Logistic Regression

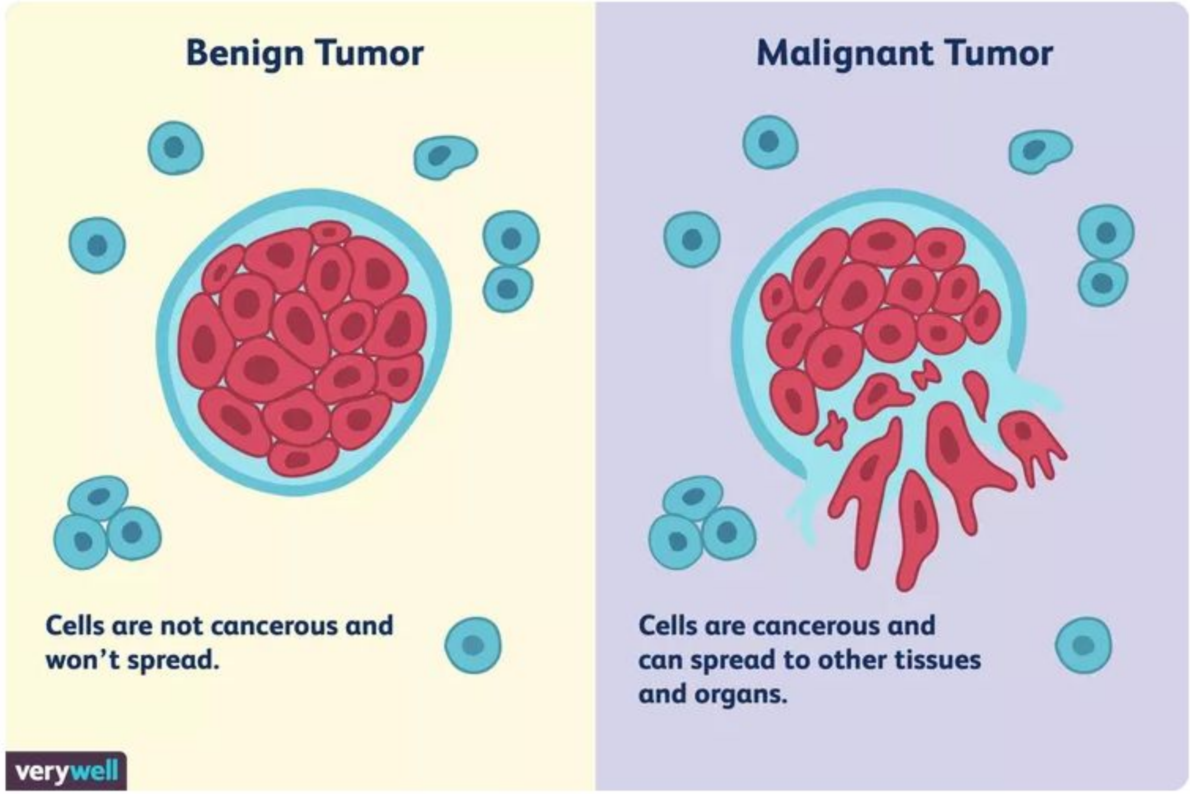
Original Slides by:
Courtney Anne Ngo
Daniel Stanley Tan, PhD
Arren Antioquia

