Maximum Entropy Classifier

Supervised Machine Learning

CSE538 - Spring 2024

Topics we will cover

- *Supervised classification (open-vocabulary)*
  - Goal of logistic regression
  - 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 Buccaneers win it!

President Biden vetoed bill

Twitter to be acquired by Apple

She <u>will</u> drive to the office, to make sure the lawyer gives the <u>will</u> to the family.

<u>will.n</u> or *<u>will.v</u>* ?

noun or 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.

$f(X) = Y$

$$
\begin{array}{ccc}
i & X & Y \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} &
\begin{pmatrix} 0.0 \\ 0.5 \\ 1.0 \\ 0.25 \\ 0.75 \end{pmatrix} &
\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}
\end{array}
$$

Supervised Classifica

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

X - features of N observations (

Y - class of each of N obs vatio

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

$$f(X) = Y$$

$$\begin{array}{ccc}
i & X & Y \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0.0 \\ 0.5 \\ 1.0 \\ 0.25 \\ 0.75 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}
\end{array}$$

Supervised Classification

Supervised Machine Learning: Build a model with examples of outcomes (i.e. Y) that one is trying to predict. (The alternative, *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 **Y** for the variable being predicted and **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.

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.

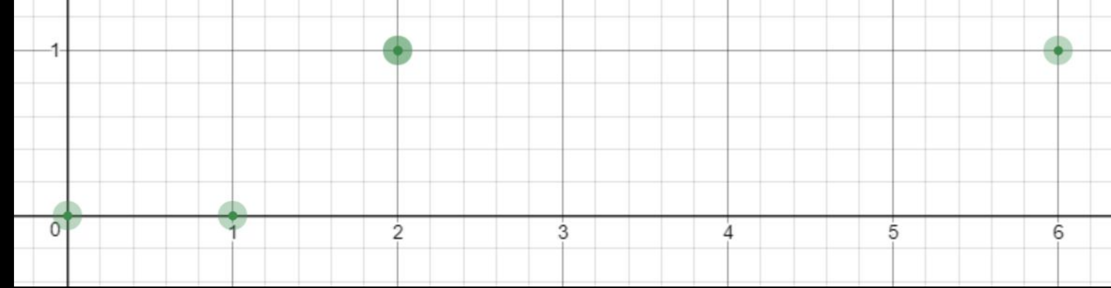
The trail was very stony.

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.

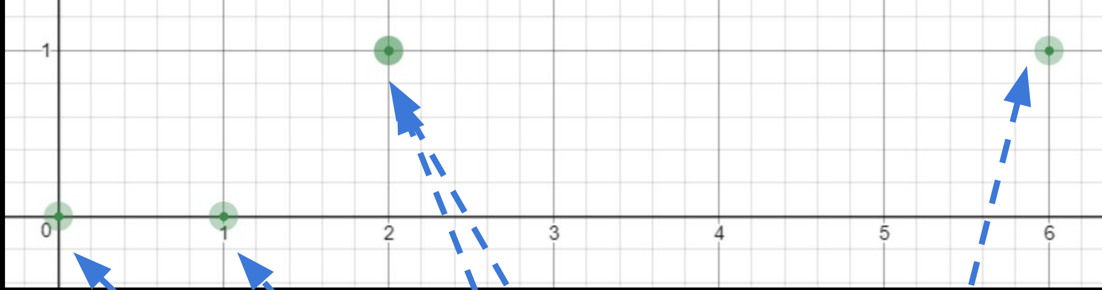
They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

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.

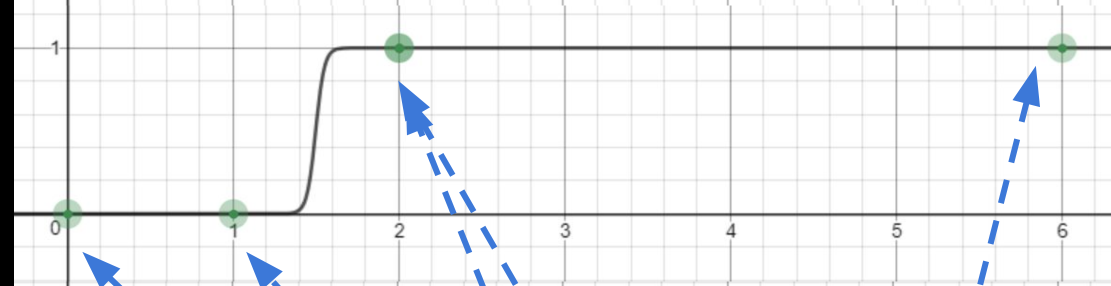
They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

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.

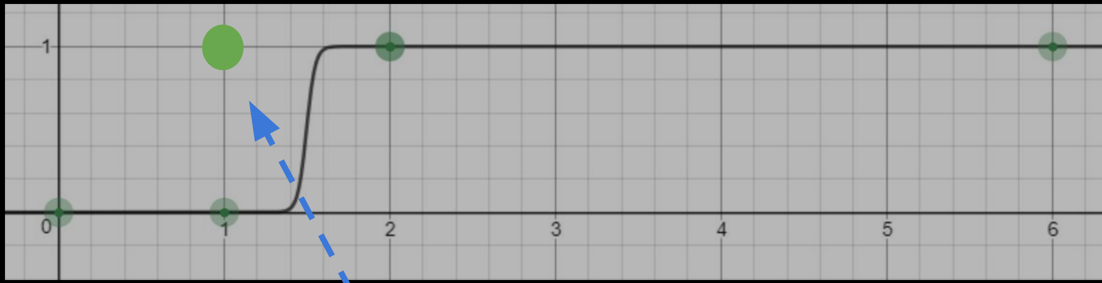
They attend Stony Brook University. *Next to the brook Gandalf lay thinking.*

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.*

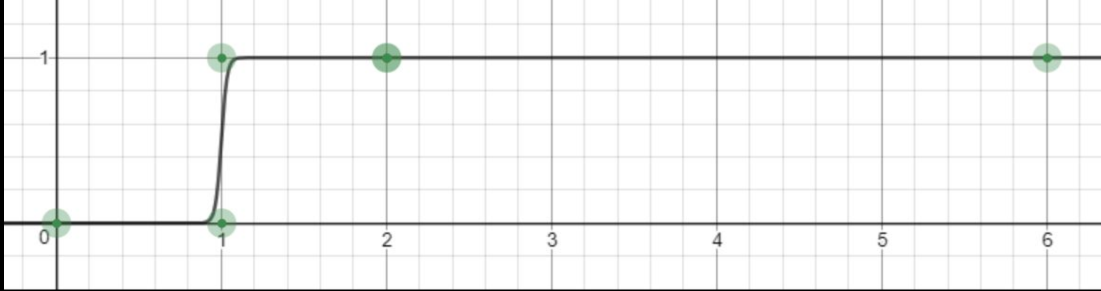
The trail was very stony. *Her degree is from SUNY Stony Brook.*

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.*

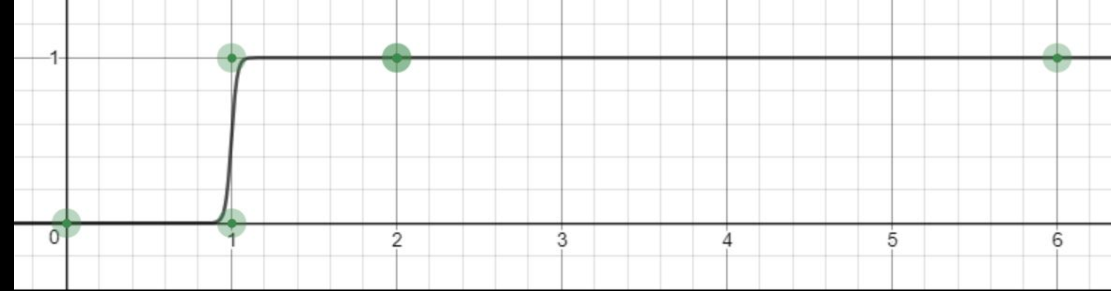
The trail was very stony. *Her degree is from SUNY Stony Brook.*

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

They attend Binghamton.

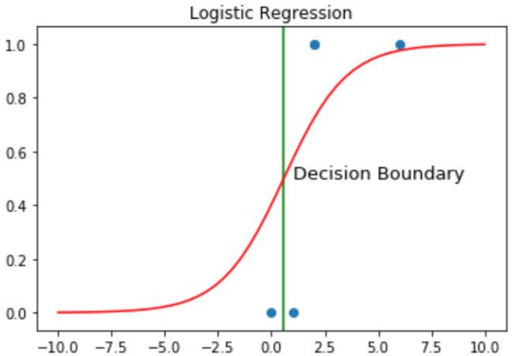
x	y
2	1
1	0
0	0
6	1
2	1
1	1

Logistic Regression



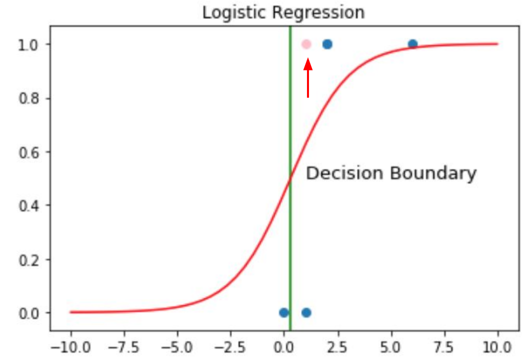
Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

```
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])
```

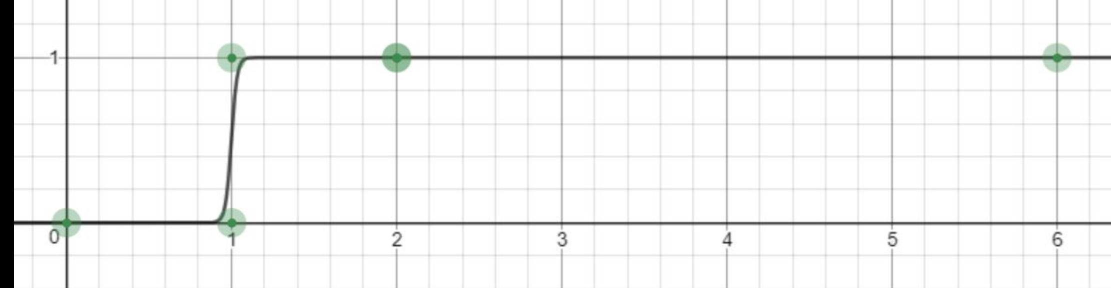
```
Out[80]: [<matplotlib.lines.Line2D at 0x11a60f160>]
```




```
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])
```

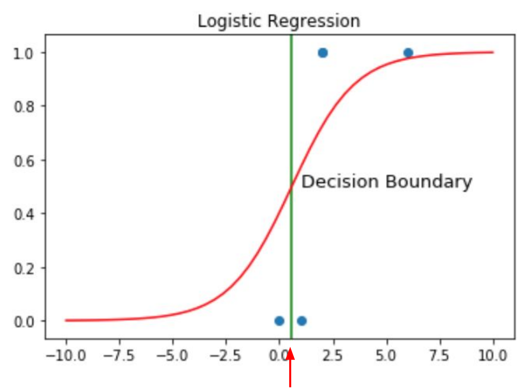
1	1

Logistic Regression



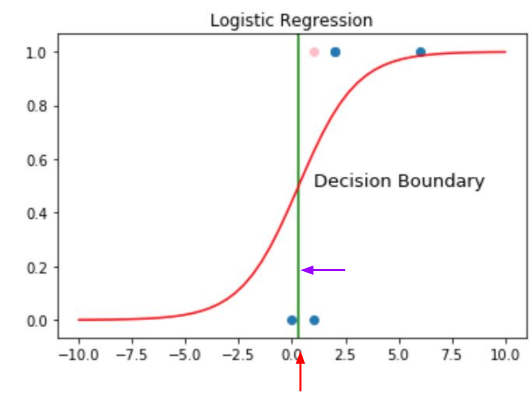
Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

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




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

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

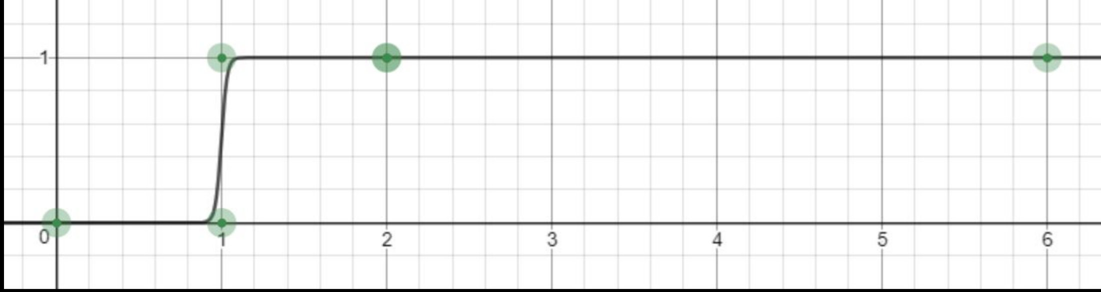
```
Out[80]: [<matplotlib.lines.Line2D at 0x11a60f160>]
```




