Maximum Entropy Classifier
&
Supervised Machine Learning

CSE538 - Spring 2025

Topics

- *Supervised classification (open-vocabulary)*
- Goal of logistic regression (aka Maximum Entropy Classifier)
- The “loss function” -- what logistic regression tries to optimize
- Logistic regression with multiple features
- How to evaluation: Training and test datasets
- Overfitting: role of regularization

Text Classification

The Eagles win it!

Twitter to be acquired by Apple

Tariff delay as Trump and Trudeau reach deal

She will drive to the office, to make sure the lawyer gives the will to the family.

will.n or *will.v* ?

noun verb

I like the the movie.

The movie is like terrible.

Supervised Classification

Supervised Classification

X - features of N observations (i.e. words)

Y - class of each of N observations

GOAL: Produce a *model* that outputs the most likely class y_i, given features x_i.

$$f(X) = Y$$

Supervised Classification

X - features of N observations (i.e. words)

Y - class of each of N observations

GOAL: Produce a *model* that outputs the most likely class y_i, given features x_i.

$$f(X) = Y$$

i	X	Y
0	0.0	0
1	0.5	0
2	1.0	1
3	0.25	0
4	0.75	1

Supervised Classificati

X - features of N observations (

Y - class of each of N obse vatio

Some function or rules to go from *X* to *Y*, as close as possible.

GOAL: Produce a *model* that outputs the most likely class y_i, given features x_i.

$$f(X) = Y$$

i	X	Y
0	0.0	0
1	0.5	0
2	1.0	1
3	0.25	0
4	0.75	1

Supervised Classification

Supervised Machine Learning: Build a model with examples of outcomes (i.e. Y) that one is trying to predict.
unsupervised machine learning: tries to learn with only an X).

Classification: The outcome (Y) is a discrete class.
for example: $y \in$ {not-noun, noun}

$y \in$ {noun, verb, adjective, adverb}

$y \in$ {positive_sentiment, negative_sentiment}).

Classification as Producing a Probability

Binary classification goal: Build a "model" that can estimate $P(A=1|B=?)$

i.e. given B, yield (or “predict”) the probability that A=1

Classification as Producing a Probability

Binary classification goal: Build a “model” that can estimate P(A=1|B=?)

i.e. given B, yield (or “predict”) the probability that A=1

In machine learning, the tradition is to use $\mathbf{Y}$ for the variable being predicted and $\mathbf{X}$ for the features use to make the prediction.

Classification as Producing a Probability

Binary classification goal: Build a “model” that can estimate $P(Y=1|X=?)$

i.e. given X, yield (or “predict”) the probability that $Y=1$

In machine learning, the tradition is to use $\mathbf{Y}$ for the variable being predicted and $\mathbf{X}$ for the features use to make the prediction.

Classification as Producing a Probability

Binary classification goal: Build a “model” that can estimate P(Y=1|X=?)

i.e. given X, yield (or “predict”) the probability that Y=1

In machine learning, the tradition is to use **Y** for the variable being predicted and **X** for the features use to make the prediction.

Example: **Y**: 1 if target is verb, 0 otherwise;
X: 1 if “was” occurs before target; 0 otherwise

I was reading for NLP. *We were fine.* *I am good.*

The cat was very happy. *We enjoyed the reading material.* *I was good.*

Classification as Producing a Probability

Binary classification goal: Build a “model” that can estimate P(Y=1|X=?)

i.e. given X, yield (or “predict”) the probability that Y=1

In machine learning, the tradition is to use **Y** for the variable being predicted and **X** for the features use to make the prediction.

Example: **Y**: 1 if target is verb, 0 otherwise;
X: 1 if “was” occurs before target; 0 otherwise

I was reading for NLP. *We were fine.* *I am good.*

The cat was very happy. *We enjoyed the reading material.* *I was good.*

Classification as Producing a Probability

Example: **Y**: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

Classification as Producing a Probability

Example: **Y**: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

Classification as Producing a Probability

Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

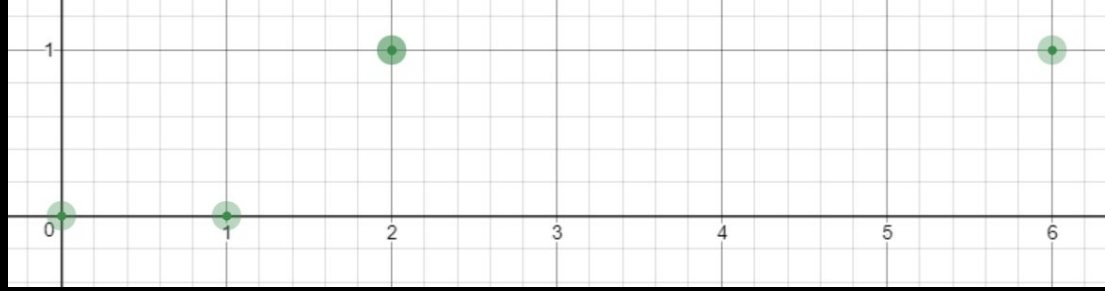
They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

x	y
2	1
1	0
0	0
6	1
2	1

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

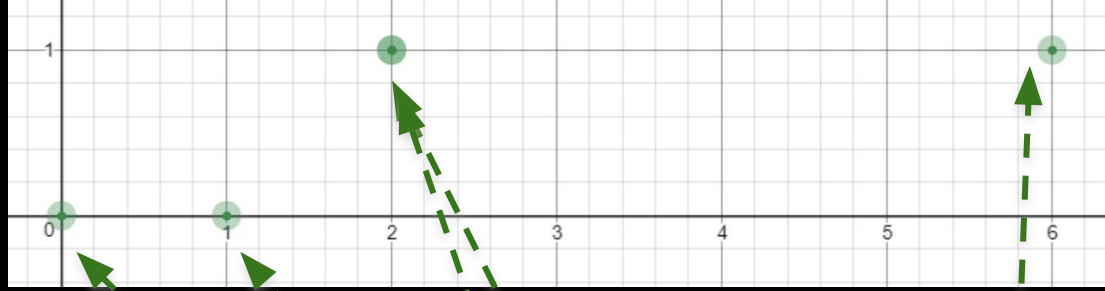
They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

x	y
2	1
1	0
0	0
6	1
2	1

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

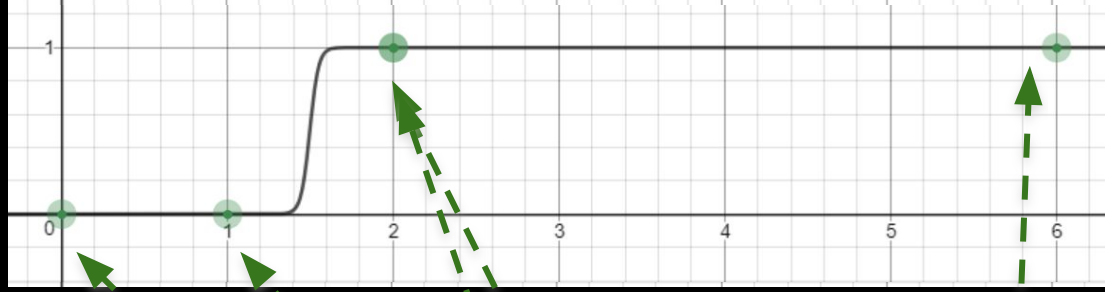
They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

x	y
2	1
1	0
0	0
6	1
2	1

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

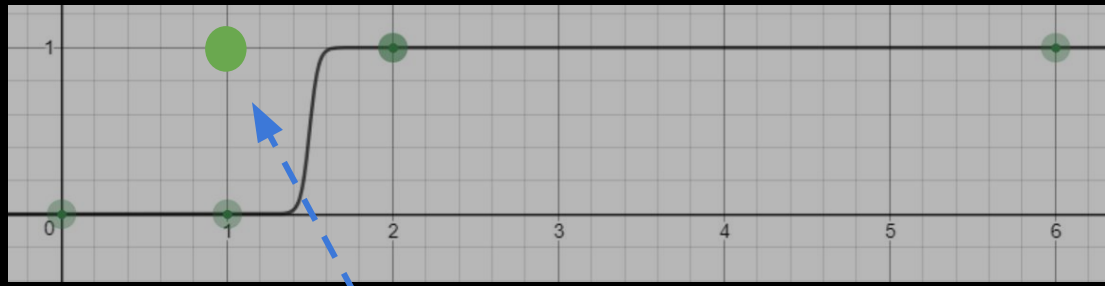
They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

x	y
2	1
1	0
0	0
6	1
2	1

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

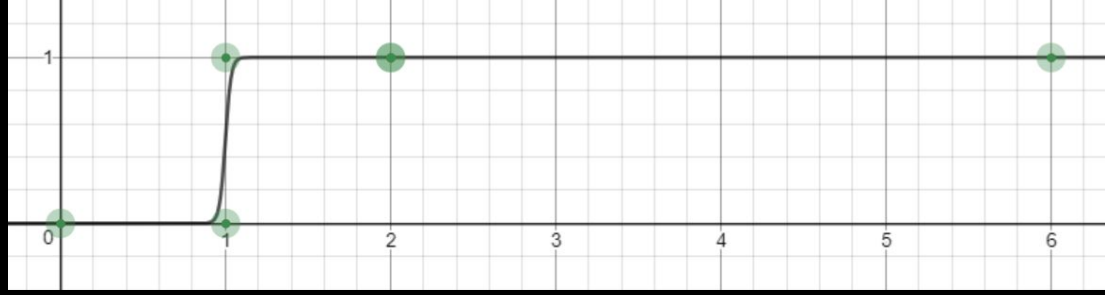
The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

They attend Binghamton.

x	y
2	1
1	0
0	0
6	1
2	1
1	1

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

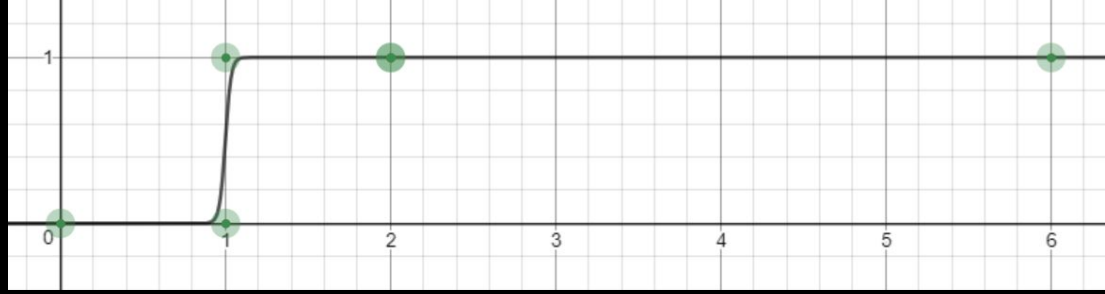
The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

They attend Binghamton.

x	y
2	1
1	0
0	0
6	1
2	1
1	1

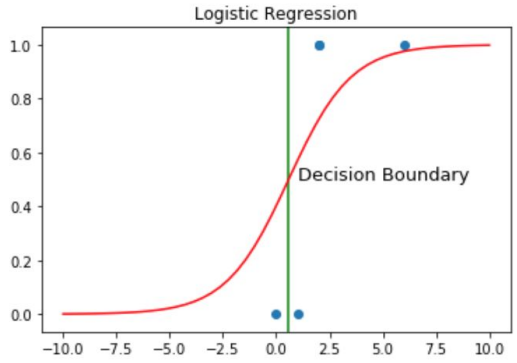
Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;

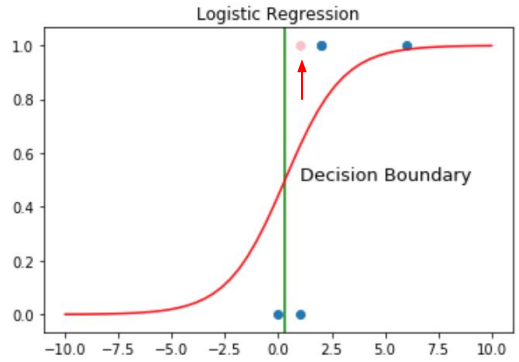
X: number of capital letters in target and surrounding words.

```
Out[43]: [<matplotlib.lines.Line2D at 0x116e68d68>]
```




```
In [78]: 1 -b_0/b_1
Out[78]: 0.5824799517820446

In [28]: 1 logisticRegr.predict(x)
Out[28]: array([1, 1, 0, 1, 1])
```

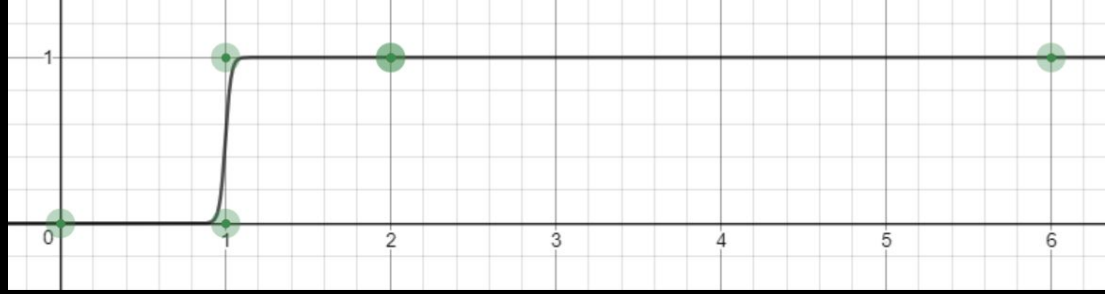
```
Out[80]: [<matplotlib.lines.Line2D at 0x11a60f160>]
```




```
In [81]: 1 -b2_0/b2_1
Out[81]: 0.31089309388058134

In [82]: 1 logisticRegr2.predict(x2)
Out[82]: array([1, 1, 0, 1, 1, 1])
```

1	1

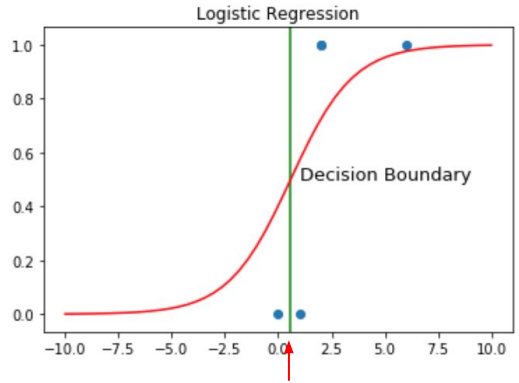
Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;

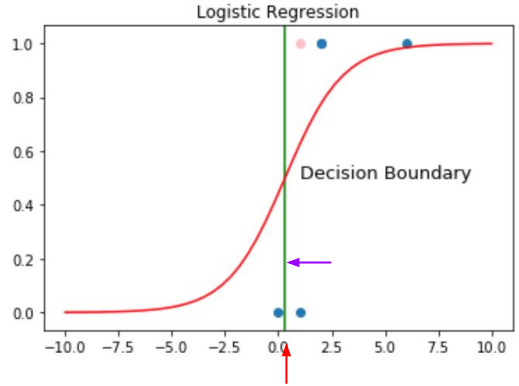
X: number of capital letters in target and surrounding words.

```
Out[43]: [<matplotlib.lines.Line2D at 0x116e68d68>]
```




```
In [78]: 1 -b_0/b_1
Out[78]: 0.5824799517820446

In [28]: 1 logisticRegr.predict(x)
Out[28]: array([1, 1, 0, 1, 1])
```

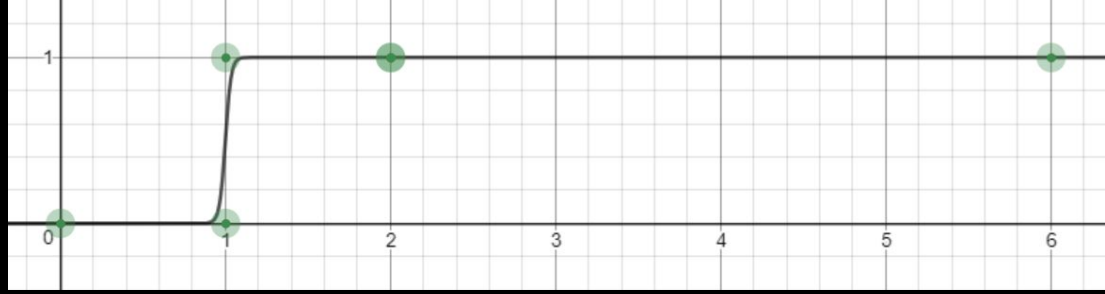
```
Out[80]: [<matplotlib.lines.Line2D at 0x11a60f160>]
```




```
In [81]: 1 -b2_0/b2_1
Out[81]: 0.31089309388058134

In [82]: 1 logisticRegr2.predict(x2)
Out[82]: array([1, 1, 0, 1, 1, 1])
```

1	1

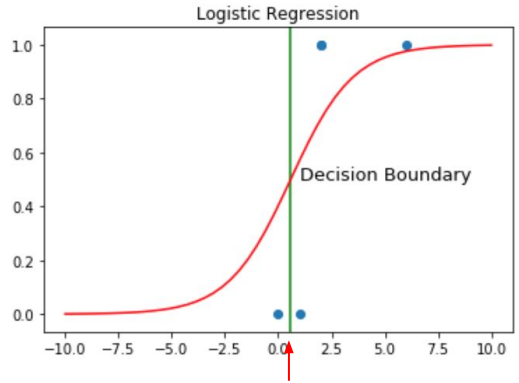
Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;

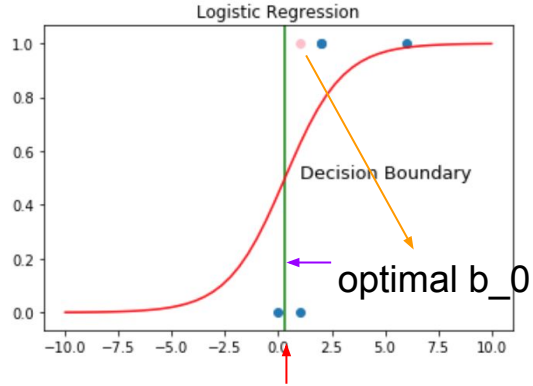
X: number of capital letters in target and surrounding words.

```
Out[43]: [<matplotlib.lines.Line2D at 0x116e68d68>]
```




```
In [78]: 1 -b_0/b_1
Out[78]: 0.5824799517820446

In [28]: 1 logisticRegr.predict(x)
Out[28]: array([1, 1, 0, 1, 1])
```

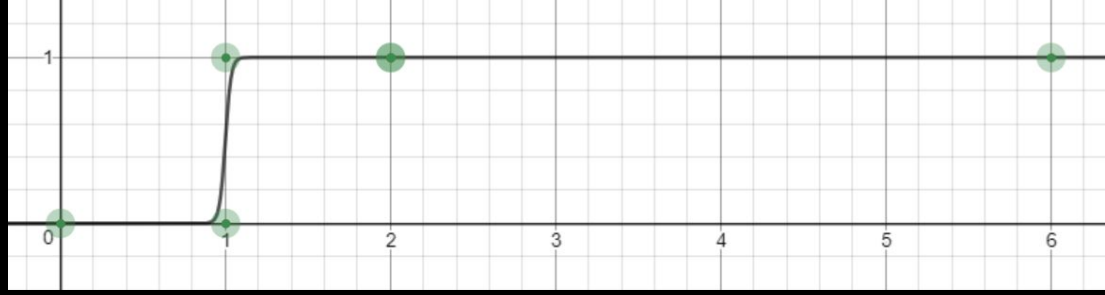
```
Out[80]: [<matplotlib.lines.Line2D at 0x11a60f160>]
```




```
In [81]: 1 -b2_0/b2_1
Out[81]: 0.31089309388058134

In [82]: 1 logisticRegr2.predict(x2)
Out[82]: array([1, 1, 0, 1, 1, 1])
```

1	1

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

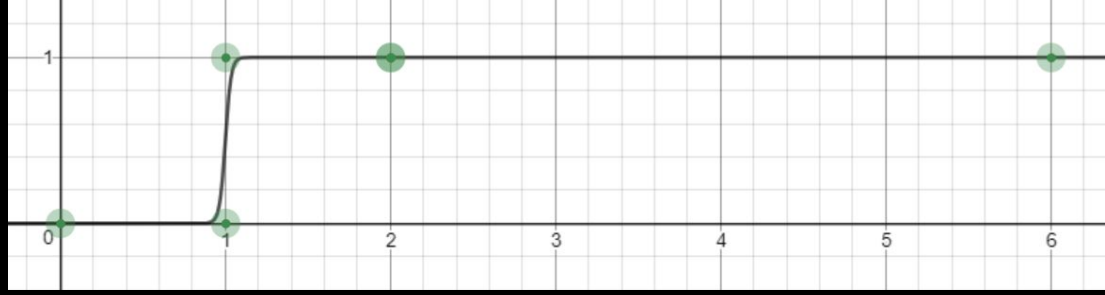
The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

They attend Binghamton.

x	y
2	1
1	0
0	0
6	1
2	1
1	1

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;

X1: number of capital letters in target and surrounding words.

Let's add a feature! X2: does the target word start with a capital letter?

They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

They attend Binghamton.

x2	x1	y
1	2	1
0	1	0
0	0	0
1	6	1
1	2	1
1	1	1

Logistic Regression on features (*x*)

$Y_i \in \{0, 1\}$; X is a **single value** and can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(\beta_0 + \sum_{j=1}^{m} \beta_j x_{ij})}}$$

m: number of features
(columns of *x*)

Logistic Regression on features (*x*)

$Y_i \in \{0, 1\}$; X is a **single value** and can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(\beta_0 + \sum_{j=1}^{m} \beta_j x_{ij})}}$$

$$= \frac{1}{1+e^{-(x_i \beta)}}$$

m: number of features (columns of *x*)

Logistic Regression on features (*x*)

$Y_i \in \{0, 1\}$; X is a **single value** and can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(\beta_0 + \sum_{j=1}^{m} \beta_j x_{ij})}}$$

$$= \frac{1}{1+e^{-(x_i \beta)}}$$

Vector notation

β and x_i are vectors of size m

first feature is intercept:
$x_{*,0} = [1, 1 ..., 1]_N$

m: number of features
(columns of *x*)

Logistic Regression on features (*x*)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(x_i\beta)}}$$

Goal: take in the variable *x* and
return a probability that *Y* is 1.

Logistic Regression on features (x)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(x_i\beta)}}$$

Goal: take in the variable x and
return a probability that Y is 1.

Note that there are only two variables on the right: x_i, β

Logistic Regression on features (*x*)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(x_i\beta)}}$$

Goal: take in the variable *x* and
return a probability that *Y* is 1.

Note that there are only two variables on the right: x_i, β

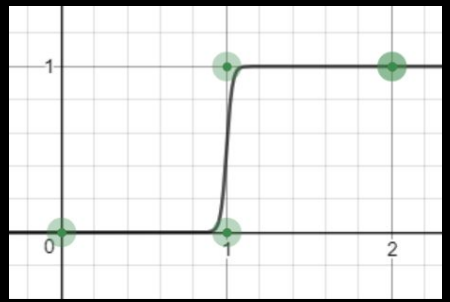
x_i is given. β must be learned.

