Review

Topics

Model representation

Loss and cost functions

Gradient descent

Forward / backward propagation

Regularization

Model evaluation

Classification

Vectorization

Model Representation

weights & biases

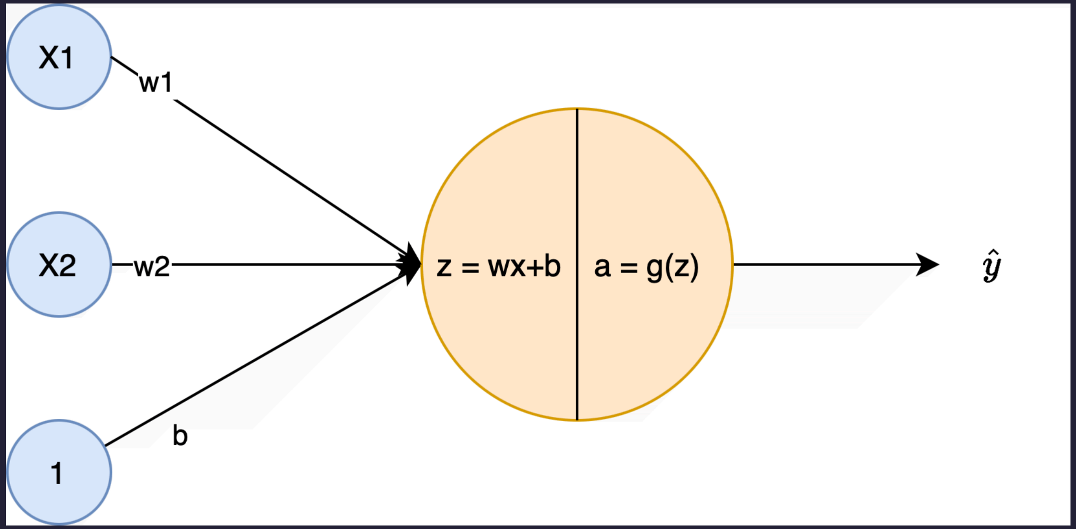
activation functions (output layer)

- linear (identity) for regression
- sigmoid for binary classification
- softmax for multi-class classification

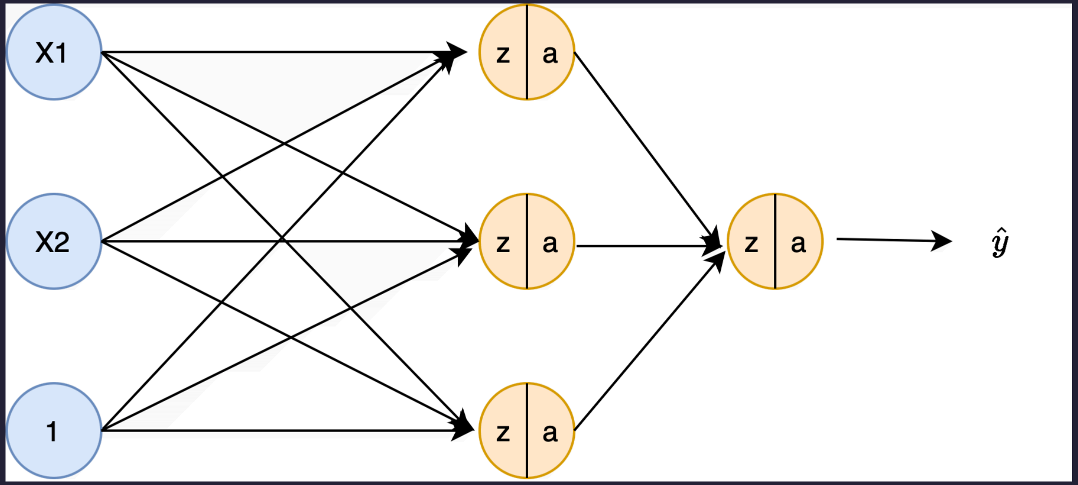
activation functions (hidden layers; neural networks only)