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

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

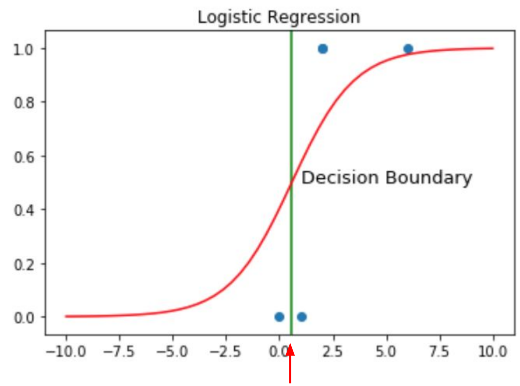
1	1

Logistic Regression



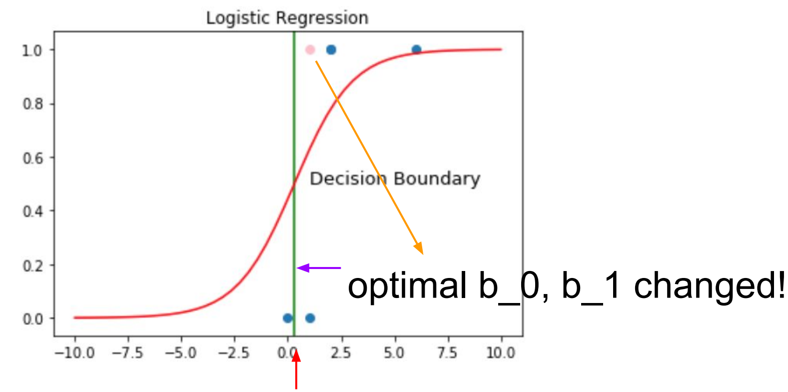
Example: Y: 1 if target is a part of a proper noun, 0 otherwise;
X: number of capital letters in target and surrounding words.

```
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])
```