Logistic Regression on features (*x*)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(x_i\beta)}}$$

HOW? Try different β values until "best fit" to the training data (example X and Y).



x_i is given. β must be learned.

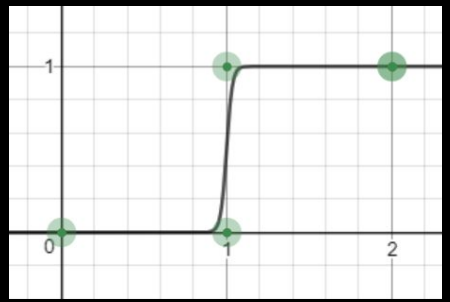
"best fit" : whatever maximizes the likelihood function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

$$P(Y_i = 1|X_i = x) = \frac{1}{1+e^{-(x_i\beta)}}$$

HOW? Try different β values until "best fit" to the training data (example X and Y).

x_i is given. β must be learned.



“best fit” : whatever maximizes the *likelihood* function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

“best fit” : whatever maximizes the *likelihood* function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

“best fit” : more efficient to maximize *log likelihood* :

“best fit” : whatever maximizes the *likelihood* function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

“best fit” : more efficient to maximize *log likelihood* :

$$\ell(\beta) = \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

$$p_i \equiv P(Y_i = 1|X_i = x)$$

“best fit” : whatever maximizes the *likelihood* function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

“best fit” : more efficient to maximize *log likelihood* :

$$\ell(\beta) = \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

“best fit” for neural networks: software designed to **minimize** rather than maximize (typically, normalized by N, the number of examples.)

“best fit” : whatever maximizes the *likelihood* function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

“best fit” : more efficient to maximize *log likelihood* :

$$\ell(\beta) = \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

“best fit” for neural networks: software designed to **minimize** rather than maximize (typically, normalized by N, number of examples.) *"log loss"* or *"normalized log loss":*

$$J(\beta) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

Maximum Entropy Classifier
&
Supervised Machine Learning

CSE538 - Spring 2025

Adithya V Ganesan

bit.ly/cse538sp25-211

Supervised Classification

X - features of N observations (i.e. words)

Y - class of each of N observations

GOAL: Produce a *model* that outputs the most likely class y_i, given features x_i.

$$f(X) = Y$$

i	X	Y
0	0.0	0
1	0.5	0
2	1.0	1
3	0.25	0
4	0.75	1

Supervised Classific

X - features of N observations (

Y - class of each of N obse vatio

Some function or rules to go from X to Y, as close as possible.

GOAL: Produce a *model* that outputs the most likely class y_i, given features x_i.

$$f(X) = Y$$

i	X	Y
0	0.0	0
1	0.5	0
2	1.0	1
3	0.25	0
4	0.75	1

Supervised Classific

X - features of N observations (

Y - class of each of N obse vatio

Some function or rules to go from X to Y, as close as possible.

GOAL: Produce a *model* that outputs the most likely class y_i, given features x_i.

$$f(X) = Y$$

i	X	Y
0	0.0	0
1	0.5	0
2	1.0	1
3	0.25	0
4	0.75	1

Learn function or rules to go from X to Y, **as close as possible**

Logistic Regression on features (*x*)

$Y_i \in \{0, 1\}$; X can be anything numeric.

$$P(Y_i = 1 | X_i = x) = \frac{1}{1+e^{-(x_i \beta)}}$$

Goal: take in the variable *x* and
return a probability that *Y* is 1.

“best fit” : whatever maximizes the *likelihood* function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

“best fit” : whatever maximizes the *likelihood* function:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i} (1 - P(Y_i = 1|x_i))^{1-y_i}$$

“best fit” : more efficient to maximize *log likelihood* :

$$\ell(\beta) = \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

“best fit” for neural networks: software designed to **minimize** rather than maximize (typically, normalized by N, number of examples.) *"log loss"* or *"normalized log loss":*

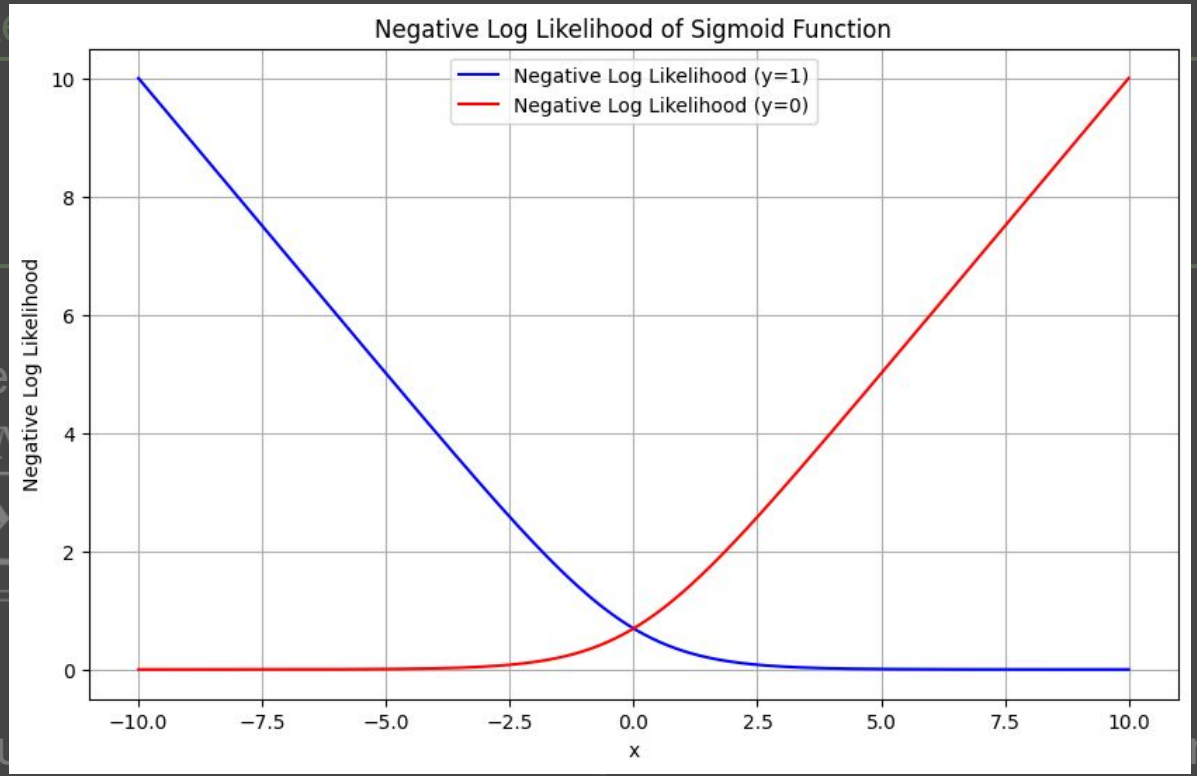
$$J(\beta) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

“best fit” : what

$$L(\beta|X,Y) \qquad 1-y_i$$

“best fit” : more

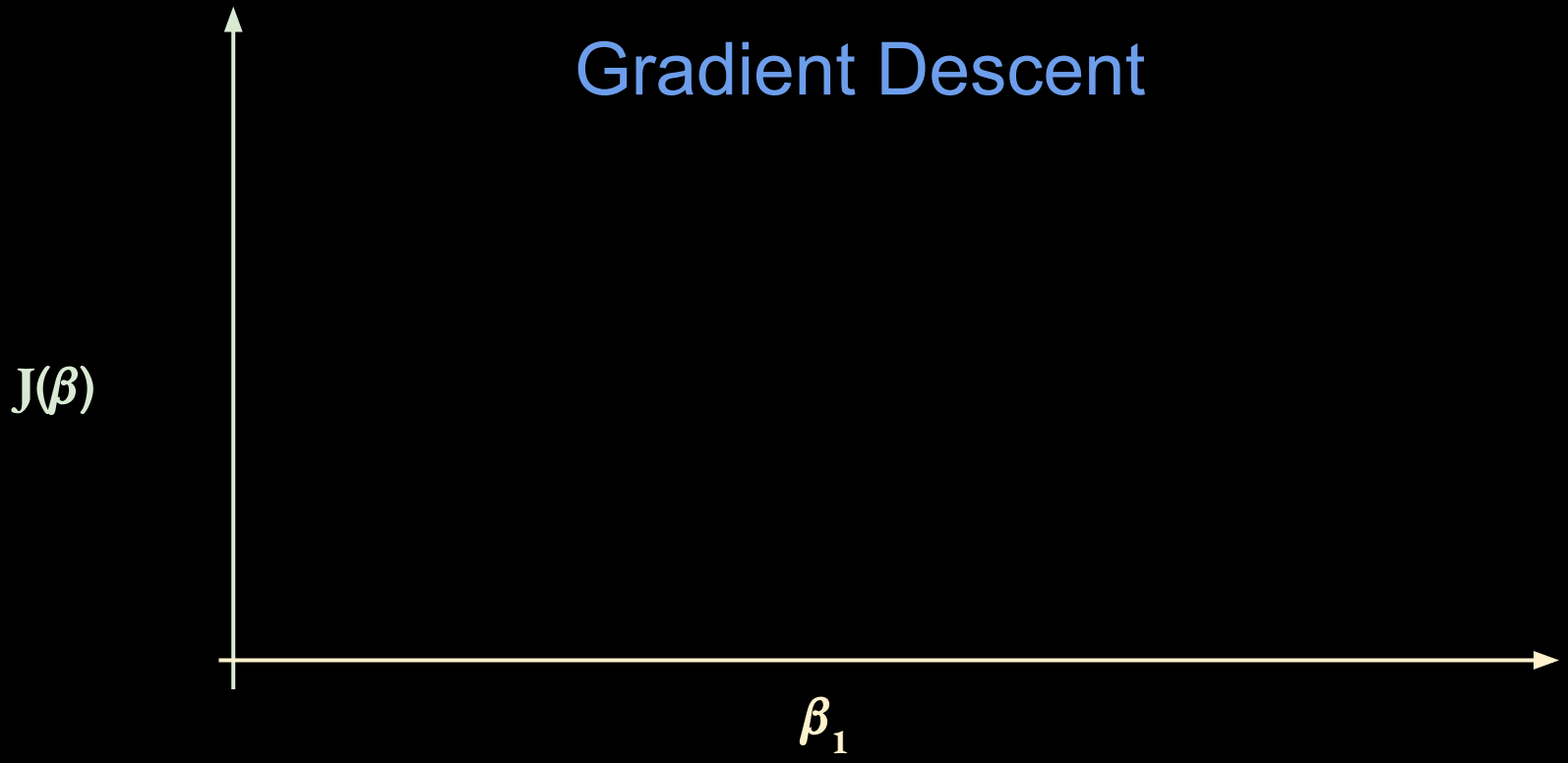
$$\ell(\beta) = \sum_{i=}$$



“best fit” for neu n maximize
(typically, normalized by N, number of examples.) *"log loss"* or *"normalized log loss":*

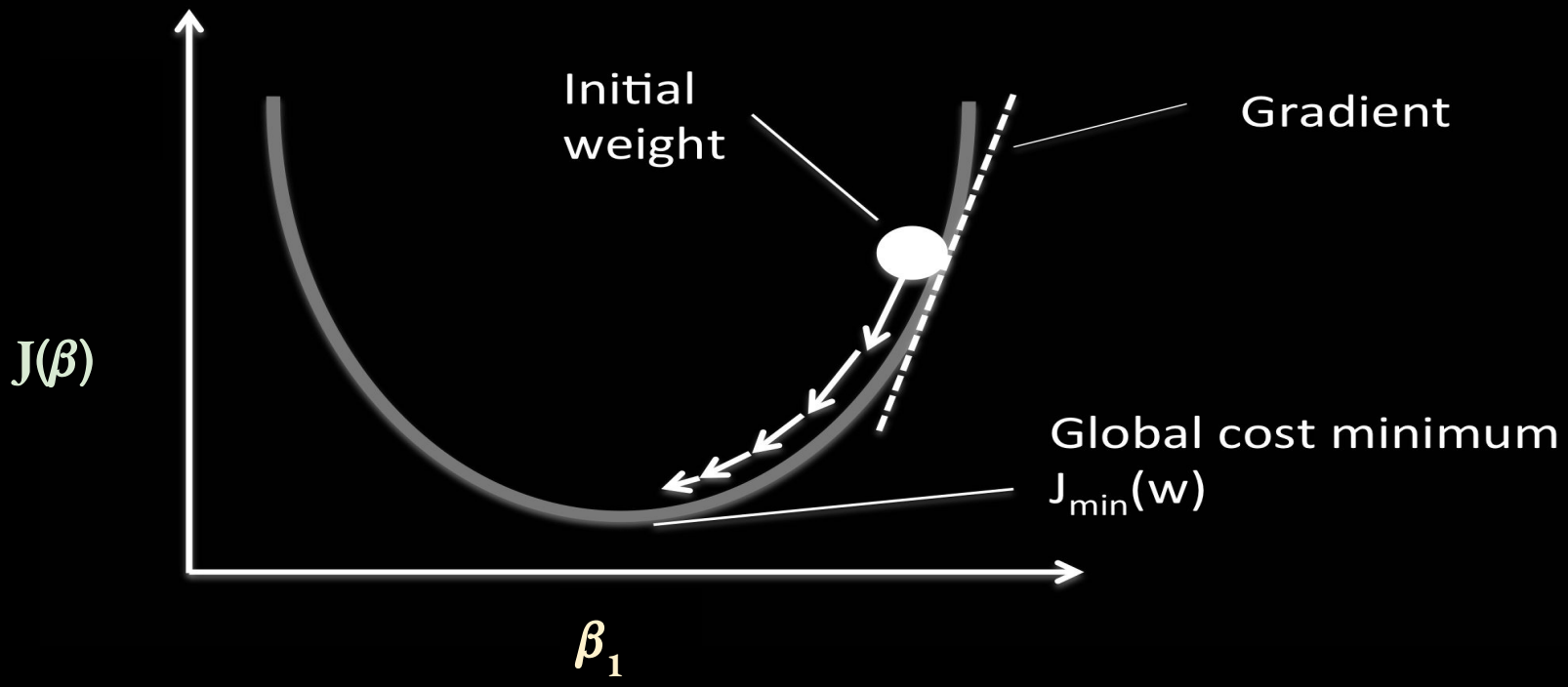
$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1-y_i)\log(1-p_i)$$

Gradient Descent



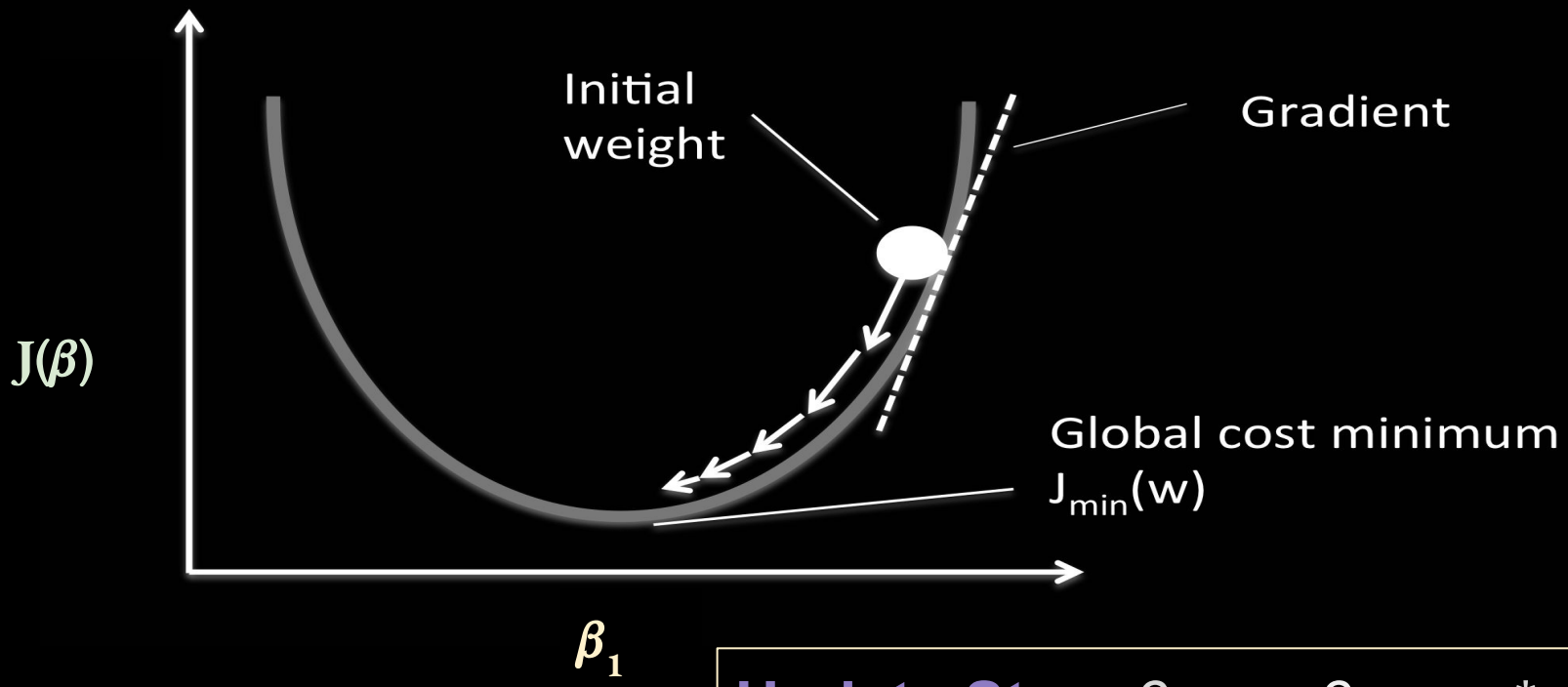
"*log loss*" or *"normalized log loss"*:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$



"*log loss*" or *"normalized log loss"*:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$



Update Step: $\beta_{new} = \beta_{old} - \alpha * grad$

"log loss" or *"normalized log loss":*

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

“best fit” : whatever maximizes the *likelihood* function:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

“best fit” : whatever maximizes the *likelihood* function:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

$$\nabla_\beta J(\beta) = \frac{1}{N}\sum_{i=1}^{N} (p_i - y_i) x_i.$$

“best fit” : whatever maximizes the *likelihood* function:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

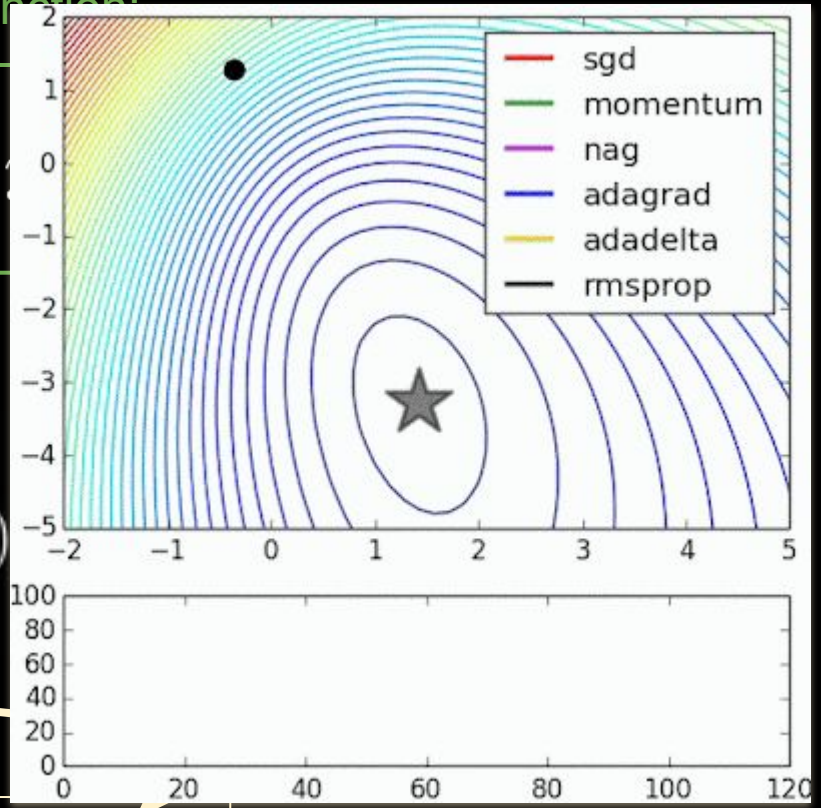
$$\nabla_\beta J(\beta) = \frac{1}{N}\sum_{i=1}^{N} (p_i - y_i) x_i.$$

Update Step: $\beta_{new} = \beta_{old} - \alpha * \text{grad}$

"best fit" : whatever maximizes the *likelihood* function:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - $$

$$\nabla_\beta J(\beta) \equiv \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)$$



Update Step: $\beta_{new} = \beta_{old} - \alpha * grad$

(Animation: Alec Radford, 2018)

"best fit" : whatever maximizes the *likelihood* function:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

Demonstration of Logistic Regression with Gradient Descent:

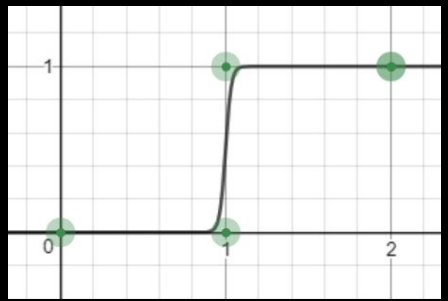
bit.ly/cse538sp25-logreg-numpy

Update Step: $\beta_{new} = \beta_{old} - \alpha * grad$

X can be multiple features

Often we want to make a classification based on multiple features:

- Number of capital letters surrounding: integer
- Begins with capital letter: {0, 1}
- Preceded by “the”? {0, 1}



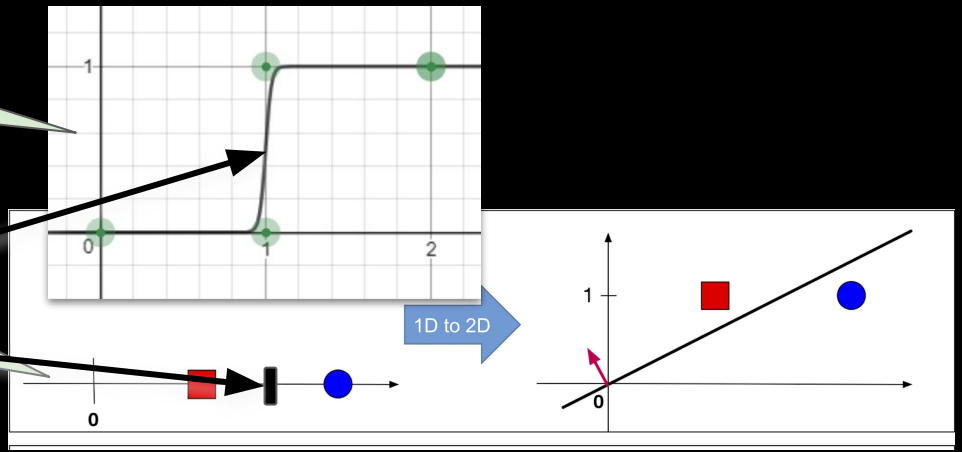
X can be multiple features

Often we want to make a classification based on multiple features:

- Numb
  surroundi
- Begins wi {0, 1}
- Prec

Y-axis is Y (i.e. 1 or 0)

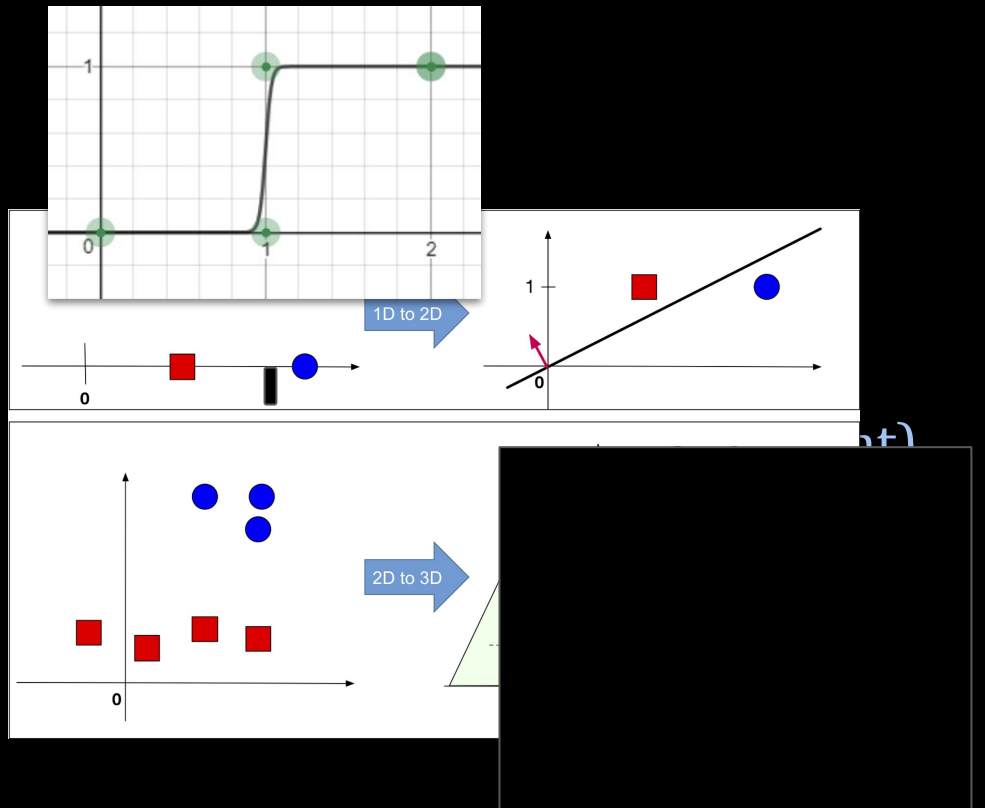
To make room for multiple Xs, let's get rid of y-axis. Instead, show **decision point**.