Updated (AY 2024 – 2025 T1) by:
Thomas James Tiam-Lee, PhD

Logistic Regression

- **Supervised**, **classification** learning algorithm

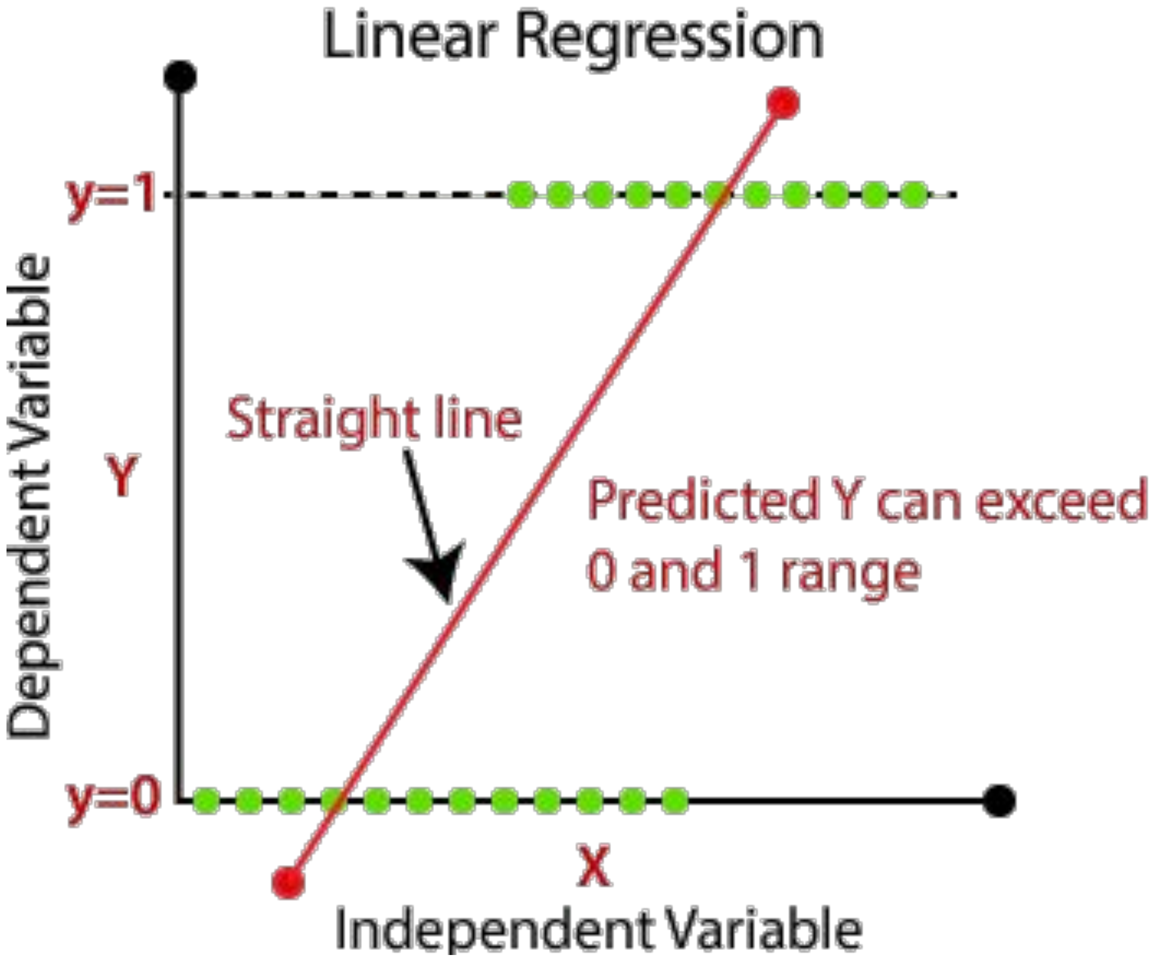
- Example:
  - predict whether a tumor is malignant or benign based on its size



Why Not Just Use Linear Regression?

- Predicted value can exceed the 0 – 1 range (does not make sense for binary classification)

- Lacks flexibility to handle certain configurations



Sigmoid Function

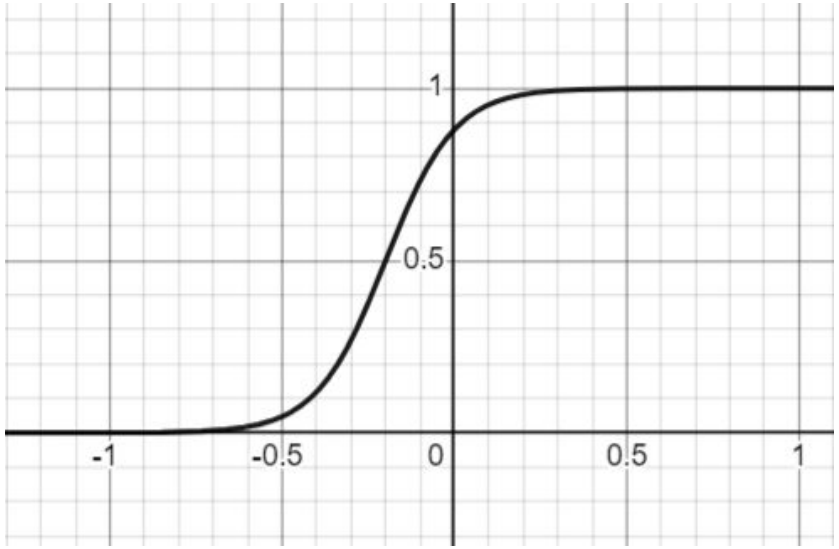
- Solution: “map” the output of a linear regression model to a 0 – 1 range using a **sigmoid function**

Sigmoid Function

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

Logistic Regression Model

$$z = \frac{1}{1 + e^{-(\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d + \theta_0)}}$$