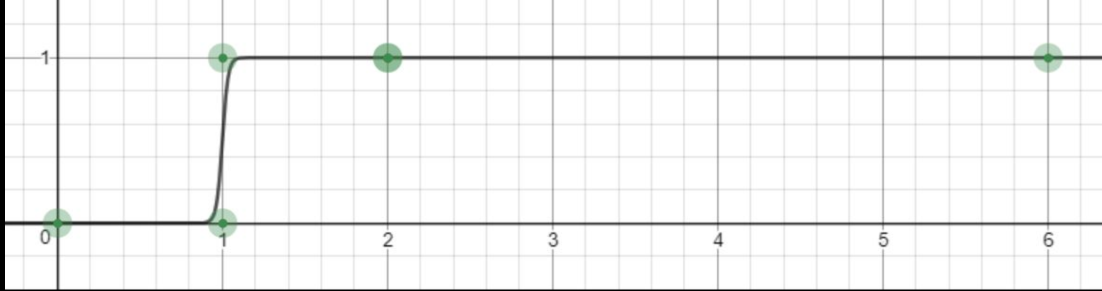
```
Out[80]: [<matplotlib.lines.Line2D at 0x11a60f160>]
```




```
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.*

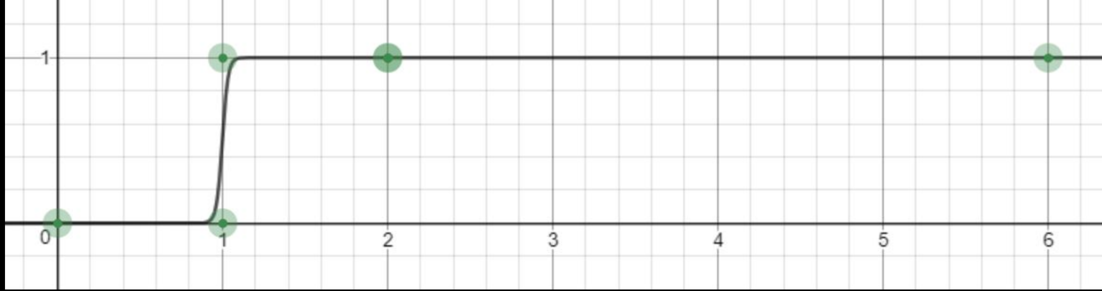
The trail was very stony. *Her degree is from SUNY Stony Brook.*

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.

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

Logistic Regression on features (*x*)

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

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

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

Logistic Regression on features (*x*)

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

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

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

<u>Vector notation</u>

β and x_i are vectors of size m

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

Logistic Regression on features (*x*)

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

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

The goal of this function is to: 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.

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

The goal of this function is to: 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.

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

The goal of this function is to: 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

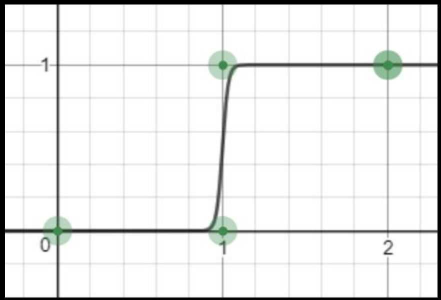
x_i is given. β must be learned.

Logistic Regression on features (*x*)

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

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

HOW? Essentially, try different B_0 and B_1 values until "best fit" to the training data (example X and Y).



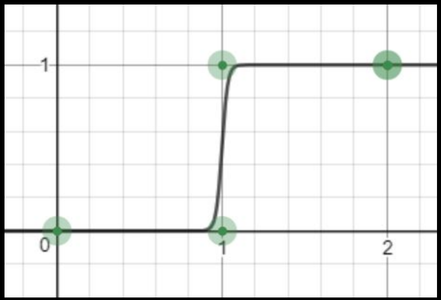
x_i is given. β must be <u>learned</u>.

“best fit” : whatever maximizes the likelihood function:

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

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

HOW? Essentially, try different B_0 and B_1 values until “best fit” to the training data (example X and Y).



x_i is given. β must be <u>learned</u>.

"best fit" : whatever maximizes the likelihood function:

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

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

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

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

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

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

"best fit" : more efficient to maximize *log likelihood* :

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

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

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

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

“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:

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

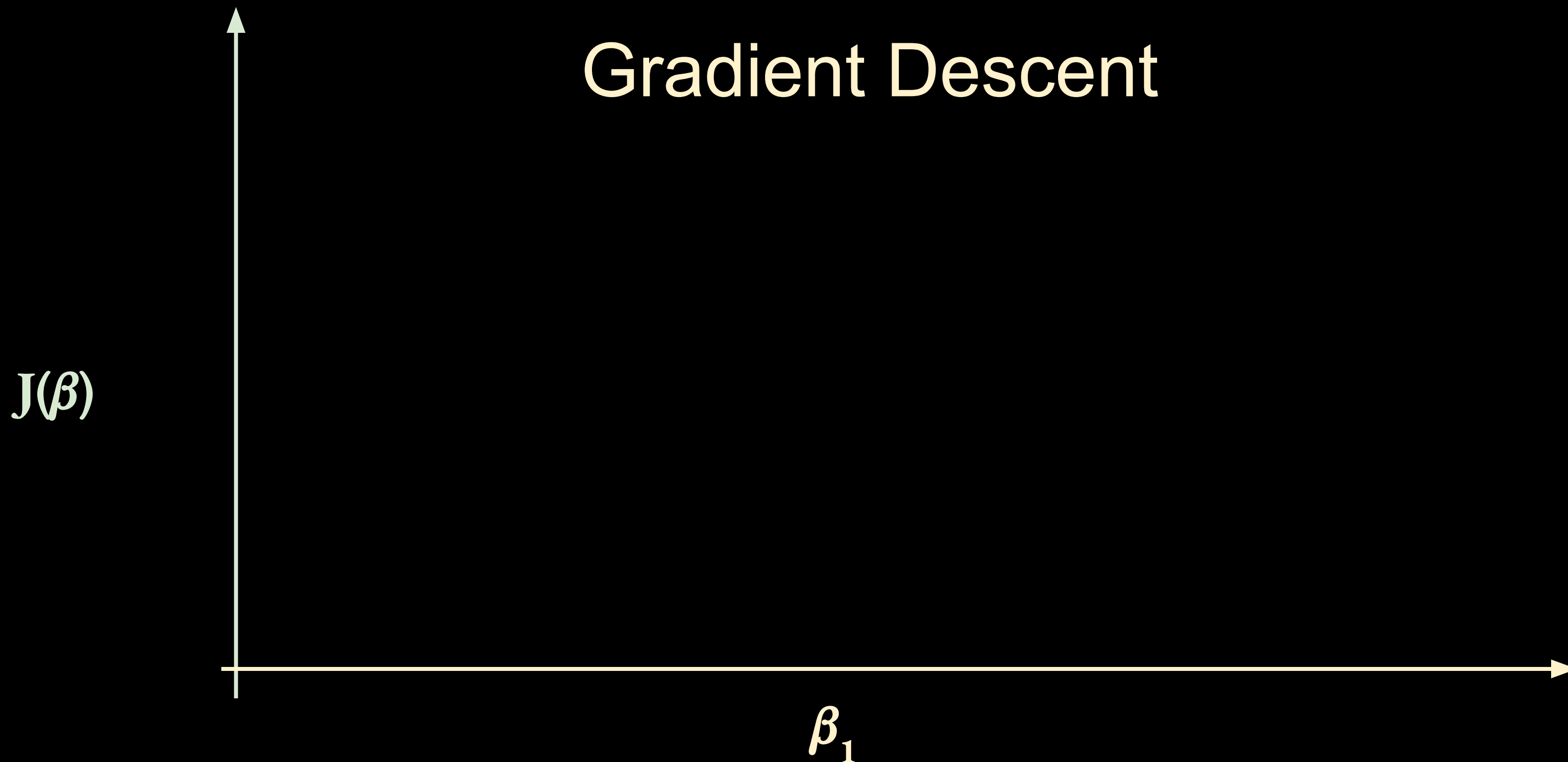
“best fit” : more efficient to maximize *log likelihood* :

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

“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":*

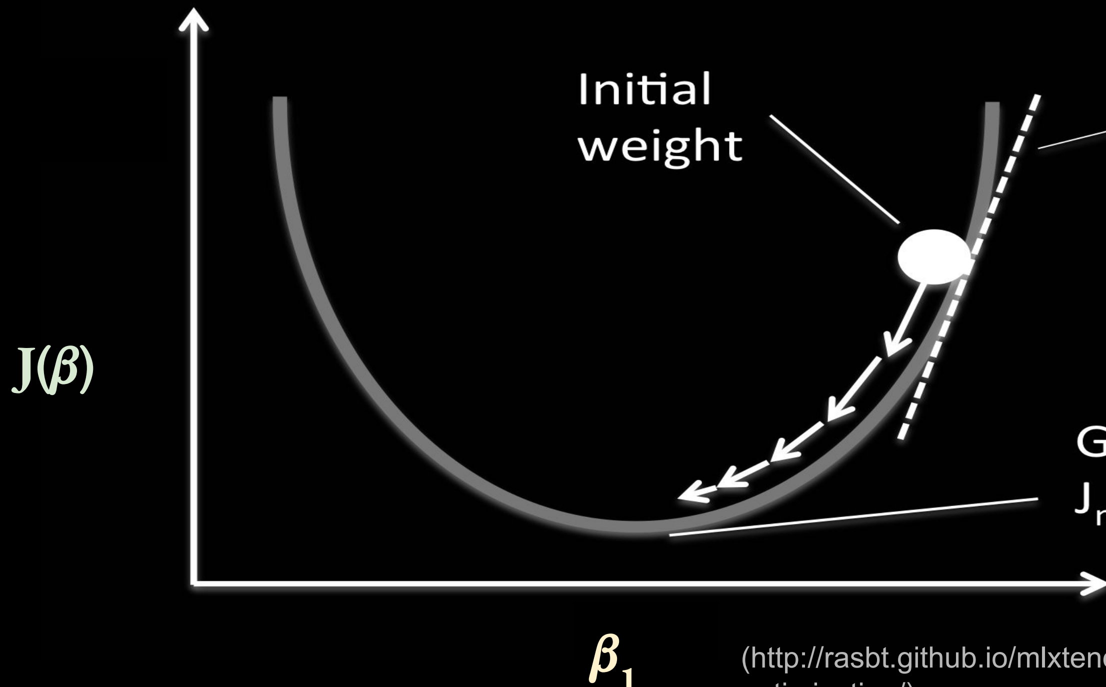
$$\boxed{J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)}$$

Gradient Descent



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

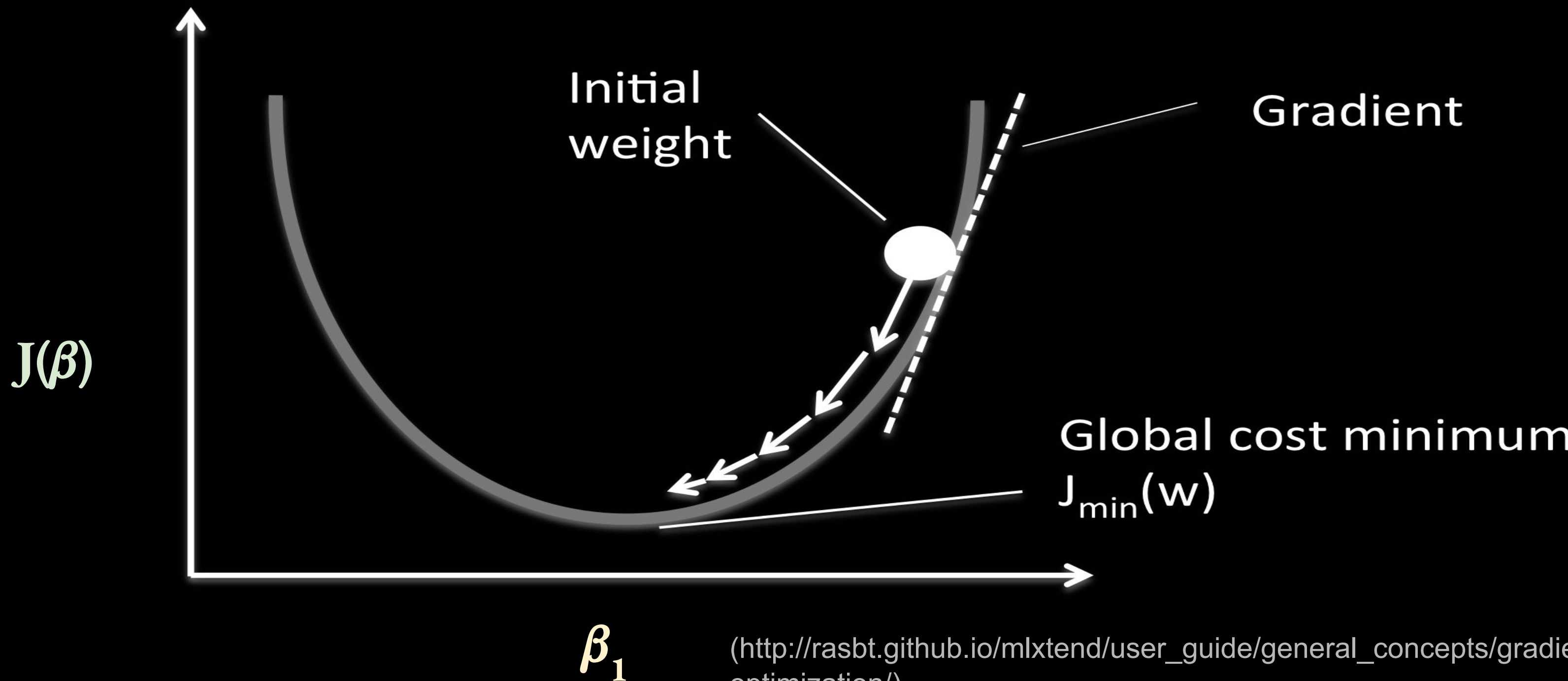
$$\boxed{J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1-y_i)\log(1-p_i)}$$



(http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/)

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

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

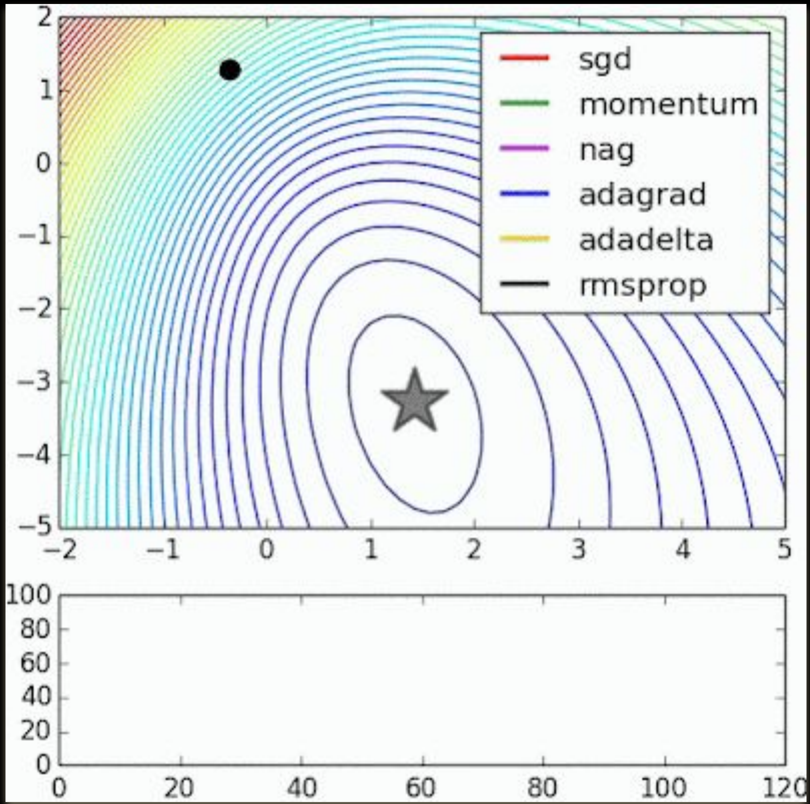


(http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/)

Update Step:

α: Learning Rate

$$\boxed{\beta_{new} = \beta_{prev} - \alpha * \text{grad}}$$



(Animation: Alec Radford, 2018)

Update Step:

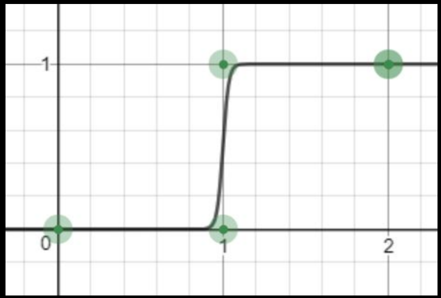
α: Learning Rate

$$\boxed{\beta_{new} = \beta_{prev} - \alpha * \text{grad}}$$

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}



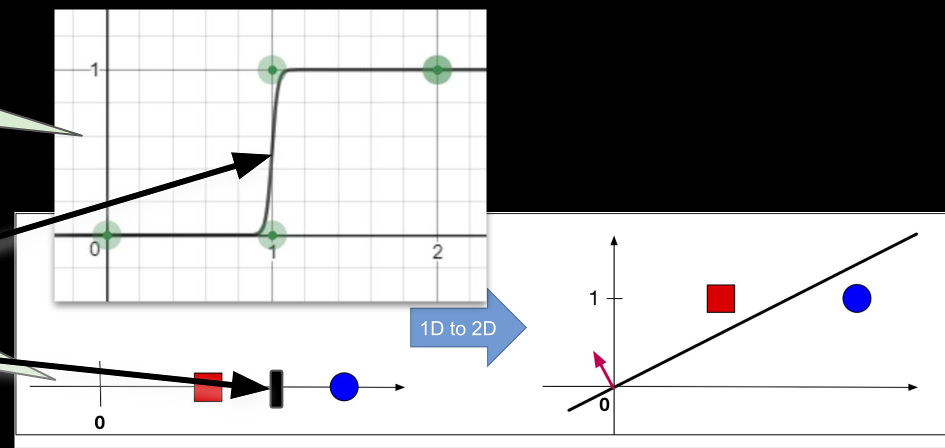
X can be multiple features

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

- Numb
  surroundin
- Begins with {0, 1}
- Prec

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

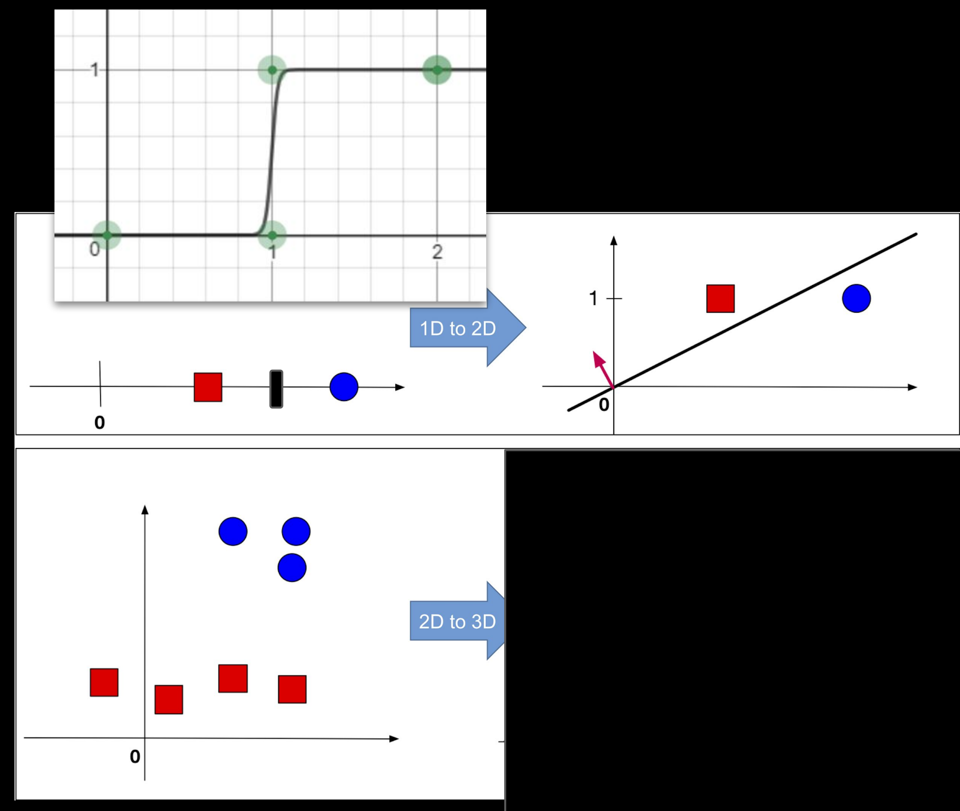
To make room for multiple Xs, let's get rid of y-axis. Instead, show **decision point**.



X can be multiple features

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

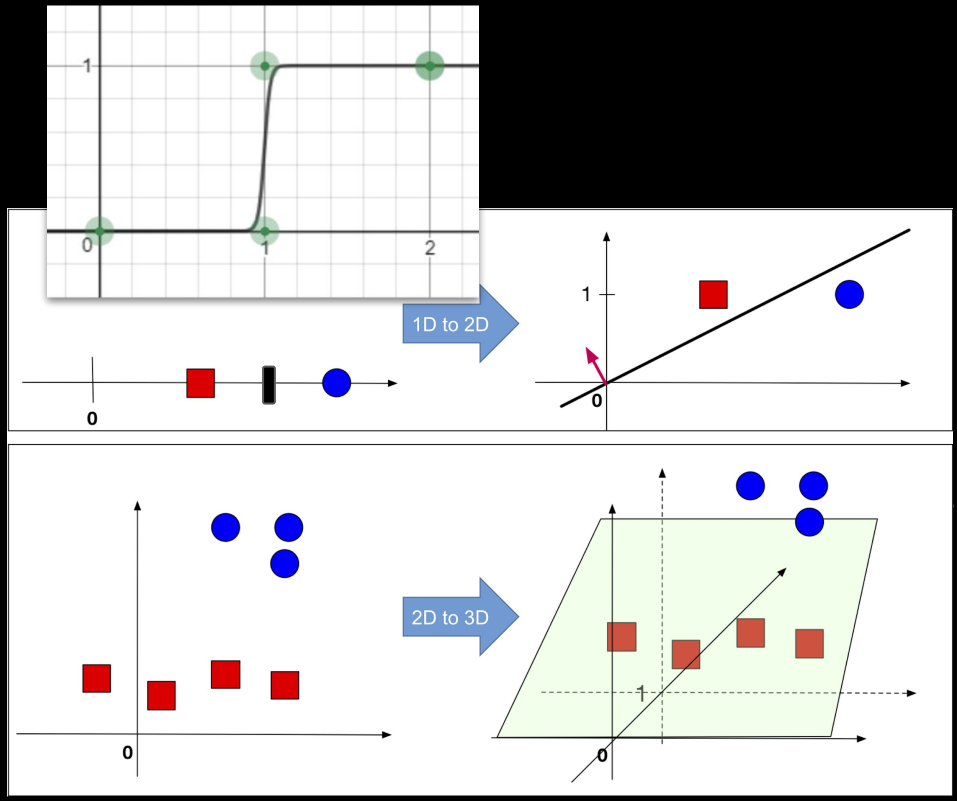
- Number of capital letters surrounding: integer
- Begins with capital letter: {0, 1}
- Preceded by “the”? {0, 1}



X can be multiple features

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

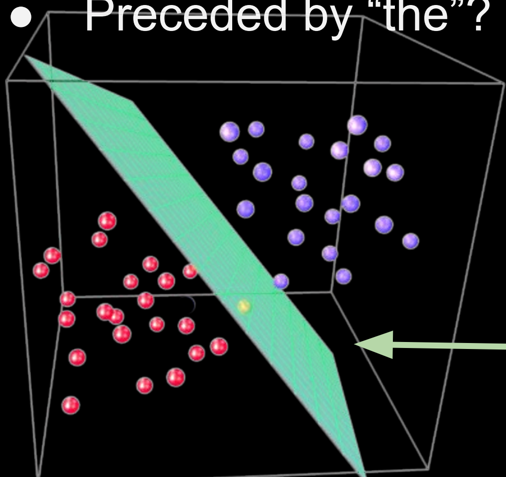
- Number of capital letters surrounding: integer
- Begins with capital letter: {0, 1}
- Preceded by “the”? {0, 1}



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}



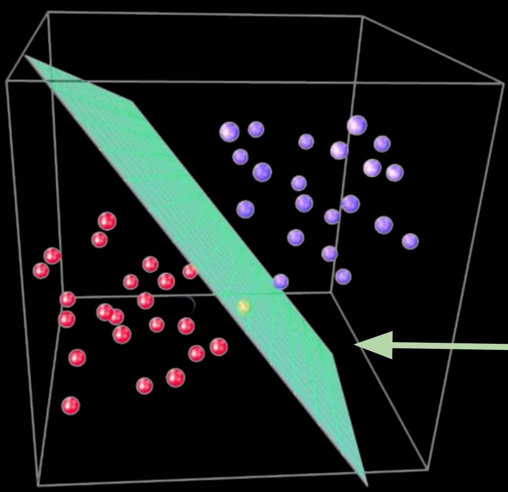
We’re learning a linear (i.e. flat) *separating hyperplane*, but fitting it to a *logit* outcome.

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

Logistic Regression

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

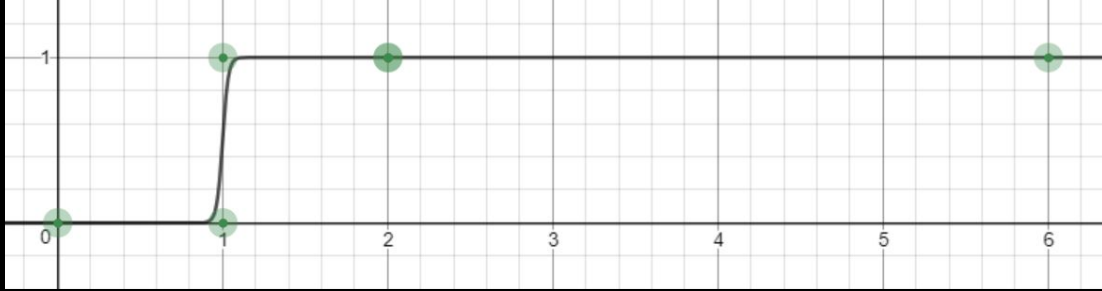
$$logit(p_i) = log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \sum_{j=1}^{m} \boxed{\beta_j x_{ij}} = 0$$



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

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

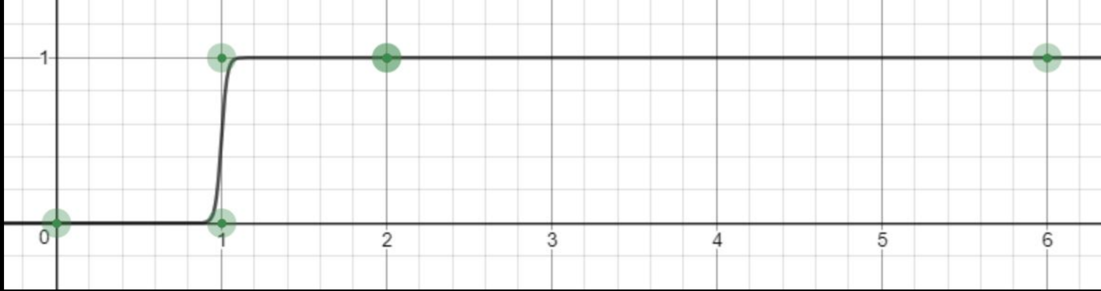
The trail was very stony. *Her degree is from SUNY Stony Brook.*

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

Terminology

β ≈ weight ≈ coefficient ≈ parameters ≈ θ

Logistic Regression ≈ Maximum Entropy Classifier

loss function ≈ cost function

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 Intro: Logistic Regression

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

   `nn.module` object maps *X* to *y_pred*

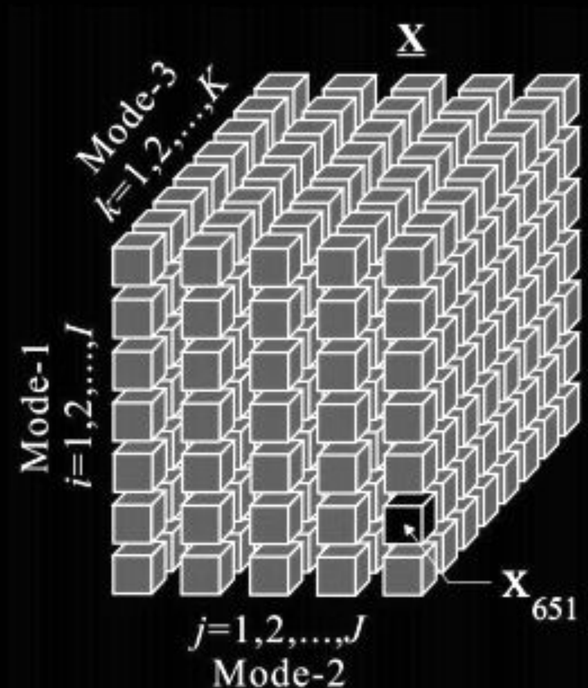
3. Loss function (training loop)

   `loop` that evaluates *ypred* versus *y*

4. Autograd (backward pass)

   `torch computation` that updates the parameters

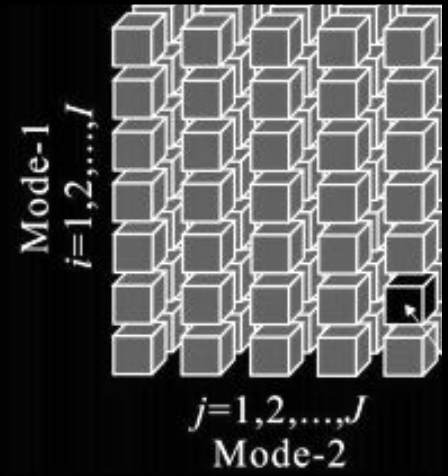
PyTorch: 1. Tenors



A multi-dimensional matrix

(i.stack.imgur.com)

PyTorch: 1. Tenors



(i.stack.imgur.com)

- A multi-dimensional matrix

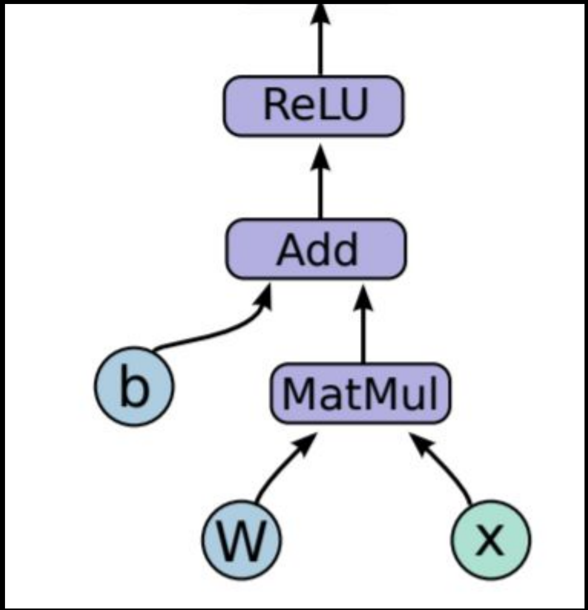
A 2-d tensor is just a matrix.
1-d: vector
0-d: a constant / scalar

Note: Linguistic ambiguity:
Dimensions of a Tensor =/=
Dimensions of a Matrix

PyTorch: 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):

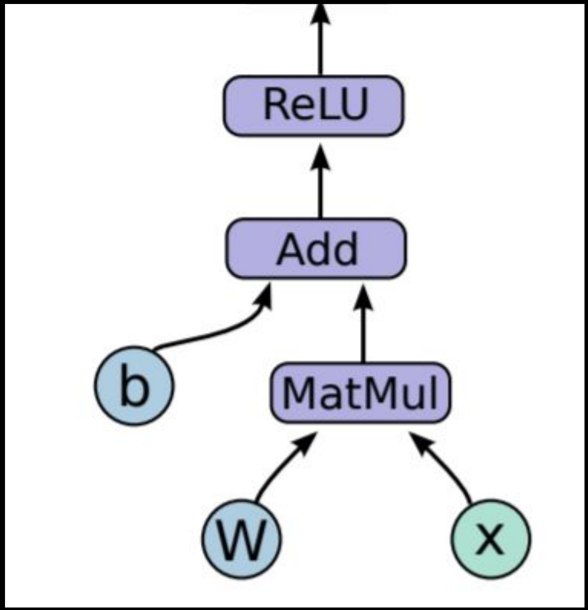


PyTorch: 2. Numeric functions as a graph/network (forward pass)

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

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

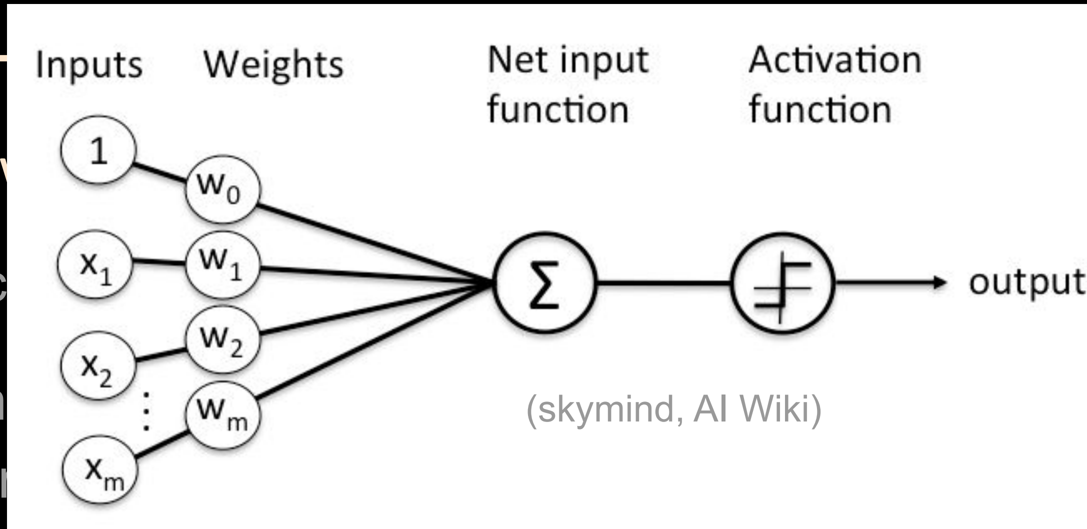
```
import torch
from torch import nn #predefined nodes

