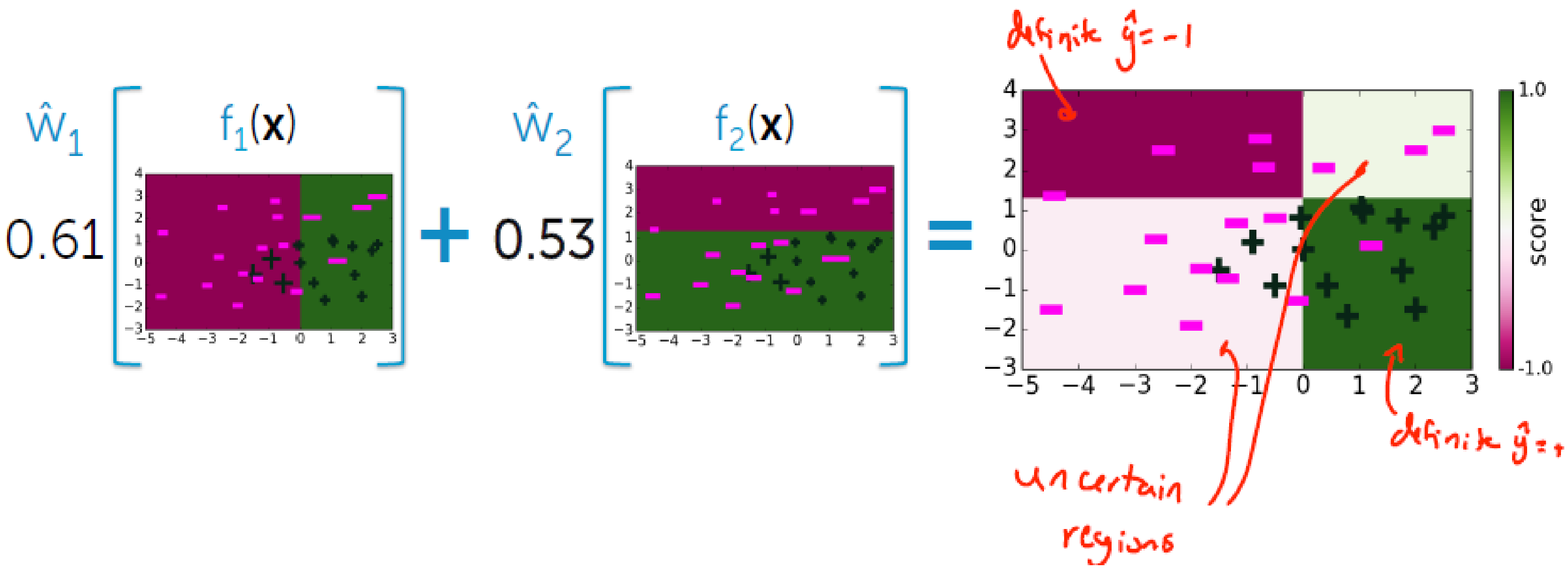


Learned decision stump $f_2(\mathbf{x})$ on weighted data



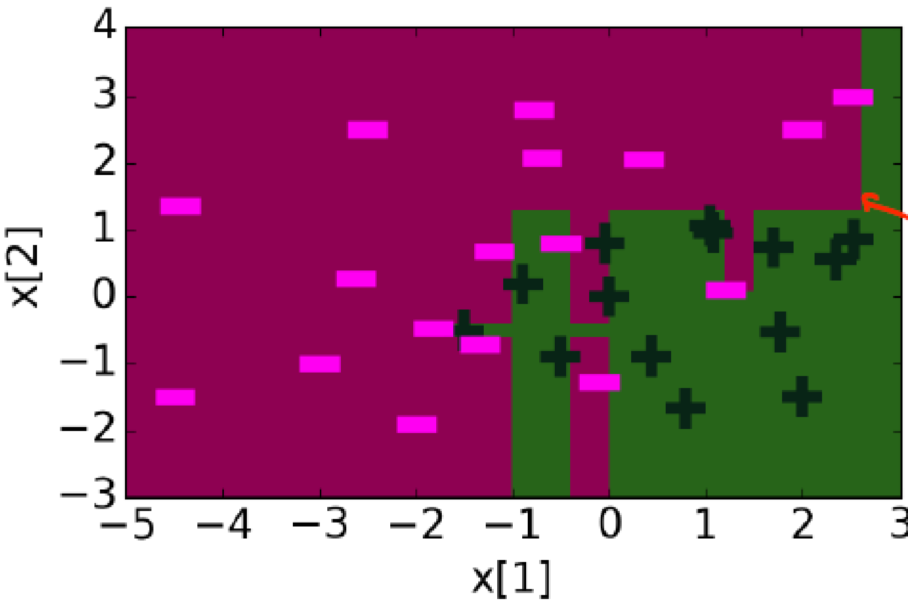
AdaBoost: example

Ensemble becomes weighted sum of learned classifiers



AdaBoost: example

Decision boundary of ensemble classifier after 30 iterations



training_error = 0

Decision boundary is crazy!!