X can be multiple features

Often we want to make a classification based on multiple features:

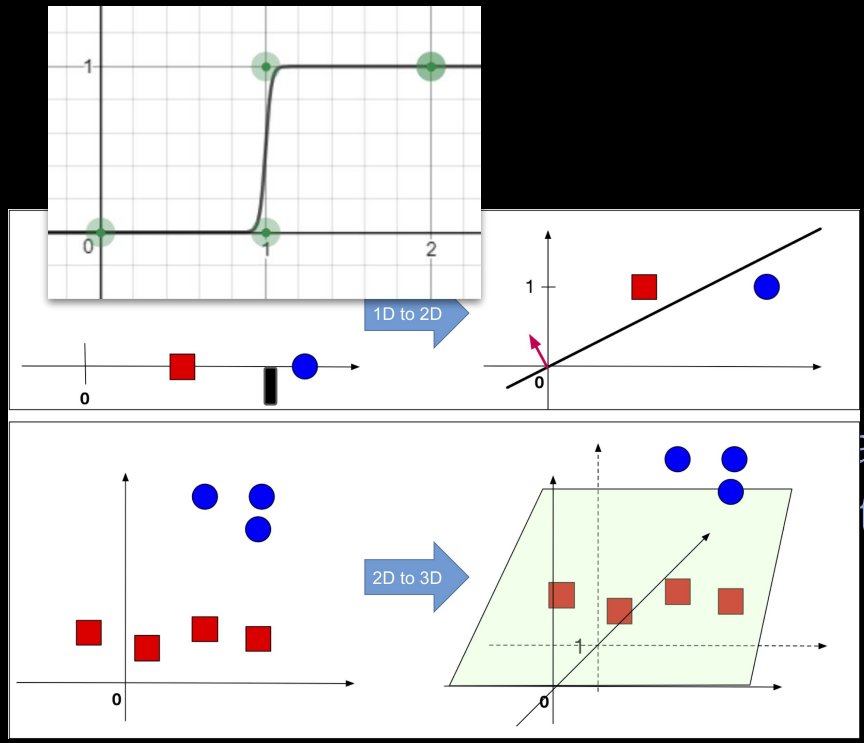
- Number of capital letters surrounding: integer
- Begins with capital letter: {0, 1}
- Preceded by “the”? {0, 1}



X can be multiple features

Often we want to make a classification based on multiple features:

- Number of capital letters surrounding: integer
- Begins with capital letter: {0, 1}
- Preceded by “the”? {0, 1}

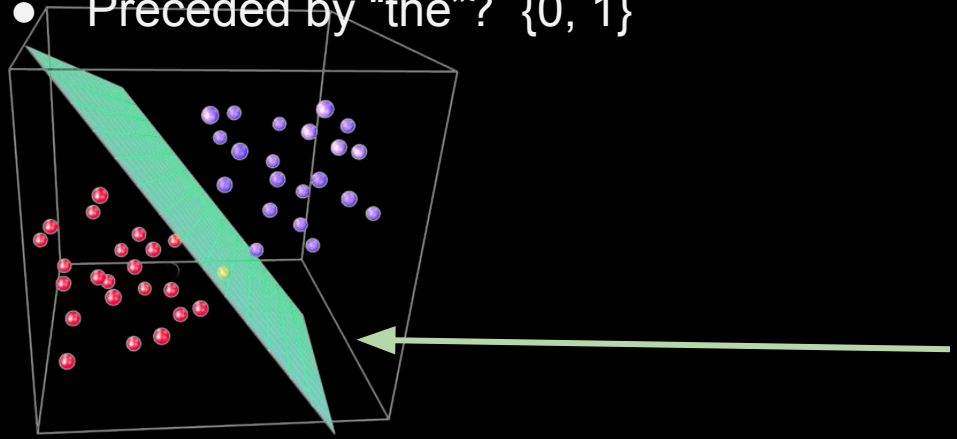


at)
ting

X can be multiple features

Often we want to make a classification based on multiple features:

- Number of capital letters surrounding: integer
- Begins with capital letter: {0, 1}
- Preceded by “the”? {0, 1}



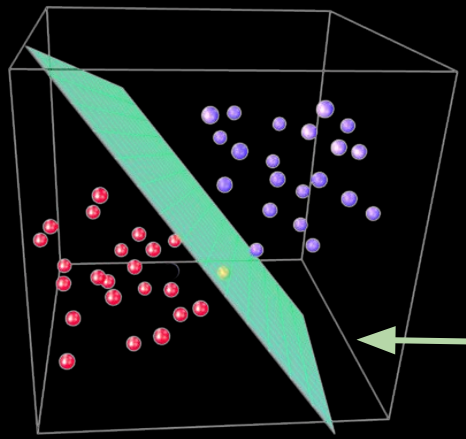
We’re learning a linear (i.e. flat) *separating hyperplane*, but fitting it to a *logit* outcome.

(https://www.linkedin.com/pulse/predicting-outcomes-probabilities-logistic-regression-konstantinidis/)

Logistic Regression

$Y_i \in \{0, 1\}$; X can be anything numeric.

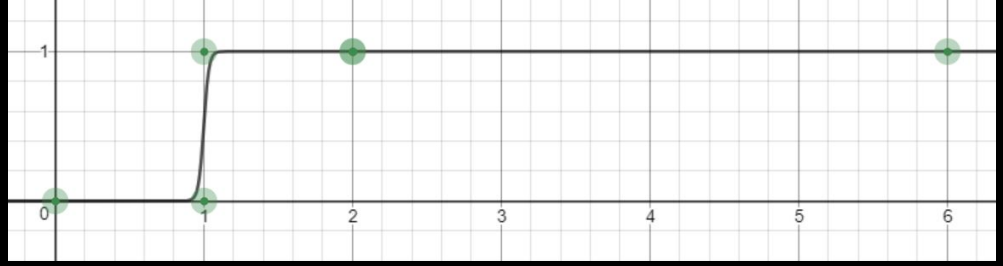
$$logit(p_i) = log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \sum_{j=1}^{m} \beta_j x_{ij} \quad =0$$



We're still learning a linear *separating hyperplane*, but fitting it to a *logit* outcome.

(https://www.linkedin.com/pulse/predicting-outcomes-probabilities-logistic-regression-konstantinidis/)

Logistic Regression



Example: Y: 1 if target is a part of a proper noun, 0 otherwise;

X1: number of capital letters in target and surrounding words.

Let's add a feature! X2: does the target word start with a capital letter?

They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

The trail was very stony. *Her degree is from SUNY Stony Brook.*

The Taylor Series was first described by Brook Taylor, the mathematician.

They attend Binghamton.

x2	x1	y
1	2	1
0	1	0
0	0	0
1	6	1
1	2	1
1	1	1

Terminology

β ≈ weight ≈ coefficient ≈ parameters ≈ θ

Logistic Regression ≈ Maximum Entropy Classifier

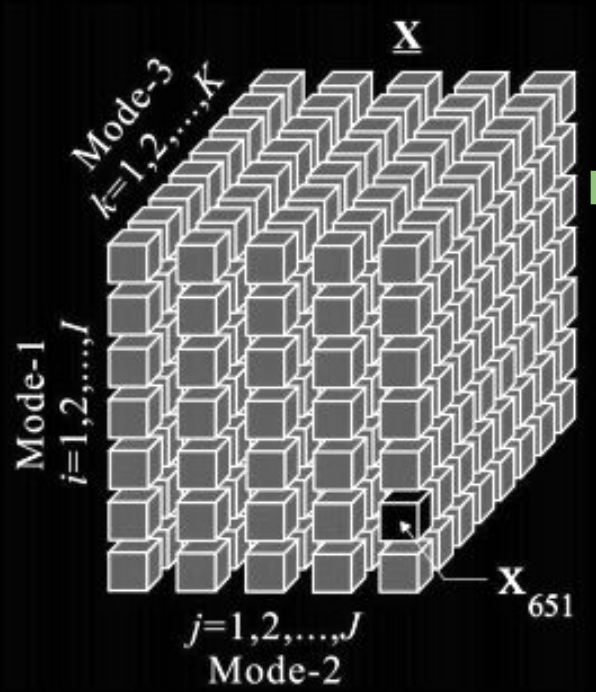
loss function ≈ cost function

Stochastic Gradient Descent ≈ Optimizer

PyTorch Intro: Logistic Regression

1. Tensors

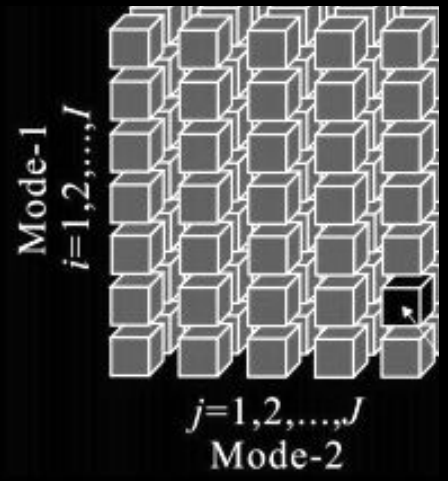
PyTorch: 1. Tensors



A multi-dimensional matrix

(i.stack.imgur.com)

PyTorch: 1. Tensors



(i.stack.imgur.com)

- A multi-dimensional matrix

A 2-d tensor is just a matrix.
- 1-d: vector
- 0-d: a constant / scalar

Note: Linguistic ambiguity:
Dimensions of a Tensor ≠ Dimensions of a Matrix

PyTorch Intro: Logistic Regression

1. Tensors
2. Numeric functions as a graph/network *(forward pass)*

PyTorch Intro: Logistic Regression

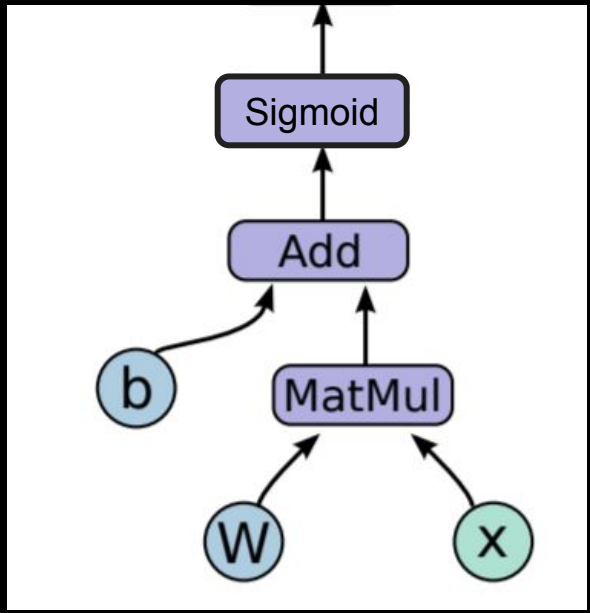
1. Tensors
2. Numeric functions as a graph/network *(forward pass)*

Why is it specifically called forward pass? Does it mean there is a backward pass?
More on this in the Back Propagation lecture in Part 2.

PyTorch: 2. Numeric functions as a graph/network (forward pass)

Efficient, high-level built-in **linear algebra** for neural network operations.

Can be conceptualized as a graph of operations on tensors (matrices):

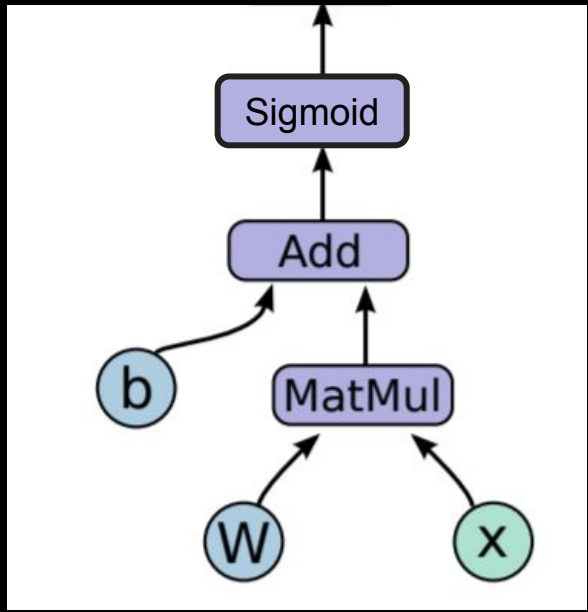


PyTorch: 2. Numeric functions as a graph/network (forward pass)

Efficient, high-level built-in **linear algebra** for neural network operations.

Can be conceptualized as a graph of operations on tensors (matrices):

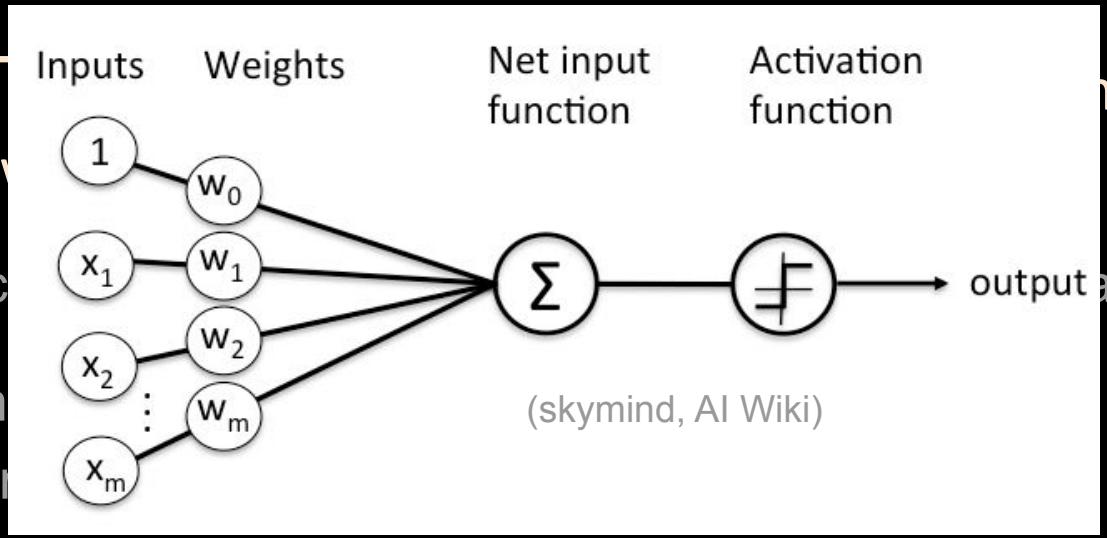
```
import torch
from torch import nn #predefined nodes

x = torch.Tensor(input)
W = torch.random.randn(x.shape[1], 1) #weights
z = torch.matmul(x, beta)
yhat = nn.functional.sigmoid(z)
```



Py
(for
network

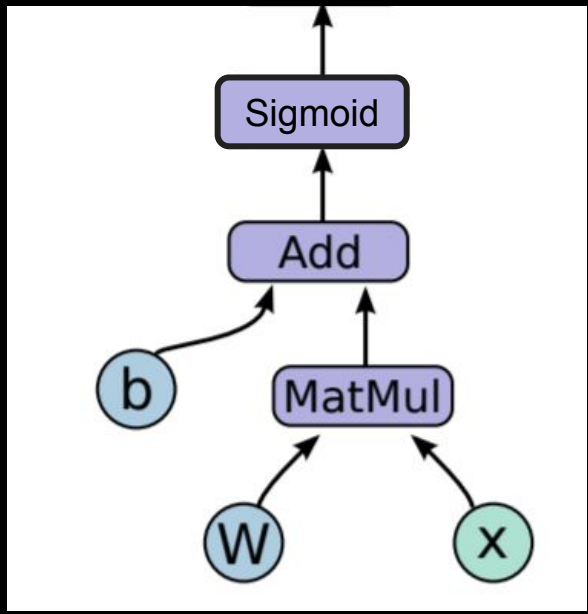
Effic
al network operations.

Can
oper



(skymind, AI Wiki)

```
import torch
from torch import nn #predefined nodes

x = torch.Tensor(input)
w = torch.random.randn(X.shape[1], 1) #weights
b = torch.random.randn(1, 1) #intercept
z = torch.matmul(x, w)
z += b
yhat = nn.functional.sigmoid(z)
```



PyTorch: 2. Numeric functions as a graph/network (forward pass: defined in "forward" method of nn.Module)

```
class LogReg(nn.Module):
    def __init__(self, num_feats,
                 learn_rate = 0.01, device = torch.device("cpu") ):
        #the constructor; define any layer objects (e.g. Linear)
        super(LogReg, self).__init__()
        self.linear = nn.Linear(num_feats+1, 1)
```

PyTorch: 2. Numeric functions as a graph/network (forward pass: defined in "forward" method of nn.Module)

```
class LogReg(nn.Module):
    def __init__(self, num_feats,
                 learn_rate = 0.01, device = torch.device("cpu") ):
        #the constructor; define any layer objects (e.g. Linear)
        super(LogReg, self).__init__()
        self.linear = nn.Linear(num_feats+1, 1)
```

The nn.Module class registers the weights and biases in self.linear variable as Parameters of the model.

These weights and biases are stored as special tensors called the nn.Parameters, which allow them to have other member objects that support training of models.

PyTorch: 2. Numeric functions as a graph/network (forward pass: defined in "forward" method of nn.Module)

```
class LogReg(nn.Module):
    ...

    def forward(self, X):
        #This is where the model itself is defined.
        #For logistic regression the model takes in X and returns
        #the results of a decision function

        newX = torch.cat((X, torch.ones(X.shape[0], 1)), 1) #add intercept
        z = self.linear(newX)
        return nn.functional.sigmoid(z)
                                #logistic function on the linear output
```

PyTorch: 2. Numeric functions as a graph/network (forward pass: defined in "forward" method of nn.Module)

```
class LogReg(nn.Module):
    def __init__(self, num_feats,
                 learn_rate = 0.01, device = torch.device("cpu") ):
        #the constructor; define any layer objects (e.g. Linear)
        super(LogReg, self).__init__()
        self.linear = nn.Linear(num_feats+1, 1)

    def forward(self, X):
        #This is where the model itself is defined.
        #For logistic regression the model takes in X and returns
        #the results of a decision function

        newX = torch.cat((X, torch.ones(X.shape[0], 1)), 1)
                                    #add intercept
        z = self.linear(newX)
        return nn.functional.sigmoid(z)
                                    #logistic function on the linear output
```

PyTorch Intro: Logistic Regression

1. Tensors
2. Numeric functions as a graph/network (forward pass)
3. Loss function (training loop)

PyTorch: 3. Loss Function (training loop)

```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
def loss_func(ypred, y):
     return torch.mean(-torch.sum(y*torch.log(y_pred)))
               + torch.mean(-torch.sum((1-y)*torch.log(1-y_pred)))
#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = model(X)
    loss = loss_func(ypred, y)
    #backward: /(applies gradient descent)
    loss.backward()
    sgd.step()
    if i % 20 == 0:
        print("  epoch: %d, loss: %.5f" %(i, loss.item()))
```

PyTorch: 3. Loss Function (training loop)

```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss() #computationally optimized for GPUs

#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = model(X)
    loss = loss_func(ypred, y)
    #backward: /(applies gradient descent)
    loss.backward()
    sgd.step()

    if i % 20 == 0:
        print("  epoch: %d, loss: %.5f" %(i, loss.item()))
```

PyTorch Intro: Logistic Regression

1. Tensors
2. Numeric functions as a graph/network (forward pass)
3. Loss function (training loop)
4. Autograd (backward pass)

PyTorch: 3. Loss Function (training loop)

```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()

#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = model(X)
    loss = loss_func(ypred, y)
    #backward: /(applies gradien
    loss.backward()
    sgd.step()

    if i % 20 == 0:
        print("  epoch: %d, loss: %.5f" %(i, loss.item()))
```

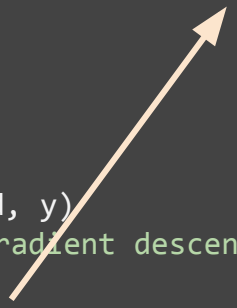
To Optimize Betas (all weights/parameters within the neural net):

Stochastic Gradient Descent (SGD)
-- optimize over one sample each iteration

Mini-Batch SDG:
--optimize over *b* samples each iteration

PyTorch: 4. Autograd (backward pass)

```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()

#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = model(X)
    loss = loss_func(ypred, y)
    #backward: /(applies gradient descent
    loss.backward()
    sgd.step()

    if i % 20 == 0:
        print("  epoch: %d, loss: %.5f" %(i, loss.item()))
```



Calling loss.backward() will trigger the torch autograd to traverse the submodules within nn.Module class and compute the gradients.