x = torch.Tensor(input)
w= torch.random.randn(X.shape, 1) #weights
z = torch.matmul(x, beta)
yhat = nn.functional.relu(z)
loss = nn.MSELoss(yhat, torch.Tensor(y))
```



Py… network
(for…

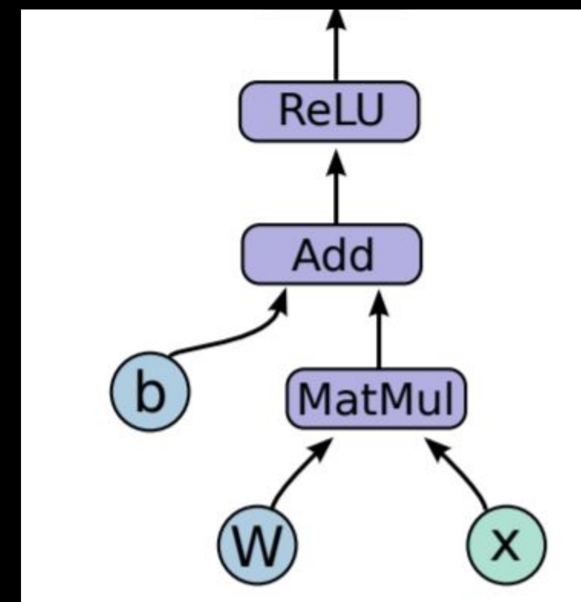
Effic… al network operations.

Can … oper…



(skymind, AI Wiki)

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

x = torch.Tensor(input)
w= torch.random.randn(X.shape, 1) #weights
z = torch.matmul(x, beta)
yhat = nn.functional.relu(z)
loss = nn.MSELoss(yhat, torch.Tensor(y))
```



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

        return 1/(1 + torch.exp(-self.linear(newX)))
                                 #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, num_classes,
                 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, num_classes)

    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

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

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.mean(-torch.sum(y*torch.log(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.mean(-torch.sum(y*torch.log(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 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: 3. Loss Function (training loop)

