Northeastern University
Khoury College of
Computer Sciences

Midterm Review

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

Wednesday | February 18, 2026

Models Seen So Far

Supervised	Unsupervised
Linear Regression	PCA
Logistic Regression	k-Means Clustering
k-NN	
LDA	

Linear Regression

Linear Regression

Model:

$$\hat{y} = \theta_0 + \theta_1 \cdot \phi(x)$$

$$\hat{Y} = X\theta$$

Linear Regression

Model:

$$\hat{y} = \theta_0 + \theta_1 \cdot \phi(x)$$

$$\hat{Y} = X\theta$$

Loss Function:

$$\ell(\theta) = \frac{1}{m} \sum_{i=0}^{m} (y_i - \hat{y}_i)^2$$

$$\ell(\theta) = \frac{1}{m} \sum (Y - X\theta)^2$$

Linear Regression

Model:

$$\hat{y} = \theta_0 + \theta_1 \cdot \phi(x)$$

$$\hat{Y} = X\theta$$

Loss Function:

$$\ell(\theta) = \frac{1}{m} \sum_{i=0}^{m} (y_i - \hat{y}_i)^2$$

$$\ell(\theta) = \frac{1}{m} \sum (Y - X\theta)^2$$

Optimize:

Closed Form:

$$\theta = (X^T X)^{-1} X^T Y$$

Linear Regression

Model:

$$\hat{y} = \theta_0 + \theta_1 \cdot \phi(x)$$

$$\hat{Y} = X\theta$$

Loss Function:

$$\ell(\theta) = \frac{1}{m} \sum_{i=0}^{m} (y_i - \hat{y}_i)^2$$

$$\ell(\theta) = \frac{1}{m} \sum (Y - X\theta)^2$$

Optimize:

Closed Form:

$$\theta = (X^T X)^{-1} X^T Y$$

Gradient Descent:

$$\frac{\partial \ell(\theta)}{\partial \theta_0} = \frac{2}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_i - y_i)$$

$$\frac{\partial \ell(\theta)}{\partial \theta_1} = \frac{2}{m} \sum_{i=1}^{m} x_i \cdot (\theta_0 + \theta_1 x_i - y_i)$$

Logistic Regression

Logistic Regression

Model:

$$\hat{y} = \sigma(\theta_0 + \theta_1 \cdot \phi(x))$$

Logistic Regression

Model:

$$\hat{y} = \sigma(\theta_0 + \theta_1 \cdot \phi(x))$$

Loss Function:

$$\ell(\theta) = -\sum_{i=1}^{m} y^{(i)} log(p_i) + (1 - y^{(i)}) log(1 - p_i)$$

Logistic Regression

Model:

$$\hat{y} = \sigma(\theta_0 + \theta_1 \cdot \phi(x))$$

Loss Function:

$$\ell(\theta) = -\sum_{i=1}^{m} y^{(i)} log(p_i) + (1 - y^{(i)}) log(1 - p_i)$$

Optimize:

Closed Form:

None - Cannot invert Sigmoid

Logistic Regression

Model:

$$\hat{y} = \sigma(\theta_0 + \theta_1 \cdot \phi(x))$$

Loss Function:

$$\ell(\theta) = -\sum_{i=1}^{m} y^{(i)} log(p_i) + (1 - y^{(i)}) log(1 - p_i)$$

Optimize:

Closed Form:

None - Cannot invert Sigmoid

Gradient Descent:

$$\frac{\partial \ell}{\partial \theta} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \cdot (\hat{y}^{(i)} - y^{(i)})$$

$$\nabla_\theta(\ell(\theta)) = \frac{1}{m} X^T(\hat{Y} - Y)$$

k-Nearest Neighbors

Model:

k-Nearest Neighbors

Model:
Non-parametric model

k-Nearest Neighbors

Model:
Non-parametric model

Loss Function:

No parameters to optimize

k-Nearest Neighbors

Model:
Non-parametric model

Loss Function:

No parameters to optimize

Optimize/Inference:

Closed Form:

Find k nearest neighbors

Majority voting

k-Nearest Neighbors

Model:

Non-parametric model

Loss Function:

No parameters to optimize

(learnable)

Optimize/Inference:

Closed Form:

Not a **learnable** parameter

Find k nearest neighbors

Majority voting

Linear Discriminant Analysis

Linear Discriminant Analysis

Model:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(Y = k)$$

Linear Discriminant Analysis

Model:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(Y = k)$$

Loss Function:

None

Has assumptions on data distributions instead

Linear Discriminant Analysis

Model:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(Y = k)$$

Loss Function:

None

Has assumptions on data distributions instead

Optimize/Inference:

Closed Form:

Estimate μ_k, $\mathbb{P}(Y = k)$ and Σ

Plug into model

LDA for Dimensionality Reduction

Model:

LDA for Dimensionality Reduction

Model:

$$S_W = \sum_{k=1}^{K} S_k = \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \mu_k)(x_i - \mu_k)^T$$

$$S_B = \sum_{k=1}^{K} N_k(\mu_k - \mu)(\mu_k - \mu)^T$$

LDA for Dimensionality Reduction

Model:

$$S_W = \sum_{k=1}^{K} S_k = \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \mu_k)(x_i - \mu_k)^T$$

$$S_B = \sum_{k=1}^{K} N_k(\mu_k - \mu)(\mu_k - \mu)^T$$

Loss Function:

Maximize $J(w) = \dfrac{w^T S_B w}{w^T S_W w}$

LDA for Dimensionality Reduction

Model:

$$S_W = \sum_{k=1}^{K} S_k = \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \mu_k)(x_i - \mu_k)^T$$

$$S_B = \sum_{k=1}^{K} N_k(\mu_k - \mu)(\mu_k - \mu)^T$$

Loss Function:

Maximize $J(w) = \dfrac{w^T S_B w}{w^T S_W w}$

Optimize/Inference:

Closed Form:

$$S_W^{-1} S_B \cdot w = \lambda w$$

Models Seen So Far

Supervised	Unsupervised
Linear Regression	PCA
Logistic Regression	k-Means Clustering
k-NN	
LDA	

Gradient Descent

$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

α : Learning Rate

Gradient Descent

$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

If α is too large?

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

If α is too small?

α : Learning Rate

Gradient Descent

When to stop?