The gradients wrt each parameter are stored under *.grad* object of the nn.Parameters type.

PyTorch: 4. Autograd (backward pass)

```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()

#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = model(X)
    loss = loss_func(ypred, y)
    #backward: /(applies gradient descer
    loss.backward()
    sgd.step()

    if i % 20 == 0:
        print("  epoch: %d, loss: %.5f" %(i, loss.item()))
```

The optimizer object's .step() uses the gradients stored in model Parameters to carry out the weight updates.

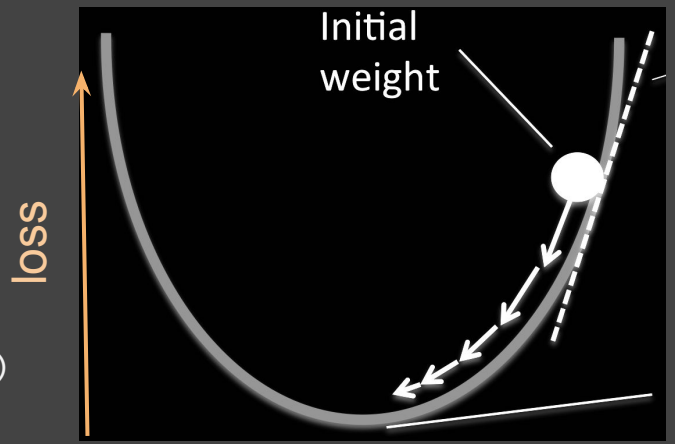
PyTorch: 4. Autograd (backward pass)

```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()

#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = model(X)
    loss = loss_func(ypred, y)
    #backward: /(applies gradient descent)
    loss.backward()
    sgd.step()

    if i % 20 == 0:
        print("  epoch: %d, loss: %.5f" %(i, loss.item()))
```

The optimizer object's .zero_grad() resets the gradients to 0.

PyTorch: 4. Autograd (backward pass)

```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()

#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = model(X)
    loss = loss_func(ypred, y)
    #backward: /(applies gradient descent)
    loss.backward()
    sgd.step()

    if i % 20 == 0:
        print("  epoch: %d, loss: %.5f" %(i, loss.item()))
```



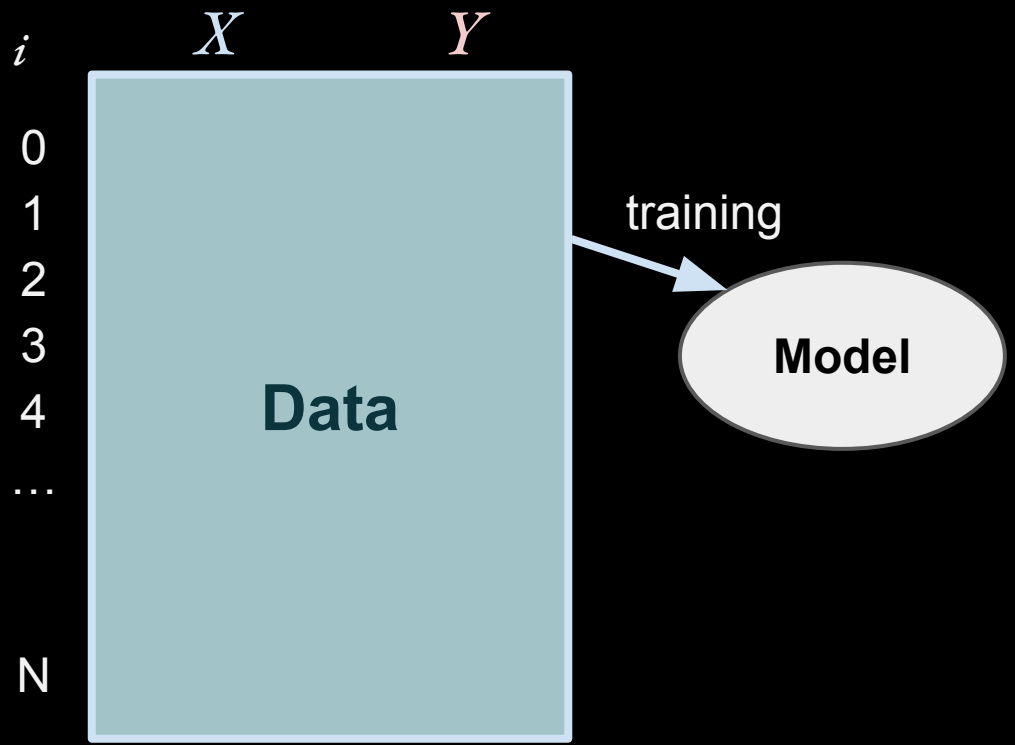
PyTorch Intro: Logistic Regression

1. Tensors
2. Numeric functions as a graph/network (forward pass)
   `nn.module` object maps *X* to *y_pred*

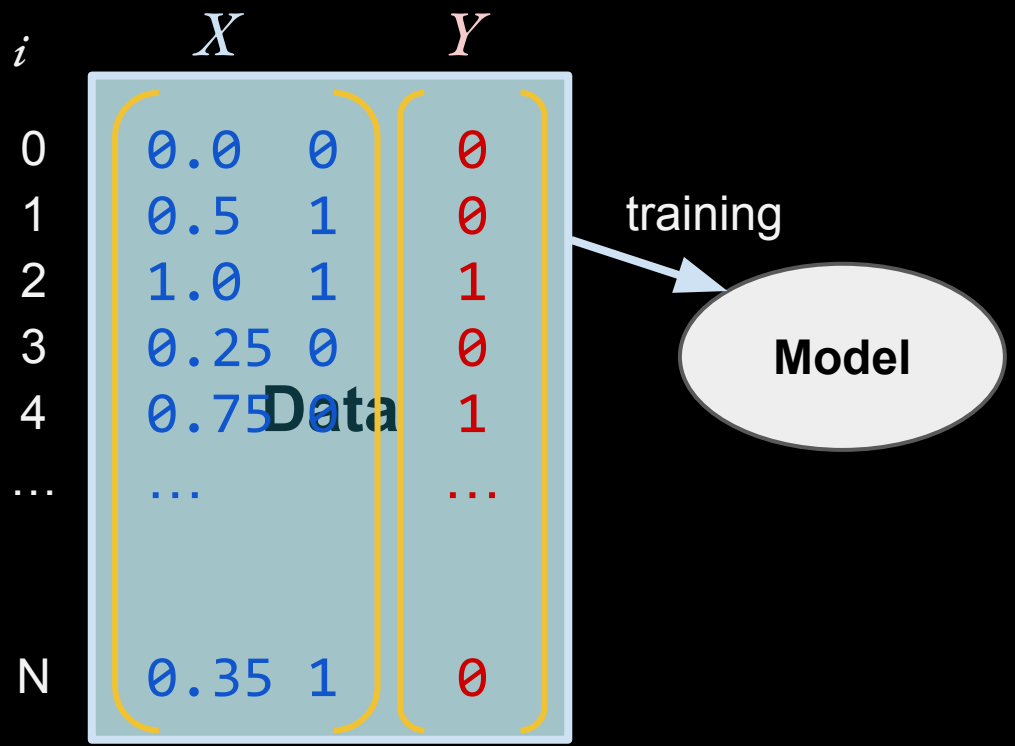
3. Loss function (training loop)
   `loop` that evaluates *ypred* versus *y*

4. Autograd (backward pass)
   `torch computation` that updates the parameters

Machine Learning: How to setup data



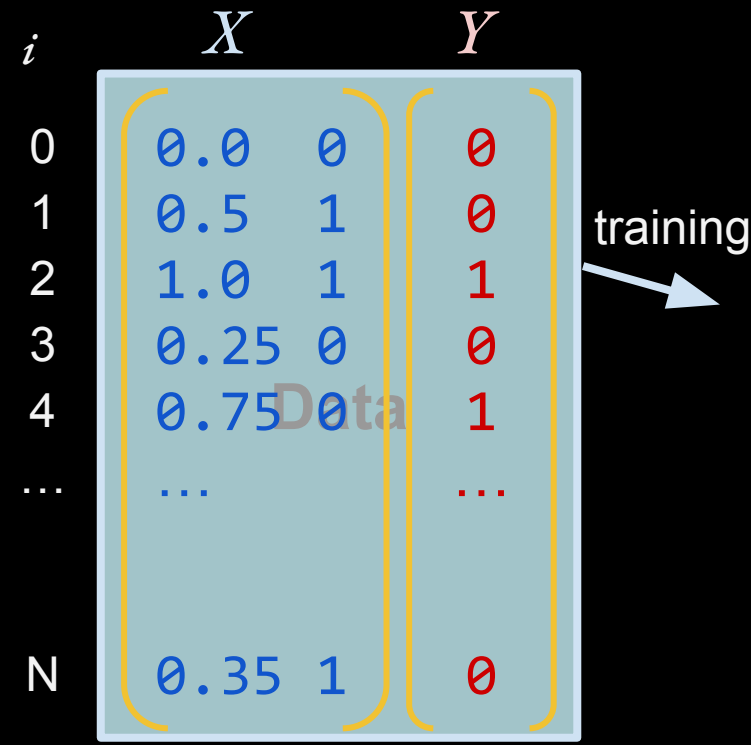
Machine Learning: How to setup data



Machine Learning: How to setup data

"Corpus"

raw data:
sequences of
characters



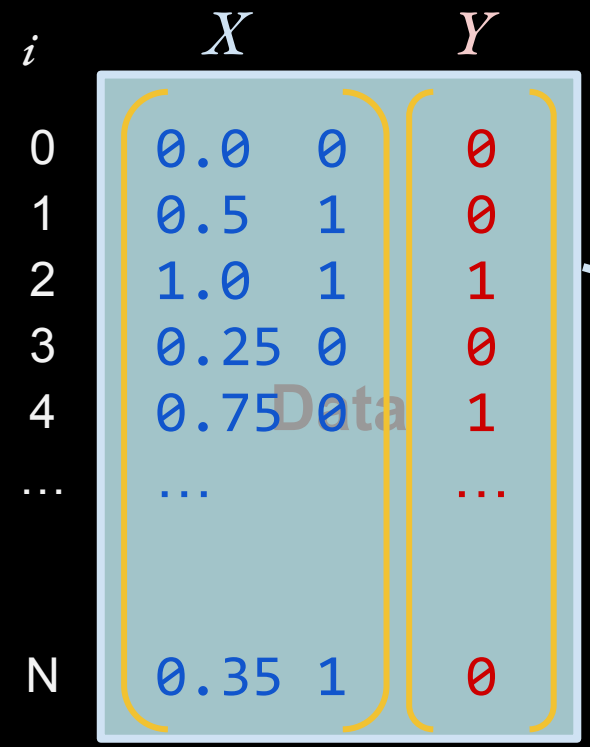
Machine Learning: How to setup data

Feature Extraction

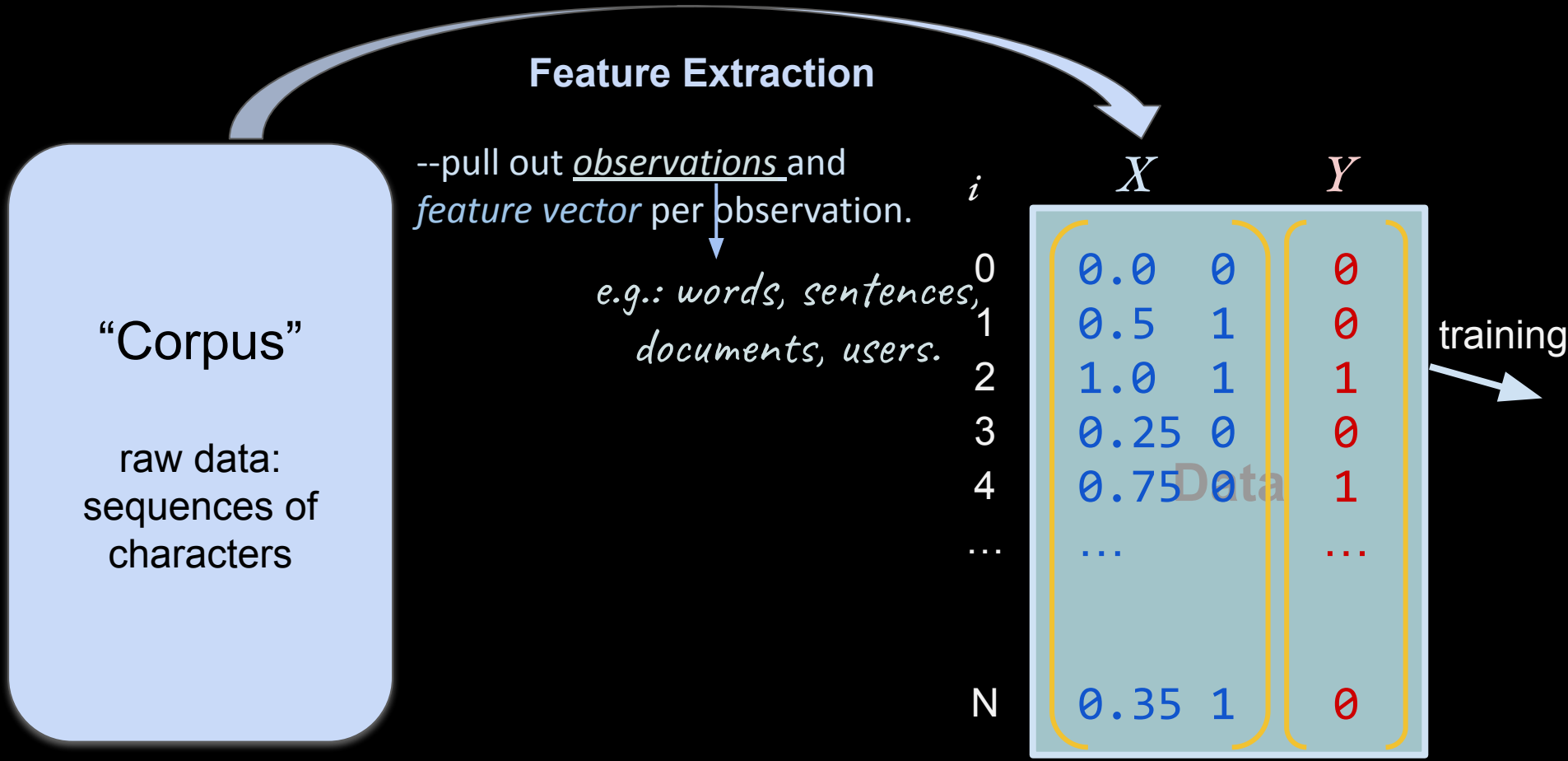
--pull out *observations* and *feature vector* per observation.

"Corpus"