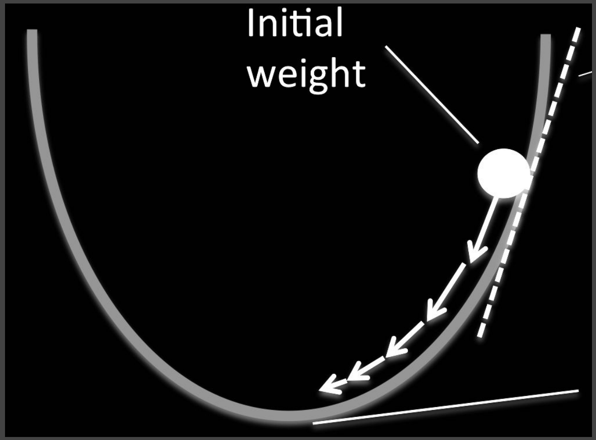
```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()
            #torch.mean(-torch.sum(y*torch.log(y_pred))
#training loop:
for i in range(epochs):
    model.train()
    sgd.zero_grad()
    #forward pass:
    ypred = 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: 4. Autograd (backward pass)

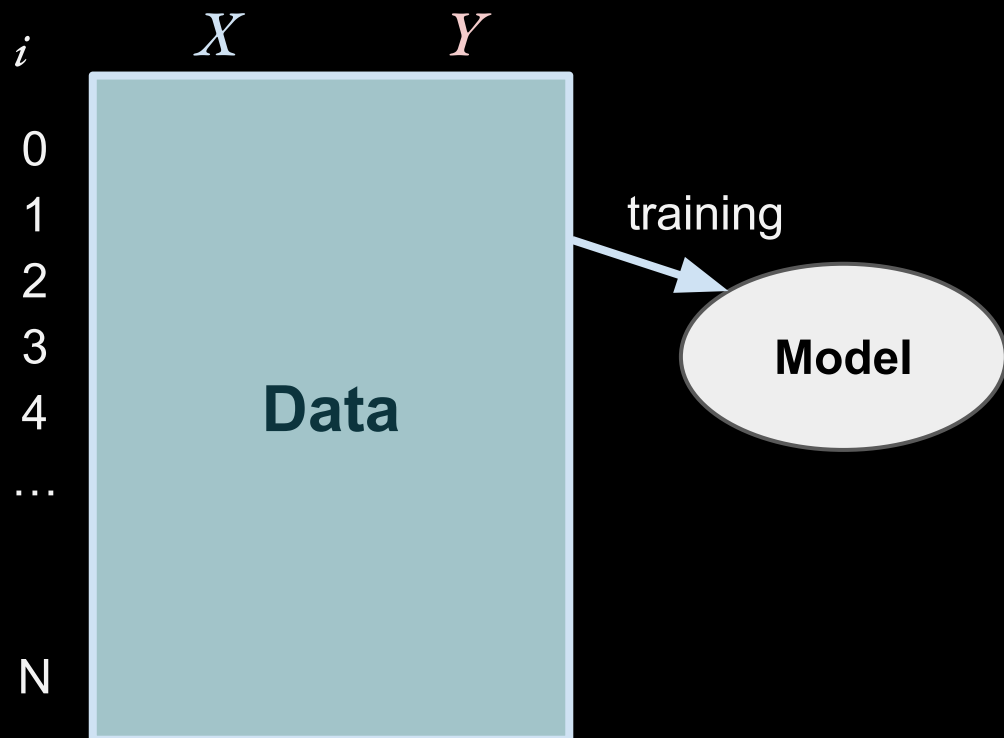
```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()

#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: 4. Autograd (backward pass)

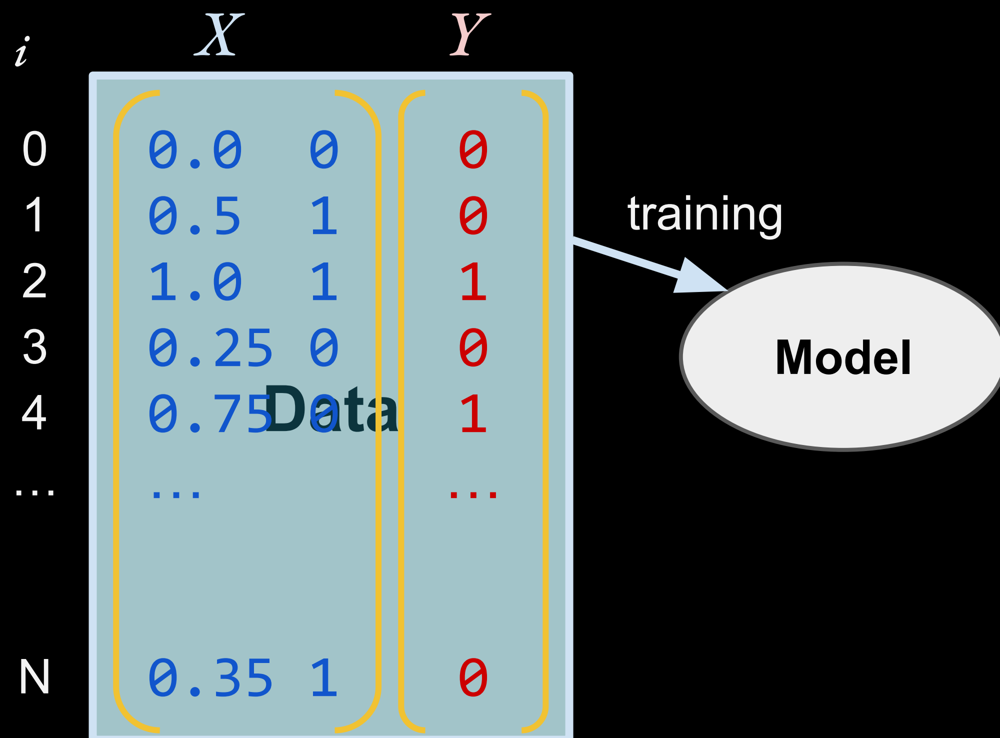
```
#runs the training loop of pytorch model:
sgd = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = torch.nn.BCELoss()

#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()))
```



Machine Learning: How to setup data



Machine Learning: How to setup data



Machine Learning: How to setup data

“Corpus”

raw data: sequences of characters

i	X		Y
0	0.0	0	0
1	0.5	1	0
2	1.0	1	1
3	0.25	0	0
4	0.75	0	1
…	…		…
N	0.35	1	0

Data

training

Machine Learning: How to setup data

Feature Extraction

"Corpus"

raw data:
sequences of
characters

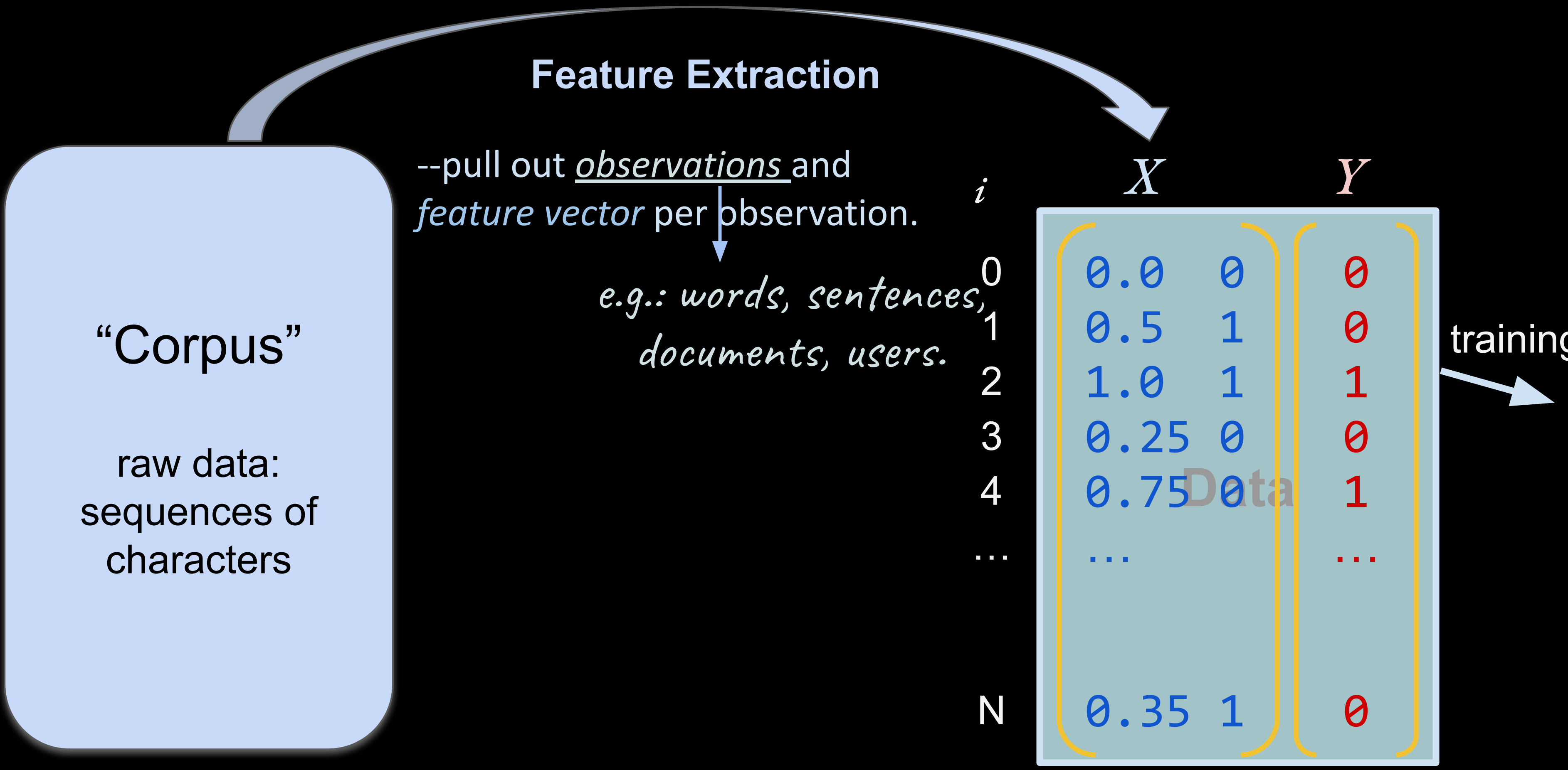
--pull out *observations* and *feature vector* per observation.

i	X		Y
0	0.0	0	0
1	0.5	1	0
2	1.0	1	1
3	0.25	0	0
4	0.75	0	1
…	…		…
N	0.35	1	0

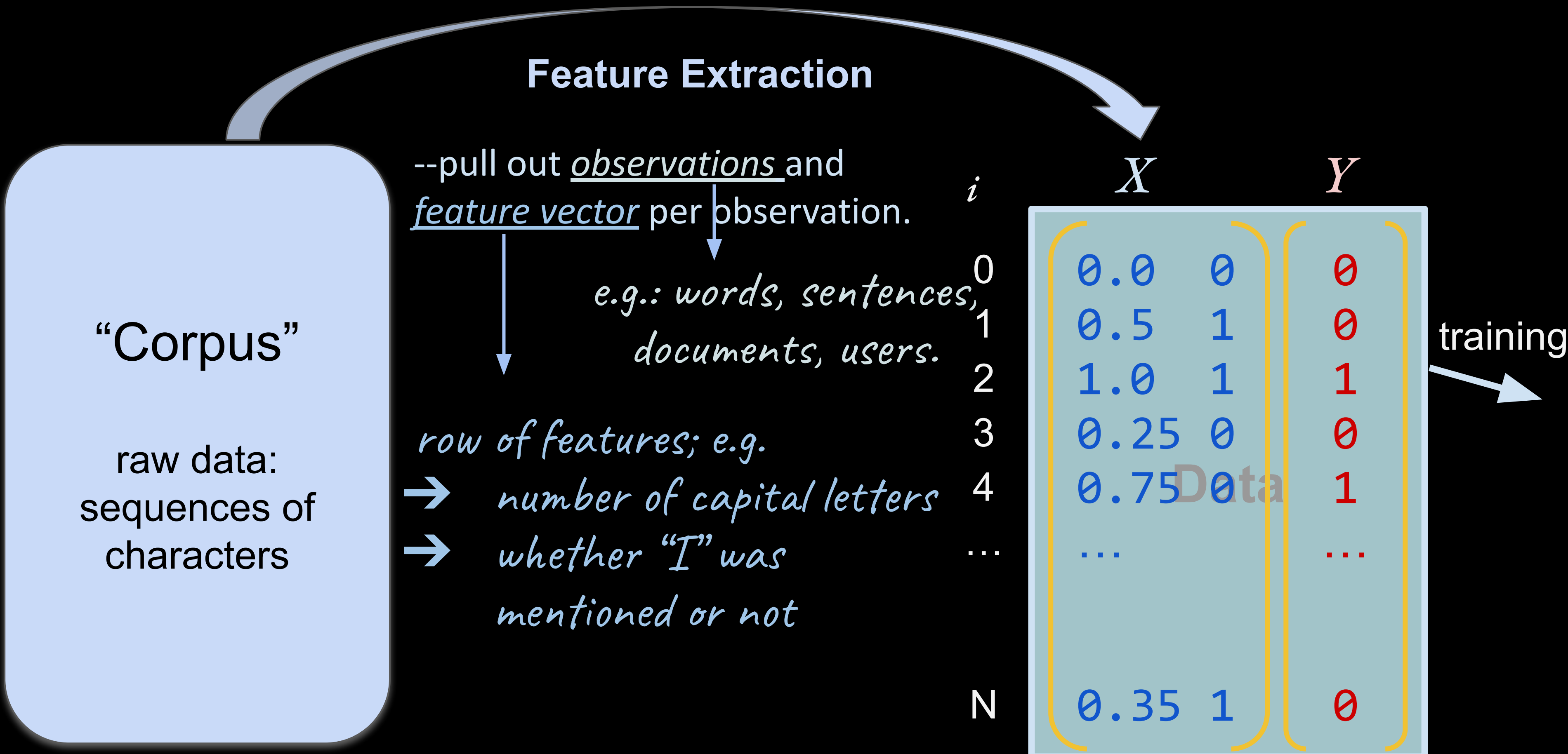
Data

training

Machine Learning: How to setup data



Machine Learning: How to setup data



Machine Learning: How to setup data

"Corpus"

raw data: sequences of characters

Feature Extraction

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

e.g.: words, sentences, documents, users.

row of features; e.g.

