INTRODUCTION TO DATA SCIENCE

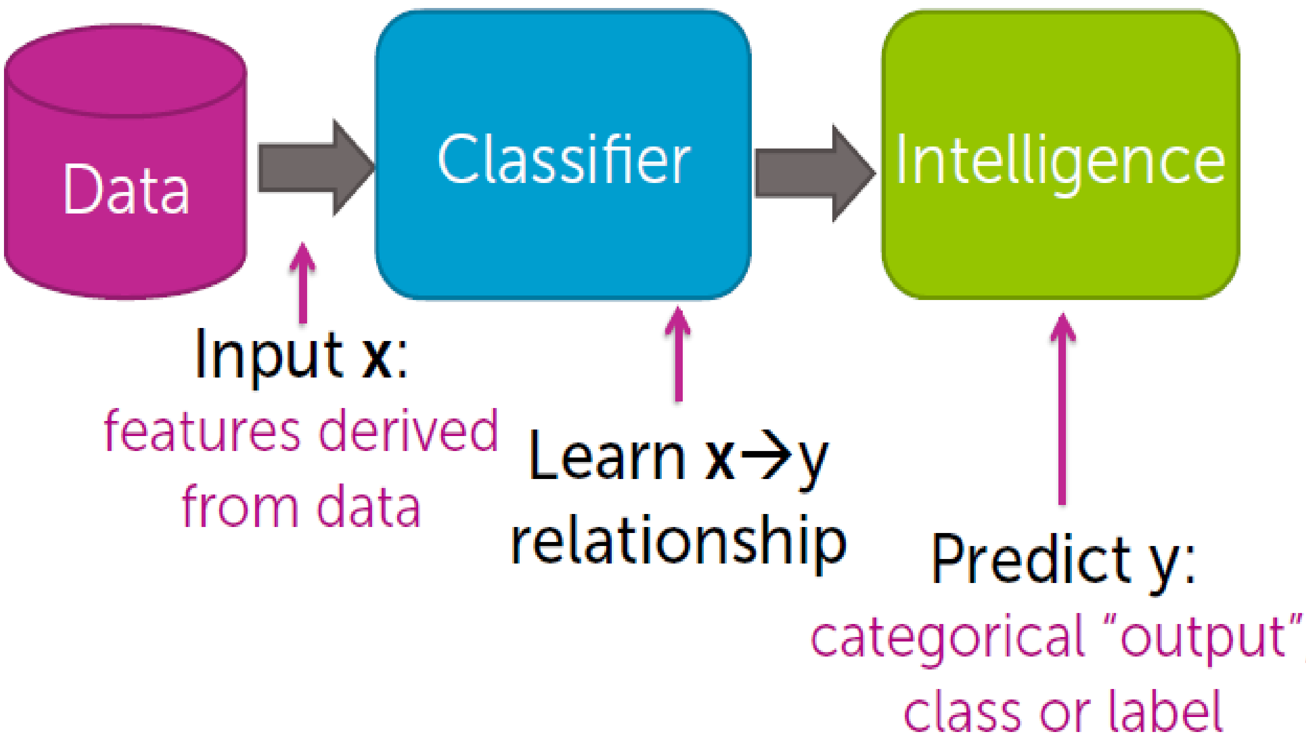
This lecture is
based on course by E. Fox and C. Guestrin, Univ of Washington

30/10, 6/11 2024

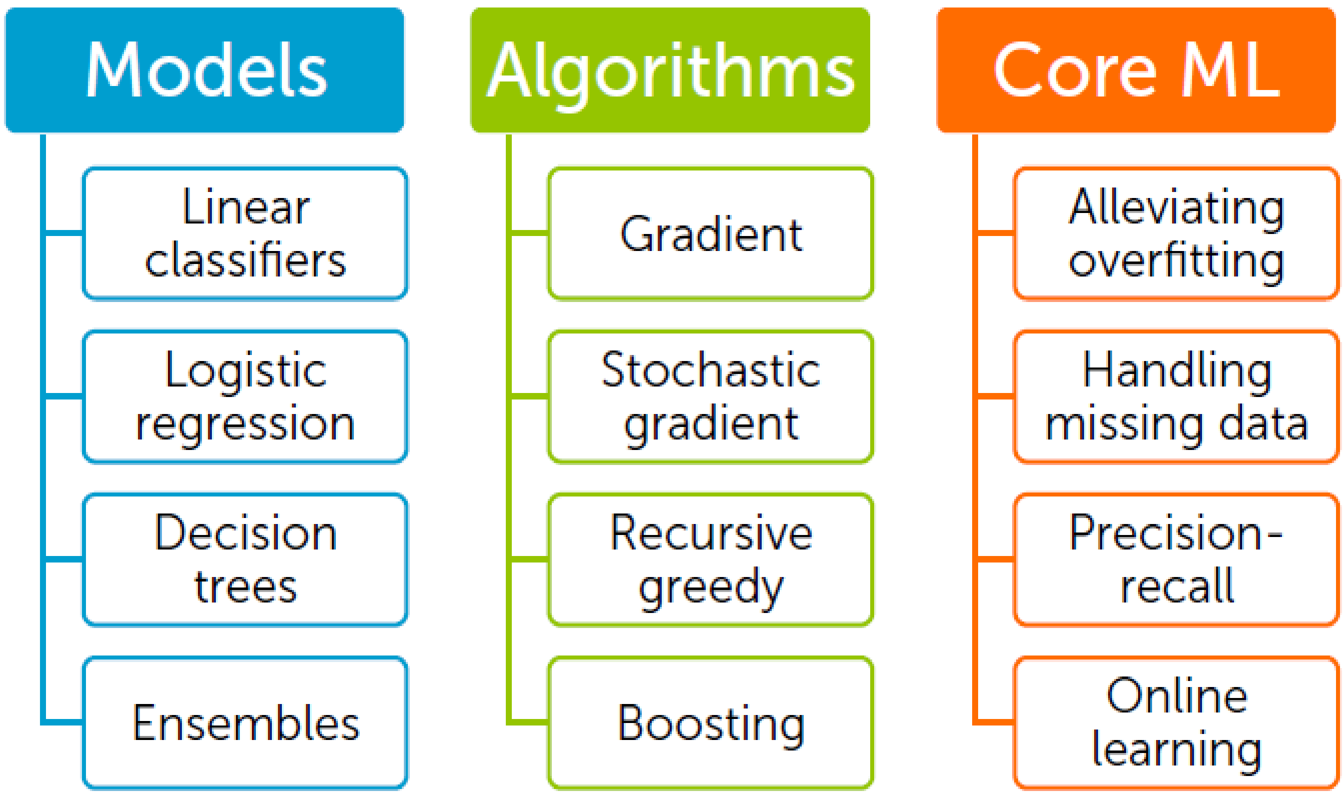
WFAiS UJ, Informatyka Stosowana
I stopień studiów

What is a classification?

From features to predictions



Overwiew of the content



Linear classifier

An inteligent restaurant review system

It's a big day & I want to book a table at a nice Japanese restaurant

Reviews

Positive reviews not positive about everything

Sample review:

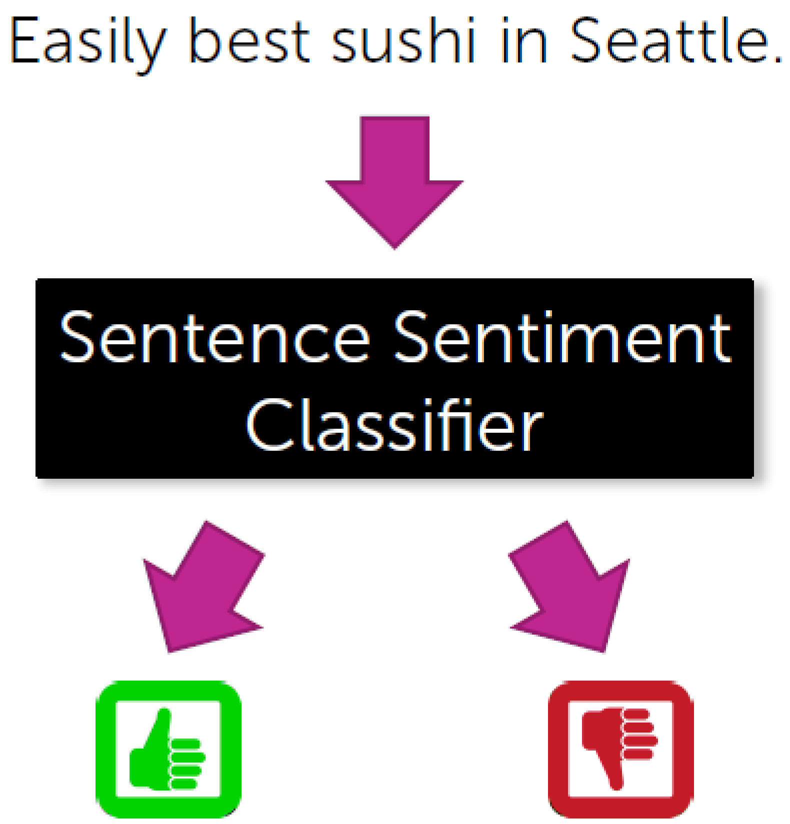
Watching the chefs create incredible edible art made the experience very unique.

My wife tried their ramen and it was pretty forgettable.

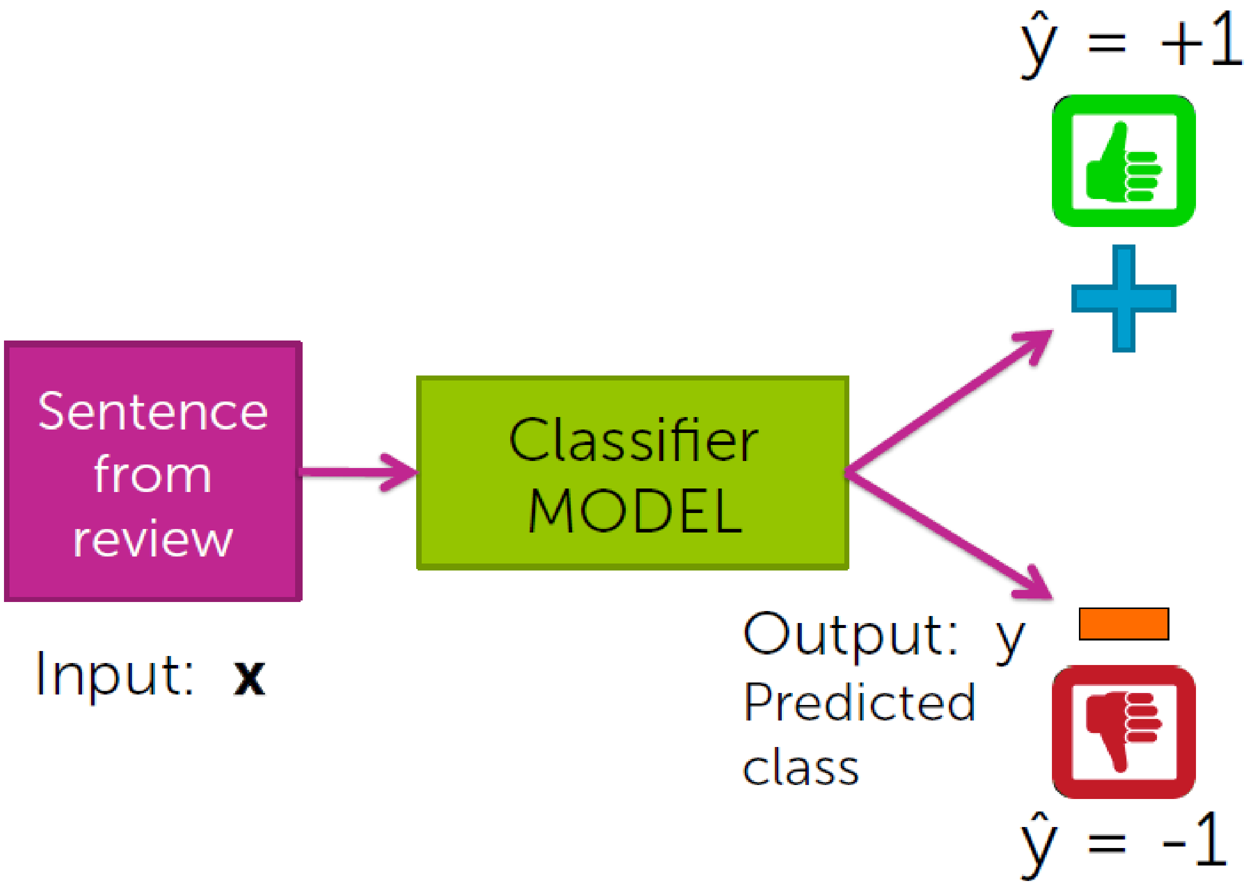
All the sushi was delicious! Easily best sushi in Seattle.

Experience

Classifying sentiment of review



Classifier



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

A (linear) classifier

Will use training data to learn a weight for each word

Word	Weight
good	1.0
great	1.5
awesome	2.7
bad	-1.0
terrible	-2.1
awful	-3.3
restaurant, the, we, where, ...	0.0
...	...

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.

Simple linear classifier

Word	Coefficient
…	…

Sentence from review

Input: **x**

Simple linear classifier

Score(**x**) = weighted count of words in sentence

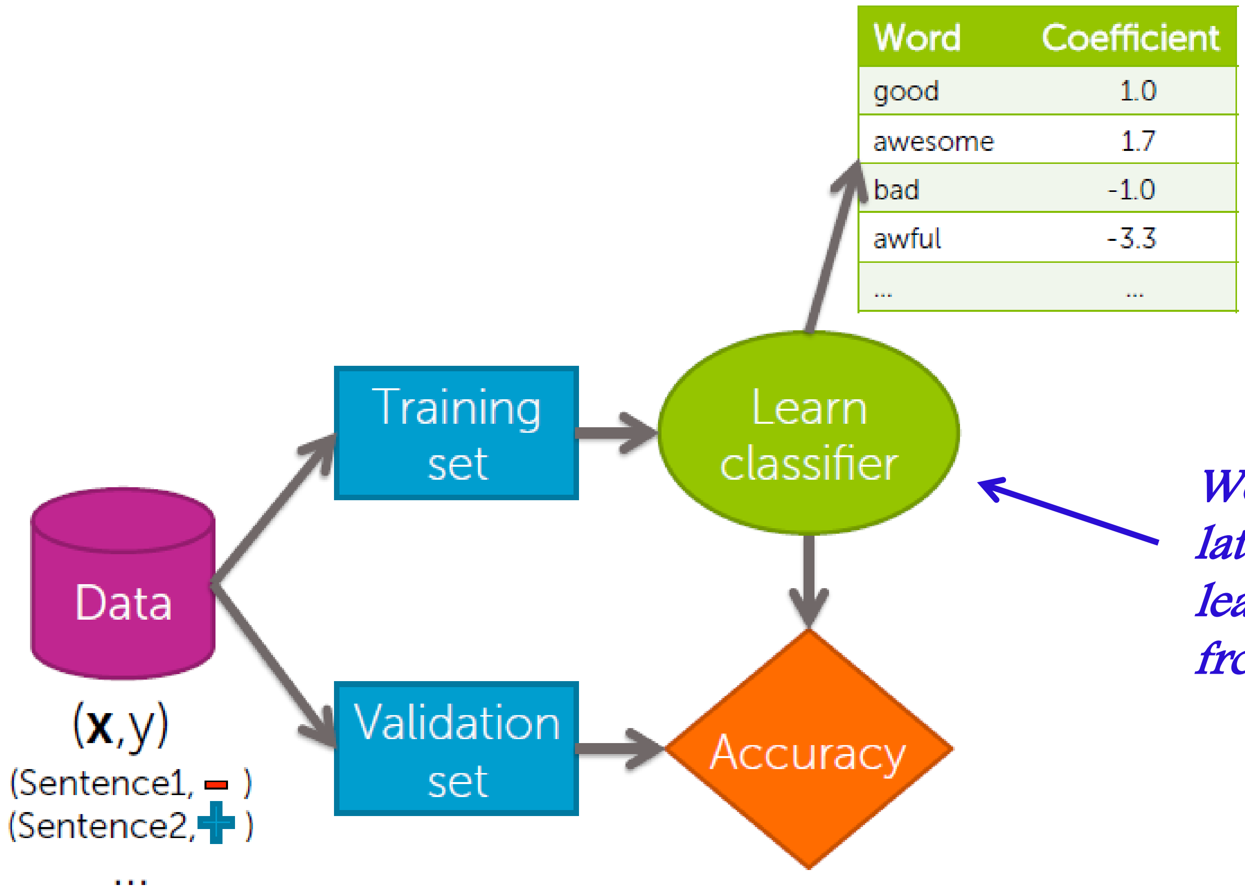
If *Score* (**x**) > *0*:

$\hat{y} = +1$

Else:

$\hat{y} = -1$

Training a classifier = Learning the coefficients

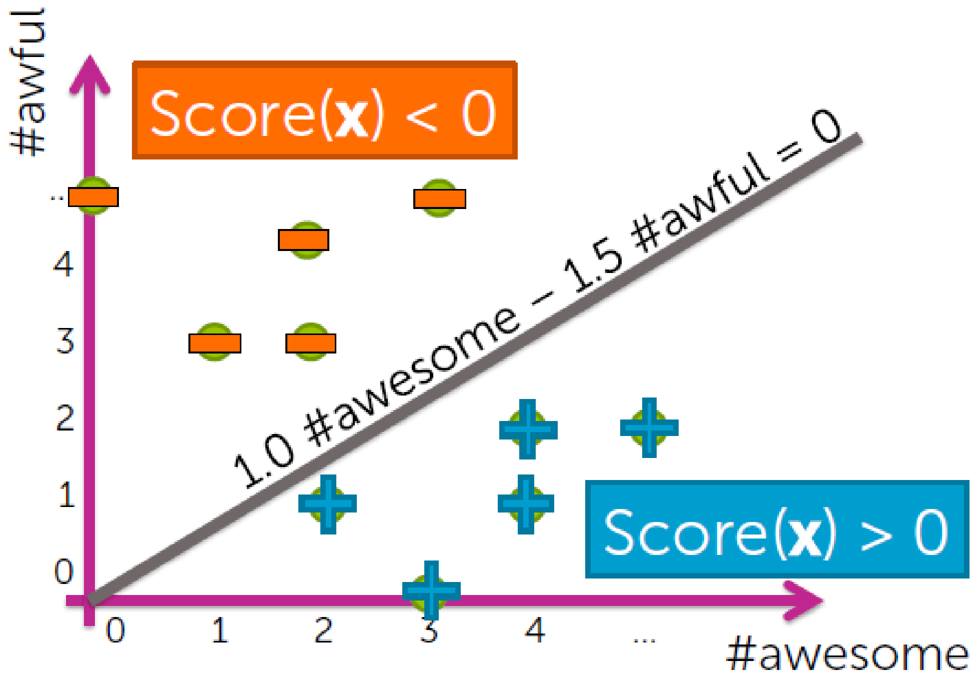


We will discuss latter how do we learn clasifier from data

Decision boundary example

Word	Coefficient
#awesome	1.0
#awful	-1.5

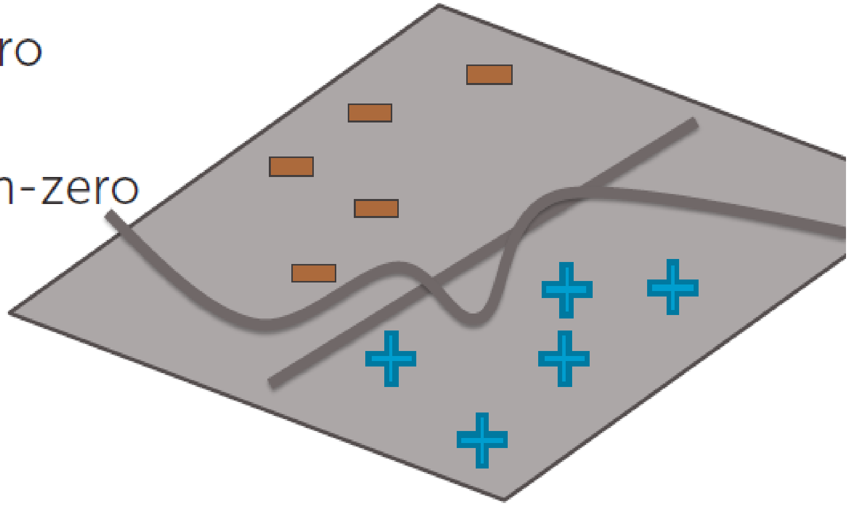
→ Score(x) = 1.0 #awesome – 1.5 #awful



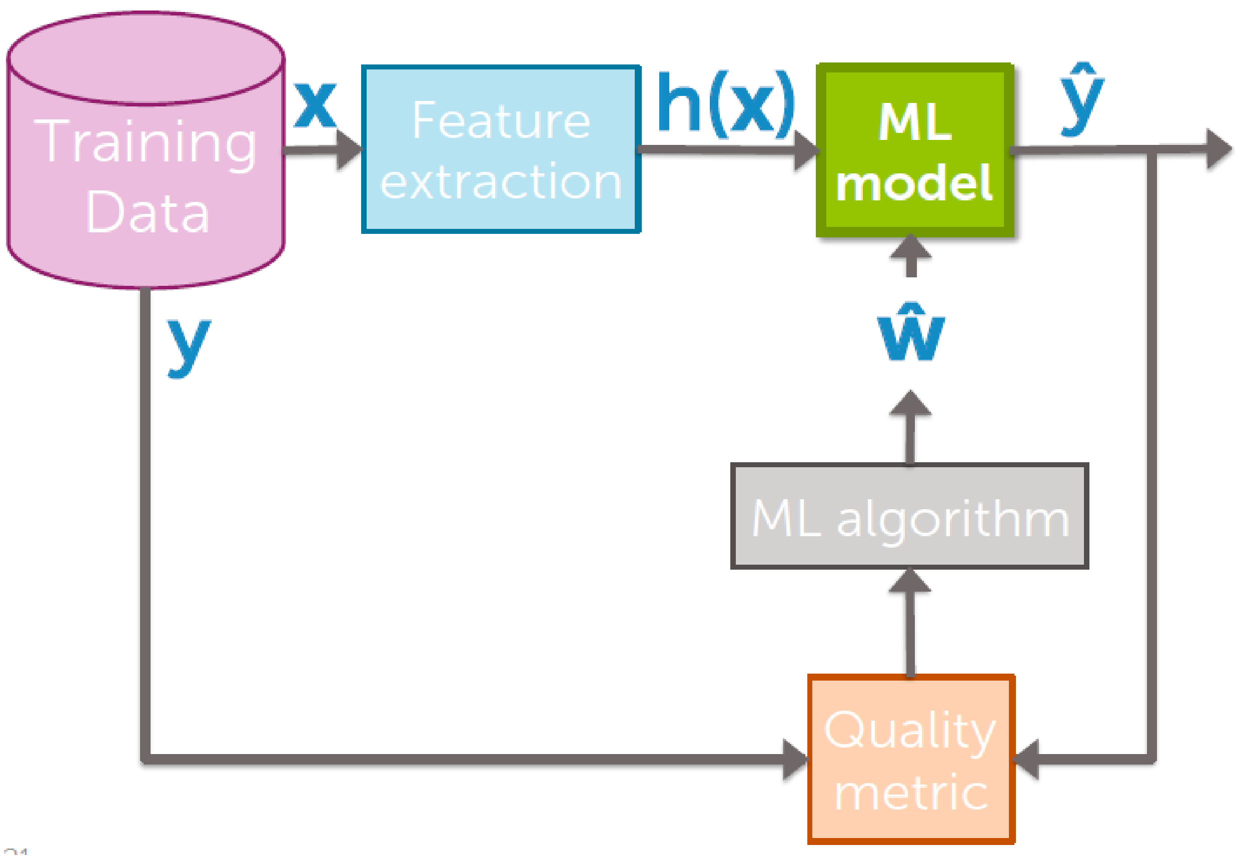
Decision boundary

Decision boundary separates positive & negative predictions

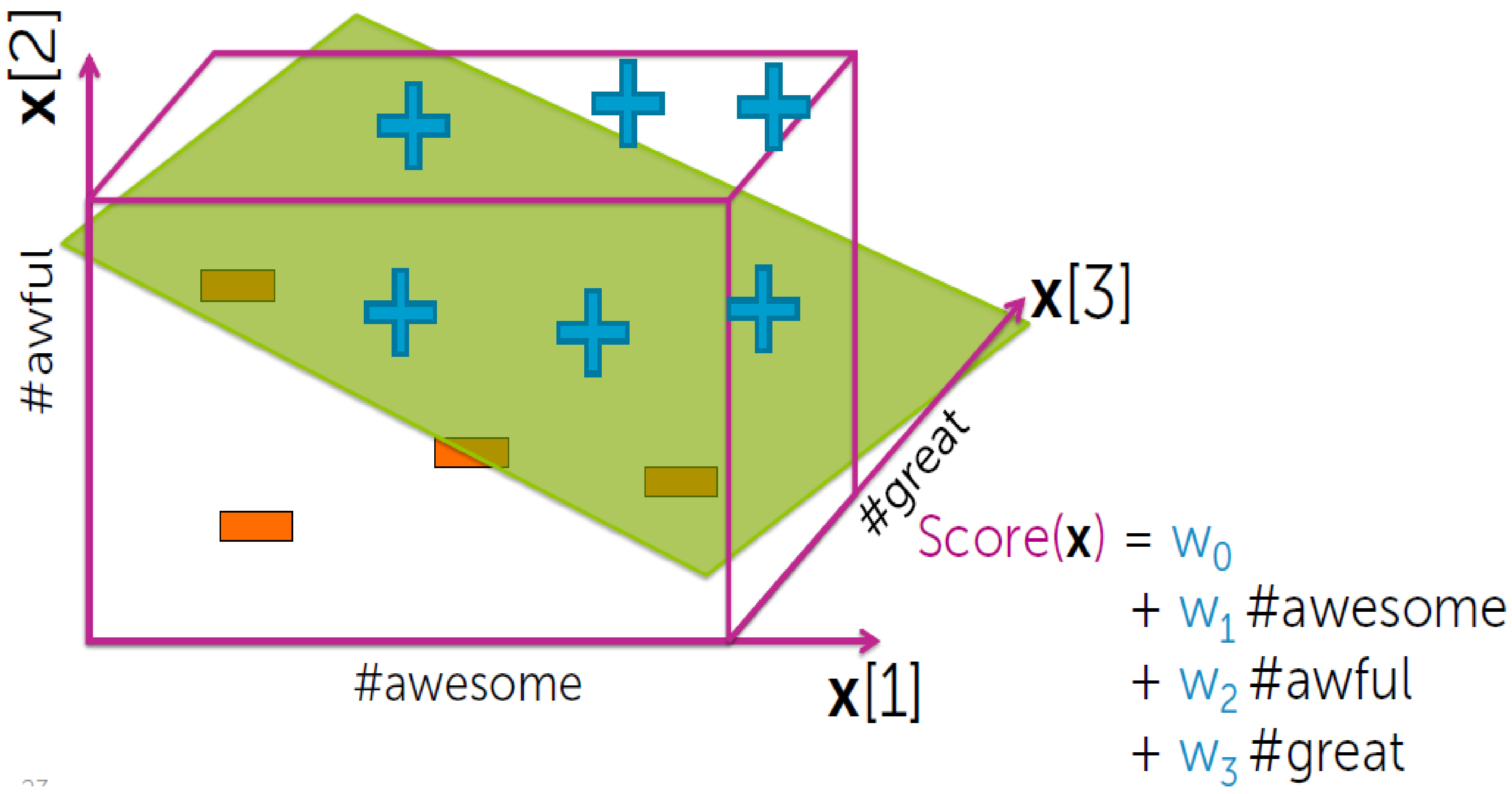
- For linear classifiers:
  - When 2 coefficients are non-zero
    ➔ line
  - When 3 coefficients are non-zero
    ➔ plane
  - When many coefficients are non-zero
    ➔ hyperplane
- For more general classifiers
  ➔ more complicated shapes



Flow chart: ML model



Coefficients of classifier



General notation

Output: y ← {-1,+1}

Inputs: $\mathbf{x}$ = ($\mathbf{x}$[1],$\mathbf{x}$[2],..., $\mathbf{x}$[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 = **x**[1] ... e.g., #awesome

feature 3 = **x**[2] ... e.g., #awful

...

feature d+1 = **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

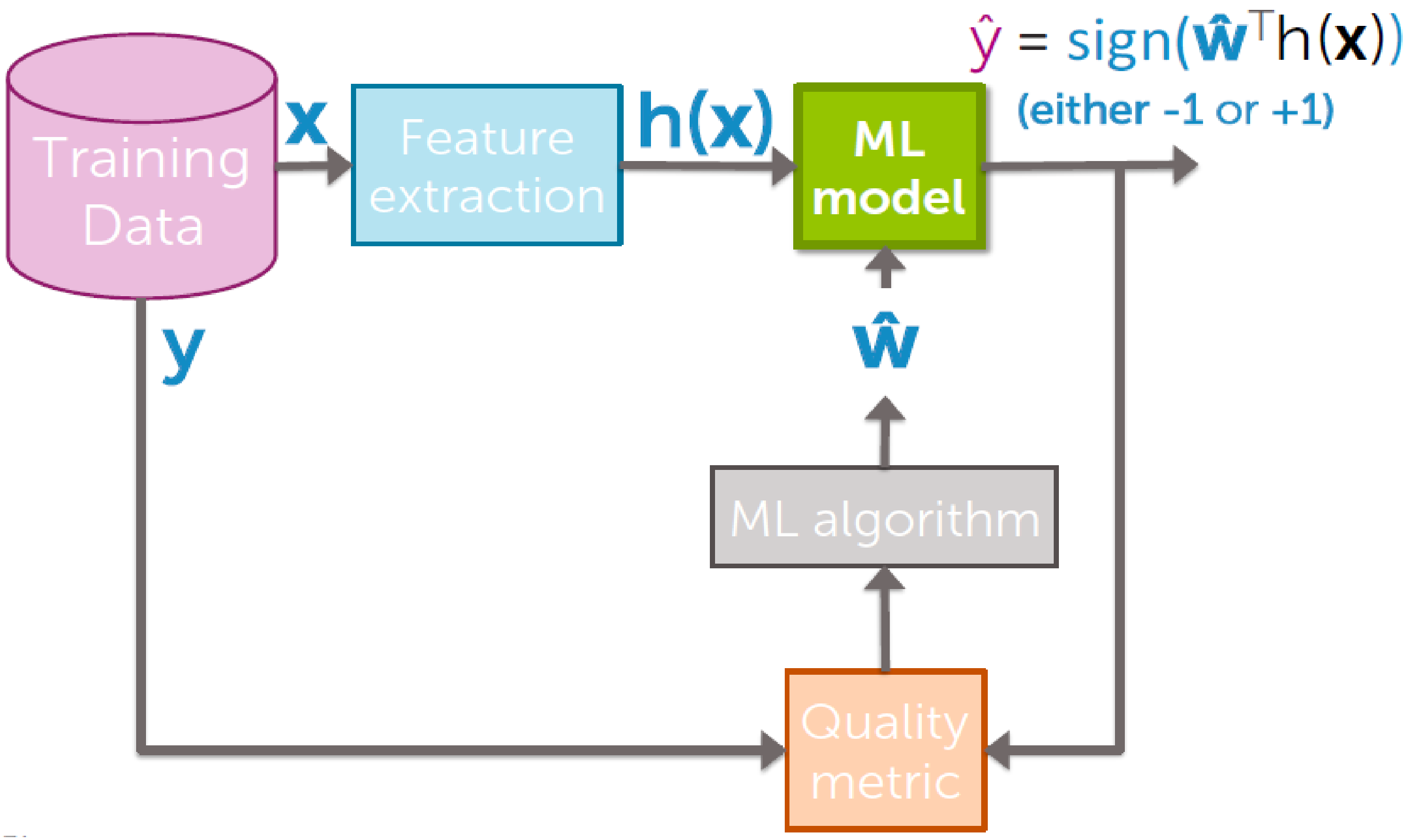
or, $\log(\mathbf{x}[7])\, \mathbf{x}[2]$ = log(#bad) x #awful

or, tf-idf("awful")

…

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

Flow chart: ML model

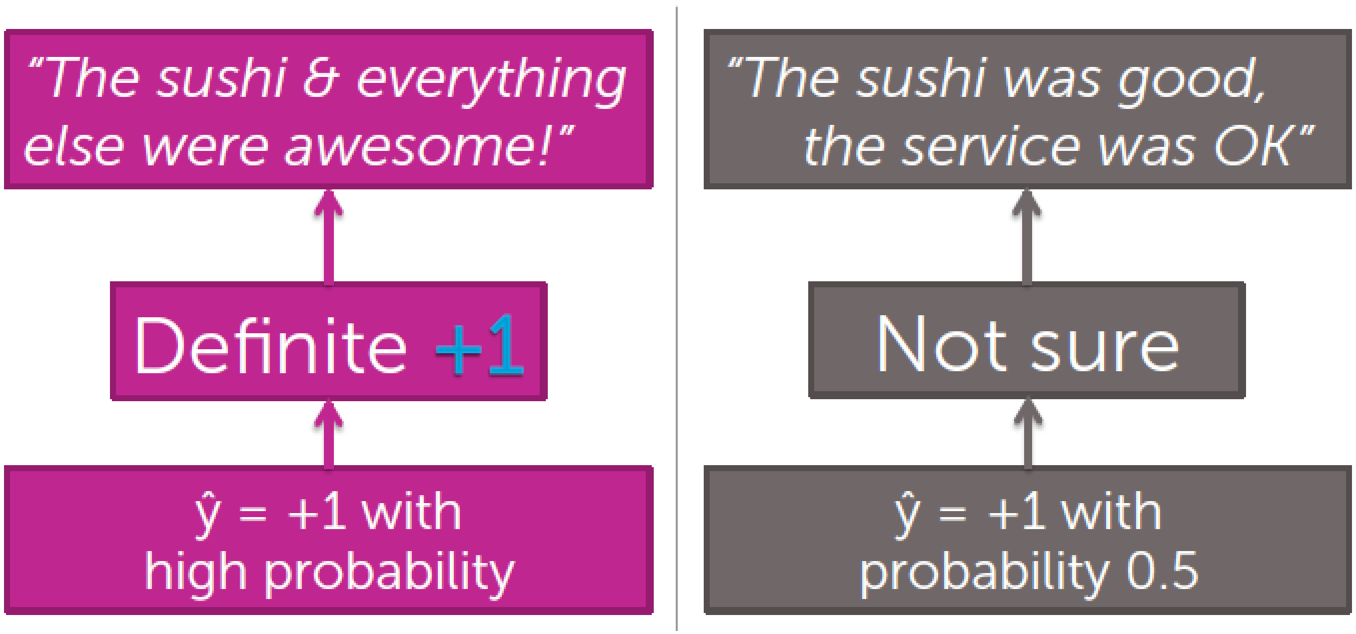


Linear classifier

- 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?



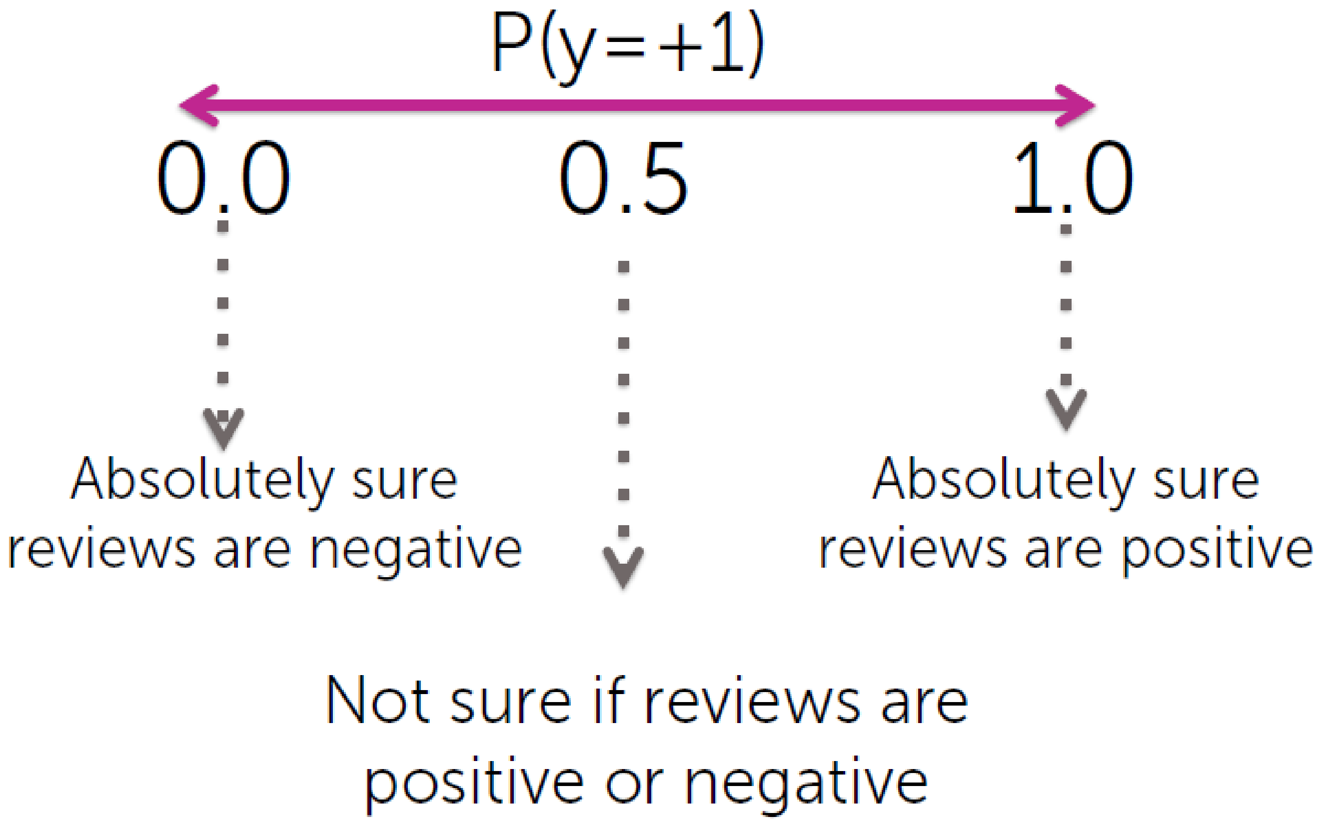
Basics of probabilities

Probability a review is positive is 0.7

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

I expect 70% of rows to have y = +1
(Exact number will vary for each specific dataset)

Interpreting probabilities as degrees of belief



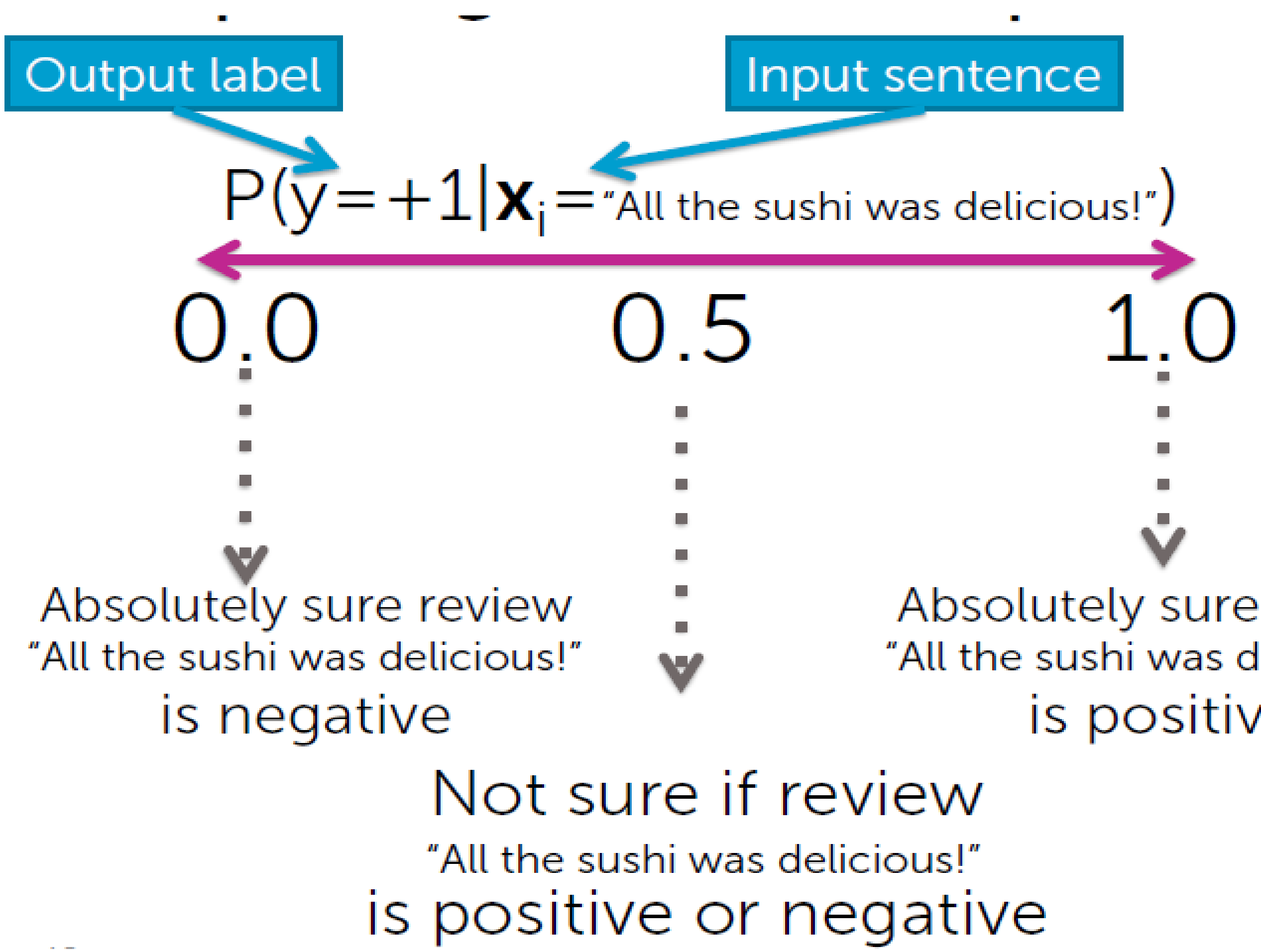
Conditional probability

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|**x**="The sushi & everything else were awesome!") = 0.99

"The sushi was good, the service was OK"

Not sure

P(y=+1|**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
...	...	...

Optimize **quality metric** on training data

Find best model $\hat{P}$ by finding best $\hat{\mathbf{w}}$

Useful for predicting $\hat{y}$

Predicting class probabilities

Sentence from review

Input: **x**

Predict most likely class

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

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

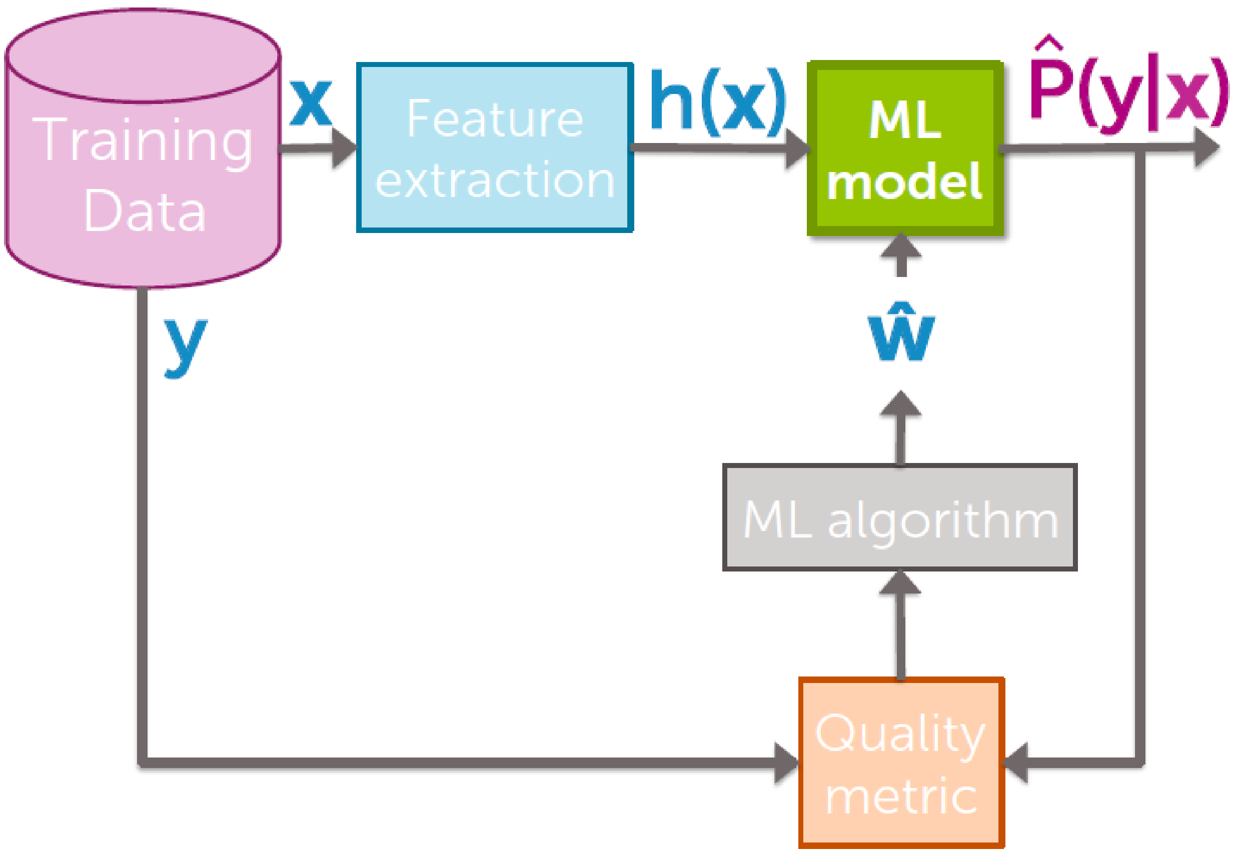
$\hat{y} = +1$

Else:

$\hat{y} = -1$

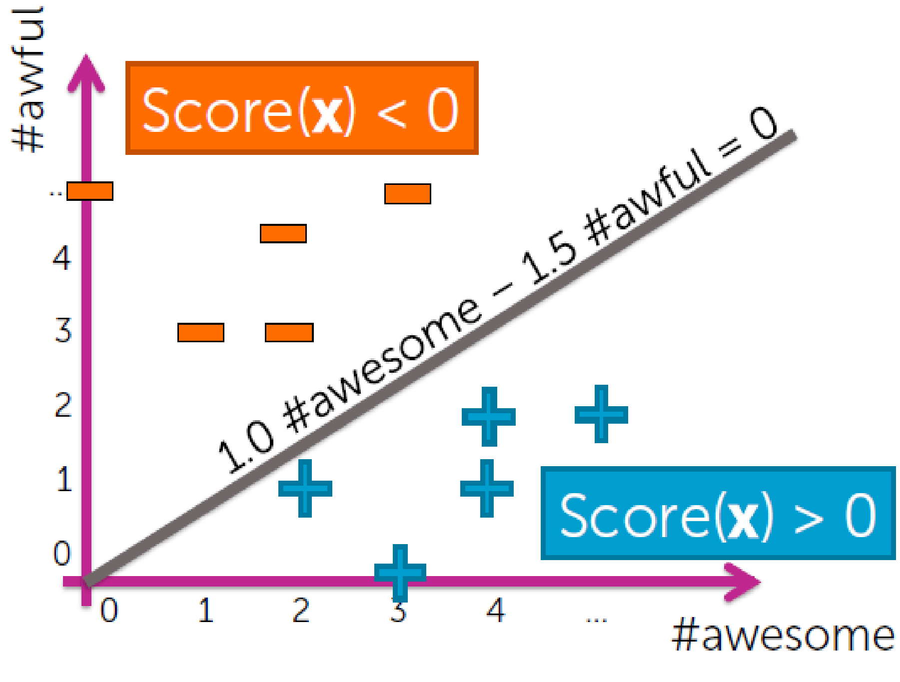
- Estimating $\hat{P}(y|x)$ improves **interpretability**:
  - Predict $\hat{y} = +1$ **and** tell me how sure you are

Flow chart: ML model



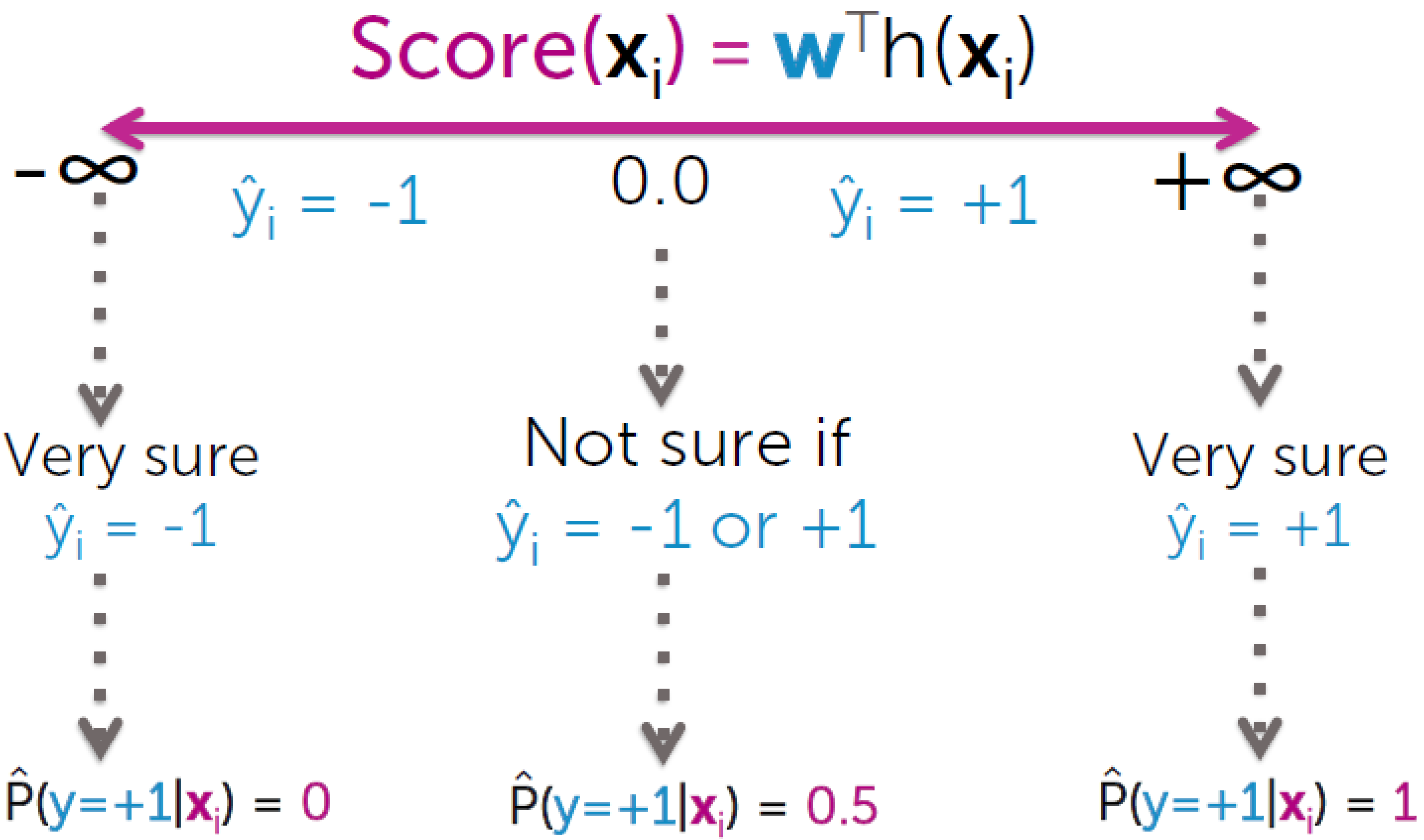
Thus far we focused on decision boundaries

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

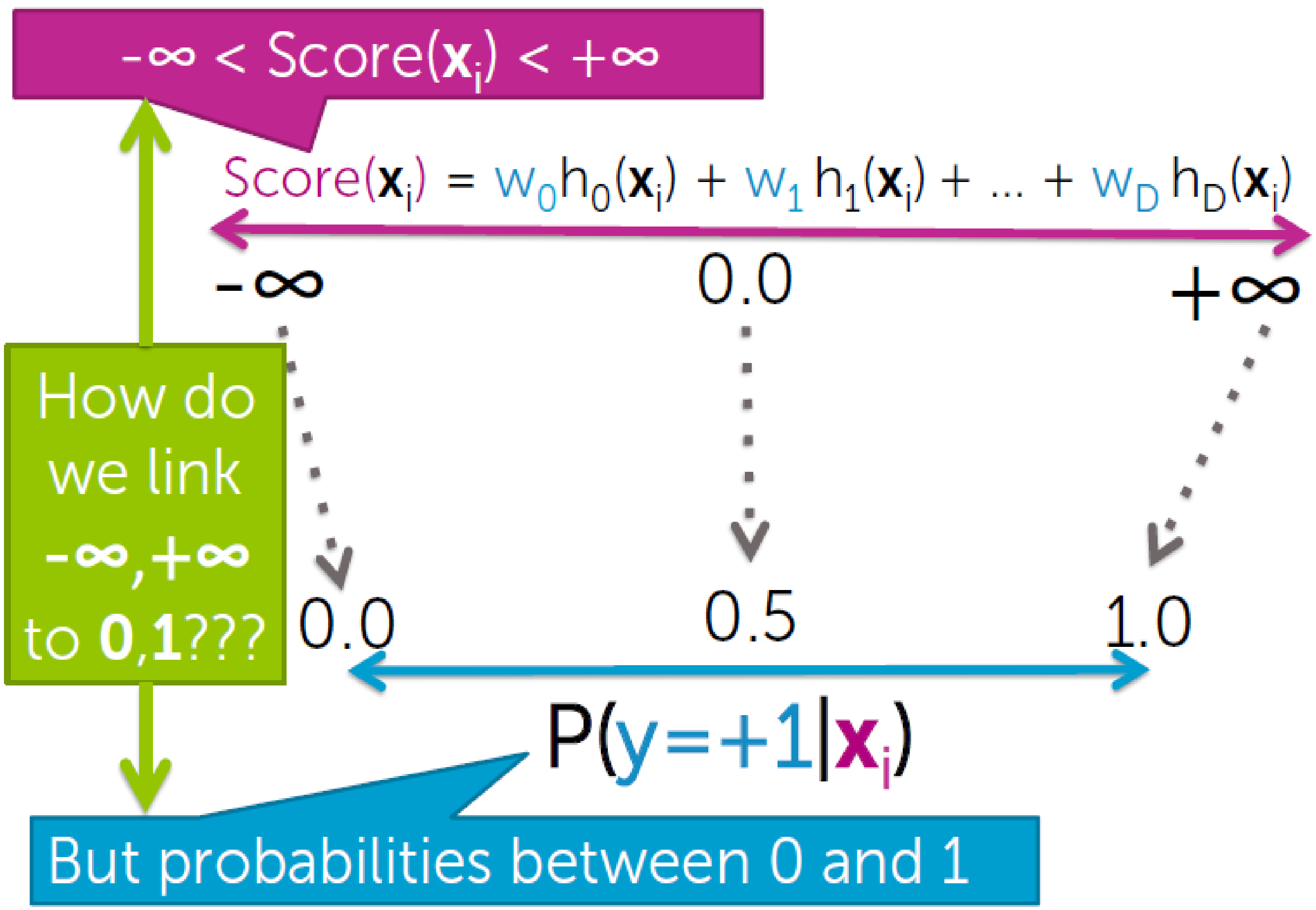


How to relate Score($\mathbf{x}_i$) to $\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}})$?

Interpreting Score(x_i)

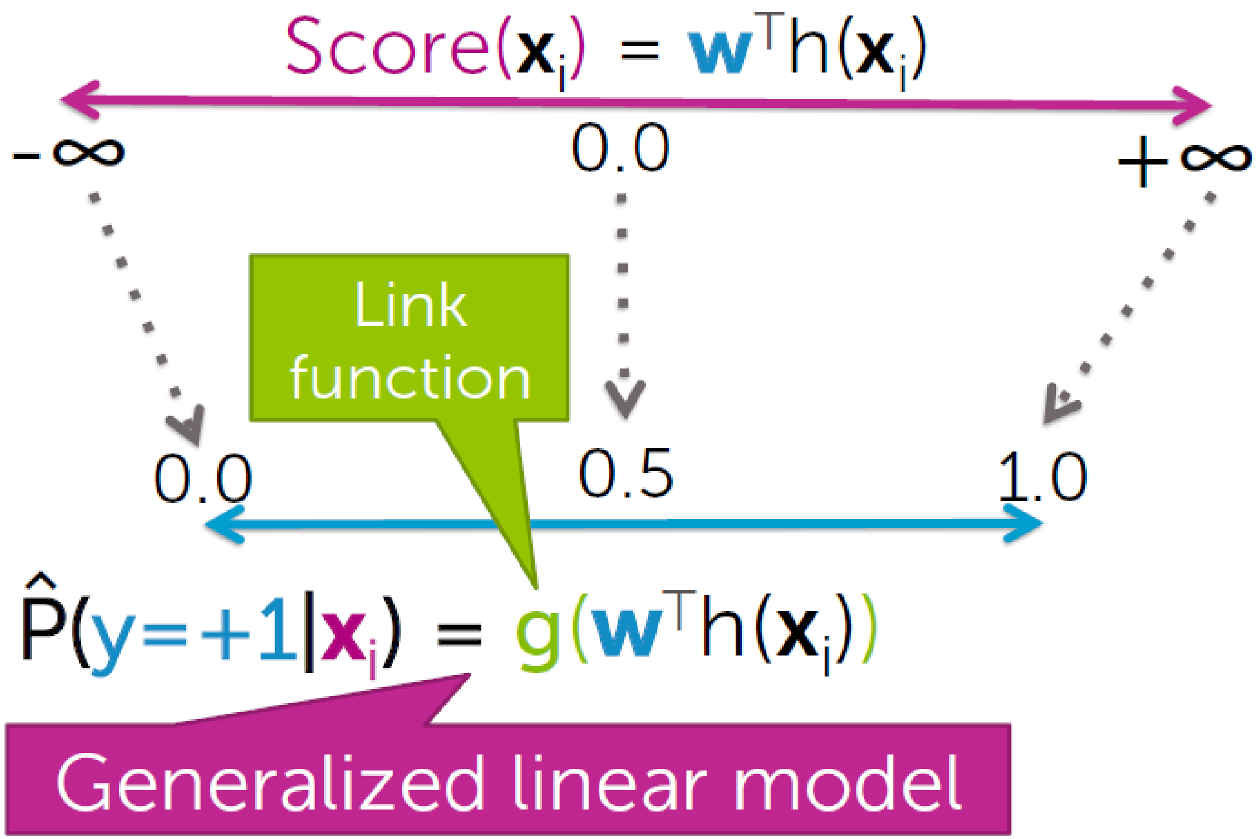


Why not just use regression to build classifier?