- tanh
- ReLU

Model Representation (cont.)



$$z = wx + b$$
$$a = g(z)$$



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$

Loss and Cost Functions

Loss function $L(\hat{y}, y)$: the error between the predicted and true values

- Mean Squared Error (MSE): $(y - \hat{y})^2$
- Mean Absolute Error (MAE): $|y - \hat{y}|$
- Binary cross-entropy: $-y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$
- Cross-entropy: $-\sum y \log(\hat{y})$

Cost function $J(w, b)$: the average loss over the entire dataset

Gradient Descent

Update weights and biases to minimize the cost function

Repeat until convergence

$$w = w - \alpha \frac{\partial J}{\partial w}$$
$$b = b - \alpha \frac{\partial J}{\partial b}$$

convergence: small change in cost function over iterations

learning rate (α): step size

- too small: slow convergence
- too large: overshooting

Forward Propagation ➡️

input: $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

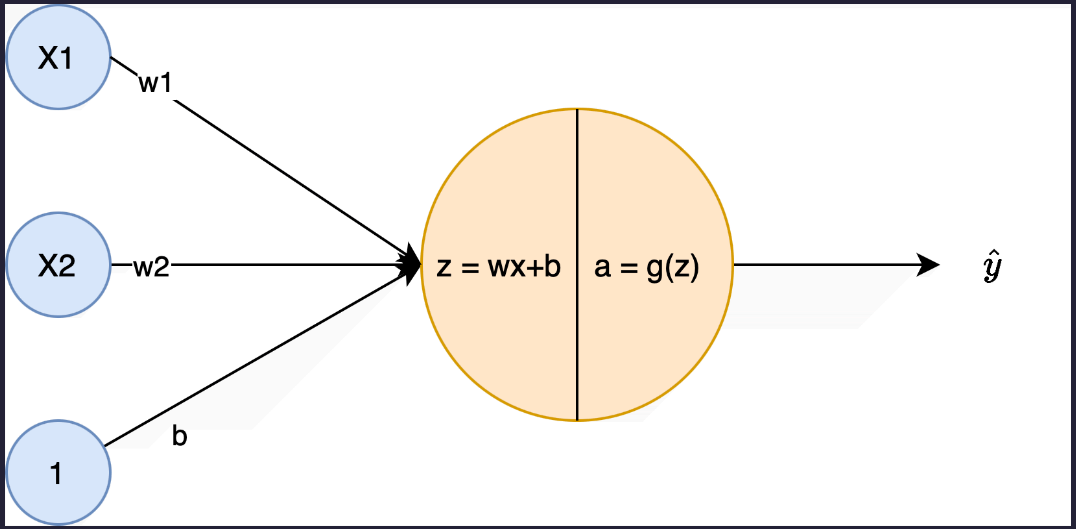
activation function: $g(z) = 1/(1 + e^{-z})$

initial weights: $w = \begin{bmatrix} 1 & 2 \end{bmatrix}$

initial bias: $b = 3$

$$z = wx + b = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 3 = 1 \cdot 1 + 2 \cdot 2 + 3 = 8$$