- *number of capital letters*
- *whether "I" was mentioned or not*
- *k features indicating whether k words were mentioned or not*

Data

i	X		Y
0	0.0	0	0
1	0.5	1	0
2	1.0	1	1
3	0.25	0	0
4	0.75	0	1
…	…		…
N	0.35	1	0

training

Machine Learning: How to setup data

Feature Extraction

X Y

Multi-hot Encoding

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

“Corpus”

raw data: sequences of characters

of features; e.g.

→ *number of capital letters*

→ *whether “I” was mentioned or not*

→ ***k features indicating whether k words were mentioned or not***

Data

Machine Learning: How to setup data

Feature Extraction

X Y

Multi-hot Encoding

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

Feature example: is word present in document?

The book was interesting so I was happy .

"Corpus"

raw data: sequences of characters

of features; e.g.

→ number of capital letters

→ whether "I" was mentioned or not

→ *k* features indicating whether *k* words were mentioned or not

Data

Machine Learning: How to setup data

Feature Extraction

X Y

Multi-hot Encoding

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

“Corpus”

Feature example: is word present in document?

The book was interesting so I was happy .

raw data: sequences of characters

`[0, 1, 1, 0, 1, …, 1, 0, 1, 1, 0, 1, …, 1]`^{k}

→ *k features indicating whether k words were mentioned or not*

Data

Machine Learning: How to setup data

Feature Extraction

X Y

Multi-hot Encoding

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

Feature example: is word present in document

"Corpus"

raw data: sequences of characters

Data

The book was interesting so I was happy .

[0, 1, 1, 0, 1, …, 1, 0, 1, 1, 0, 1, …, 1]^k

a

→ *k features indicating whether k words were mentioned or not*

sad

Machine Learning: How to setup data

Feature Extraction

X Y

Multi-hot Encoding

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

Feature example: is previous word "the"?

"Corpus"

raw data: sequences of characters

The ***book*** *was interesting so I was happy .*

[0, 1, 1, 0, 1, …, 1, 0, 1, 1, 0, 1, …, 1]^k

Data

→ *k features indicating whether k words were mentioned or not*

Machine Learning: How to setup data

Feature Extraction

"Corpus"

raw data: sequences of characters

X Y

Data

Multi-hot Encoding

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

Feature example: is previous word "the"?

The ***book*** *was interesting so I was happy .*

~~[0, 1, 1, 0, 1, …, 1, 0, 1, 1, 0, 1, …, 1]~~^{k}

→ k features indicating whether k words were mentioned or not

Machine Learning: How to setup data

Feature Extraction

X Y

"Corpus"

raw data: sequences of characters

Data

One-hot Encoding

- Each word gets an index in the vector
- All indices 0 except present word:

Feature example: is previous word "the"?

The ***book*** *was interesting so I was happy .*

`[0, 1, 0, 0, 0, …, 0, 0, 0, 0, 0, 0, …, 0]`^{k}

→ *k features indicating whether k words were mentioned or not*

Machine Learning: How to setup data

Feature Extraction

X Y

One-hot Encoding

- Each word gets an index in the vector
- All indices 0 except present word:

Feature example: which is previous word?

The book was interesting so I was happy .

[0, 1, 0, 0, 0, ..., 0, 0, 0, 0, 0, 0, ..., 0]^k

[0, 0, 1, 0, 0, ..., 0, 0, 0, 0, 0, 0, ..., 0]^k

"Corpus"

raw data: sequences of characters

Data

Machine Learning: How to setup data

Feature Extraction

X Y

One-hot Encoding

- Each word gets an index in the vector
- All indices 0 except present word:

Feature example: which is previous word?

“Corpus”

raw data: sequences of characters

The book was interesting so I was happy .

Data

`[0, 1, 0, 0, 0, …, 0, 0, 0, 0, 0, 0, …, 0]`^{k}

`[0, 0, 1, 0, 0, …, 0, 0, 0, 0, 0, 0, …, 0]`^{k}

Machine Learning: How to setup data

Feature Extraction

X Y

Multiple One-hot encodings for one observation

(1) word before; (2) word after

"Corpus"

*The book was **interesting** so I was happy .*

raw data: sequences of characters

`[0, 0, 0, 0, 1, 0, …, 0]`^{k} `[0, …, 0, 1, 0, …, 0]`^{k}

Data

Machine Learning: How to setup data

Feature Extraction

X Y

Multiple One-hot encodings for one observation

(1) word before; (2) word after

"Corpus"

*The book was **interesting** so I was happy .*

raw data: sequences of characters

Data

[0, 0, 0, 0, 1, 0, …, 0]^k [0, …, 0, 1, 0, …, 0]^k

=

[0, 0, 0, 0, 1, 0, …, 0, 0, …, 0, 1, 0, …, 0]^{2k}

Machine Learning: How to setup data

Feature Extraction

X Y

Multiple One-hot encodings for one observation

(1) word before; (2) word after; (3) percent capitals

*The book was **Interesting** so I was happy .*

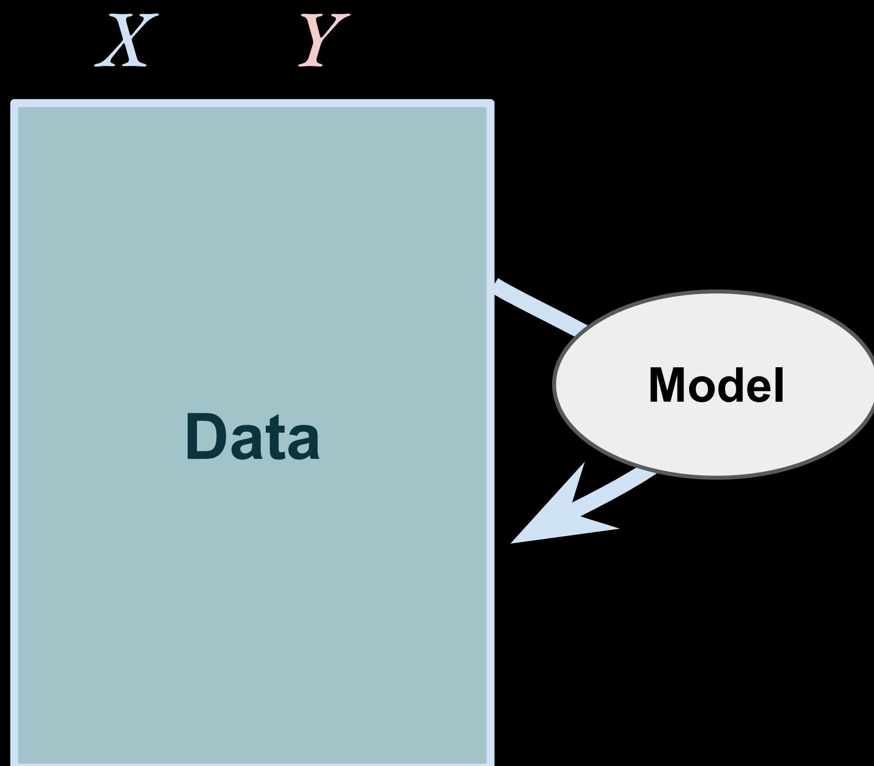
[0, 0, 0, 0, 1, 0, …, 0]^k [0, …, 0, 1, 0, …, 0]^k

=

[0, 0, 0, 0, 1, 0, …, 0, 0, …, 0, 1, 0, …, 0]^{2k}

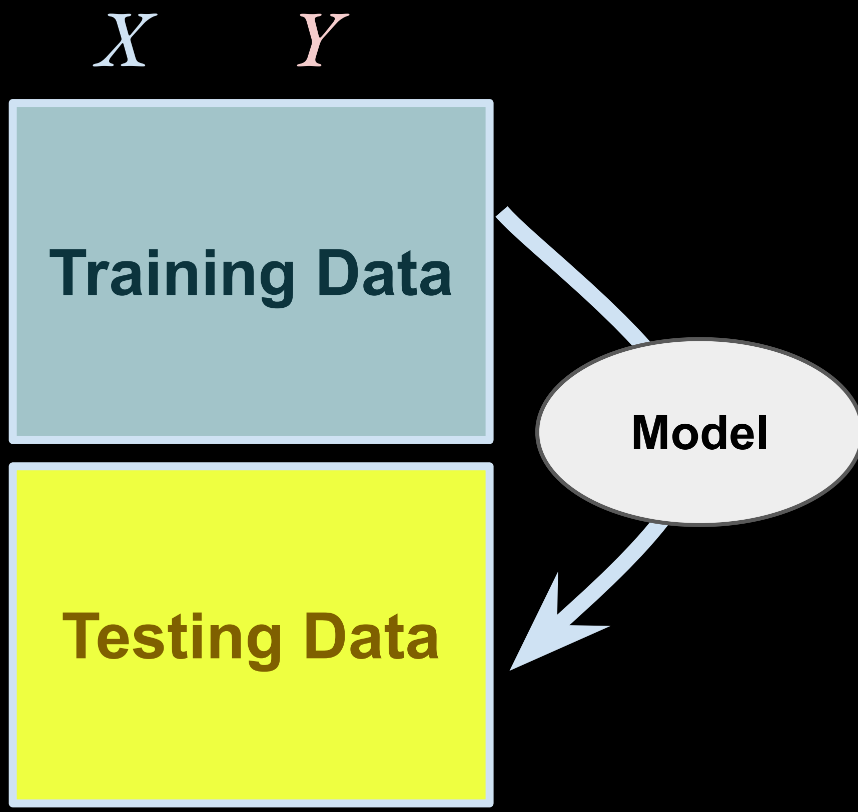
[0, 0, 0, 0, 1, 0, …, 0, 0, …, 0, 1, 0, …, 0, 0.09]$^{2k+1}$

Machine Learning: How to setup data



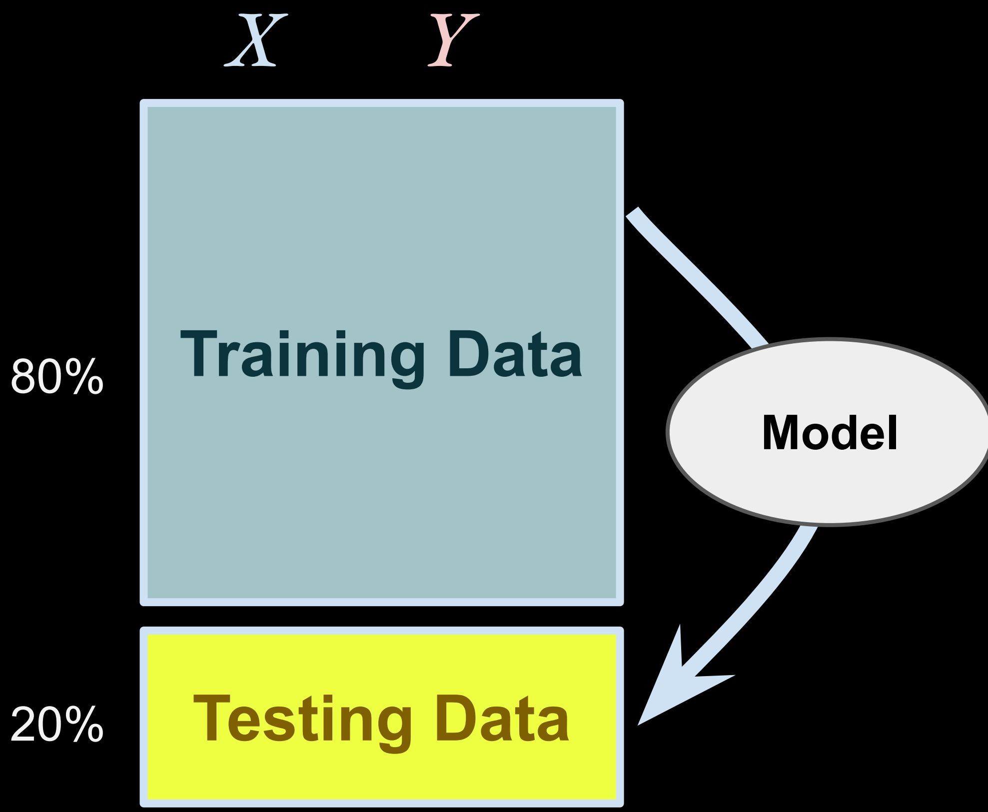
Does the model hold up?

Machine Learning Goal: Generalize to new data



Does the
model hold up?

Machine Learning Goal: Generalize to new data



Does the model hold up?