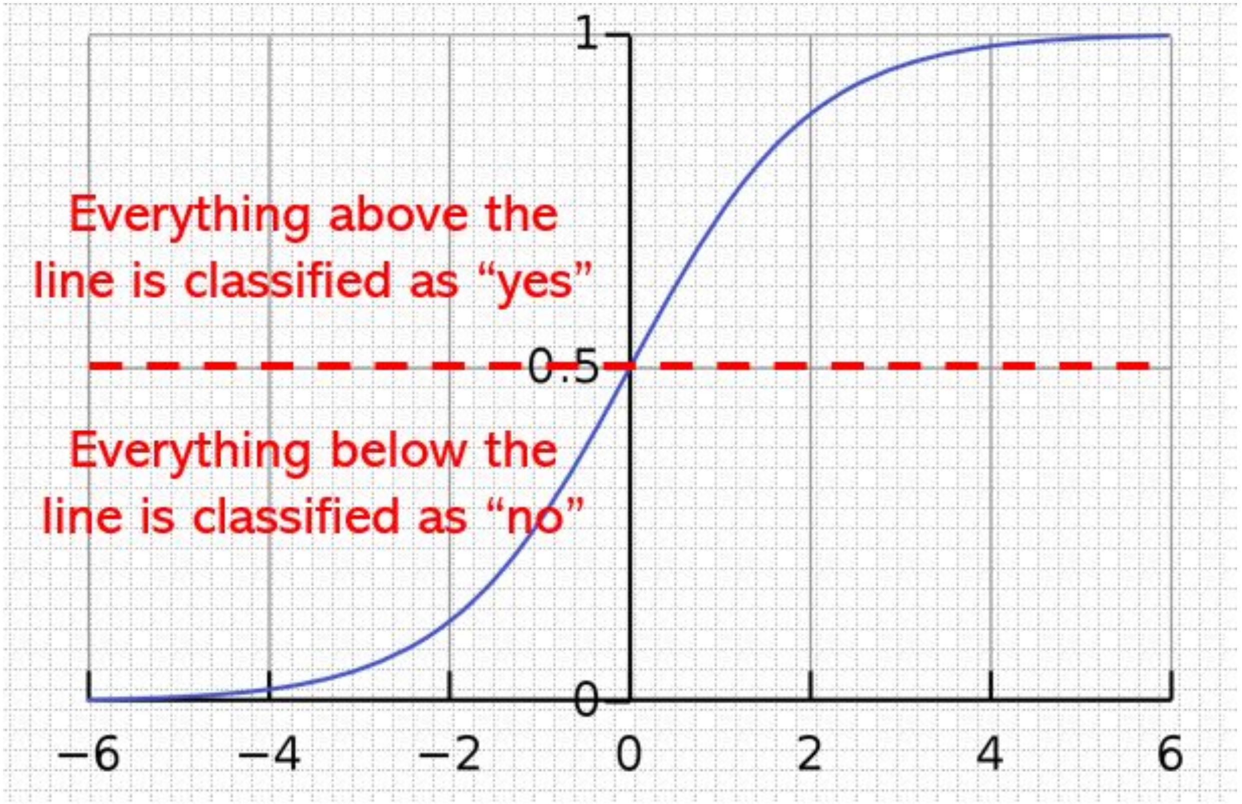


Sigmoid function is S-shaped, and always returns a value from 0 to 1 (exclusive)