Fixed Iteration	Gradient Norm	Change in Loss	Change in θ	Validation

Gradient Descent

Practical Issues

Feature Scaling / Pre-processing

Small Gradients / Plateau Regions

Adaptive Step Sizes

Gradient Descent

Batch Sizes

Batch GD

Mini Batch GD

Stochastic GD

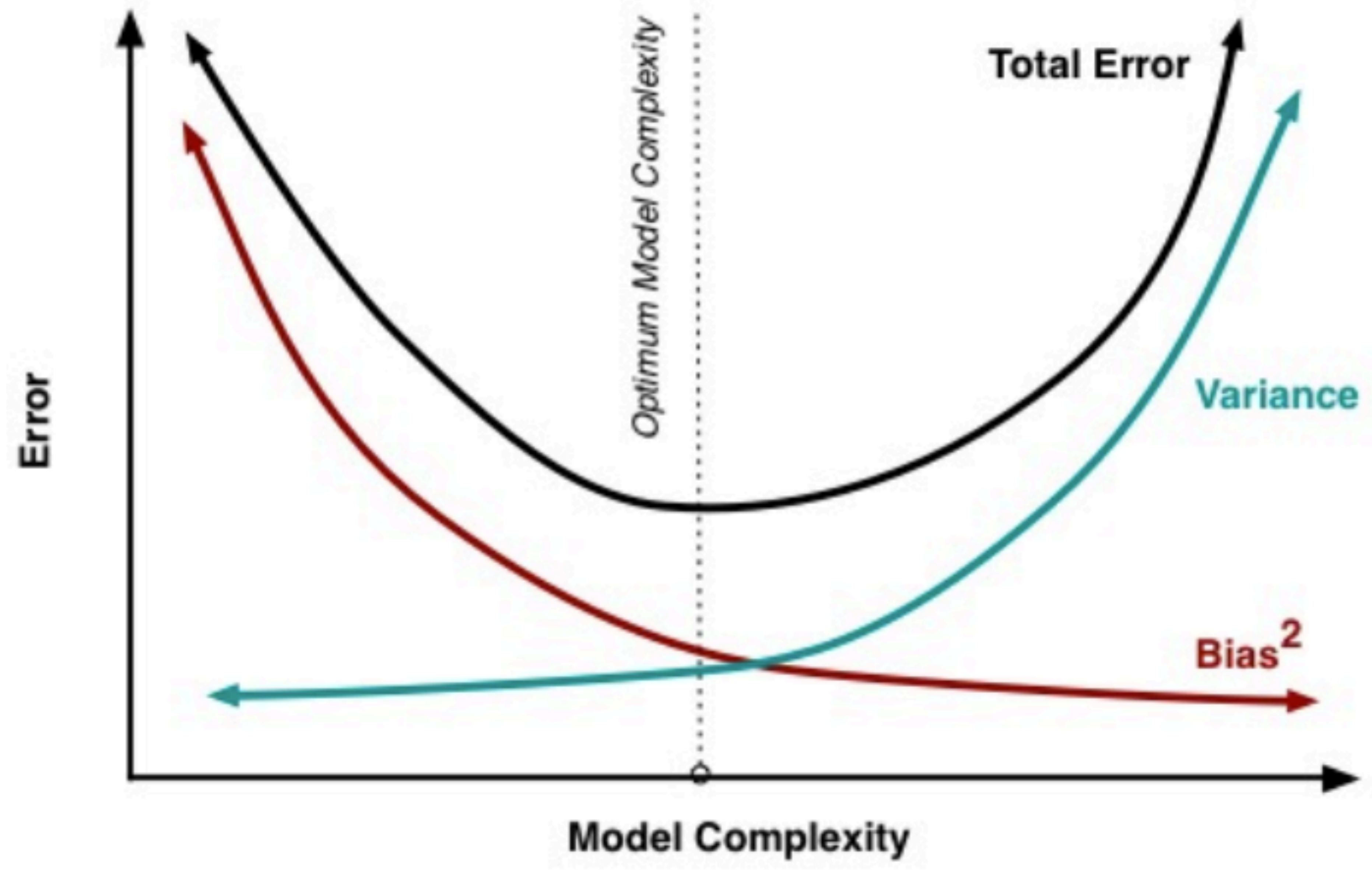
Practical Issues in ML

Overfitting

Underfitting

Practical Issues in ML

Model Complexity	Bias	Variance	Train Error	Test Error
Too Simple	High	Low	High	High
Sweet Spot	Medium	Medium	Medium	Medium
Too Complex	Low	High	Low	High