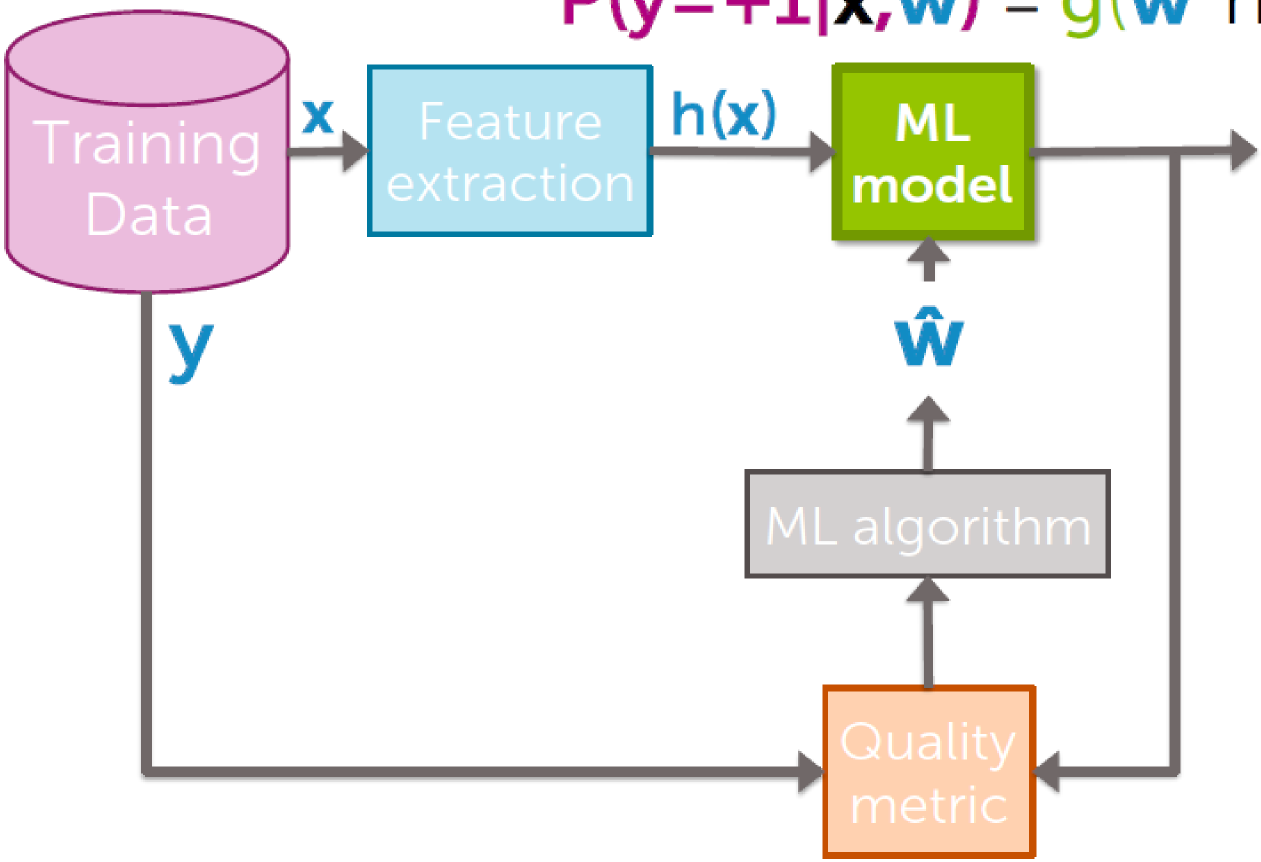
Link function

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



Flow chart: ML model

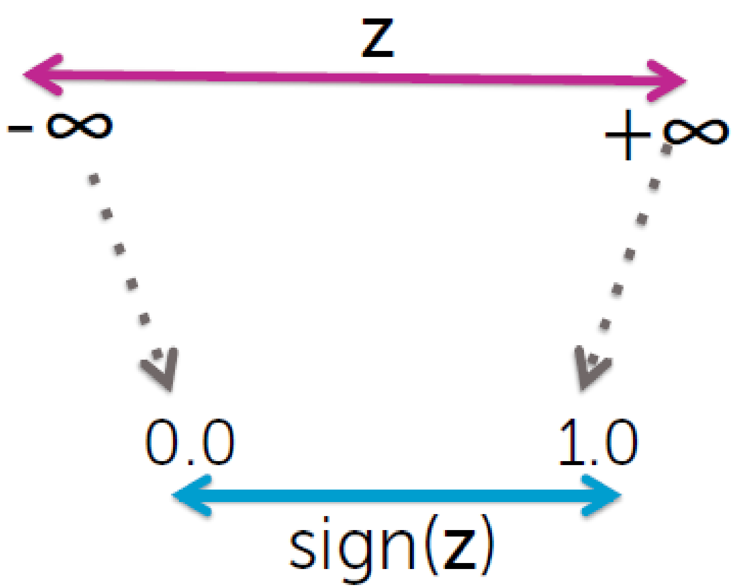
$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = g(\hat{\mathbf{w}}^T h(\mathbf{x}))$$



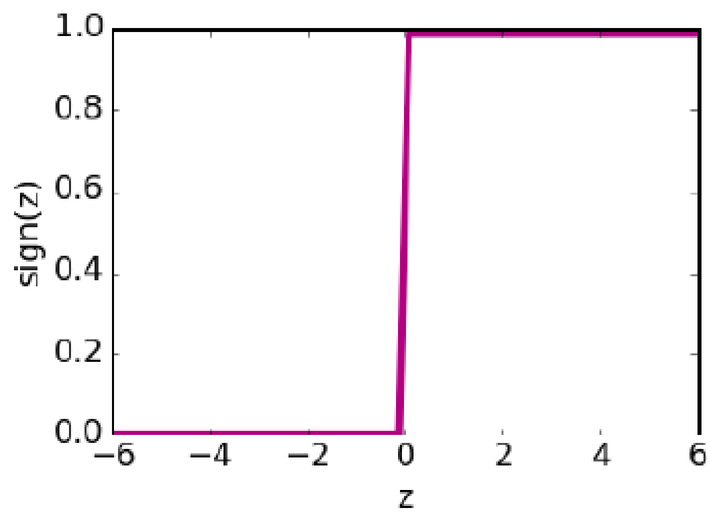
Logistic regression classifier:

- linear score with logistic link function

Simplest link function: sign(z)



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

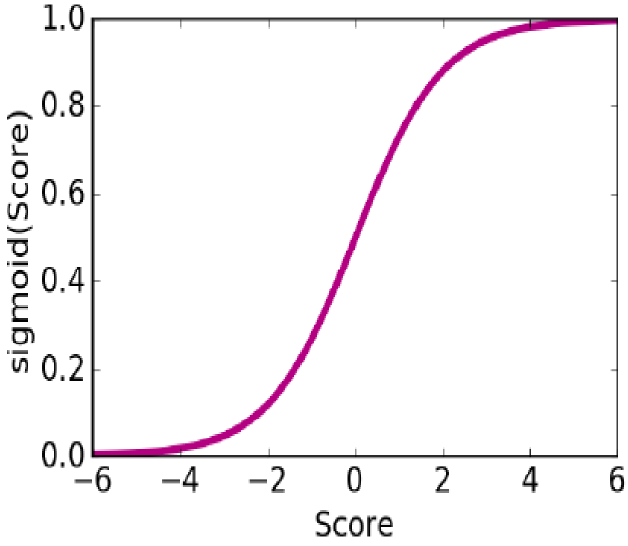


But, sign(z) only outputs -1 or +1, no probabilities in between

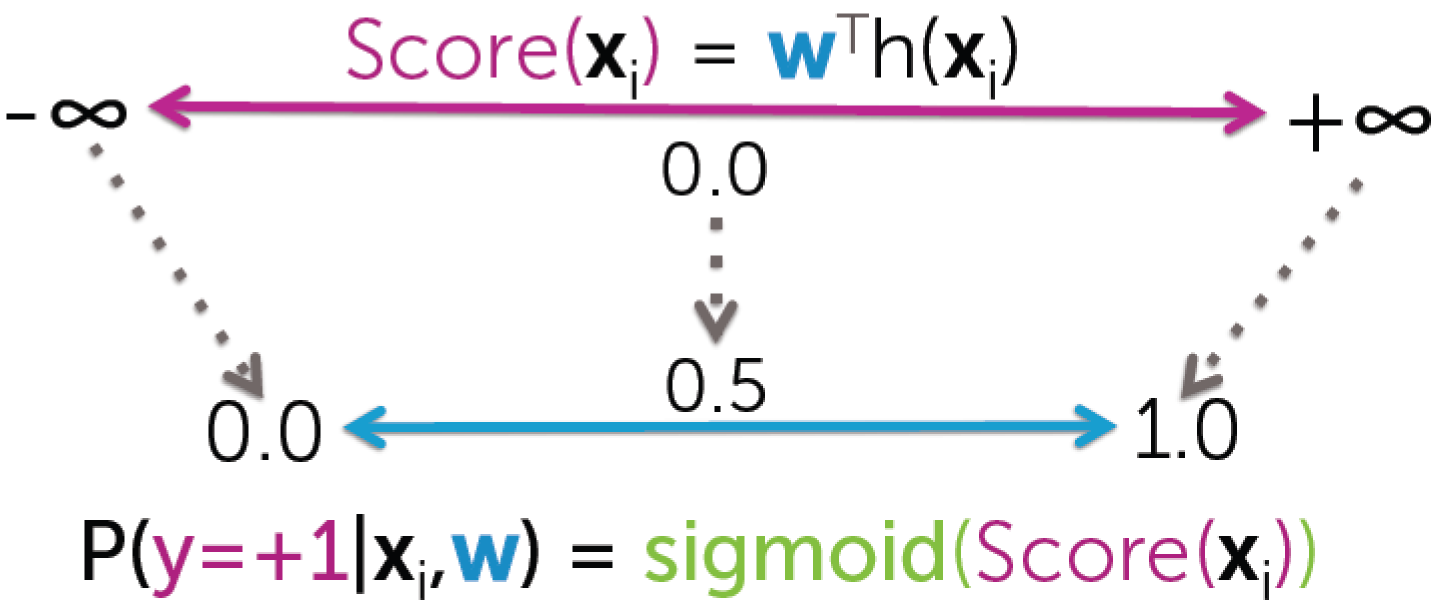
Logistic function (sigmoid, logit)

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

Score	-∞	-2	0.0	+2	+∞
sigmoid(Score)	0.0	0.12	0.5	0.88	1.0

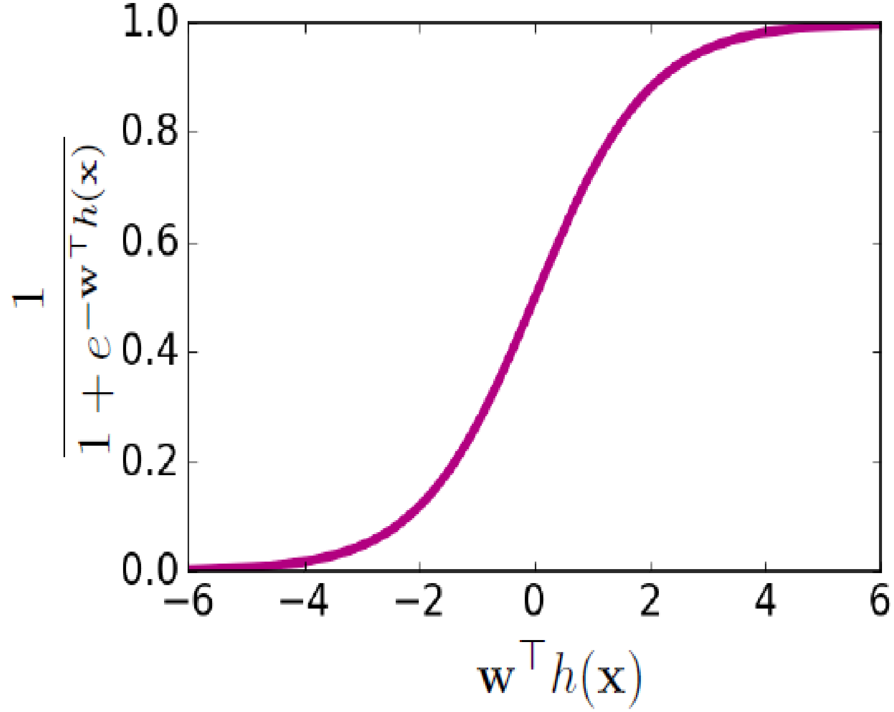


Logistic regression model



Understanding the logistic regression model

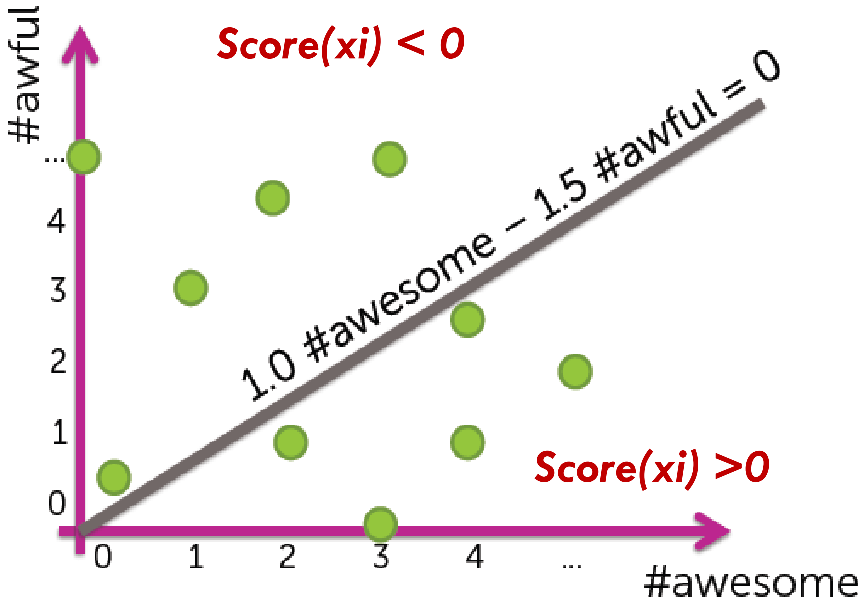
$$P(y=+1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^\top h(\mathbf{x})}}$$

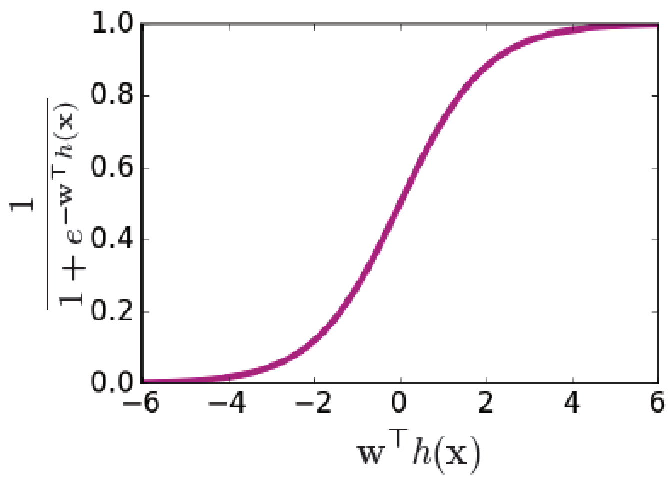


Score($\mathbf{x}_i$)	P(y=+1|$\mathbf{x}_i$,$\mathbf{w}$)
0	0.5
-2	0.12
2	0.88
4	0.98

Logistic regression

Logistic regression ➔ Linear decision boundary

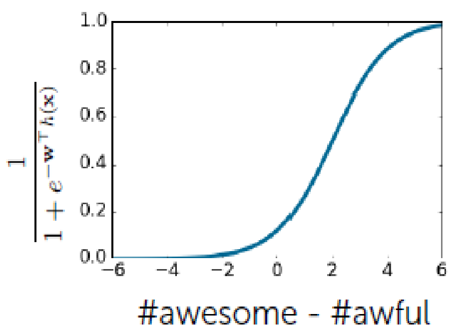




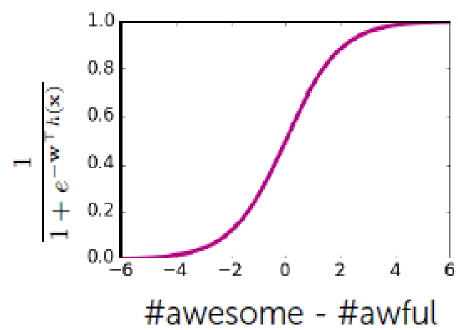
Effect of coefficients

Effect of coefficients on logistic regression model

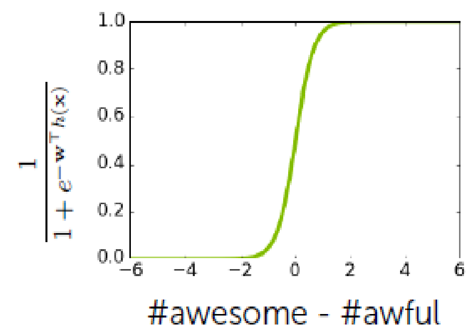
w_0	-2
$w_{\#awesome}$	+1
$w_{\#awful}$	-1



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

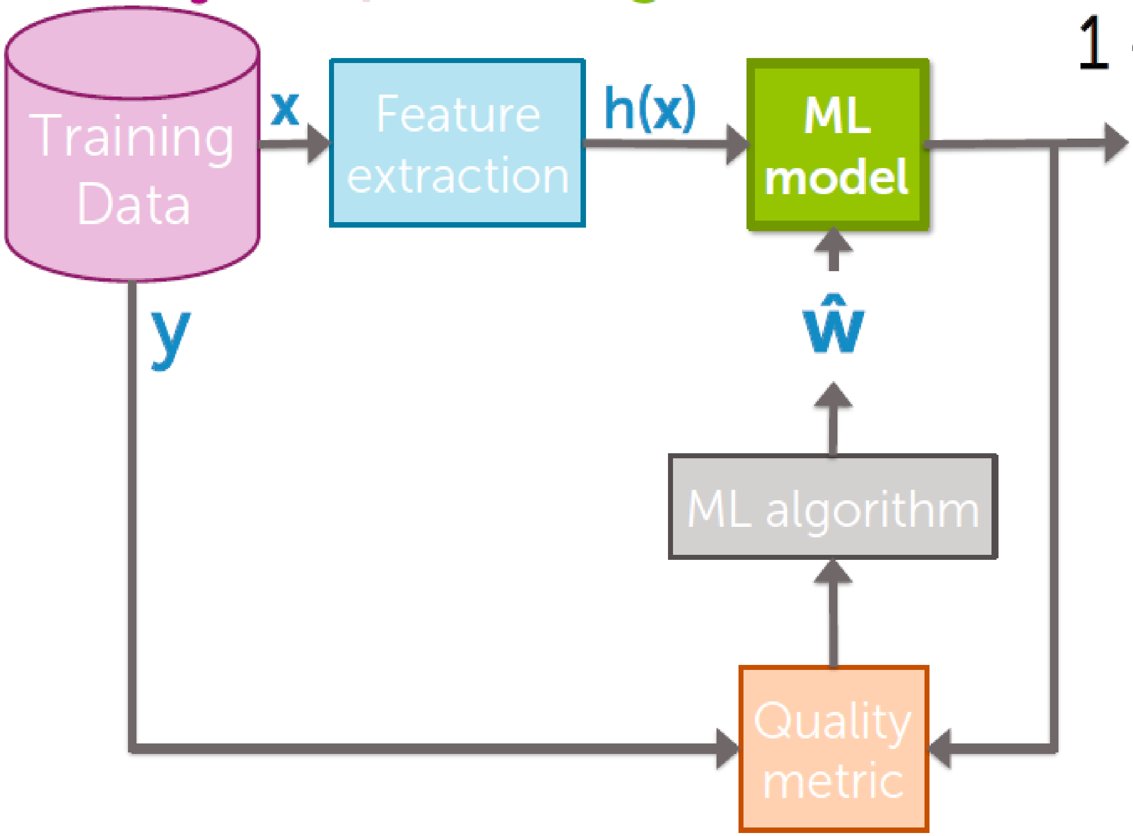


w_0	0
$w_{\#awesome}$	+3
$w_{\#awful}$	-3



Flow chart: ML model

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

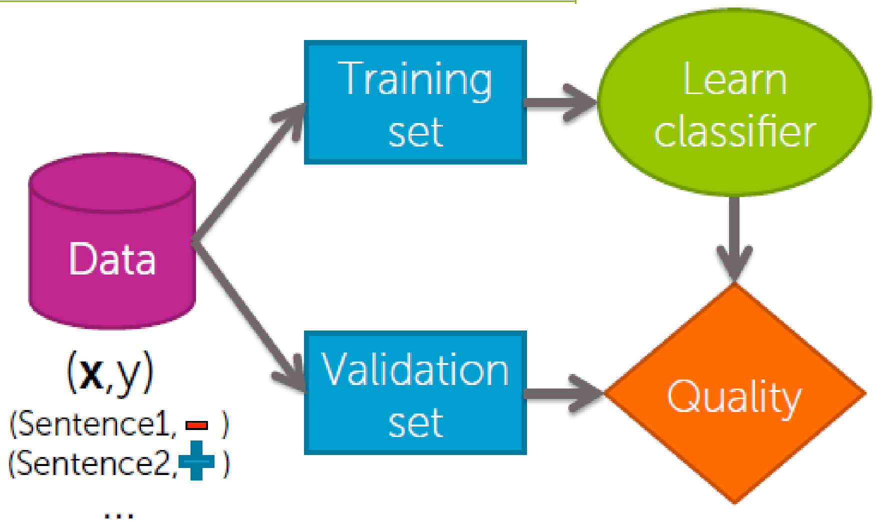


Learning logistic regression model

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
...	...	...

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



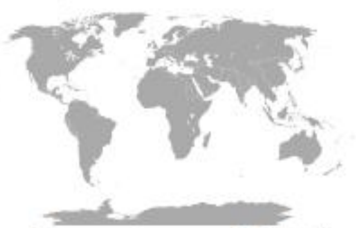
Categorical inputs

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

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

- Categorical inputs:

Gender
(Male, Female,…)

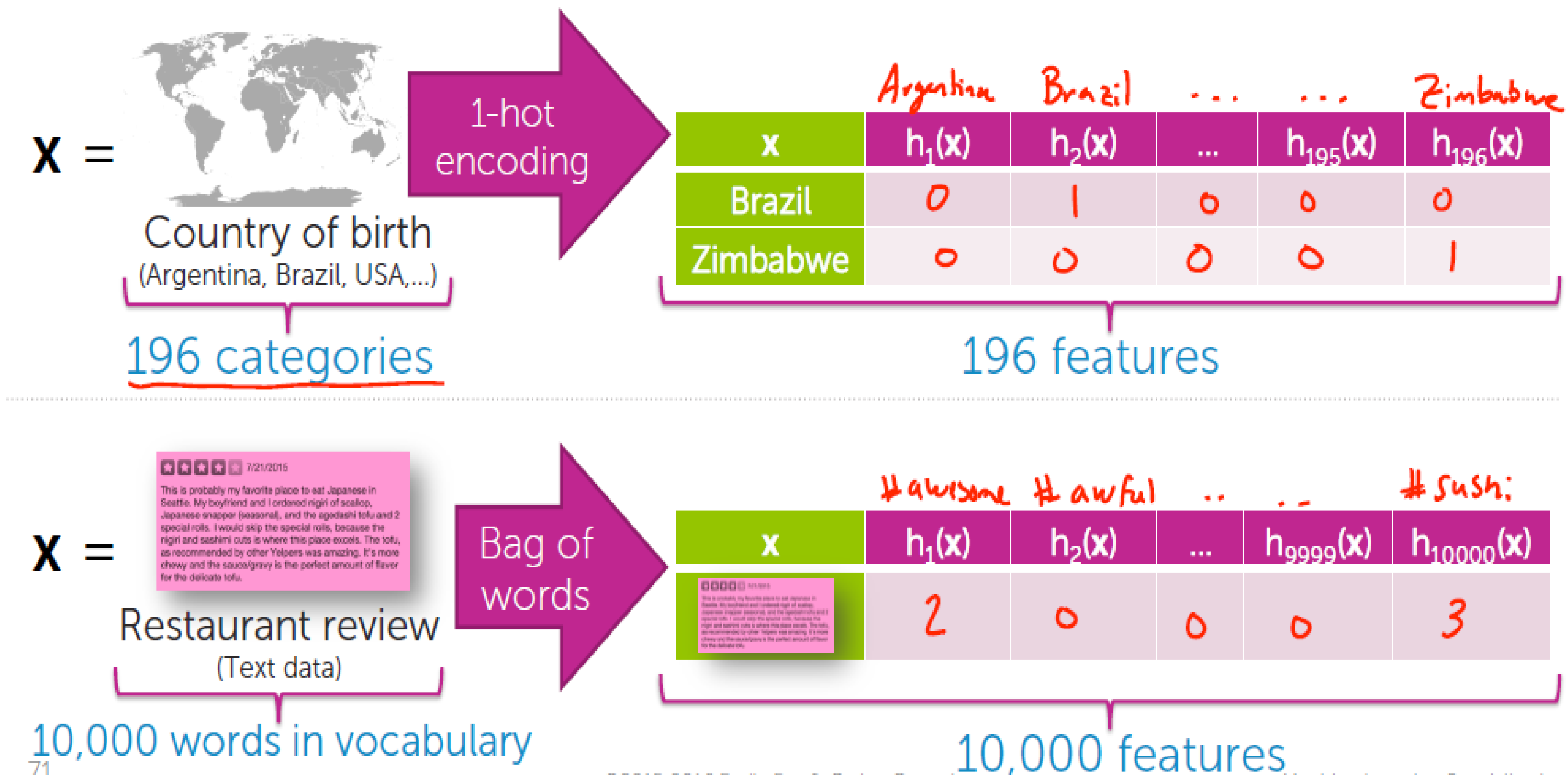


Country of birth
(Argentina, Brazil, USA,…)

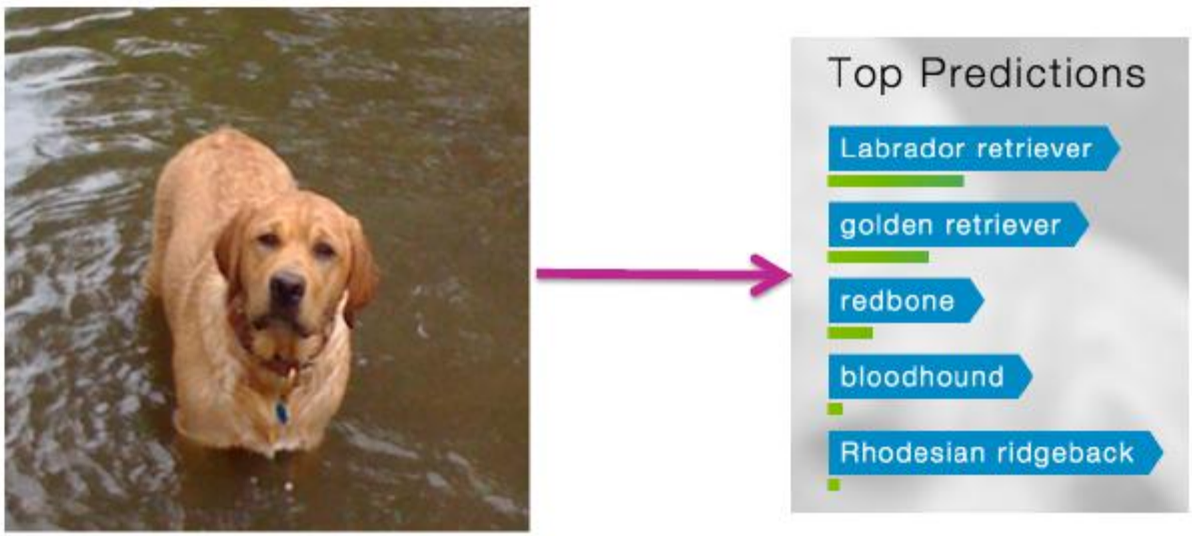
Zipcode
(10005, 98195,…)

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

Encoding categories as numeric features



Multiclass classification

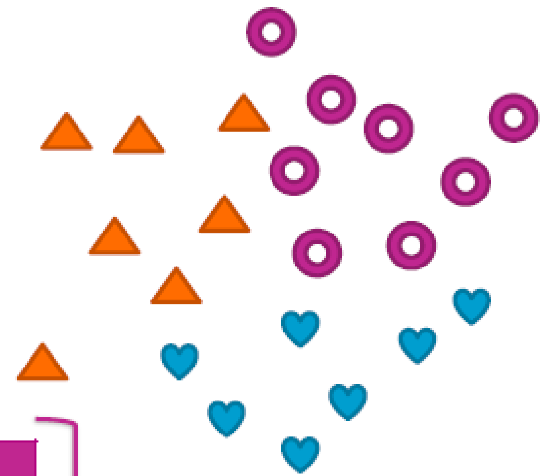


Input: x
Image pixels

Output: y
Object in image

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:

$\hat{P}(y=\triangle|\mathbf{x})$

$\hat{P}(y=\heartsuit|\mathbf{x})$

$\hat{P}(y=\circ|\mathbf{x})$

1 versus all

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

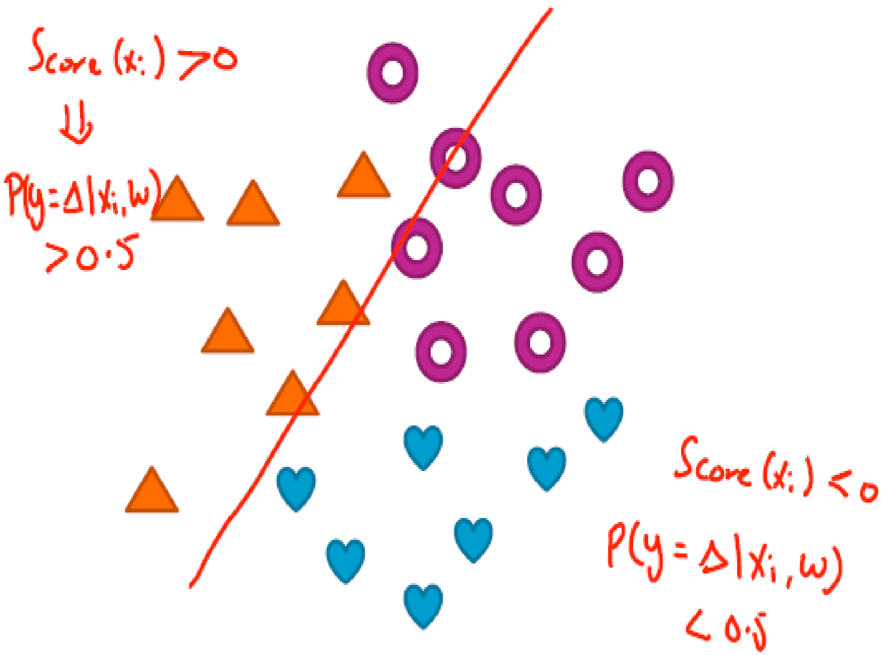
+1 class: points with $y_i=\triangle$
-1 class: points with $y_i=\heartsuit$ OR $\circ$

↓

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

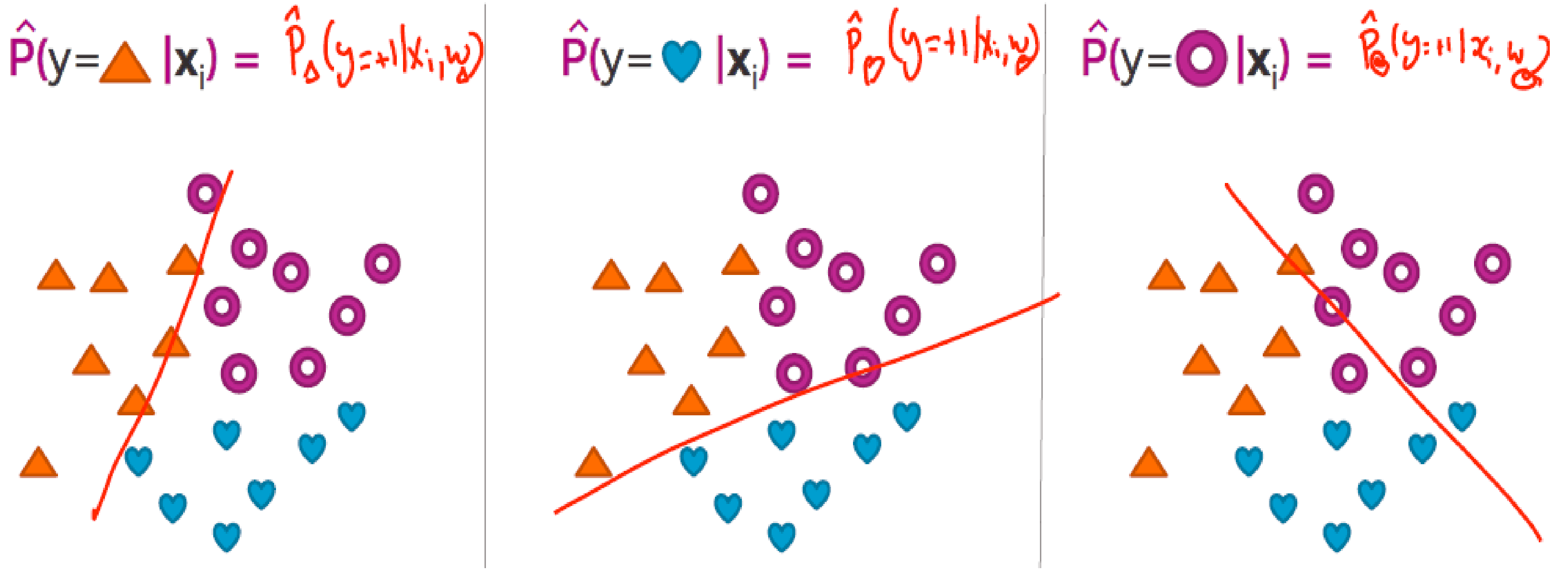
↓

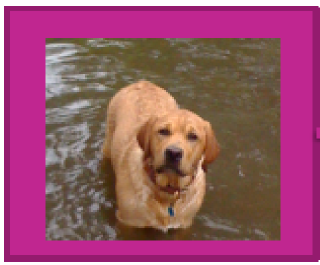
Predict: $\hat{P}(y=\triangle|\mathbf{x}_i)= \hat{P}_{\triangle}(y=+1|\mathbf{x}_i)$



1 versus all

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





Input: $\mathbf{x}_i$

Multiclass training

$\hat{P}_c(y=+1|\mathbf{x})$ = estimate of 1 vs all model for each class

Predict most likely class

max_prob = 0; ŷ = 0

For c = 1,...,C:

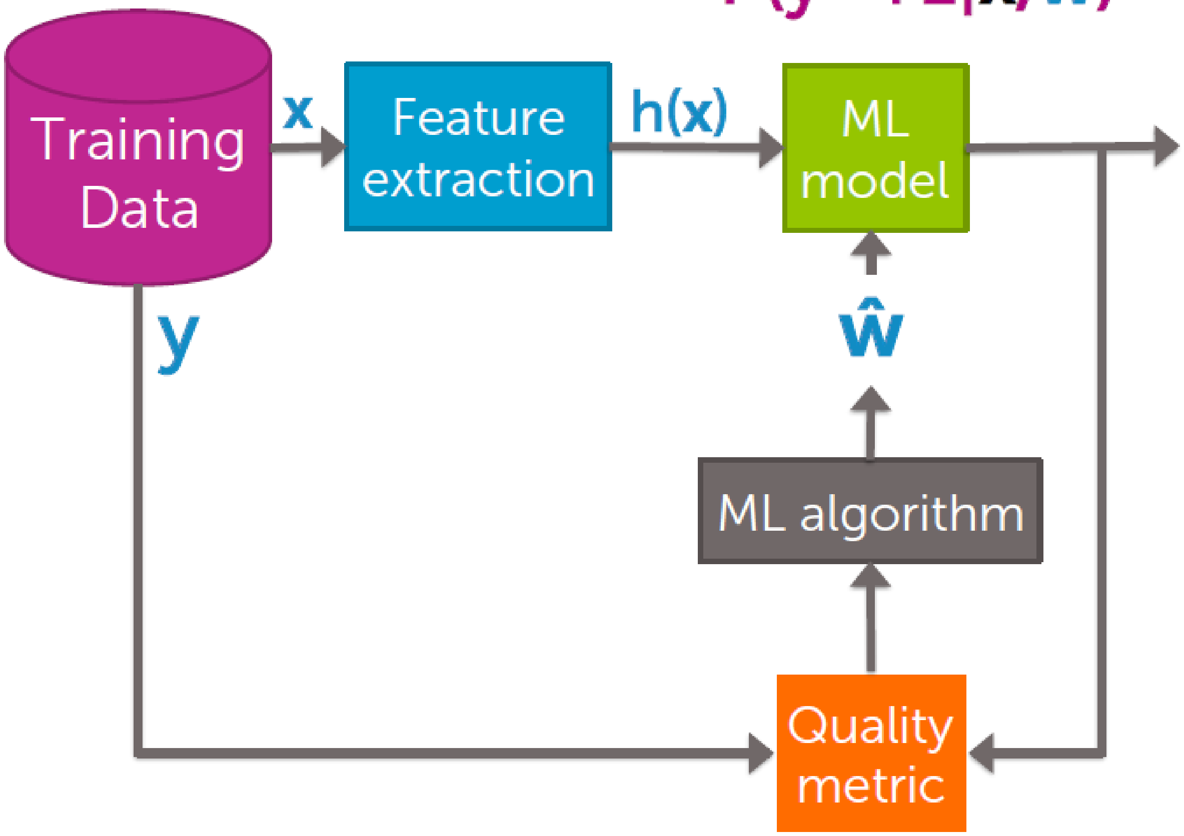
    If $\hat{P}_c(y=+1|\mathbf{x}_i)$ > max_prob:

        $\hat{y}$ = c

        max_prob = $\hat{P}_c(y=+1|\mathbf{x}_i)$

Summary: Logistic regression classifier

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



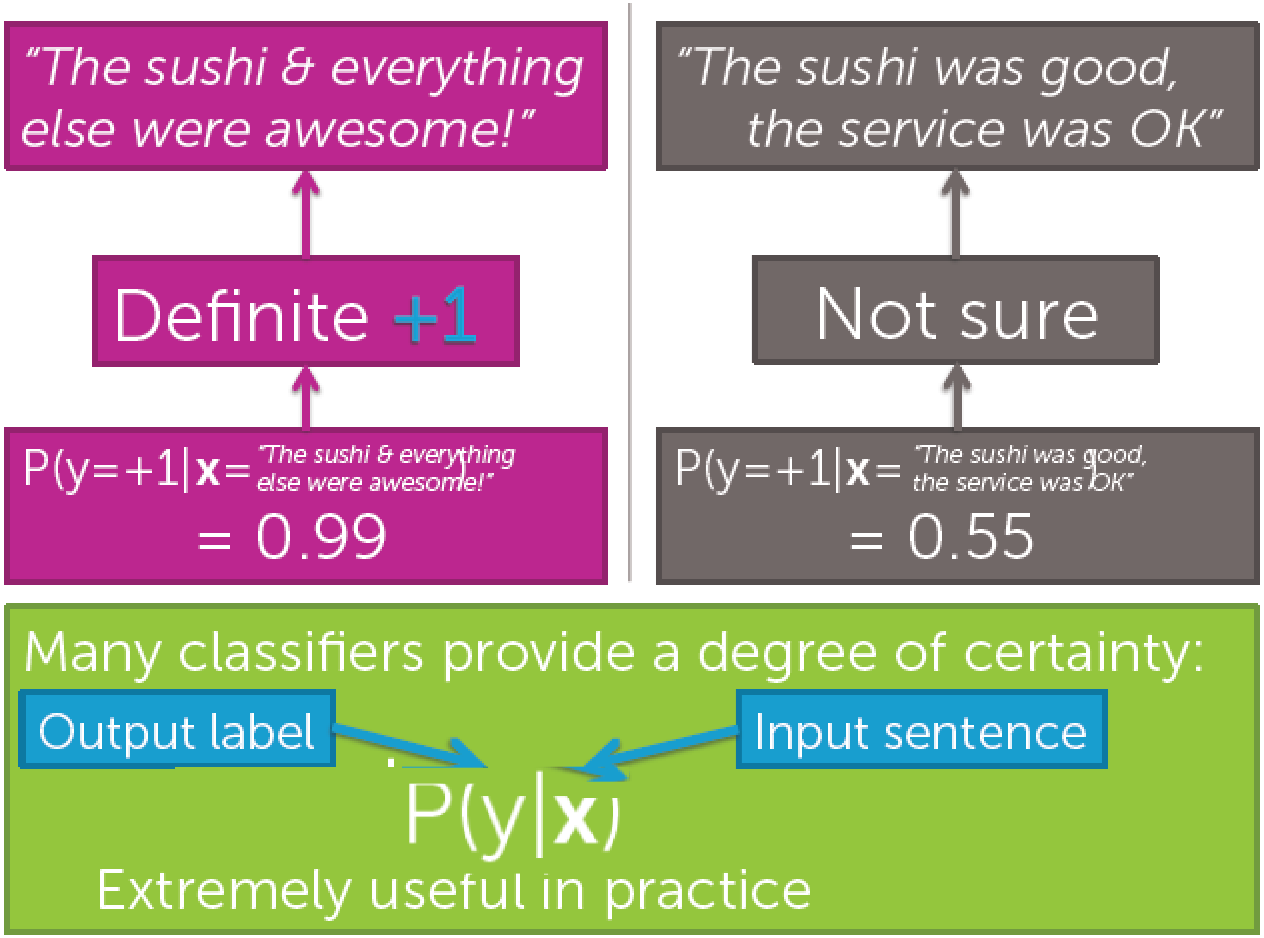
What you can do now…

- Describe decision boundaries and linear classifiers
- Use class probability to express degree of confidence in prediction
- Define a logistic regression model
- Interpret logistic regression outputs as class probabilities
- Describe impact of coefficient values on logistic regression output
- Use 1-hot encoding to represent categorical inputs
- Perform multiclass classification using the 1-versus-all approach

Linear classifier

- Parameters learning

Learn a probabilistic classification model



A (linear) classifier

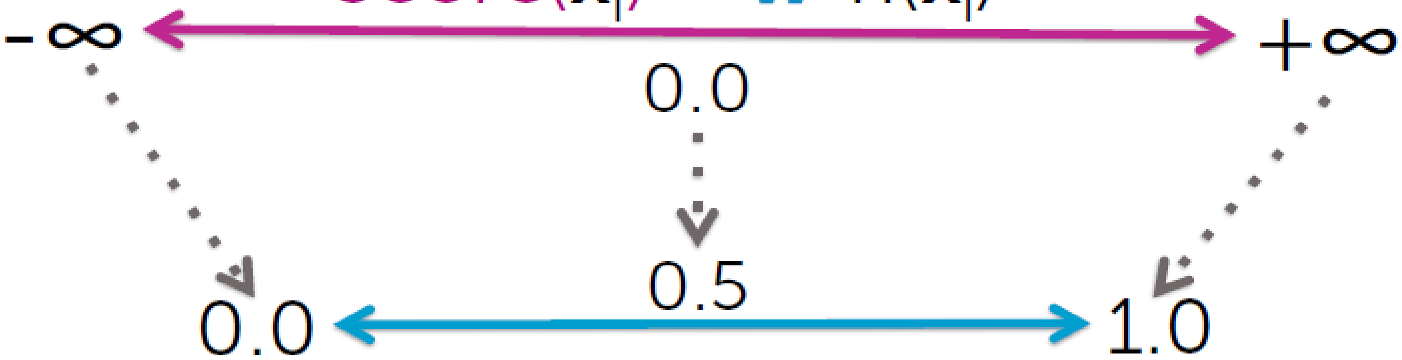
- Will use training data to learn a weight or coefficient for each word

Word	Coefficient	Value
	$\hat{w}_0$	-2.0
good	$\hat{w}_1$	1.0
great	$\hat{w}_2$	1.5
awesome	$\hat{w}_3$	2.7
bad	$\hat{w}_4$	-1.0
terrible	$\hat{w}_5$	-2.1
awful	$\hat{w}_6$	-3.3
restaurant, the, we, …	$\hat{w}_7, \hat{w}_8, \hat{w}_9, \ldots$	0.0
…		…

Logistic regression

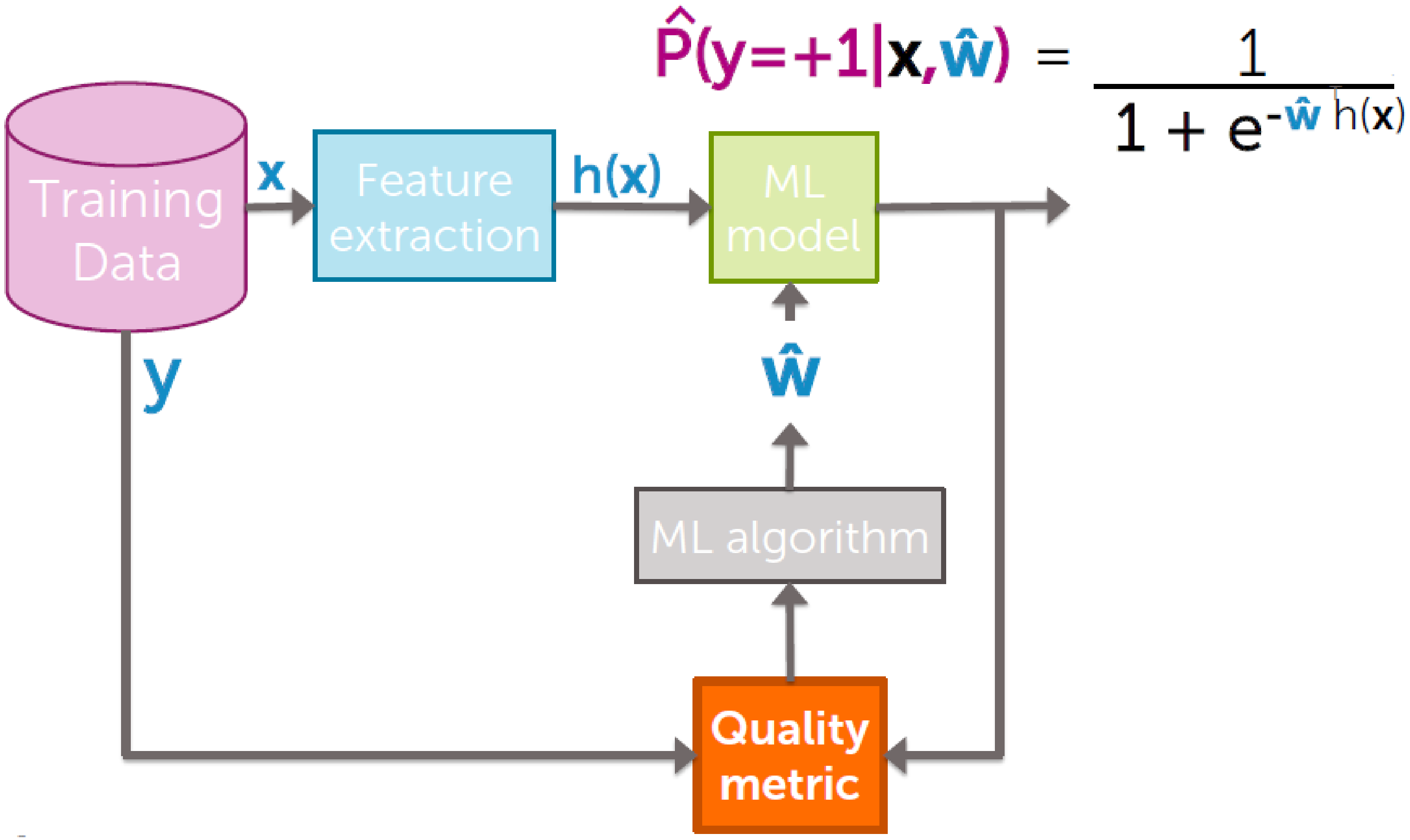
Logistic regression model

$$\text{Score}(\mathbf{x}_i) = \hat{\mathbf{w}}^T h(\mathbf{x}_i)$$



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

Flow chart: **Quality metric**



Learning problem

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
1	1	+1
2	4	-1
0	3	-1
0	1	-1
2	1	+1

Finding best coefficients

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

x[1] = #awesome	x[2] = #awful	y = sentiment
2	1	+1
4	1	+1
1	1	+1
2	1	+1

$P(y=+1|x_i,\mathbf{w}) = 0.0$

$P(y=+1|x_i,\mathbf{w}) = 1.0$

Pick $\hat{\mathbf{w}}$ that makes

Quality metric: probability of data

x[1] = #awesome	x[2] = #awful	y = sentiment
2	1	+1

x_1 $y_1 =$

If model good, should predict:

$\hat{y}_1 = +1$

Pick **w** to maximize:

$P(y=+1 \mid x_1, w) = P(y=+1 \mid x[1]=2, x[2]=1, w)$

x[1] = #awesome	x[2] = #awful	y = sentiment
0	2	-1

$x_2 =$ $y_2 =$

If model good, should predict:

$\hat{y}_2 = -1$

Pick **w** to maximize:

$P(y=-1 \mid x_2, w)$

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_i: +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 $\mathbf{w}$

$$\ell(\mathbf{w}) = P(y=+1|\mathbf{x}[1]=2, \mathbf{x}[2]=1,\mathbf{w}) \; P(y=-1|\mathbf{x}[1]=0, \mathbf{x}[2]=2,\mathbf{w}) \; P(y=-1|\mathbf{x}[1]=3, \mathbf{x}[2]=3,\mathbf{w}) \; P(y=+1|\mathbf{x}[1]=4, \mathbf{x}[2]=1,\mathbf{w})$$

$$= P(y_1|\mathbf{x}_1,\mathbf{w}) \; P(y_2|\mathbf{x}_2,\mathbf{w}) \; P(y_3|\mathbf{x}_3,\mathbf{w}) \; P(y_4|\mathbf{x}_4,\mathbf{w})$$

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

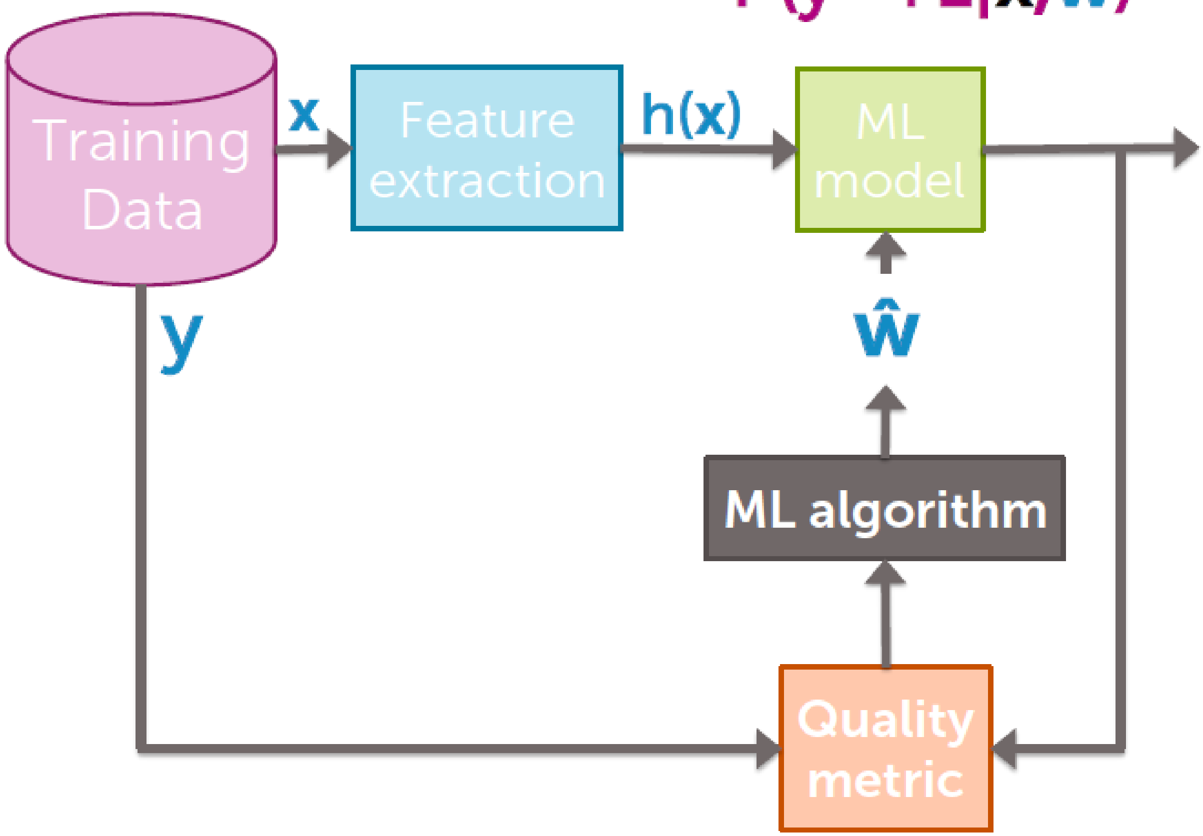
Num. of data points → N

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

17

Flow chart: ML algorithm

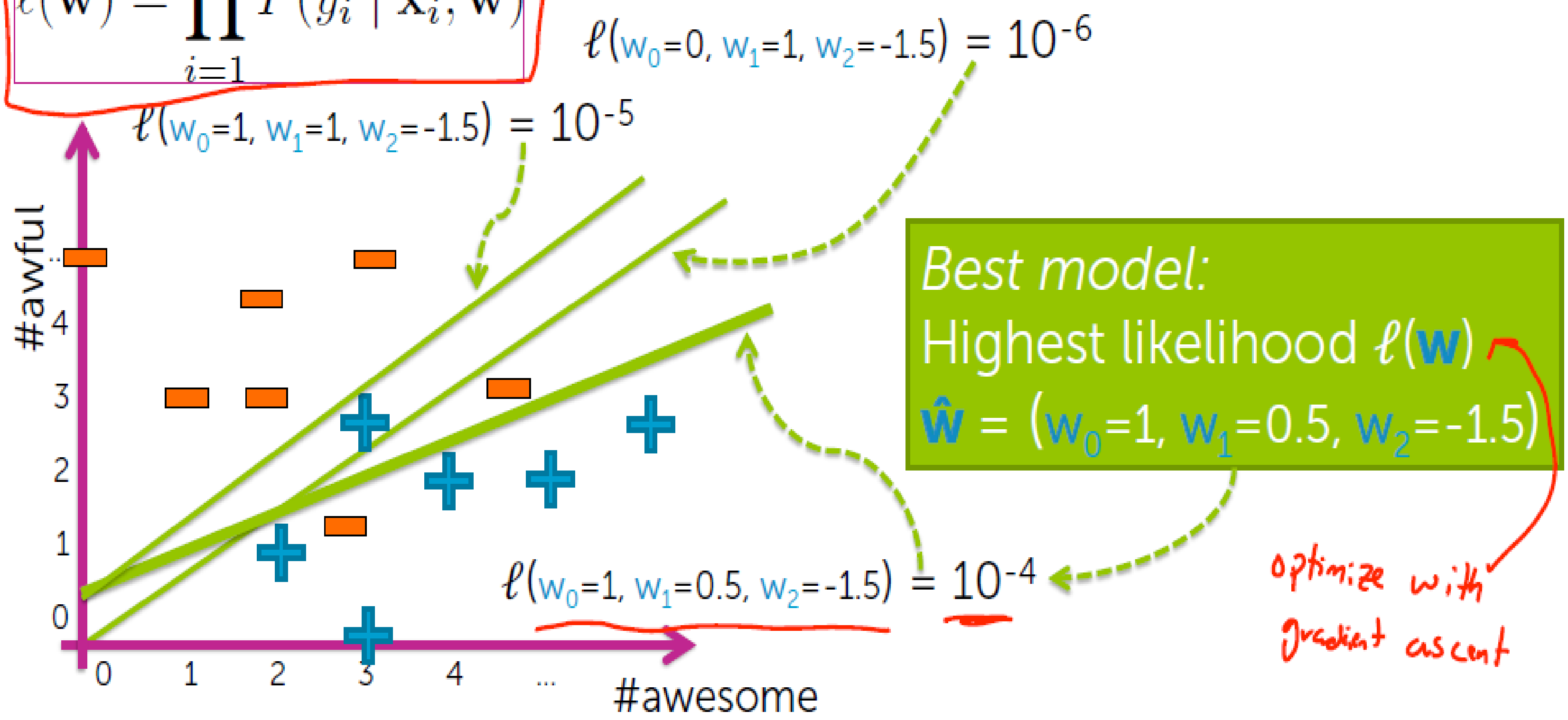
$$\hat{P}(y=+1|\mathbf{x},\hat{\mathbf{w}}) = \frac{1}{1+e^{-\hat{\mathbf{w}}^\top h(\mathbf{x})}}$$



Find „best” classifier

Maximize likelihood over all possible w_0, w_1, w_2

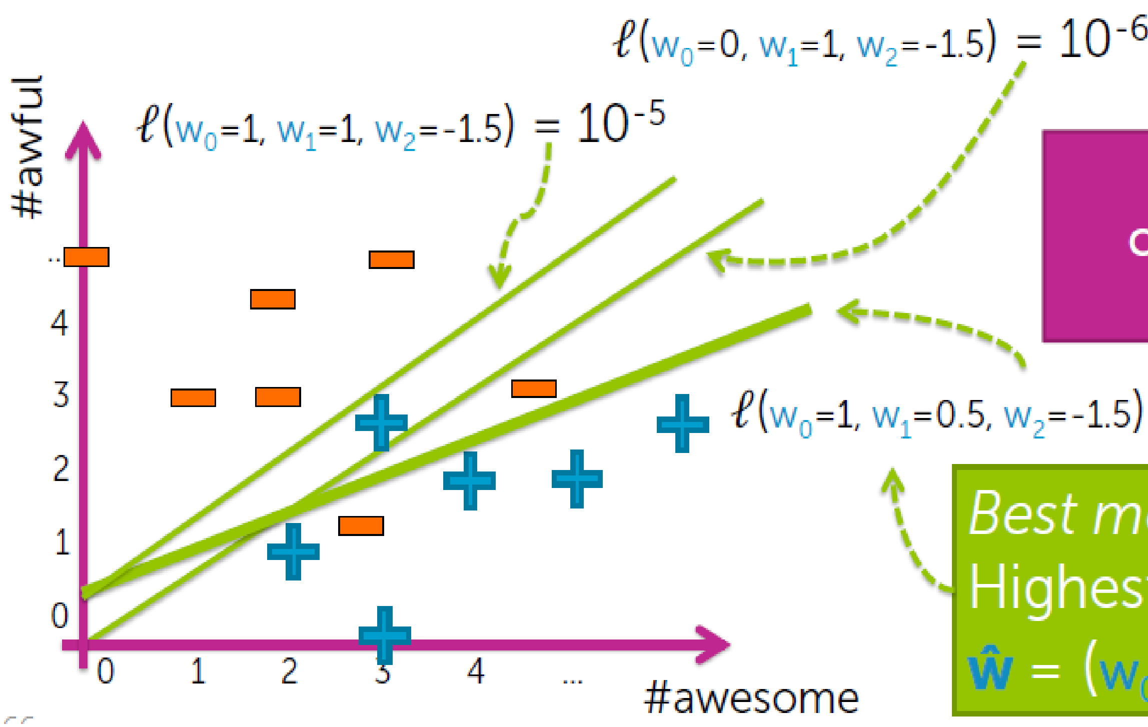
$$\ell(\mathbf{w}) = \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$



Find best classifier

Maximize quality metric over all possible w_0, w_1, w_2

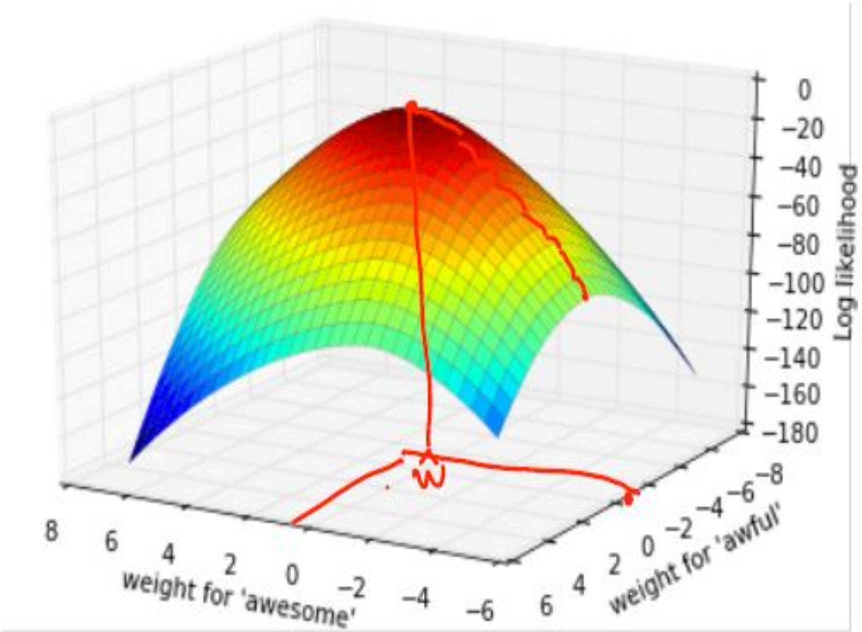
Likelihood $\ell(\mathbf{w})$



Find best model coefficients **w** with gradient ascent!