$$a = g(z) = \frac{1}{1 + e^{-8}} \approx 0.9997$$



Forward Propagation ➡️ using `numpy`

input: $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

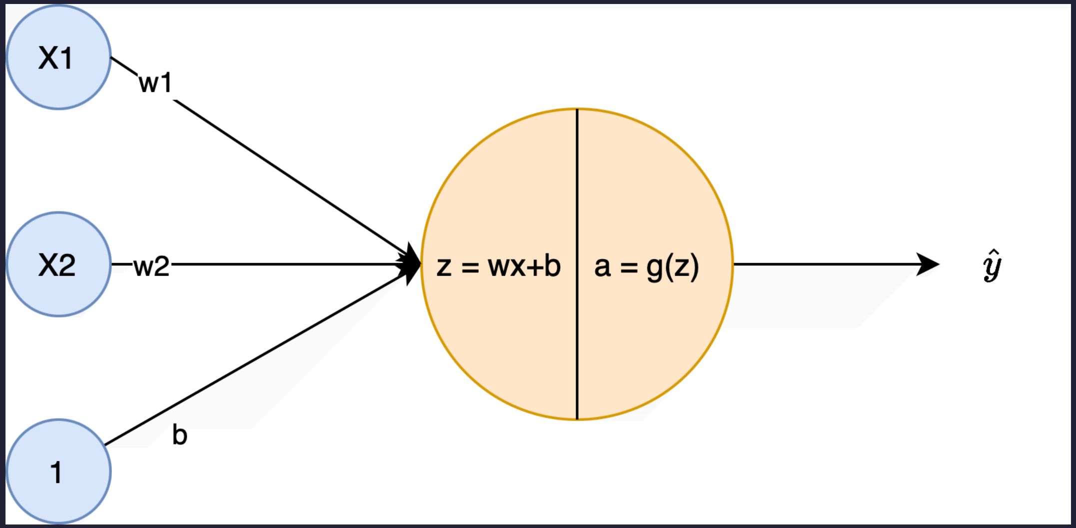
activation function: $g(z) = 1/(1 + e^{-z})$

initial weights: $w = \begin{bmatrix} 1 & 2 \end{bmatrix}$

initial bias: $b = 3$

```
x = np.array([1, 2])
w = np.array([1, 2])
b = 3

z = np.dot(w, x) + b
a = 1 / (1 + np.exp(-z))
print(a)
```



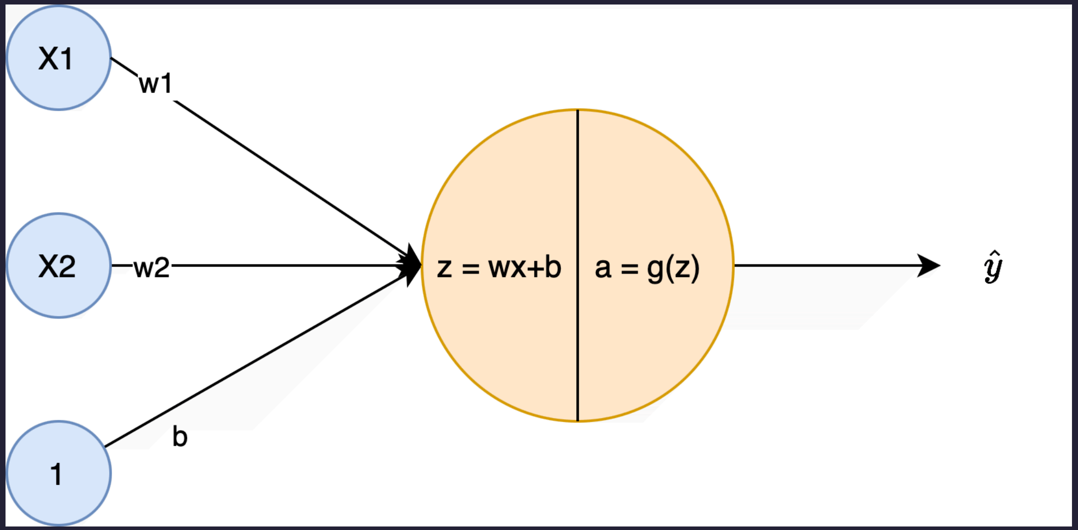
Compute loss

true value: $y = 1$

loss function: binary cross-entropy

$$\begin{aligned} L(a, y) &= -y \log(a) - (1 - y) \log(1 - a) \\ &= -1 \log(0.9997) - (1 - 1) \log(1 - 0.9997) \\ &\approx -0.0003 \end{aligned}$$

```
y = 1
loss = -y * np.log(a) - (1-y) * np.log(1-a)
print(loss)
```