Probably overfitting

AdaBoost: learning ensemple

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathrm{w}}_t f_t(\mathbf{x})\right)$$

$$\hat{\mathrm{w}}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

Boosted decision stumps

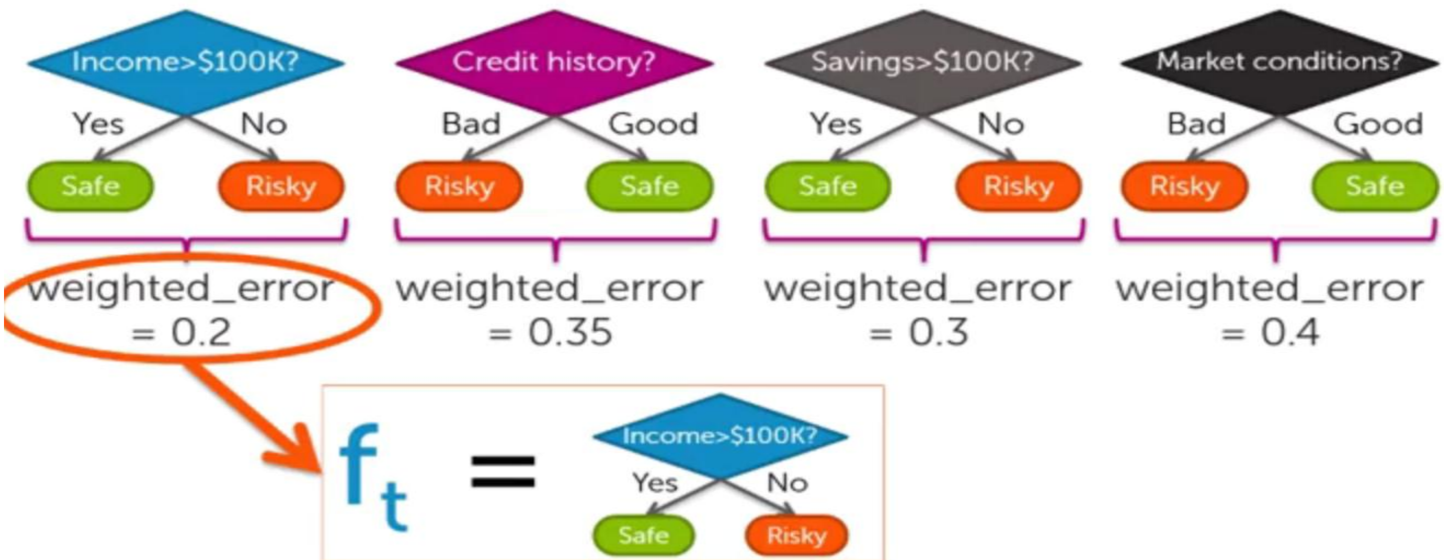
- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick decision stump with lowest weighted training error according to α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})\right)$$

Boosted decision stumps

Finding best next decision stump $f_t(x)$

Consider splitting on each feature:



$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right) = 0.69$$

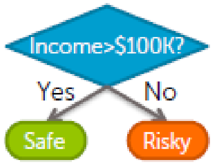
Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick decision stump with lowest weighted training error according to α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})\right)$$

Boosted decision stumps

Updating weights α_i



$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} = \alpha_i\, e^{-0.69} = \alpha_i/2 & \text{,if } f_t(x_i) = y_i \\ \alpha_i e^{\hat{w}_t} = \alpha_i\, e^{0.69} = 2\,\alpha_i & \text{,if } f_t(x_i) \neq y_i \end{cases}$$

Credit	Income	y	ŷ	Previous weight α	New weight α
A	$130K	Safe	Safe	0.5	0.5/2 = 0.25
B	$80K	Risky	Risky	1.5	0.75
C	$110K	Risky	Safe	1.5	2 * 1.5 = 3
A	$110K	Safe	Safe	2	1
A	$90K	Safe	Risky	1	2
B	$120K	Safe	Safe	2.5	1.25
C	$30K	Risky	Risky	3	1.5
C	$60K	Risky	Risky	2	1
B	$95K	Safe	Risky	0.5	1
A	$60K	Safe	Risky	1	2
A	$98K	Safe	Risky	0.5	1