Best model:
Highest likelihood $\ell(\mathbf{w})$
$\hat{\mathbf{w}} = (w_0=1, w_1=0.5, w_2=-1.5)$

Maximizing likelihood



Maximize function over all possible w_0, w_1, w_2

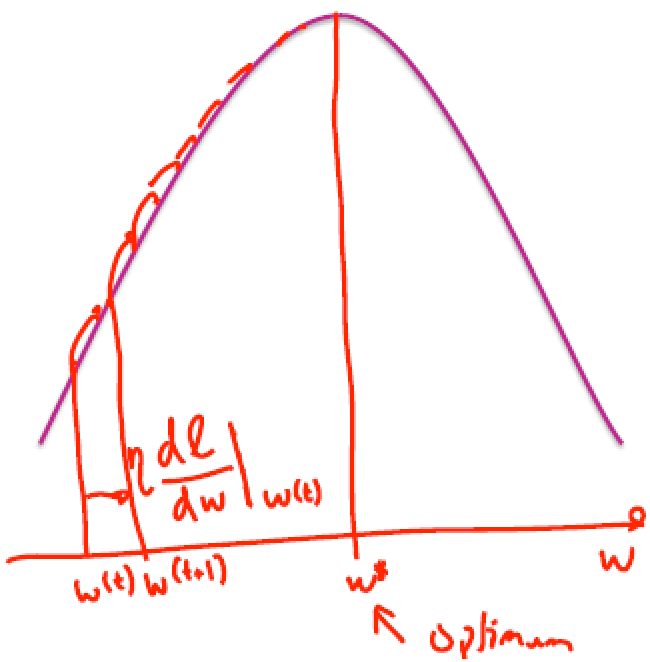
$$\max_{w_0, w_1, w_2} \prod_{i=1}^{N} P(y_i \mid \mathbf{x}_i, \mathbf{w})$$

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

No closed-form solution ➔ use gradient ascent

Gradient ascent

Finding the max via hill climbing



Algorithm:

while not converged

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

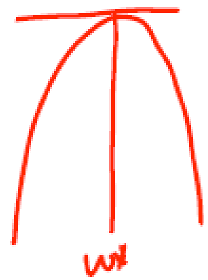
step size

Gradient ascent

Convergence criteria

For convex functions, optimum occurs when

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



In practice, stop when

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

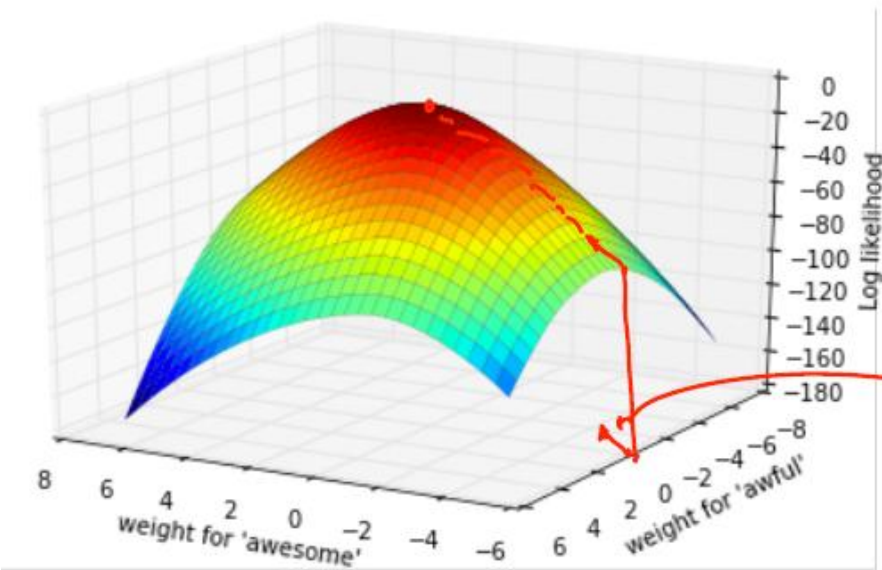
↑ tolerance

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

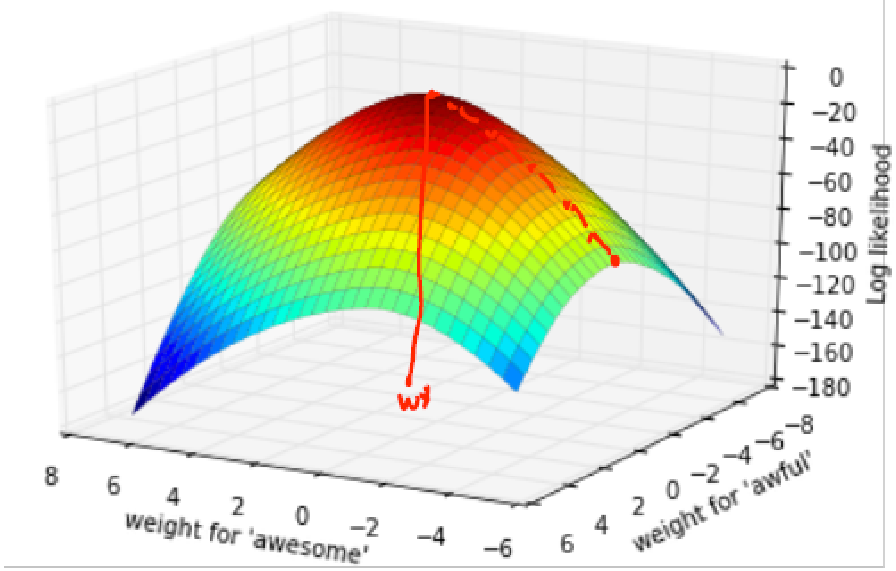


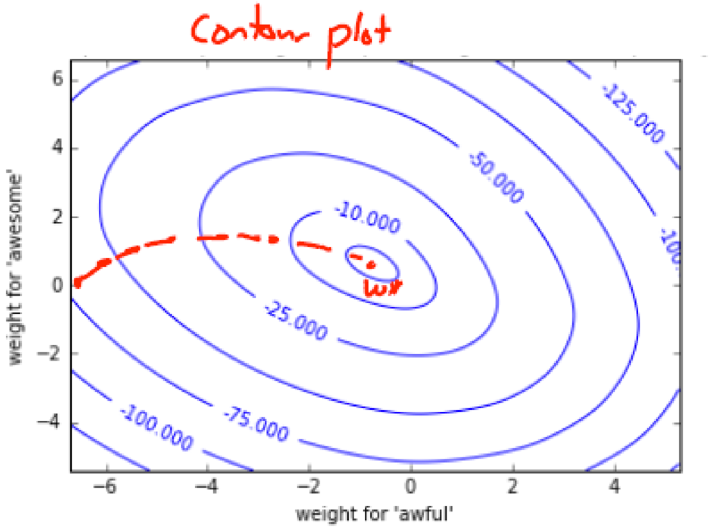
$$\nabla \ell(\mathbf{w}) = \begin{bmatrix} \frac{\partial \ell}{\partial w_0} \\ \frac{\partial \ell}{\partial w_1} \\ \vdots \\ \frac{\partial \ell}{\partial w_D} \end{bmatrix}$$

← D+1 dim vector

Gradient ascent

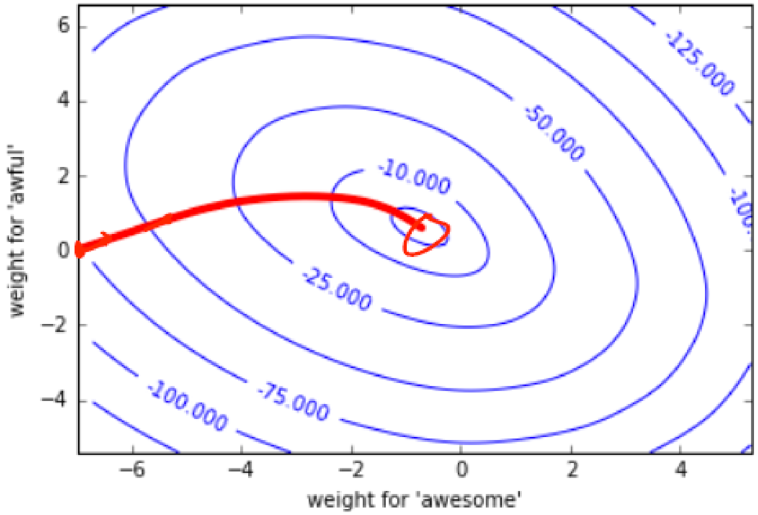
Contour plots





Gradient ascent

Gradient ascent



Algorithm:

$w^{(0)} = 0$, random, or something smart.

while not converged

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \nabla \ell(\mathbf{w}^{(t)})$$

step size

The log trick, often used in ML…

- Products become sums:

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

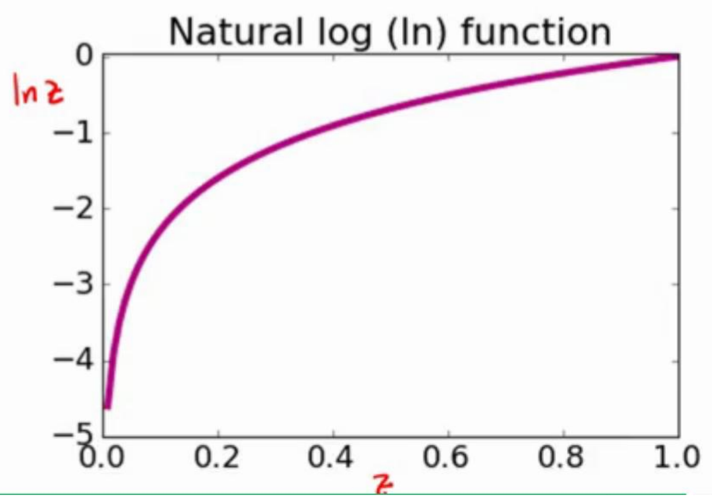
- Doesn't change maximum!
  - If $\hat{\mathbf{w}}$ maximizes f(**w**):

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

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

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

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



Derivative for logistic regression

Derivative of (log-)likelihood

Sum over data points

Feature value

Difference between truth and prediction

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

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

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

$h_1(x)$ = # awesome

x[1]	x[2]	y	$P(y=+1\|x_i,w)$	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:

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

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

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

Derivative for logistic regression

Derivative of (log-)likelihood: Interpretation

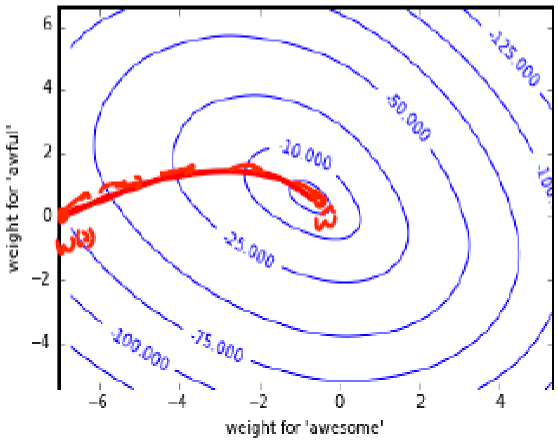
Sum over data points — Feature value — Difference between truth and prediction

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

Δ_i

If $h_j(\mathbf{x}_i)=1$:	$P(y=+1 \mid x_i, w) \approx 1$	$P(y=+1 \mid x_i, w) \approx 0$
$y_i=+1$	$\Delta_i = (1-1) \approx 0$ → don't change anything	$\Delta_i \approx 1 \Rightarrow$ increase w_j $\Rightarrow$ Score(x_i) becomes larger $\Rightarrow P(y=+1 \mid x_i, w)$ increases
$y_i=-1$	$\Delta_i = -1 \Rightarrow w_j$ to decrease $\Rightarrow$ Score(x_i) decreases $\Rightarrow P(y=+1 \mid x_i, w)$ decrease	$\Delta_i \approx 0 \Rightarrow$ don't change anything

Gradient ascent for logistic regression



init $\mathbf{w}^{(1)}=0$ (or randomly, or smartly), t=1

while $||\nabla \ell(\mathbf{w}^{(t)})|| > \varepsilon$ ← tolerance

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

for j=0,…,D ← coefficient

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

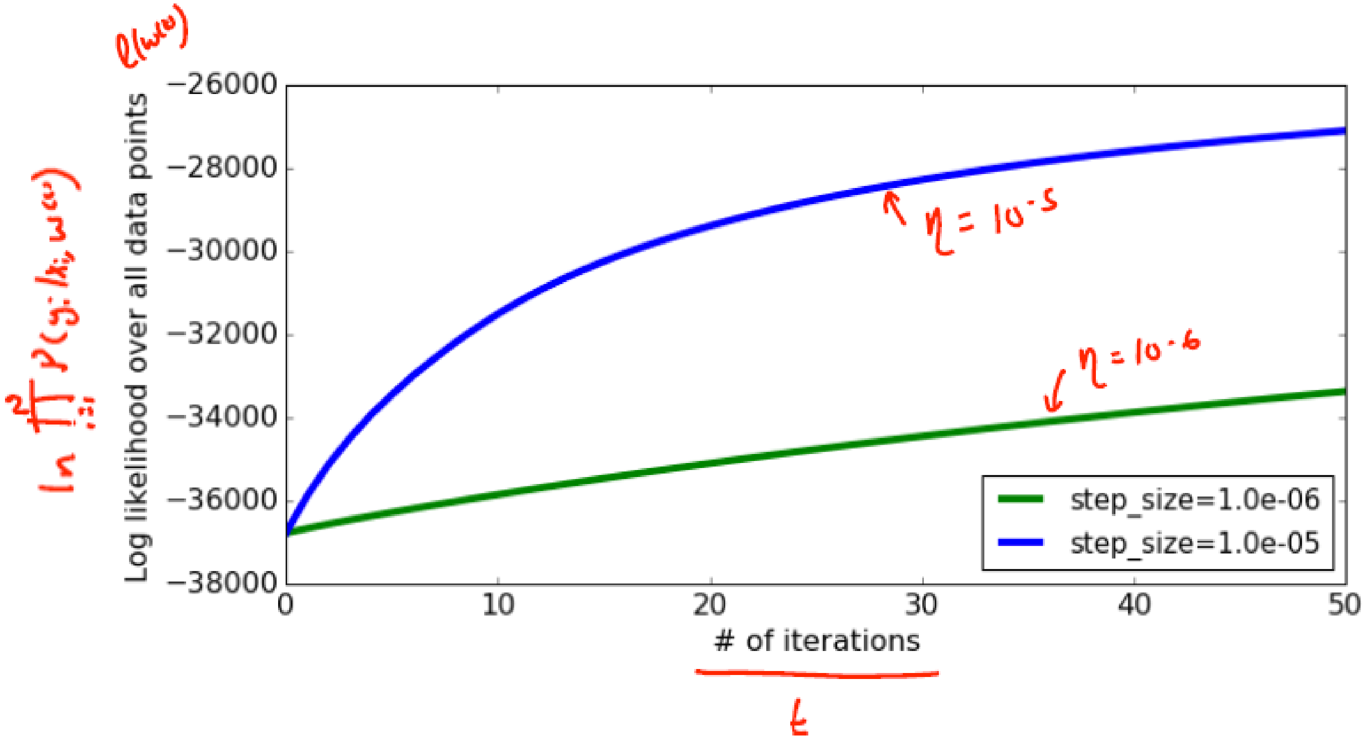
$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta$ partial[j]

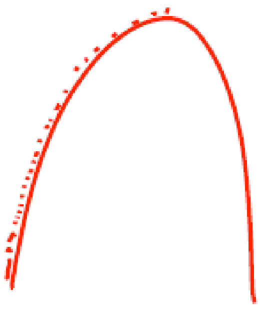
t ← t + 1

step size (η); $\frac{\partial \ell(w^{(t)})}{\partial w_j}$ (partial[j])

Choosing the step size

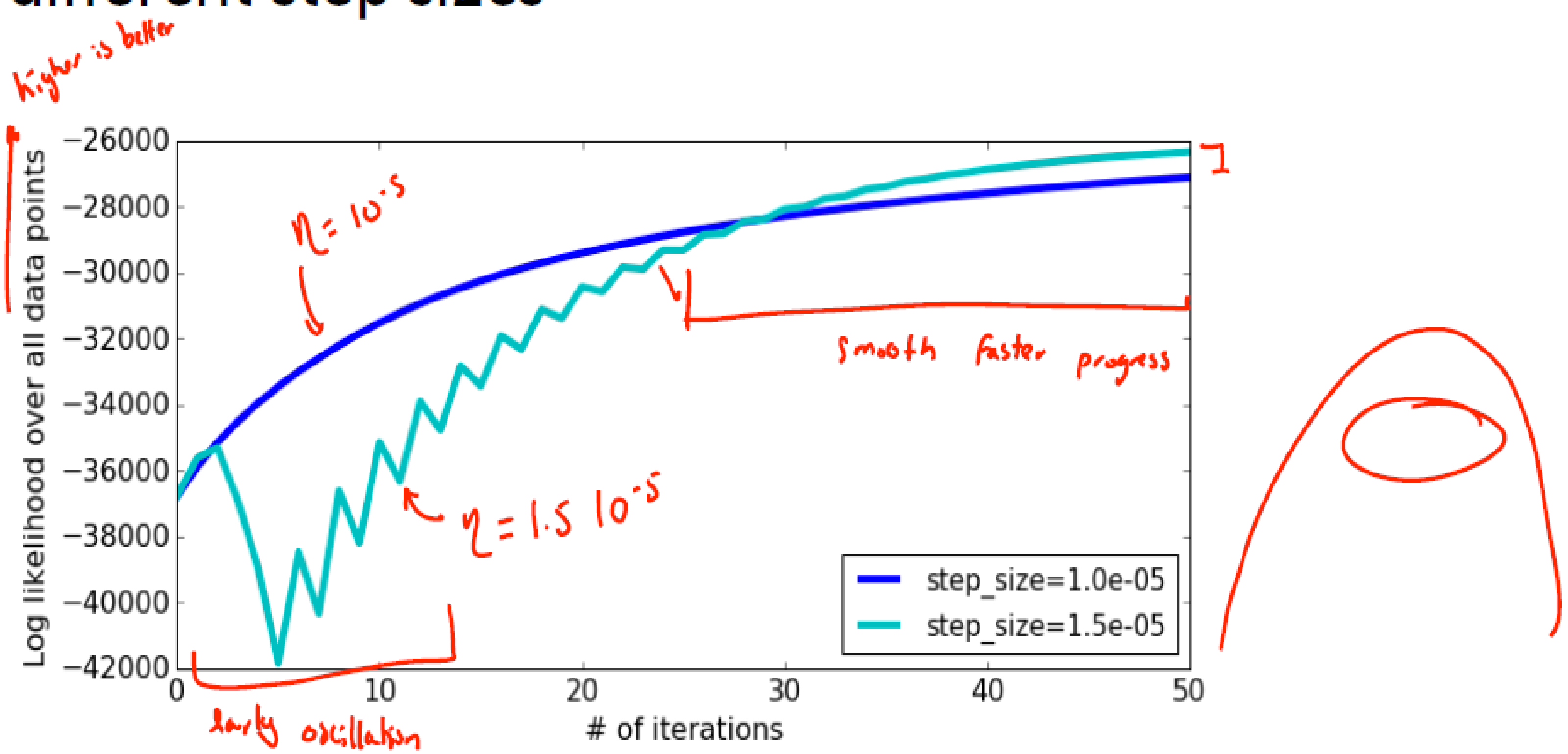
If step size is too small, can take a long time to converge





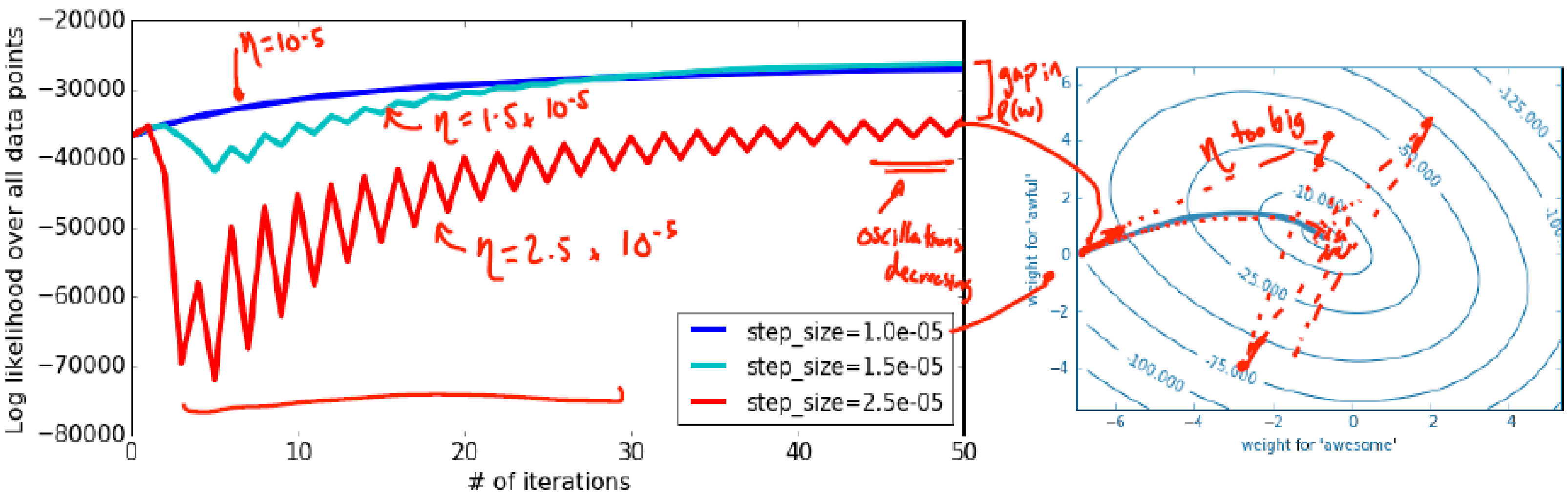
Choosing the step size

Compare converge with
different step sizes



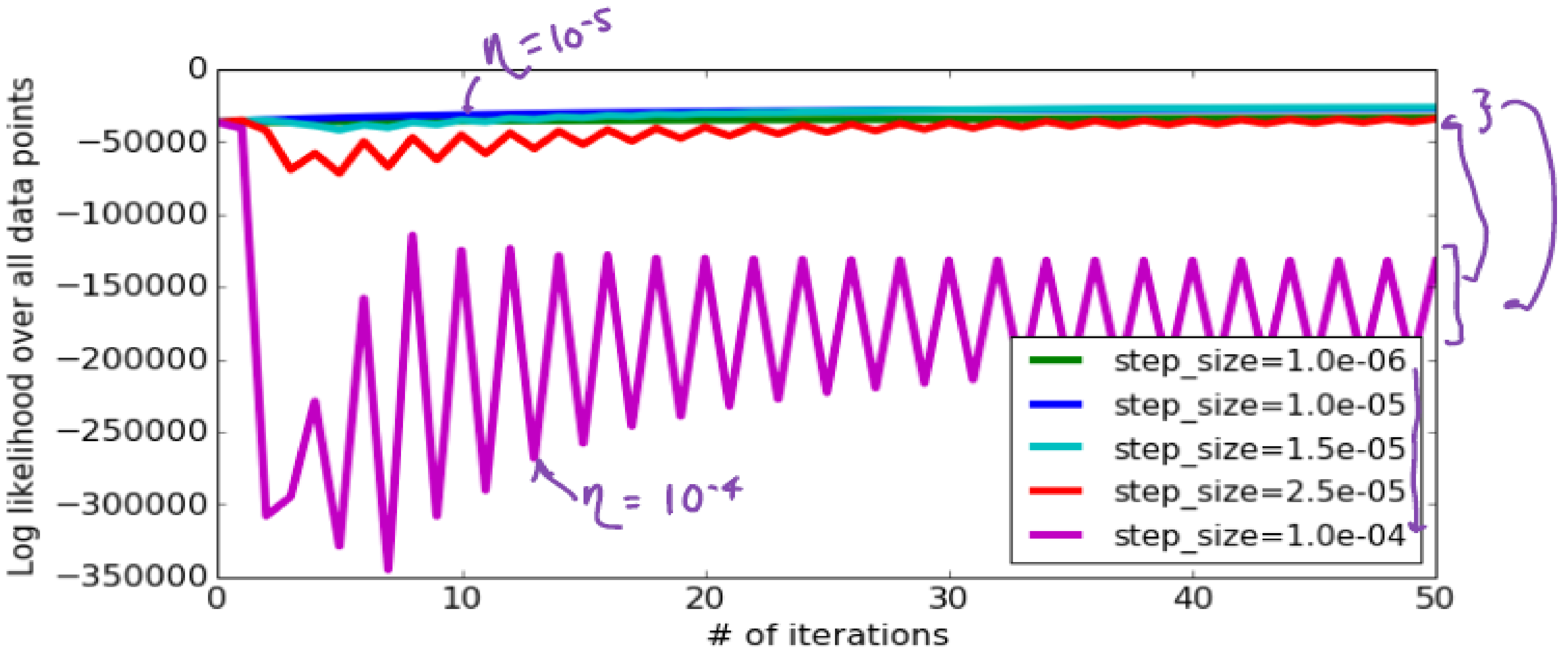
Choosing the step size

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

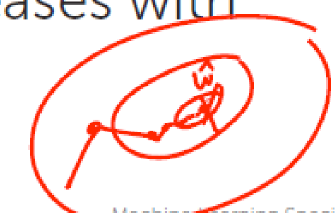
Simple rule of thumb for picking step size η

- 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

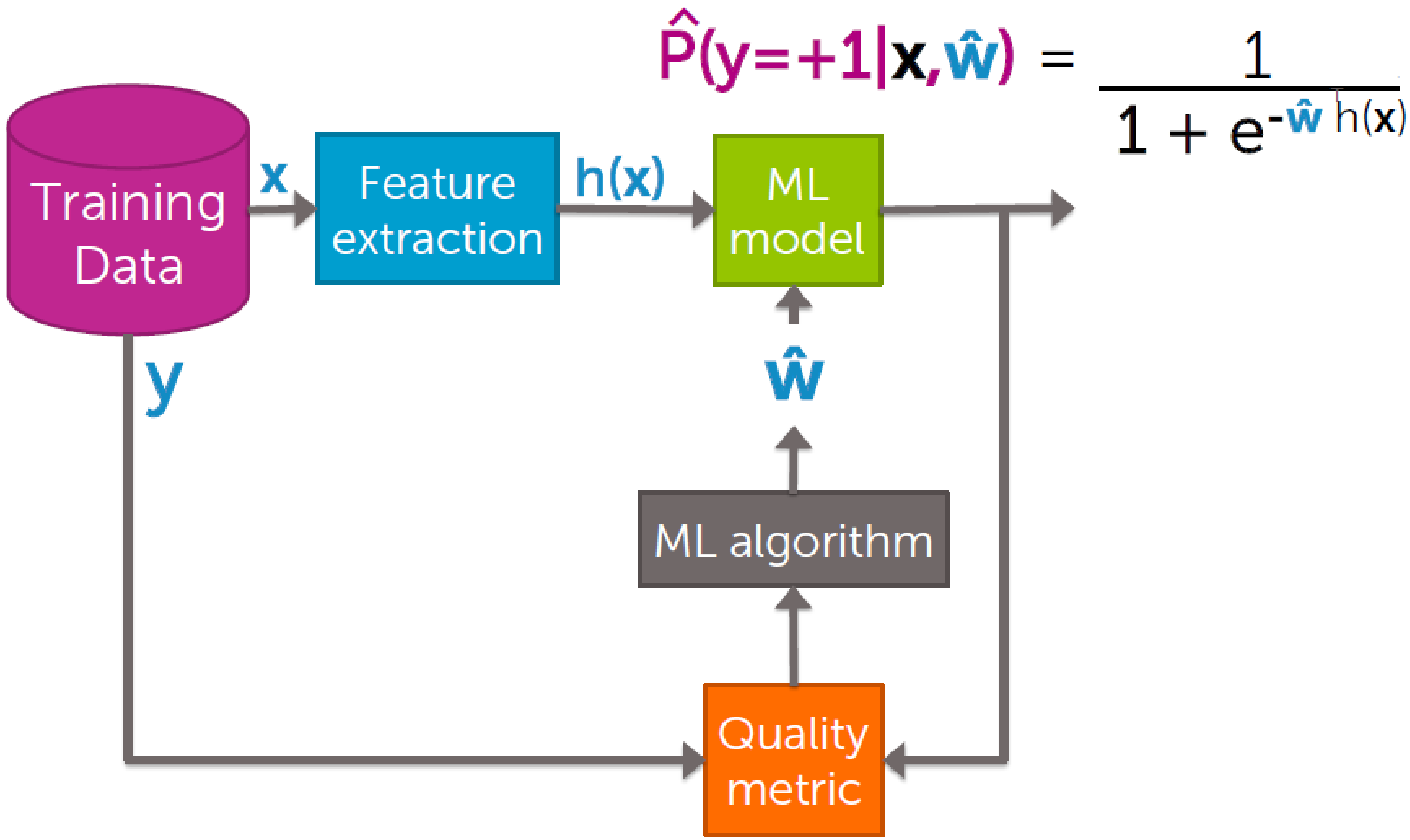
- *Advanced tip*: can also try step size that decreases with iterations, e.g.,

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



30/10, 6/11 2024

Flow chart: final look at it



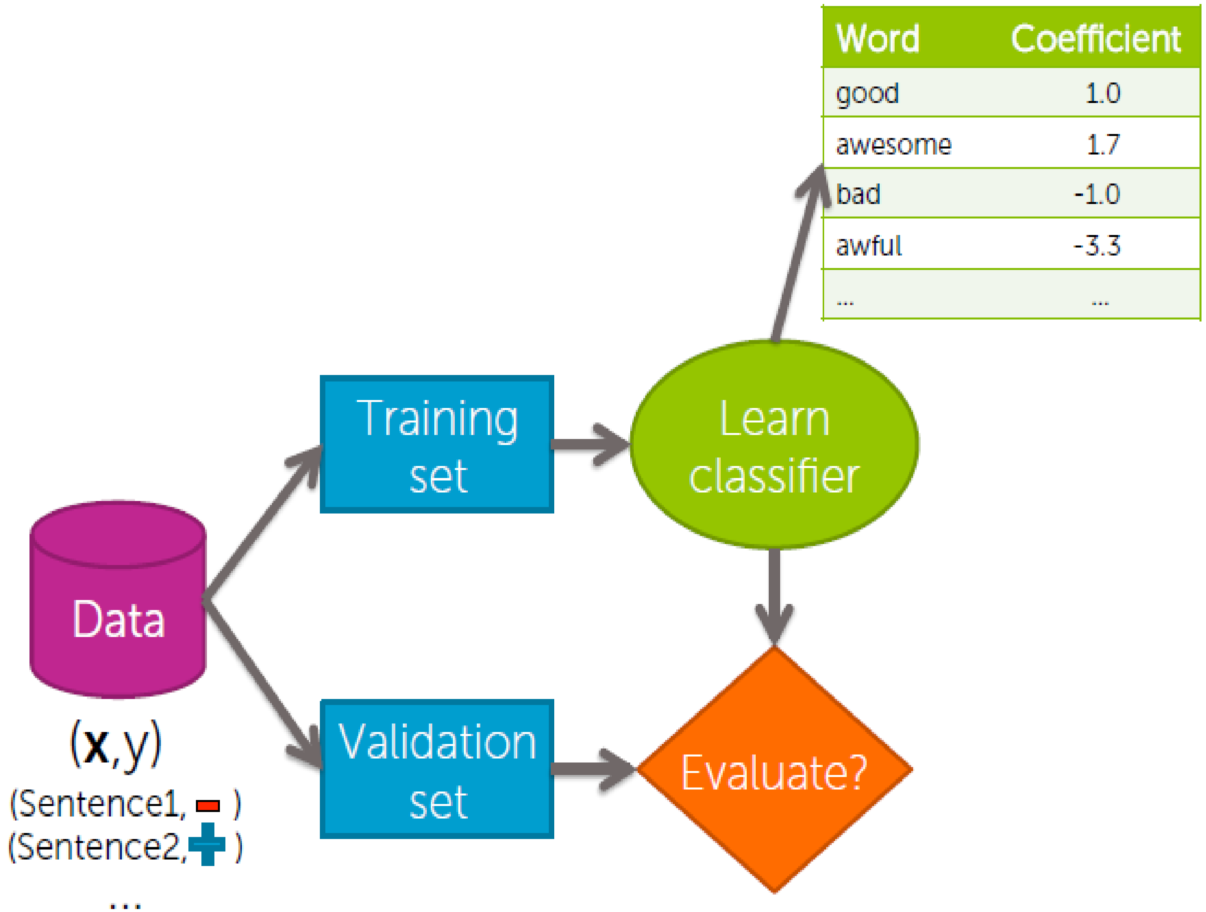
What you can do now

- Measure quality of a classifier using the likelihood function
- Interpret the likelihood function as the probability of the observed data
- Learn a logistic regression model with gradient descent
- (Optional) Derive the gradient descent update rule for logistic regression

Linear classifier

- Overfitting & regularization

Training a classifier = Learning the coefficients



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

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

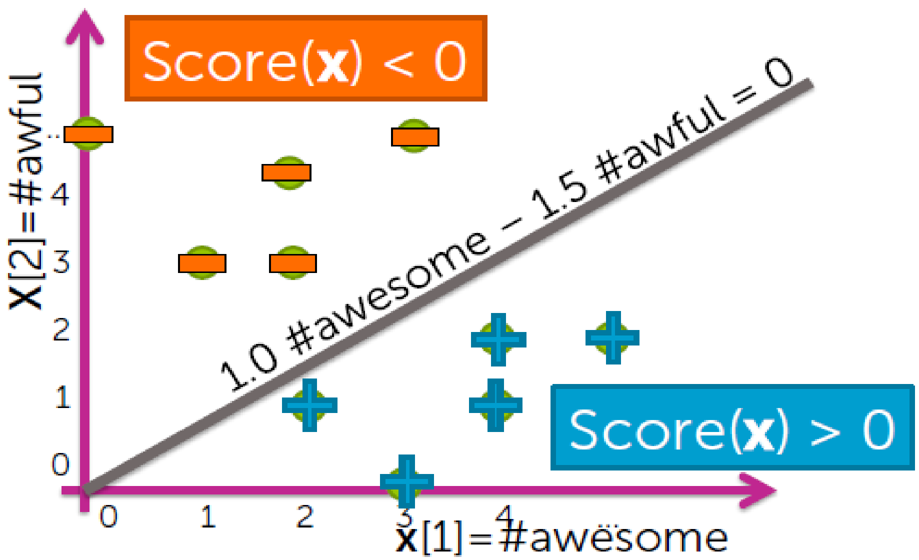
  - Best possible value is 1.0

Overfitting in classification

Decision boundary example

Word	Coefficient
#awesome	1.0
#awful	-1.5

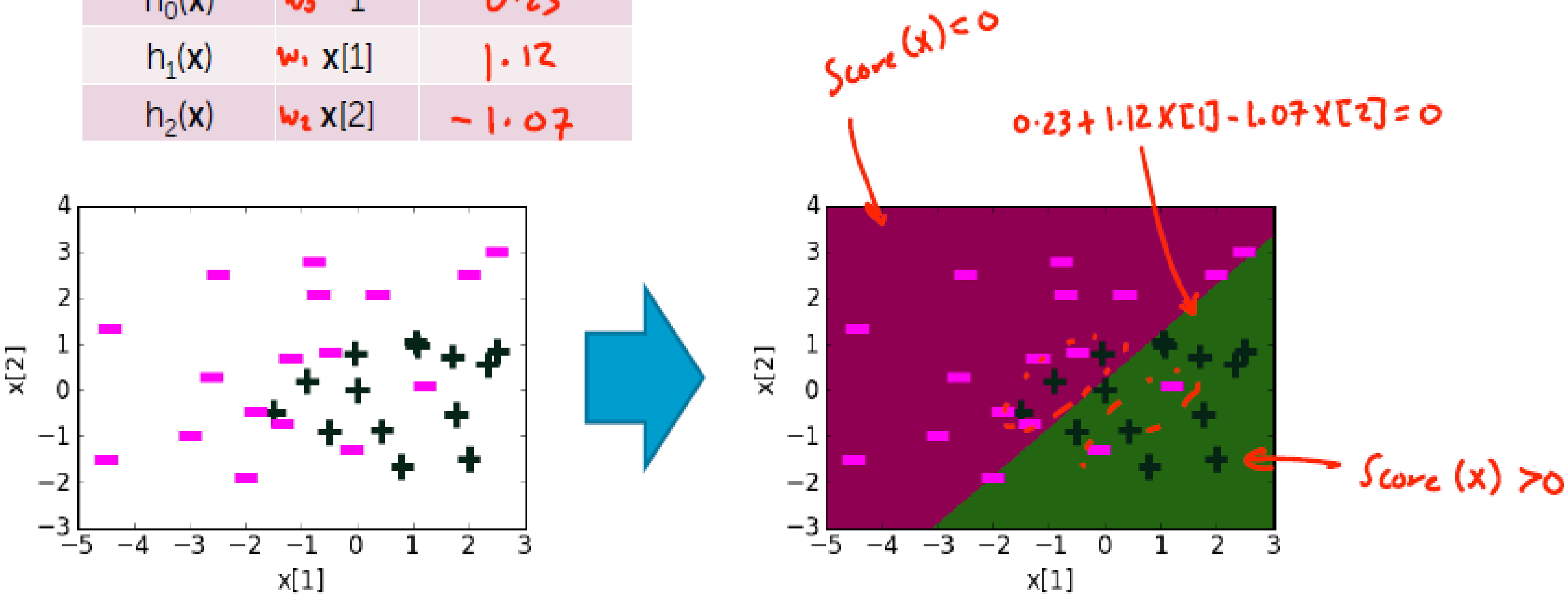
Score(x) = 1.0 #awesome – 1.5 #awful



Overfitting in classification

Learned decision boundary

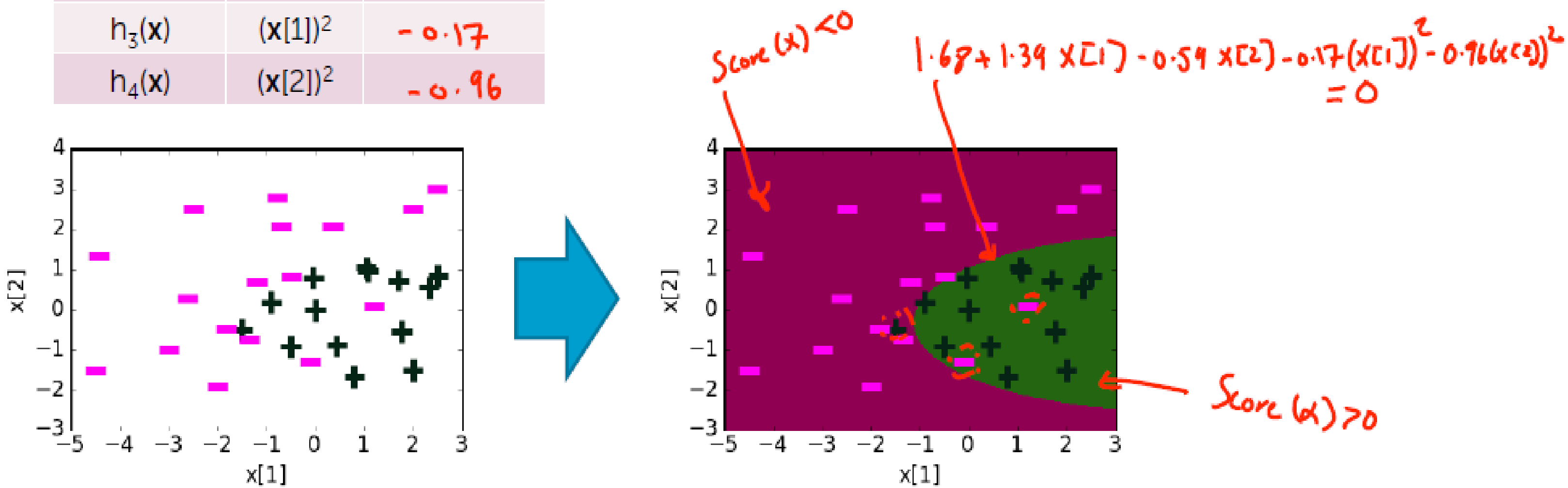
Feature	Value	Coefficient learned
$h_0(\mathbf{x})$	w_0 1	0.23
$h_1(\mathbf{x})$	w_1 x[1]	1.12
$h_2(\mathbf{x})$	w_2 x[2]	-1.07



Overfitting in classification

Quadratic features (in 2d)

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

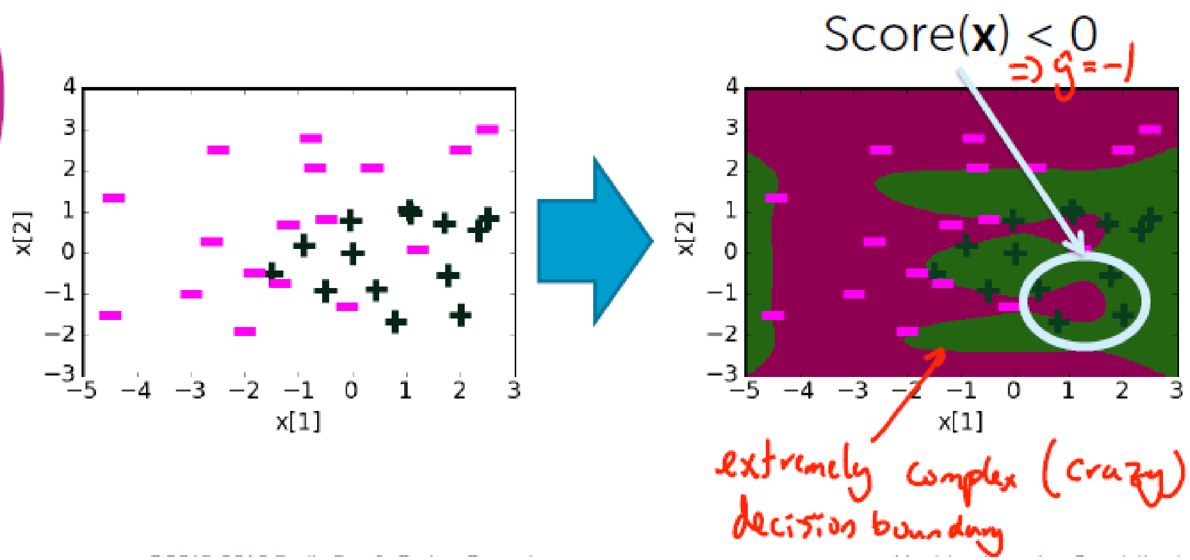


Overfitting in classification

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



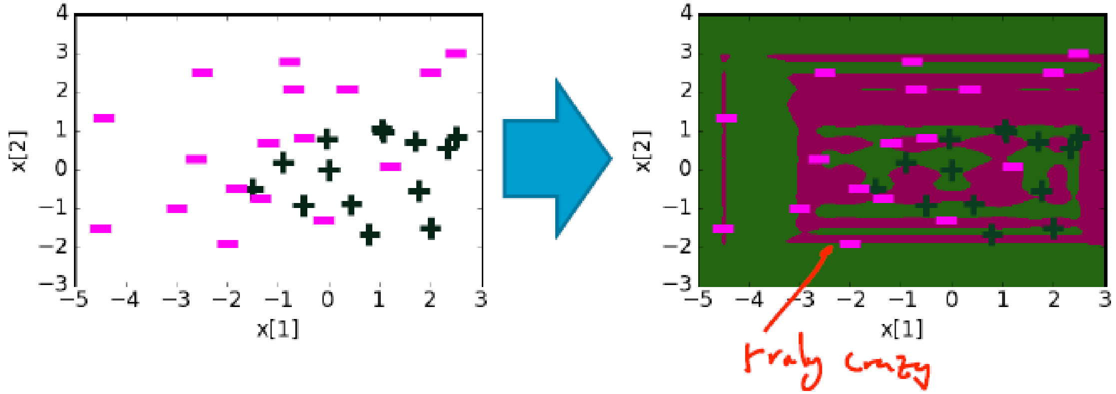
18

Overfitting in classification

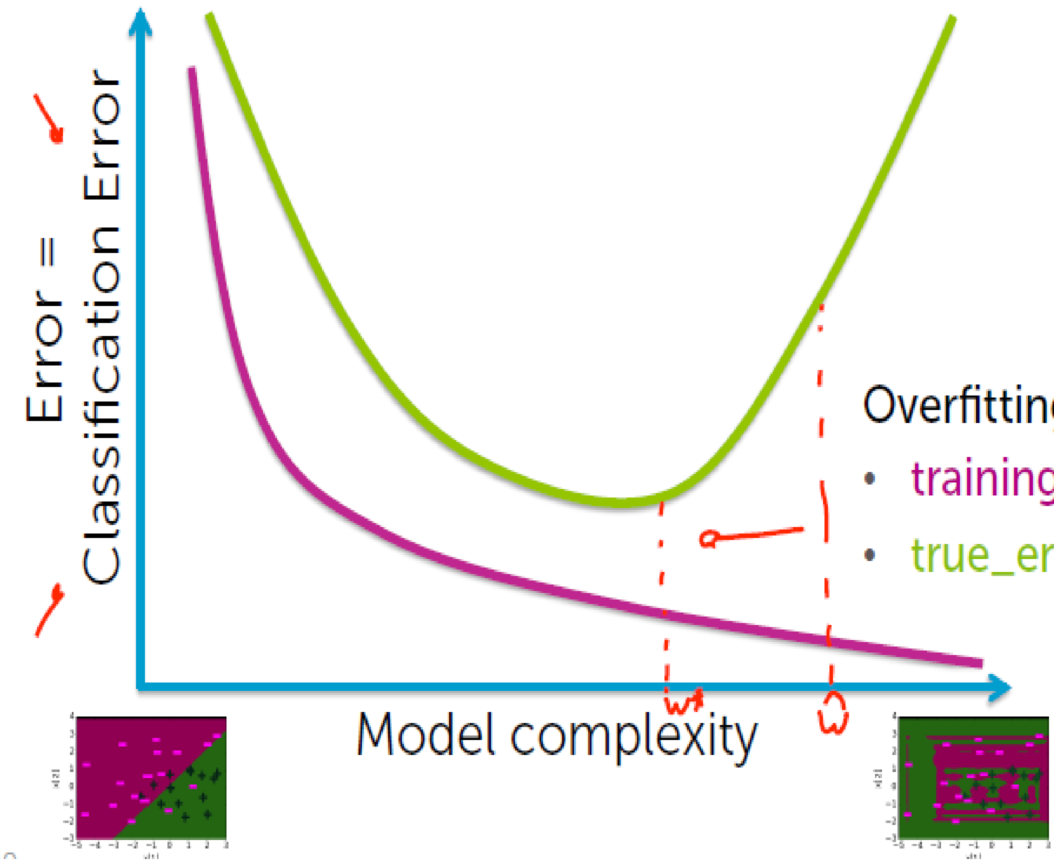
Degree 20 features (in 2d)

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

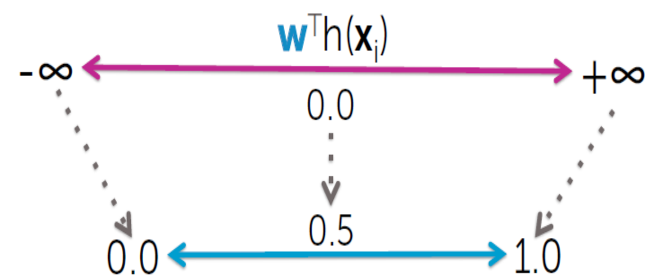
↓

$\hat{w}^T h(\mathbf{x}_i)$ is very positive (or very negative)
→ **sigmoid**($\hat{w}^T h(\mathbf{x}_i)$) 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

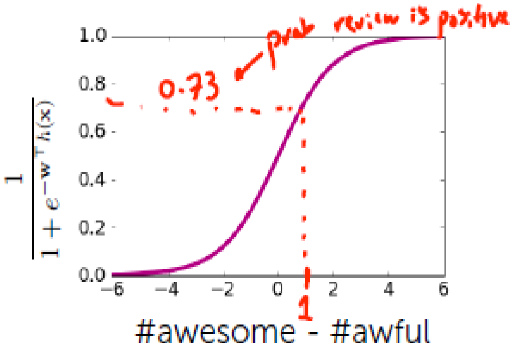
With increasing coefficients model becomes overconfident on predictions

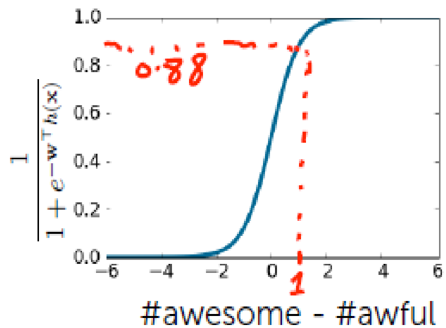
Input x: #awesome=2, #awful=1

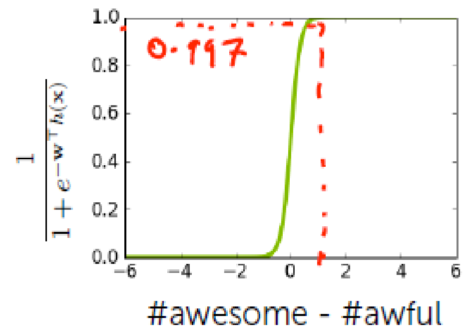
w_0	0
$w_{\#awesome}$	+1
$w_{\#awful}$	-1

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

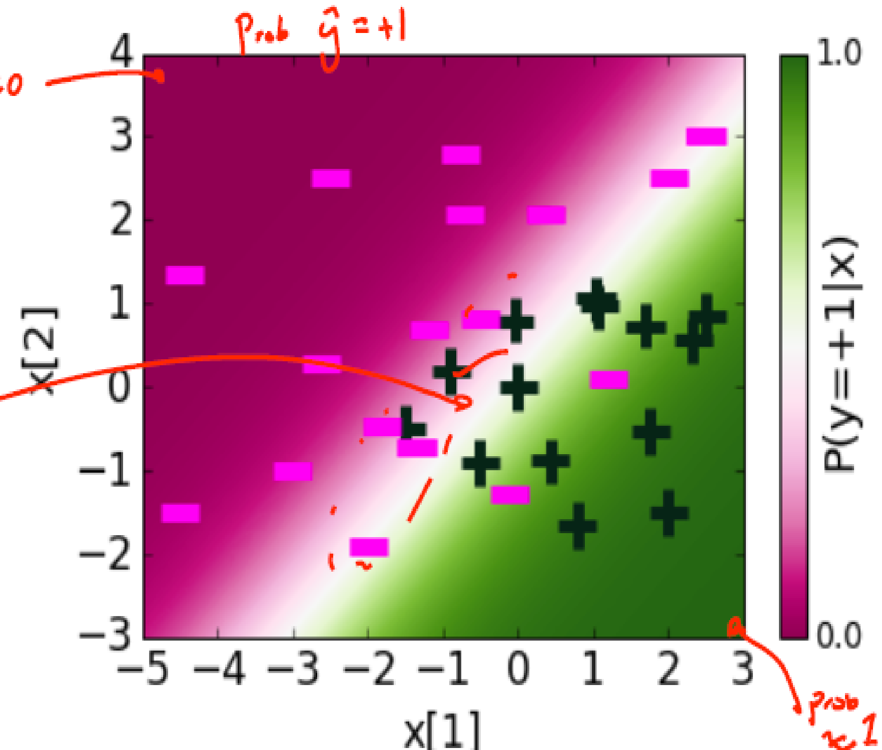
$$P(y = +1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^\top h(\mathbf{x})}}$$

Make sense

wide region of uncertainty

prob ≈ 0.5

prob ≈0



30/10, 6/11 2024

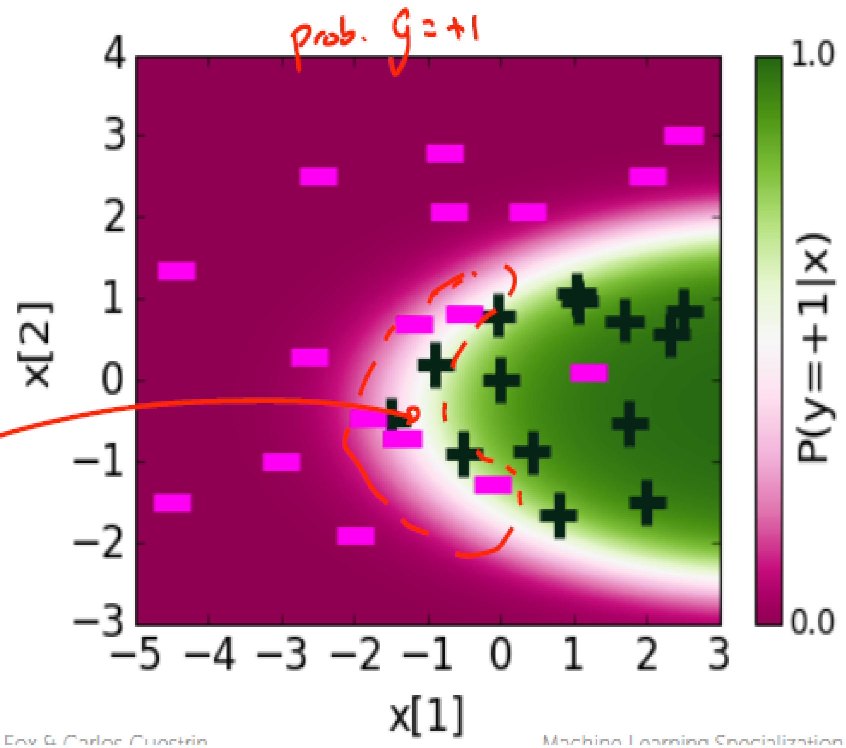
Quadratic features: learned probabilities

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

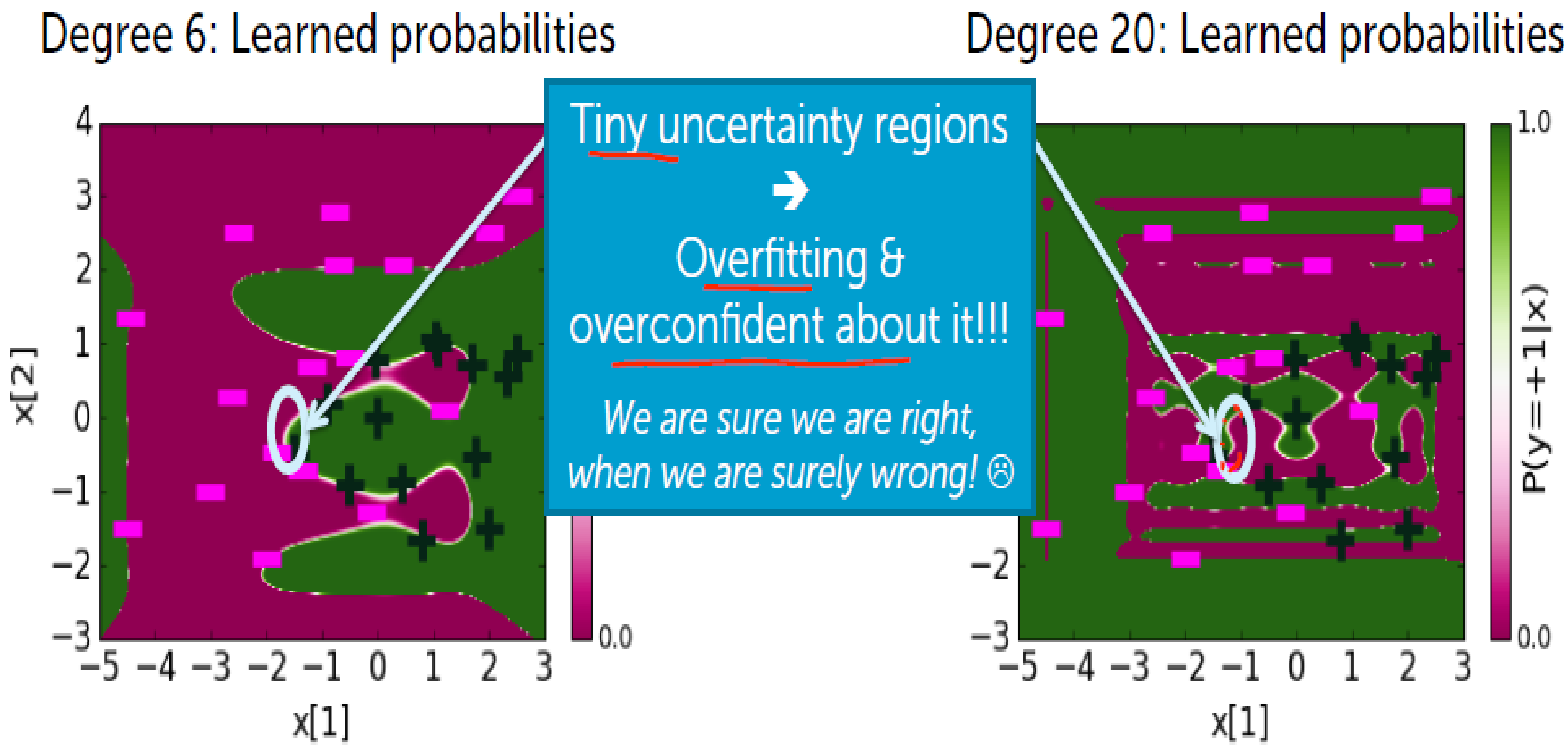
$$P(y=+1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^\top h(\mathbf{x})}}$$

uncertainty region narrower



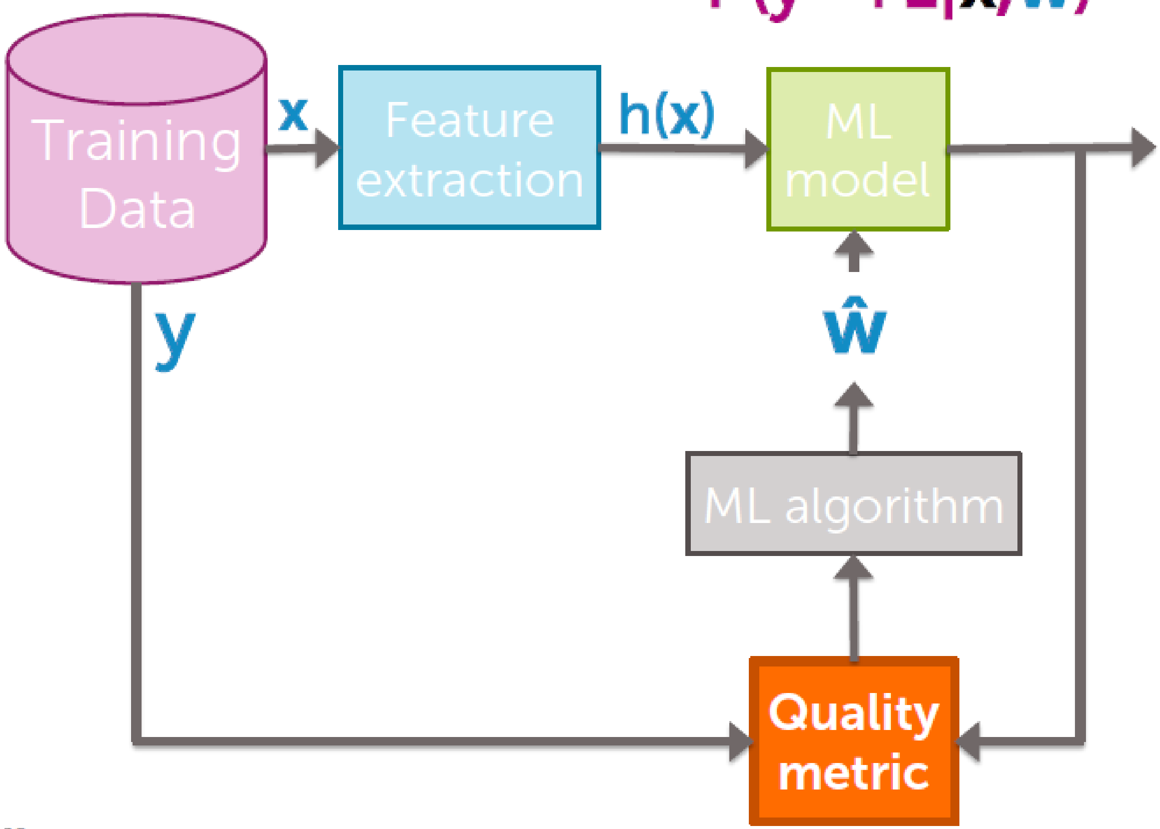
30/10, 6/11 2024

Overfitting → overconfident predictions



Quality metric → penelazing large coefficients

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

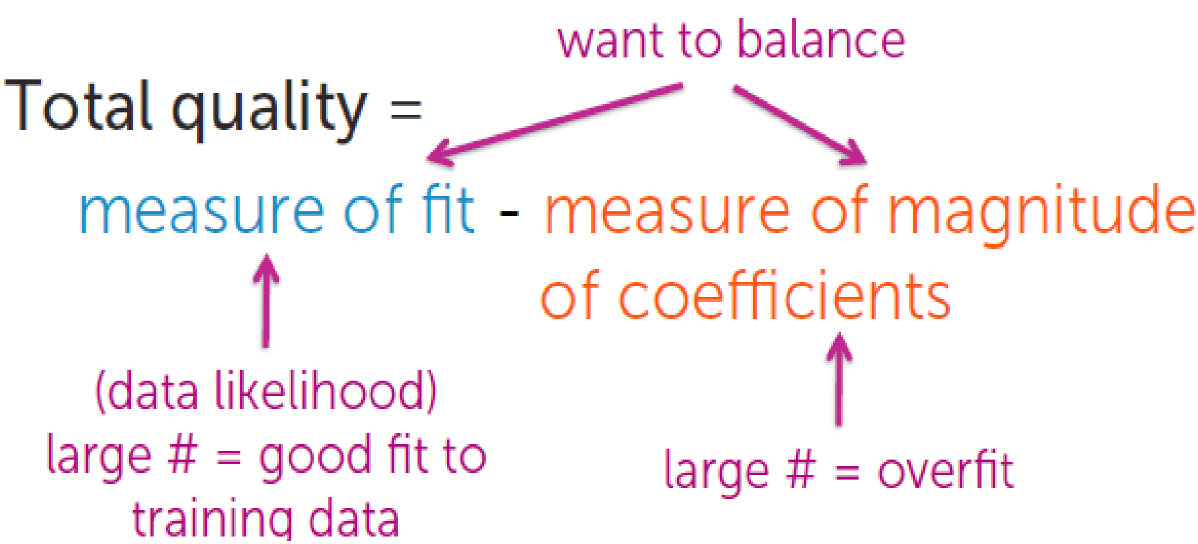


Desired total cost format

Want to balance:

i. How well function fits data

ii. Magnitude of coefficients



Maximum likelihood estimation (MLE)

- Measure of fit = Data likelihood

  - Choose coefficients **w** that maximize likelihood:

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

  - Typically, we use the log of likelihood function
    (simplifies math and has better convergence properties) ⟵ !!!

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

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

Consider specific total cost

$\max_{w}$

Total quality =
measure of fit - measure of magnitude of coefficients

$\ell(\mathbf{w})$ ← log data likelihood

$||\mathbf{w}||_2^2$ ← L_2 penalty

Consider resulting objectives

What if $\hat{\mathbf{w}}$ selected to minimize

$$\ell(\mathbf{w}) - \lambda||\mathbf{w}||_2^2$$

tuning parameter = balance of fit and magnitude

If λ=0:

Reduces $\max_w \ell(w)$ → Standard (unpenalized) MLE solution

If $\lambda=\infty$:

$\max_w \ell(w) - \infty ||w||_2^2$ → only care about penalizing w, large coefficients → $w=0$

If λ in between:

Balance data fit against the magnitude of the coefficients

 Machine Learning Speci...