Backward Propagation ⬅️

compute gradients

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}$$

$$= (a - y)x = (0.9997 - 1)\cdot\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}-0.0003\\-0.0006\end{bmatrix}$$

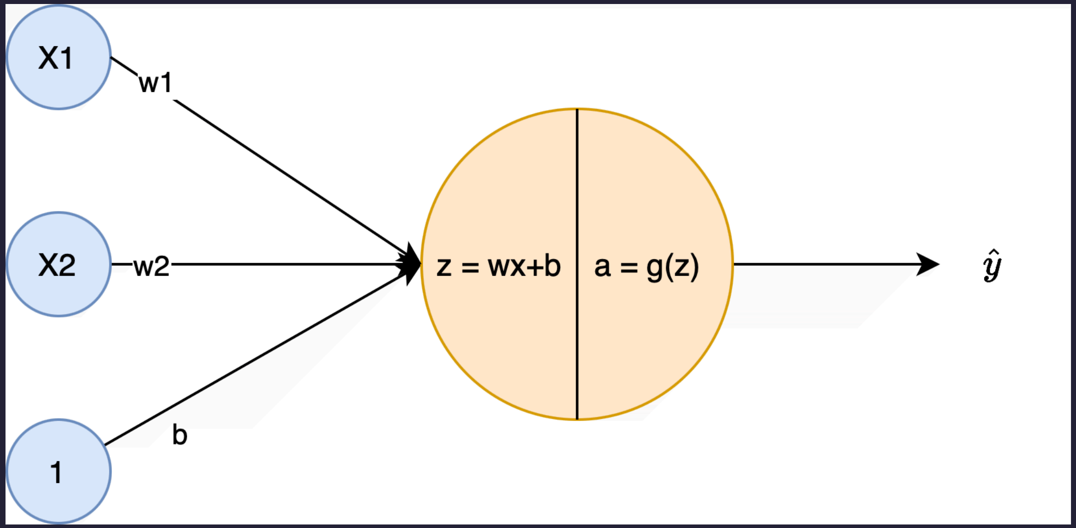
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

$$= a - y = 0.9997 - 1 = -0.0003$$

update weights and bias ($\alpha = 1$)

$$w = w - \alpha\frac{\partial L}{\partial w} = \begin{bmatrix}1\\2\end{bmatrix} - \begin{bmatrix}-0.0003\\-0.0006\end{bmatrix} = \begin{bmatrix}1.0003\\2.0006\end{bmatrix}$$

$$b = b - \alpha\frac{\partial L}{\partial b} = 3 - (-0.0003) = 3.0003$$



Regularization

Minimize both `loss` and `complexity`

$$J(\vec{w}, b) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)})}_{\text{loss}} + \lambda \underbrace{\sum_{j=1}^{n} w_j^2}_{\text{complexity}}$$

- j: index of the feature ($j = 1, 2, \ldots, n$)
- w_j: weight of the feature j
- **loss**: how well the model fits the data (same as before)
- **complexity**: how complex the model is
- λ **(lambda): regularization parameter**
  - large λ: **Complexity** dominates
  - small λ: **Complexity** close to zero $\Rightarrow$ Non-regularized model

Model Evaluation

Evaluate generalization performance

- Training set: used to train the model
- Validation set: used to tune hyperparameters
- Test set: used to evaluate the model