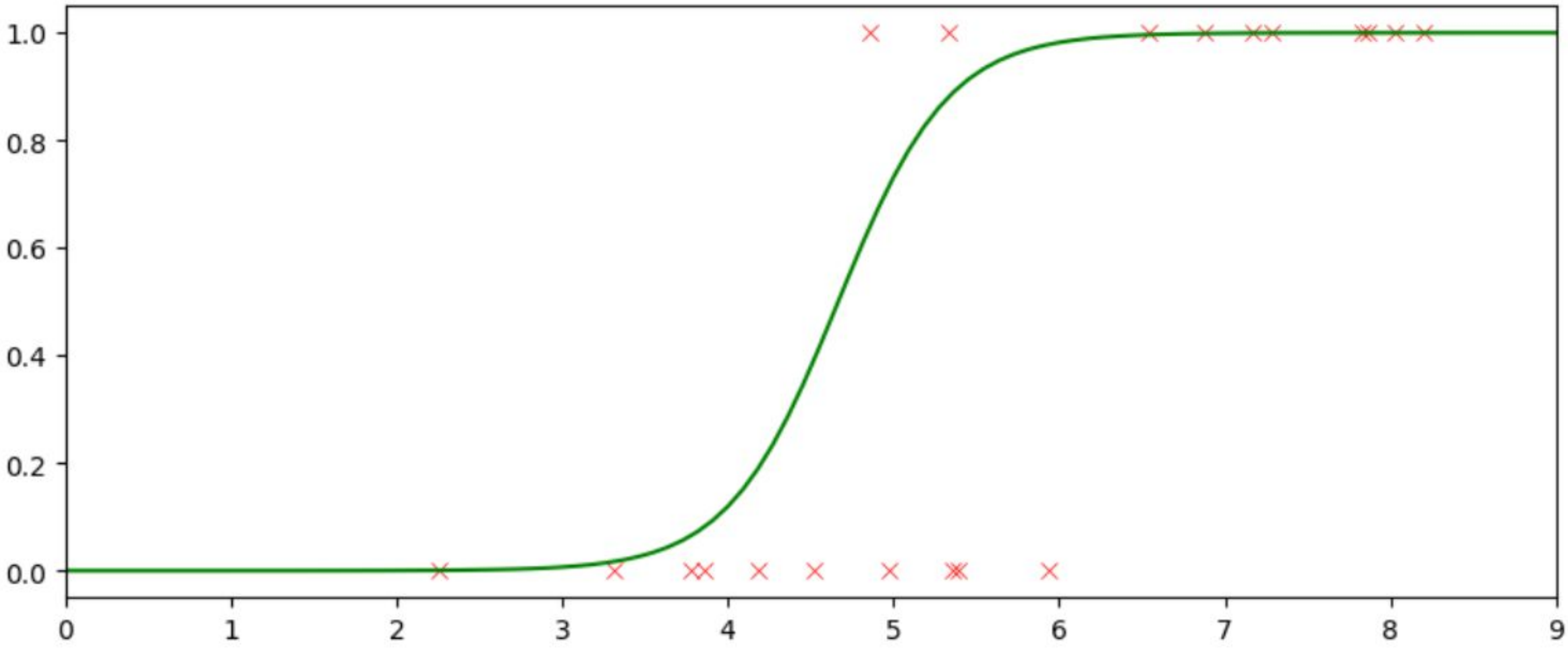
Interpretation of Model Output

- We interpret the output as the **probability that the input is classified as the positive class**.

- Mapping to class:
  - Set a threshold (i.e., 0.5)
    - If output ≥ threshold, classify as “yes”
    - If output ≤ threshold, classify as “no"



Logistic Regression Loss Function



Logistic Regression Loss Function

$$\prod_{i=1}^{n} \underbrace{{z^{(i)}}^{y^{(i)}}}_{\text{Used when the label is positive (1)}} \underbrace{\left(1 - z^{(i)}\right)^{(1-y^{(i)})}}_{\text{Used when the label is negative (0)}}$$

Logistic Regression Loss Function

$$\prod_{i=1}^{n} \underbrace{{z^{(i)}}^{y^{(i)}}}_{\text{Used when the label is positive (1)}} \underbrace{\left(1 - z^{(i)}\right)^{\left(1-y^{(i)}\right)}}_{\text{Used when the label is negative (0)}}$$

- Problem: this value can tend to be **very small**

Logistic Regression Loss Function

$$\sum_{i=1}^{n} \underbrace{y^{(i)} \log\left(z^{(i)}\right)}_{\text{Used when the label is positive (1)}} + \underbrace{\left(1 - y^{(i)}\right) \log\left(1 - z^{(i)}\right)}_{\text{Used when the label is negative (0)}}$$

- Solution: We use the **log probability** instead of the probability. This operation **preserves the order** (monotonic)

Logistic Regression Loss Function

$$\sum_{i=1}^{n} \underbrace{y^{(i)} \log\left(z^{(i)}\right)}_{\text{Used when the label is positive (1)}} + \underbrace{\left(1 - y^{(i)}\right) \log\left(1 - z^{(i)}\right)}_{\text{Used when the label is negative (0)}}$$

- Final issue: in a typical loss function, we want **lower scores** to be better!

Logistic Regression Loss Function

$$-\sum_{i=1}^{n} \underbrace{y^{(i)} \log\left(z^{(i)}\right)}_{\text{Used when the label is positive (1)}} + \underbrace{\left(1 - y^{(i)}\right) \log\left(1 - z^{(i)}\right)}_{\text{Used when the label is negative (0)}}$$

- Solution: just **negate** everything!

Logistic Regression Loss Function

$$l(\theta, X, y) = -\sum_{i=1}^{n} \underbrace{y^{(i)} \log\left(z^{(i)}\right)}_{\text{Used when the label is positive (1)}} + \underbrace{\left(1 - y^{(i)}\right) \log\left(1 - z^{(i)}\right)}_{\text{Used when the label is negative (0)}}$$

- Optimization problem: $\mathrm{argmin}_{\theta}\, l(\theta, X, y)$

Derivative of Loss Function

$$\frac{\partial}{\partial\theta}\left(-\sum_{i=1}^{n} y^{(i)}\log\left(z^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - z^{(i)})\right)$$

$$= (\boldsymbol{X\theta} - \boldsymbol{y})^{\boldsymbol{T}}\boldsymbol{X}$$

(essentially the same as the derivative of mean squared error)

Gradient Descent

procedure gradientdescent(θ):

while not converged do:

$$\theta := \theta - \alpha \frac{\partial}{\partial \theta} l(\theta)$$

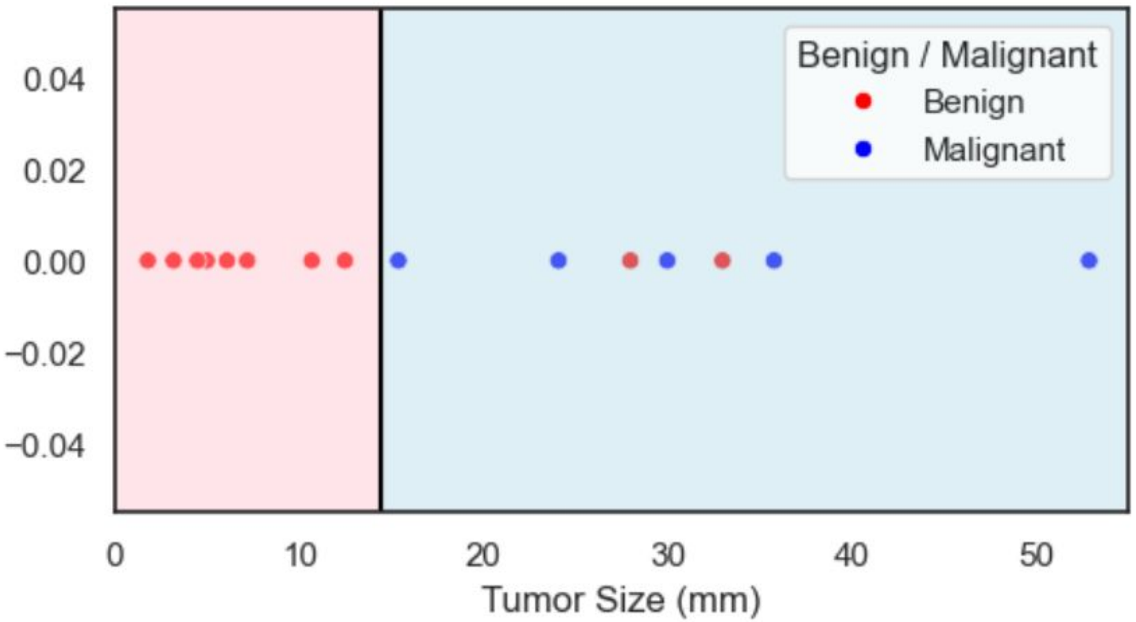
return θ

α is the
learning rate,
determines how large the
update will be

$\frac{\partial}{\partial \theta_i} l(\theta)$ is the
gradient of the loss

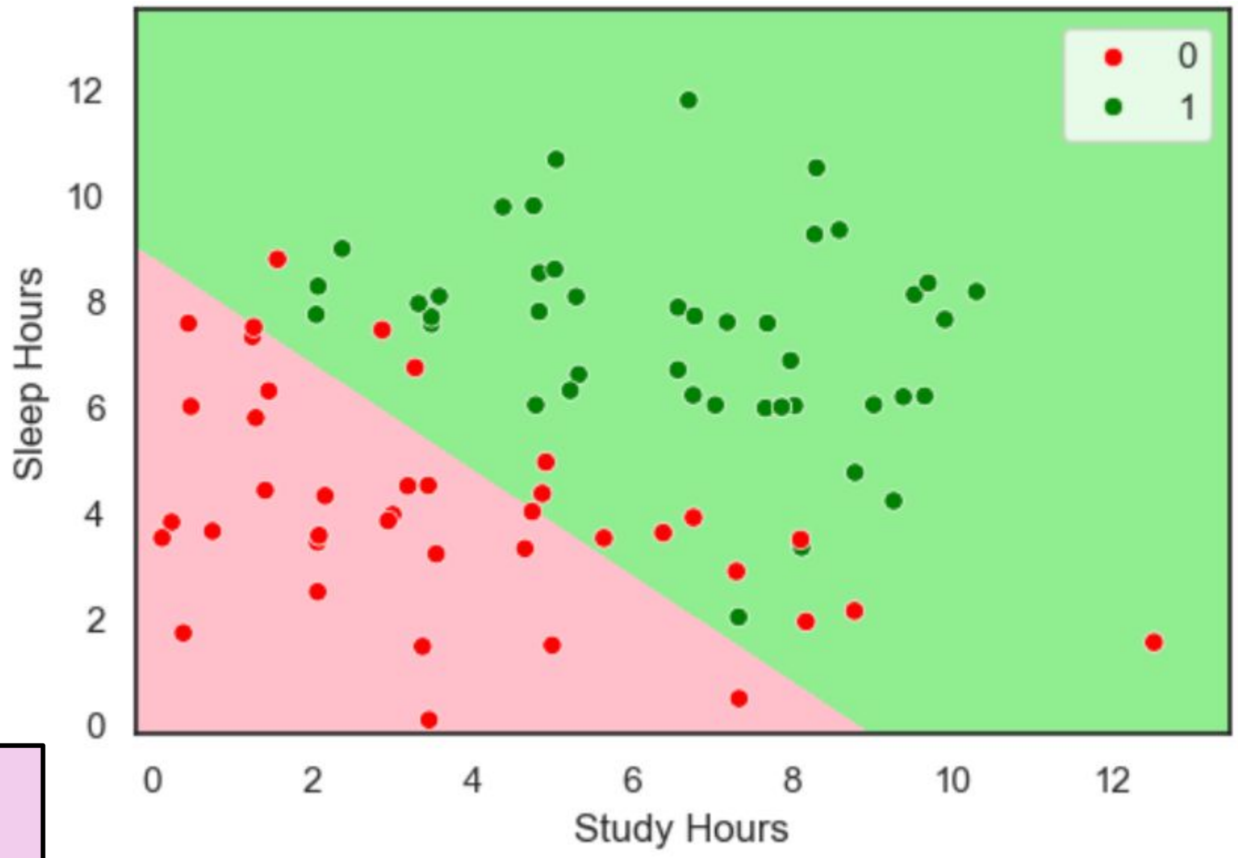
Decision Boundary

- Shows the prediction results across the feature space.