30/10, 6/11 2024

Consider resulting objectives

What if $\hat{\mathbf{w}}$ selected to minimize

$$\ell(\mathbf{w}) - \lambda||\mathbf{w}||_2^2$$

tuning parameter = balance of fit and magnitude

L_2 regularized logistic regression

Pick λ using:

- Validation set (for large datasets)
- Cross-validation (for smaller datasets)

Bias-variance tradeoff

Large λ:

high bias, low variance

(e.g., $\hat{\mathbf{w}} = 0$ for $\lambda = \infty$)

In essence, λ controls model complexity

Small λ:

low bias, high variance

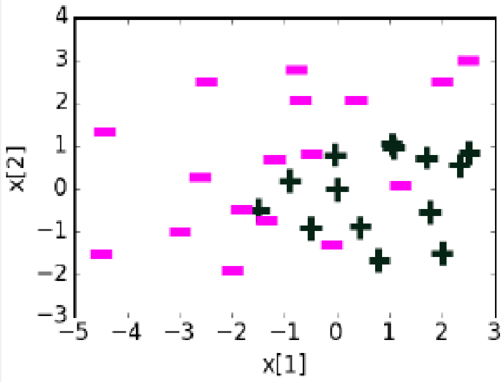
(e.g., maximum likelihood (MLE) fit of high-order polynomial for $\lambda = 0$)

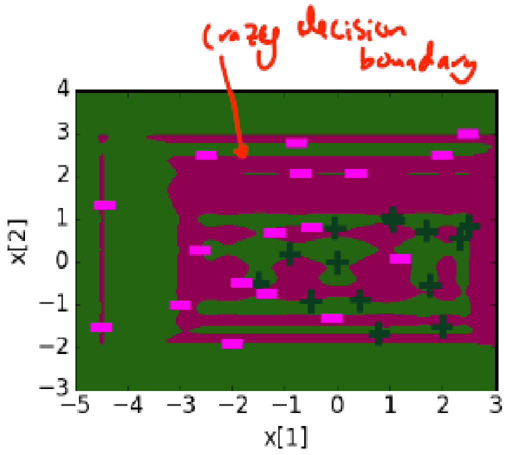
Visualizing effect of regularisation

Degree 20 features, λ=0

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

Coefficients range from -3170 to 3803



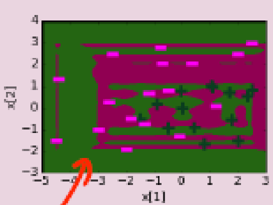
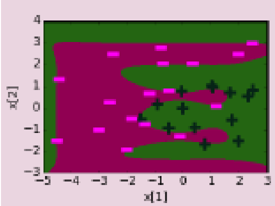
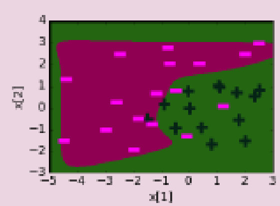
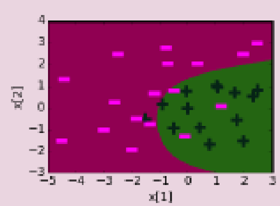
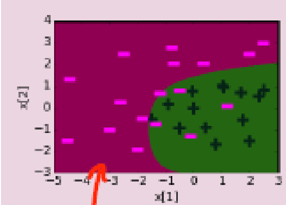


Visualizing effect of regularisation

Degree 20 features,
effect of regularization penalty λ

very large

smaller coefficients

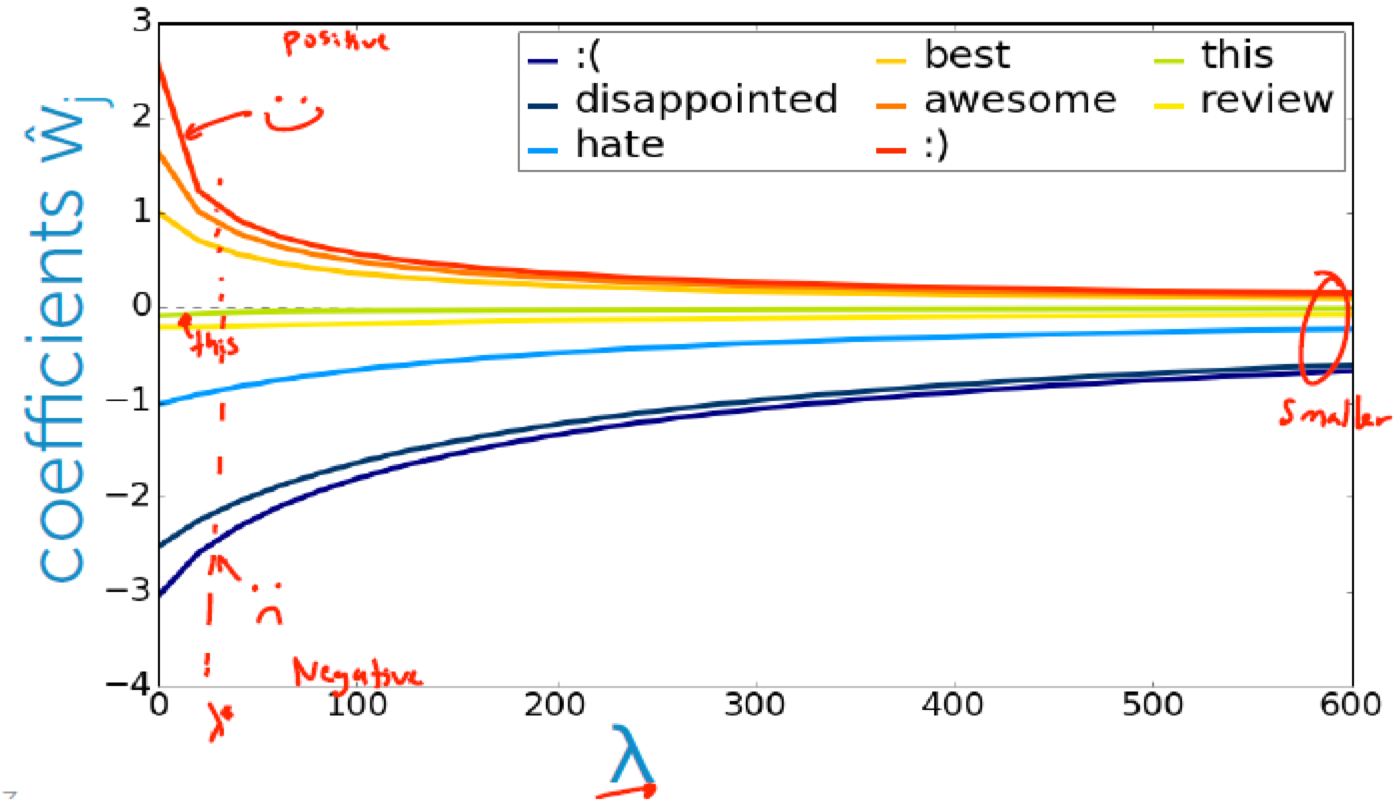
Regularization	λ = 0	λ = 0.00001	λ = 0.001	λ = 1	λ = 10
Range of coefficients	-3170 to 3803	-8.04 to 12.14	-0.70 to 1.25	-0.13 to 0.57	-0.05 to 0.22
Decision boundary					

crazy decision boundary

nicer smoother

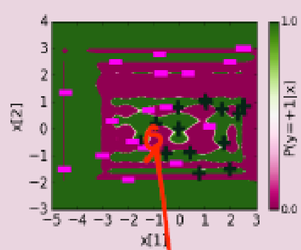
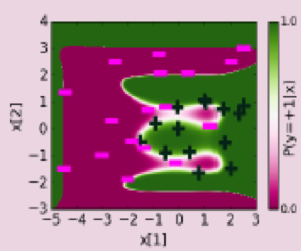
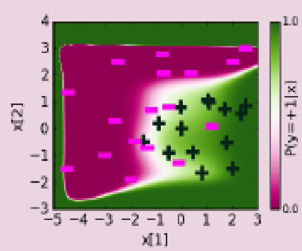
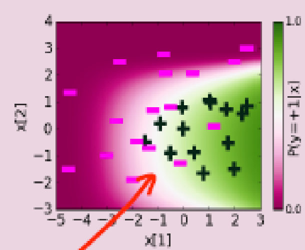
Effect of regularisation

Coefficient path



Visualizing effect of regularisation

Degree 20 features:
regularization reduces “overconfidence”

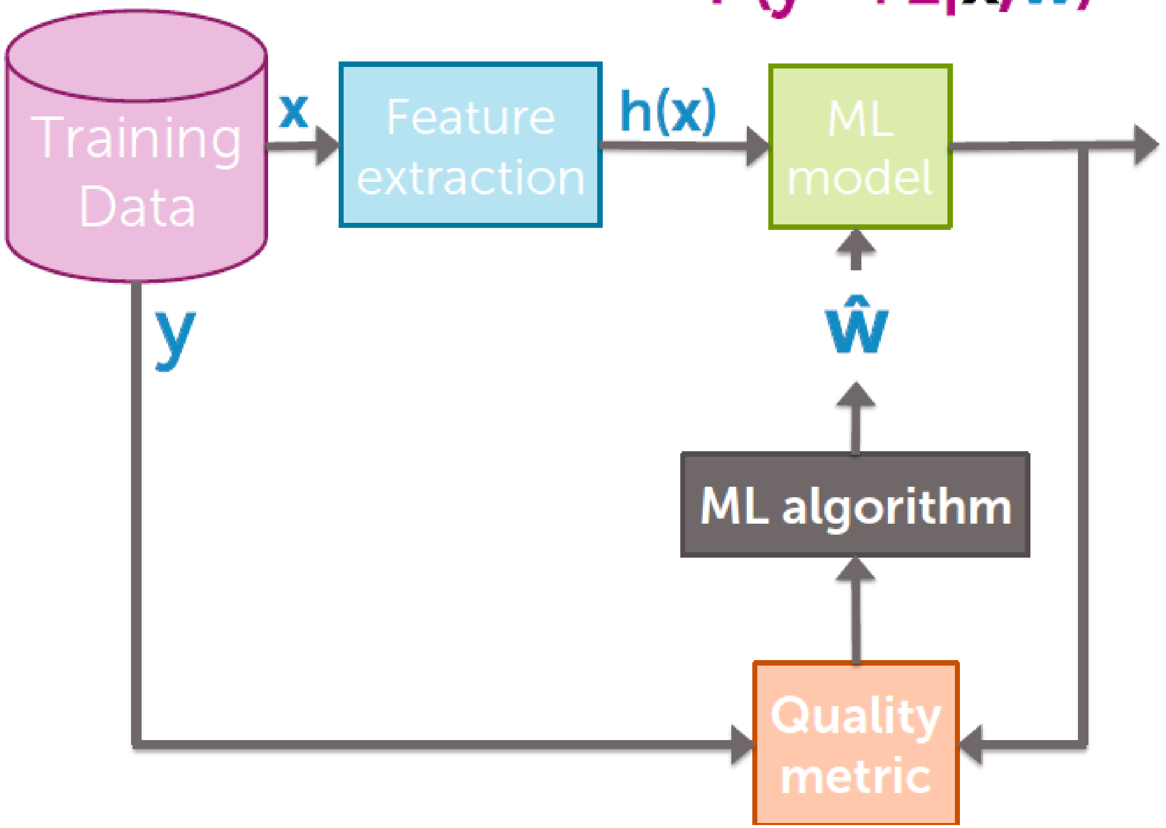
Regularization	$\lambda = 0$	$\lambda = 0.00001$	$\lambda = 0.001$	$\lambda = 1$
Range of coefficients	-3170 to 3803	-8.04 to 12.14	-0.70 to 1.25	-0.13 to 0.57
Learned probabilities				

highly over confident

Very natural uncertainty region

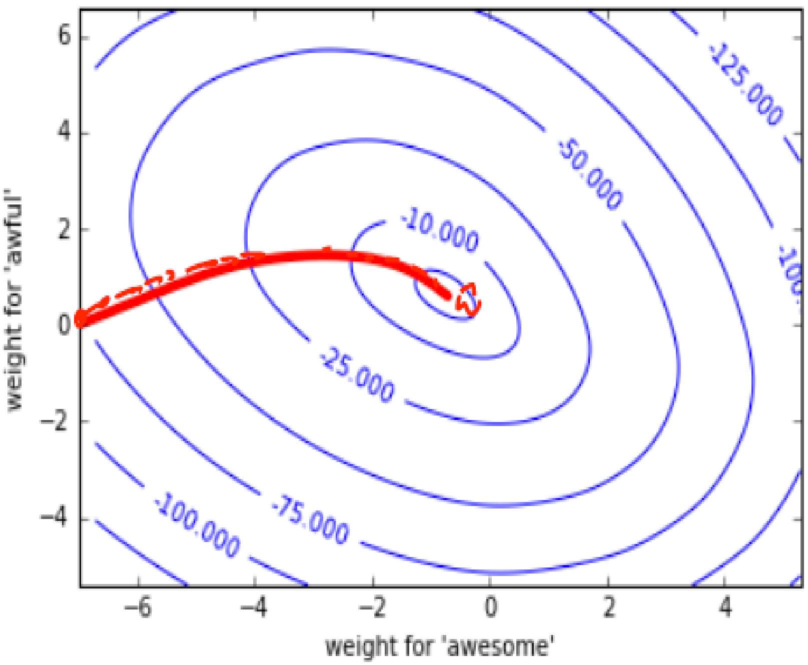
Flow chart: ML algorithm

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



Lets discuss now finding best L2-regularized linear classifier with gradient ascent

Gradient ascent



Algorithm:

while not converged

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \nabla \ell(\mathbf{w}^{(t)})$$

need the gradient of regularized log likelihood

Gradient of L2 regularized log-likelihood

Total quality =
measure of fit - measure of magnitude of coefficients

$\ell(\mathbf{w})$ $\lambda \|\mathbf{w}\|_2^2$

$$\text{Total derivative} = \frac{\partial \ell(\mathbf{w})}{\partial \mathbf{w}_j} - \lambda \frac{\partial \|\mathbf{w}\|_2^2}{\partial \mathbf{w}_j}$$

Gradient of L2 regularized log-likelihood

Derivative of (log-)likelihood

Sum over data points

Feature value

Difference between truth and prediction

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

Derivative of L_2 penalty

$$\frac{\partial ||\mathbf{w}||_2^2}{\partial \mathbf{w}_j} = \frac{\partial}{\partial w_j}\left[w_0^2 + w_1^2 + w_2^2 + \dots + w_j^2 + \dots + w_D^2\right] = 2 w_j$$

Gradient of L2 regularized log-likelihood

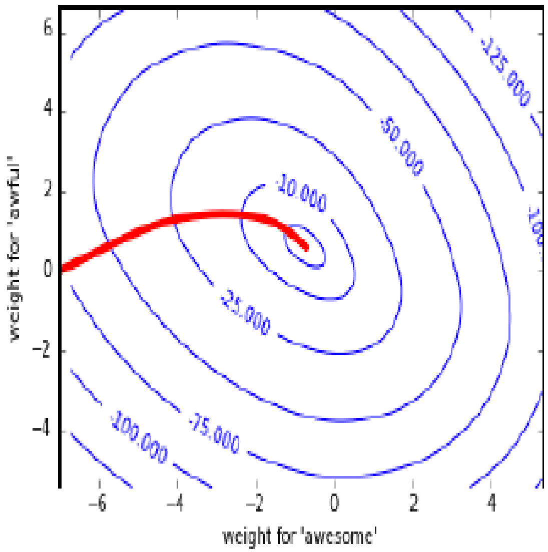
Understanding contribution of L_2 regularization

Term from L_2 penalty

$$\frac{\partial \ell(\mathbf{w})}{\partial \mathbf{w}_j} \overbrace{- 2\lambda \mathbf{w}_j}$$

	$-2\lambda w_j$	Impact on w_j
$w_j > 0$	< 0	decreases w_j => w_j becomes closer to 0
$w_j < 0$	> 0	increases w_j => w_j becomes closer to 0

Gradient ascent with L2 regularization



init $\mathbf{w}^{(1)}=0$ (or randomly, or smartly), $t=1$

while not converged:

for $j=0,\ldots,D$

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

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta\,(\text{partial}[j] - 2\lambda\, w_j^{(t)})$$

$t \leftarrow t+1$

step size

$\frac{\partial \ell(w)}{\partial w_j}$

only change !!

Logistic regression with L1 regularization

Recall sparsity (many $\hat{w}_j$=0) gives efficiency and interpretability

Efficiency:

- If size(**w**) = 100B, each prediction is expensive
- If $\hat{\mathbf{w}}$ sparse, computation only depends on # of non-zeros

many zeros

$$\hat{y}_i = sign\left(\sum_{\hat{\mathbf{w}}_j \neq 0} \hat{\mathbf{w}}_j h_j(\mathbf{x}_i)\right)$$

Interpretability:

- Which features are relevant for prediction?

Sparse logistic regression

Total quality =

measure of fit - measure of magnitude of coefficients

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

L_1 regularized logistic regression

Leads to **sparse** solutions!

L1 regularised logistic regression

Just like L2 regularization, solution is governed by a continuous parameter λ

$$\ell(\mathbf{w}) - \lambda\|\mathbf{w}\|_1$$

tuning parameter = balance of fit and sparsity

If λ=0:

No regularization → standard MLE solution

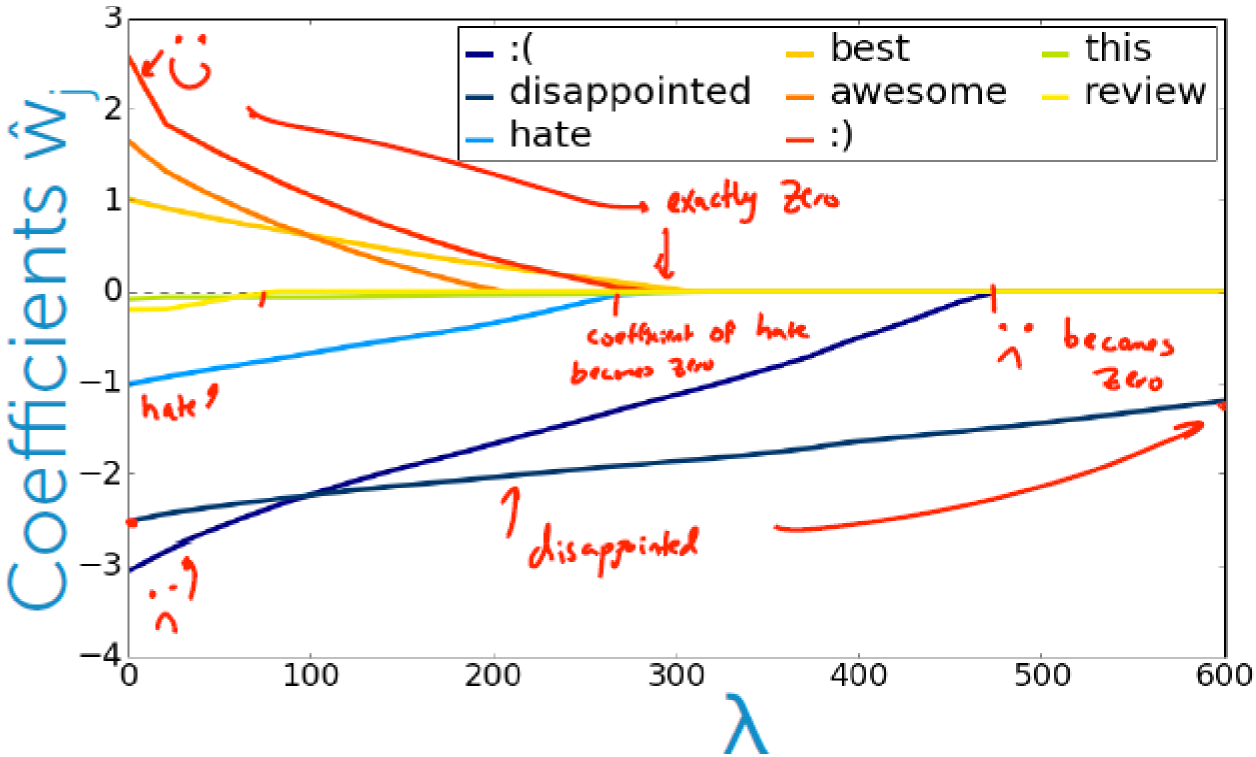
If $\lambda=\infty$:

all weight is on regularization → $\hat{w}=0$

If λ in between:

Sparse solutions: some $\hat{w}_j \neq 0$, many other $\hat{w}_j = 0$

L1 regularised logistic regression

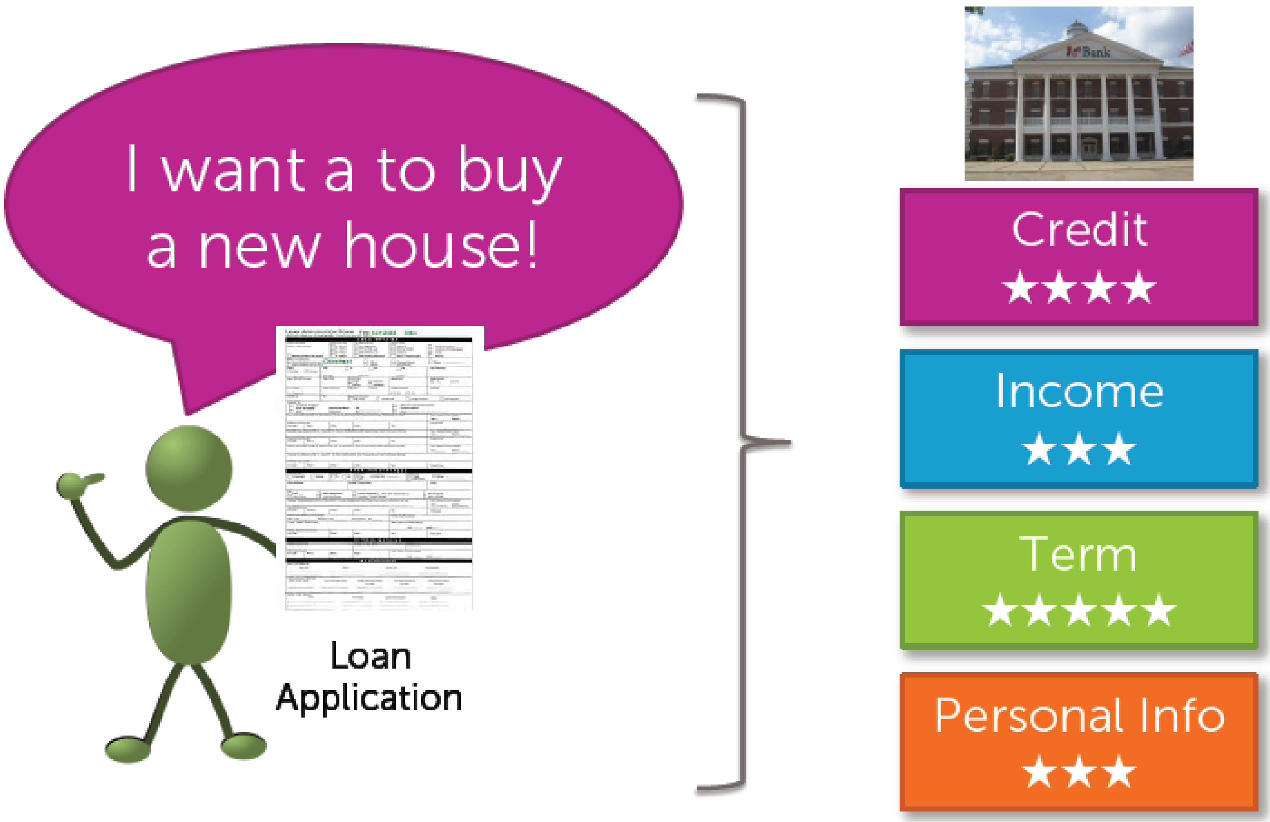


What you can do now…

- Identify when overfitting is happening
- Relate large learned coefficients to overfitting
- Describe the impact of overfitting on decision boundaries and predicted probabilities of linear classifiers
- Motivate the form of L_2 regularized logistic regression quality metric
- Describe what happens to estimated coefficients as tuning parameter λ is varied
- Interpret coefficient path plot
- Estimate L_2 regularized logistic regression coefficients using gradient ascent
- Describe the use of L_1 regularization to obtain sparse logistic regression solutions

Decision trees

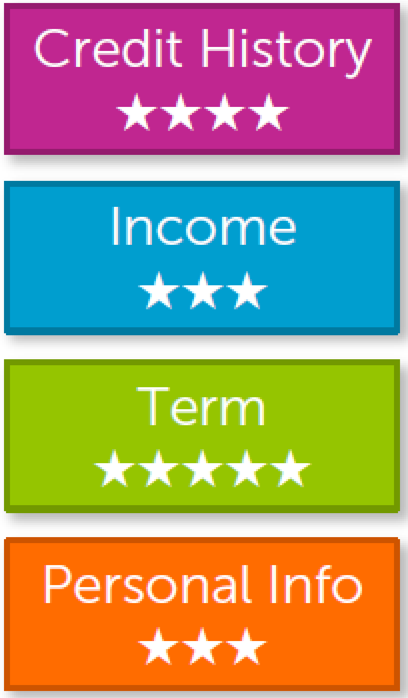
What makes a loan risky?



Credit history explained

Did I pay previous loans on time?

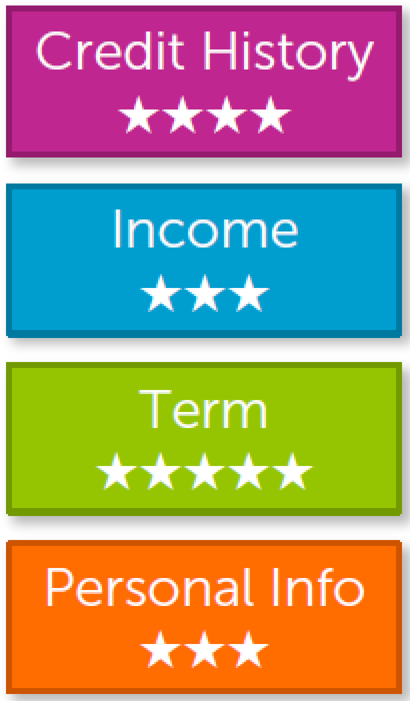
Example: excellent, good, or fair



Income

What's my income?

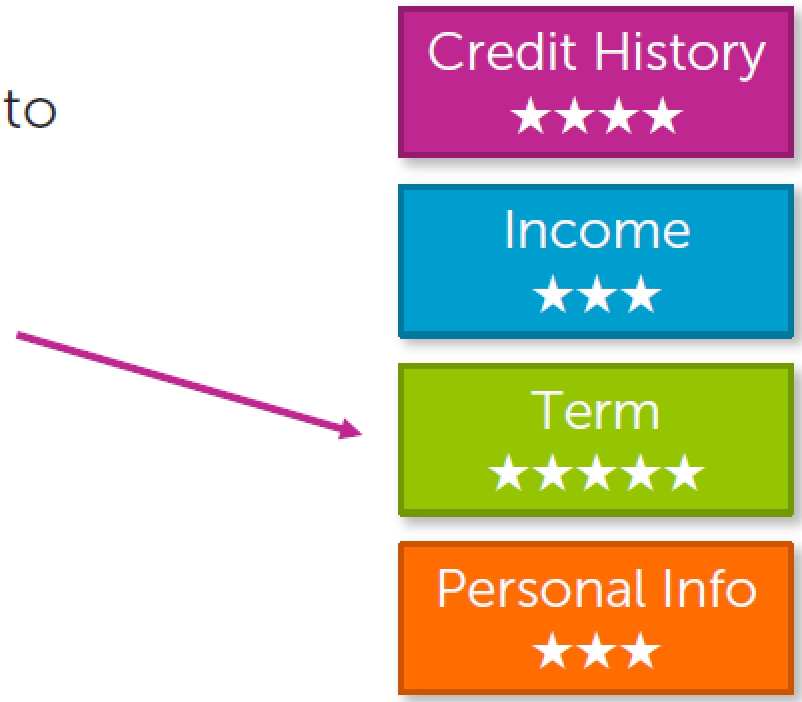
Example:
$80K per year



Loan terms

How soon do I need to pay the loan?

Example: 3 years, 5 years,...

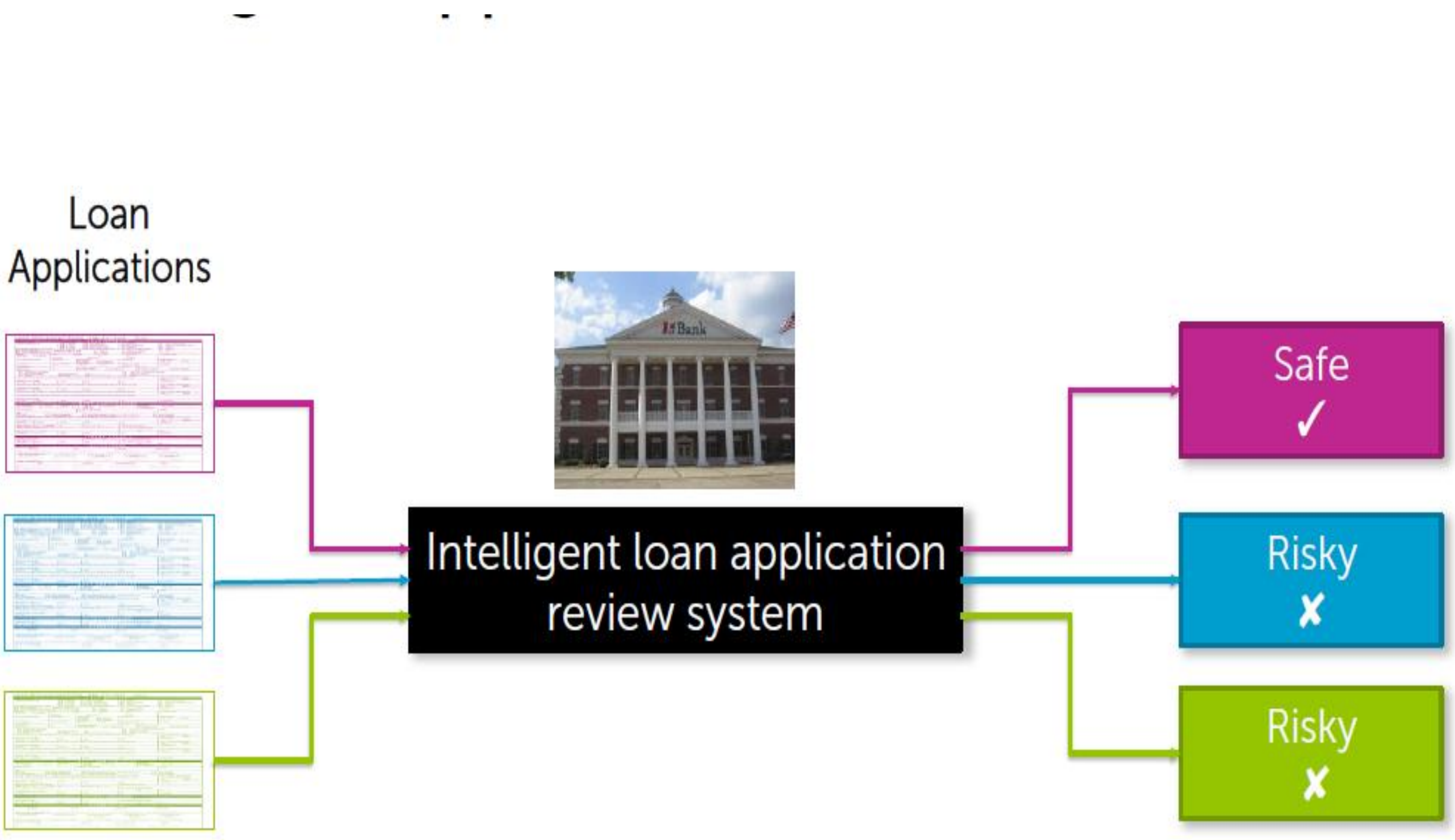


Personal information

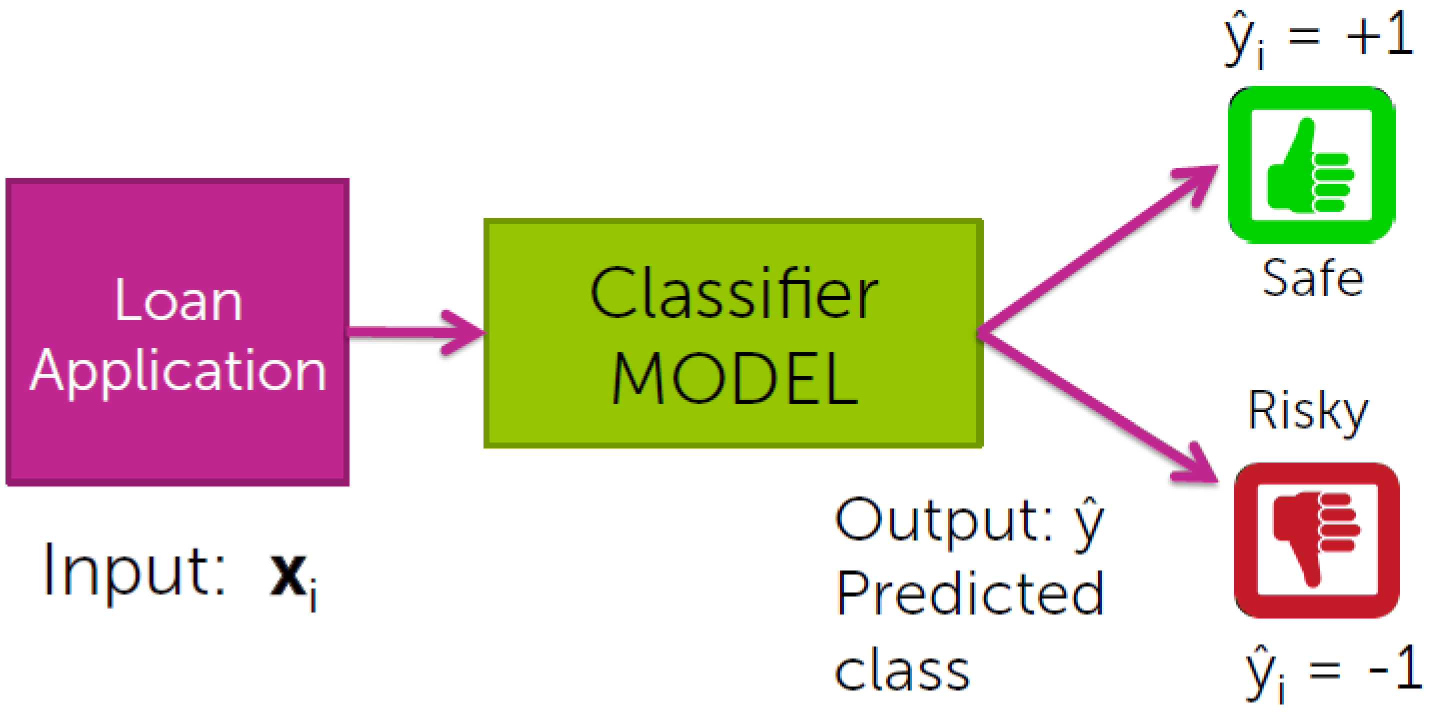
Age, reason for the loan, marital status,...

Example: Home loan for a married couple

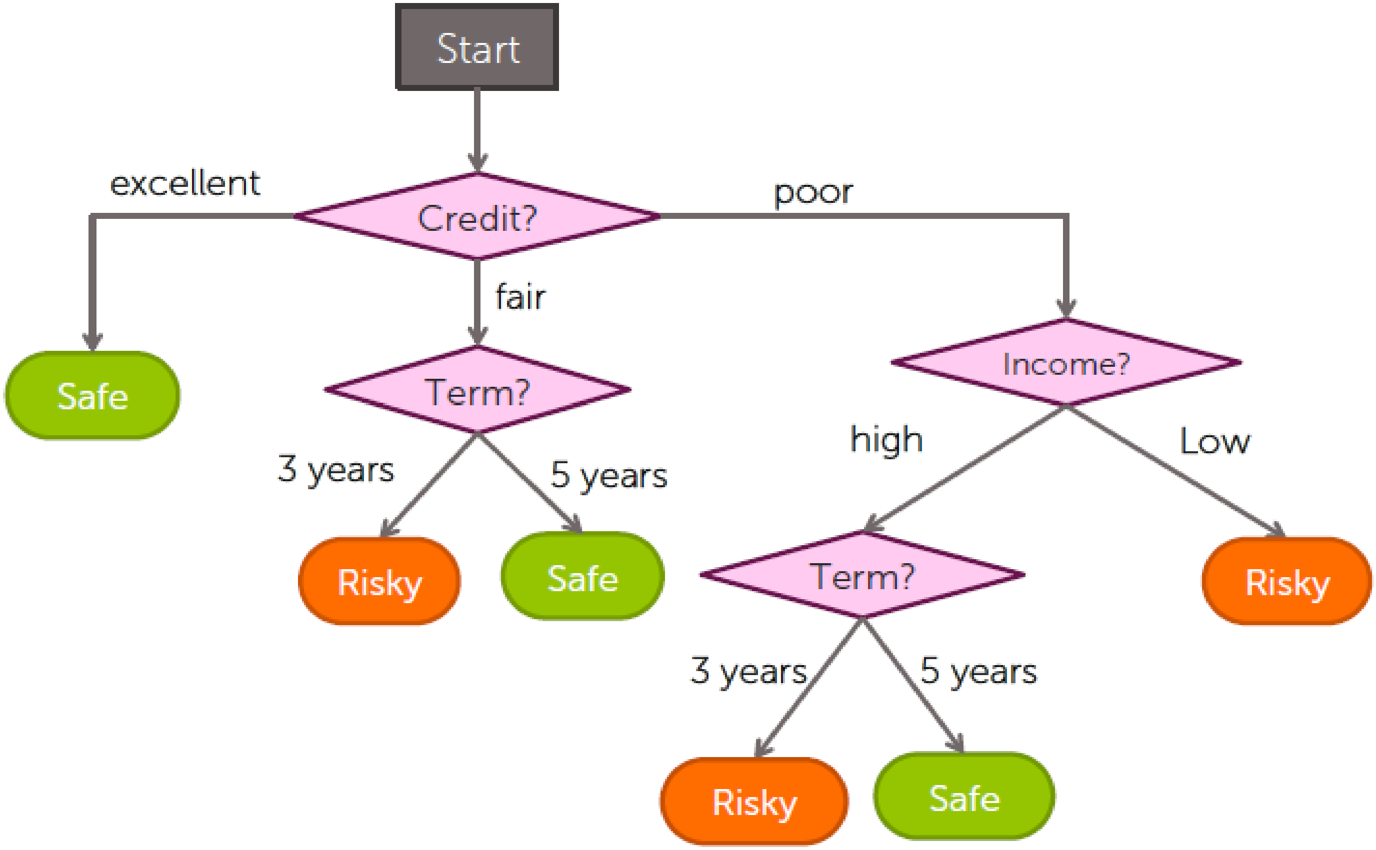
Inteligent application



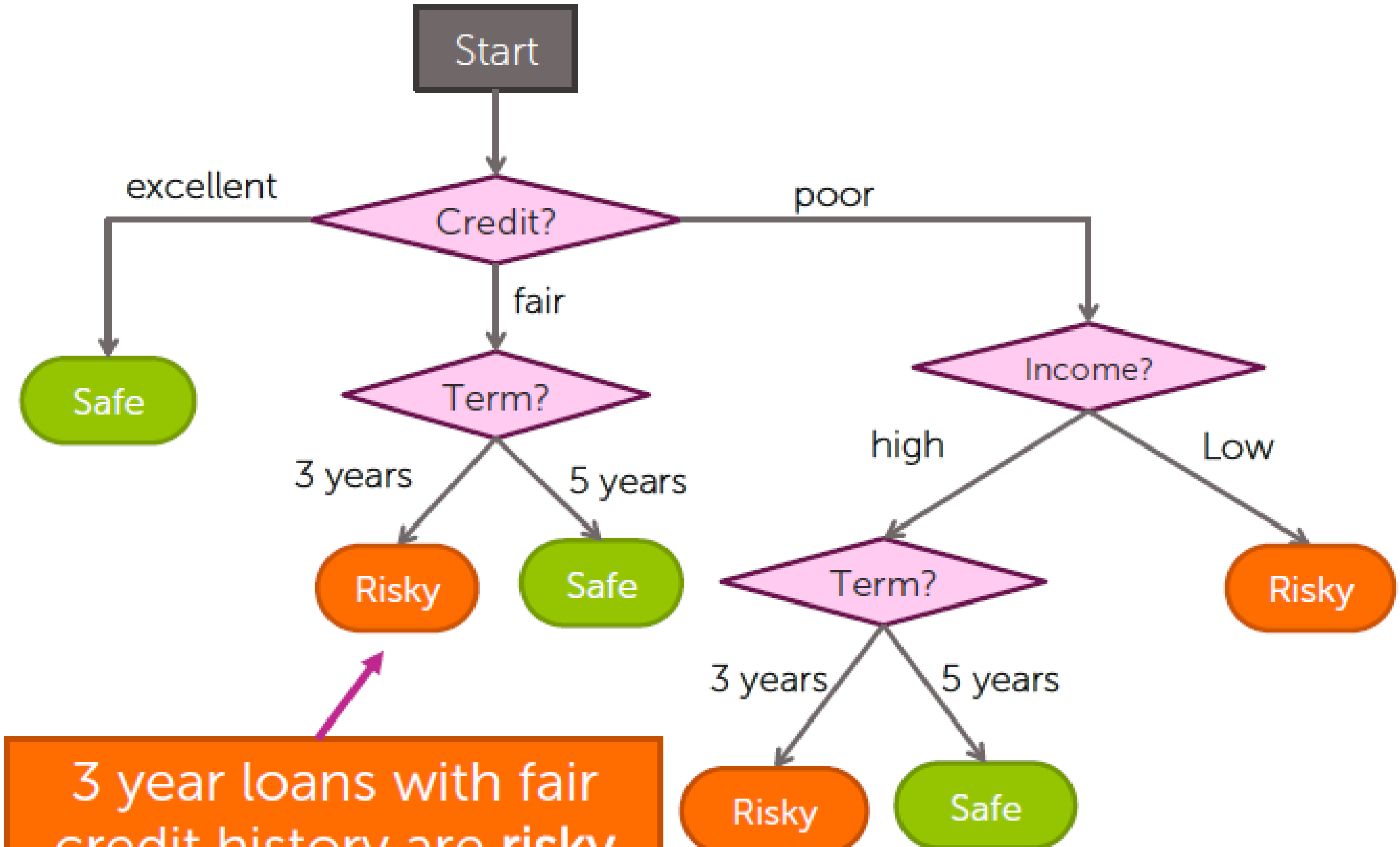
Classifier: review type



Classifier: decision trees

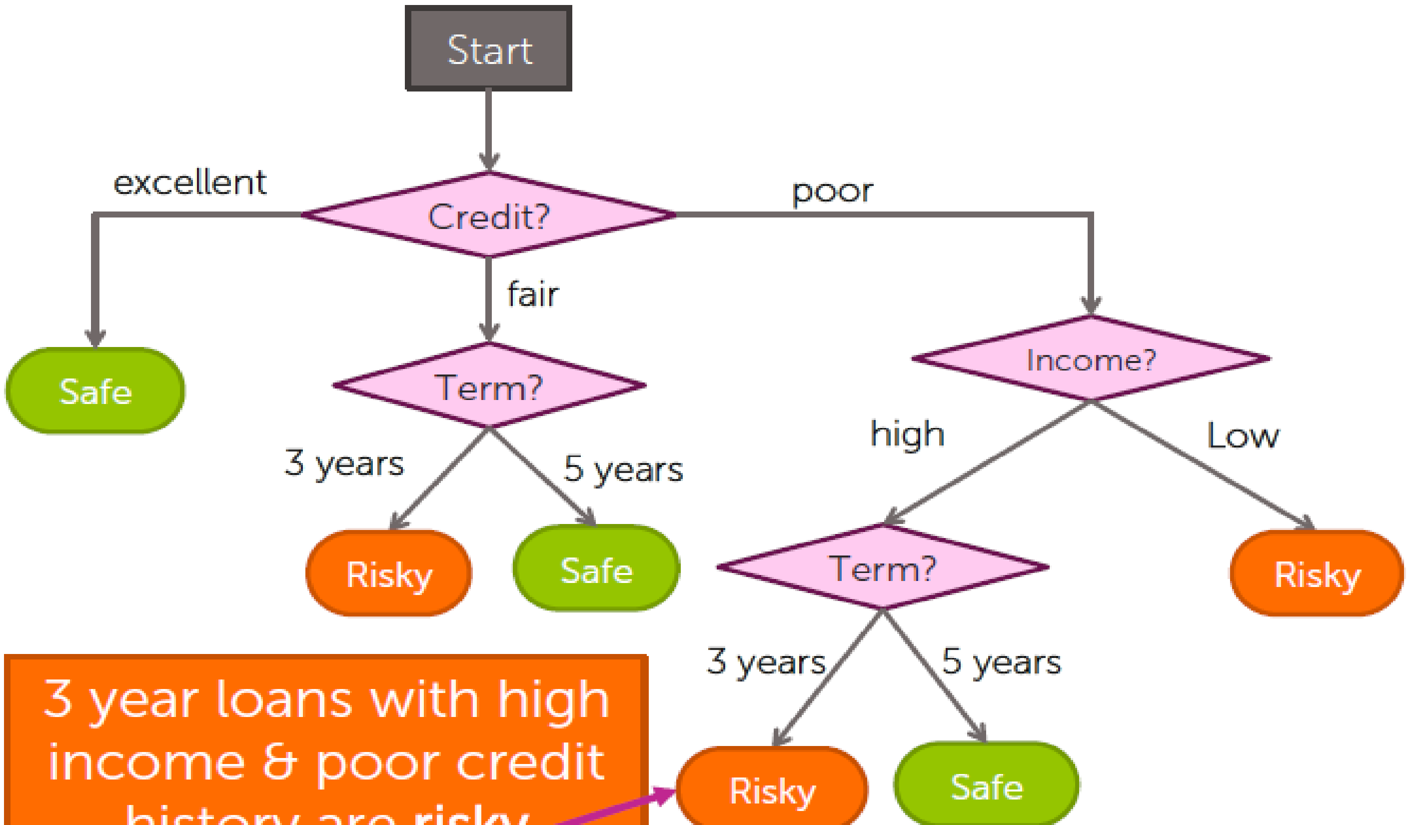


Scoring a loan application



3 year loans with fair credit history are **risky**

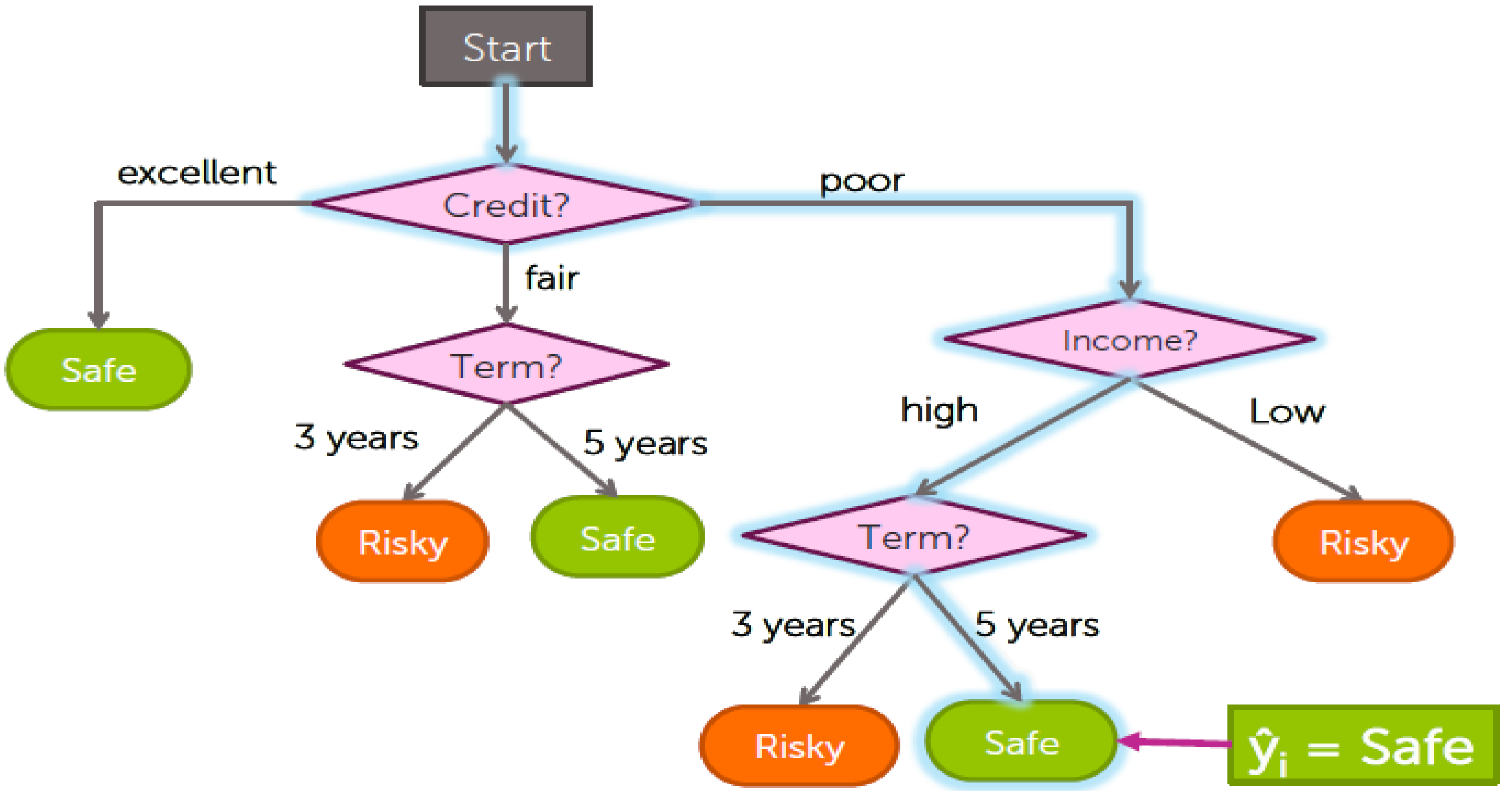
Scoring a loan application



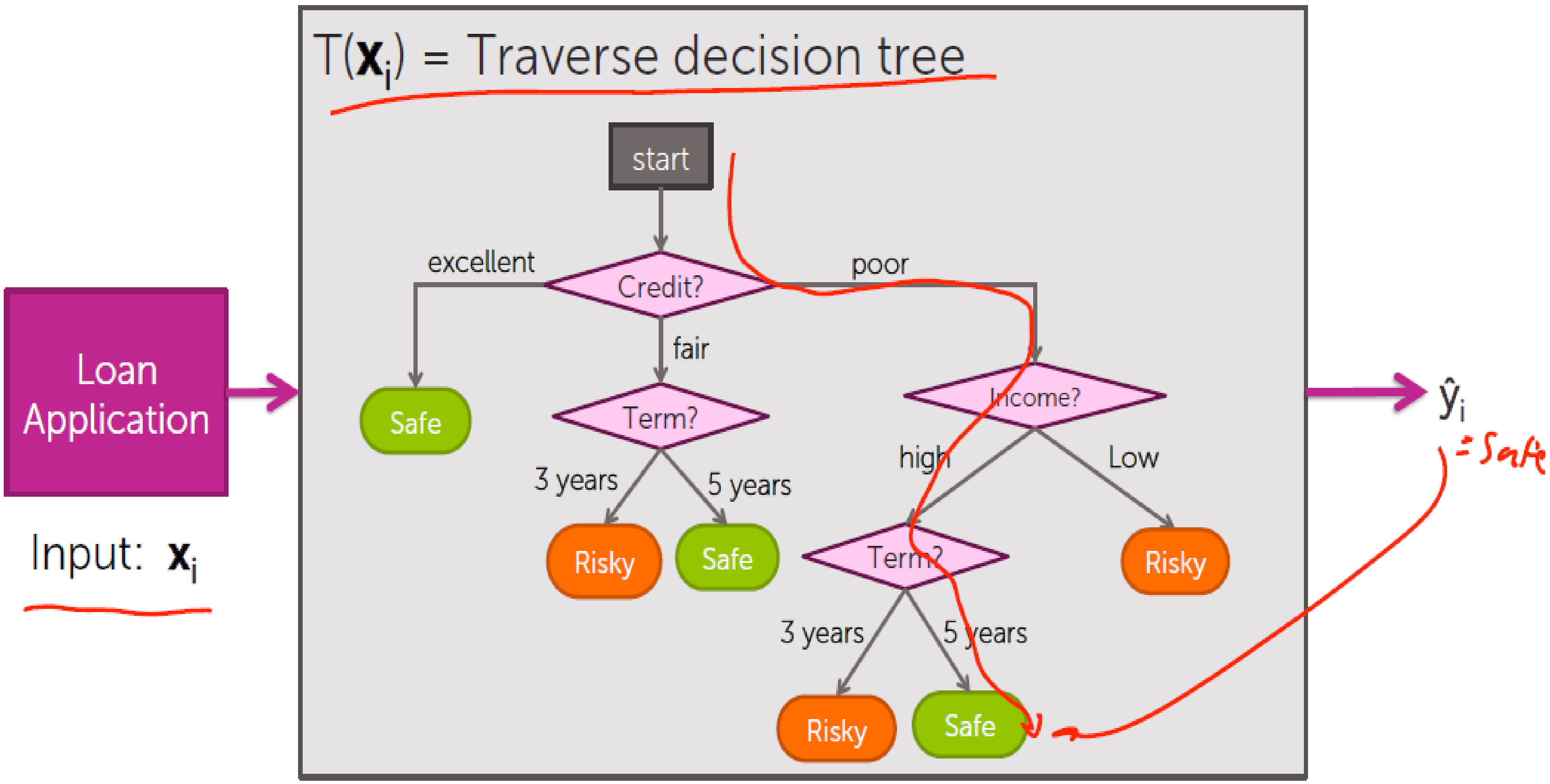
3 year loans with high income & poor credit history are **risky**

Scoring a loan application

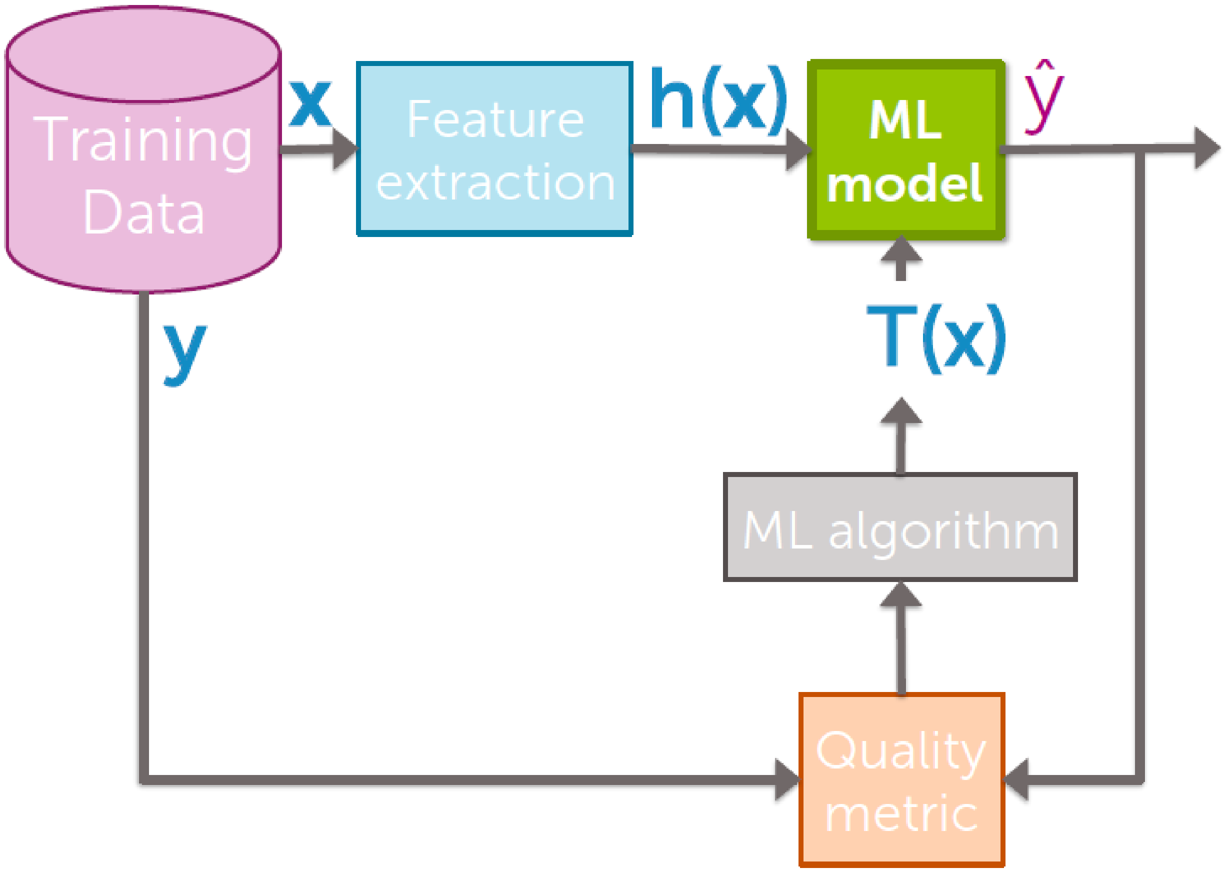
$\mathbf{x}_i$ = (Credit = poor, Income = high, Term = 5 years)



Decision tree model



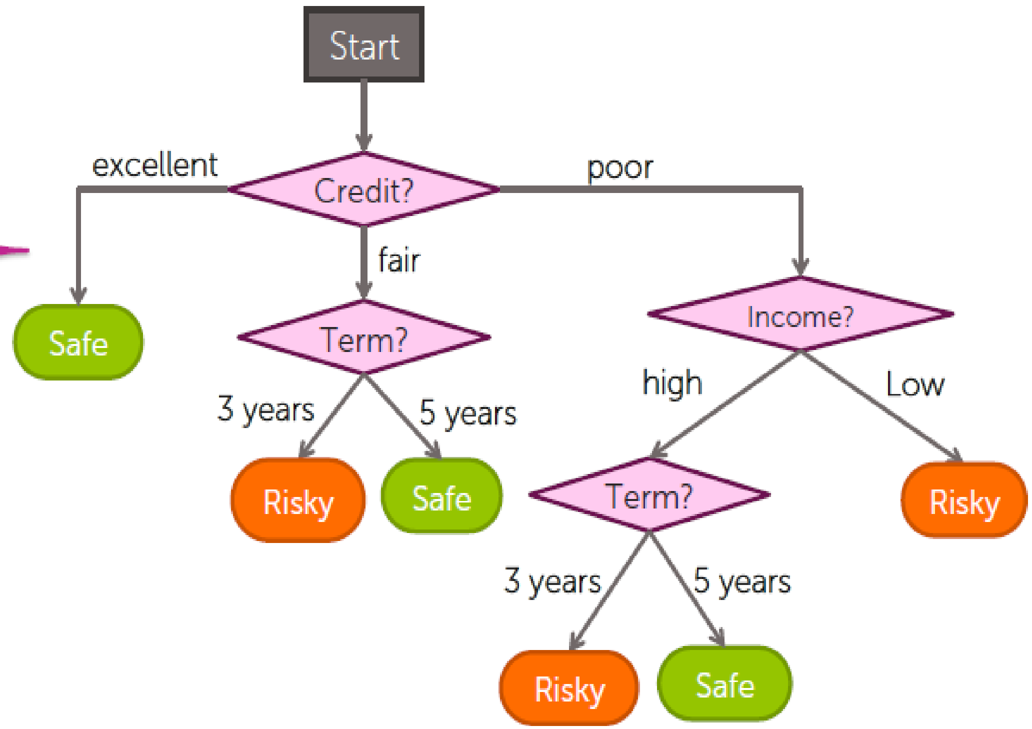
Flow chart: 
ML
model


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

Learn decision tree from data

$h_1(x)$ $h_2(x)$ $h_3(x)$ Loan Status

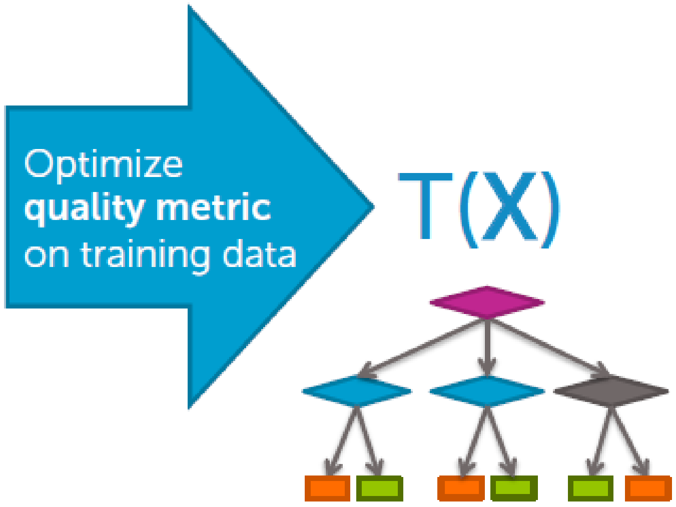
Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	5 yrs	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	low	safe
poor	3 yrs	high	risky
poor	5 yrs	low	safe
fair	3 yrs	high	safe



Learn decision tree from data

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

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



Quality metric: Classification error

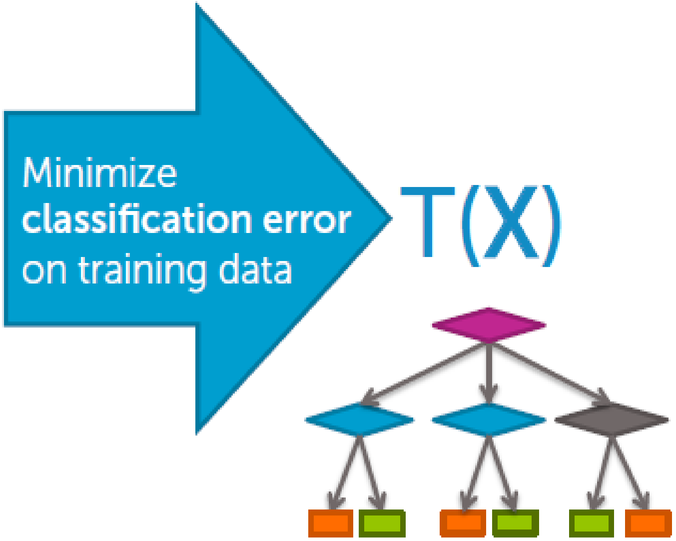
- Error measures fraction of mistakes

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

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

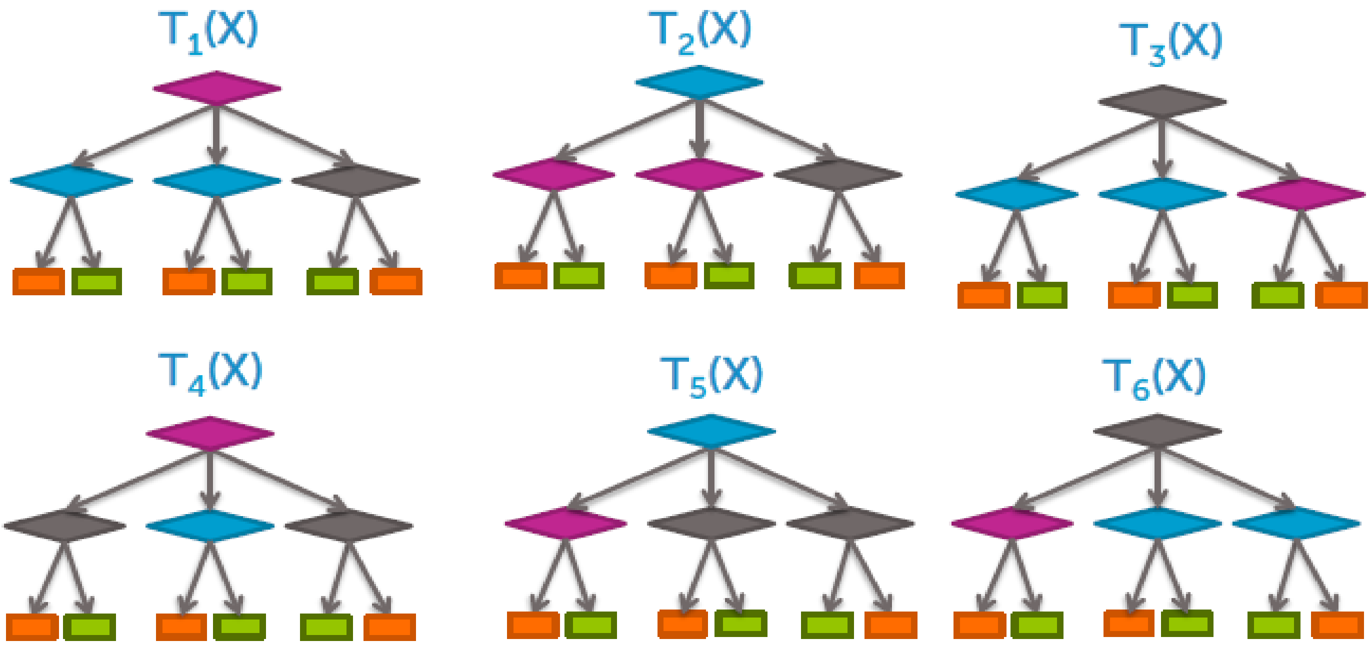
Find the tree with lowest classification error

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



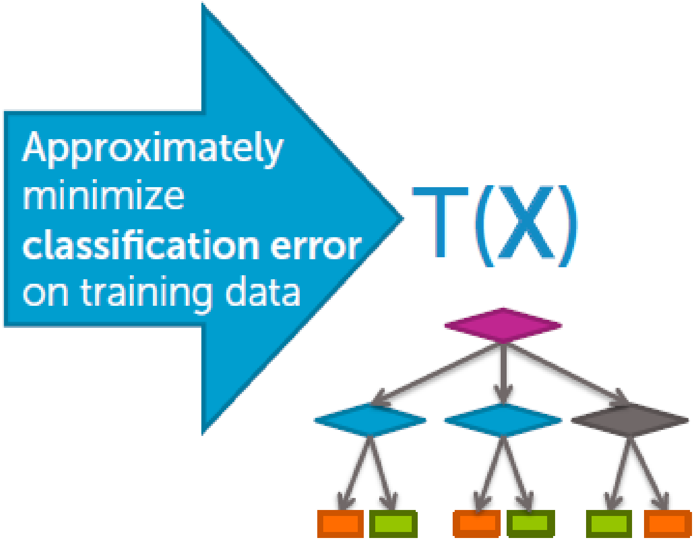
How do we find the best tree?