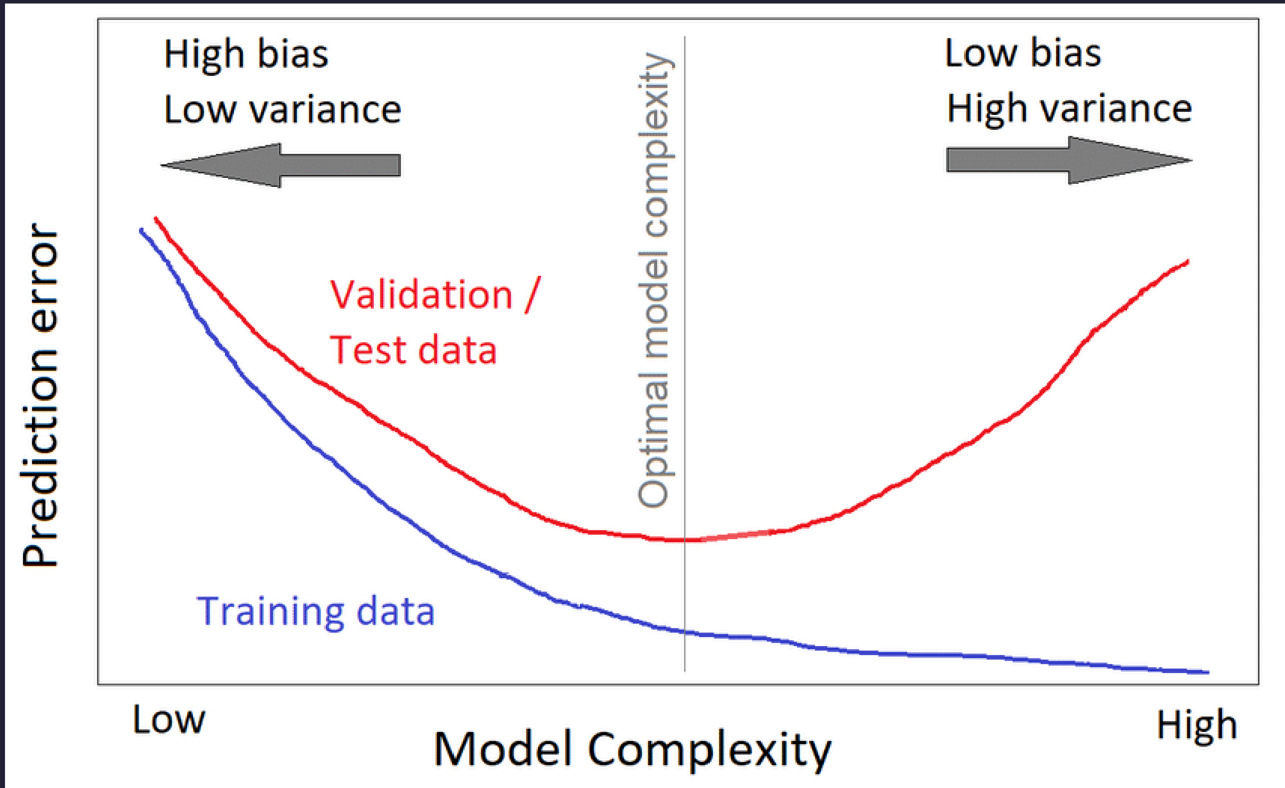
Bias-variance tradeoff

High variance (overfit)

- J_{train} is low
- J_{cv} is high
- $J_{cv} > J_{train}$

High bias (underfit)

- J_{train} is high
- J_{cv} is high
- $J_{cv} \approx J_{train}$



Will collecting more data help?

$J_{train} < J_{cv}$

➡️ high variance (overfit)