1 Feature

Even though sigmoid is curvy, the decision boundaries are always straight lines (with respect to the feature space)! Think about why this is the case.



2 Features

Multinomial Logistic Regression

- What if we have more than 2 classes?

$$prob_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

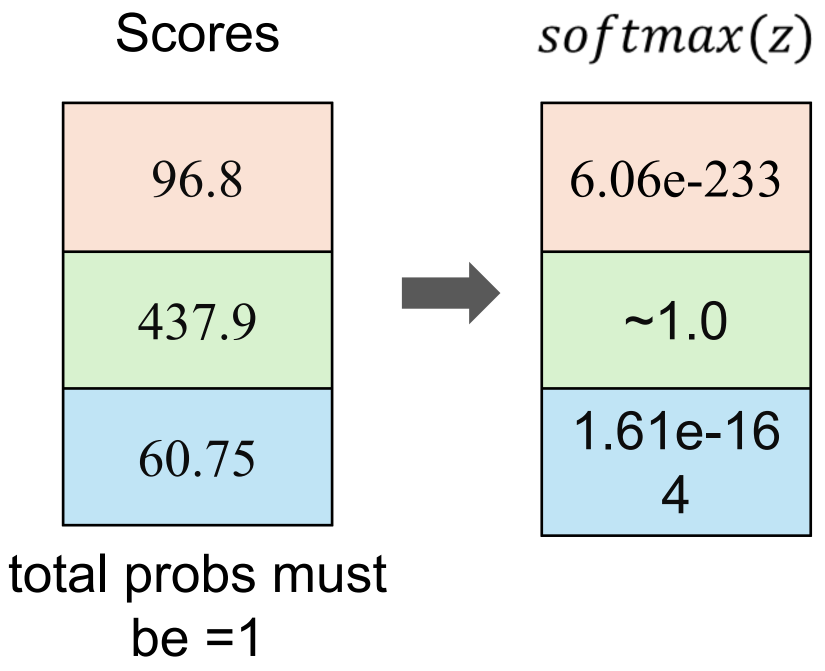
$$\underset{\text{input}}{\overset{x_i}{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}} \quad \underset{\text{a set of weights for each class}}{\overset{W}{\begin{bmatrix} 0.2 & 1.5 & 0 \\ -0.5 & 1.3 & 0.25 \\ 0.1 & 2.1 & 0.2 \\ 2 & 0 & -0.3 \end{bmatrix}}} + \underset{\text{a bias for each class}}{\overset{b}{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}} \Rightarrow \underset{\text{total score before probability}}{\overset{z = x_i W + b}{\begin{bmatrix} 96.8 \\ 437.9 \\ 60.75 \end{bmatrix}}} \Rightarrow \underset{\text{total probs must be =1}}{\overset{softmax(z)}{\begin{bmatrix} 6.06\text{e-}233 \\ \sim 1.0 \\ 1.61\text{e-}164 \end{bmatrix}}}$$

- It's like we just created 3 "separate" classifiers, and choose the largest score.

Softmax Function

$$prob_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

- **Goal:** Convert the scores into a probabilistic representation, that totals into 1.



- We can also just get the largest score, but we do softmax because:
  - We want to interpret results as probability
  - We need to get the derivative, especially later in neural networks

Multinomial Log. Reg. Loss Function

$$l(\theta, X, y) = -\sum_{i=1}^{n} {y^{(i)}}^T \log\left(p^{(i)}\right)$$

- **Note:** In multiclass classification the "label" for a single instance is converted to a one-hot encoded vector.

Class
Red
Red
Blue
Red
Blue
Green
Green
Green
Blue

Class	Predictions
[1, 0, 0]	[0.5, 0.2, 0.3]
[1, 0, 0]	[0.6, 0.1, 0.4]
[0, 1, 0]	[0.1, 0.8, 0.1]
[1, 0, 0]	[0.8, 0.1, 0.1]
[0, 1, 0]	[0.7, 0.2, 0.1]
[0, 0, 1]	[0.1, 0.8, 0.1]
[0, 0, 1]	[0.3, 0.5, 0.2]
[0, 0, 1]	[0.2, 0.2, 0.6]
[0, 1, 0]	[0.5, 0.4, 0.1]