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



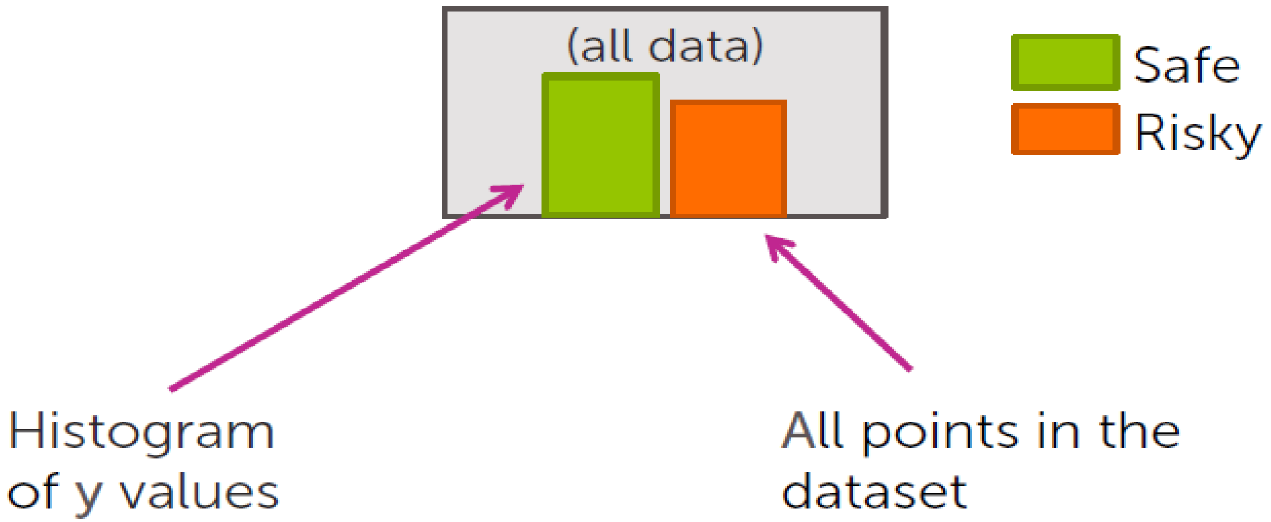
Simple (greedy) algorithm finds good tree

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



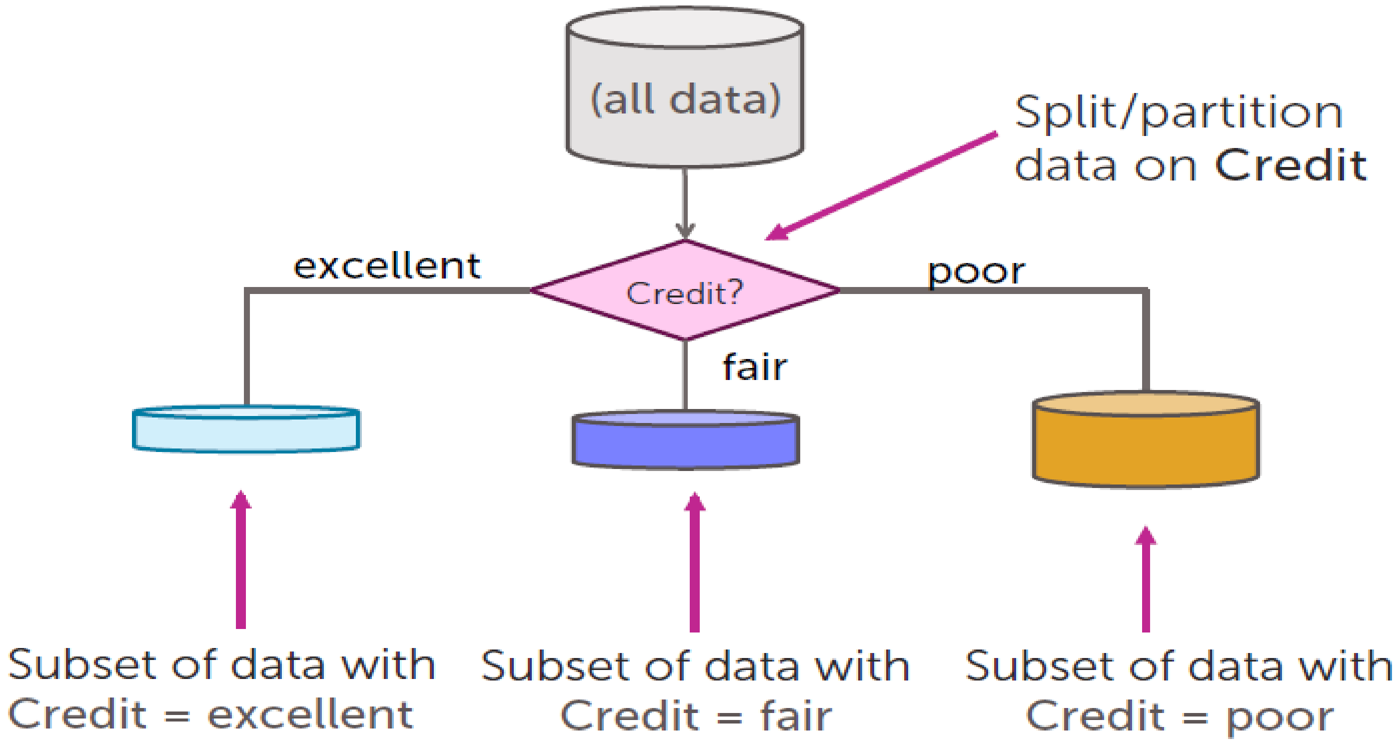
Greedy algorithm

Step 1: Start with an empty tree



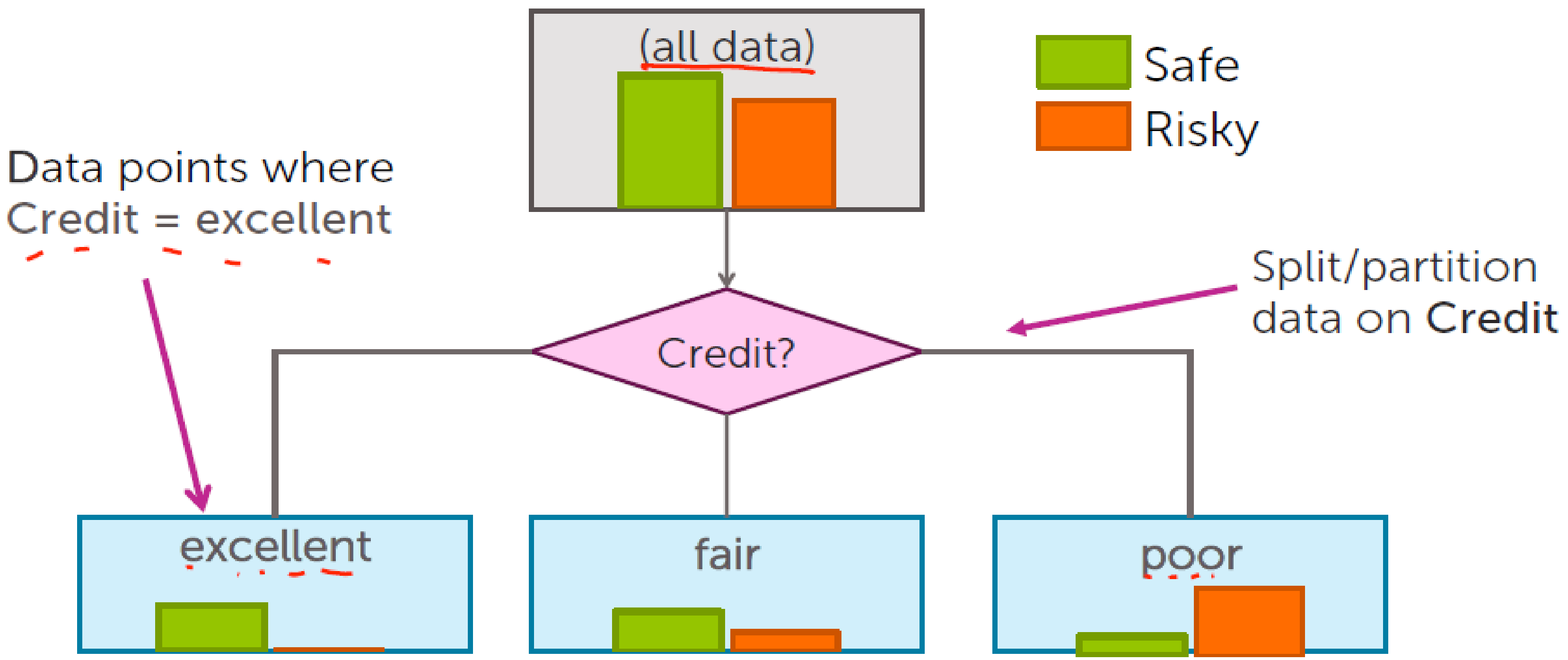
Greedy algorithm

Step 2: Split on a feature



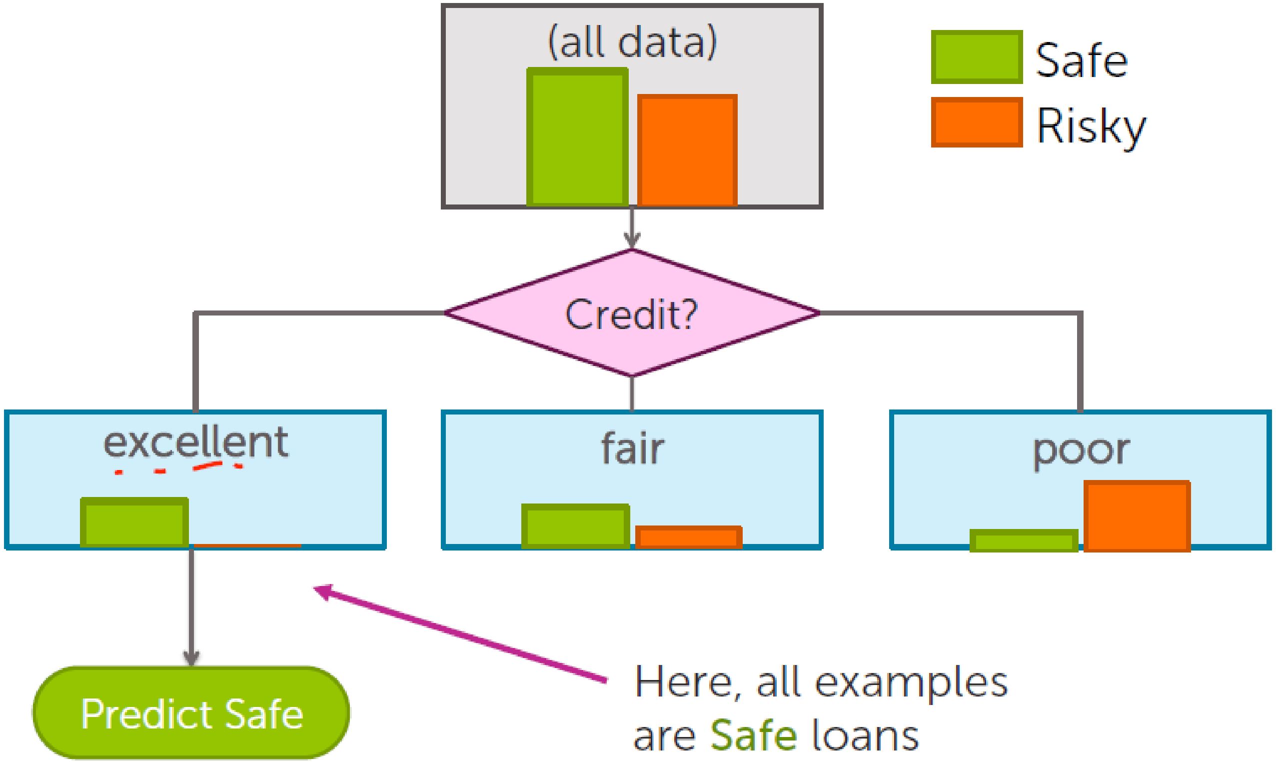
Greedy algorithm

Feature split explained



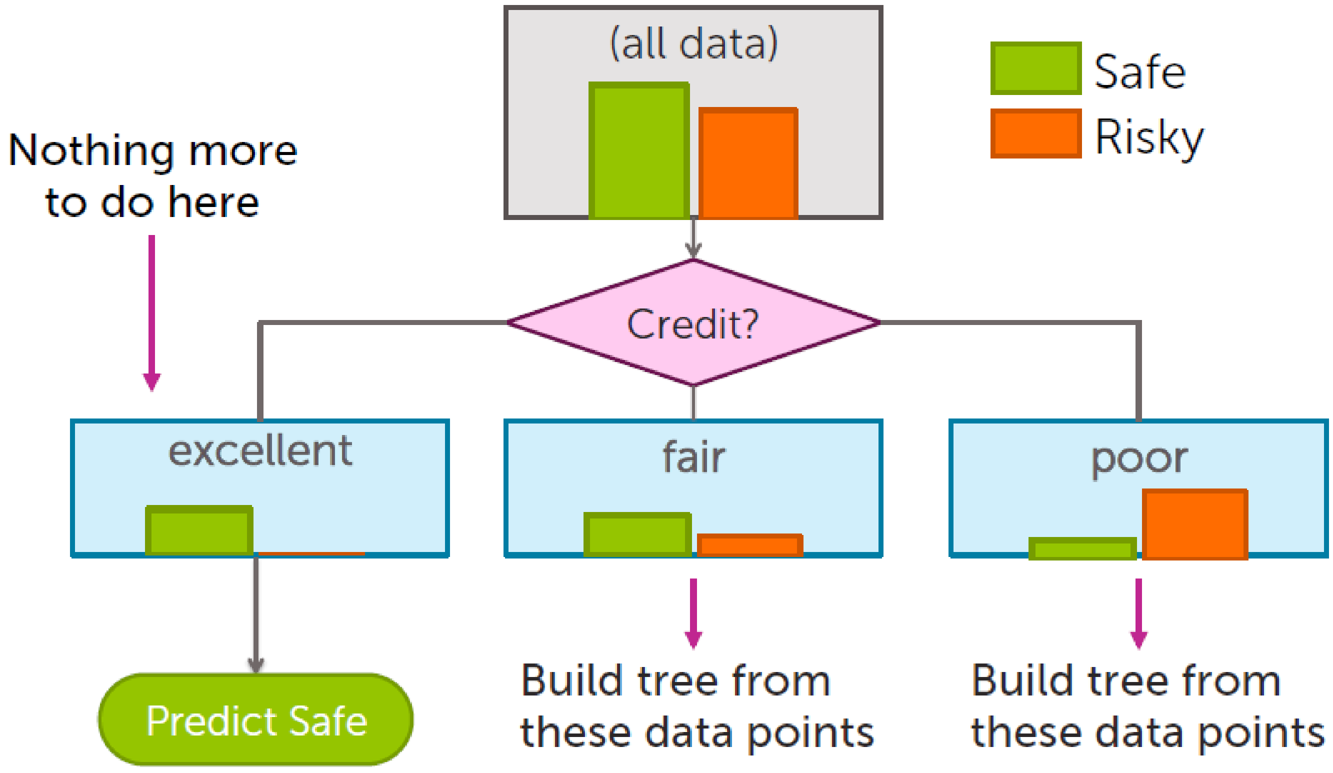
Greedy algorithm

Step 3: Making predictions



Greedy algorithm

Step 4: Recursion



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

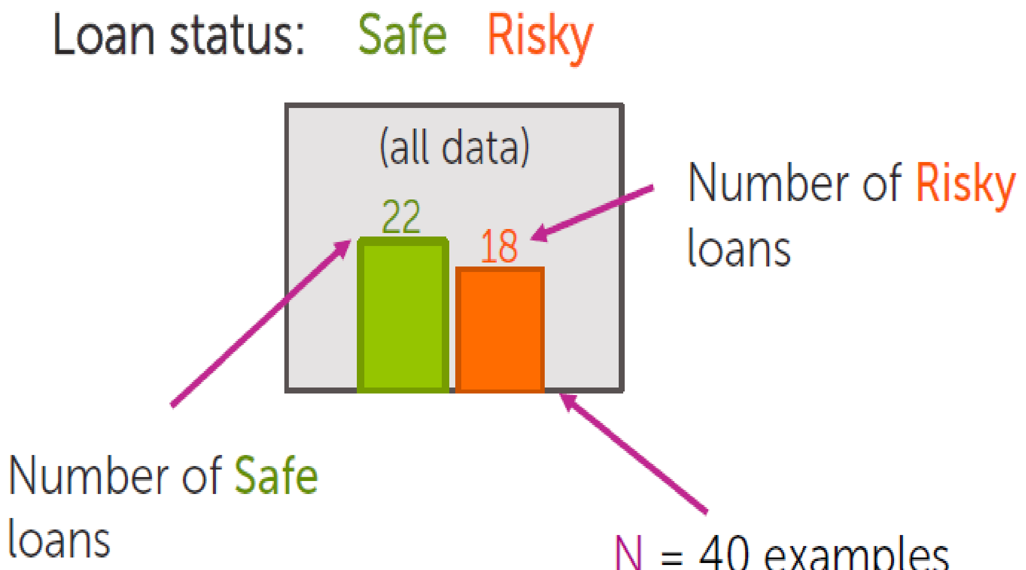
Problem 2: Stopping condition

Recursion

Feature split learning

Start with all the data

Assume N = 40, 3 features

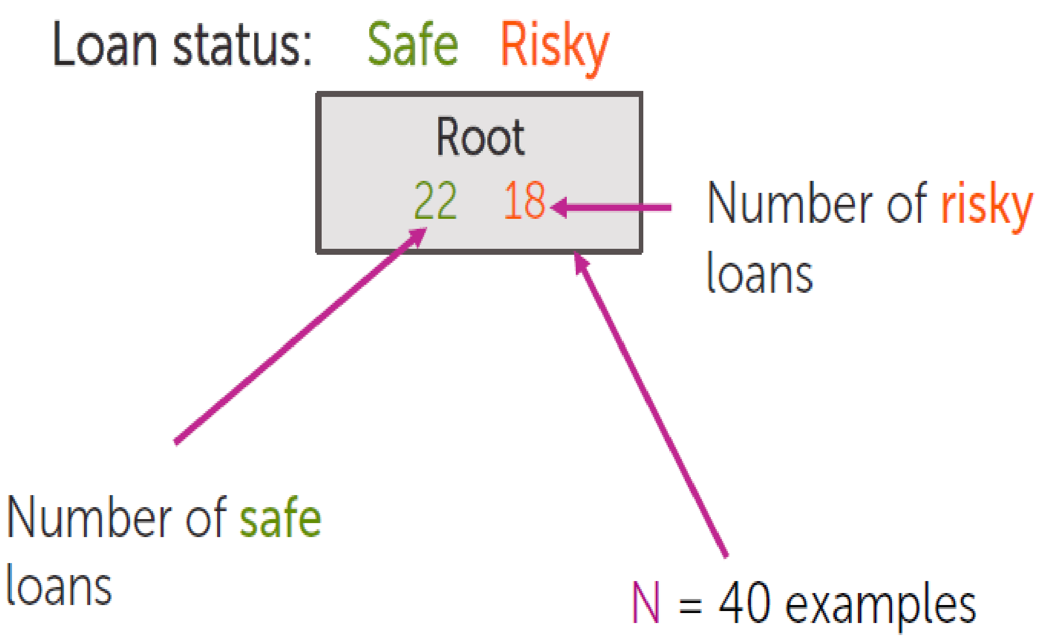


Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	5 yrs	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	low	safe
poor	3 yrs	high	risky
poor	5 yrs	low	safe
fair	3 yrs	high	safe

Feature split learning

Start with all the data

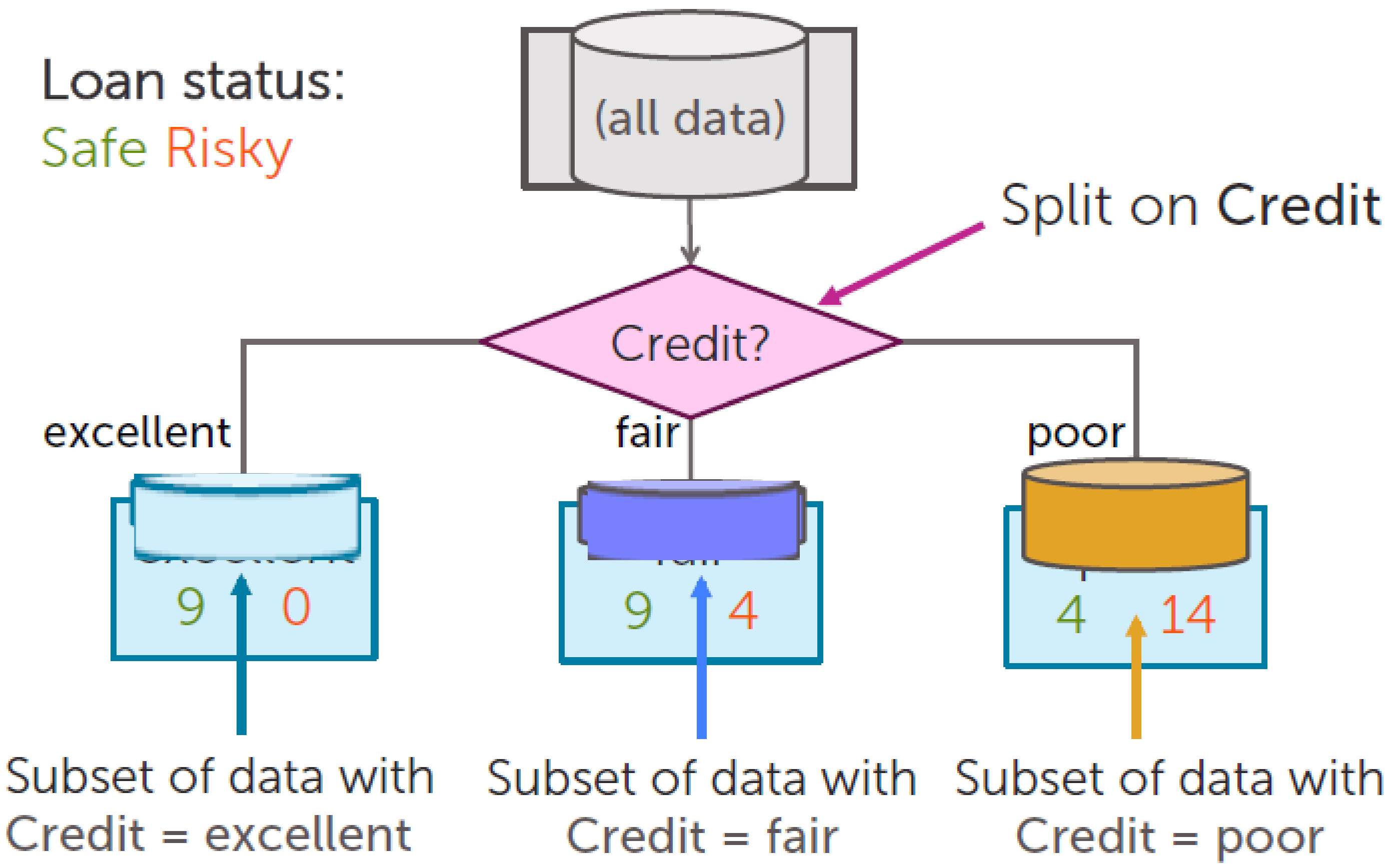
Assume N = 40, 3 features



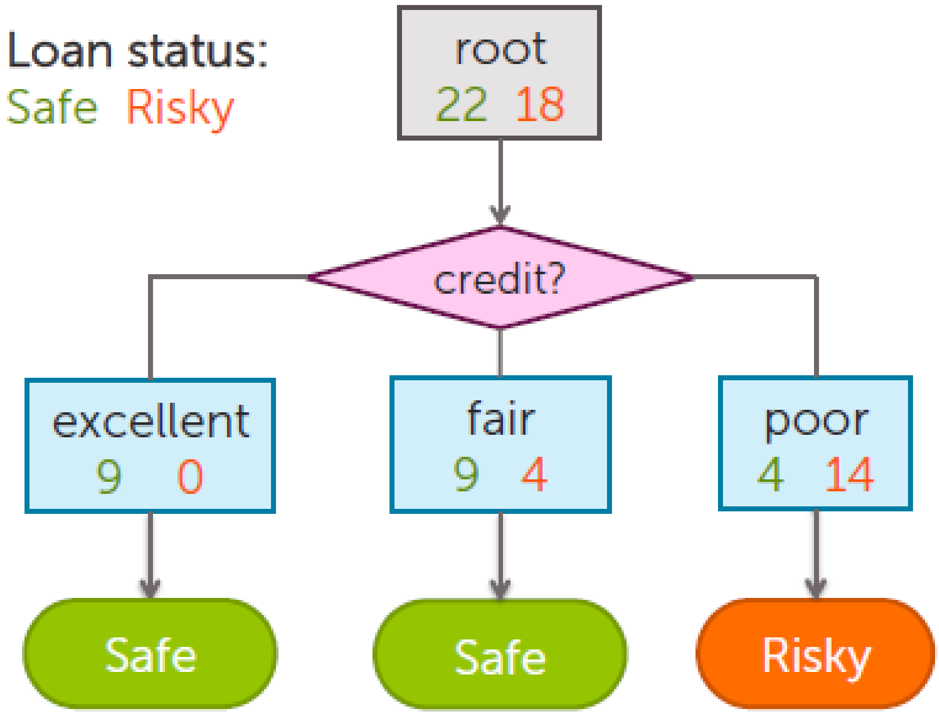
Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	5 yrs	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	low	safe
poor	3 yrs	high	risky
poor	5 yrs	low	safe
fair	3 yrs	high	safe

Compact notation

Decision stump: single level tree

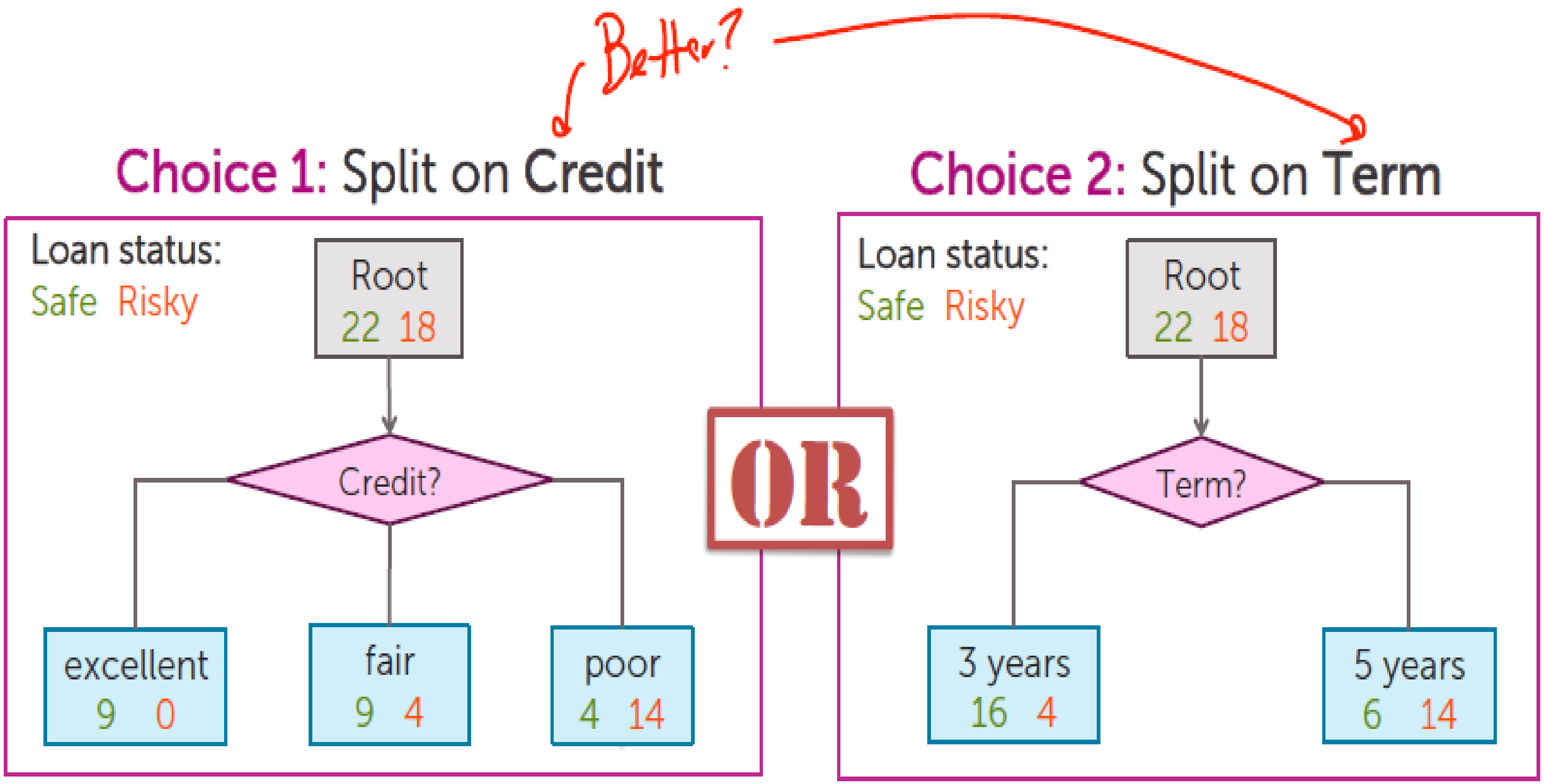


Making predictions with a decision stump

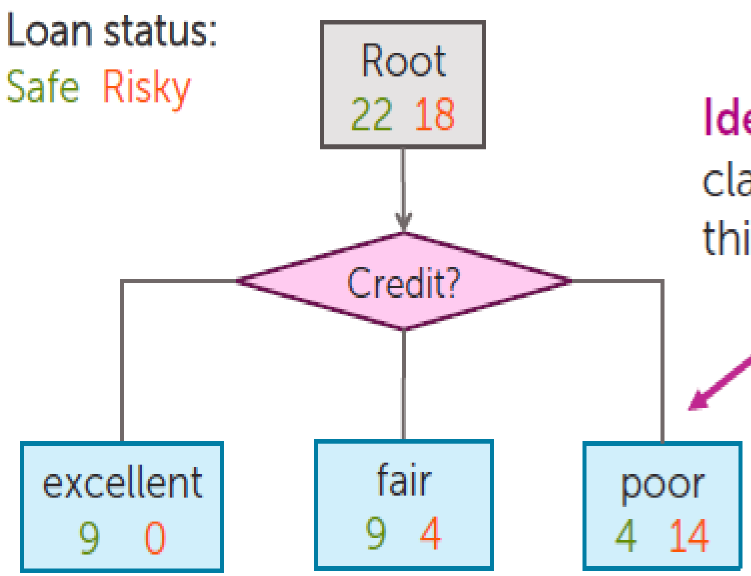


For each intermediate node, set ŷ = majority value

How do we select the best feature to split on?



How do we measure effectiveness of a split?

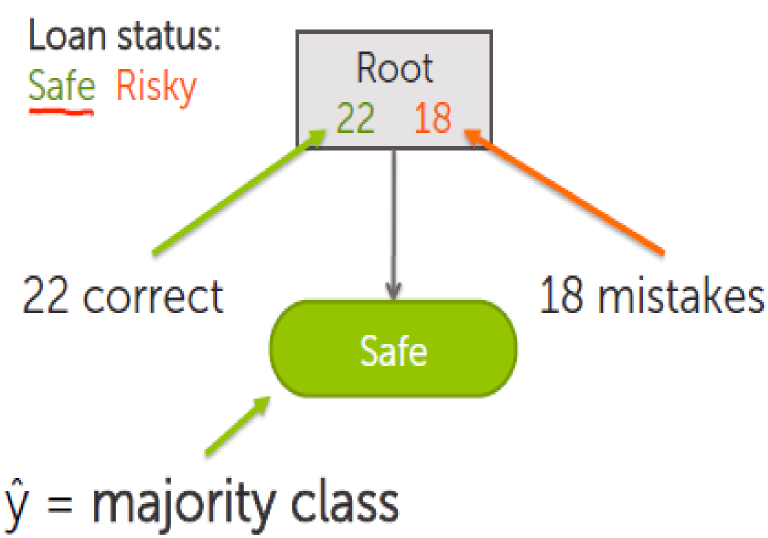


Idea: Calculate classification error of this decision stump

$$\text{Error} = \frac{\#\ \text{mistakes}}{\#\ \text{data points}}$$

Calculating classification error

- Step 1: ŷ = class of majority of data in node
- Step 2: Calculate classification error of predicting ŷ for this data



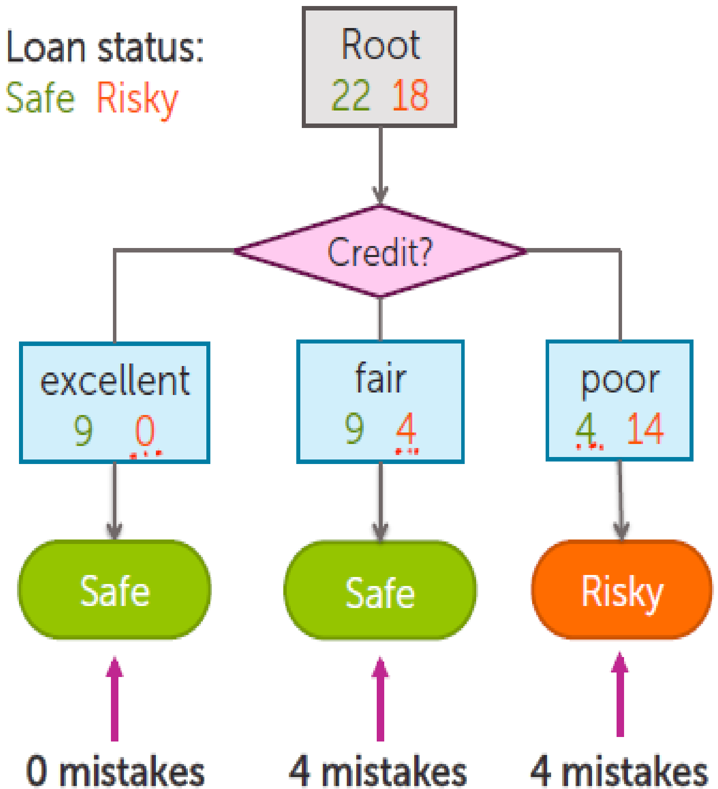
$$\text{Error} = \frac{18}{22+18}$$

$$= 0.45$$

Tree	Classification error
(root)	0.45

Classification error

Choice 1: Split on Credit



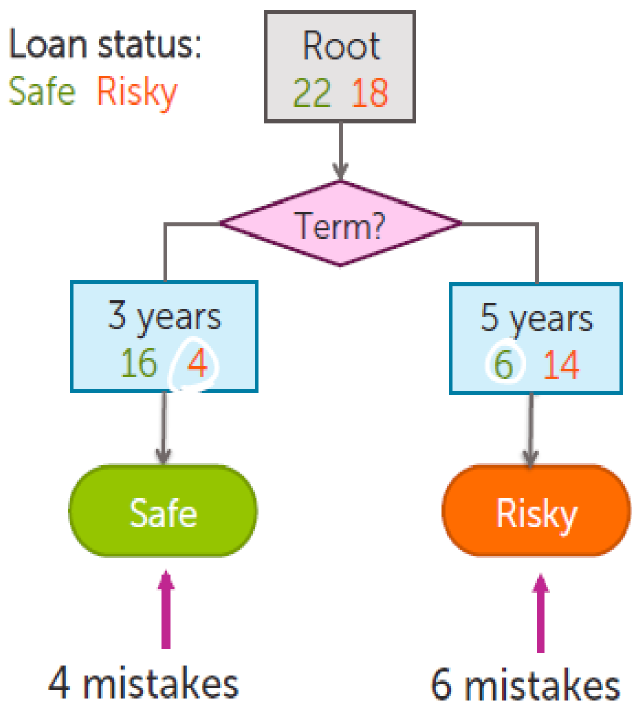
$$\text{Error} = \frac{4+4}{40}$$

$$= 0.20$$

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

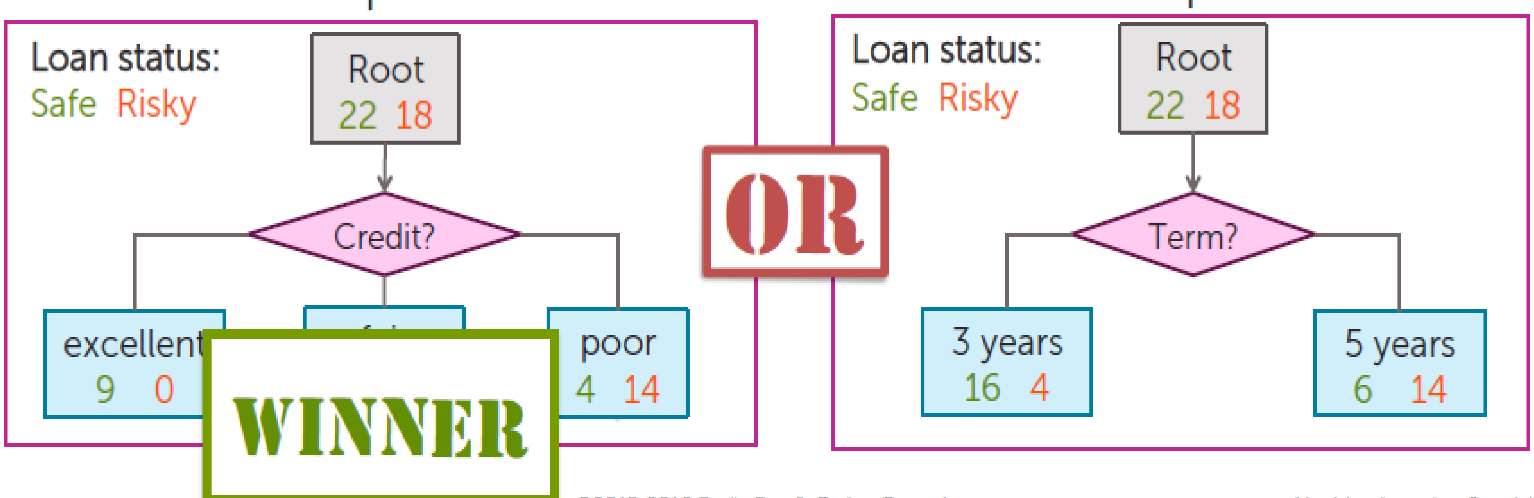
Choice 1 vs Choise 2

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**



30/10, 6/11 2024

Feauture split selection algorithm

- Given a subset of data M (a node in a tree)
- For each feature $h_i(\mathbf{x})$: ↙ credit, term, income
  1. Split data of M according to feature $h_i(\mathbf{x})$
  2. Compute classification error split
- Chose feature $h^*(\mathbf{x})$ with lowest classification error ↖ credit

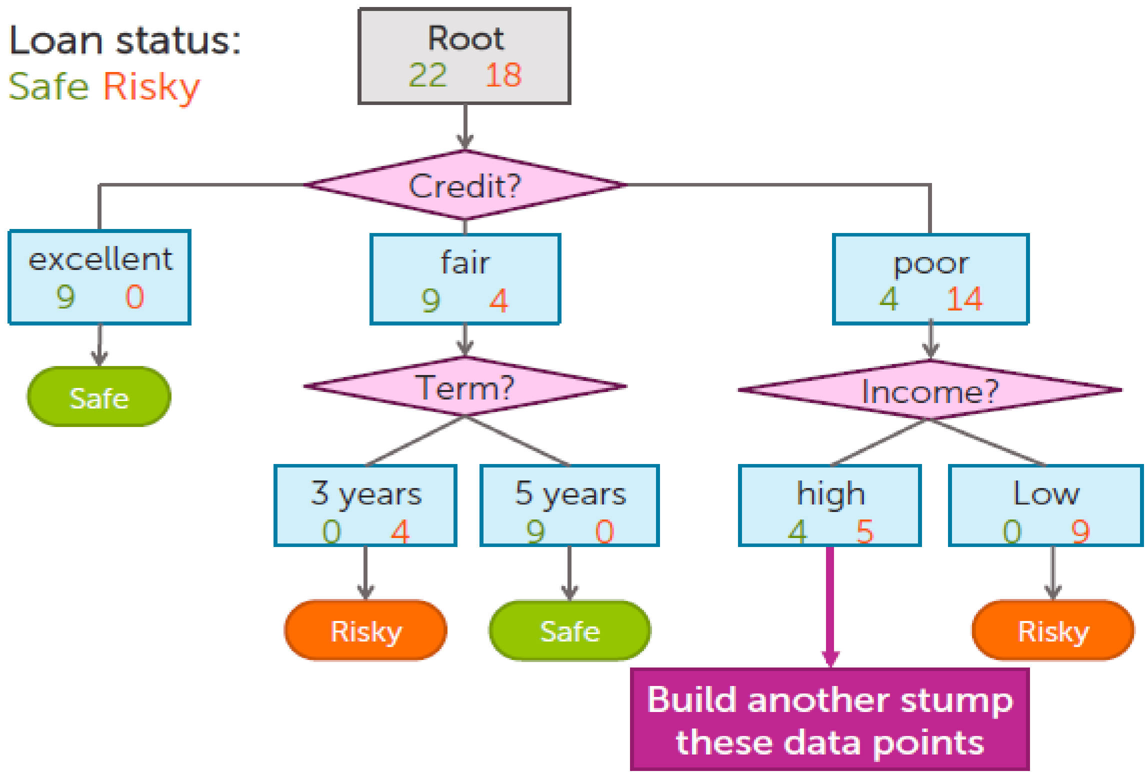
Greedy decision tree learning algorithm

- 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

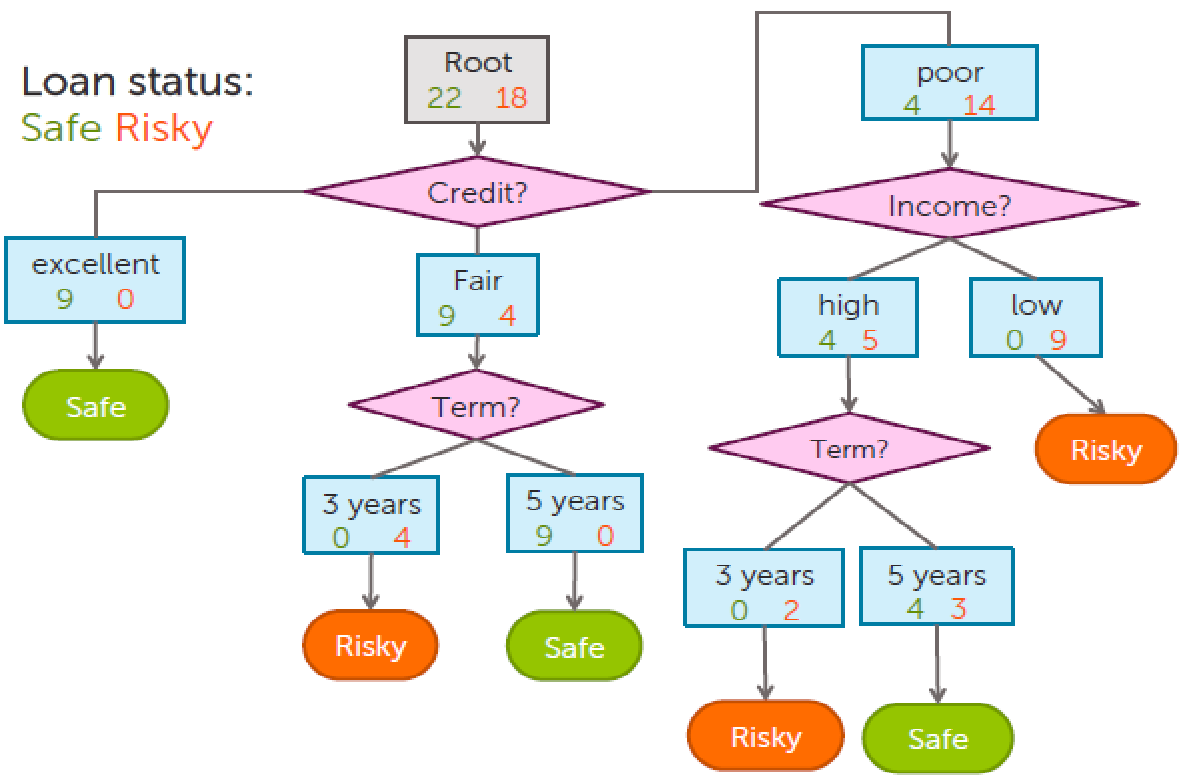
Recursive stump learning

Second level



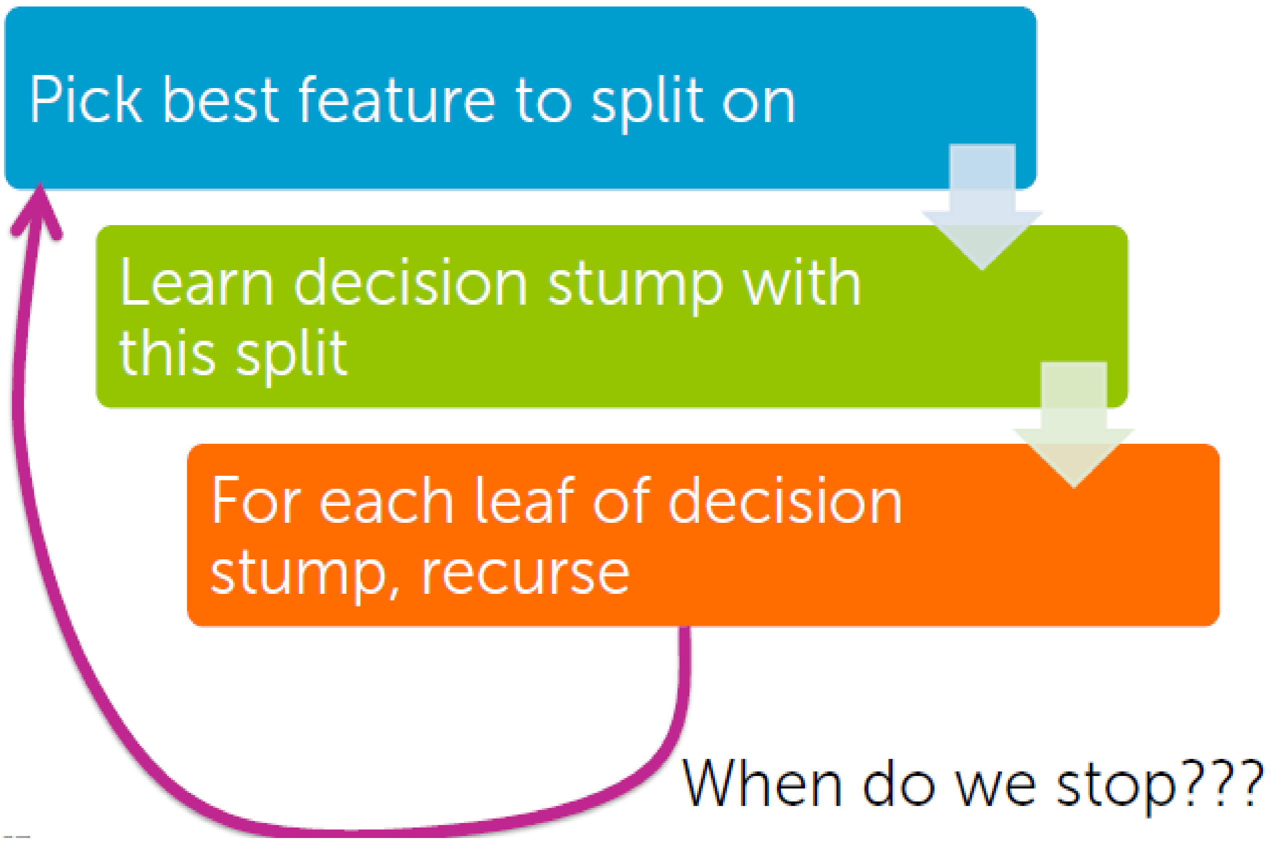
Recursive stump learning

Final decision tree

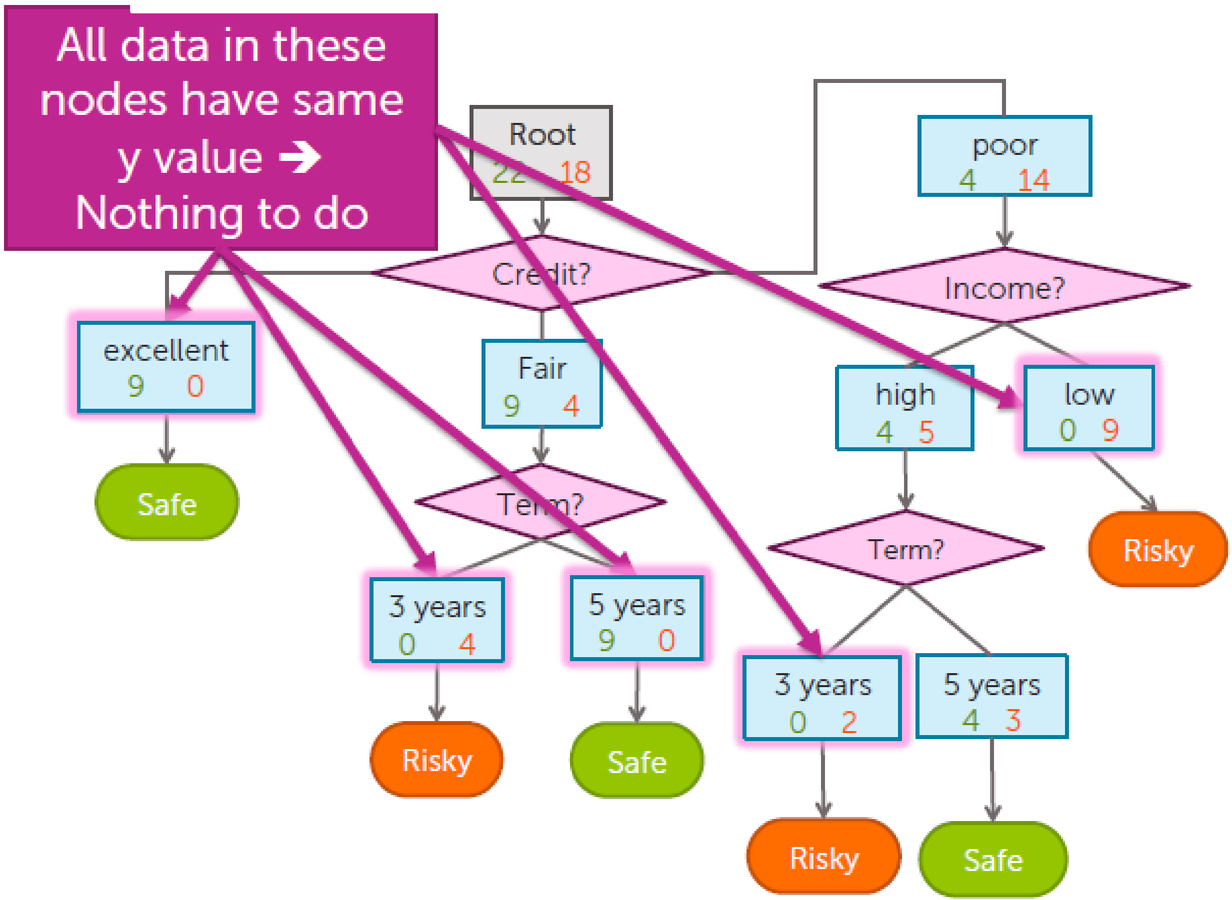


Simple greedy decision tree learning

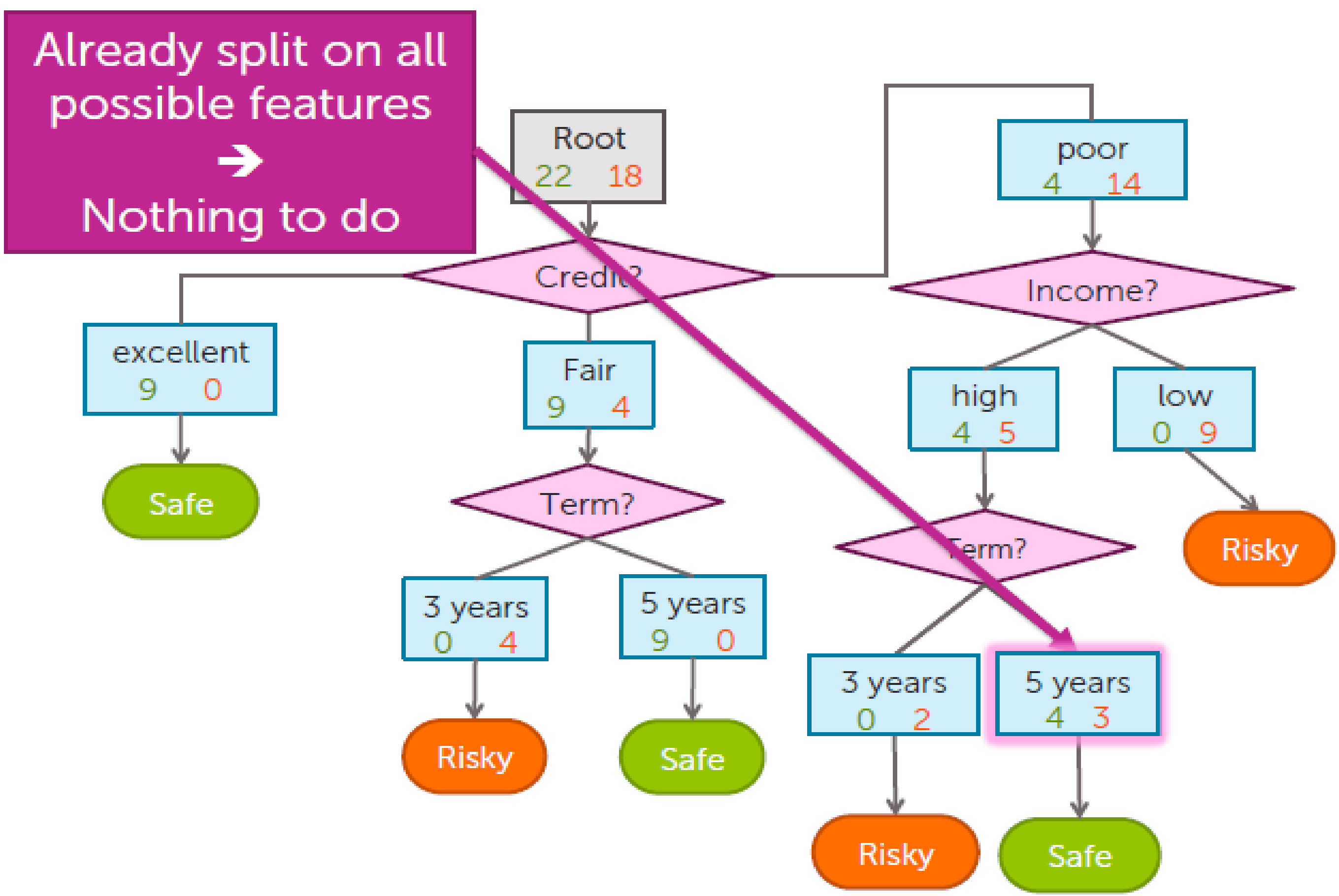
Recursive algorithm



Stopping condition 1



Stopping condition 2



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

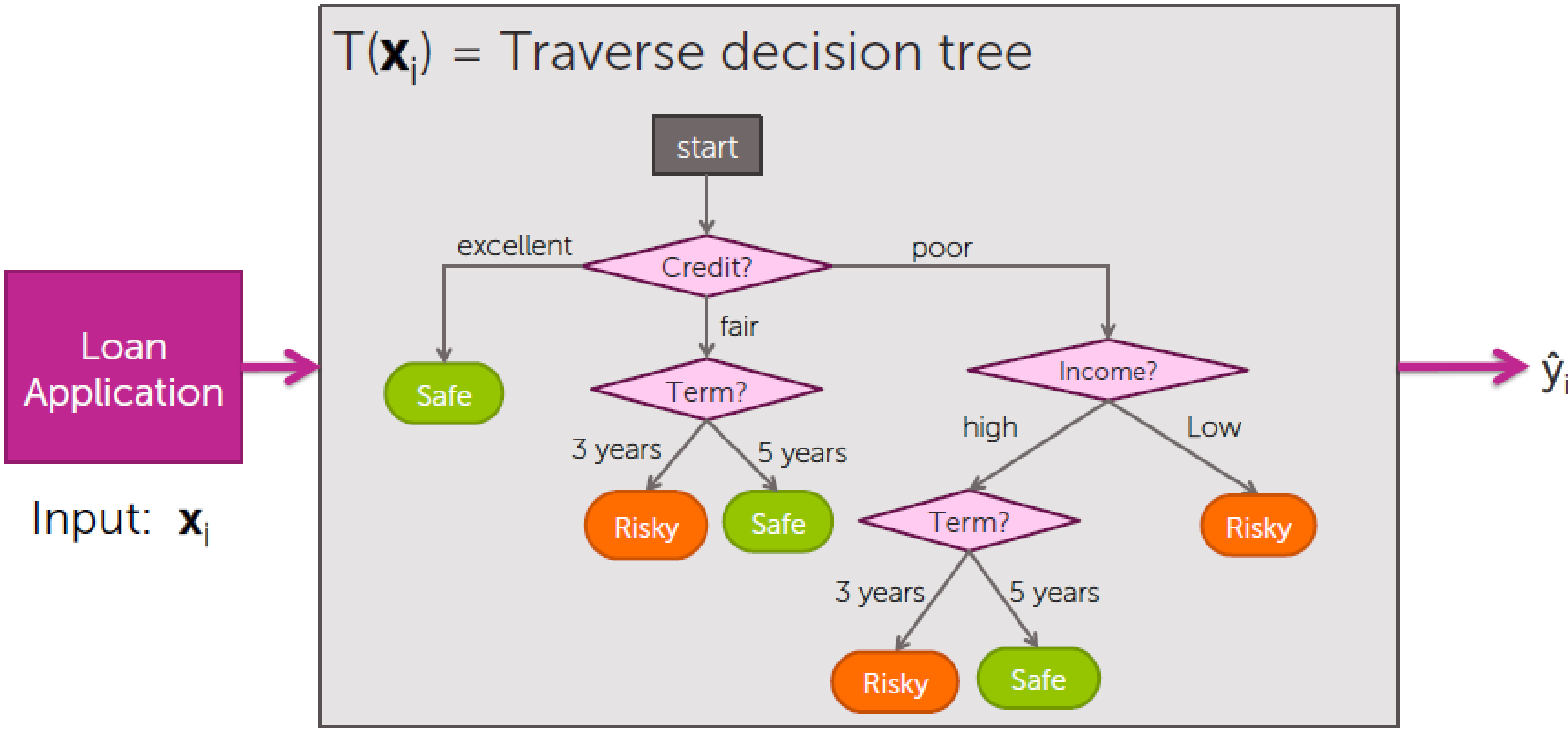
Pick feature split leading to lowest classification error