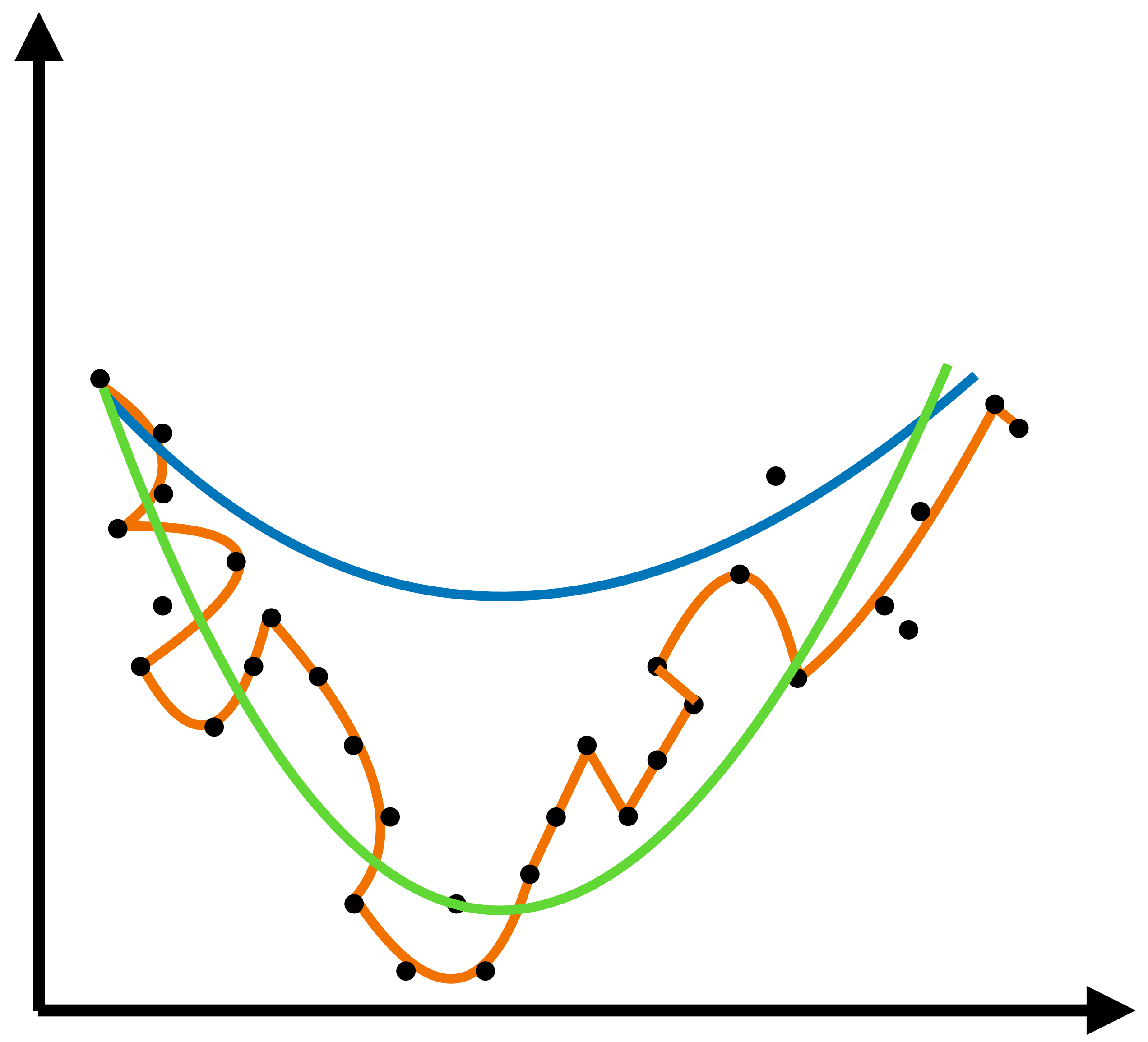
Practical Issues

Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m} \sum (Y - X\theta)^2$$

$$L(\theta) = \frac{1}{m} \sum (Y - X\theta)^2 + \lambda ||\theta||^2$$

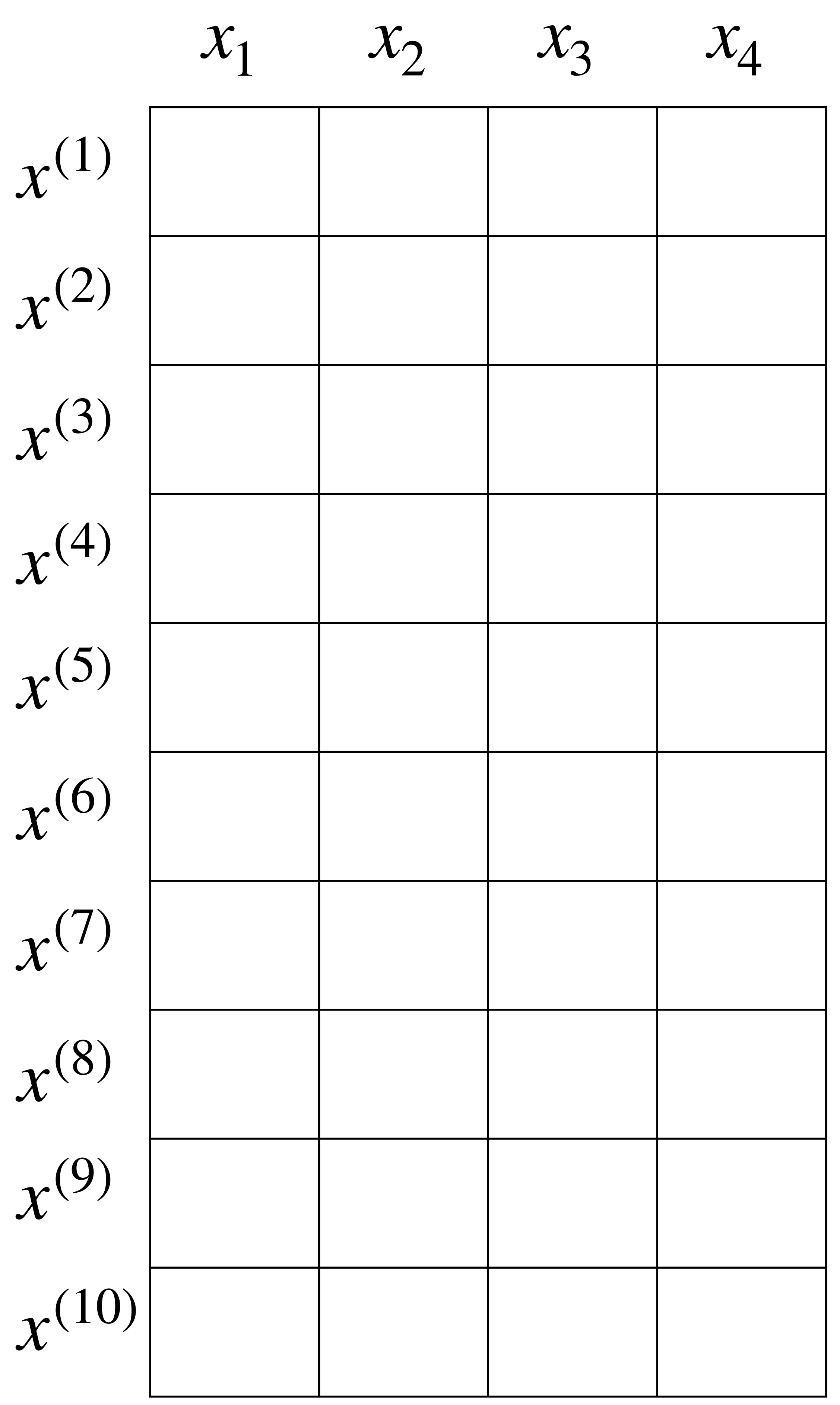


Practical Issues

k-Fold Cross Validation

k-Fold Cross Validation

Algorithm



Algorithm

1. Shuffle the dataset randomly
2. Split data into k equally-sized folds (or partitions)
3. for each fold $i = 1,\ 2,\ \dots,\ k$:
    - 3a. Use fold i as the validation set
    - 3b. Use the remaining $k - i$ folds as the training set
    - 3c. Train the model on the training set
    - 3d. Evaluate on the validation set, record performance metric
4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm

	x_1	x_2	x_3	x_4
$x^{(1)}$				
$x^{(2)}$				
$x^{(3)}$				
$x^{(4)}$				
$x^{(5)}$				
$x^{(6)}$				
$x^{(7)}$				
$x^{(8)}$				
$x^{(9)}$				
$x^{(10)}$				