Multinomial Log. Reg. Loss Function

$$l(\theta, X, y) = -\sum_{i=1}^{n} {y^{(i)}}^T \log(p^{(i)})$$

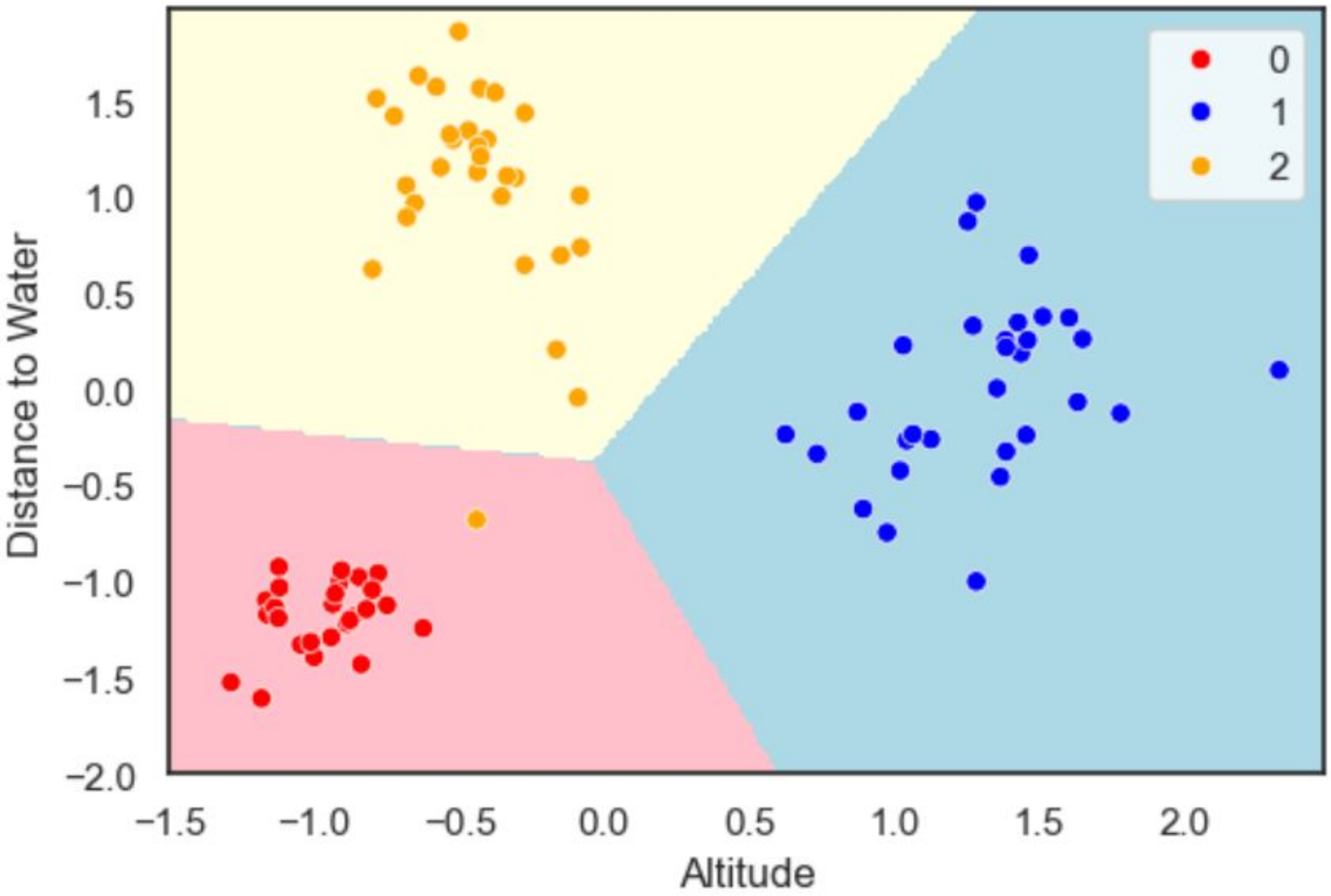
The loss function totals the log of these probabilities and negates it

- **Note:** In multiclass classification the "label" for a single instance is converted to a one-hot encoded vector.

Class
Red
Red
Blue
Red
Blue
Green
Green
Green
Blue

Class	Predictions
[1, 0, 0]	[0.5, 0.2, 0.3]
[1, 0, 0]	[0.6, 0.1, 0.4]
[0, 1, 0]	[0.1, 0.8, 0.1]
[1, 0, 0]	[0.8, 0.1, 0.1]
[0, 1, 0]	[0.7, 0.2, 0.1]
[0, 0, 1]	[0.1, 0.8, 0.1]
[0, 0, 1]	[0.3, 0.5, 0.2]
[0, 0, 1]	[0.2, 0.2, 0.6]
[0, 1, 0]	[0.5, 0.4, 0.1]

Decision Boundary for Multiclass



Evaluation of Classification Models

- Confusion Matrix
- Accuracy
- Precision
- Recall
- F1-Score

Confusion Matrix

- Shows us the statistics of the prediction

		Classified As	
		1	0
Real Label	1	48	2
	0	1	49

Confusion Matrix

- Shows us the statistics of the prediction

Classified As

Real Label

	1	0
1	True positive	False negative
0	False positive	True negative

Accuracy

- Number of correctly classified instances over all instances

- $$\frac{TP+TN}{TP+TN+FP+FN}$$

		Classified As	
		1	0
Real Label	1	True positive	False negative
	0	False positive	True negative

Precision

- Out of all instances predicted as positive, how many are really positive?

- $$\frac{TP}{TP+FP}$$

Classified As

Real Label	1	0
1	True positive	False negative
0	False positive	True negative

Recall

- Out of all positive instances, how many are predicted as positive?