Stopping conditions 1 & 2

Recursion

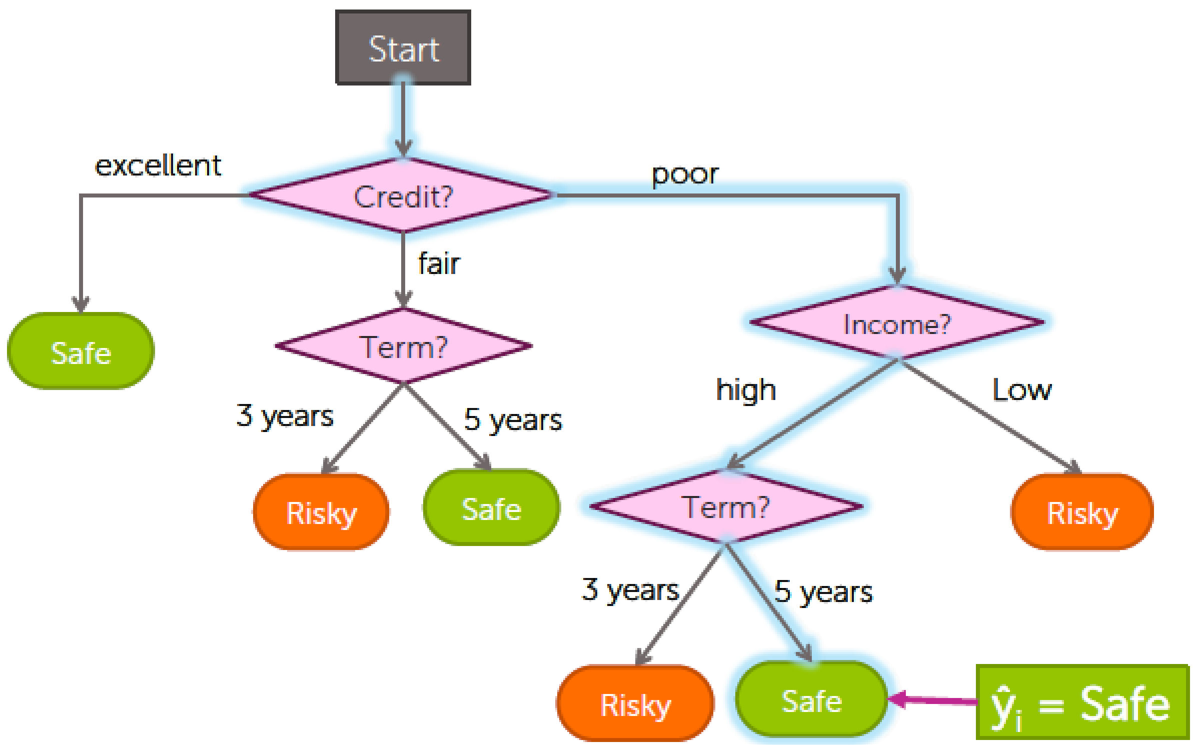
Predictions with decision trees

Decision tree model



Traversing a decision tree

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

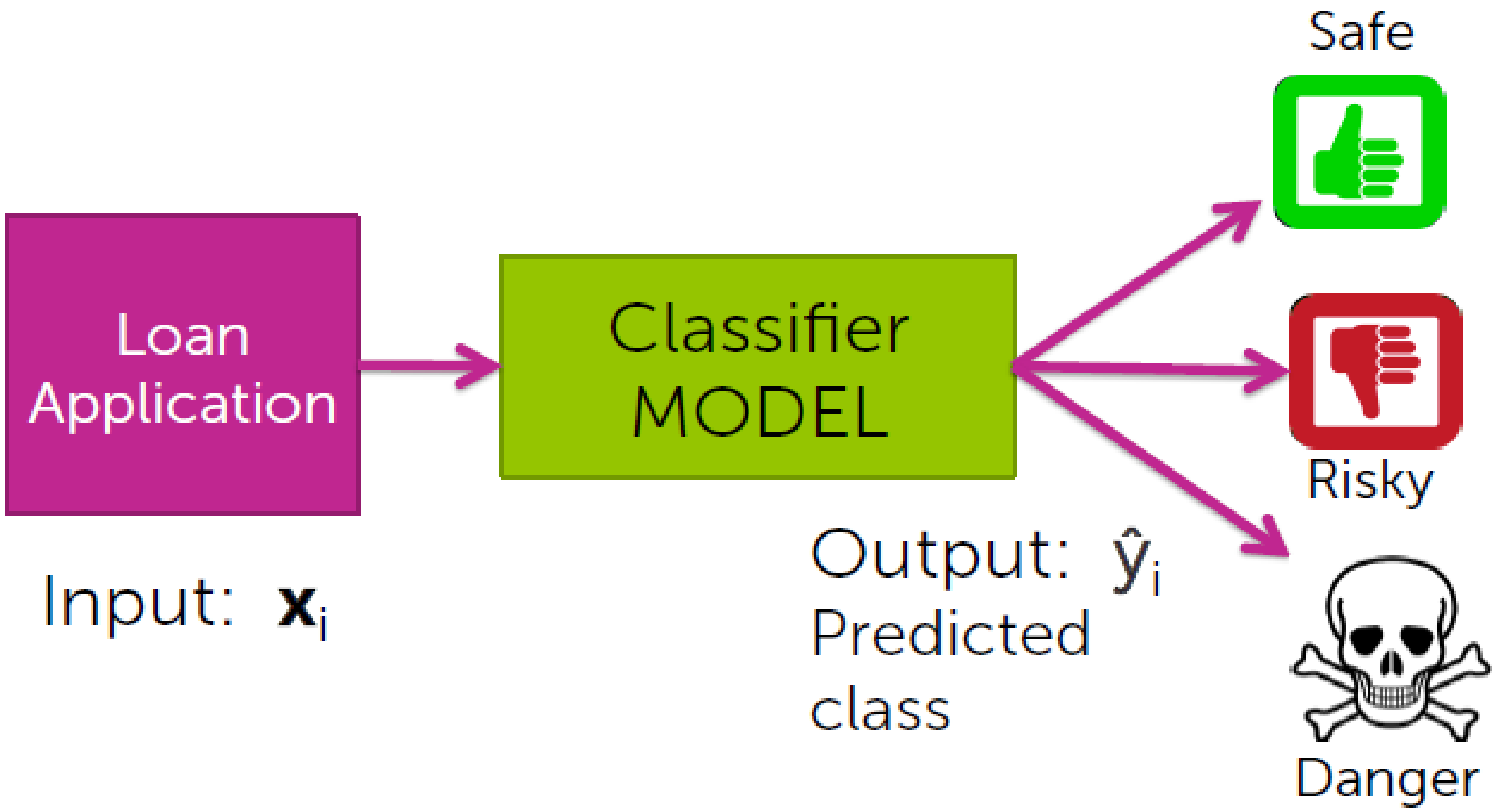


Predictions with decision tree

predict(tree_node, input)

- If current tree_node is a leaf:
  - return majority class of data points in leaf
- else:
  - next_note = child node of tree_node whose feature value agrees with input
  - return predict(next_note, input)

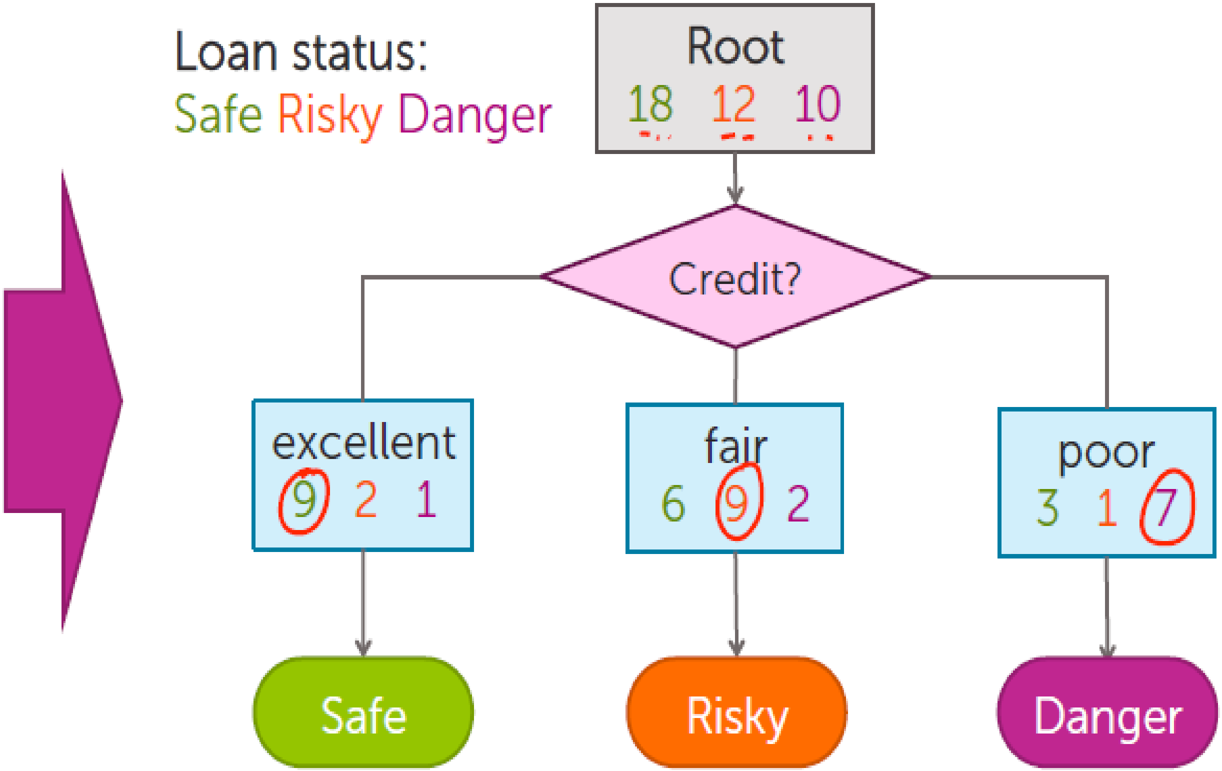
Multiclass prediction



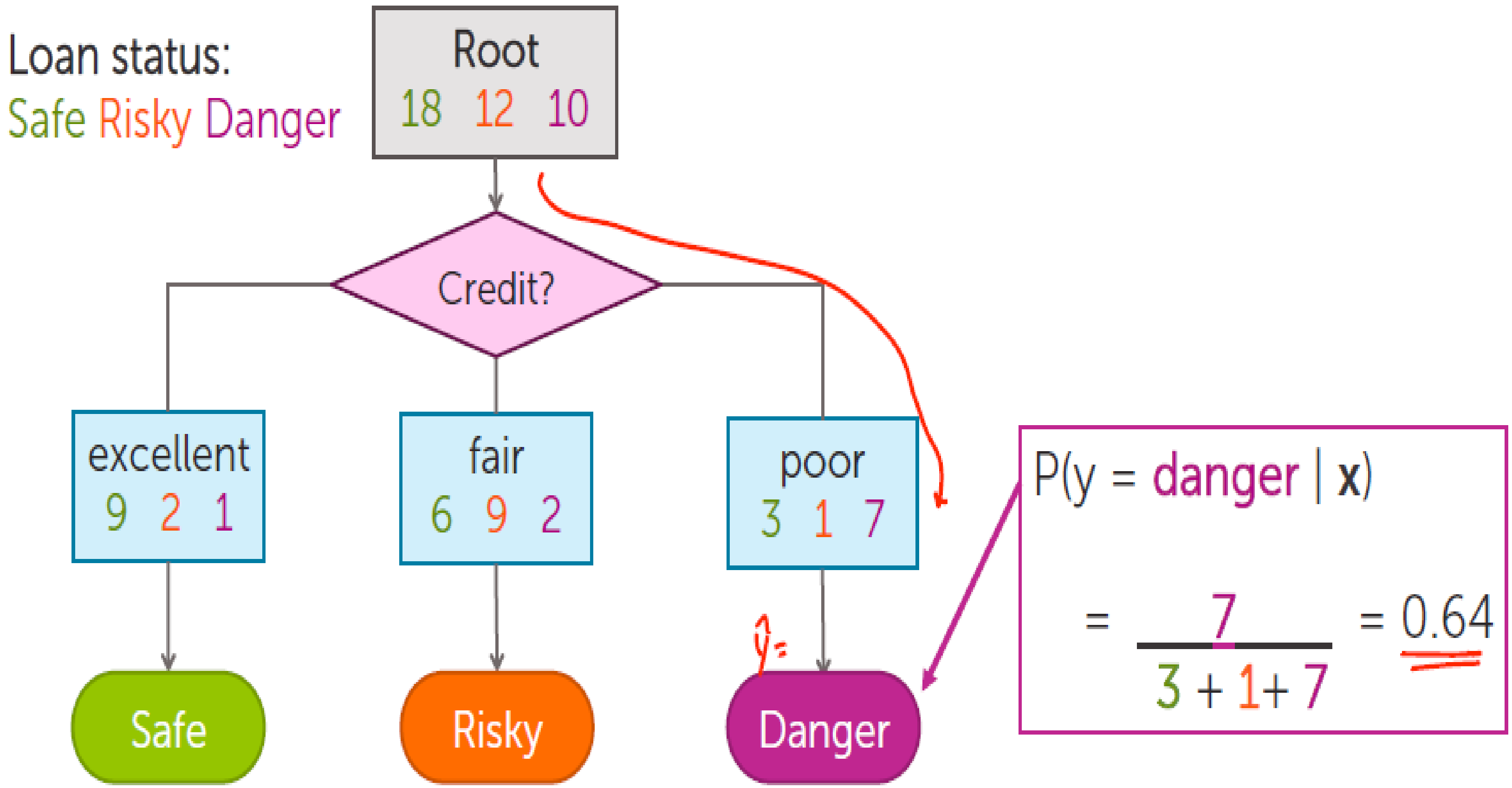
Multiclass decision stump

N = 40,
1 feature,
3 classes

Credit	y
excellent	safe
fair	risky
fair	safe
poor	danger
excellent	risky
fair	safe
poor	danger
poor	safe
fair	safe
...	...



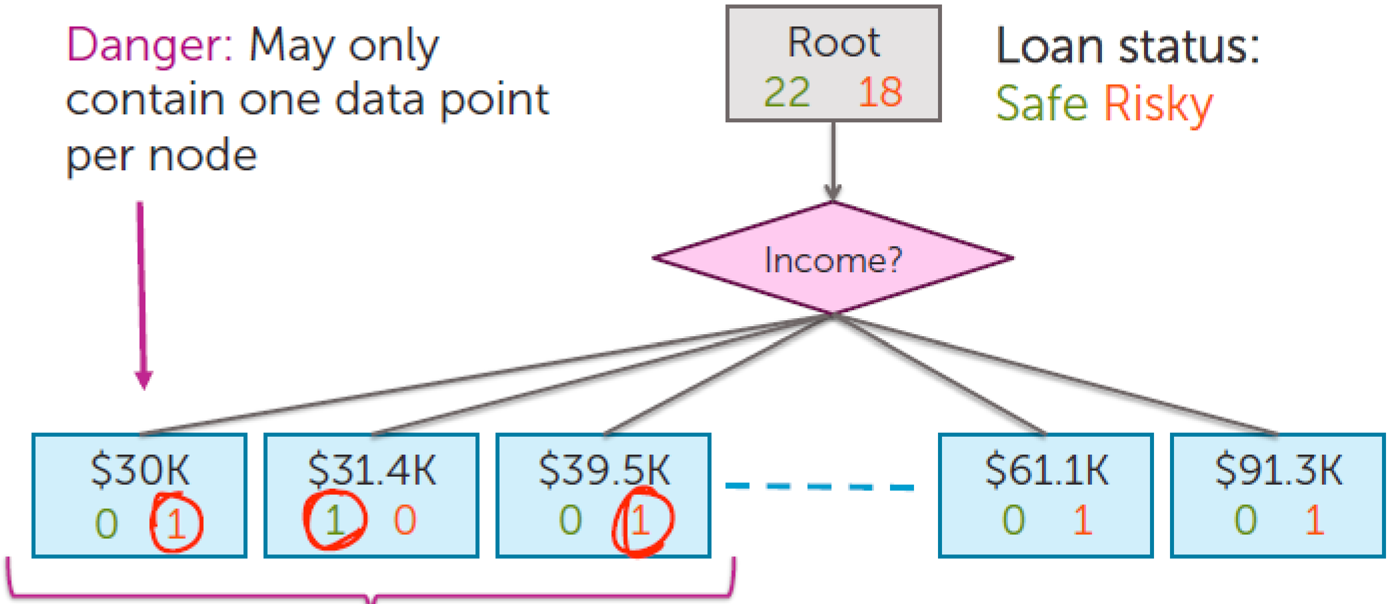
Predicting probabilities with decision trees



How to use real values inputs

Split on each numeric value?

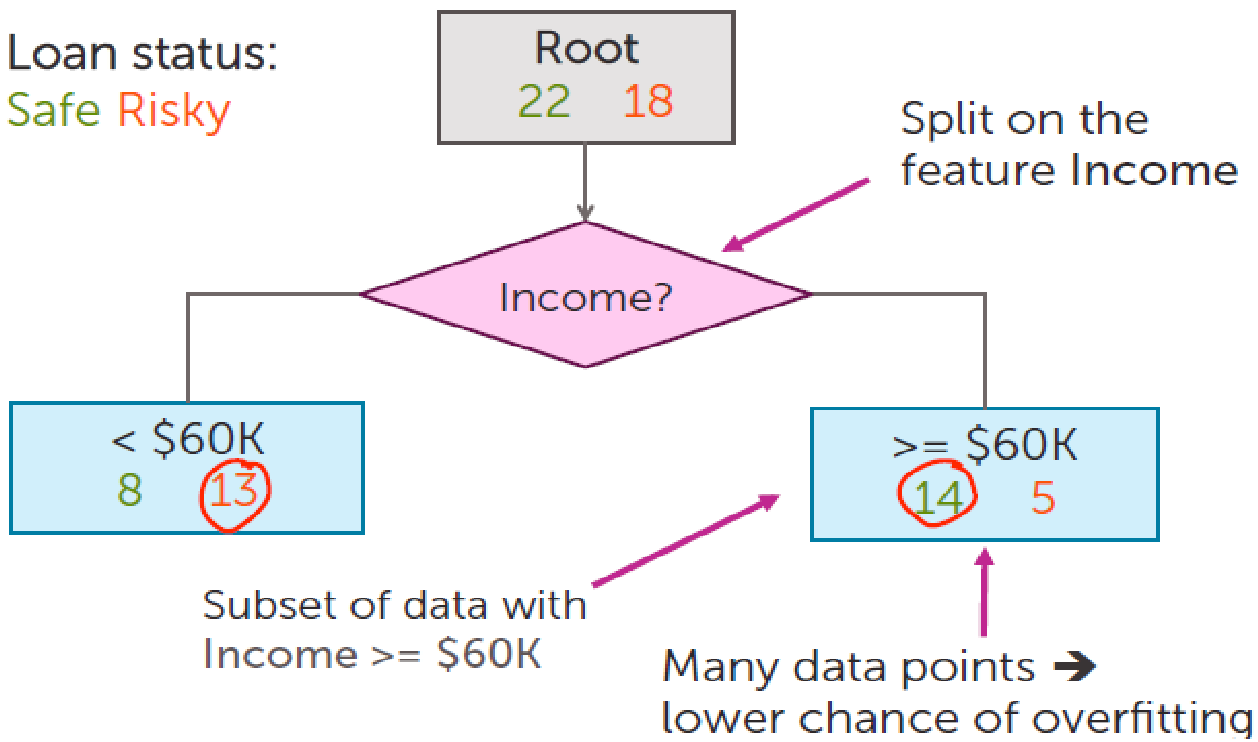
Danger: May only contain one data point per node



Can’t trust prediction (overfitting)

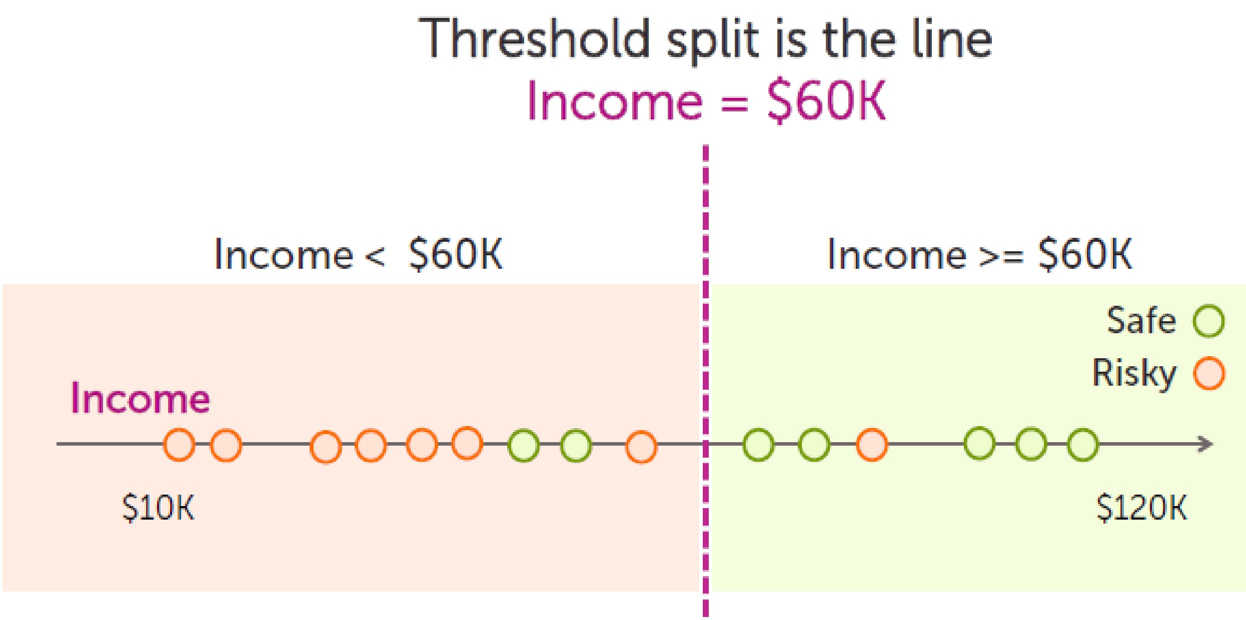
How to use real values inputs

Alternative: Threshold split



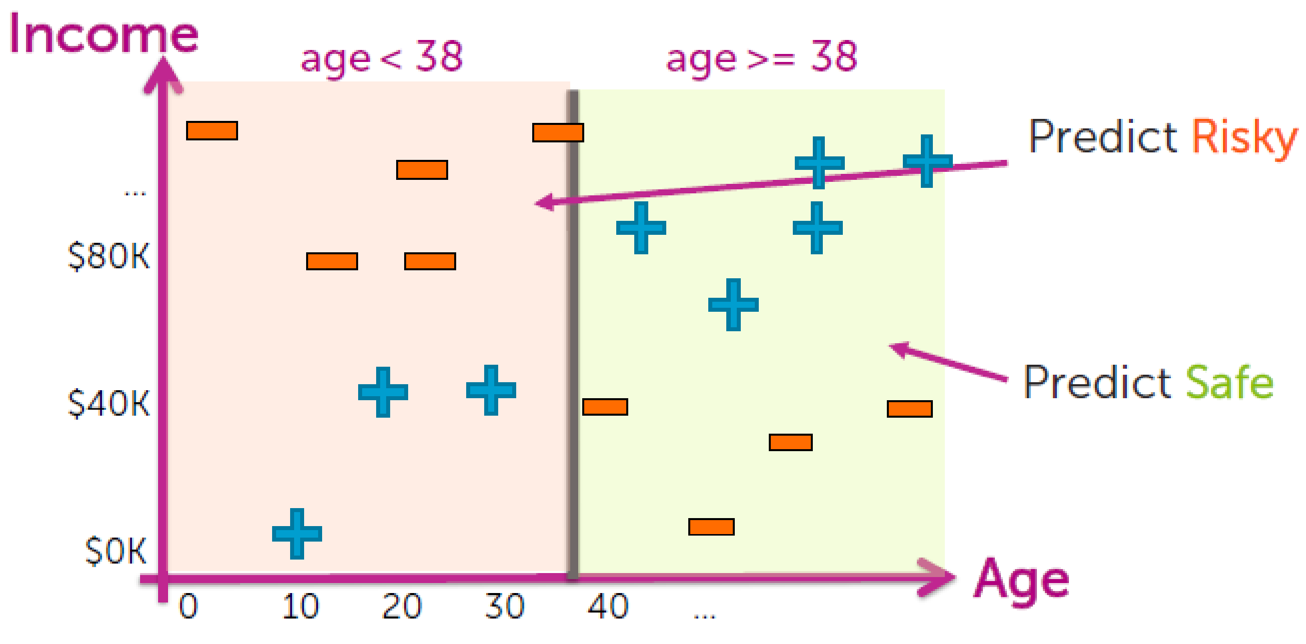
Visualizing the threshold split

Threshold splits in 1-D



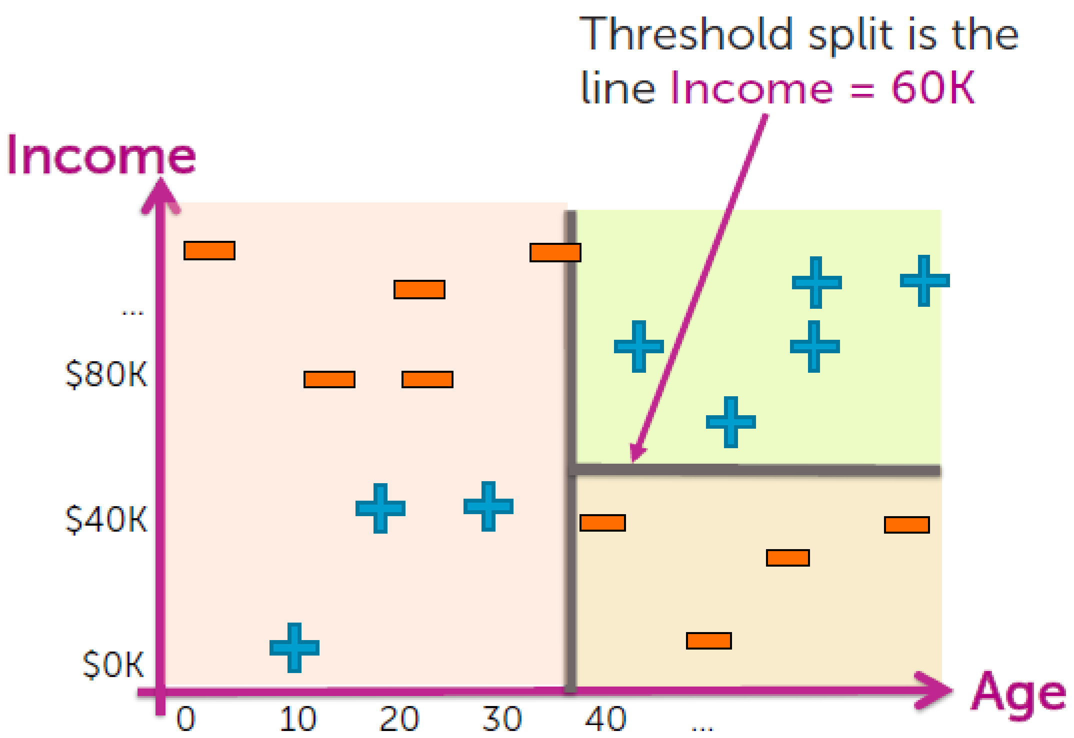
Visualizing the threshold split

Split on Age >= 38



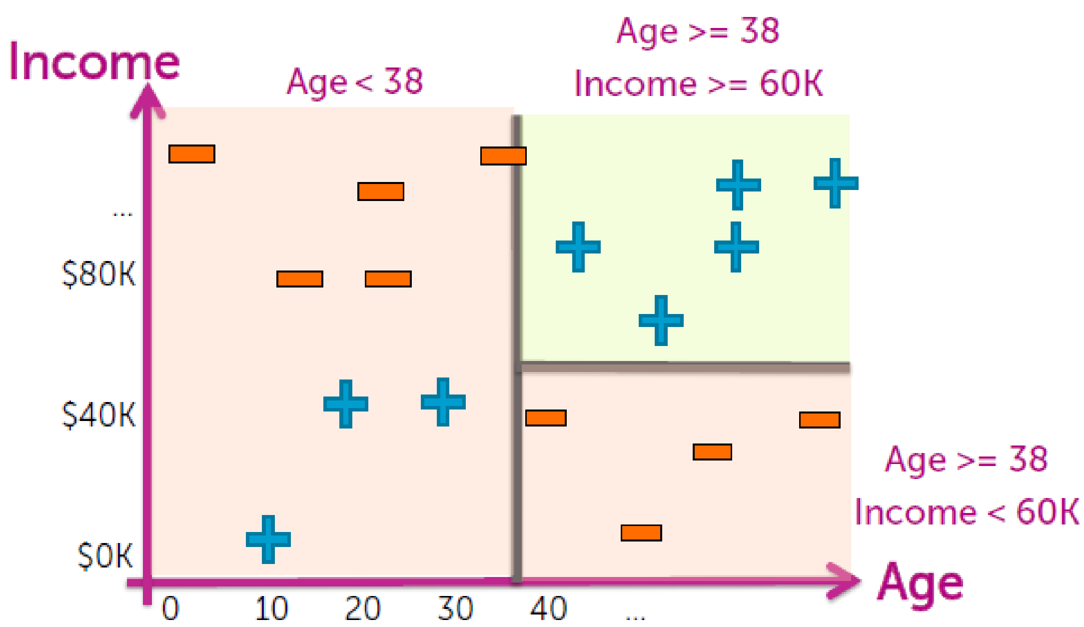
Visualizing the threshold split

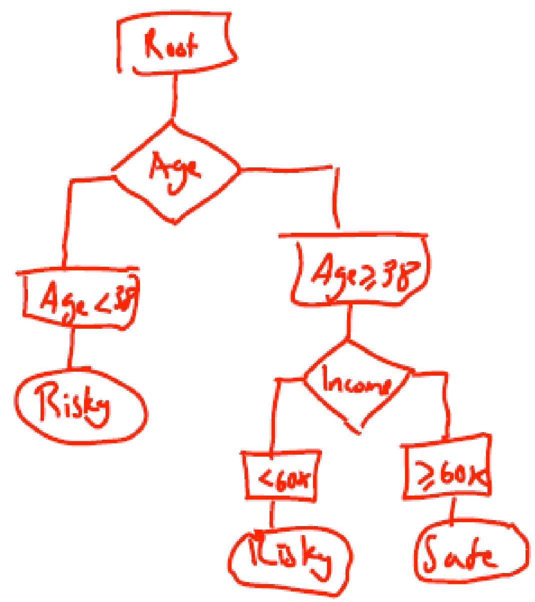
Depth 2: Split on Income >= $60K



Visualizing the threshold split

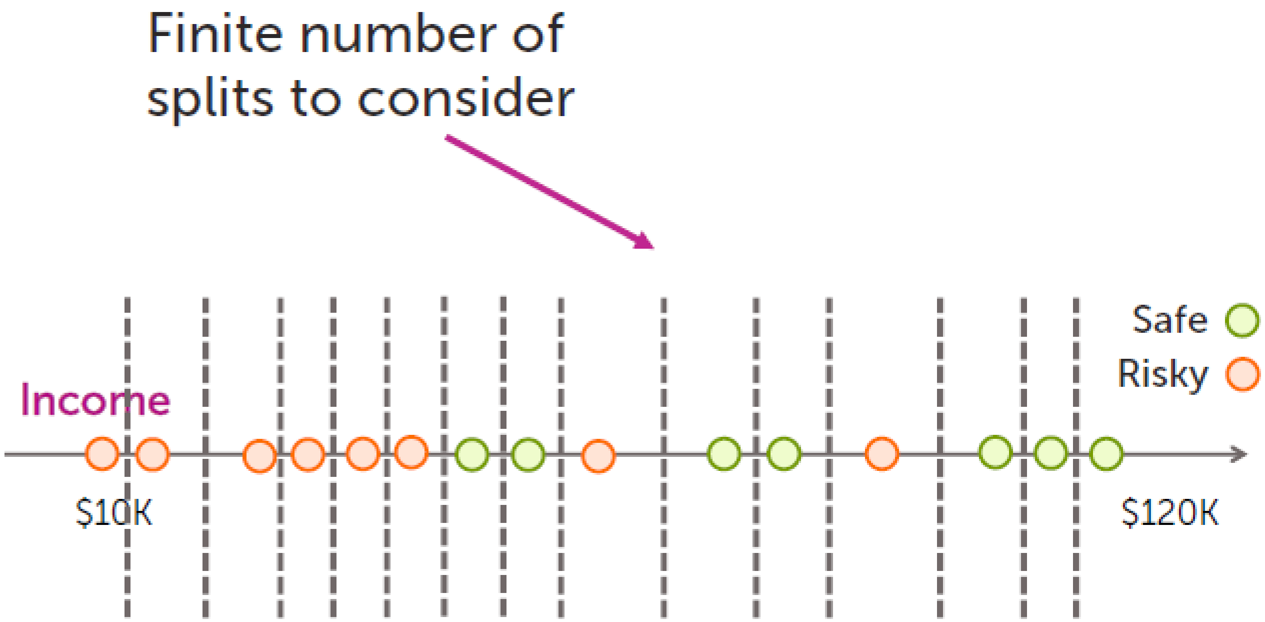
Each split partitions the 2-D space





Finding the best threshold split

Only need to consider mid-points



Finding the best threshold split

Threshold split selection algorithm

Income

- Step 1: Sort the values of a feature $h_j(\mathbf{x})$:

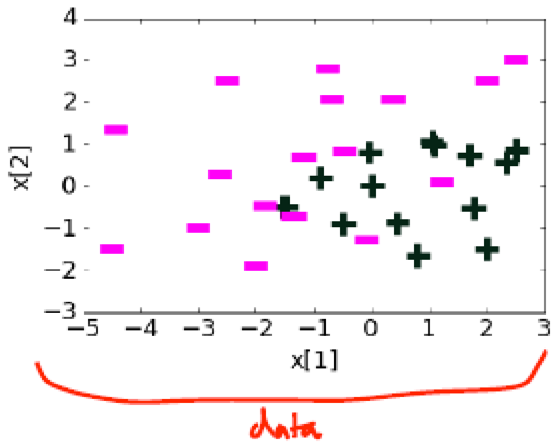
  Let $\{v_1, v_2, v_3, \dots v_N\}$ denote sorted values

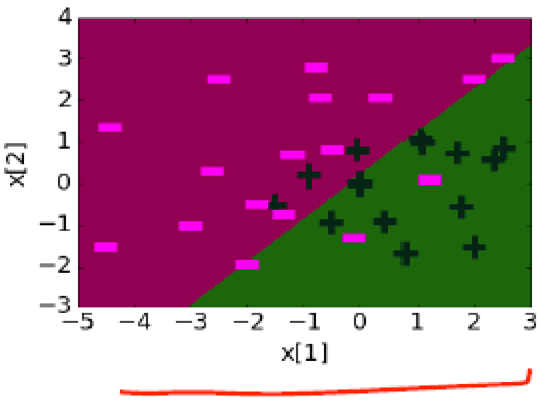
- Step 2:
  - For i = 1 … N-1
    - Consider split $t_i = (v_i + v_{i+1}) / 2$
    - Compute classification error for treshold split $h_j(\mathbf{x}) >= t_i$
  - Chose the t^* with the lowest classification error

Decision trees vs logistic regression

Logistic regression

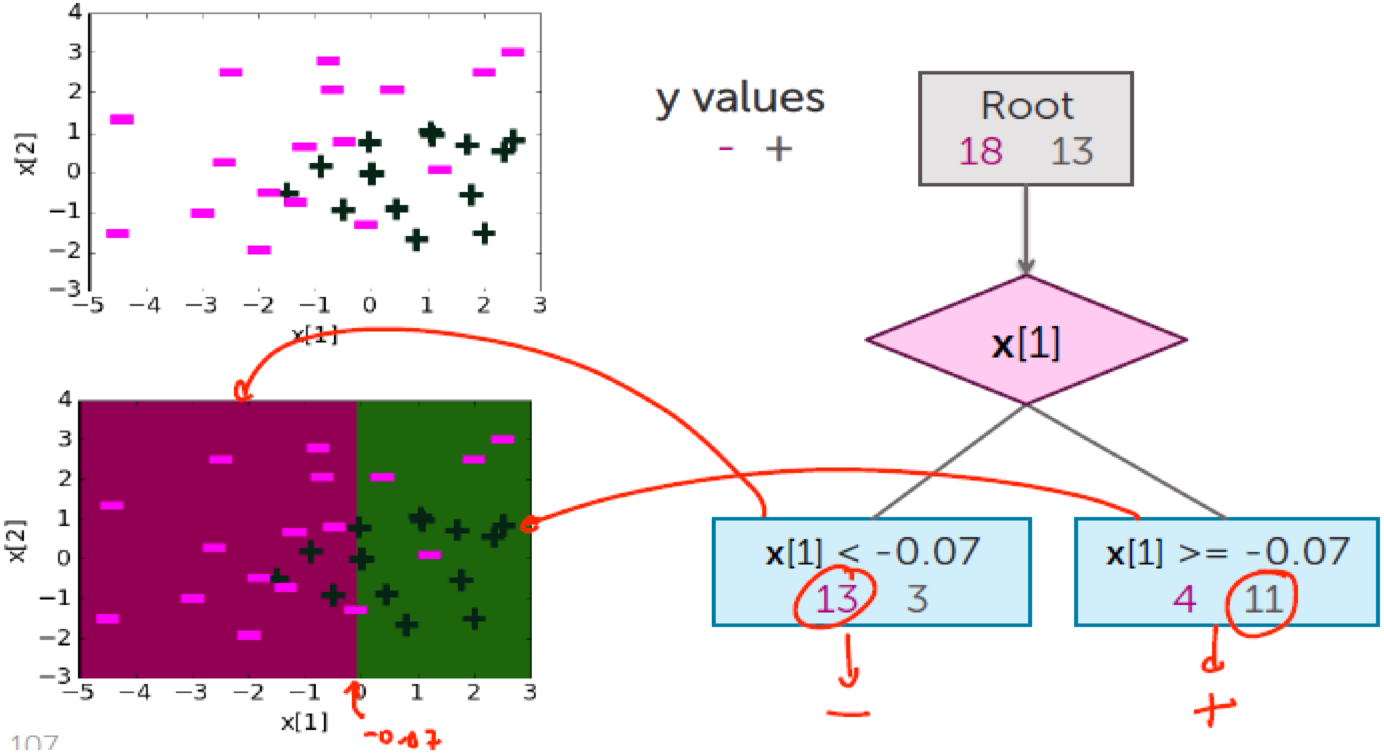
Feature	Value	Weight Learned
$h_0(\mathbf{x})$	1	0.22
$h_1(\mathbf{x})$	$\mathbf{x}[1]$	1.12
$h_2(\mathbf{x})$	$\mathbf{x}[2]$	-1.07





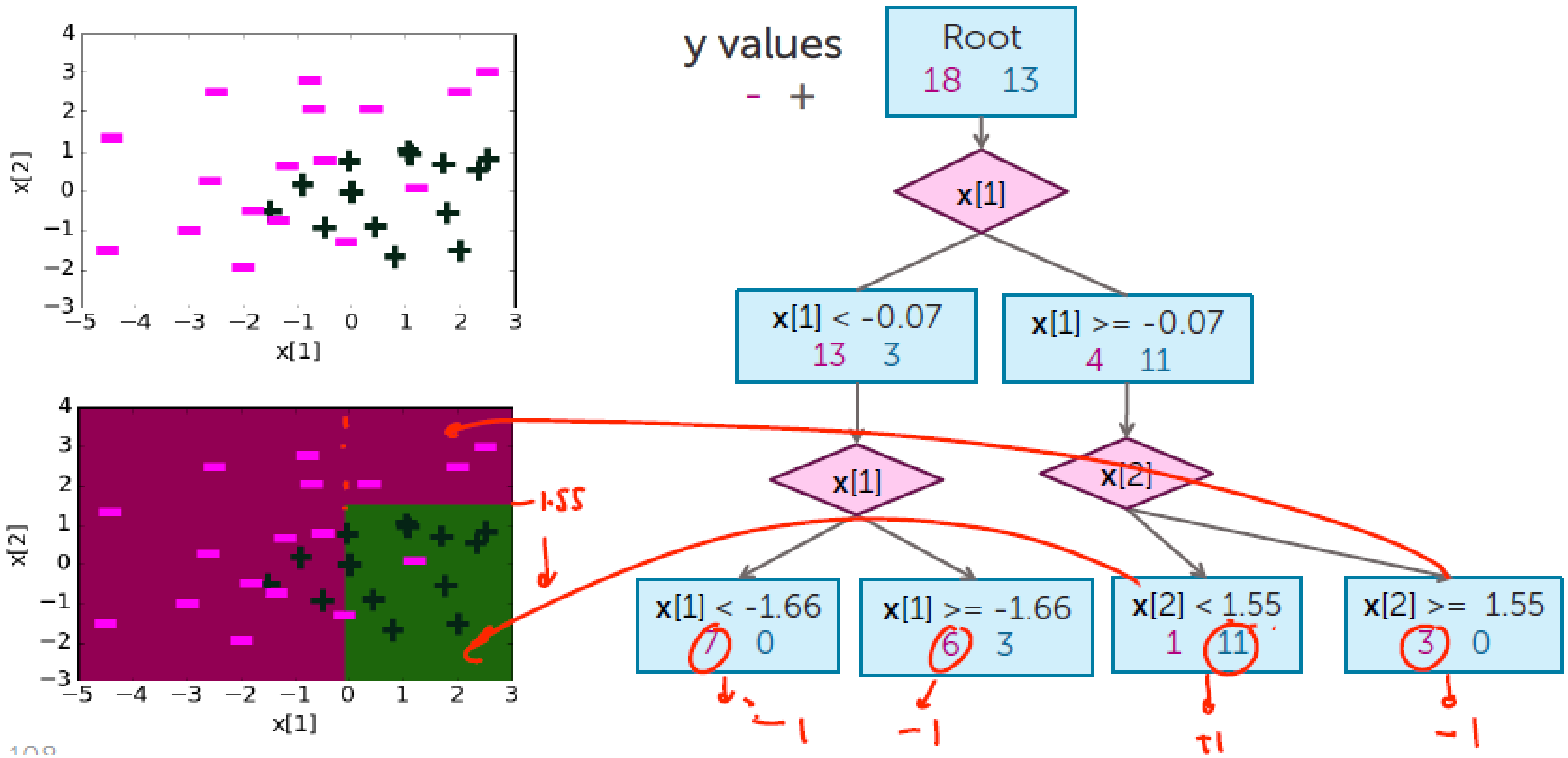
Decision trees vs logistic regression

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



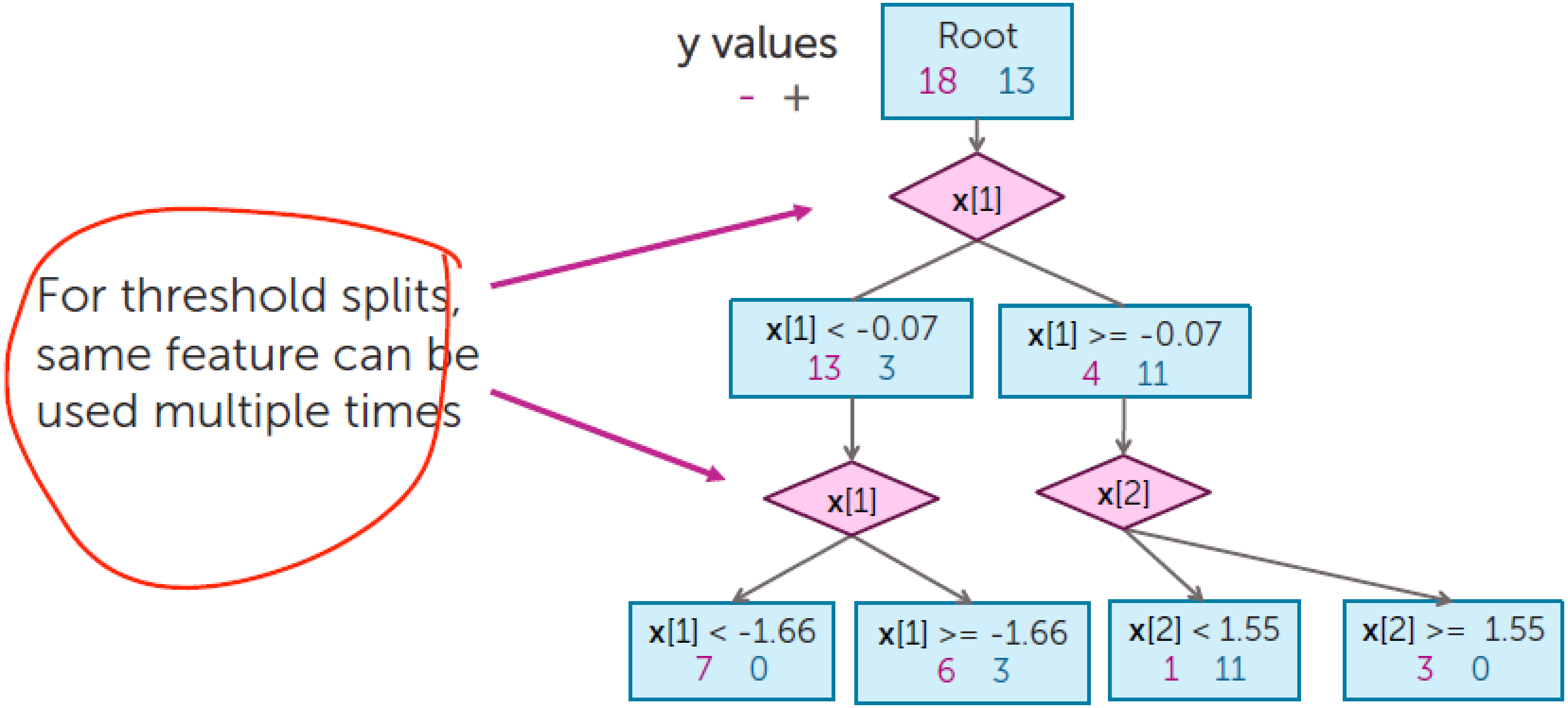
Decision trees vs logistic regression

Depth 2



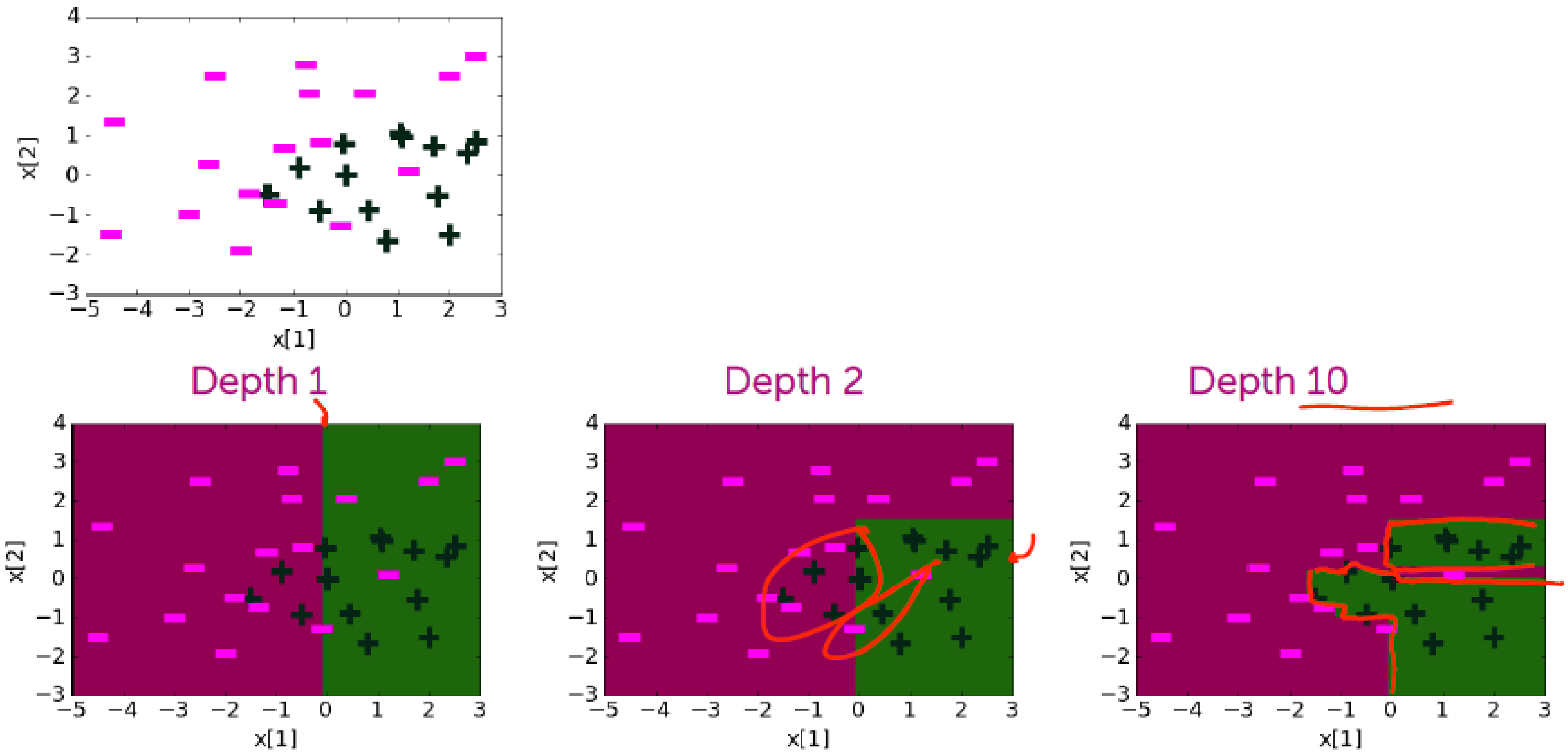
Decision tree vs logistic regression

Threshold split caveat



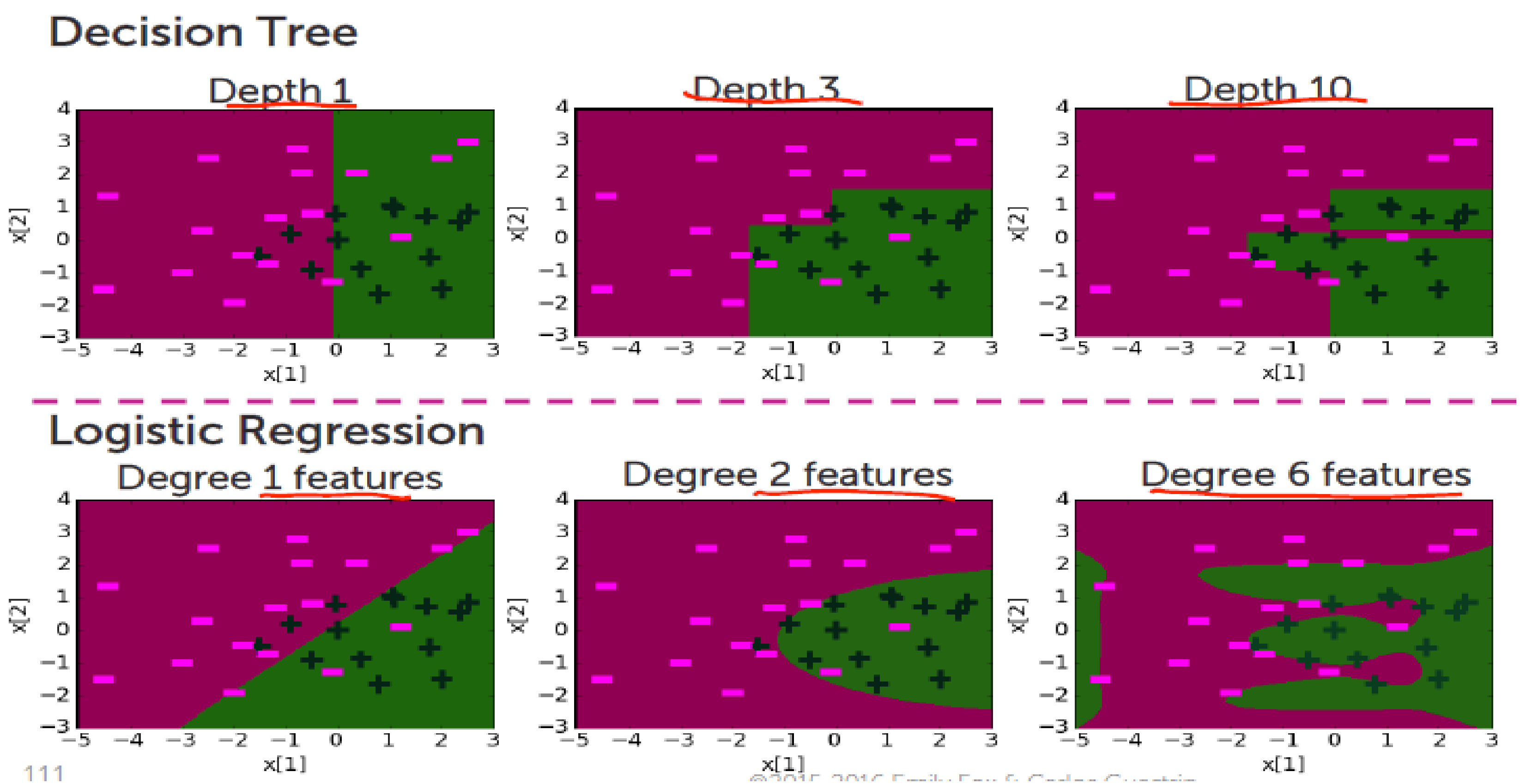
Decision tree vs logistic regression

Decision boundaries



Decision tree vs logistic regression

Comparing decision boundaries



What you can do now

- Define a decision tree classifier
- Interpret the output of a decision trees
- Learn a decision tree classifier using greedy algorithm
- Traverse a decision tree to make predictions
  - Majority class predictions
  - Probability predictions
  - Multiclass classification

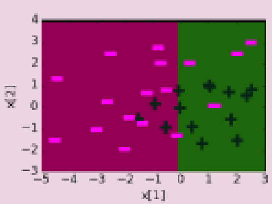
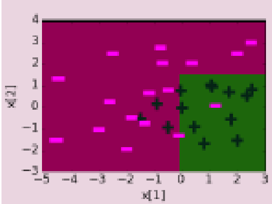
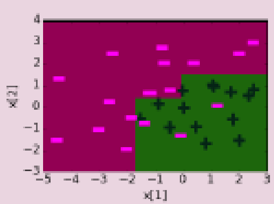
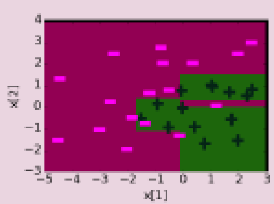
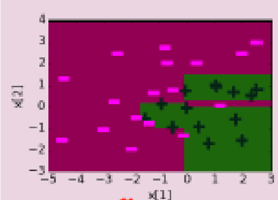
Overfitting in decision trees

Overfitting in decision tree

What happens when we increase depth?

Training error reduces with depth →

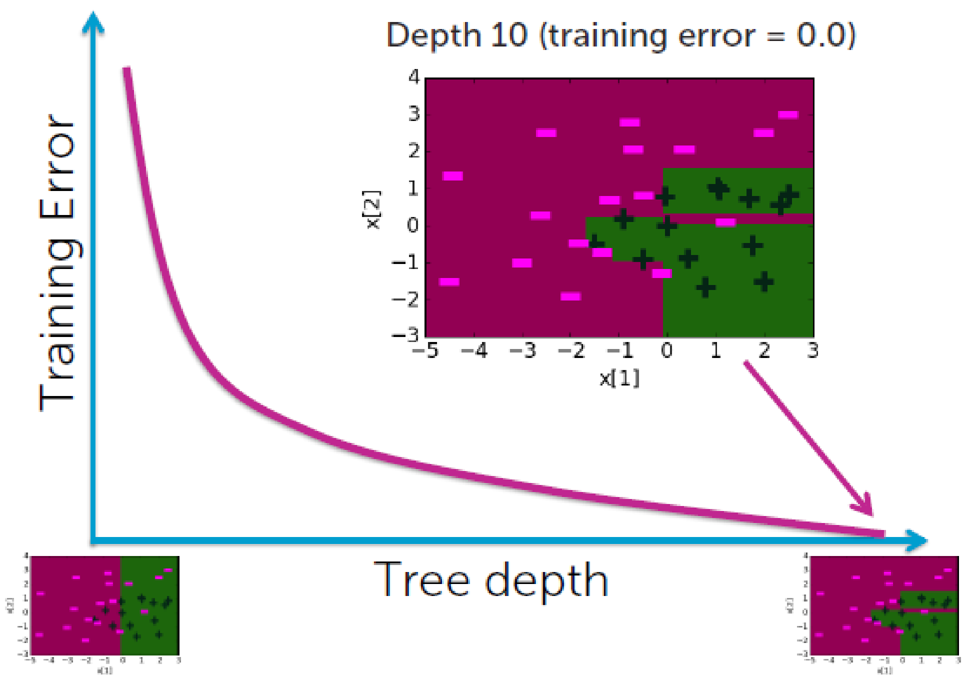
Big warning!!

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 →

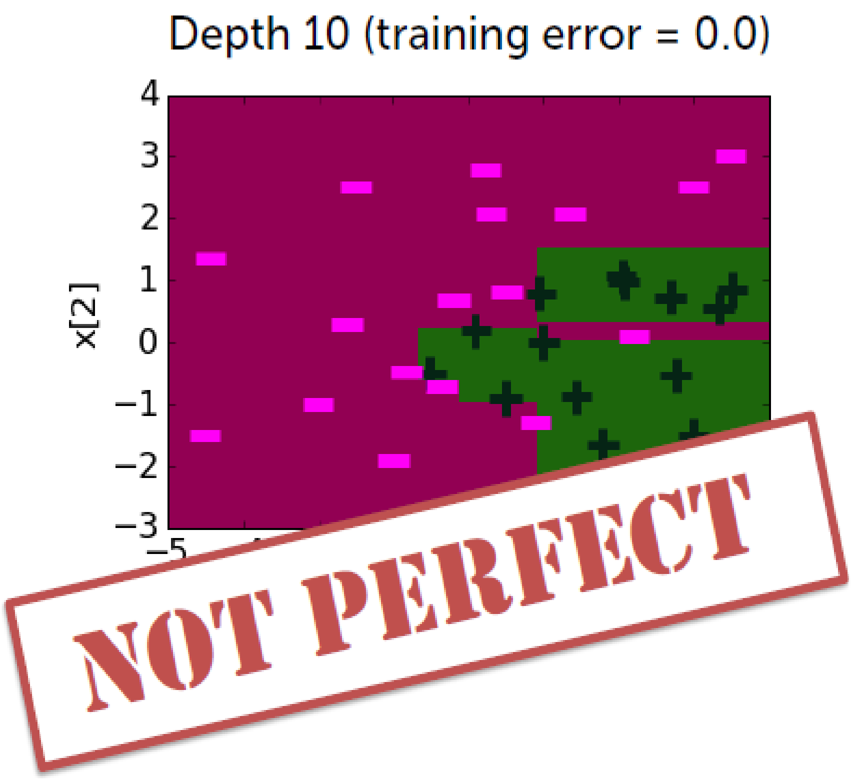
Overfitting in decision tree

Deeper trees ➔ lower training error



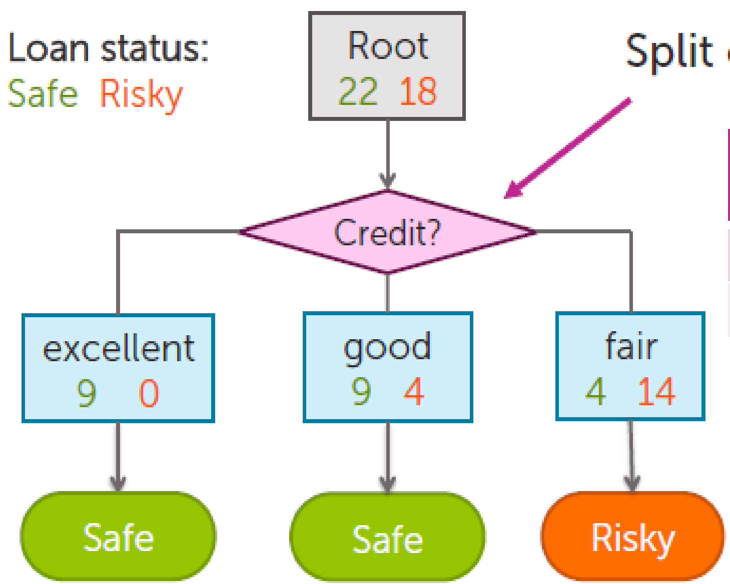
Overfitting in decision tree

Training error = 0: Is this model perfect?



Overfitting in decision tree

Why training error reduces with depth?