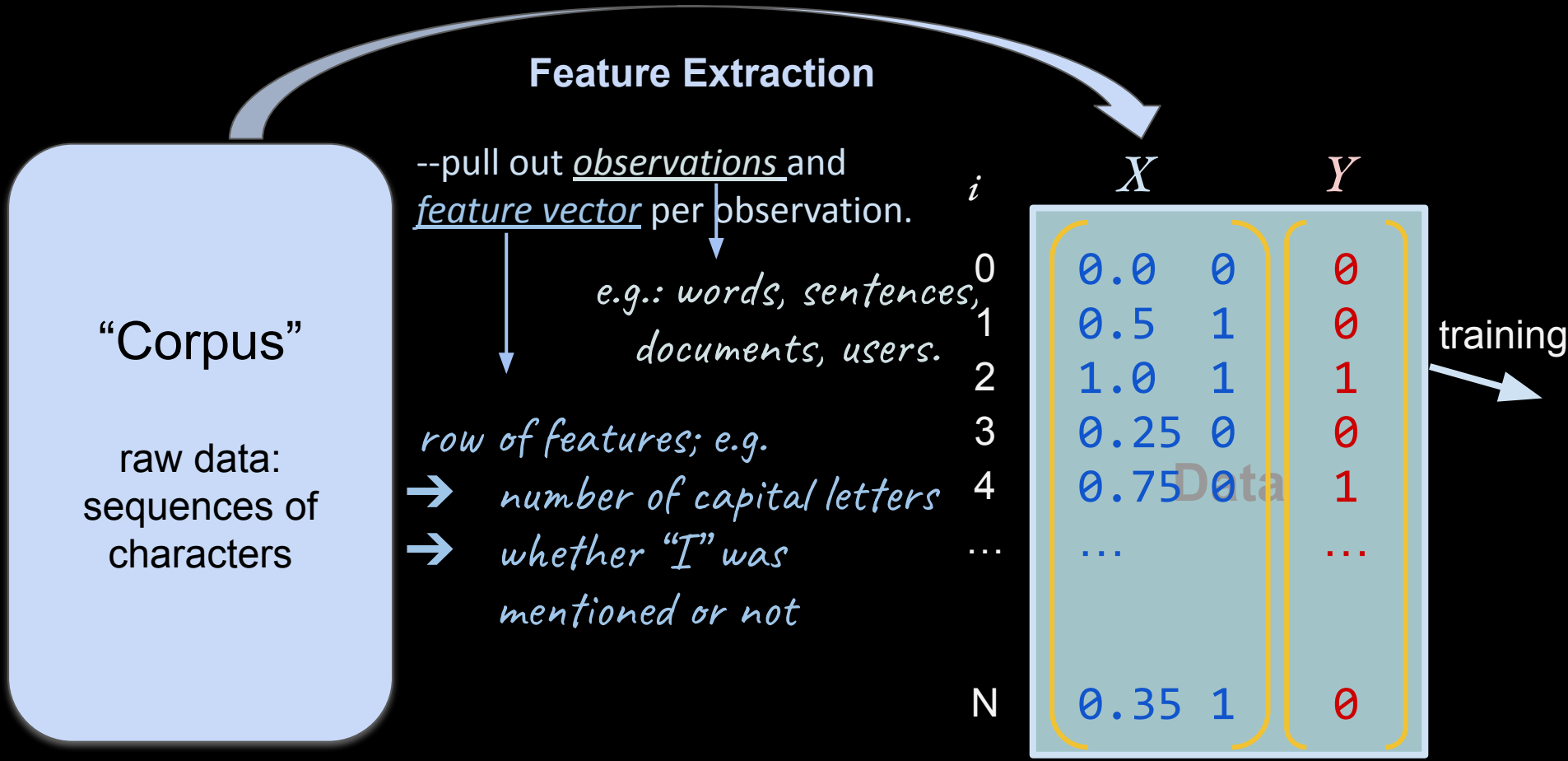
raw data: sequences of characters



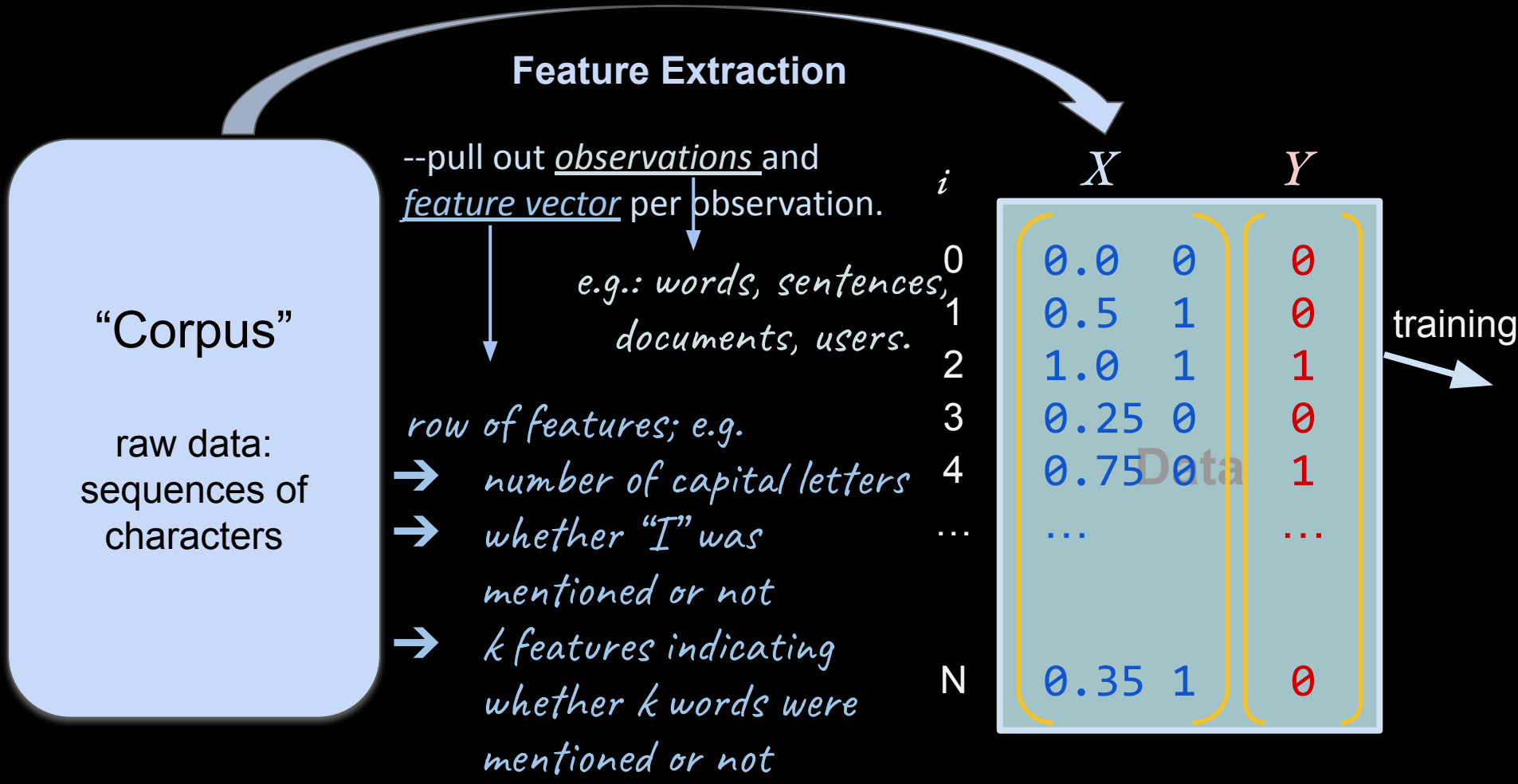
Machine Learning: How to setup data



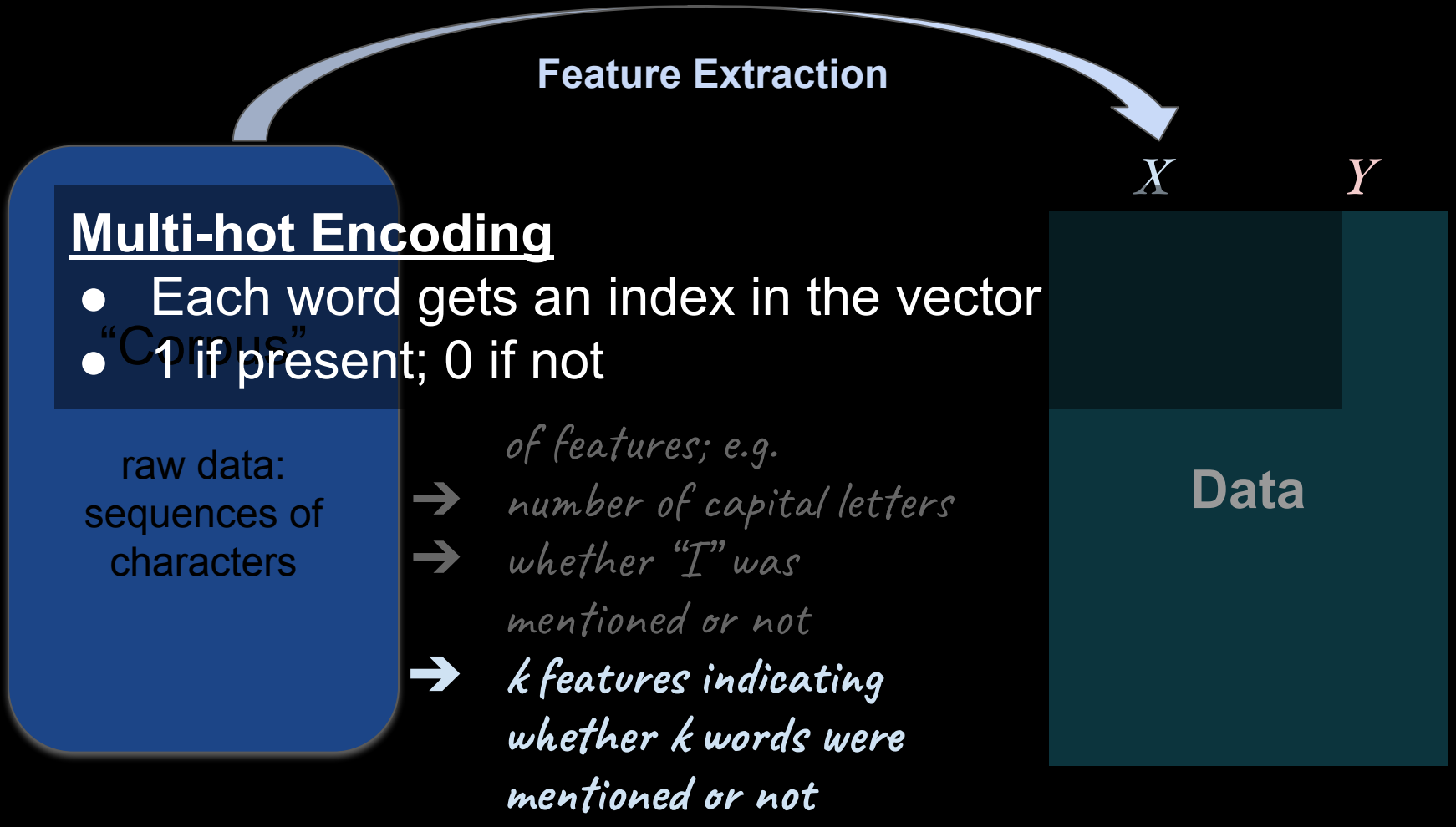
Machine Learning: How to setup data



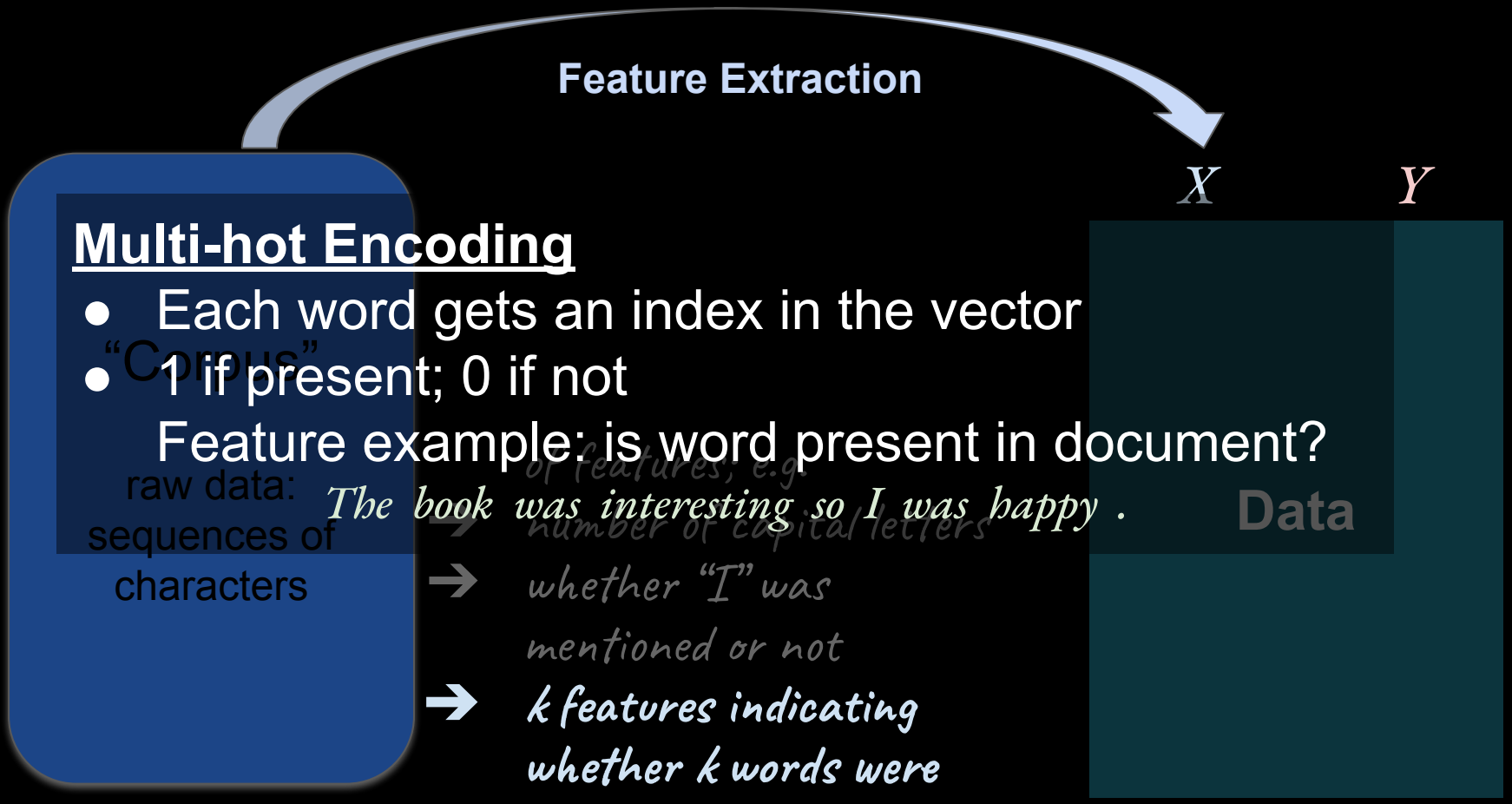
Machine Learning: How to setup data



Machine Learning: How to setup data



Machine Learning: How to setup data

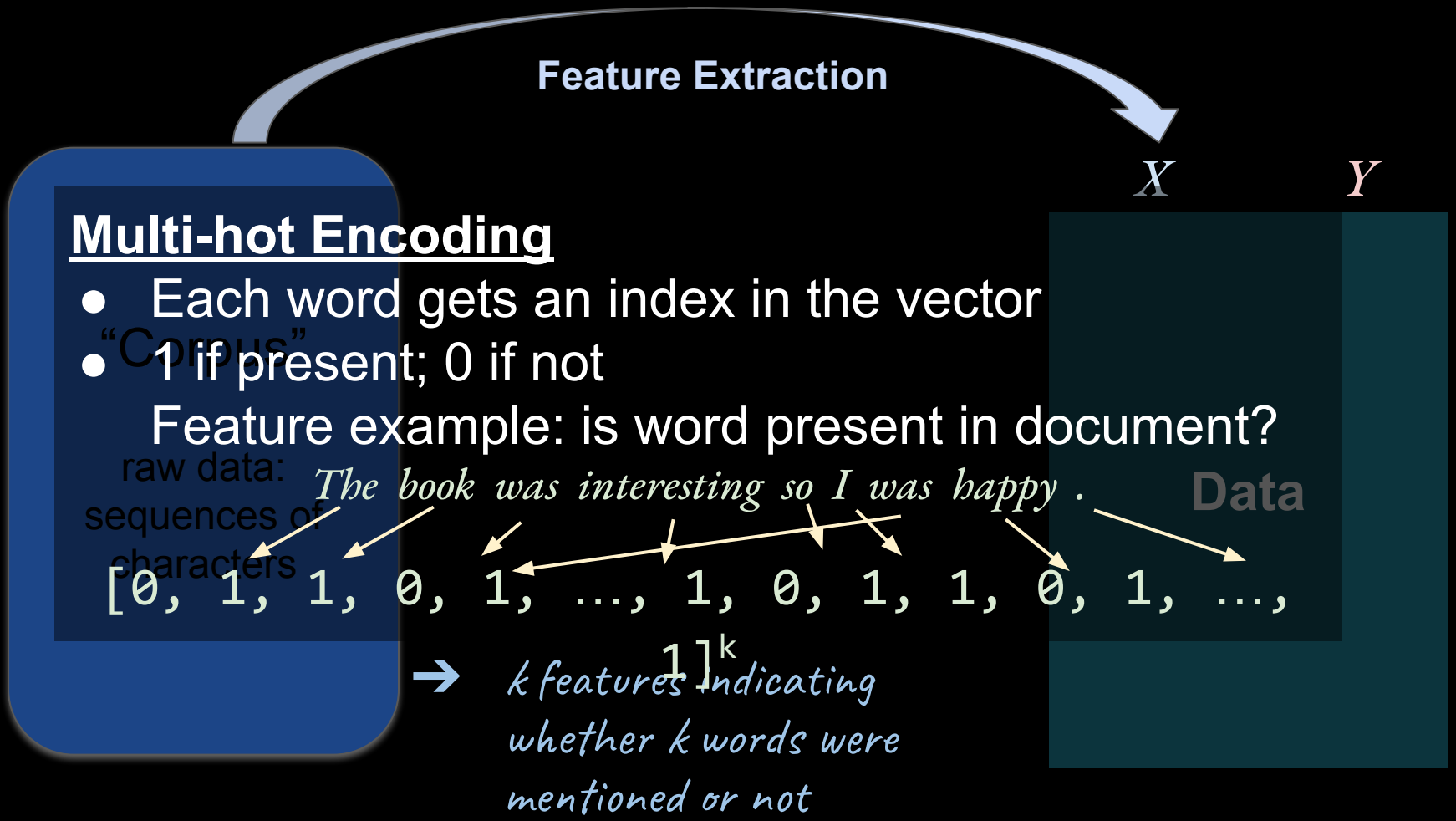


Multi-hot Encoding

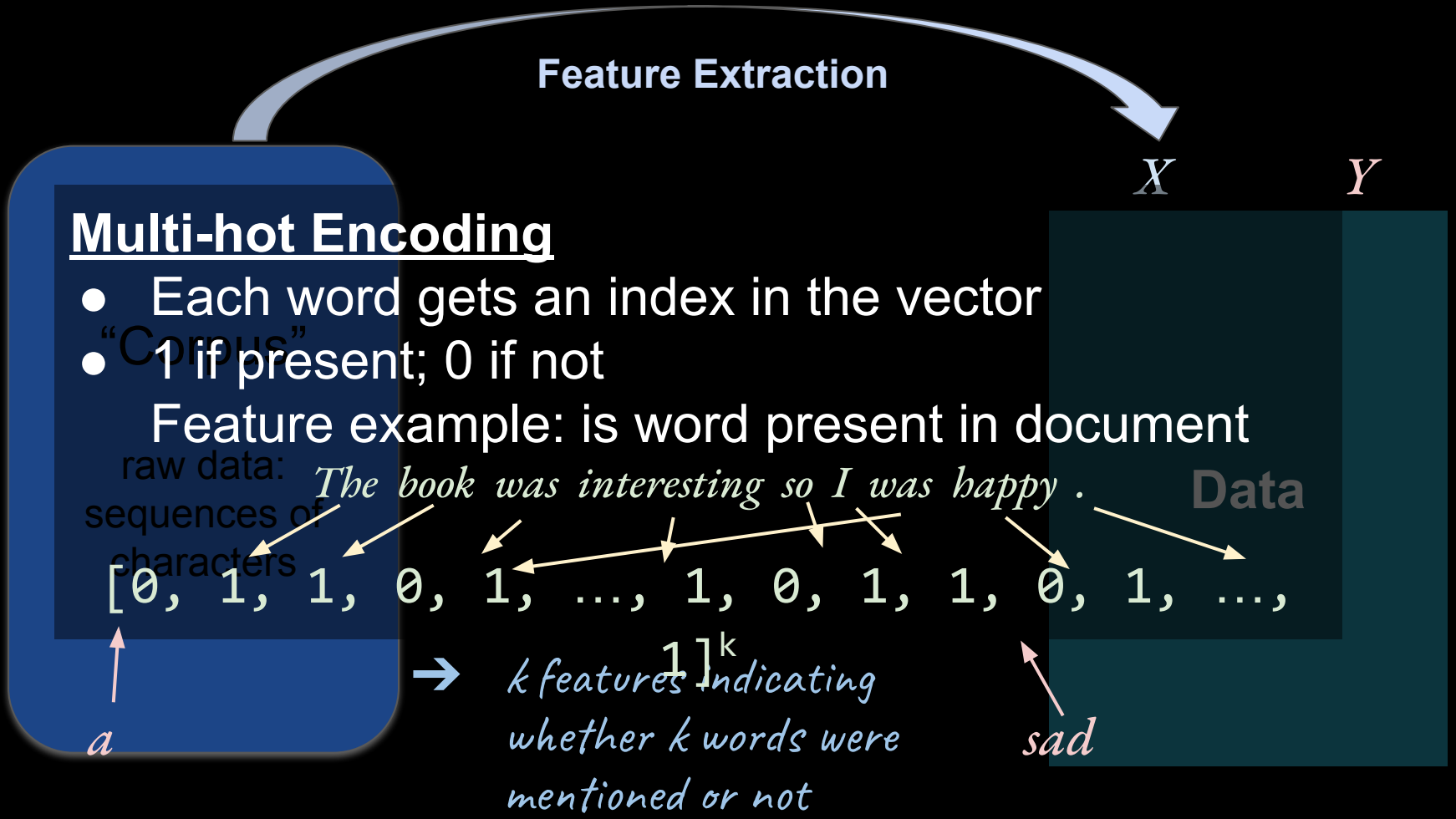
- Each word gets an index in the vector
- 1 if present; 0 if not

  Feature example: is word present in document?