➡️ More data helps

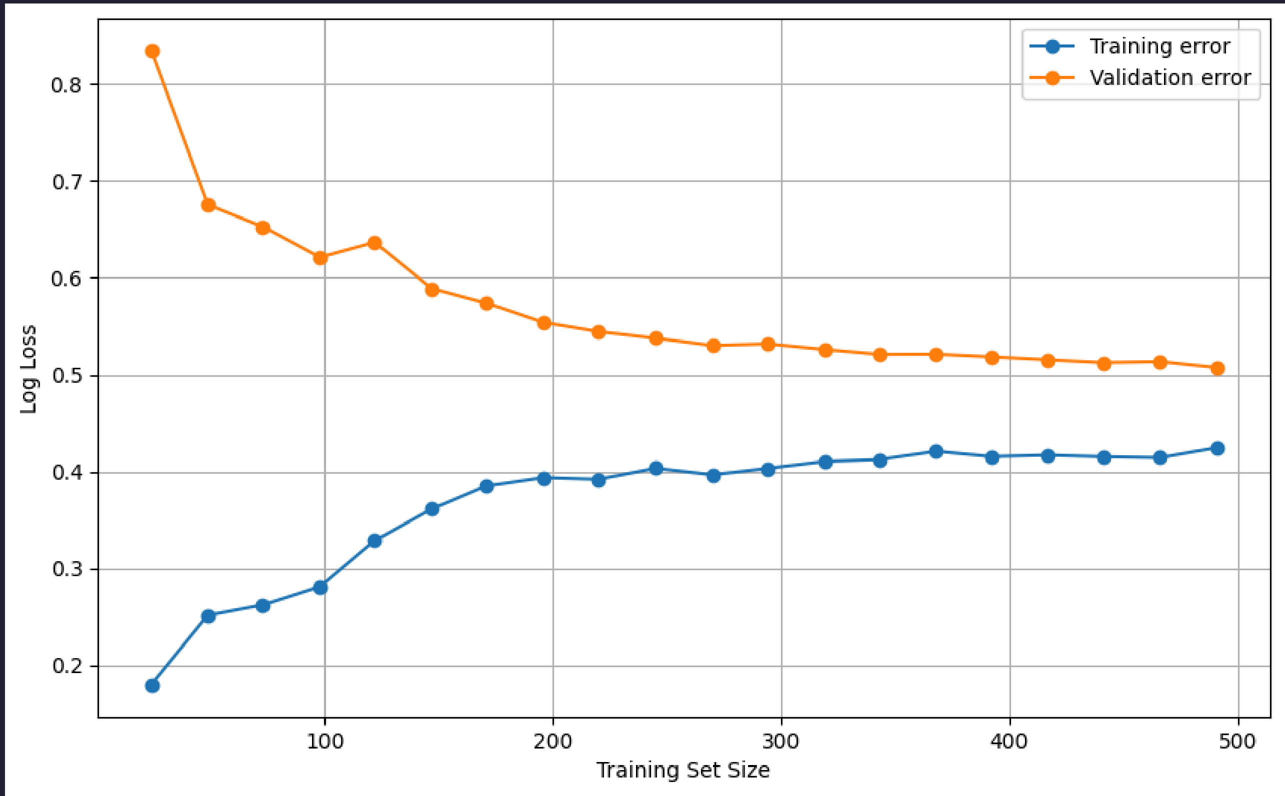
$J_{train} \approx J_{cv}$

$J_{train} > J_{reference}$

➡️ high bias (underfit)