Split on **credit**

Tree	Training error
(root)	0.45
split on credit	0.20

Training error improved by 0.25 because of the split

Overfitting in decision tree

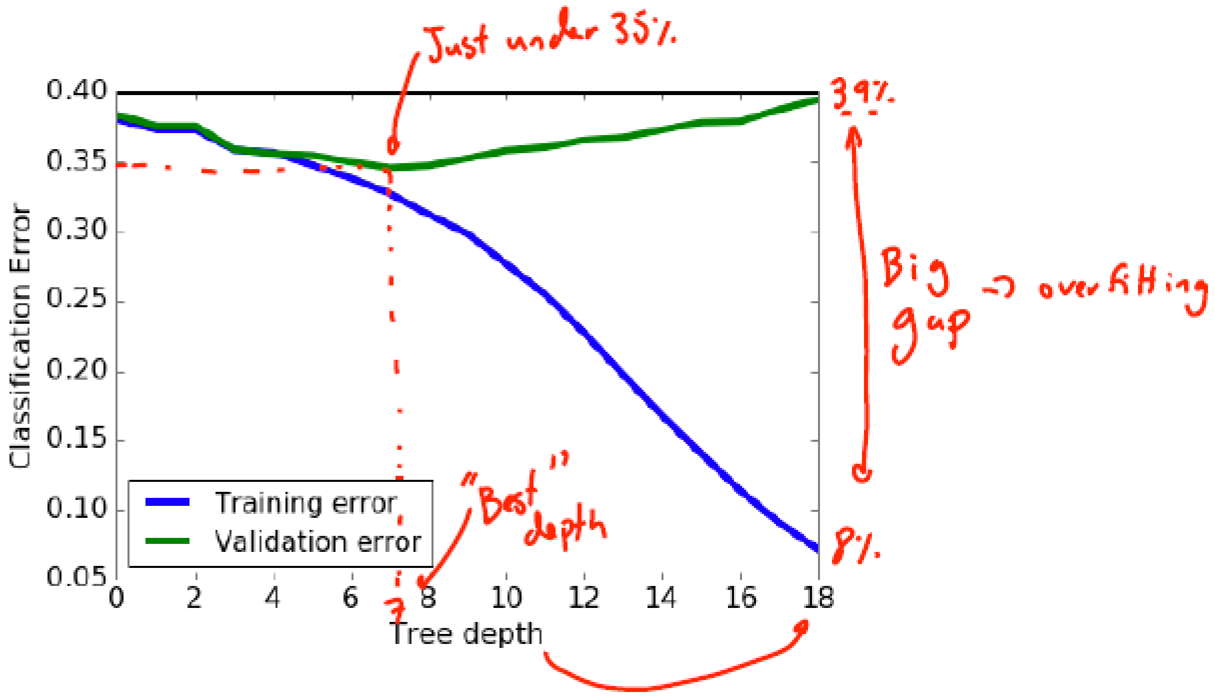
Feature split selection algorithm

- Given a subset of data M (a node in a tree)
- For each feature $h_i(x)$:
  1. Split data of M according to feature $h_i(x)$
  2. Compute classification error split
- Chose feature $h^*(x)$ with lowest classification error

By design, each split reduces training error

Overfitting in decision tree

Decision trees overfitting on loan data



Simplest tree is better

Principle of *Occam's Razor*

"Among competing hypotheses, the one with fewest assumptions should be selected", William of Occam, 13th Century

Symptoms: S_1 and S_2

Diagnosis 1: 2 diseases

Two diseases D_1 and D_2 where D_1 explains S_1, D_2 explains S_2

OR

SIMPLER

Diagnosis 2: 1 disease

Disease D_3 explains both symptoms S_1 and S_2

Simplest tree is better

Occam's Razor for decision trees

When two trees have similar classification error on the validation set, pick the simpler one

Complexity	Train error	Validation error
Simple	0.23	0.24
Moderate	0.12	0.15
Complex	0.07	0.15
Super complex	0	0.18

Same validation error

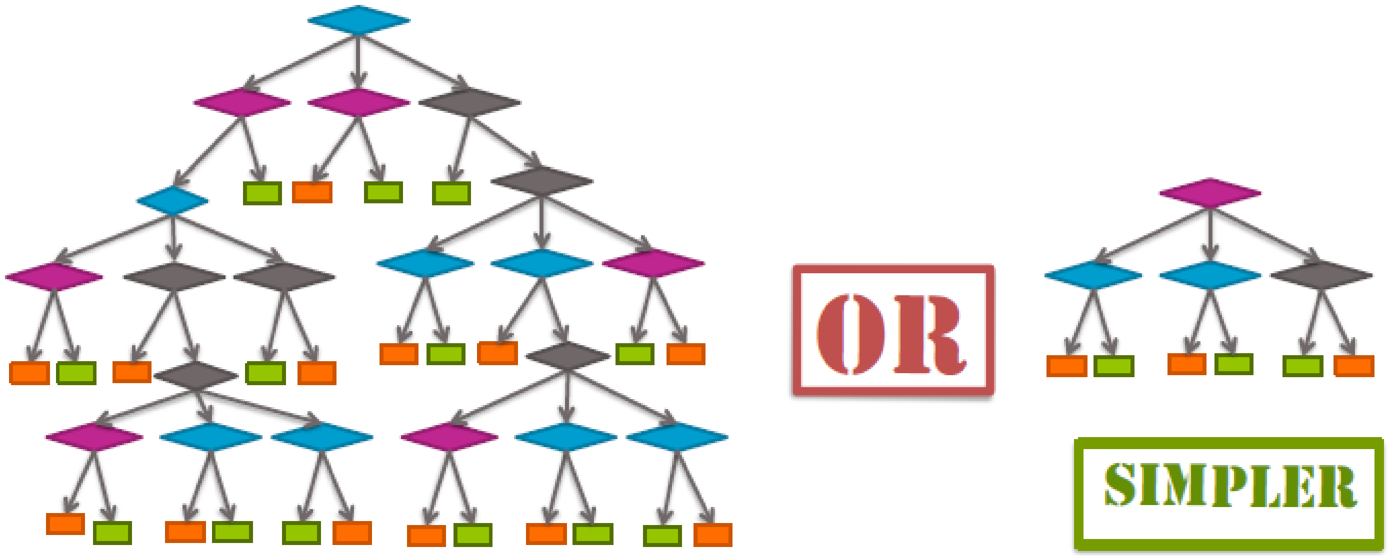
pick

bad!

Overfit

Simplest tree is better

Which tree is simpler?



Simplest tree is better

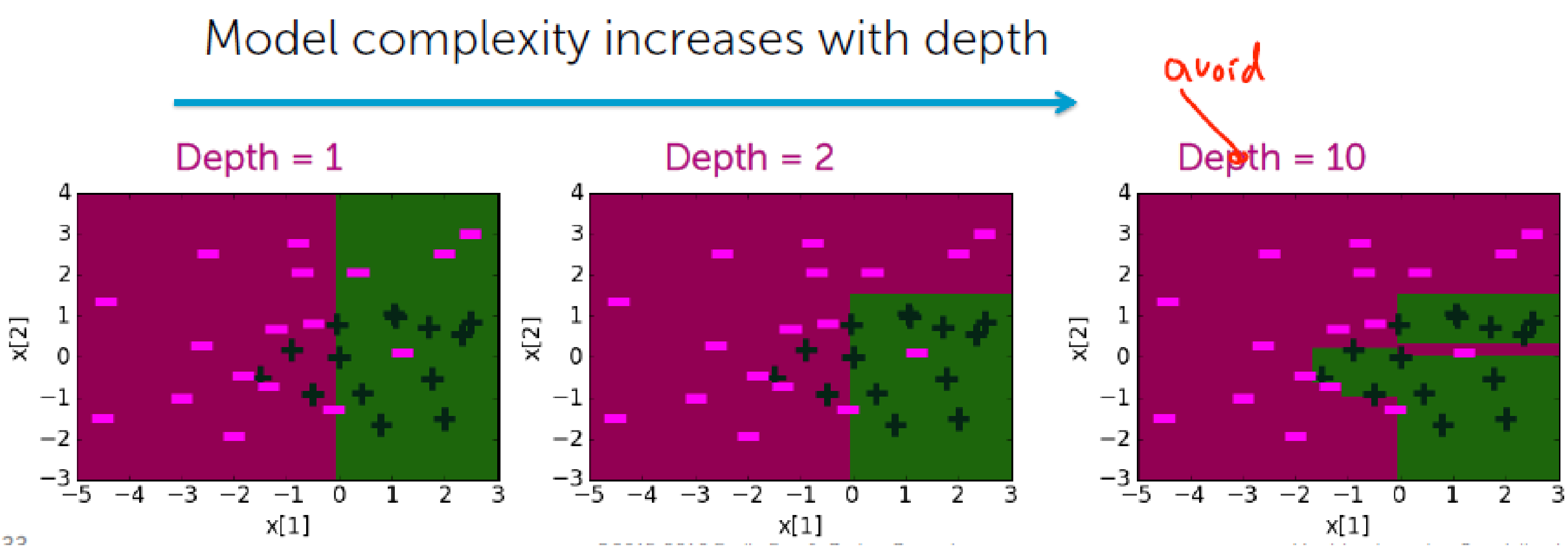
How do we pick simpler trees?

1. **Early Stopping:** Stop learning algorithm **before** tree become too complex

2. **Pruning:** Simplify tree **after** learning algorithm terminates

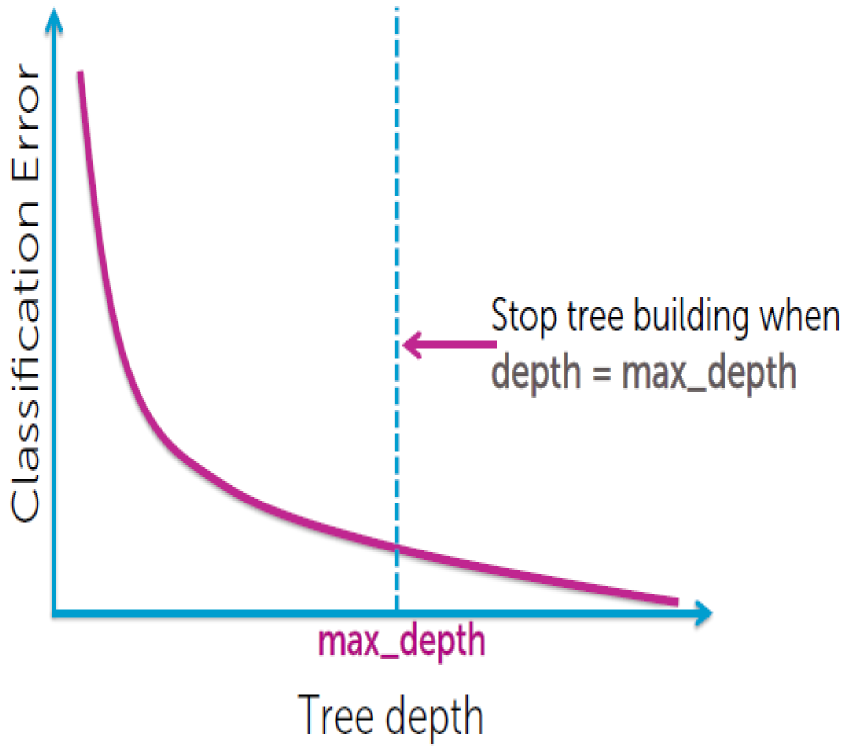
Early stopping for learning decision trees

Deeper trees → Increasing complexity

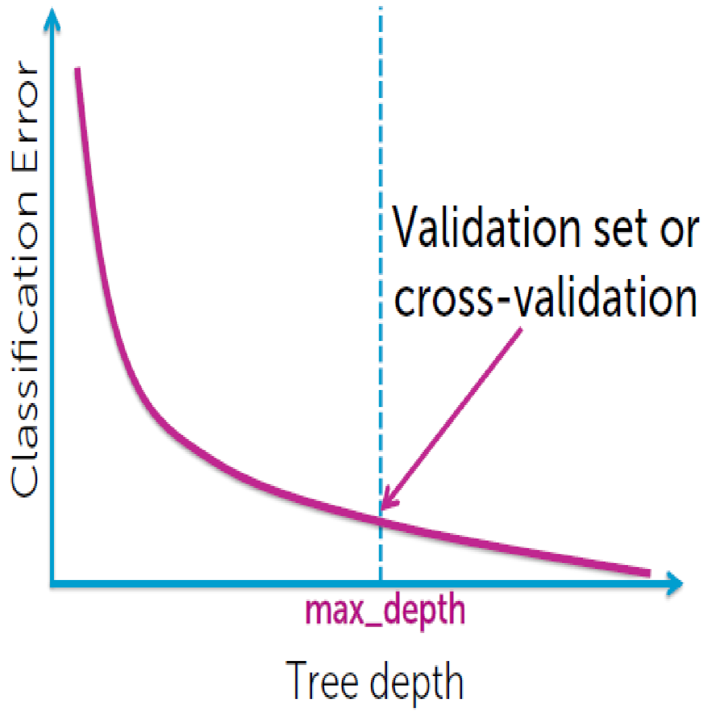


Early stopping condition 1

Limit depth of tree

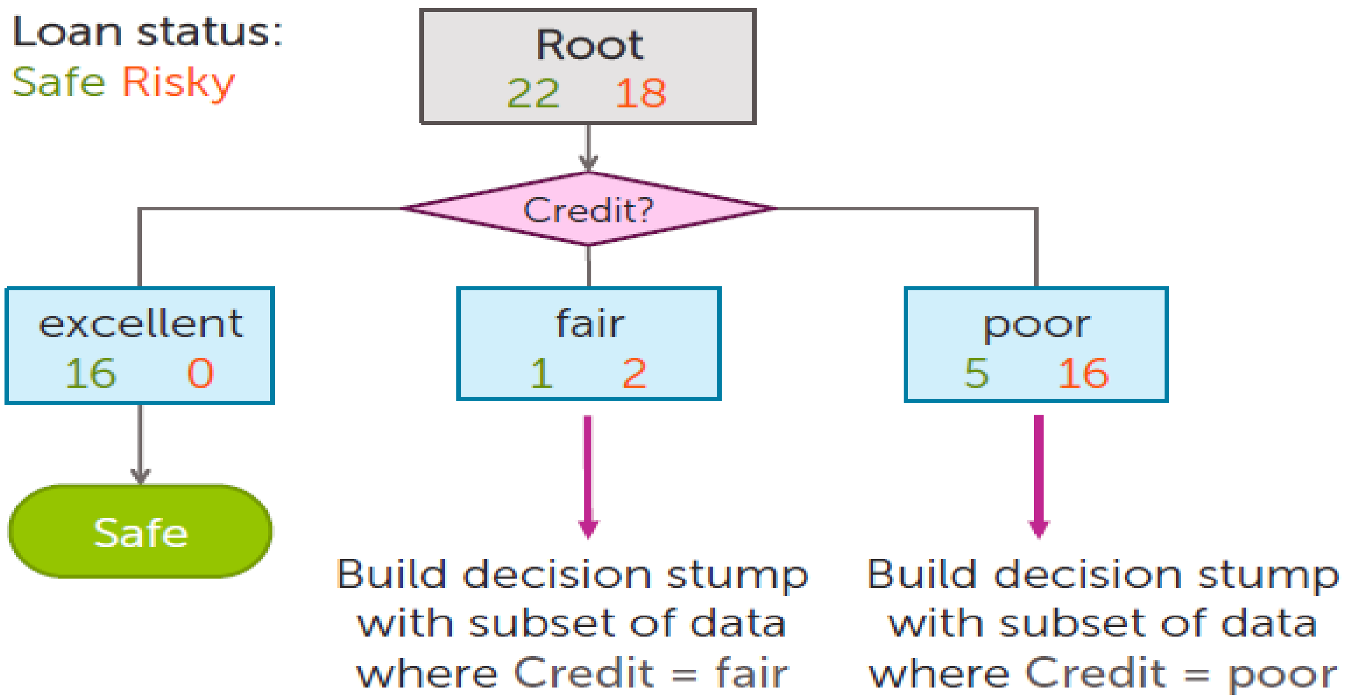


Picking value for max_depth???



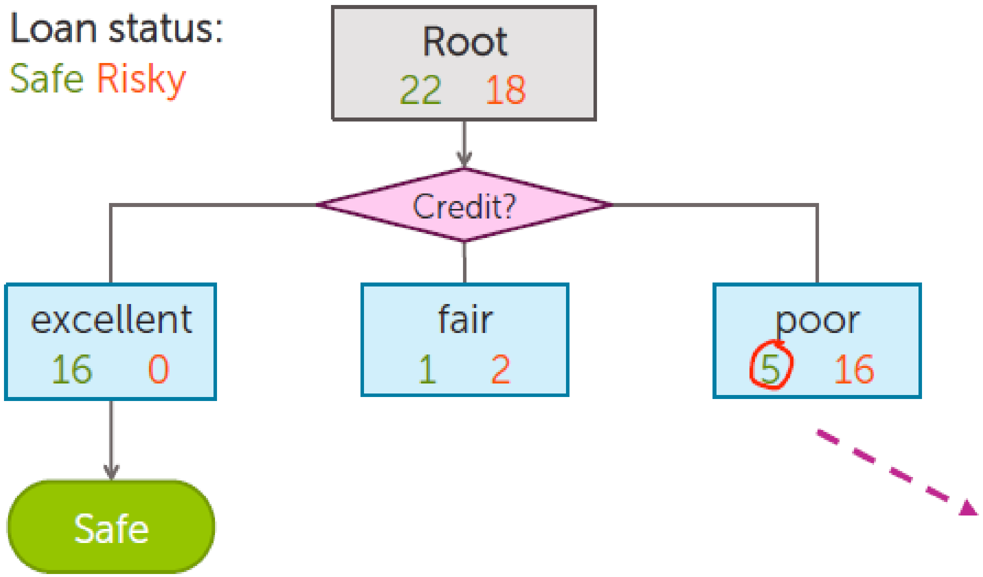
Early stopping condition 2

Decision tree recursion review



Early stopping condition 2

Split selection for credit=poor

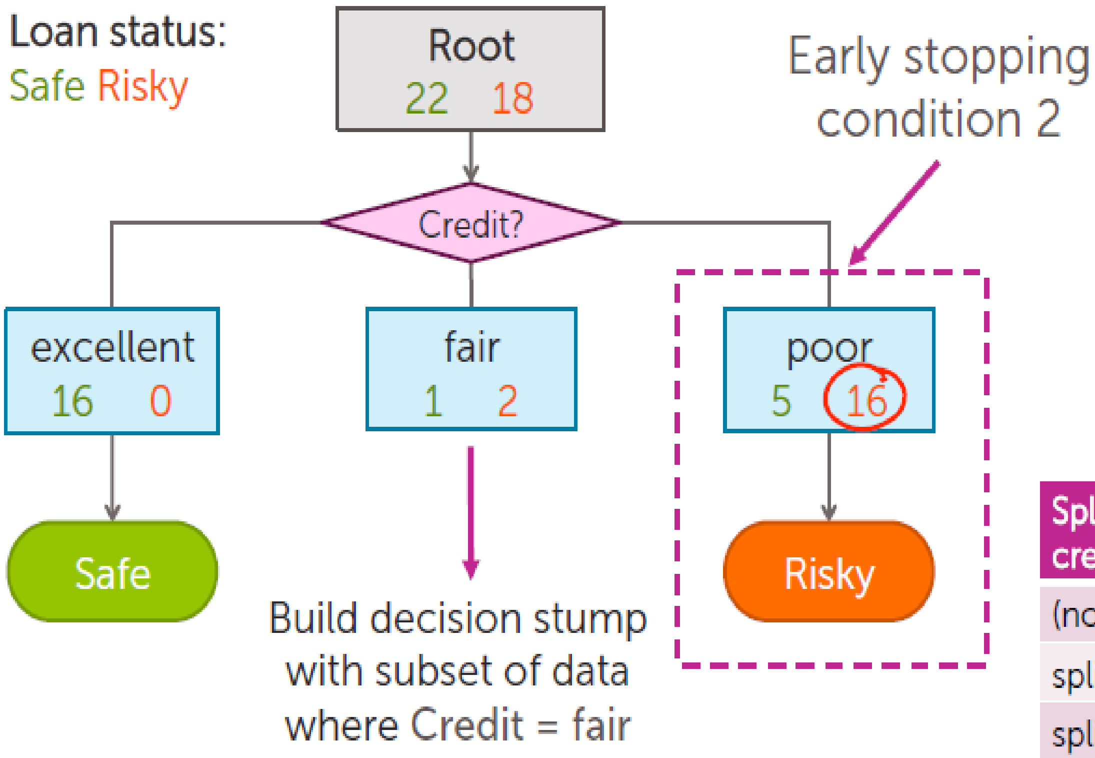


No split improves classification error
➔ Stop!

Splits for credit=poor	Classification error
(no split)	0.24
split on term	0.24
split on income	0.24

Early stopping condition 2

No split improves classification error

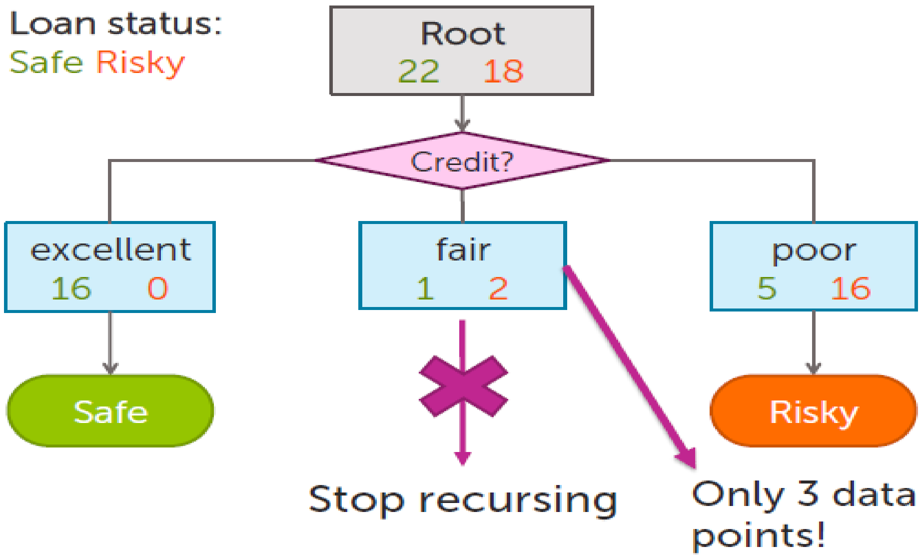


Splits for credit=poor	Classification error
(no split)	0.24
split on term	0.24
split on income	0.24

Early stopping condition 3

Stop if number of data points contained in a node is too small

Can we trust nodes with very few points?



Early stopping: Summary

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

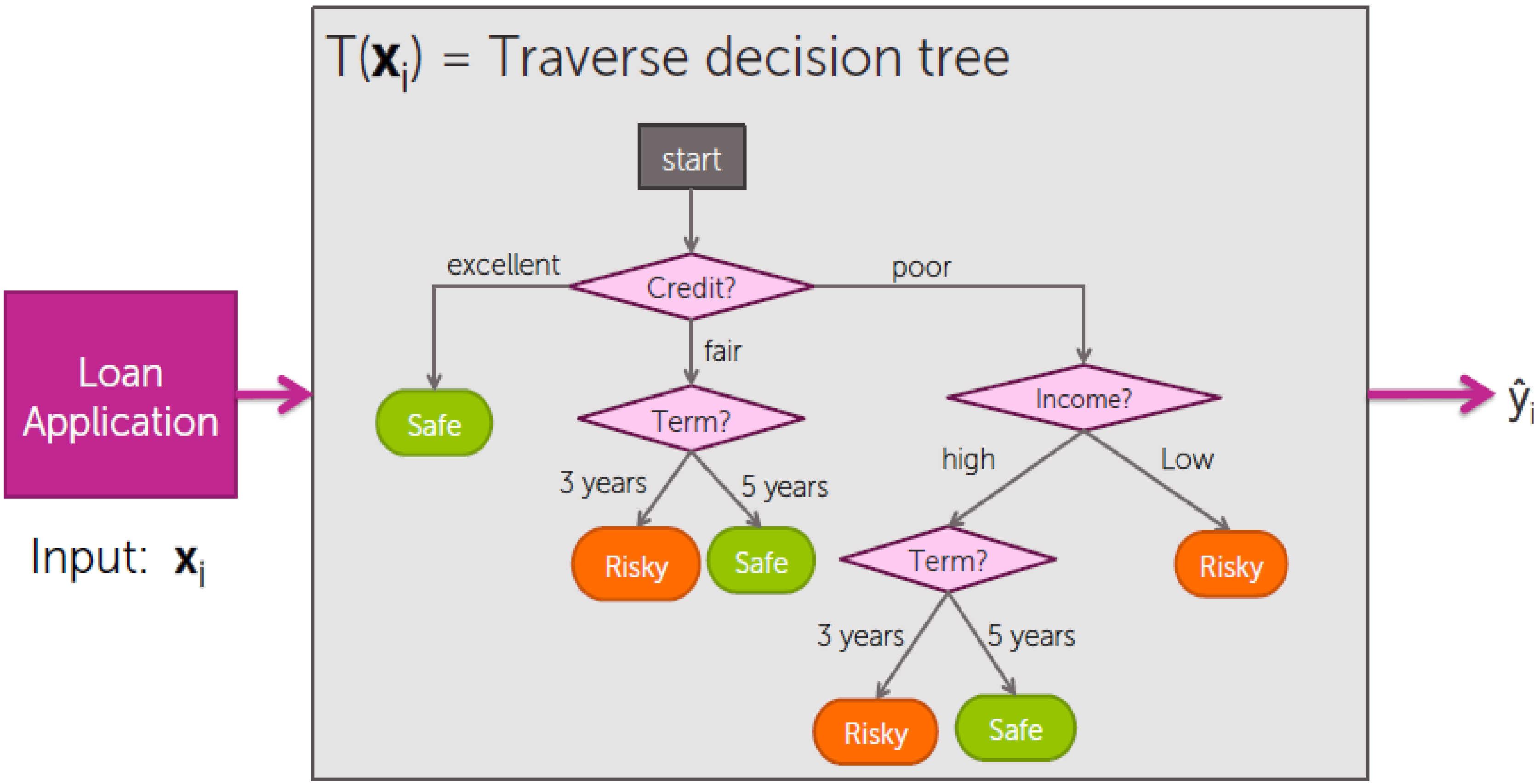
or

Early stopping conditions 1, 2 & 3

Recursion

Strategies for handling missing data

Decision tree review



Missing data

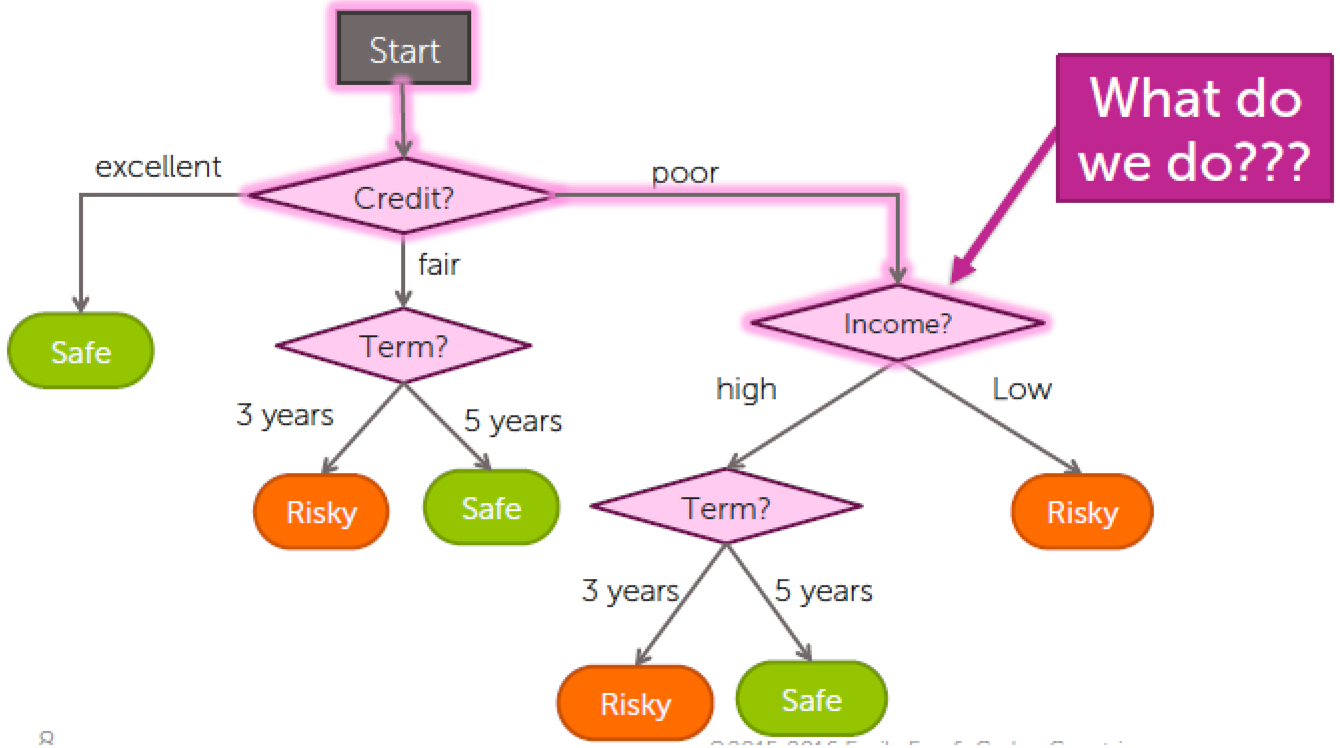
Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	?	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	high	safe
poor	?	high	risky
poor	5 yrs	low	safe
fair	?	high	safe

Loan application may be 3 or 5 years

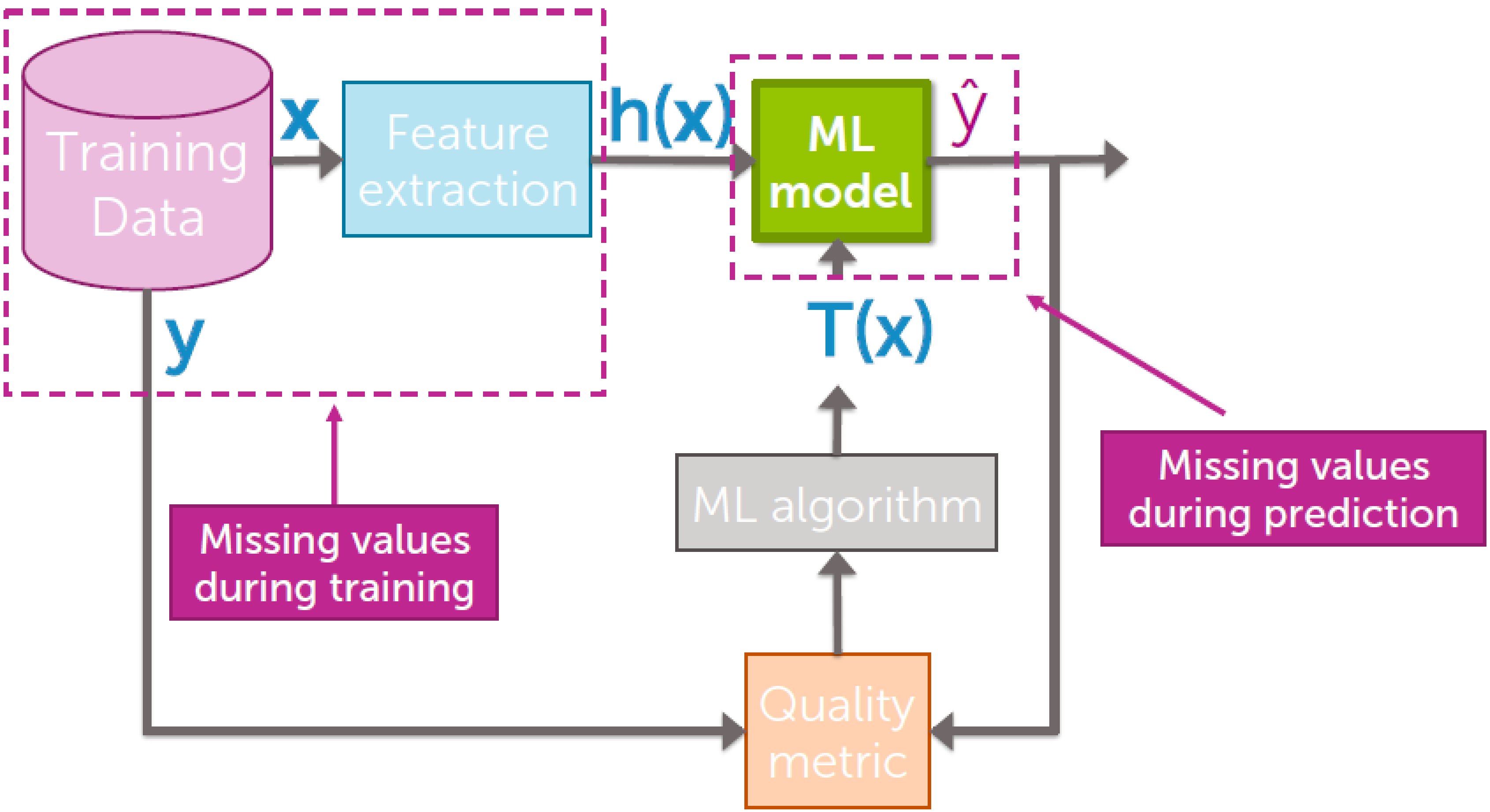
1. Training data: Contains "unknown" values
2. Predictions: Input at prediction time contains "unknown" values

Missing values during predictions

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

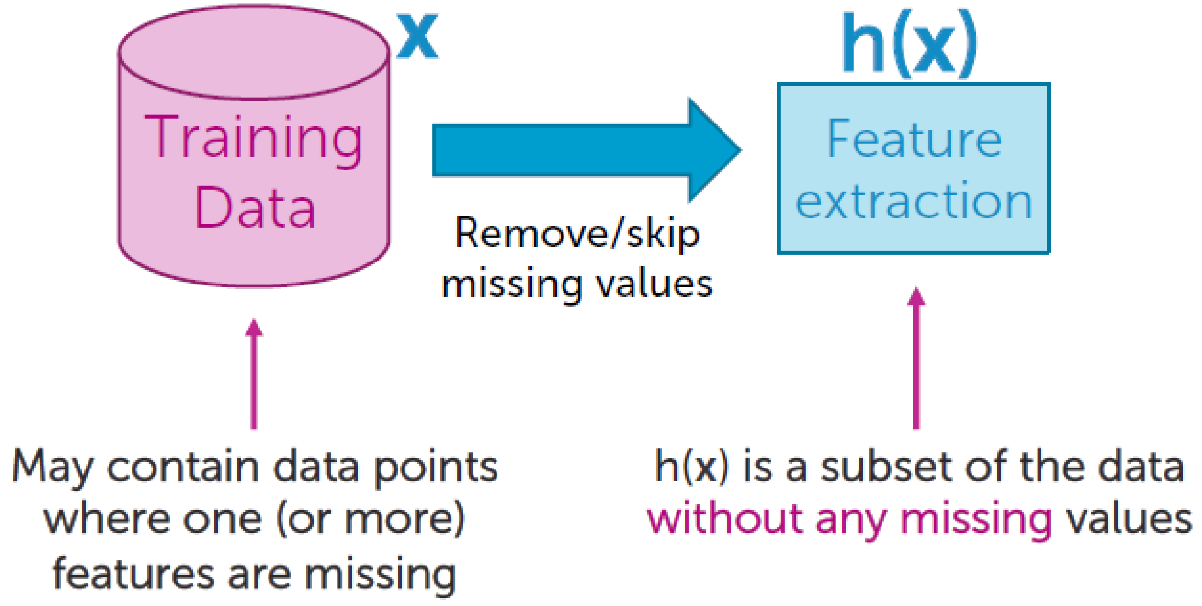


Missing values



Handling missing data

Idea 1: Purification by skipping/removing



Handling missing data

Idea 1: Skip data points with missing values

x

N = 9, 3 features

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	low	risky
poor	3 yrs	?	safe
fair	?	high	safe

Skip data points with missing values →

h(x)

N = 6, 3 features

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

Handling missing data

The challenge with Idea 1

x

N = 9, 3 features

Warning: More than 50% of the loan terms are unknown!

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

Skip data points with missing values

much much smaller

h(x)

N = 3, 3 features

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

Missing data

Idea 2: Skip features with missing values

x

N = 9, 3 features

fewer features

h(x)

N = 9, 2 features

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

Skip features with many missing values

Credit	Income	y
excellent	high	safe
fair	low	risky
fair	high	safe
poor	high	risky
excellent	low	risky
fair	high	safe
poor	high	risky
poor	low	safe
fair	high	safe

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

Missing value skipping: Pros and Cons

Pros

- Easy to understand and implement
- Can be applied to any model (decision trees, logistic regression, linear regression,…)

Cons

- Removing data points and features may remove important information from data
- Unclear when it's better to remove data points versus features
- Doesn't help if data is missing at prediction time

Data is precious

Main drawback of skipping strategy

x

N = 9, 3 features

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	low	risky
poor	?	low	safe
fair	?	high	safe

Data is precious, don't throw it away!

Skip data points with missing values

h(x)

N = 6, 3 features

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

Data is precious

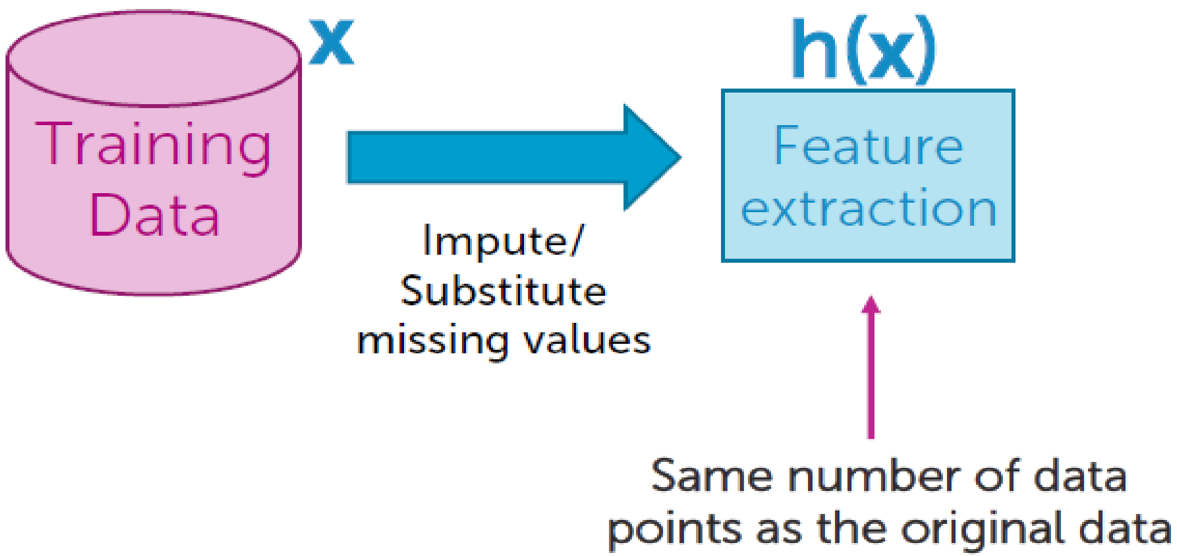
Can we keep all the data?

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

Use other data points in **x** to "guess" the "?"

Handling mising data

Idea 2: Purification by imputing



Handling mising data

Idea 2: Imputation/Substitution

N = 9, 3 features

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

Fill in each missing value with a calculated guess

N = 9, 3 features

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	3 yrs	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	3 yrs	low	safe
fair	3 yrs	high	safe

Example

Example: Replace ? with most common value

3 year loans: 4 ← Best guess (Simple)
5 year loans: 2

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

Purification by imputing →

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	3 yrs	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	3 yrs	low	safe
fair	3 yrs	high	safe

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

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

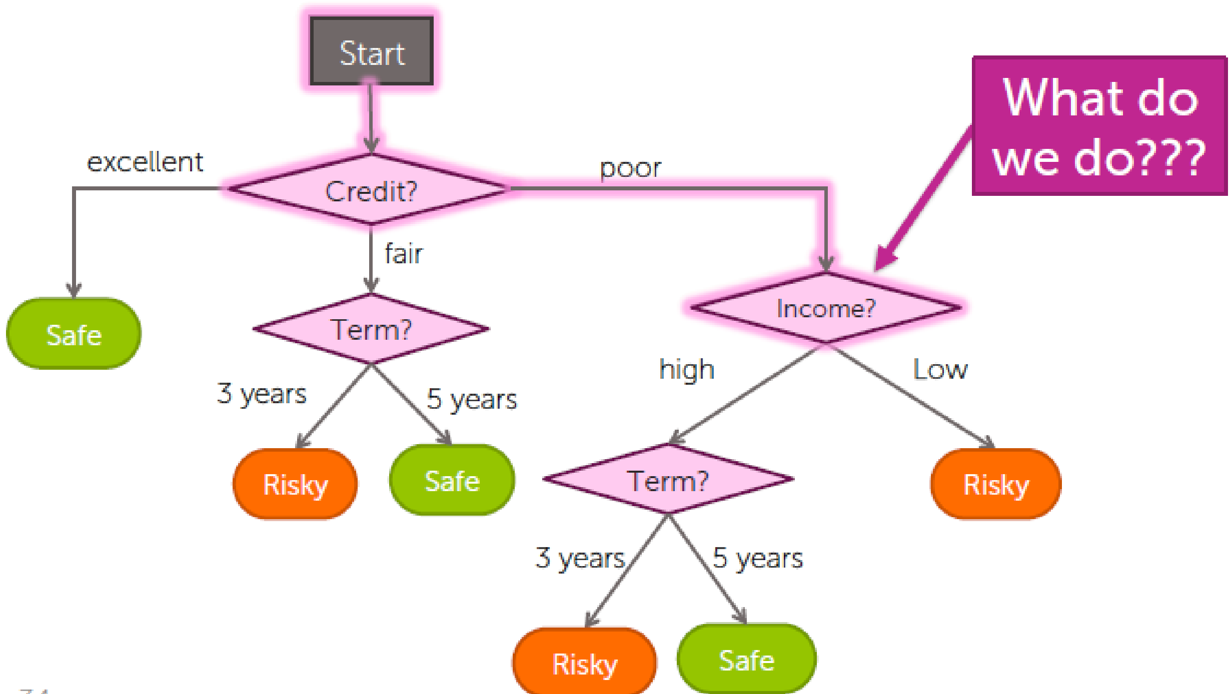
- May result in systematic errors

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

Strategy 3: addapt algorithm

Missing values during prediction: *revisited*

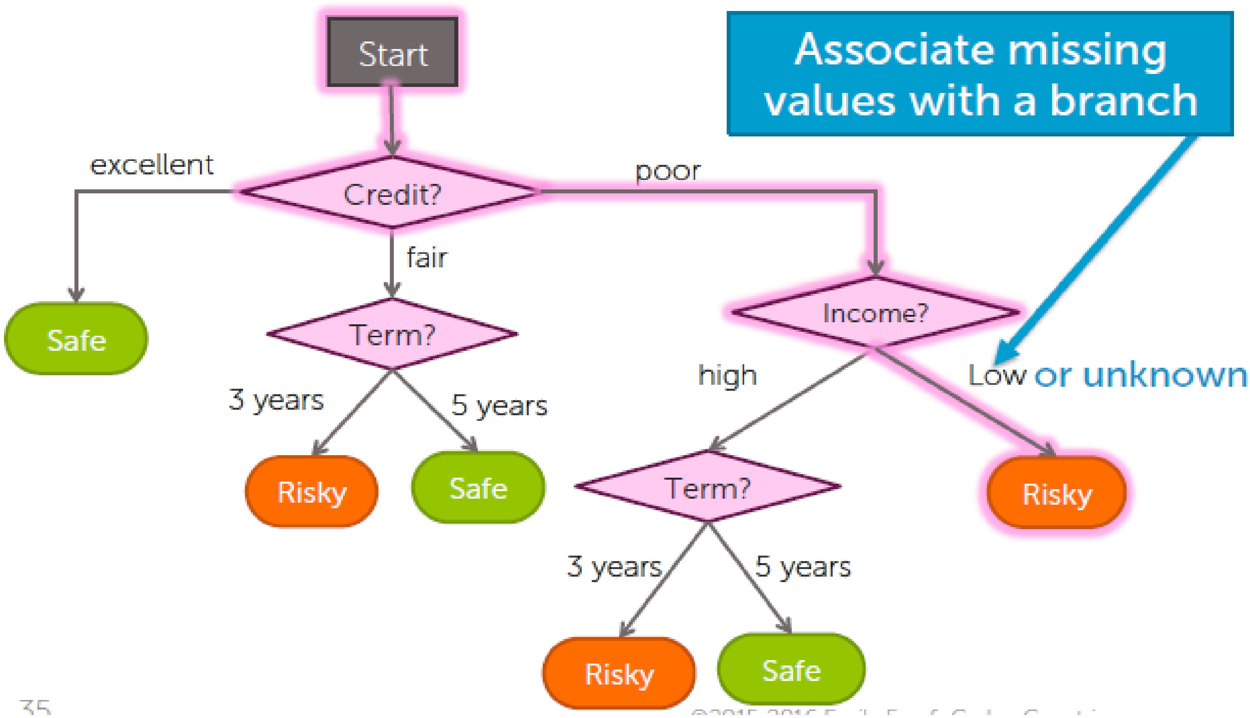
$\mathbf{x}_i$ = (Credit = poor, Income = ?, Term = 5 years)



Strategy 3: addapt algorithm

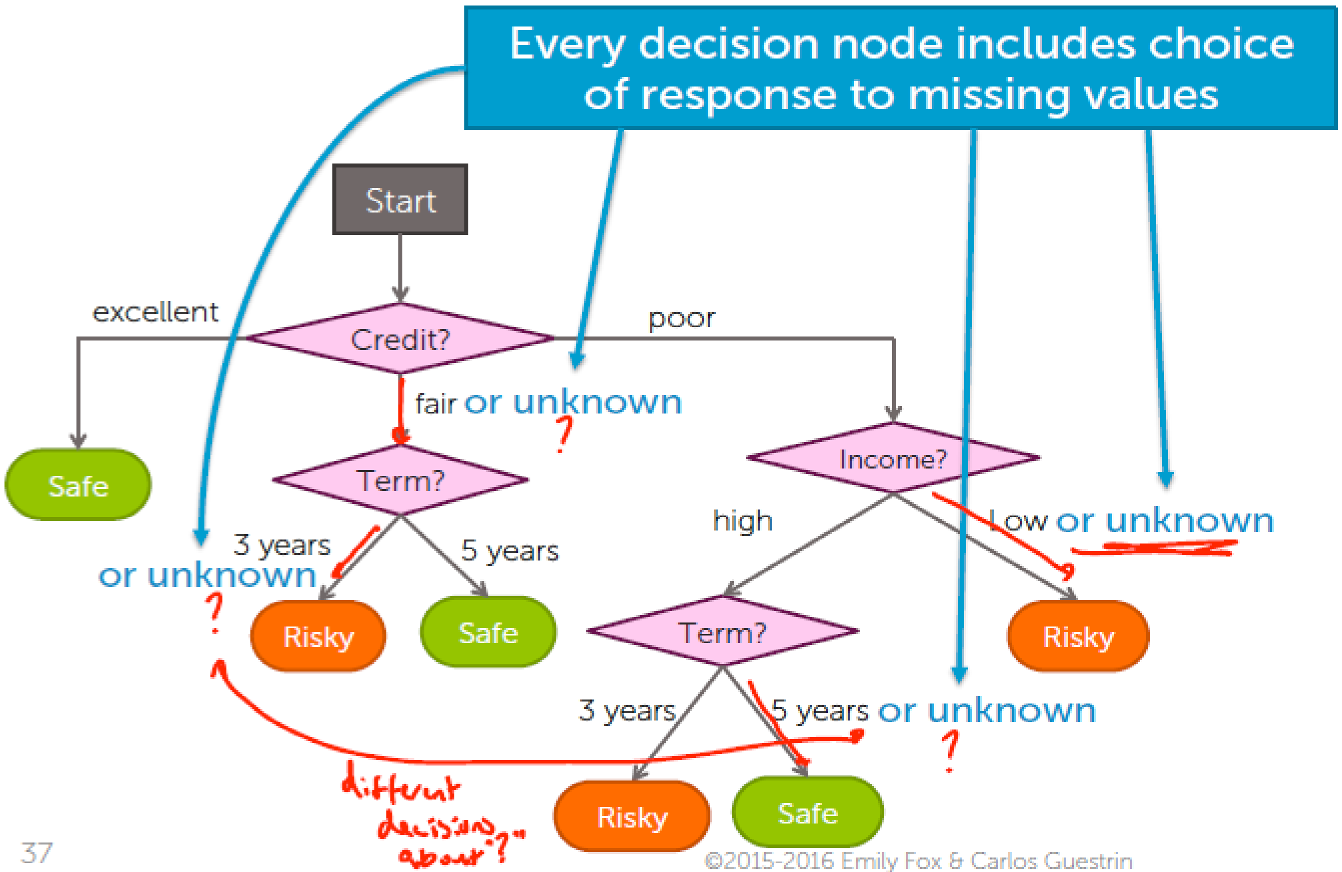
Add missing values to the tree definition

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



Strategy 3: addapt algorithm

Add missing value choice to every decision node

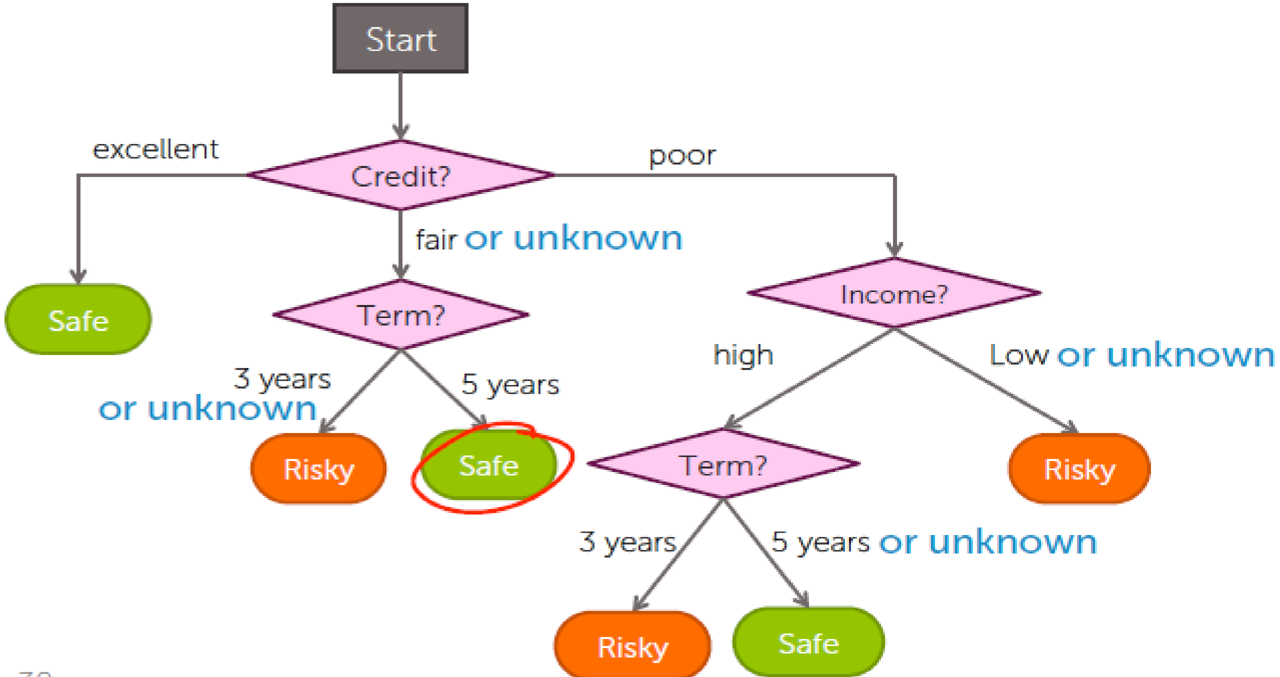


30/10, 6/11 2024

Strategy 3: addapt algorithm

Prediction with missing values becomes simple

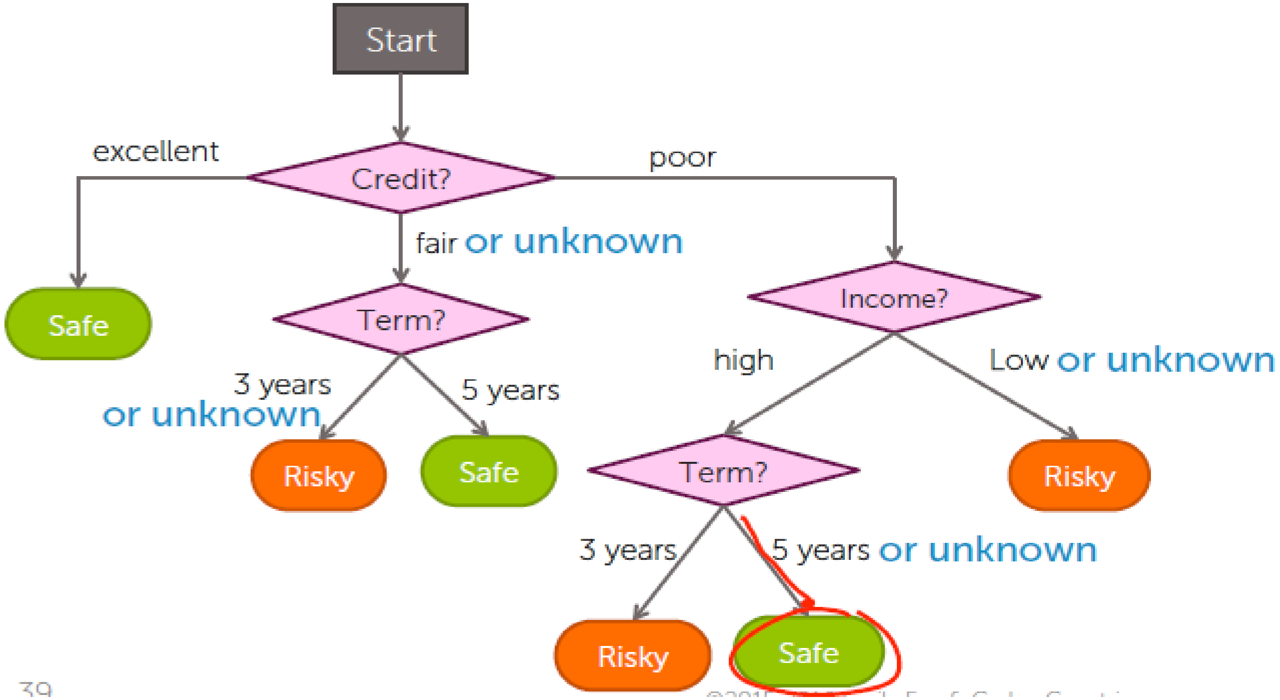
$\mathbf{x}_i$ = (Credit = ?, Income = high, Term = 5 years)



Strategy 3: addapt algorithm

Prediction with missing values becomes simple

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



Strategy 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

Feature split selection with missing data

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

Pick feature split leading to lowest classification error

Must select feature & branch for missing values!

Feature split selection with missing data

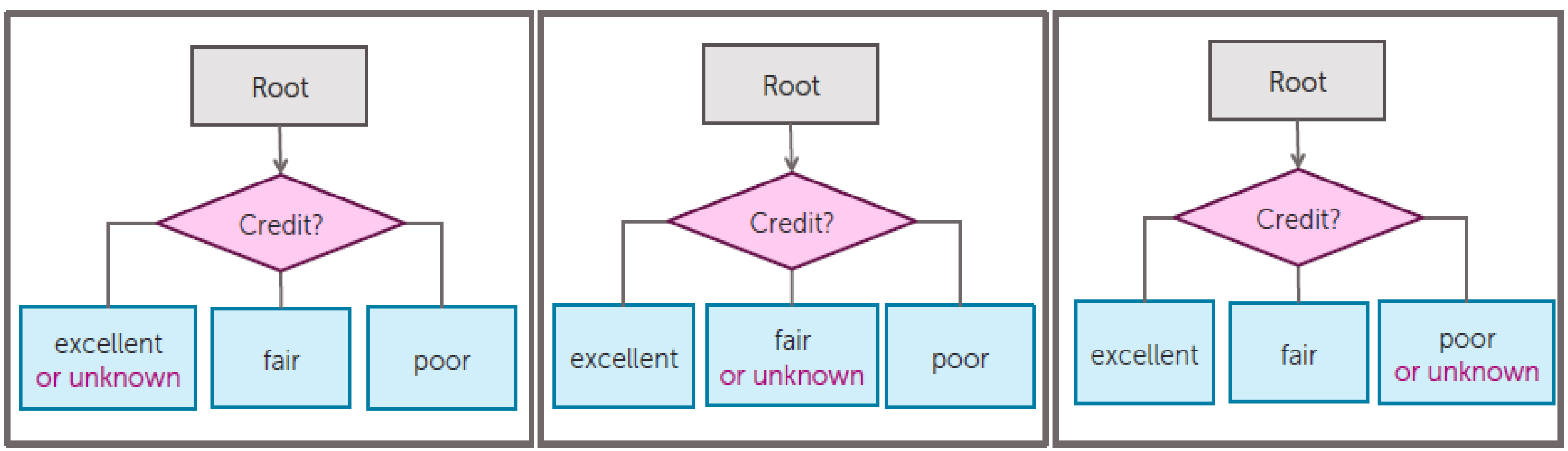
Should missing go left, right, or middle?

Choose branch that leads to lowest classification error!

Choice 1: Missing values go with Credit=excellent

Choice 2: Missing values go with Credit=fair

Choice 3: Missing values go with Credit=poor



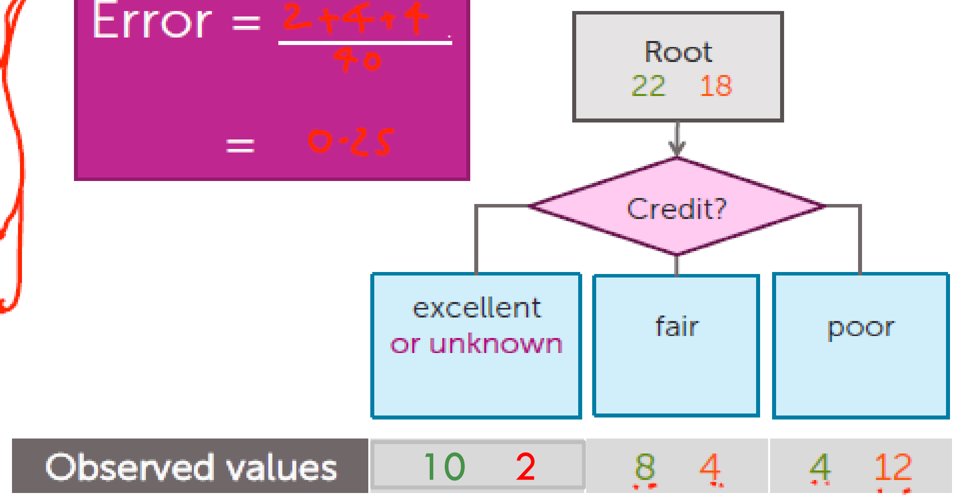
Feature split selection with missing data

Computing classification error of decision stump with missing data

N = 40, 3 features

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

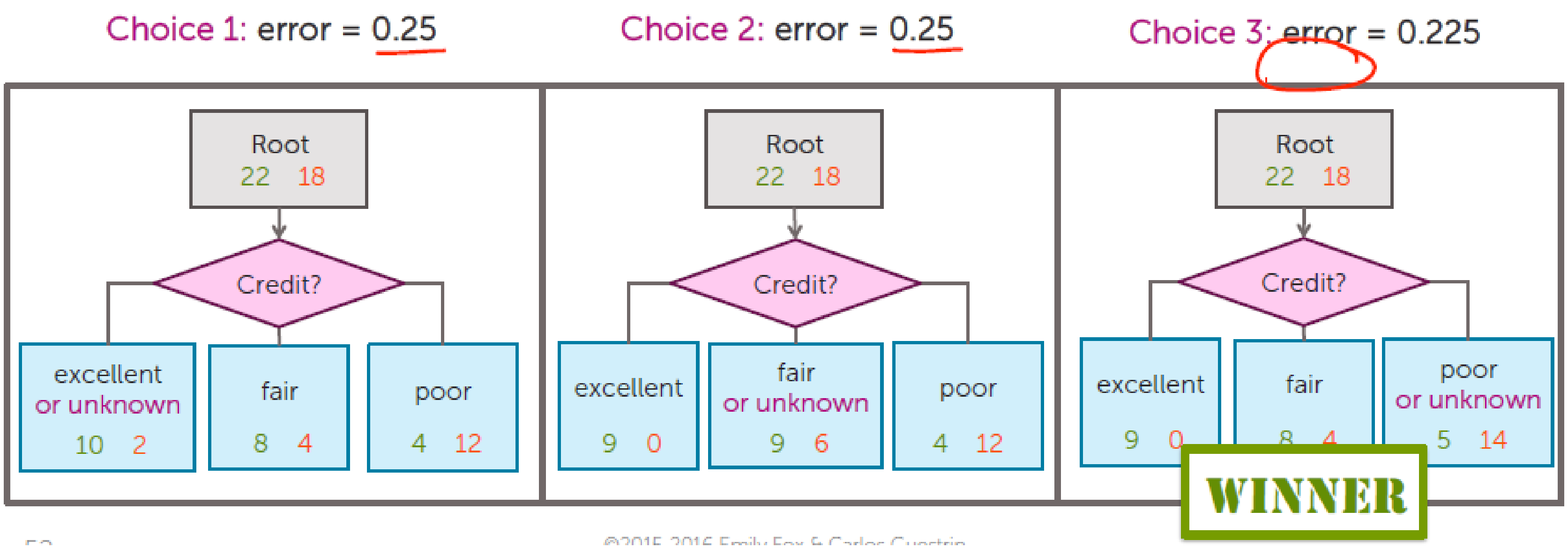
$$\text{Error} = \frac{2+4+4}{40} = 0.25$$



Feature split selection with missing data

Use classification error to decide

Best choice → assign "unknown" to Credit = poor



Feature split selection with missing data

- Given a subset of data M (a node in a tree)
- For each feature $h_i(x)$:
  1. Split data points of M where $h_i(x)$ is *not* "unknown" according to feature $h_i(x)$
  2. Consider assigning data points with "unknown" value for $h_i(x)$ to each branch
     A. Compute classification error split & branch assignment of "unknown" values
- Chose feature $h^*(x)$ & branch assignment of "unknown" with lowest classification error

What can you do now

Describe common ways to handling missing data:

1. Skip all rows with any missing values
2. Skip features with many missing values
3. Impute missing values using other data points

Modify learning algorithm (decision trees) to handle missing data:

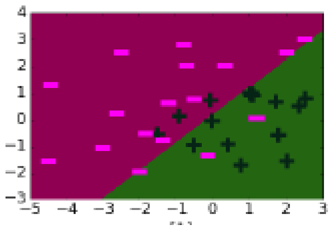
1. Missing values get added to one branch of split
2. Use classification error to determine where missing values go

Ensemble classifiers and boosting

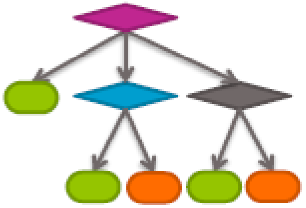
30/10, 6/11 2024

Simple classifiers

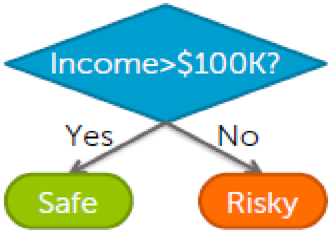
Simple (weak) classifiers are good!