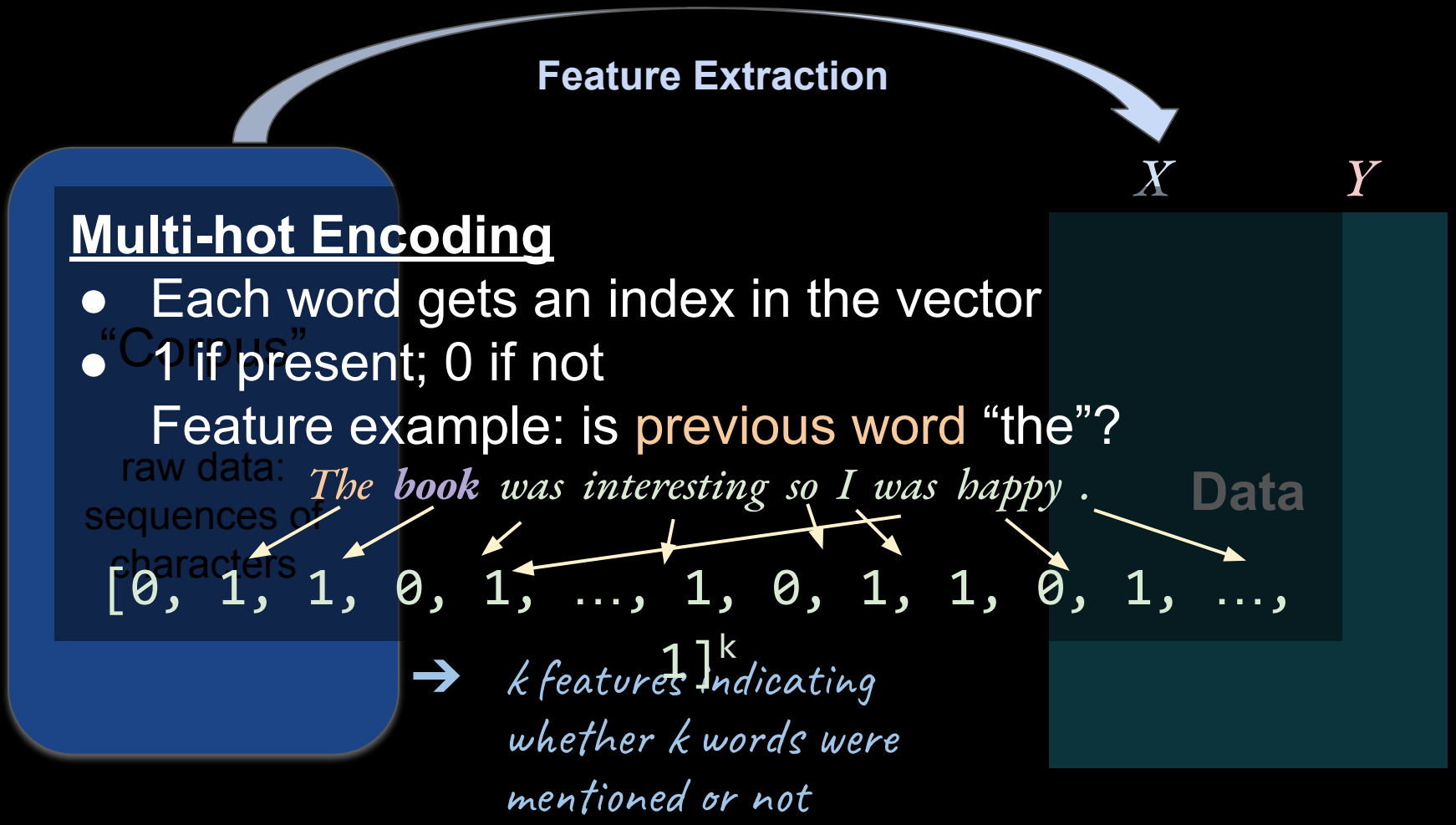
Machine Learning: How to setup data



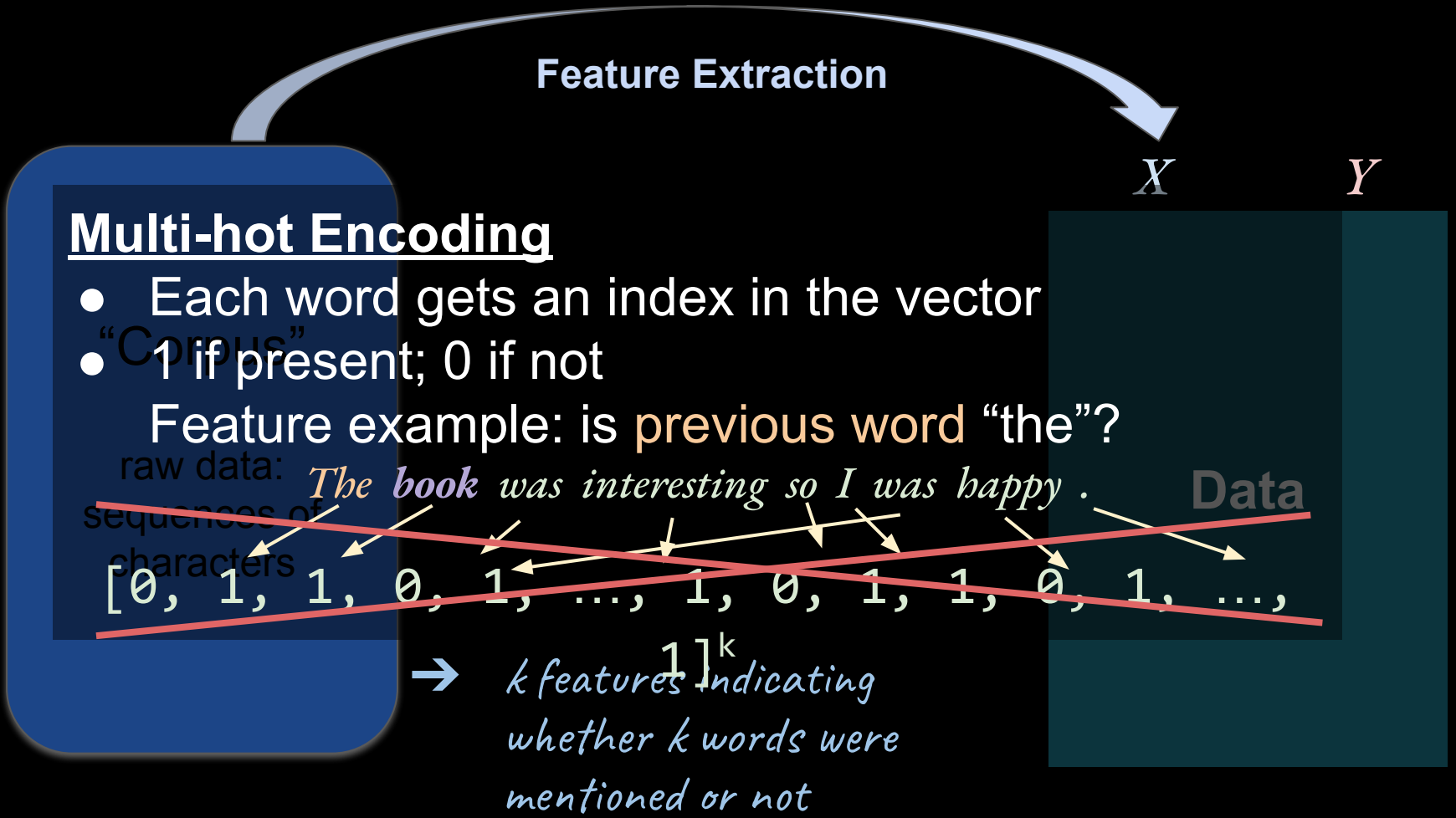
Machine Learning: How to setup data



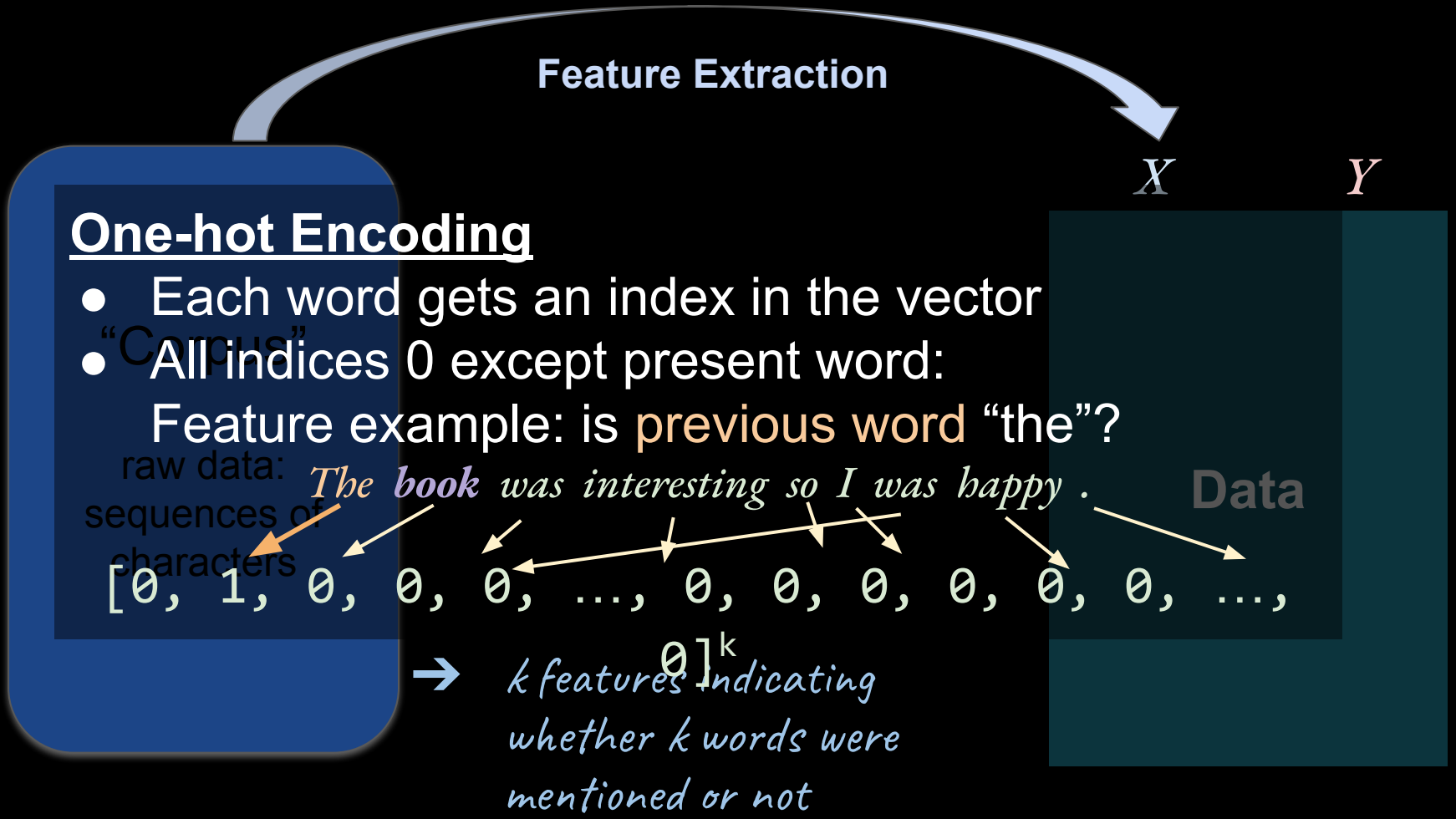
Machine Learning: How to setup data



Machine Learning: How to setup data



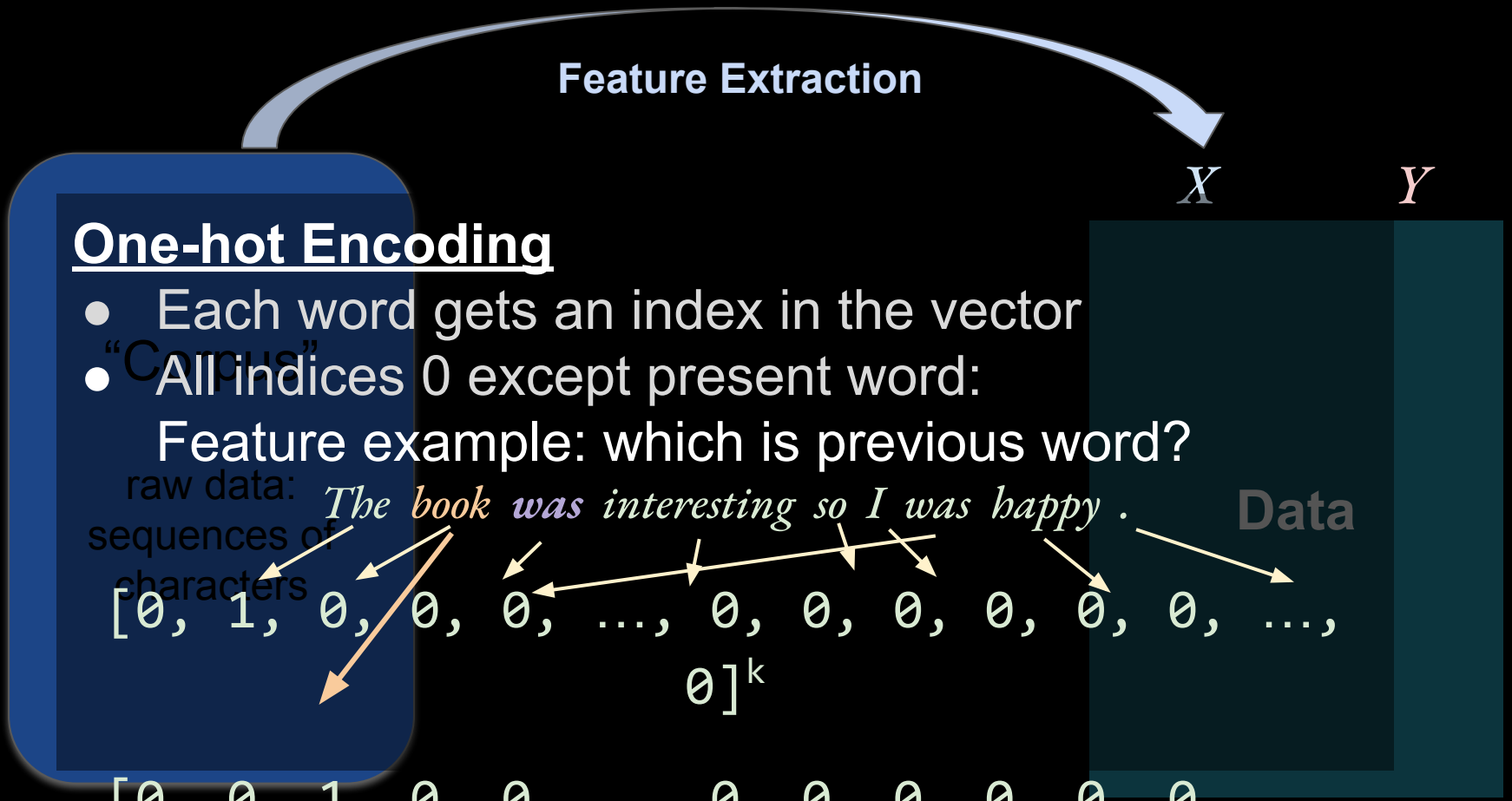
Machine Learning: How to setup data



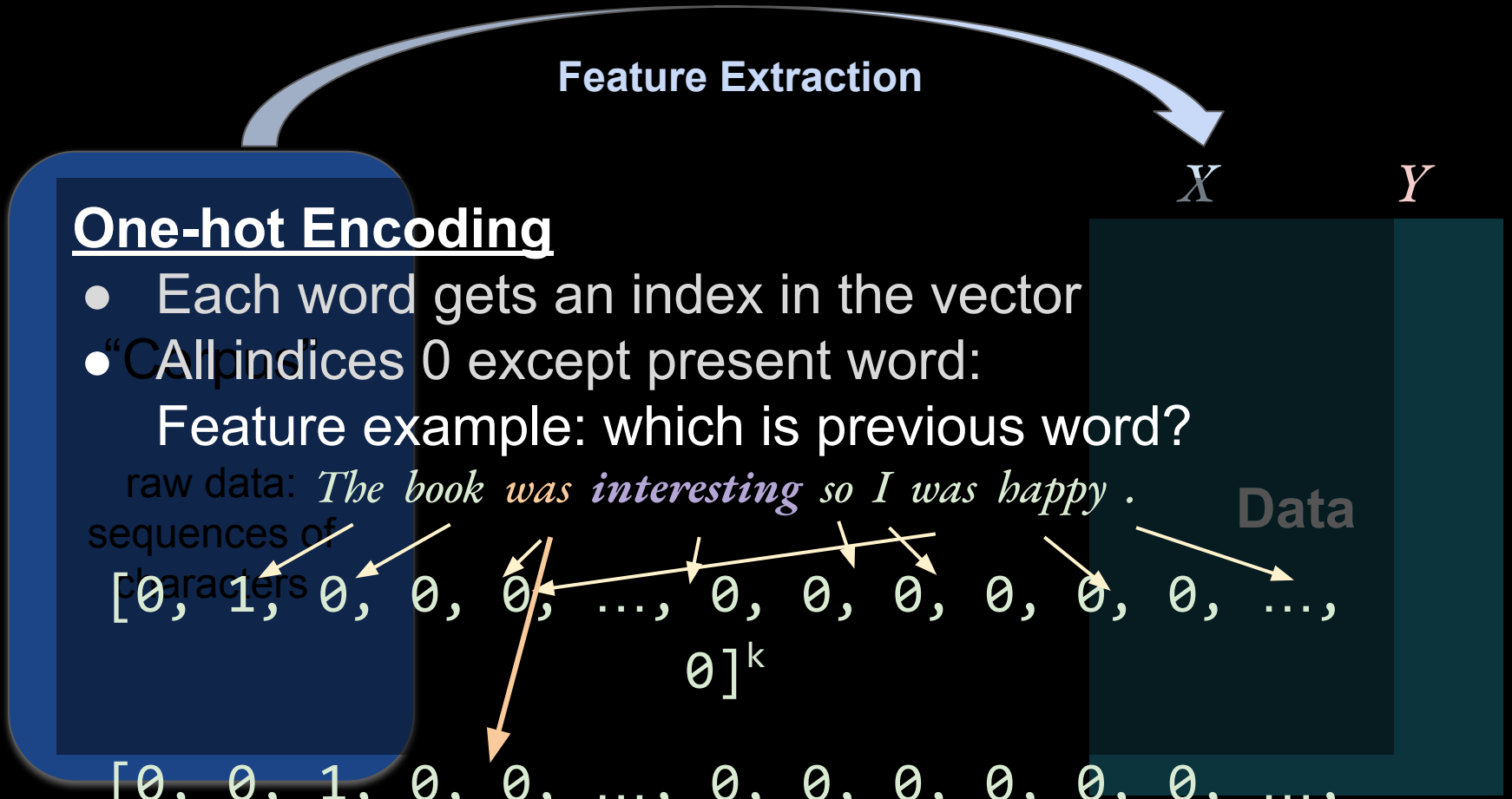
One-hot Encoding

- Each word gets an index in the vector
- All indices 0 except present word:
  Feature example: is previous word "the"?