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

$X \quad = \quad Y$

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

$$X = Y$$

x_1	x_2	...				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

$$X = Y$$

x_1	x_2	…				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 $=$ Y

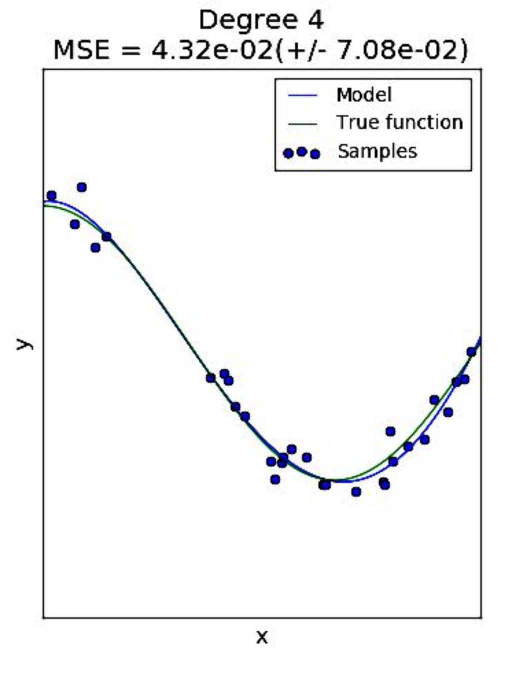
x_1	x_2	...				Y
0.5	0	0.6	1	0	0.25	1
0	0.5	[illegible]	[illegible]	[illegible]	0	1
0	0	[illegible]	[illegible]	[illegible]	0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

“overfitting”

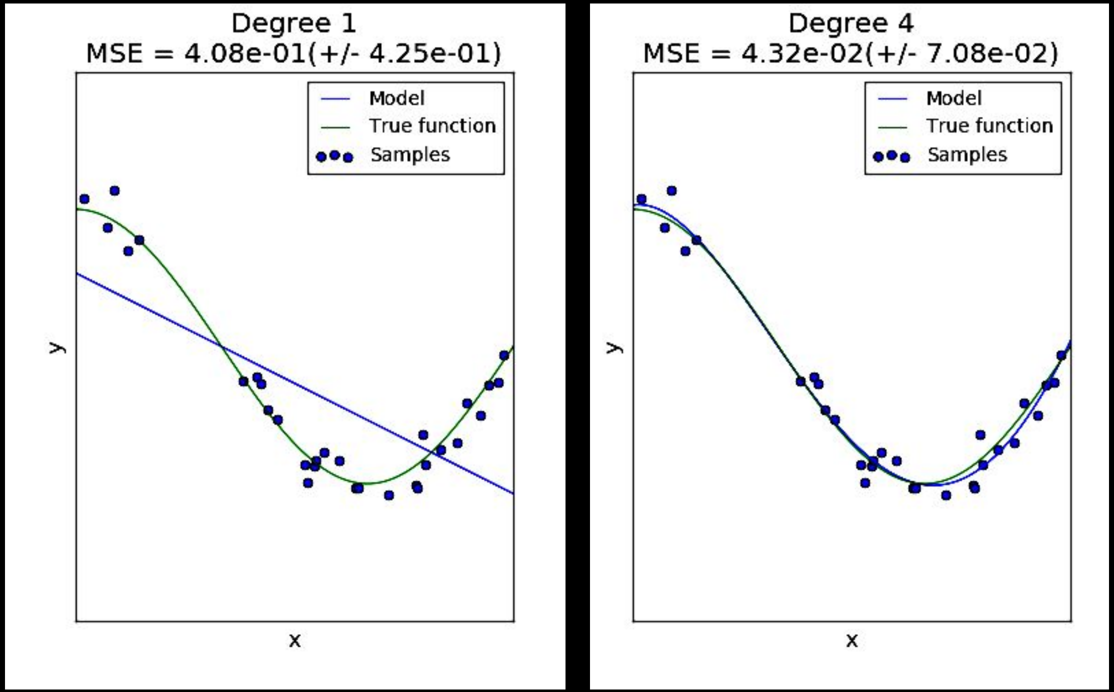
$$1.2 + -63*x_1 + \boxed{179*x_2} + 71*x_3 + 18*x_4 + -59*x_5 + 19*x_6 = logit(Y)$$

Python Example

Overfitting (1-d non-linear example)



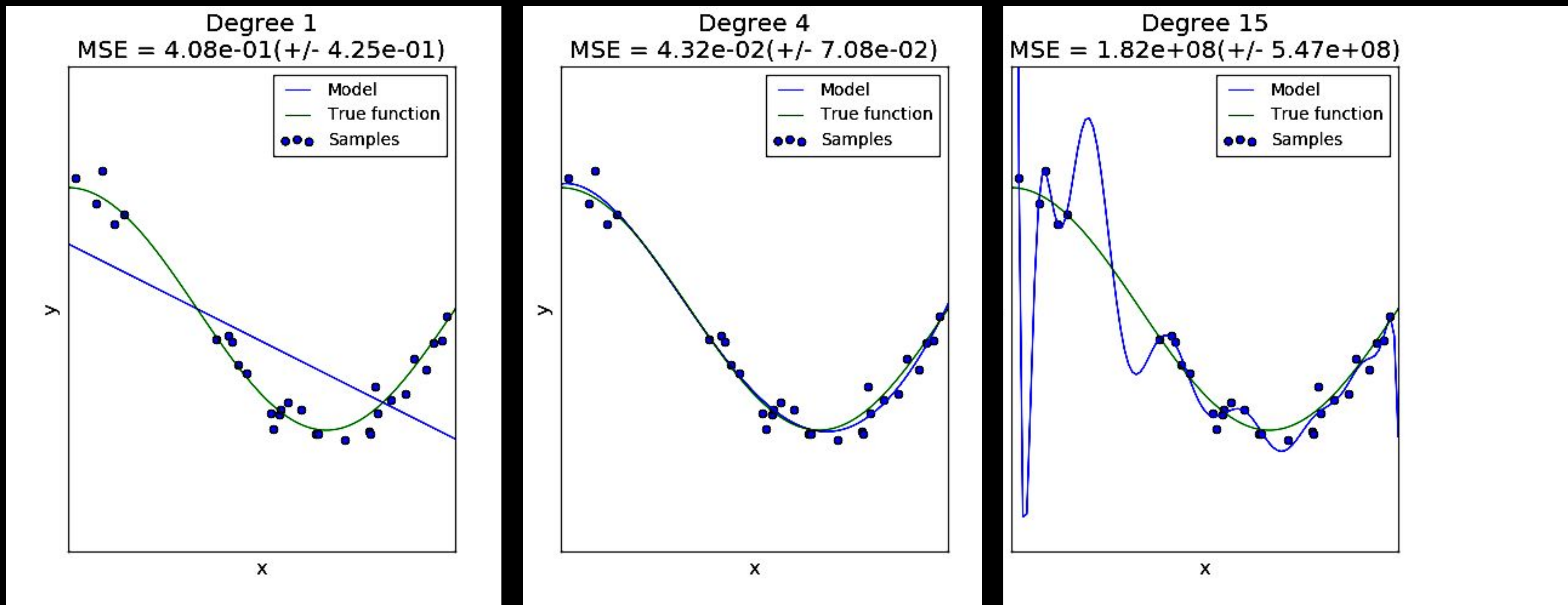
Overfitting (1-d non-linear example)



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

$$X \qquad = \qquad Y$$

x_1	x_2	…				Y
0.5	0	0.6	1	0	0.25	1
0	0.5	0.[illegible]	0	[illegible]	0	1
0	0	[illegible]	[illegible]	[illegible]	0.5	0
0	0	0	0	1	1	0
0.25	1	1.25	1	0.1	2	1

“overfitting”

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

Logistic Regression - Regularization

X = Y

x_1	x_2	...				Y
0.5	0	0.6	[illegible]	[illegible]	[illegible]	1
0	0.5	[illegible]	[illegible]	[illegible]	[illegible]	1
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0
0.25	[illegible]	[illegible]	[illegible]	0.1	2	1

"overfitting": generally due to trying to fit too many features given the number of observations.

1.2 + $-63 * x_1$ + 17[illegible]x_2 + $71 * x_3$ + $18 * x_4$ + $-59 * x_5$ + $19 * x_6$ = $logit(Y)$

Logistic Regression - Regularization

X $=$ Y

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?

Logistic Regression - Regularization

X $= Y$

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

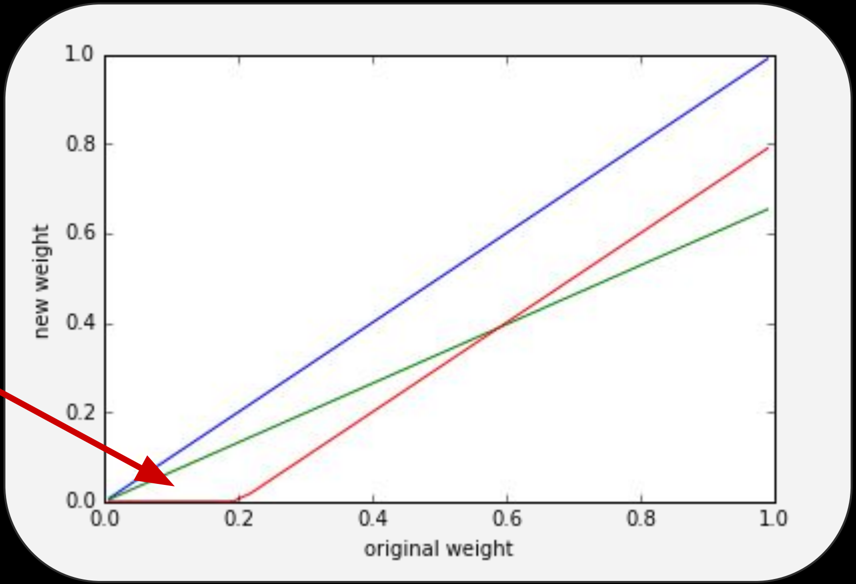
L1 Regularization - “The Lasso”

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.



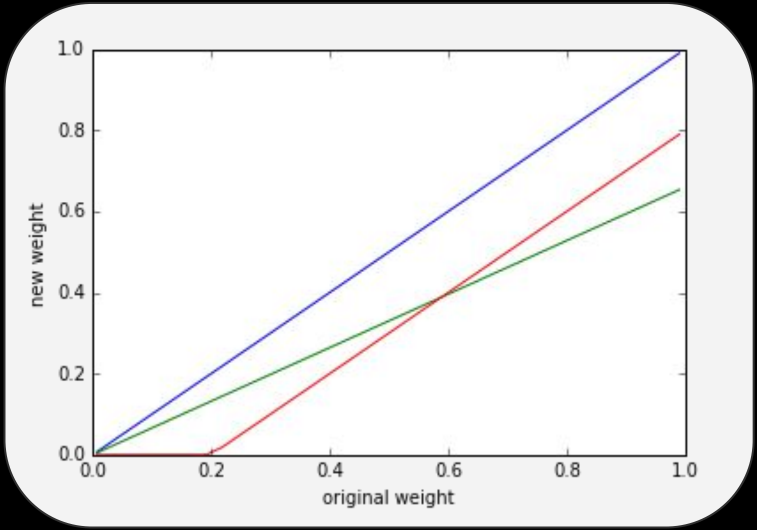
Logistic Regression - Regularization

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”

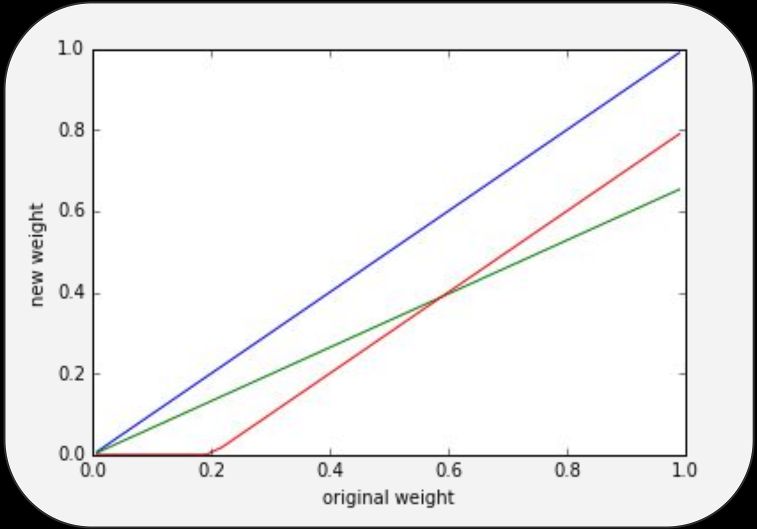
Zeros out features by adding values that keep from perfectly fitting the data.