Logistic regression w. simple features



Shallow decision trees

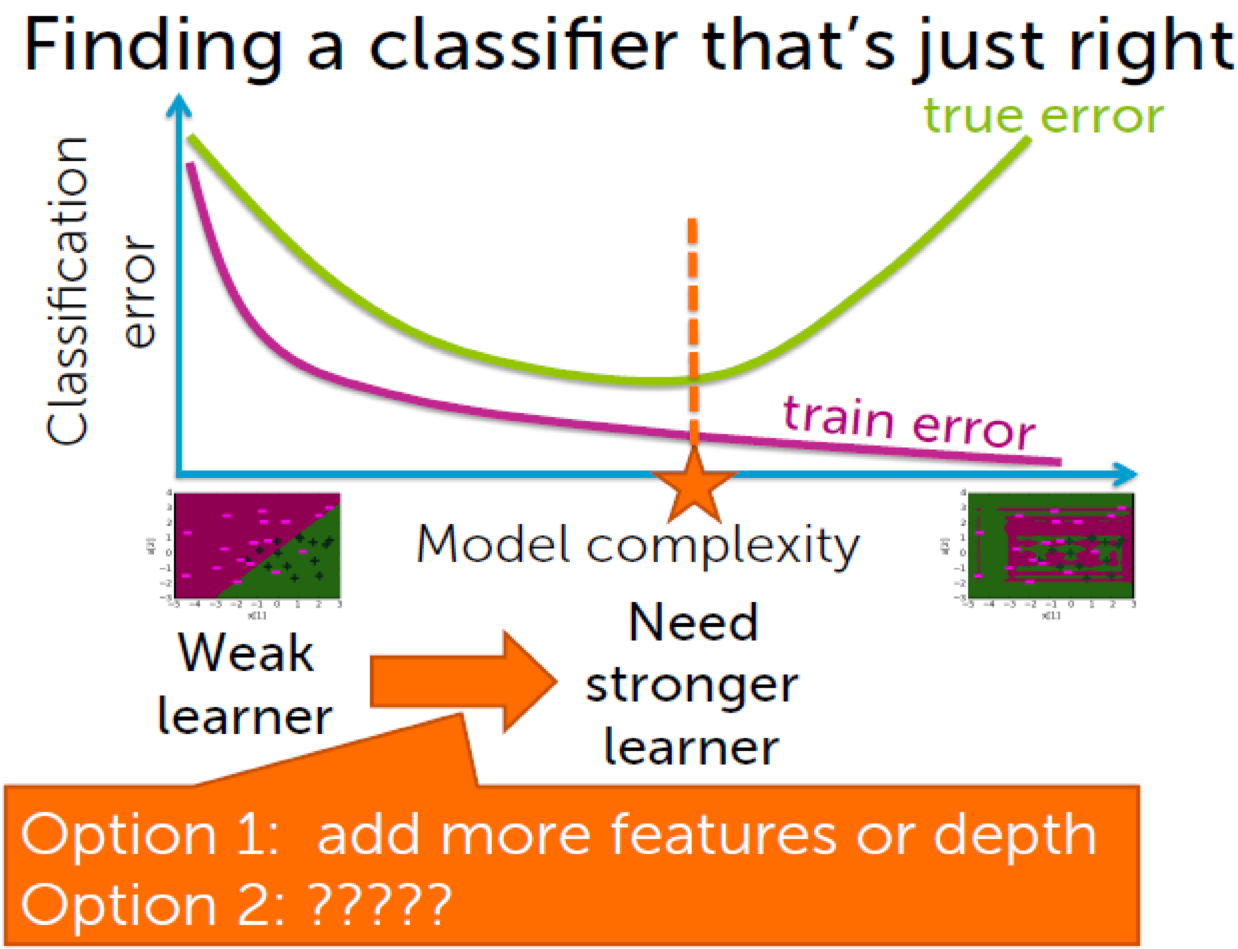


Decision stumps

Low variance. Learning is fast!

But high bias…

Simple classifiers



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)*

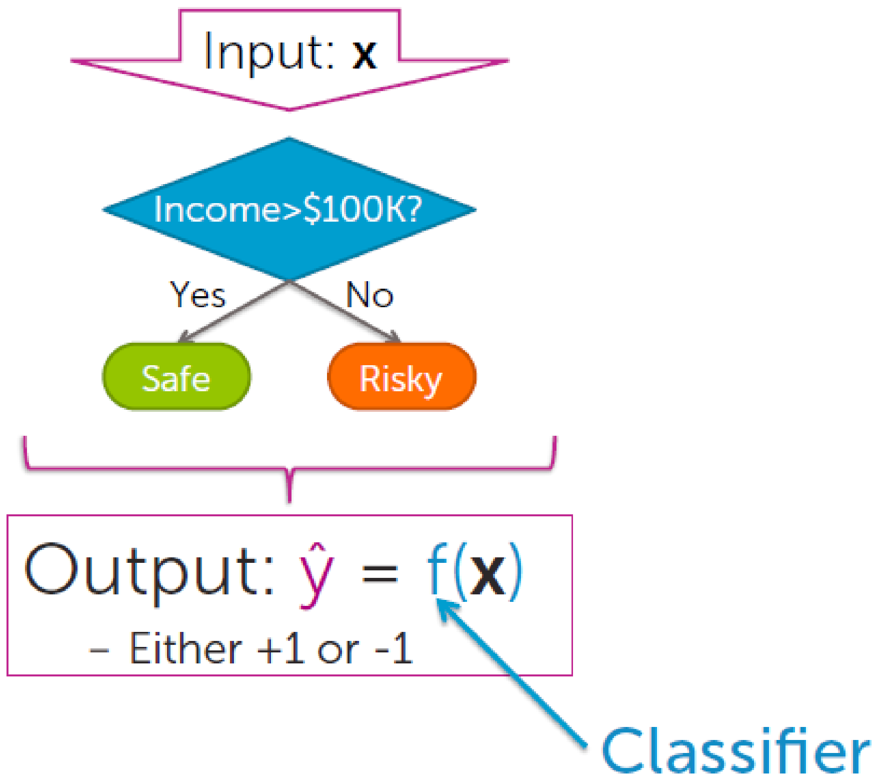
↓

Boosting

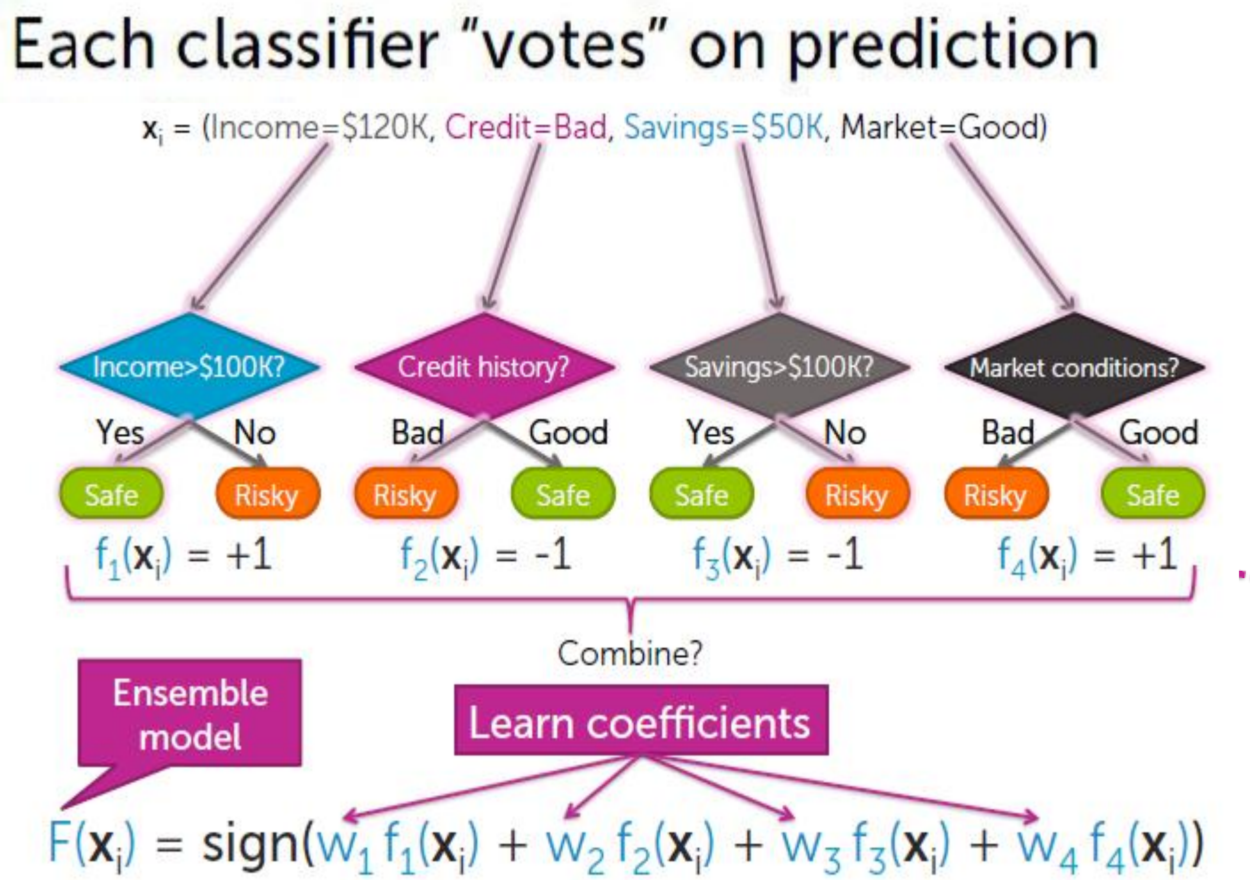
↓

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

A single classifier



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:

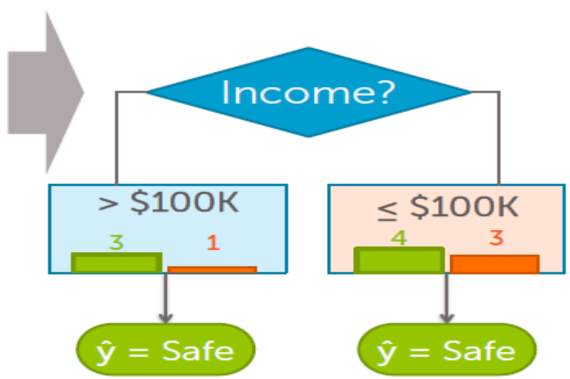
$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathbf{w}}_t f_t(\mathbf{x})\right)$$

Boosting

Training a classifier

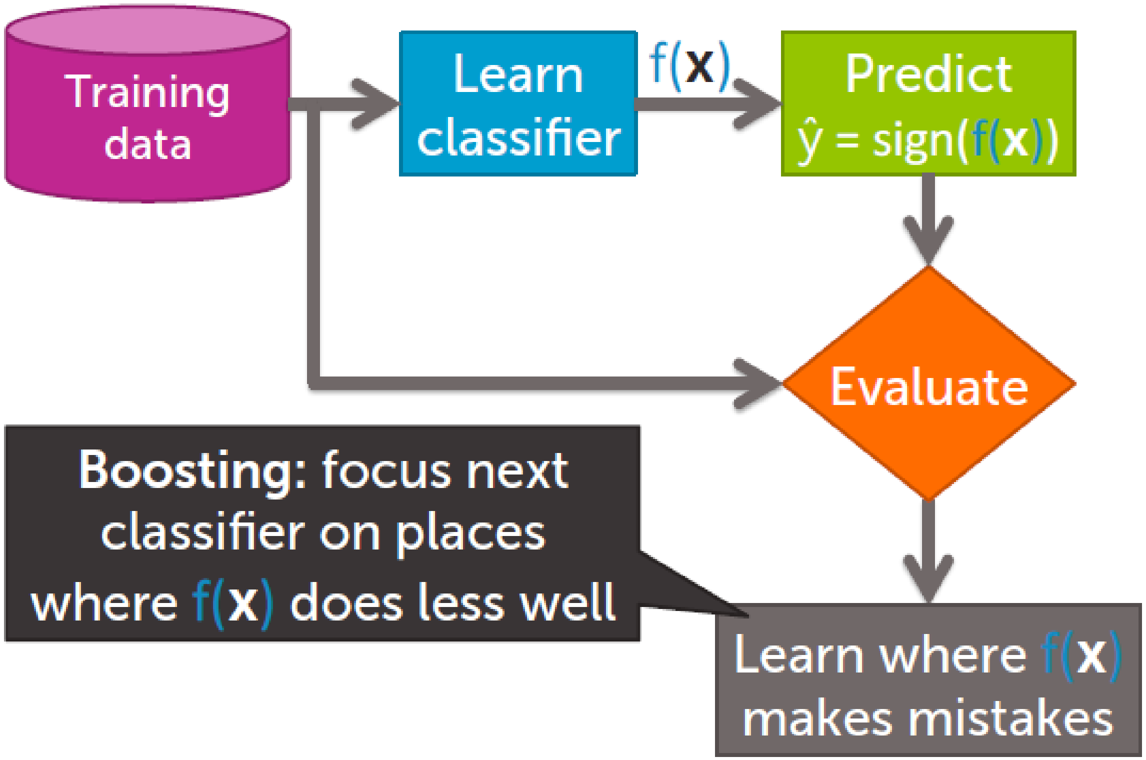
Learning decision stump

Credit	Income	y
A	$130K	Safe
B	$80K	Risky
C	$110K	Risky
A	$110K	Safe
A	$90K	Safe
B	$120K	Safe
C	$30K	Risky
C	$60K	Risky
B	$95K	Safe
A	$60K	Safe
A	$98K	Safe



Boosting

Boosting = Focus learning on "hard" points



Weighted data

Learning on weighted data:

More weight on "hard" or more important points

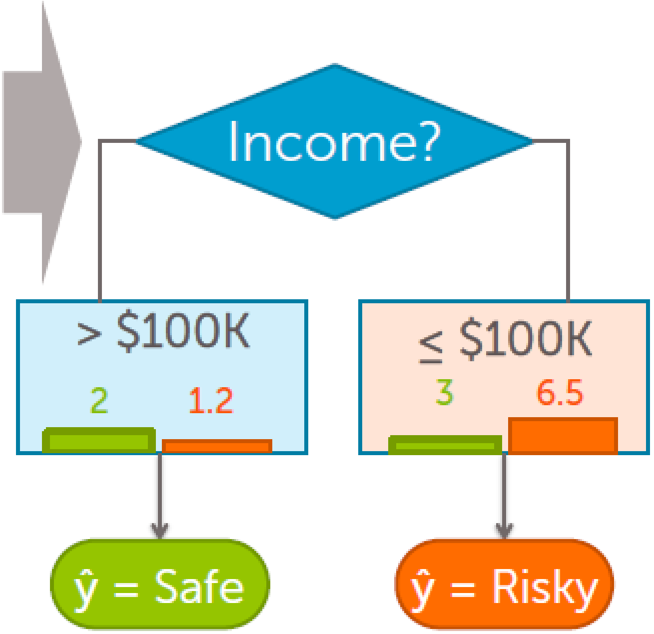
- Weighted dataset:
  - Each $\mathbf{x}_i, y_i$ weighted by α_i
    - More important point = higher weight α_i
- Learning:
  - Data point j counts as α_i data points
    - E.g., $\alpha_i = 2$ ➔ count point twice

Weighted data

Learning a decision stump on weighted data

Increase weight α of harder/ misclassified points

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	1.2
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9



Use sum over weights of the data points

Learning from weighted data in general

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

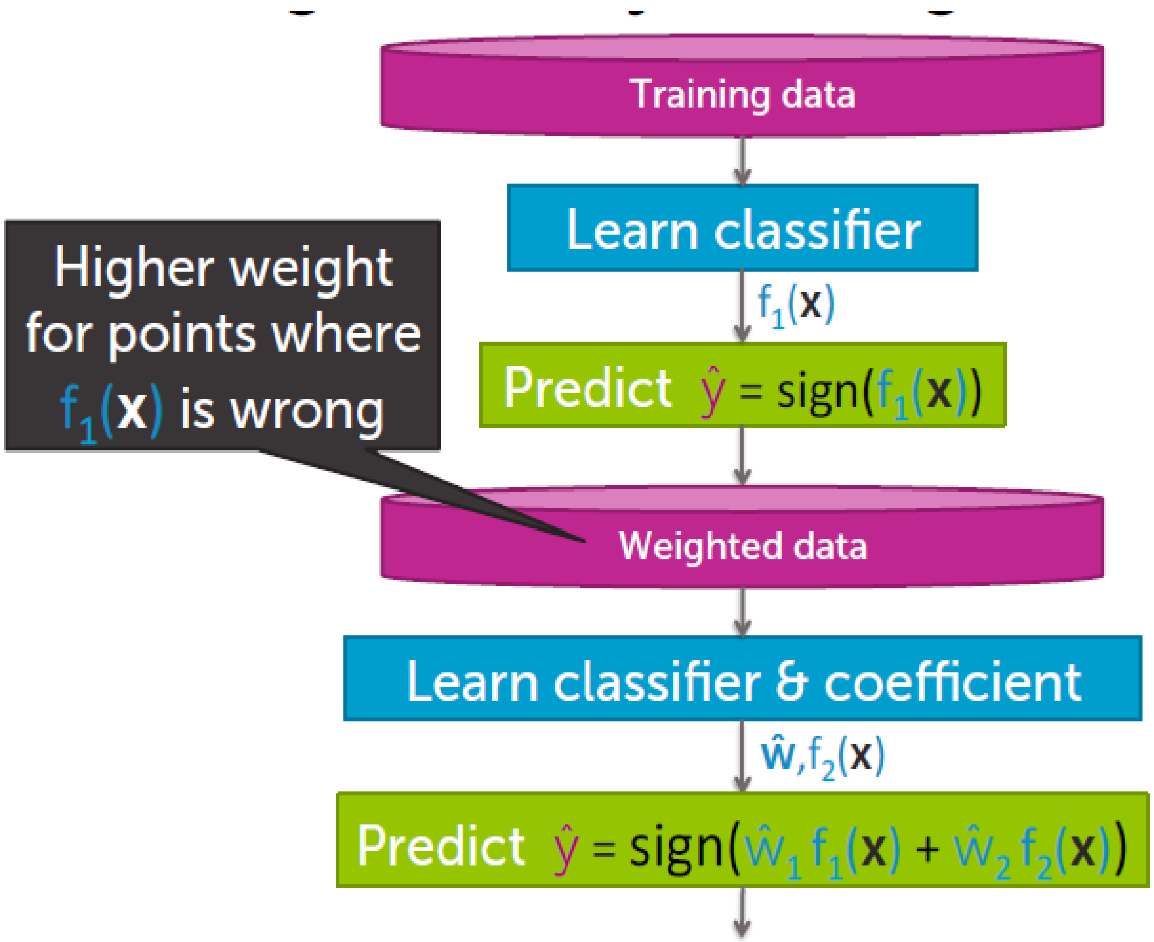
- E.g., gradient ascent for logistic regression:

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



AdaBoost: learning ensemble

[Freund & Schapire 1999]

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

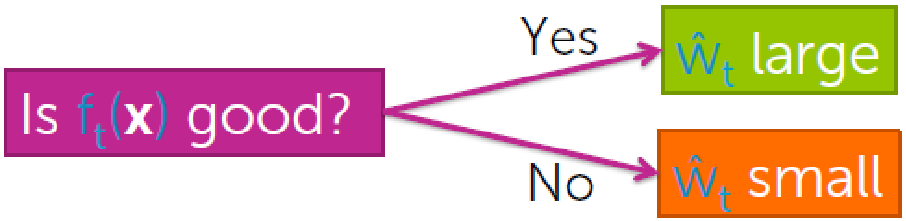
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$ ← Problem 1: How much do I trust f_t?
  - Recompute weights α_i ← Problem 2: weigh mistakes more?

- 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 $f_t(\mathbf{x})$

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

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

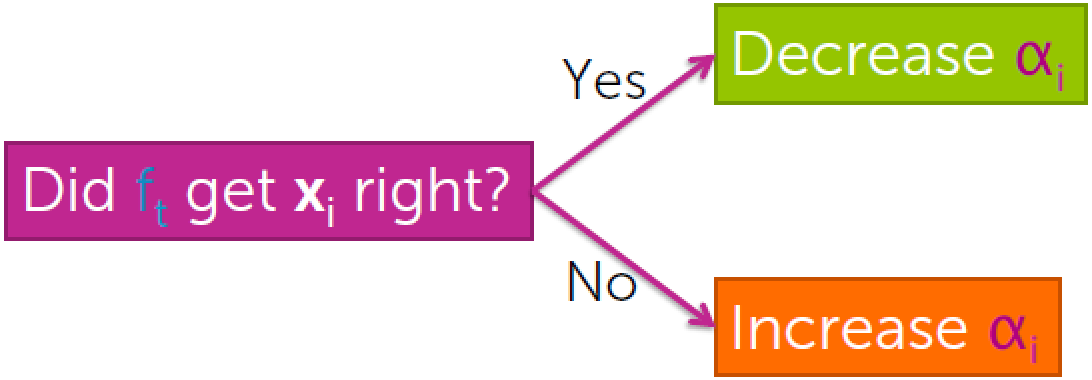
$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

- Final model predicts by:

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

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 α_i gets very **large**

If $\mathbf{x}_i$ often correct, weight α_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 $t = 1,\ldots,T$
  - 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:

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

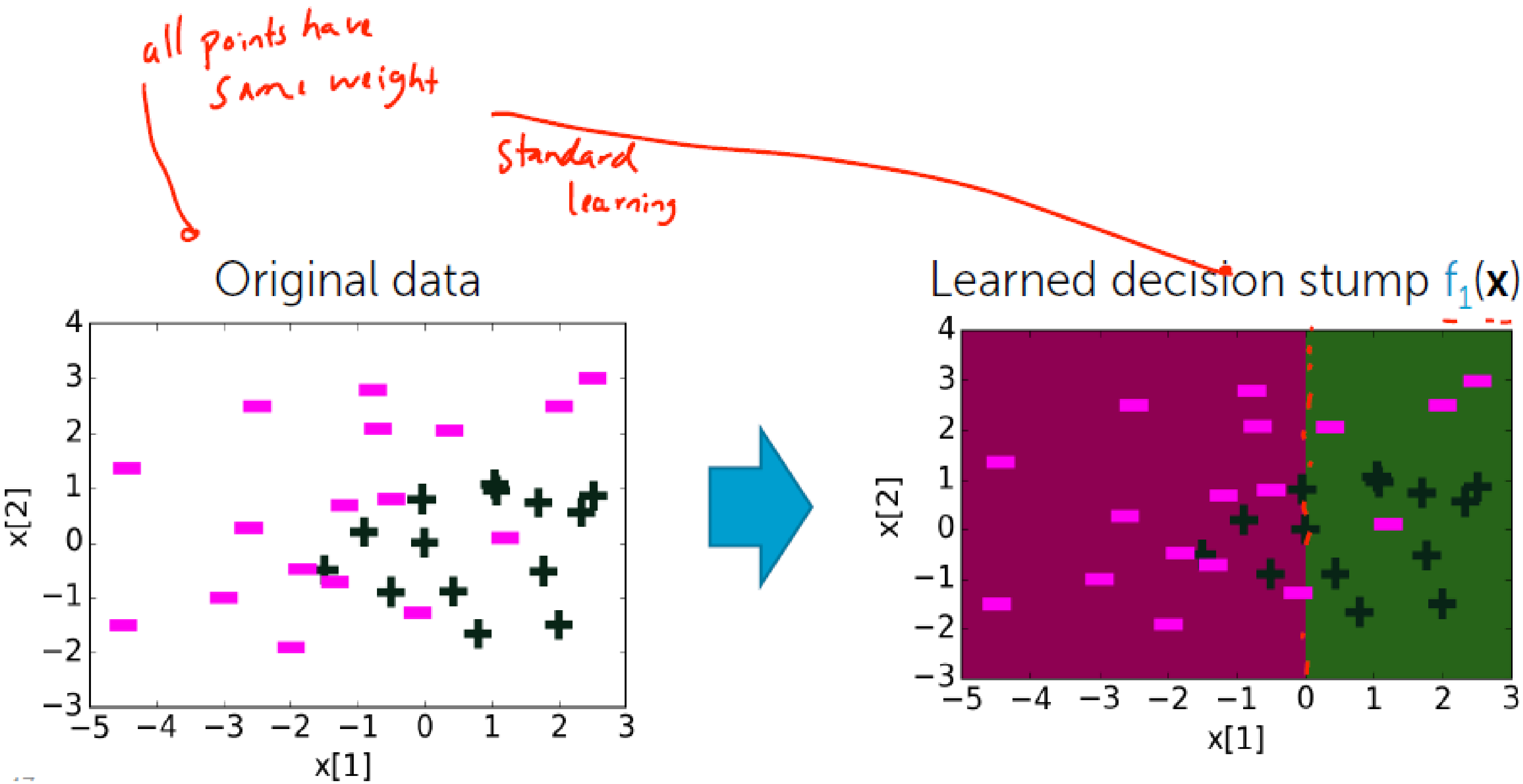
$$\hat{w}_t = \frac{1}{2} \ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right)$$

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

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

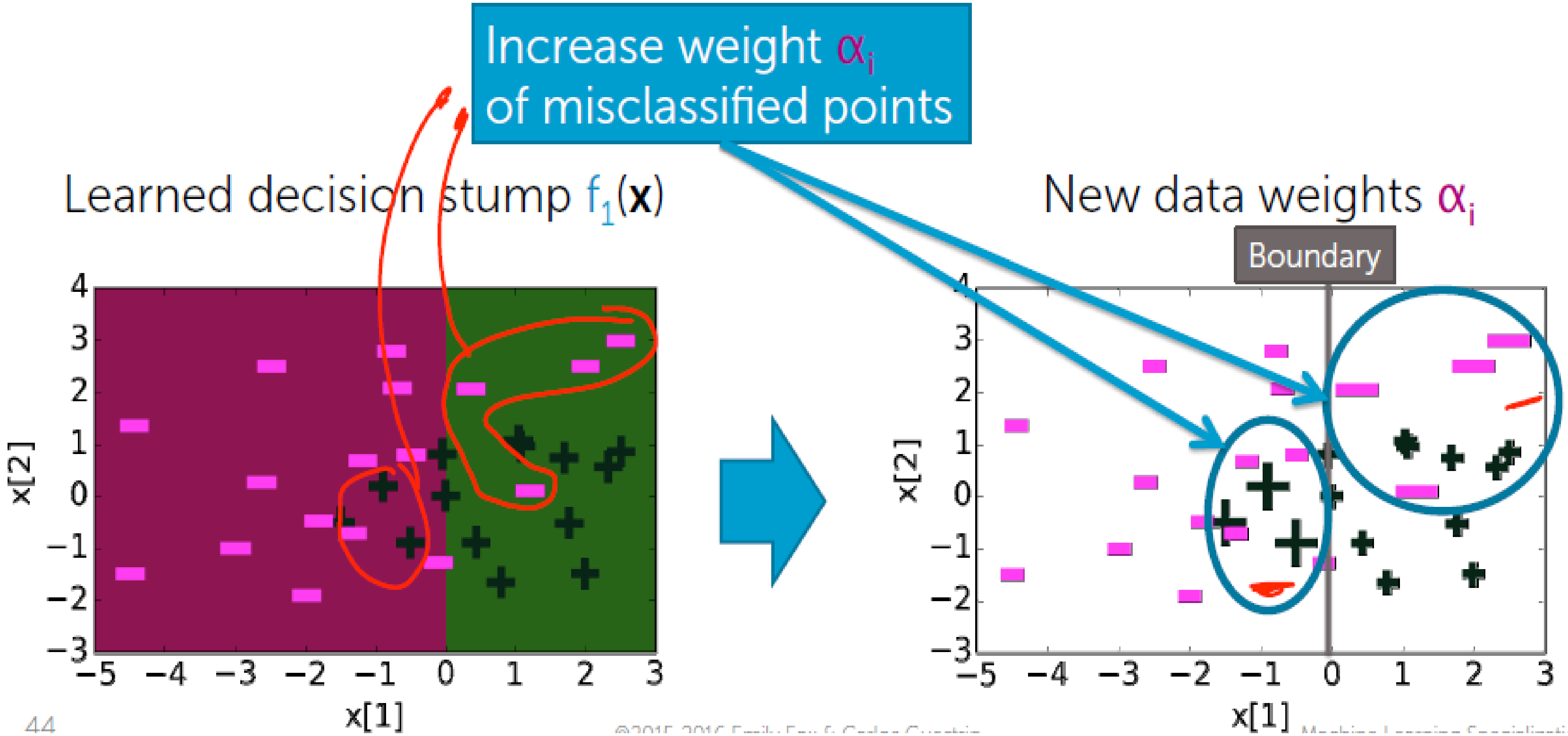
AdaBoost: example

t=1: Just learn a classifier on original data



AdaBoost: example

Updating weights α_i

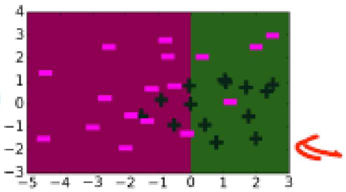


30/10, 6/11 2024

AdaBoost: example

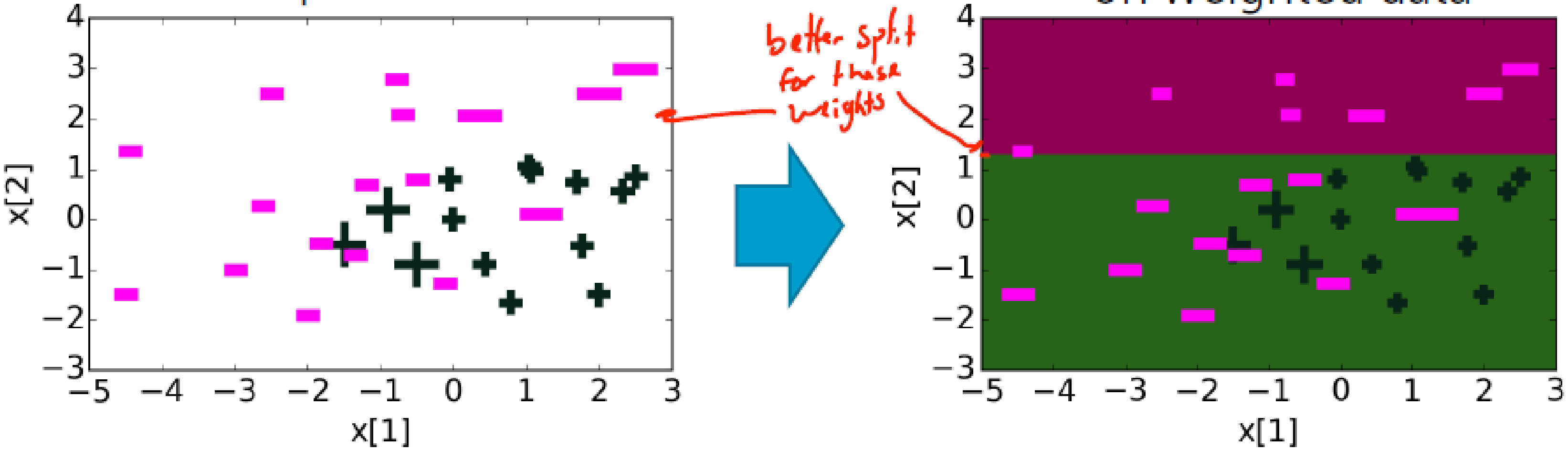
t=2: Learn classifier on weighted data

$f_1(\mathbf{x})$



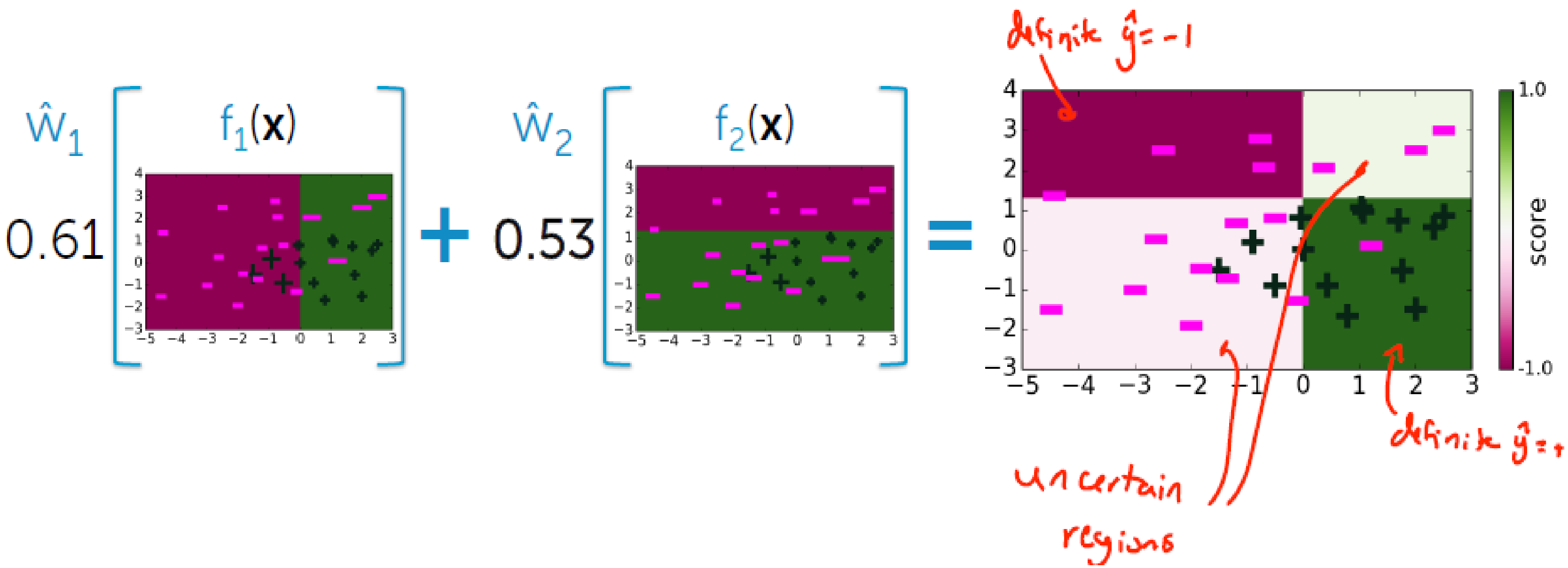
Weighted data: using α_i chosen in previous iteration

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



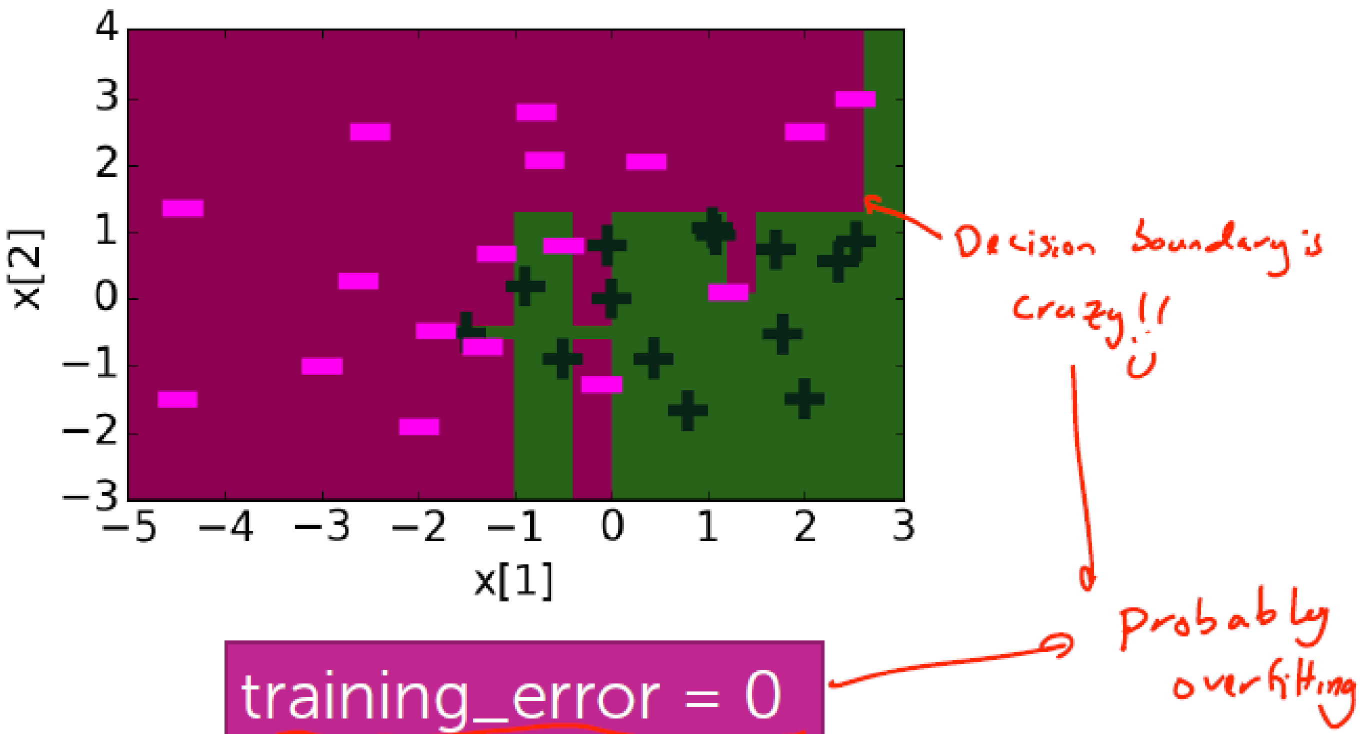
AdaBoost: example

Ensemble becomes weighted sum of learned classifiers



AdaBoost: example

Decision boundary of ensemble classifier after 30 iterations



training_error = 0

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:

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

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

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

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

Boosted decision stumps

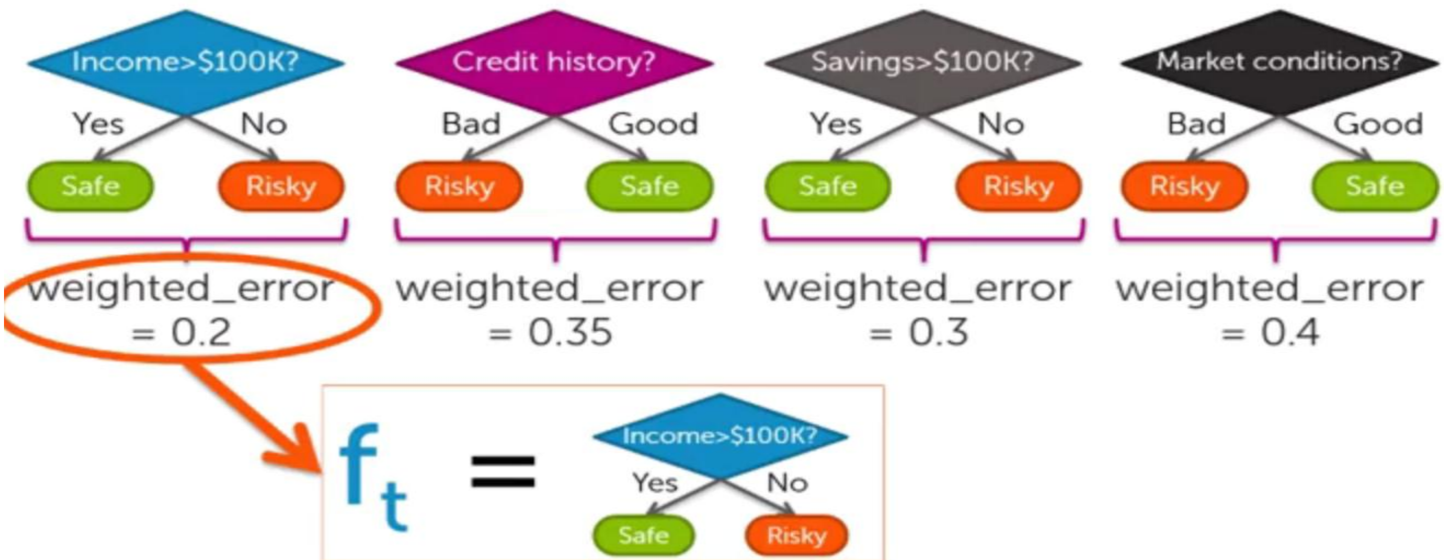
- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick decision stump with lowest weighted training error according to α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

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

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:

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

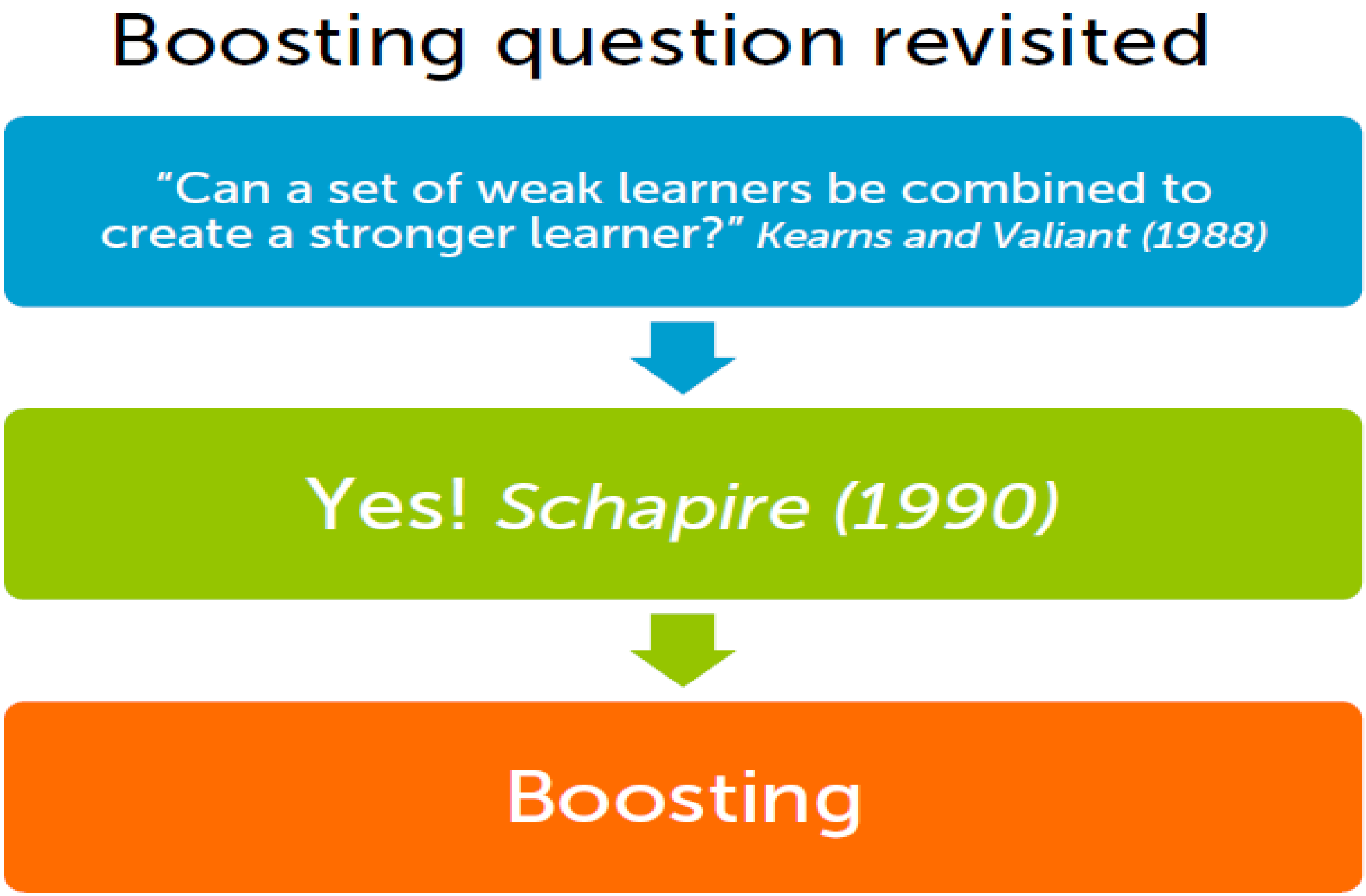
Boosted decision stumps

Updating weights α_i

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

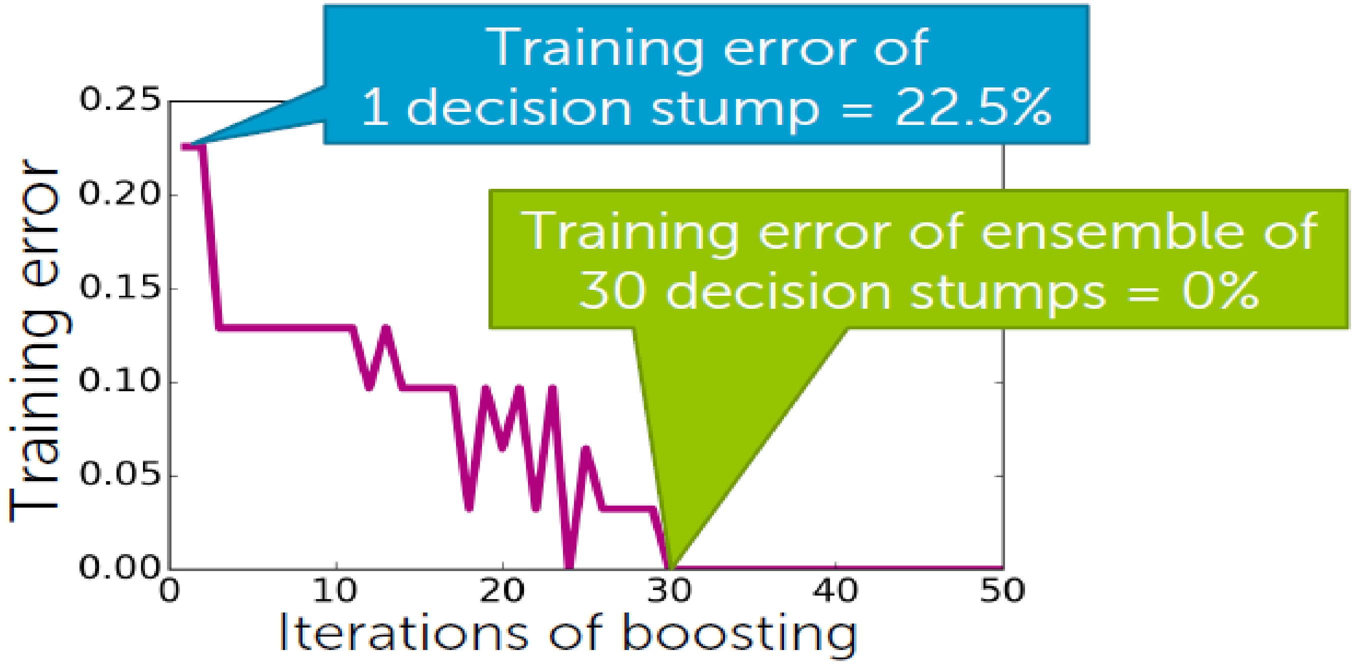
Credit	Income	y	ŷ	Previous weight α	New weight α
A	$130K	Safe	Safe	0.5	0.5/2 = 0.25
B	$80K	Risky	Risky	1.5	0.75
C	$110K	Risky	Safe	1.5	2 * 1.5 = 3
A	$110K	Safe	Safe	2	1
A	$90K	Safe	Risky	1	2
B	$120K	Safe	Safe	2.5	1.25
C	$30K	Risky	Risky	3	1.5
C	$60K	Risky	Risky	2	1
B	$95K	Safe	Risky	0.5	1
A	$60K	Safe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

Boosting convergence & overfitting



Boosting convergence & overfitting

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



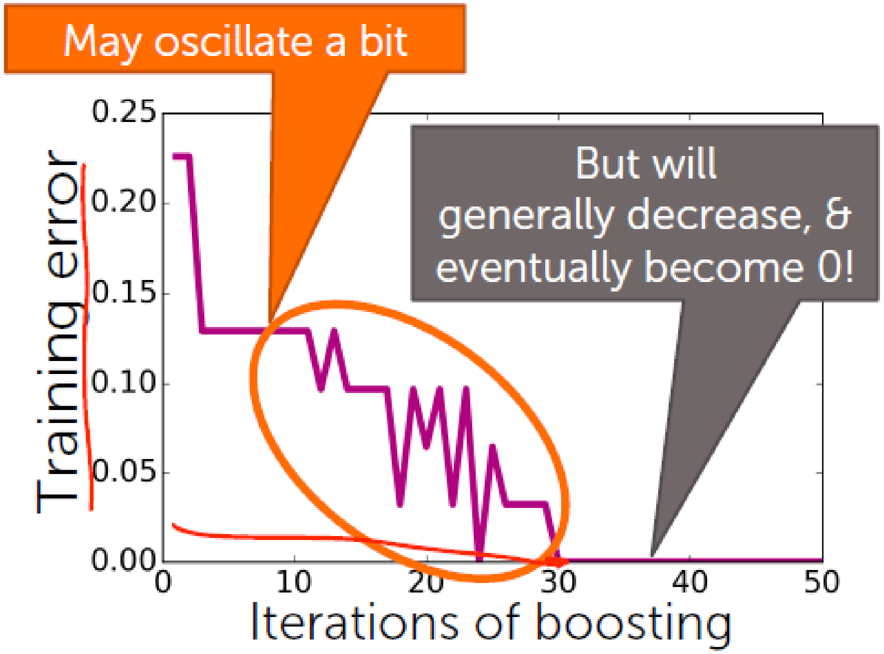
Boosted decision stumps on toy dataset

Boosting convergence & overfitting

AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞



Boosting convergence & overfitting

Condition of AdaBoost Theorem

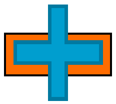
Under some technical conditions…

Training error of boosted classifier → 0 as $T \to \infty$

Condition = At every t, can find a weak learner with weighted_error(f_t) < 0.5

Not always possible

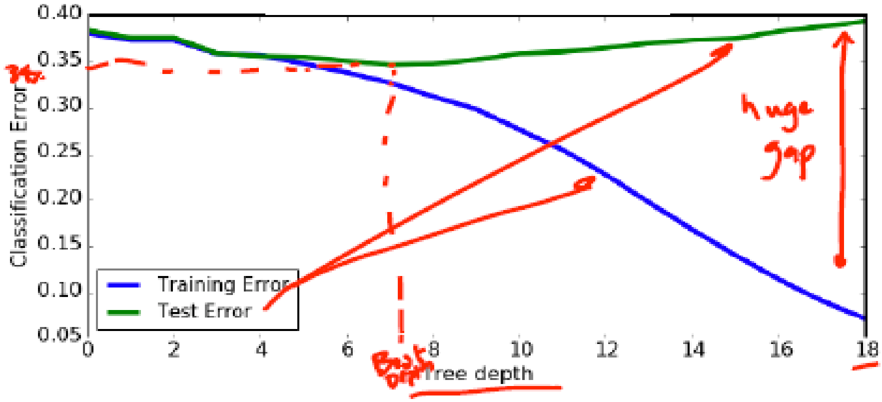
Extreme example: No classifier can separate a +1 on top of -1



Nonetheless, boosting often yields great training error

Example

Decision trees on loan data

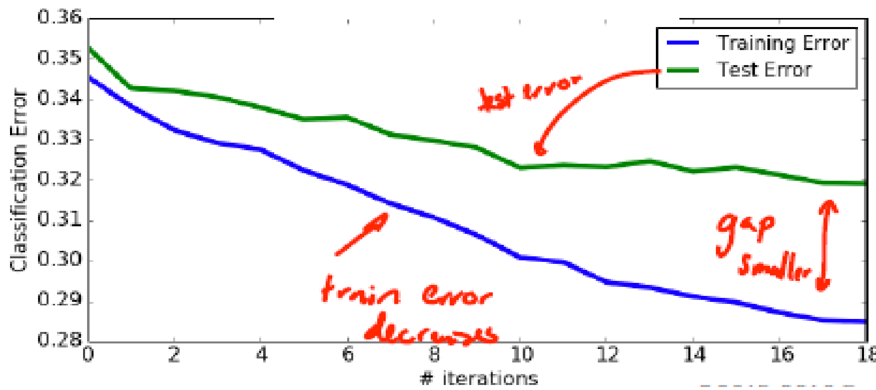


39% test error

Overfitting

8% training error

Boosted decision stumps on loan data



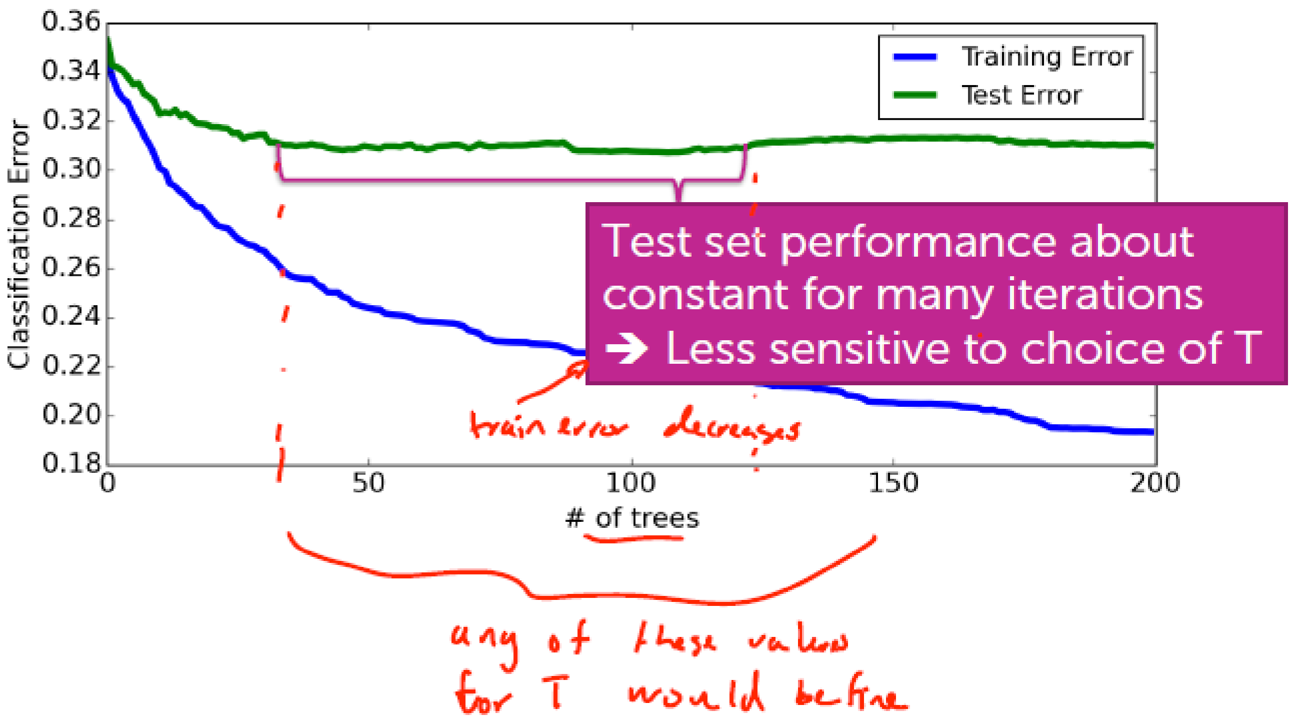
32% test error

Better fit & lower test error

28.5% training error

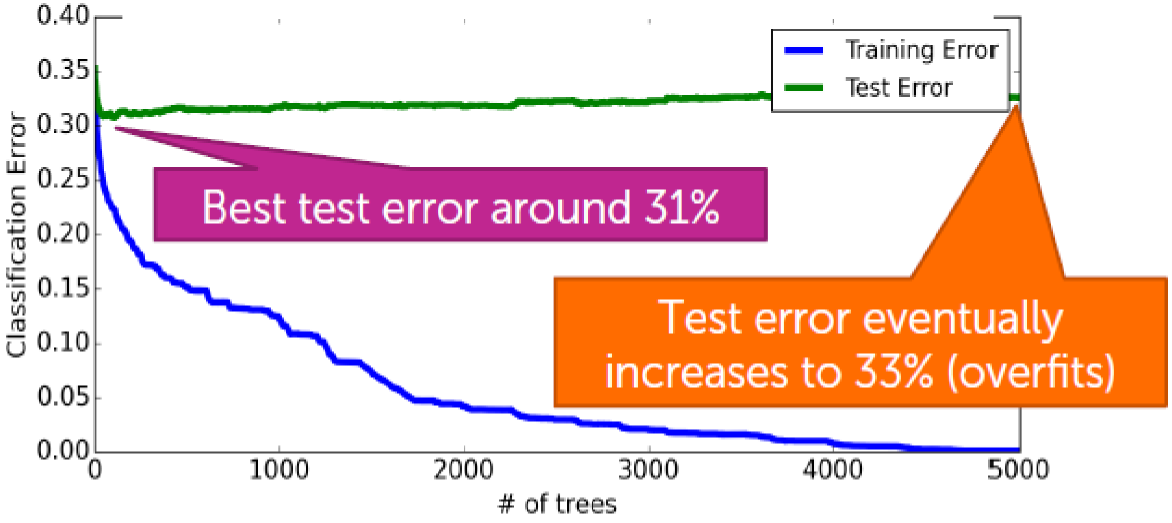
Example

Boosting tends to be robust to overfitting



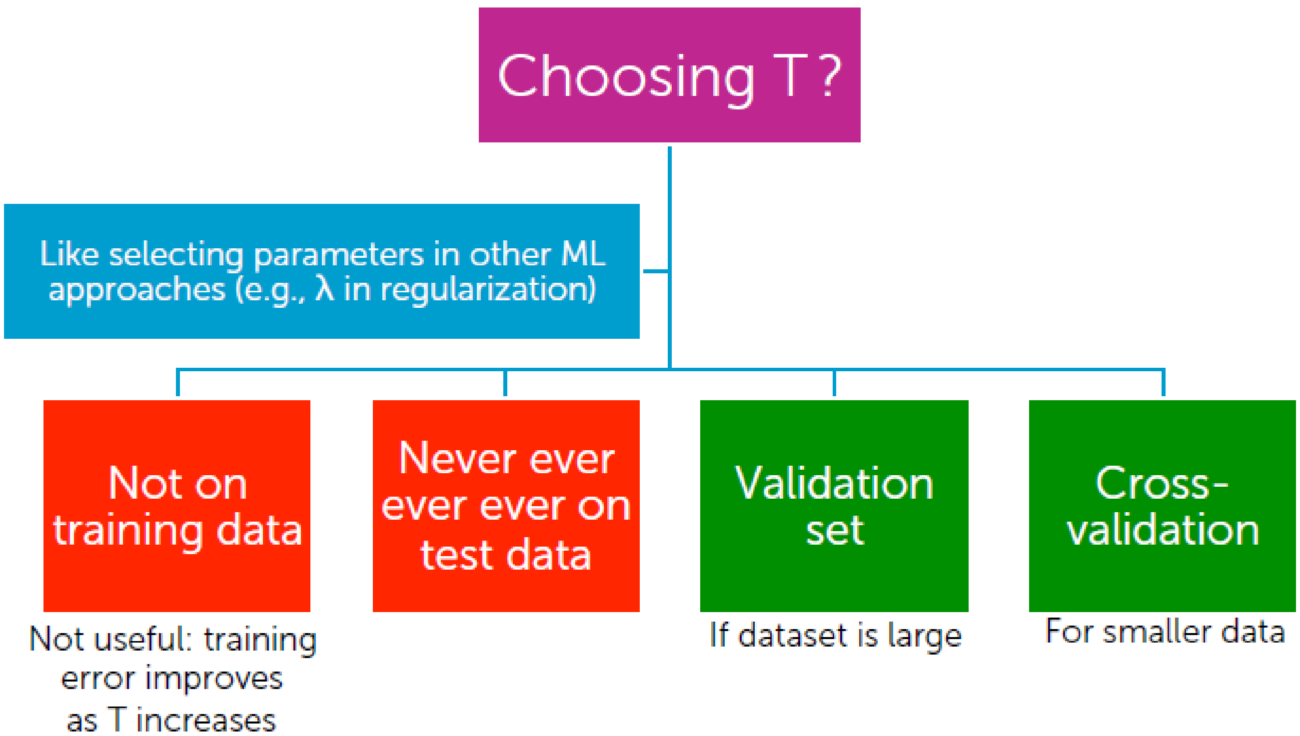
Example

But boosting will eventually overfit,
so must choose max number of components T



Example

How do we decide when to stop boosting?



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

What you can do now

- Identify notion ensemble classifiers
- Formalize ensembles as the weighted combination of simpler classifiers
- Outline the boosting framework – sequentially learn classifiers on weighted data
- Describe the AdaBoost algorithm
  - Learn each classifier on weighted data
  - Compute coefficient of classifier
  - Recompute data weights
  - Normalize weights
- Implement AdaBoost to create an ensemble of decision stumps
- Discuss convergence properties of AdaBoost & how to pick the maximum number of iterations T

Details

- Derivative of likelihood for logistic regression

The log trick, often used in ML…

- Products become sums:

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

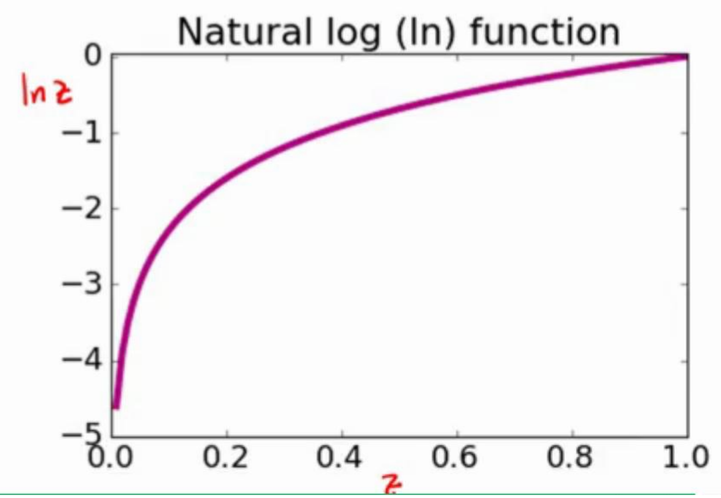
- Doesn't change maximum!
  - If $\hat{\mathbf{w}}$ maximizes f(**w**):

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

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

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

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



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:

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

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

Indicator function

if $y_i = +1$ ✓ 0

if $y_i = -1$ 0 ✓

Logistic regression

Logistic regression model: $P(y=-1|\mathbf{x},\mathbf{w})$

- 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

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

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

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

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

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

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

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

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:

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