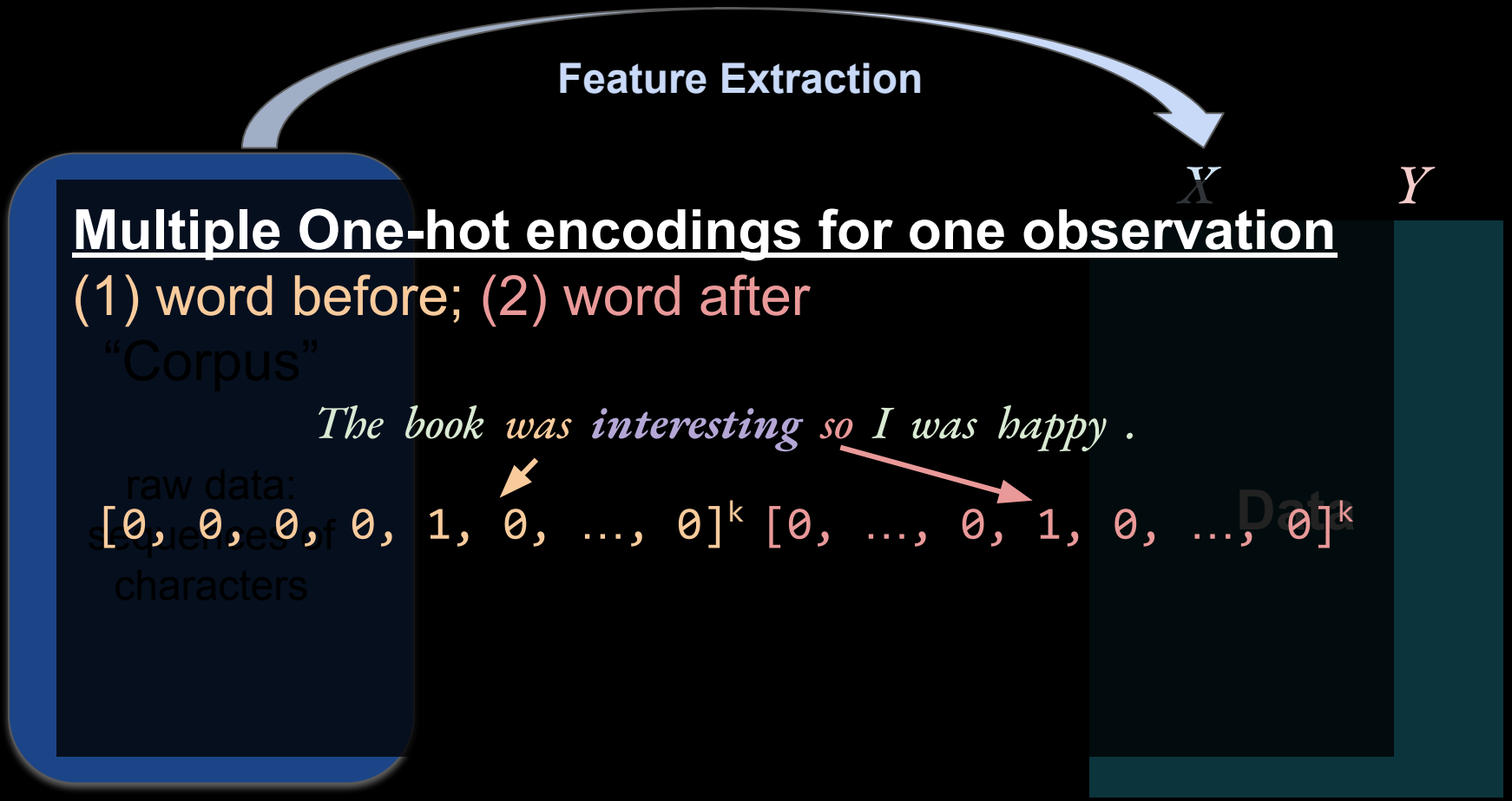
Machine Learning: How to setup data



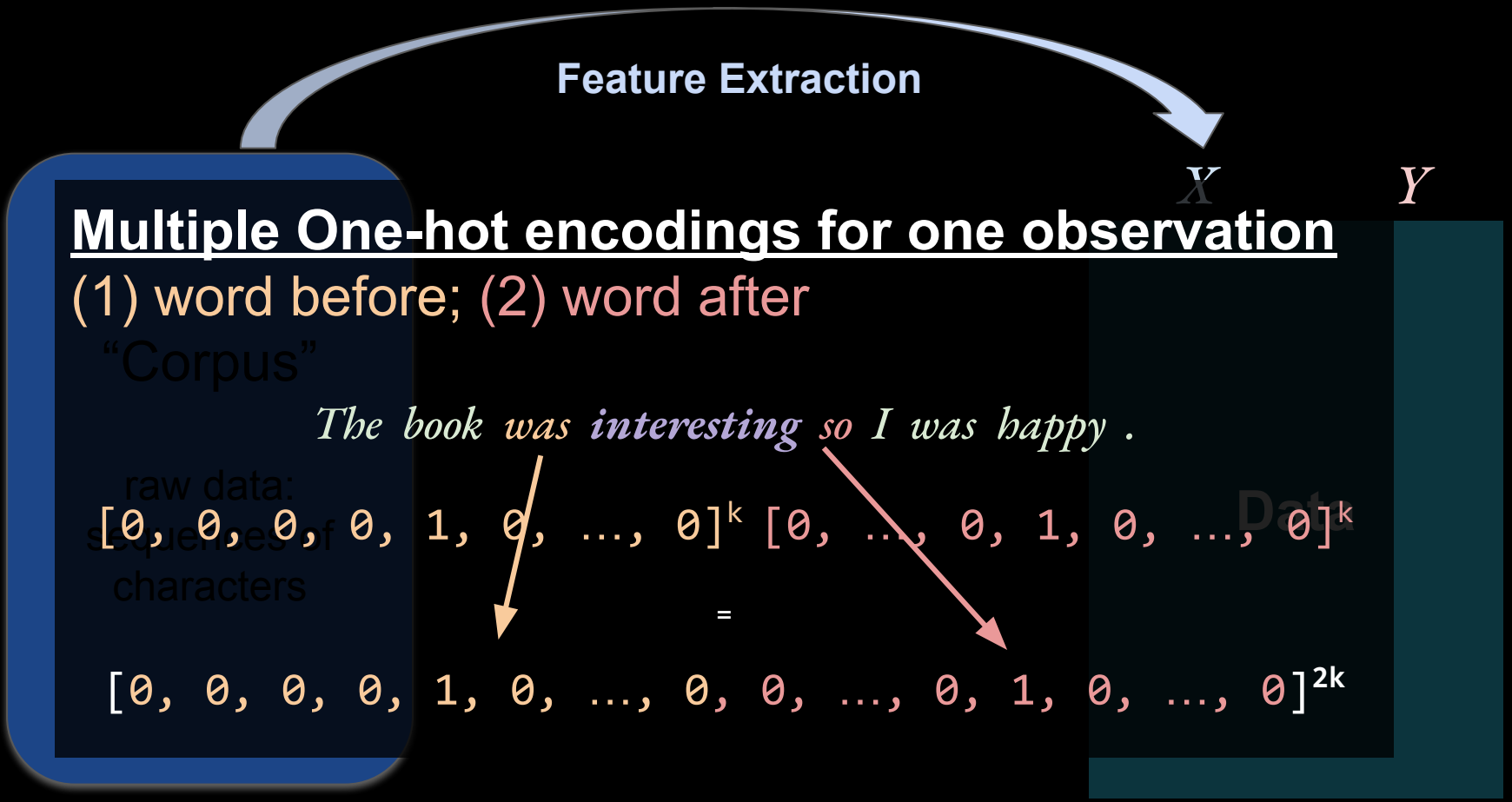
Machine Learning: How to setup data



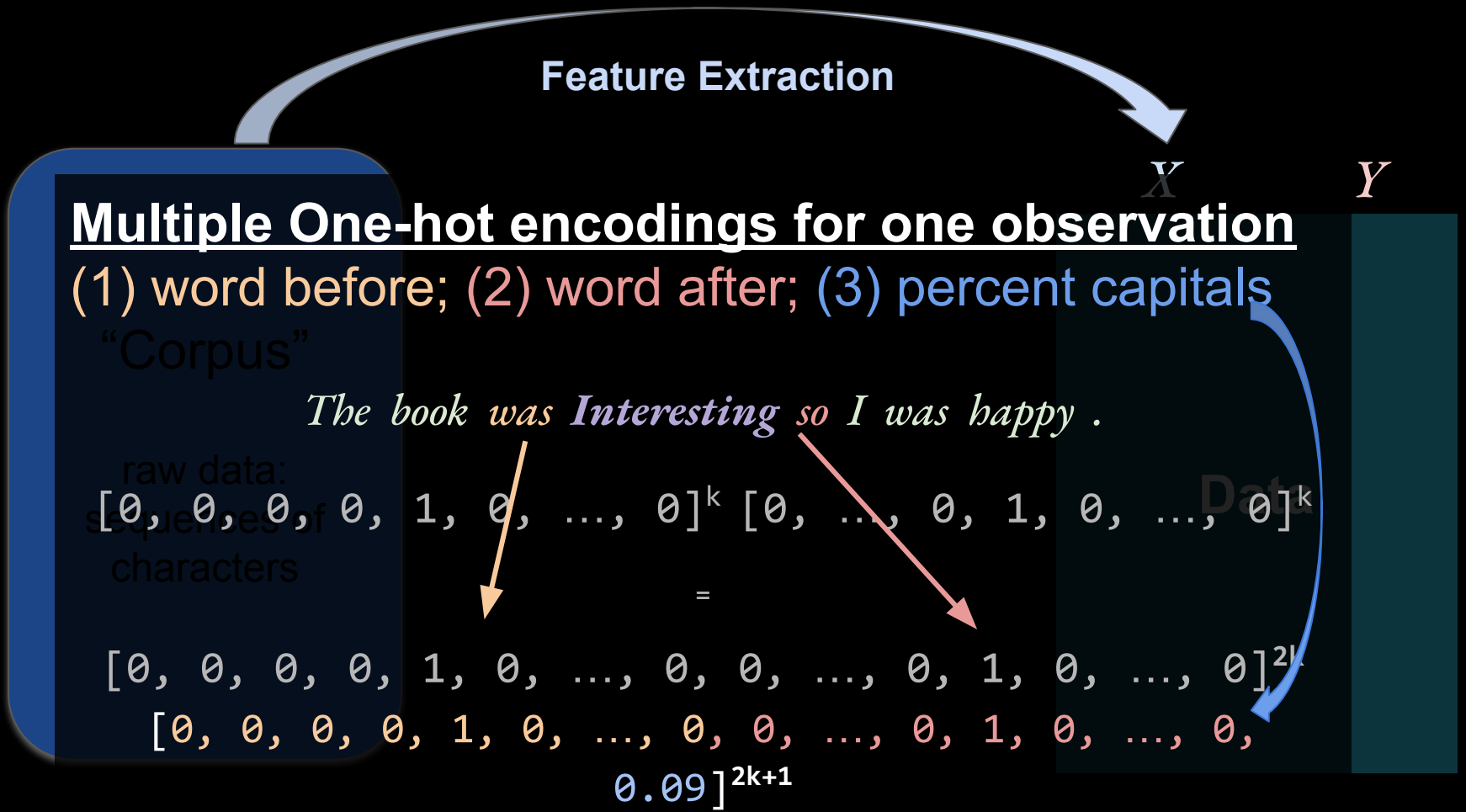
Machine Learning: How to setup data



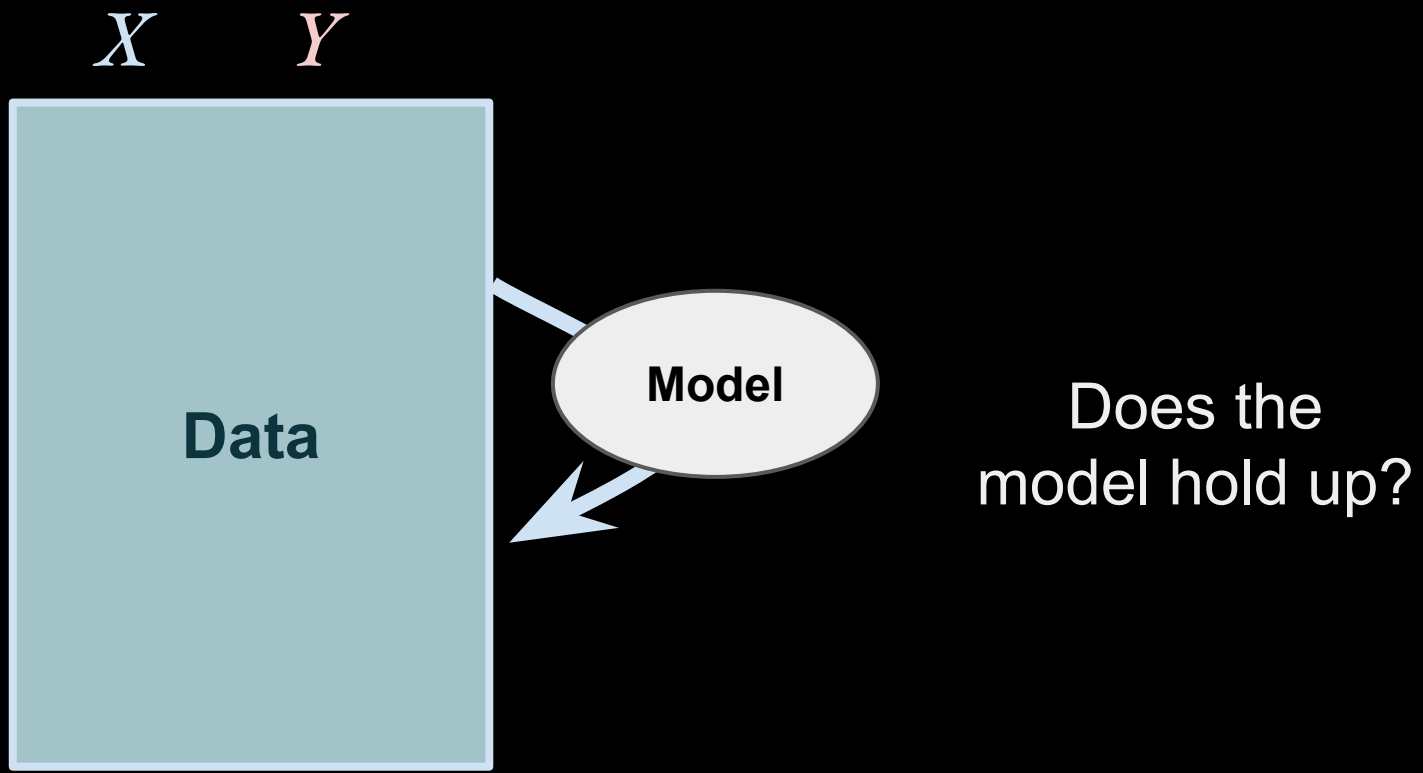
Machine Learning: How to setup data



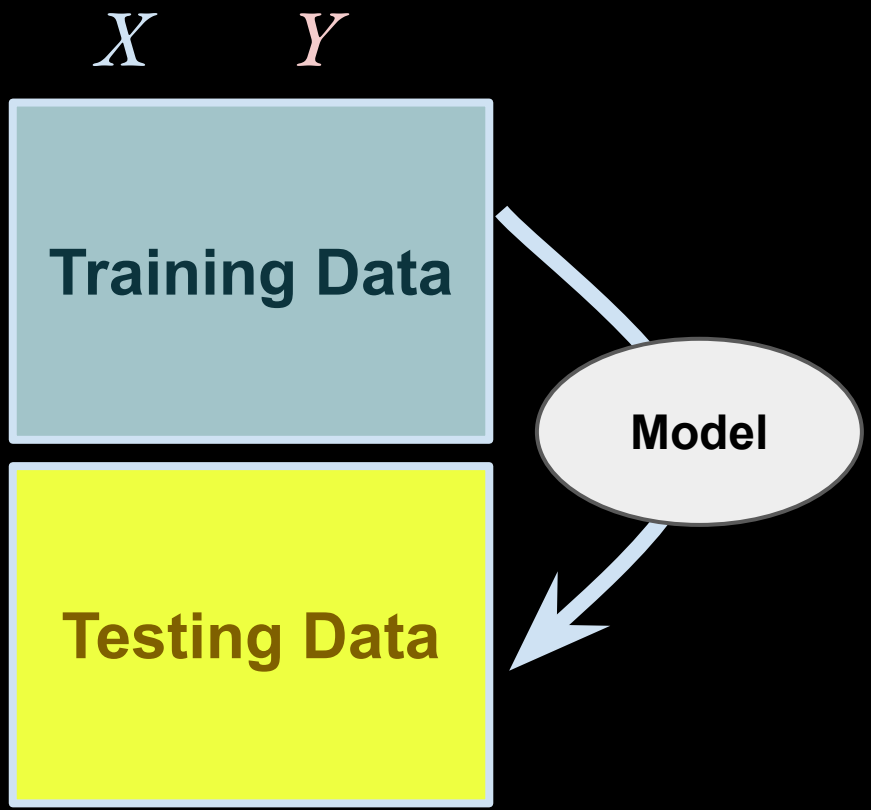
Machine Learning: How to setup data



Machine Learning: How to setup data

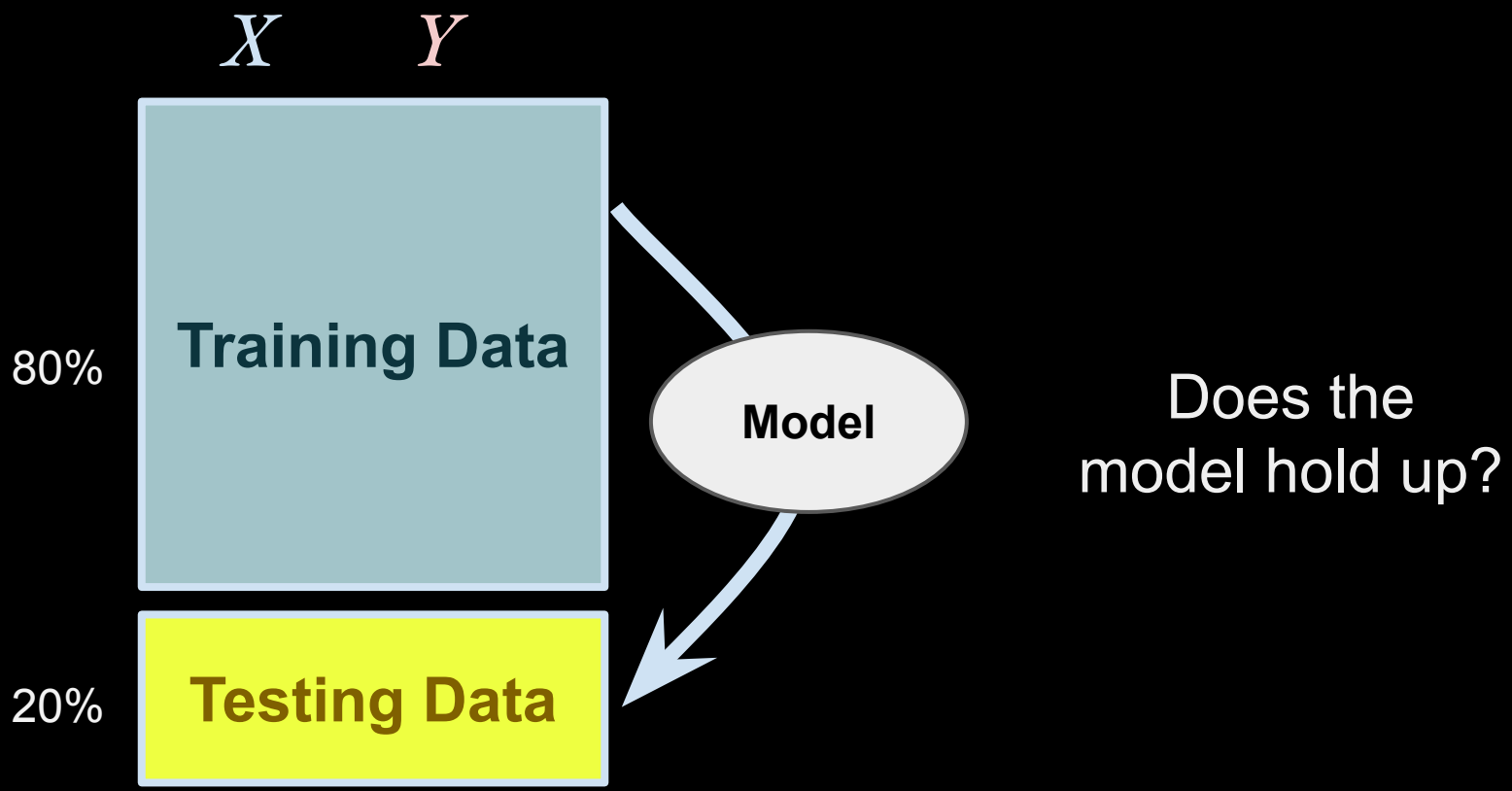


Machine Learning Goal: Generalize to new data



Does the model hold up?

Machine Learning Goal: Generalize to new data



Logistic Regression - Regularization

X						$=$	Y
0.5	0	0.6	1	0	0.25		1
0	0.5	0.3	0	0	0		1
0	0	1	1	1	0.5		0
0	0	0	0	1	1		0
0.25	1	1.25	1	0.1	2		1

Logistic Regression - Regularization

X						$=$	Y
0.5	0	0.6	1	0	0.25		1
0	0.5	0.3	0	0	0		1
0	0	1	1	1	0.5		0
0	0	0	0	1	1		0
0.25	1	1.25	1	0.1	2		1

Logistic Regression - Regularization

x_1	x_2	...	X			=	Y
0.5	0	0.6	1	0	0.25		1
0	0.5	0.3	0	0	0		1
0	0	1	1	1	0.5		0
0	0	0	0	1	1		0
0.25	1	1.25	1	0.1	2		1

$1.2 +$	$-63*x_1 +$	$179*x_2 +$	$71*x_3 +$	$18*x_4 +$	$-59*x_5 +$	$19*x_6 =$	$logit(Y)$

Logistic Regression - Regularization

x_1	x_2	...	X			=	Y
0.5	0	0.6	1	0	0.25		1
0	0.5	0.3	0	0	0		1
0	0	1	1	1	0.5		0
0	0	0	0	1	1		0
0.25	1	1.25	1	0.1	2		1

$1.2 +$	$-63*x_1 +$	$179*x_2 +$	$71*x_3 +$	$18*x_4 +$	$-59*x_5 +$	$19*x_6 =$	$logit(Y)$

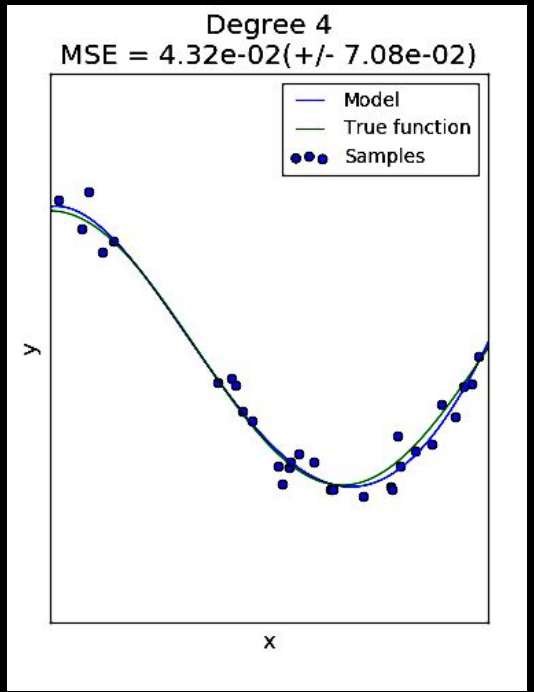
Logistic Regression - Regularization

x_1	x_2	...	X			= Y
0.5	0	0.6	1	0	0.25	1
0	0.5	0.3	0		0	1
0	0				0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

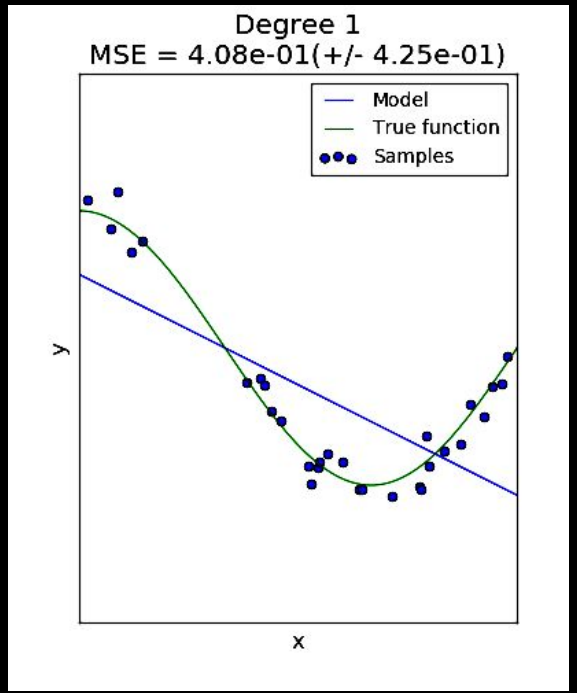
"overfitting"

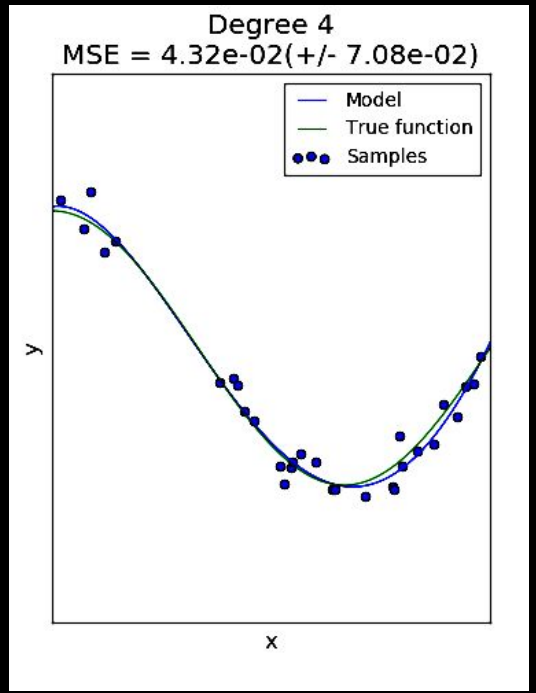
$$1.2 + -63*x_1 + 179*x_2 + 71*x_3 + 18*x_4 + -59*x_5 + 19*x_6 = logit(Y)$$

Overfitting (1-d non-linear example)



Overfitting (1-d non-linear example)

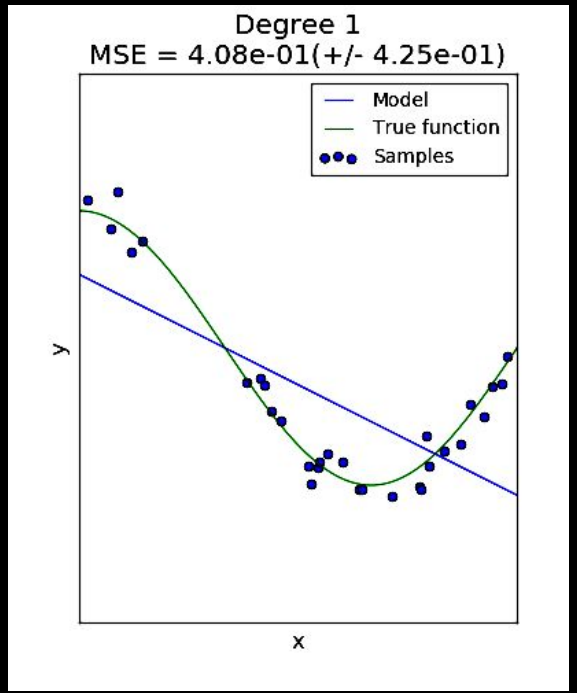


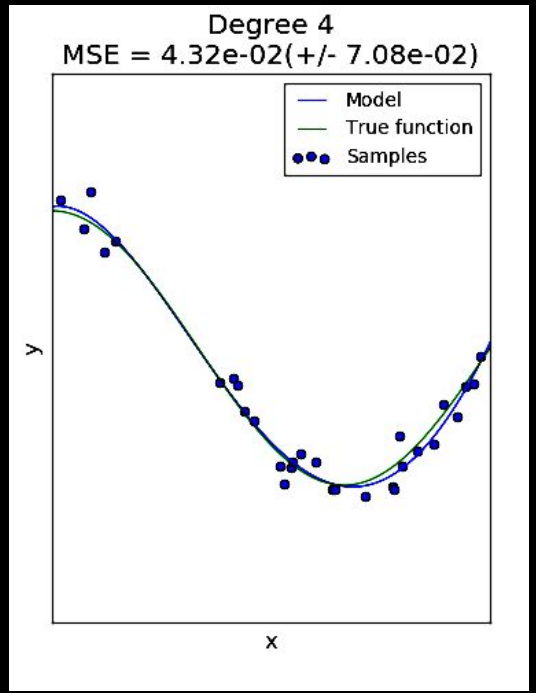


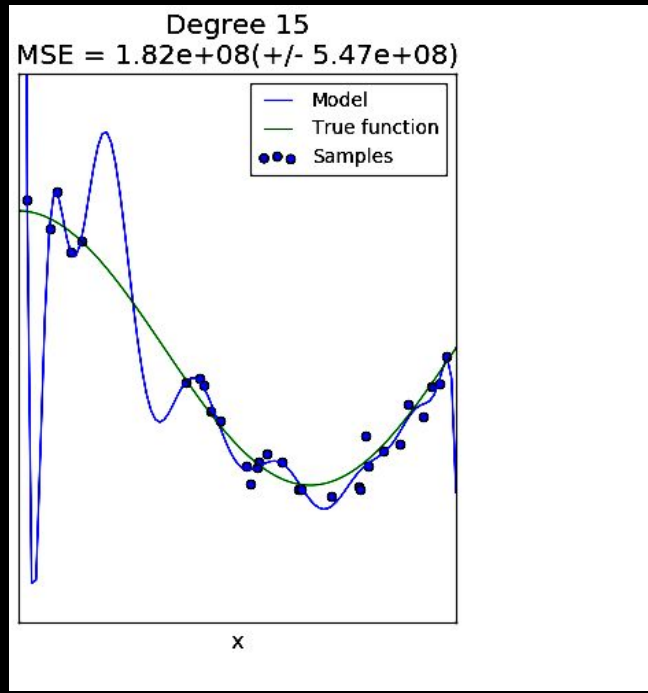
Underfit

(image credit: Scikit-learn; in practice data are rarely this clear)

Overfitting (1-d non-linear example)







Underfit

Overfit

(image credit: Scikit-learn; in practice data are rarely this clear)

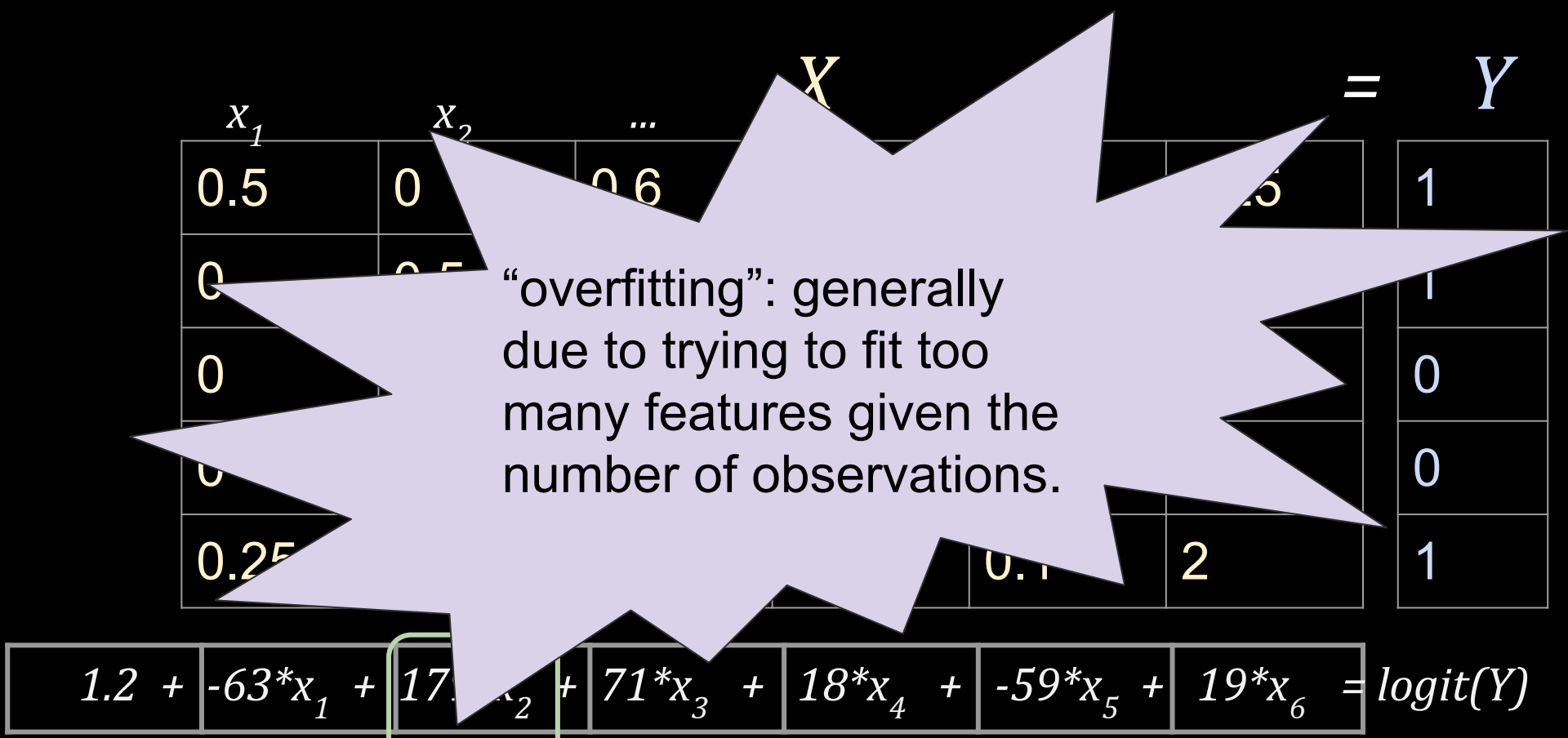
Logistic Regression - Regularization

x_1	x_2	...	X			= Y
0.5	0	0.6	1	0	0.25	1
0	0.5	0.3	0		0	1
0	0				0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

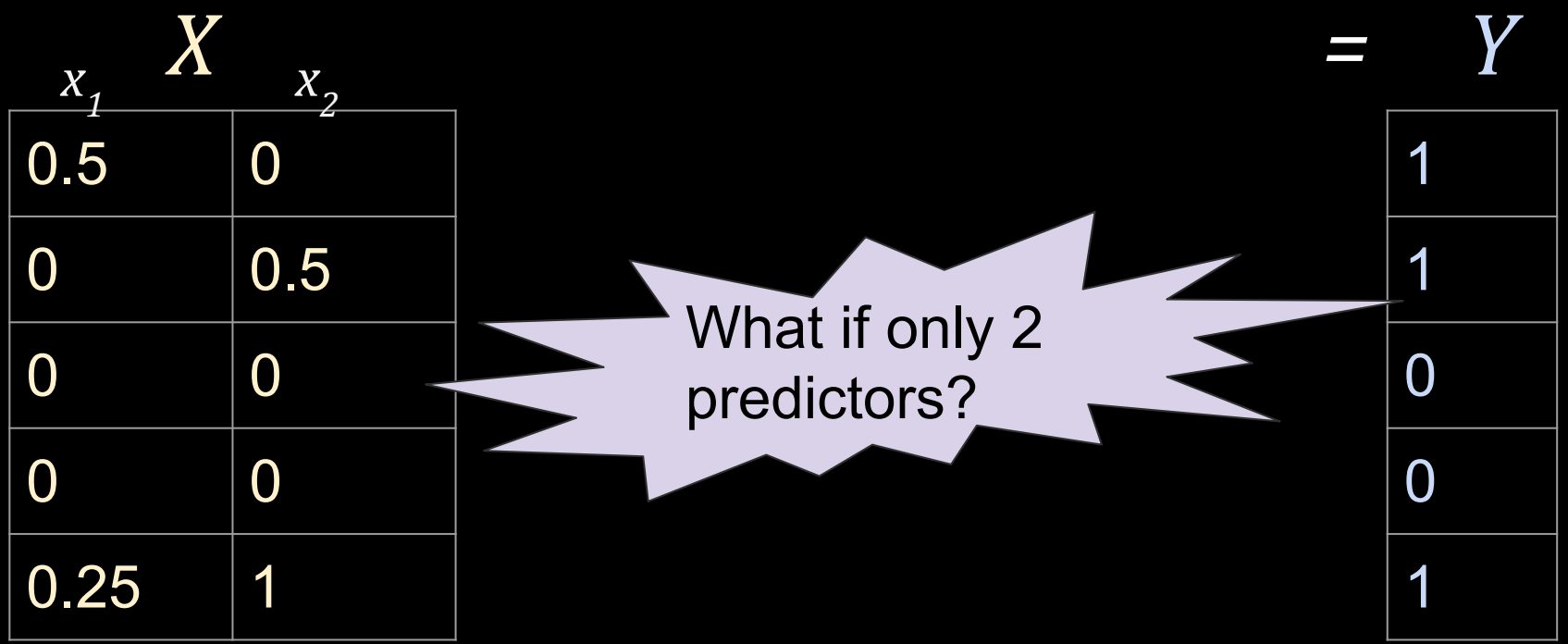
"overfitting"

$1.2 +$	$-63*x_1 +$	$179*x_2 +$	$71*x_3 +$	$18*x_4 +$	$-59*x_5 +$	$19*x_6 = logit(Y)$

Logistic Regression - Regularization



Logistic Regression - Regularization



Logistic Regression - Regularization

x_1	x_2	$= Y$
0.5	0	1
0	0.5	1
0	0	0
0	0	0
0.25	1	1

What if only 2 predictors?