➡️ More data won't help

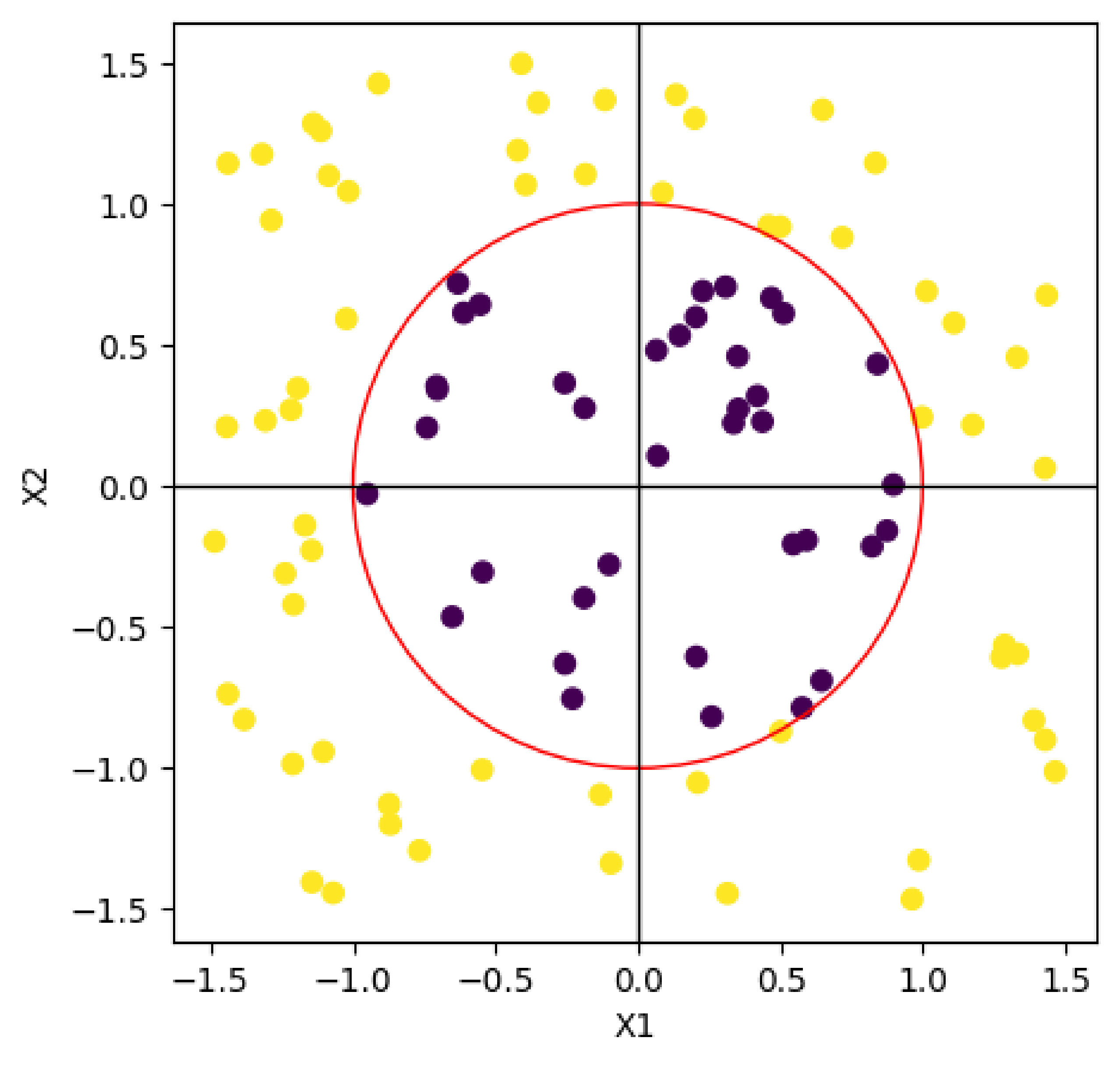
➡️ Fit a more complex model



Classification

Decision boundary

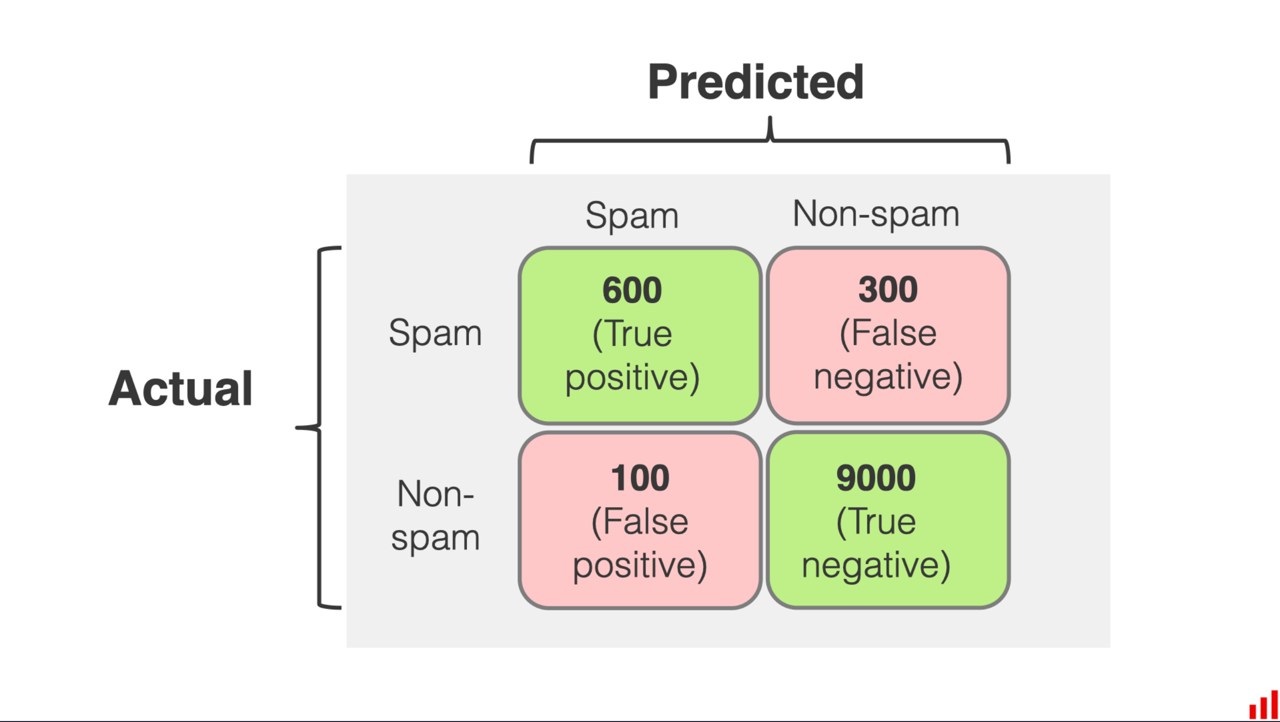
- from probability (continuous) to class label (0 or 1)
- set of points where the model outputs 0.5
- separates the classes



Classification

Confusion matrix

- different types of correct and incorrect predictions
- classifiers can be evaluated using various metrics



Classification

Evaluation metrics

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- ROC AUC: Area under the ROC curve
- PR AUC: Area under the Precision-Recall curve

When to use which metric?

Vectorization

- Avoid explicit loops
- Use matrix operations
  - element-wise operations
  - matrix multiplication (dot product)
  - broadcasting: operations between arrays of different shapes

```
# loop
for i in range(1000):
    z[i] = w[i] * x[i] + b

# vectorization
z = np.dot(w, x) + b
```

Broadcasting

```
A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2, 104.0, 52.0, 8.0],