- $$\frac{TP}{TP+FN}$$

Classified As

Real Label		1	0
	1	True positive	False negative
	0	False positive	True negative

Why is Accuracy Not Enough?

- Very high accuracy (98%), but very stupid model (just predicts 0 all the time)
- Imbalanced datasets amplify this problem
- In some domains, need to consider which metrics are important!

Classified As

Real Label

	1	0
1	0	2
0	0	98

F1-Score

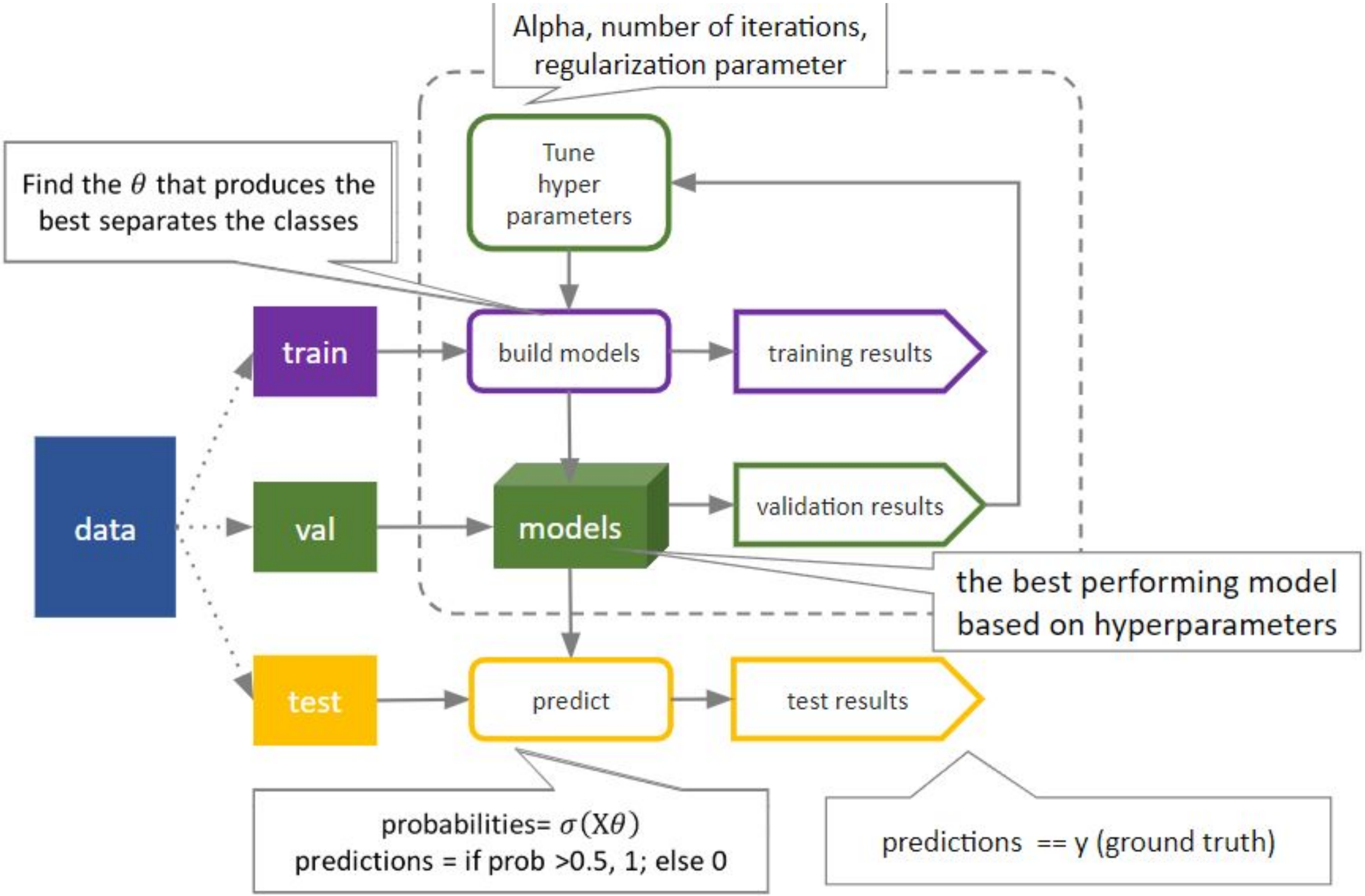
- Harmonic mean of the precision and recall

- $$2 \times \frac{precision \times recall}{precision + recall}$$

Multiclass Classification

- For multi-class classification, you have a separate precision / recall / F1-score for each class!

		Predicted		
		Negative	Neutral	Positive
Actual	Negative	700	300	0
	Neutral	200	8300	100
	Positive	0	100	300


Alpha, number of iterations, regularization parameter
Find the θ that produces the best separates the classes
Tune hyper parameters
train
build models
training results
data
val
models
validation results
the best performing model based on hyperparameters
test
predict
test results
probabilities= $\sigma(X\theta)$
predictions = if prob >0.5, 1; else 0
predictions == y (ground truth)