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

set betas that maximize *penalized* L

This is for likelihood

for log loss, would add the penalty



Logistic Regression - Regularization

Sometimes written as:

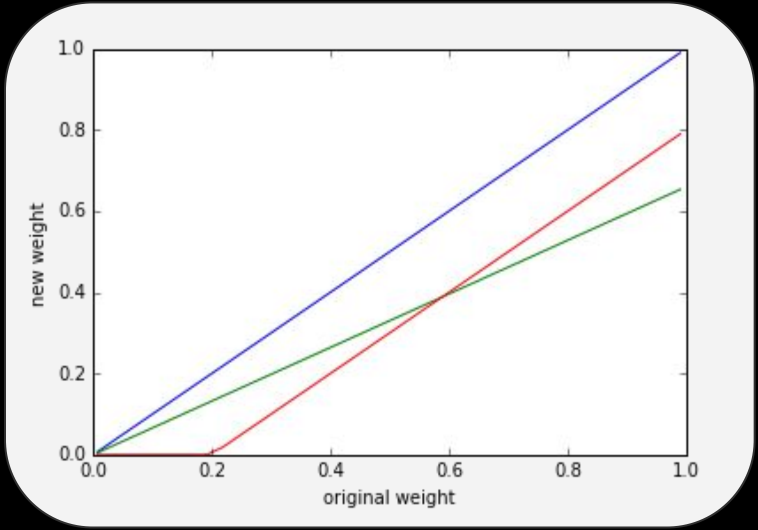
$$||\beta||_1$$

L1 Regularization - “The Lasso”

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

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

set betas that maximize *penalized* L



Logistic Regression - Regularization

Sometimes written as:

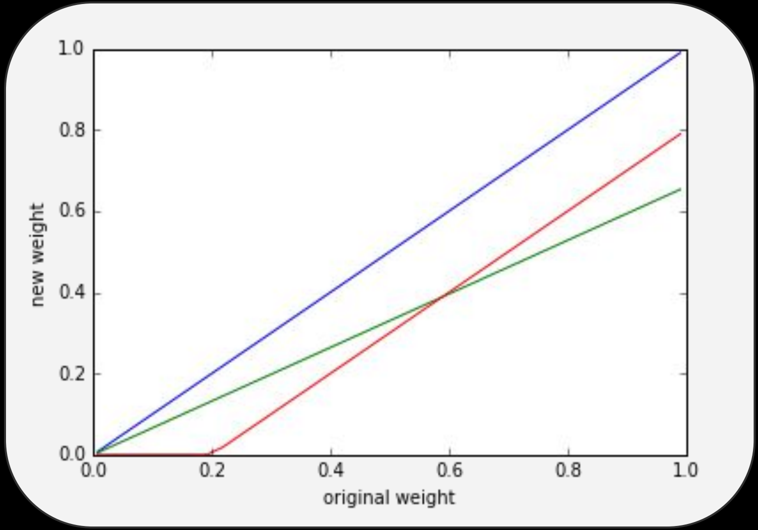
$$||\beta_j||_2^2$$

L2 Regularization - "Ridge"

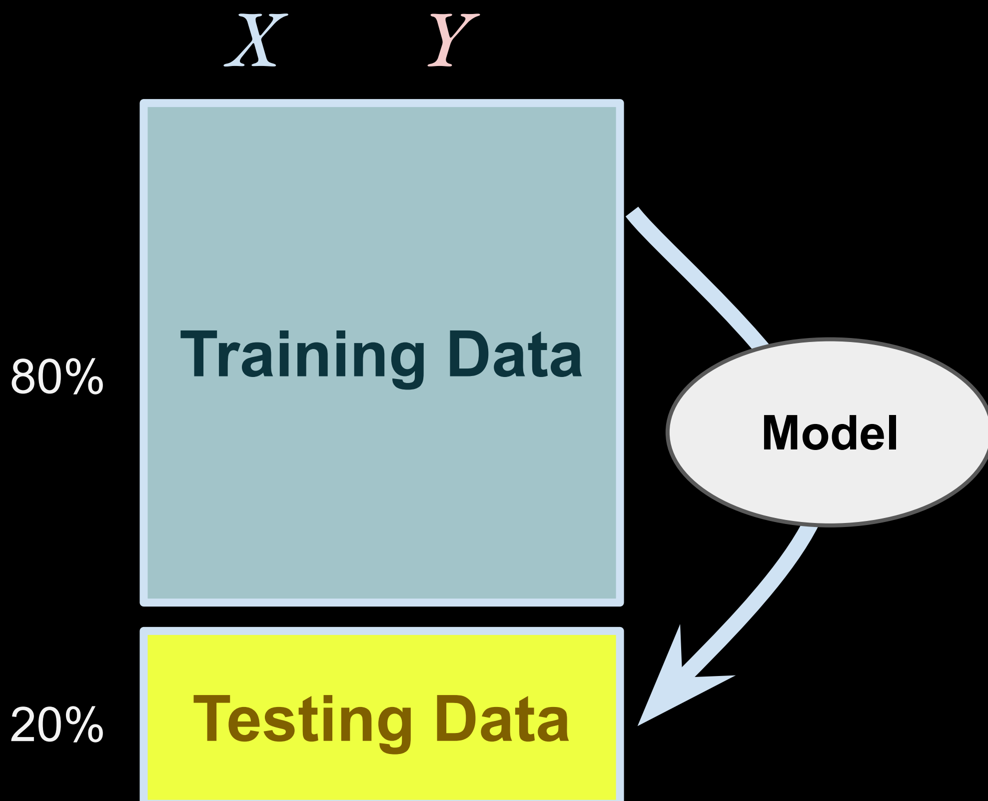
Shrinks features by adding values that keep from perfectly fitting the data.

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

set betas that maximize *penalized* L

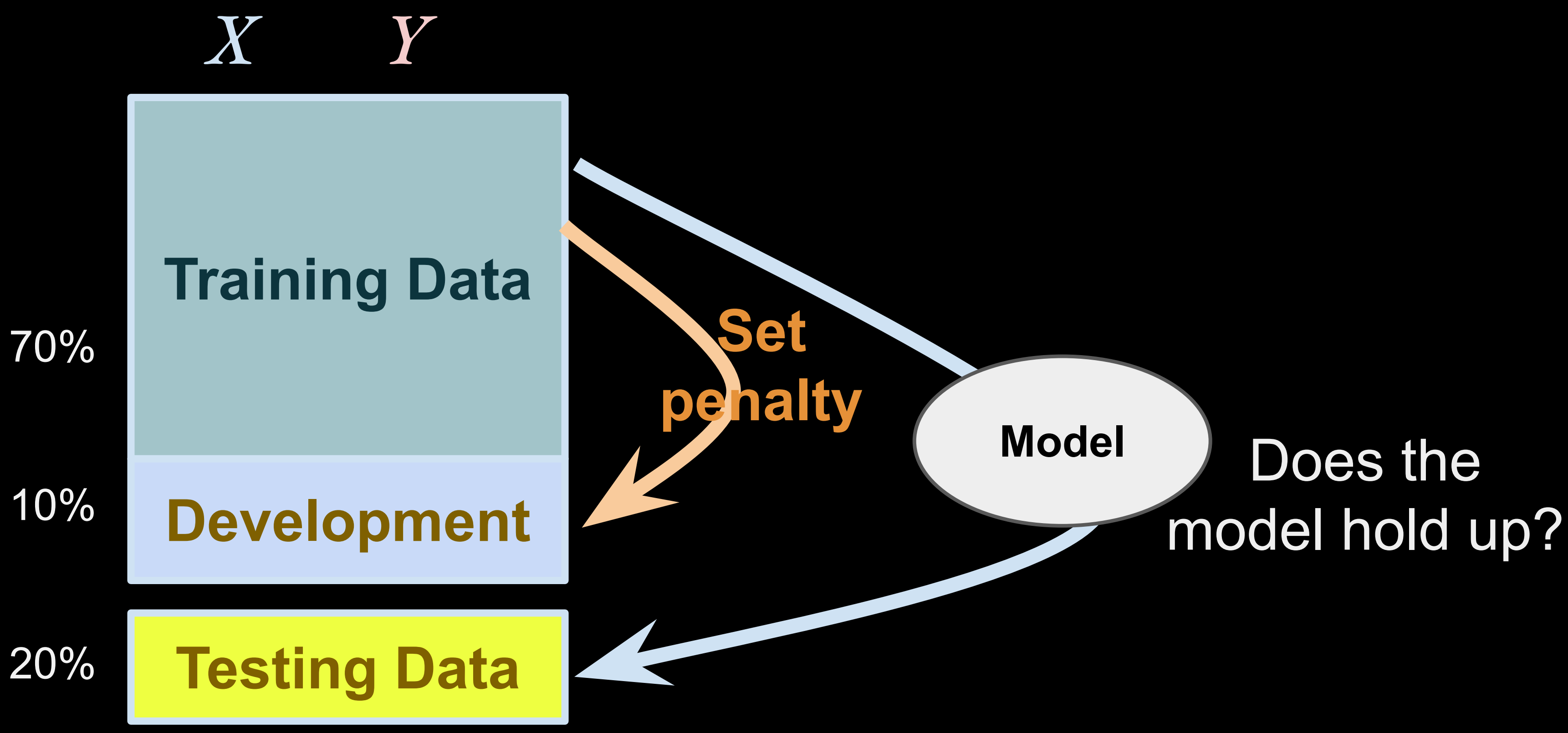


Machine Learning Goal: Generalize to new data



Does the model hold up?

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

Extra Material

Alternative to gradient descent:

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

$$p_i \equiv P(Y_i = 1 | X_i = x) = \frac{e^{\boxed{\beta_0} + \boxed{\beta_1} x_i}}{1 + e^{\boxed{\beta_0} + \boxed{\beta_1} x_i}}$$

To estimate β, one can use *reweighted least squares:*

(Wasserman, 2005; Li, 2010)

set $\hat{\beta}_0 = ... = \hat{\beta}_m = 0$ (remember to include an intercept)

1. Calculate p_i and let W be a diagonal matrix where element$(i,i) = p_i(1-p_i)$.
2. Set $z_i = logit(p_i) + \frac{Y_i - p_i}{p_i(1-p_i)} = X\hat{\beta} + \frac{Y_i - p_i}{p_i(1-p_i)}$
3. Set $\hat{\beta} = (X^T W X)^{-1} X^T W z$ //weighted lin. reg. of Z on Y.
4. Repeat from 1 until $\hat{\beta}$ converges.

Alternative to gradient descent:

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

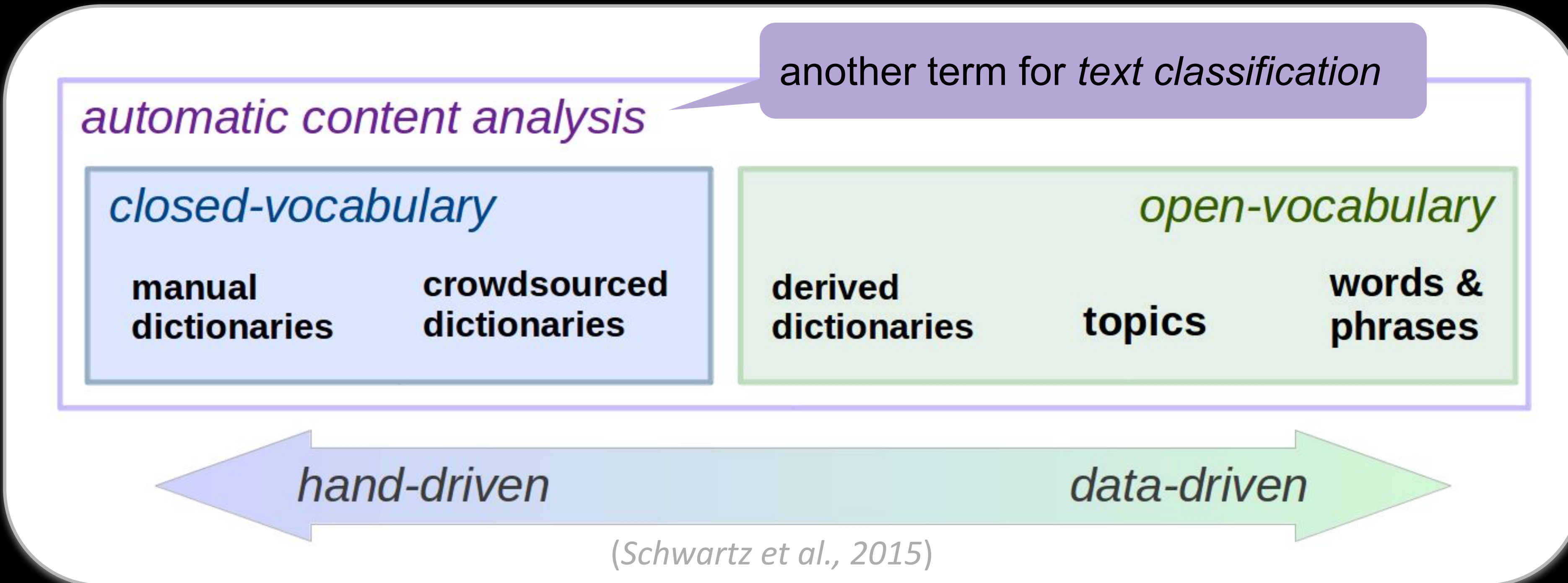
This is just one way of finding the betas that maximize the likelihood function. In practice, we will use existing libraries that are fast and support additional useful steps like **regularization**..

To estimate β , one can use *reweighted least squares:*

set $\hat{\beta}_0 = \ldots = \hat{\beta}_m = 0$ (remember to include an intercept)

1. Calculate p_i and let W be a diagonal matrix where element$(i, i) = p_i(1 - p_i)$.
2. Set $z_i = logit(p_i) + \frac{Y_i - p_i}{p_i(1 - p_i)} = X\hat{\beta} + \frac{Y_i - p_i}{p_i(1 - p_i)}$
3. Set $\hat{\beta} = (X^T W X)^{-1} X^T W z$ //weighted lin. reg. of Z on Y.
4. Repeat from 1 until $\hat{\beta}$ converges.

(Wasserman, 2005; Li, 2010)



	Proposed word lists	Accuracy	Ties
Human 3 + stats	positive: *love, wonderful, best, great, superb, still, beautiful* negative: *bad, worst, stupid, waste, boring, ?, !*	69%	16%

Figure 2: Results for baseline using introspection and simple statistics of the data (including *test* data).

PyTorch: 2. Numeric functions as a graph/network

(forward pass: defined in "forward" method of nn.Module)

```
class MultiClassLogReg(nn.Module):
    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, num_classes)

    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

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

        return self.linear(newX) #only use linear if using cross-entropy loss
```

Two equivalent options for multi-class:

option 1 (what the previous slides covered)

```
#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
                #alternative: nn.NLLLoss() #negative log likelikelihood loss
```

option 2

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

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