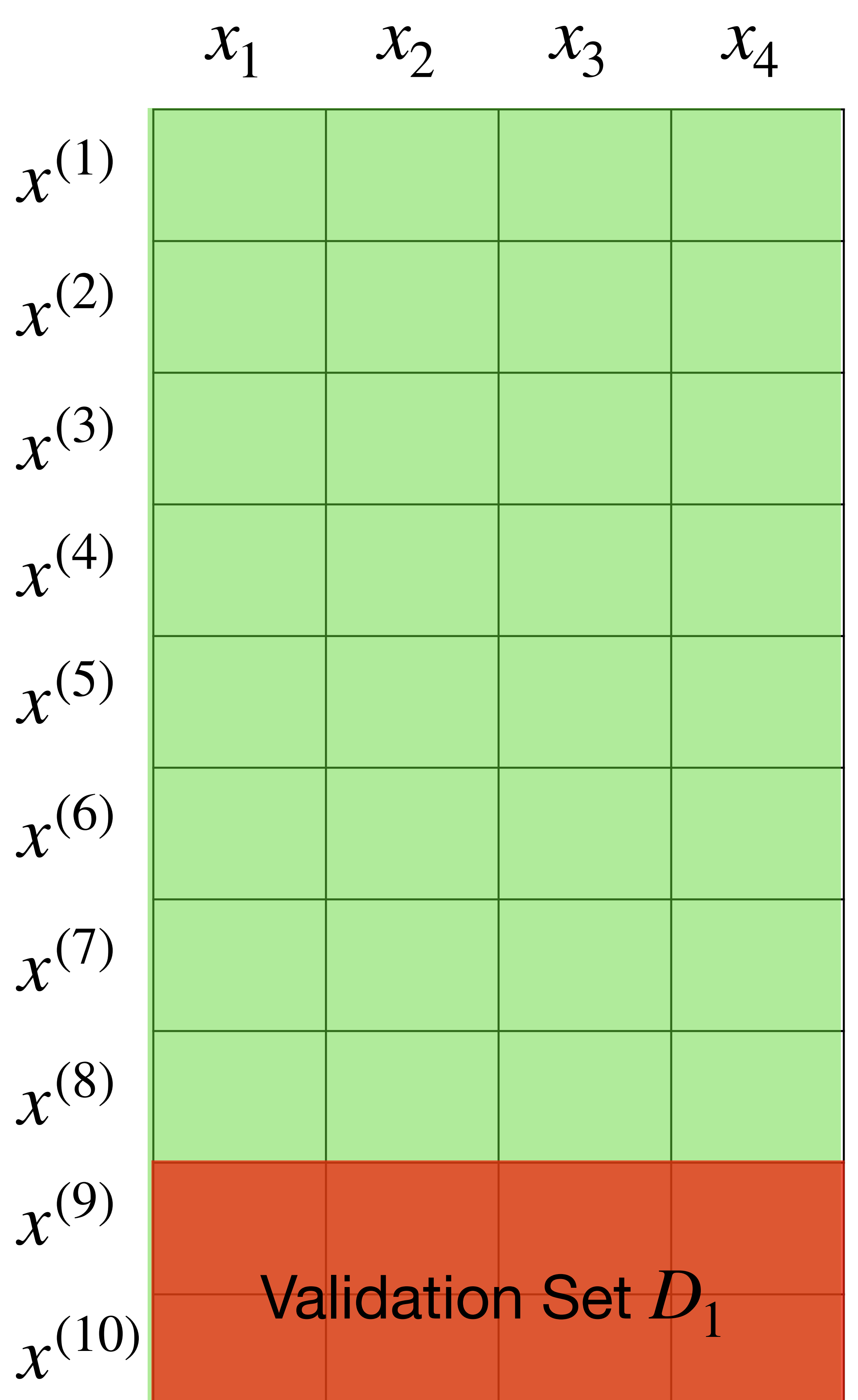
Let's say we want to run $k = 5$-fold cross validation

Algorithm

1. Shuffle the dataset randomly
2. Split data into k equally-sized folds (or partitions)
3. for each fold $i = 1, 2, \ldots, k$:
    - 3a. Use fold i as the validation set
    - 3b. Use the remaining $k - i$ folds as the training set
    - 3c. Train the model on the training set
    - 3d. Evaluate on the validation set, record performance metric
4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$CV_1 = \frac{1}{D_1} \sum_{D_1} \ell(y_{D_1}, f_\theta(D_1))$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