              [1.8, 135.0, 99.0, 0.9]]) # (3, 4)

# sum along the column
cal = A.sum(axis=0)          # array([ 59. , 239. , 155.4,  76.9]) (4,)
percentage = 100 * A / cal   # broadcasting cal from (4,) to (3, 4)

# reshape from (4,) to (1, 4)
cal = A.sum(axis=0)          # array([ 59. , 239. , 155.4,  76.9]) (4,)
cal = cal.reshape(1, 4)      # array([[ 59. , 239. , 155.4,  76.9]]) (1, 4)
percentage = 100 * A / cal   # broadcasting cal from (1, 4) to (3, 4)

# reshape using keepdims=True
cal = A.sum(axis=0, keepdims=True)  # array([[ 59. , 239. , 155.4,  76.9]]) (1, 4)
percentage = 100 * A / cal          # broadcasting cal from (1, 4) to (3, 4)
```

Exam format

- **75 minutes**
- **20-25 questions (with sub-questions)**
- **Multiple choice, short answer**
  - concepts, definitions, calculations
  - model representation, forward/backward propagation written in Python code or math equations
- **Review the slides, labs, and assignments**
  - Week 3 ~ 9 (up to Neural Networks)
- **Closed book, 1-page handwritten cheat sheet (both sides)**