A: better fit

$0 + 2*x_1 + 2*x_2$ $= logit(Y)$

Logistic Regression - Regularization

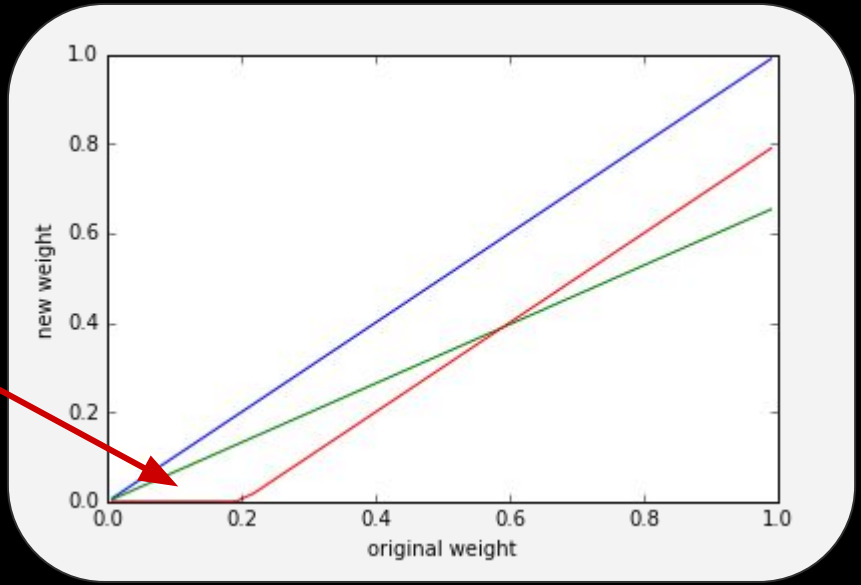
L1 Regularization - “The Lasso”

Zeros out features by adding values that keep from perfectly fitting the data.

Logistic Regression - Regularization

L1 Regularization - “The Lasso”

Zeros out features by adding values that keep from perfectly fitting the data.



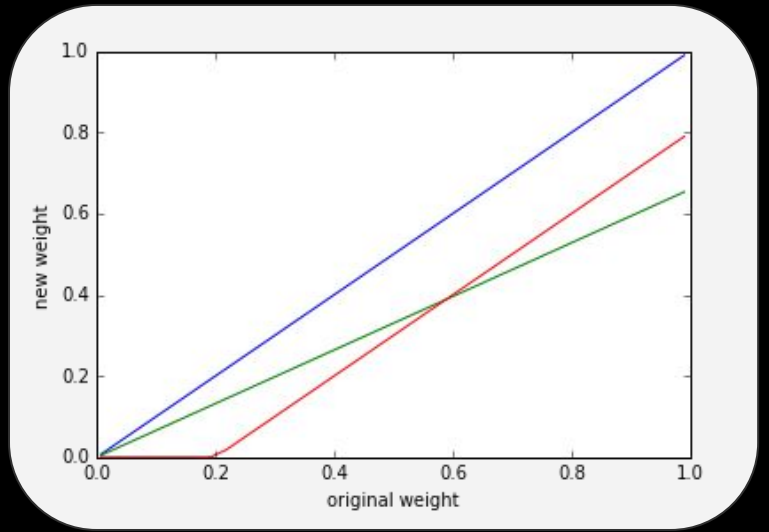
Logistic Regression - Regularization

L1 Regularization - “The Lasso”

Zeros out features by adding values that keep from perfectly fitting the data.

$$L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

set betas that maximize L



Logistic Regression - Regularization

L1 Regularization - “The Lasso”

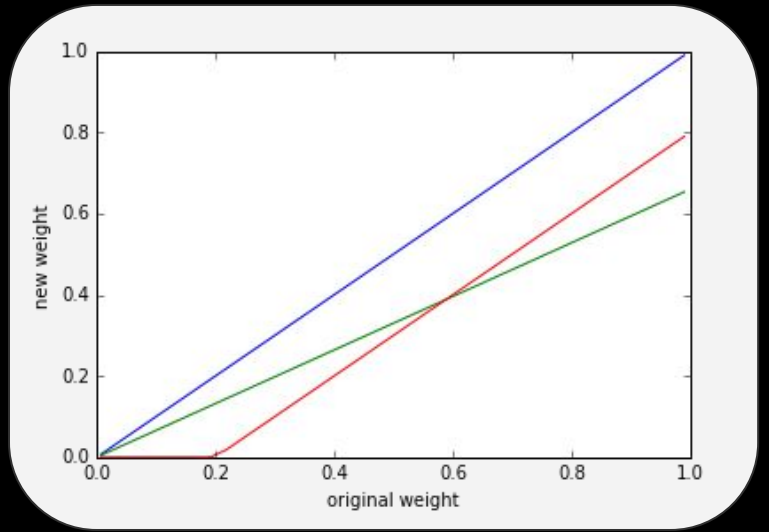
Zeros out features by adding values that keep from perfectly fitting the data.

$$L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i} - \frac{1}{C}\sum_{j=1}^{m} |\beta_j|$$

set betas that maximize *penalized* L

This is for likelihood

for log loss, would add the penalty



Logistic Regression - Regularization

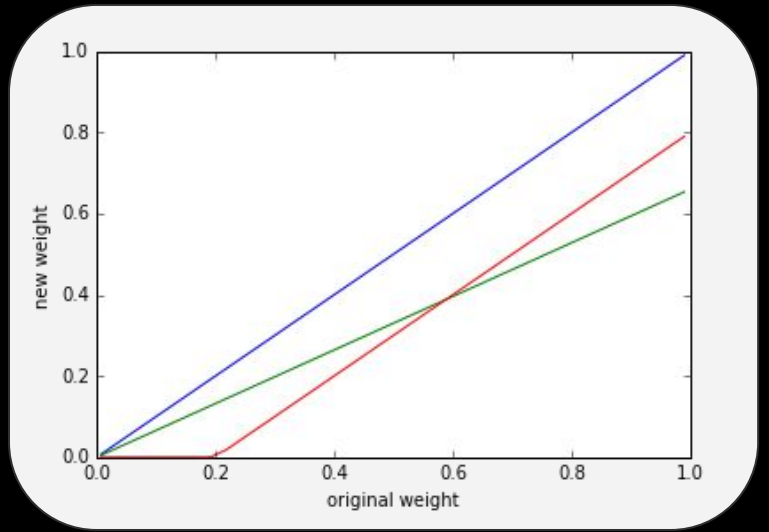
L1 Regularization - “The Lasso”

Zeros out features by adding values that keep from perfectly fitting the data.

Sometimes written as: $||\beta||_1$

$$L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i} - \frac{1}{C}\sum_{j=1}^{m} |\beta_j|$$

set betas that maximize *penalized* L



Logistic Regression - Regularization

L2 Regularization - “Ridge”

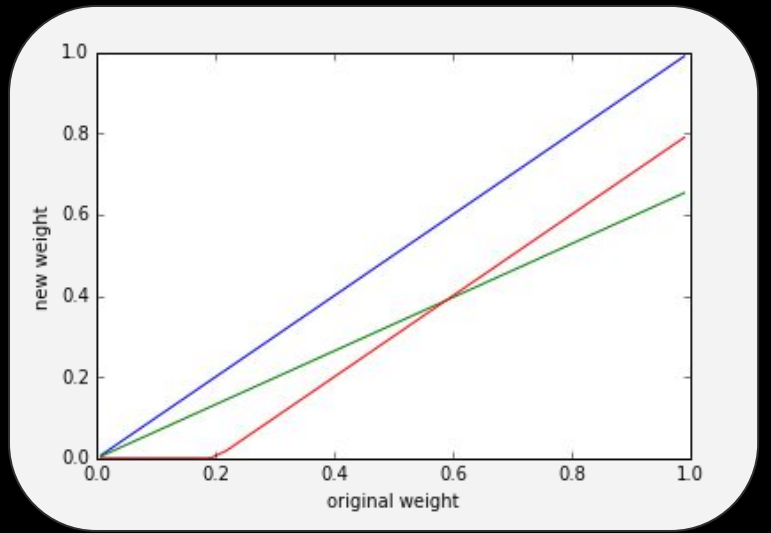
Shrinks features by adding values that keep from perfectly fitting the data.

Sometimes written as:

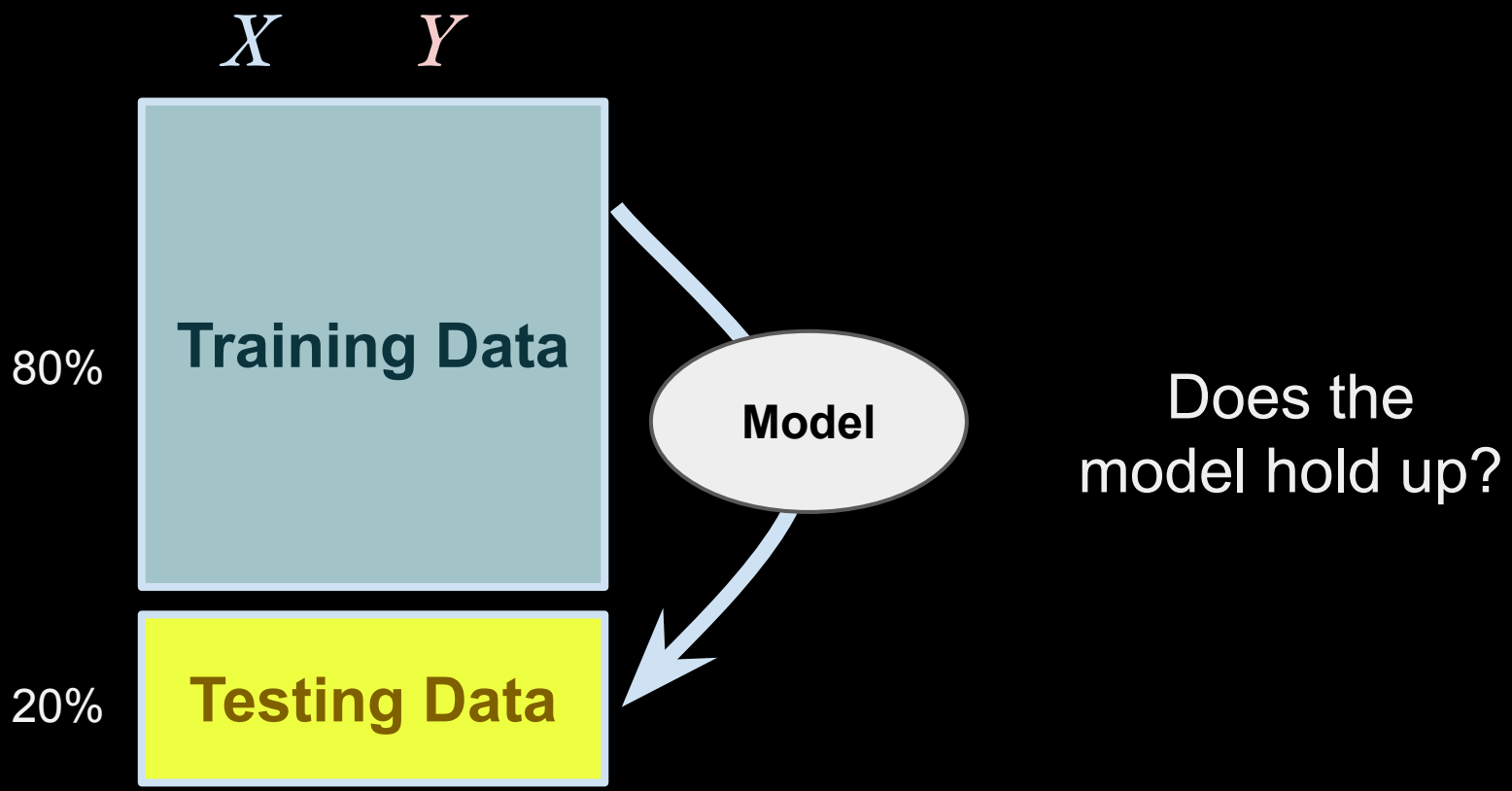
$$||\beta_j||_2^2$$

$$L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i} - \frac{1}{C}\sum_{j=1}^{m} \beta_j^2$$

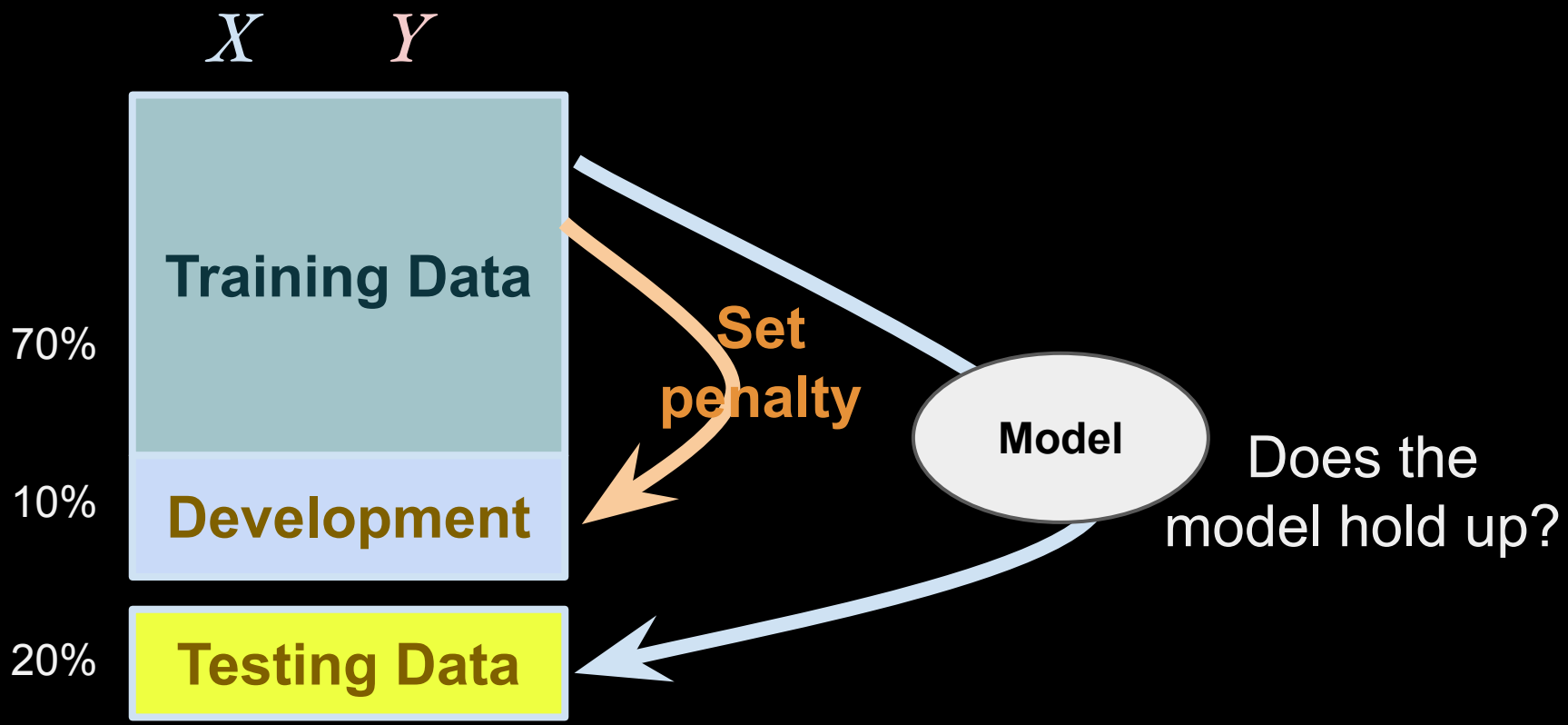
set betas that maximize *penalized* L



Machine Learning Goal: Generalize to new data



Machine Learning Goal: Generalize to new data



Logistic Regression - Review

- Probabilistic Classification: *P(Y | X)*
- Learn logistic curve based on example data
  - training + development + testing data
- Set betas based on maximizing the *likelihood*
  (or based on minimizing *log loss*)
  - “shifts” and “twists” the logistic curve
  - separation represented by hyperplane at 0.50
- Multivariate features: Multi-, One-hot encodings
- Overfitting and Regularization

From binary to multiclass, Loss:

Log Loss: $$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

Multiclass Log Loss: ???

From binary to multiclass, Loss:

Log Loss:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{y_i \log(p_i)}_{\text{class 1}} + \underbrace{(1-y_i)\log(1-p_i)}_{\text{class 0}}$$

Multiclass Log Loss:

$$J = -\frac{1}{N}\sum_{i=1}^{N}$$??

From binary to multiclass, Loss:

Log Loss:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{y_i \log(p_i)}_{\text{class 1}} + \underbrace{(1-y_i)\log(1-p_i)}_{\text{class 0}}$$

V is classes

Multiclass Log Loss:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\ p(x_{i,j})$$

From binary to multiclass, Loss:

Log Loss:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{y_i \log(p_i)}_{\text{class 1}} + \underbrace{(1-y_i)\log(1-p_i)}_{\text{class 0}}$$

V is classes

Multiclass Log Loss:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\ p(x_{i,j})$$

Equivalently:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\ (\frac{1}{1+e^{-\beta x}})$$

From binary to multiclass, Loss:

Log Loss:

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{y_i \log(p_i)}_{\text{class 1}} + \underbrace{(1-y_i)\log(1-p_i)}_{\text{class 0}}$$

V is classes

Multiclass Log Loss:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\ p(x_{i,j})$$

Equivalently:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\ (\frac{1}{1+e^{-\beta x}})$$

```
loss = torch.mean(-torch.sum(y*torch.log(y_pred))
```

From binary to multiclass, Loss:

```
    def __init__(self, num_feats, num_classes,
                 learn_rate = 0.01, device = torch.device("cpu") ):
        #the constructor; define any layer objects (e.g. Linear)
        super(MultiClassLogReg, self).__init__()
        self.linear = nn.Linear(num_feats+1, 1—num_classes)

    def forward(self, X):
        newX = torch.cat((X, torch.ones(X.shape[0], 1)), 1) #add intercept

        #logistic function on the linear output:
        y_pred = 1/(1 + torch.exp(-self.linear(newX)))
        return y_pred
```

Equivalently:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\,(\frac{1}{1+e^{-\beta x}})$$

```
loss = torch.mean(-torch.sum(y*torch.log(y_pred))
```

From binary to multiclass, Loss:

```
    def __init__(self, num_feats, num_classes,
                 learn_rate = 0.01, device = torch.device("cpu") ):
        #the constructor; define any layer objects (e.g. Linear)
        super(MultiClassLogReg, self).__init__()
        self.linear = nn.Linear(num_feats+1, 1—num_classes)

    def forward(self, X):
        newX = torch.cat((X, torch.ones(X.shape[0], 1)), 1) #add intercept

        #logistic function on the linear output:
        y_pred = 1/(1 + torch.exp(-self.linear(newX)))
        return y_pred
```

Equivalently:

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\,(\frac{1}{1+e^{-\beta x}})$$

Add i

not in prebuilt torch function

```
loss = torch.mean(-torch.sum(y*torch.log(y_pred))
```

From binary to multiclass, Loss:

```
    def __init__(self, num_feats, num_classes,
                 learn_rate = 0.01, device = torch.device("cpu") ):
        #the constructor; define any layer objects (e.g. Linear)
        super(MultiClassLogReg, self).__init__()
        self.linear = nn.Linear(num_feats+1, 1num_classes)

    def forward(self, X):
        newX = torch.cat((X, torch.ones(X.shape[0], 1)), 1) #add intercept

        #logistic function on the linear output:
        #y_pred = 1/(1 + torch.exp(-self.linear(newX)))
        y_pred = torch.log(1/(1 + torch.exp(-self.linear(newX))))
                 #or simply: nn.log_softmax(self.linear(newX))
        return y_pred
...
loss_func = nn.NLLLoss()#expects log probabilities of each class
```

not in prebuilt torch function

```
loss = torch.mean(-torch.sum(y*torch.log(y_pred))
```

Two equivalent options for multi-class in torch:

option 1: NLLLoss (easier to understand functions)

```
#in model/forward:
        return nn.log_softmax(self.linear(newX)) #log softmax is multiclass

#in loss/train:
        loss_func = nn.NLLLoss() #negative log likelihood loss
```

option 2: CrossEntropyLoss (easier to code, obfuscates functions)

```
#in model/forward:
        return self.linear(newX) #only use linear if using cross-entropy loss

#in loss/train:
        loss_func = nn.CrossEntropyLoss() #includes log softmax
```