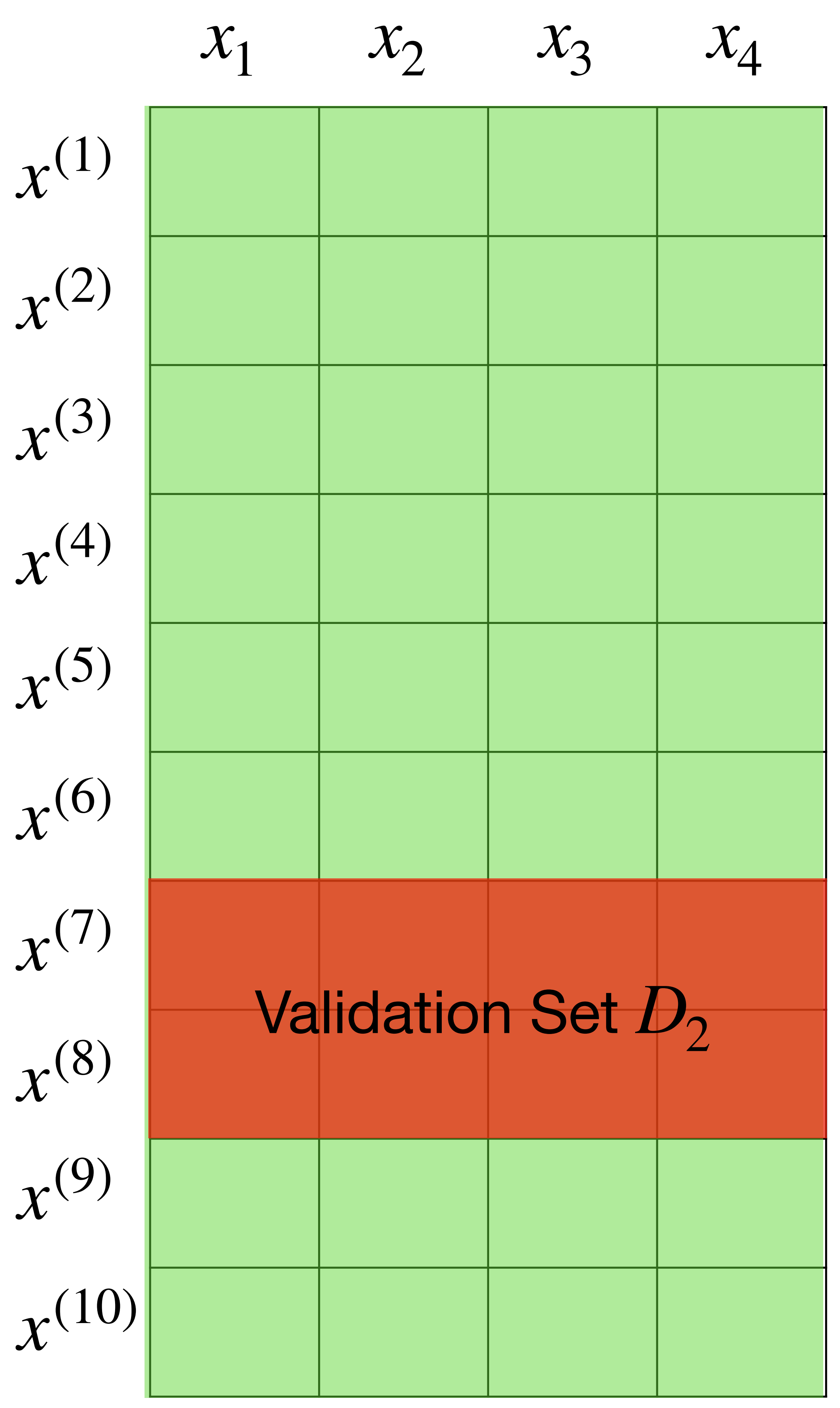
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$CV_2 = \frac{1}{D_2} \sum_{D_2} \ell(y_{D_2}, f_\theta(D_2))$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

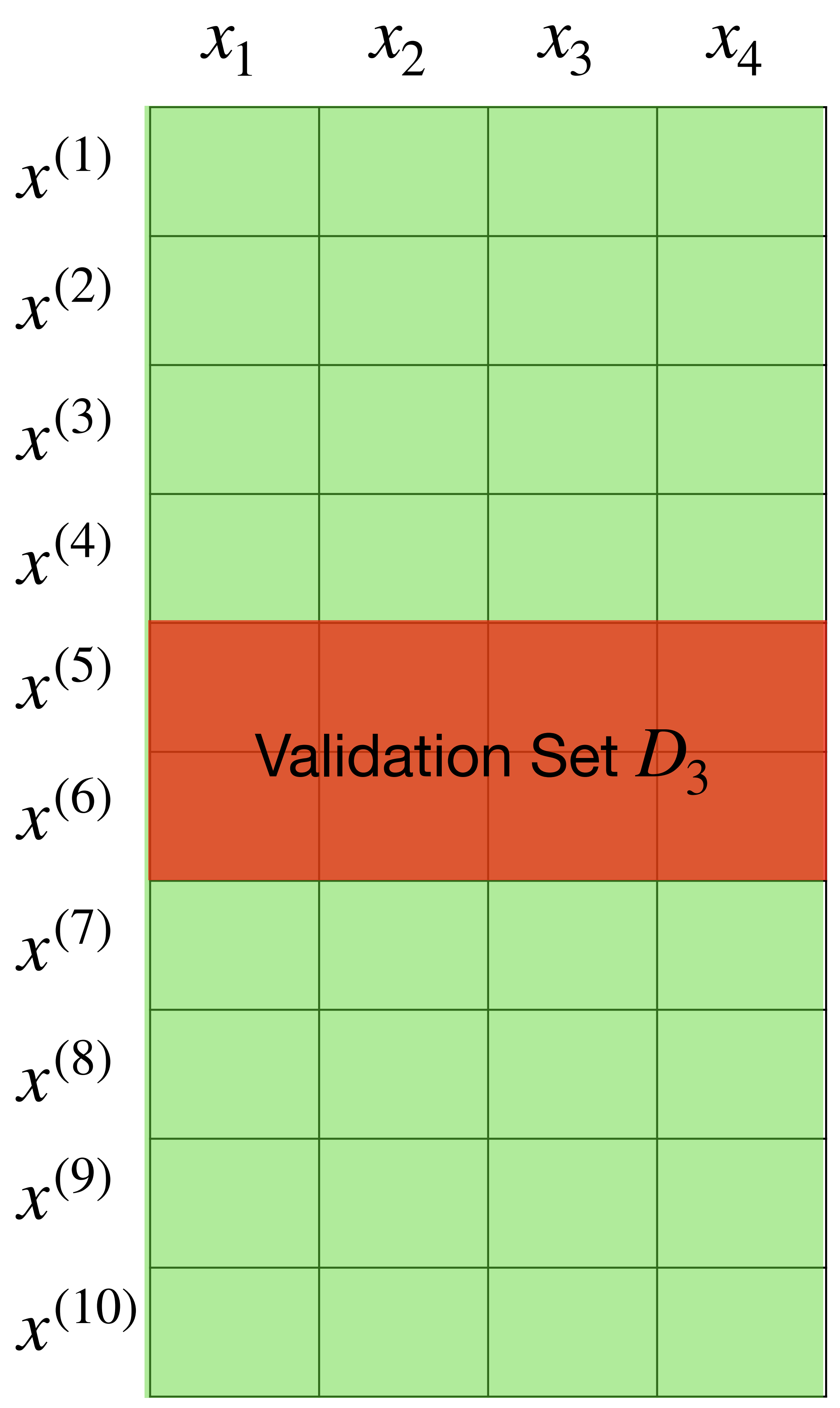
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$CV_3 = \frac{1}{D_3} \sum_{D_3} \ell(y_{D_3}, f_\theta(D_3))$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

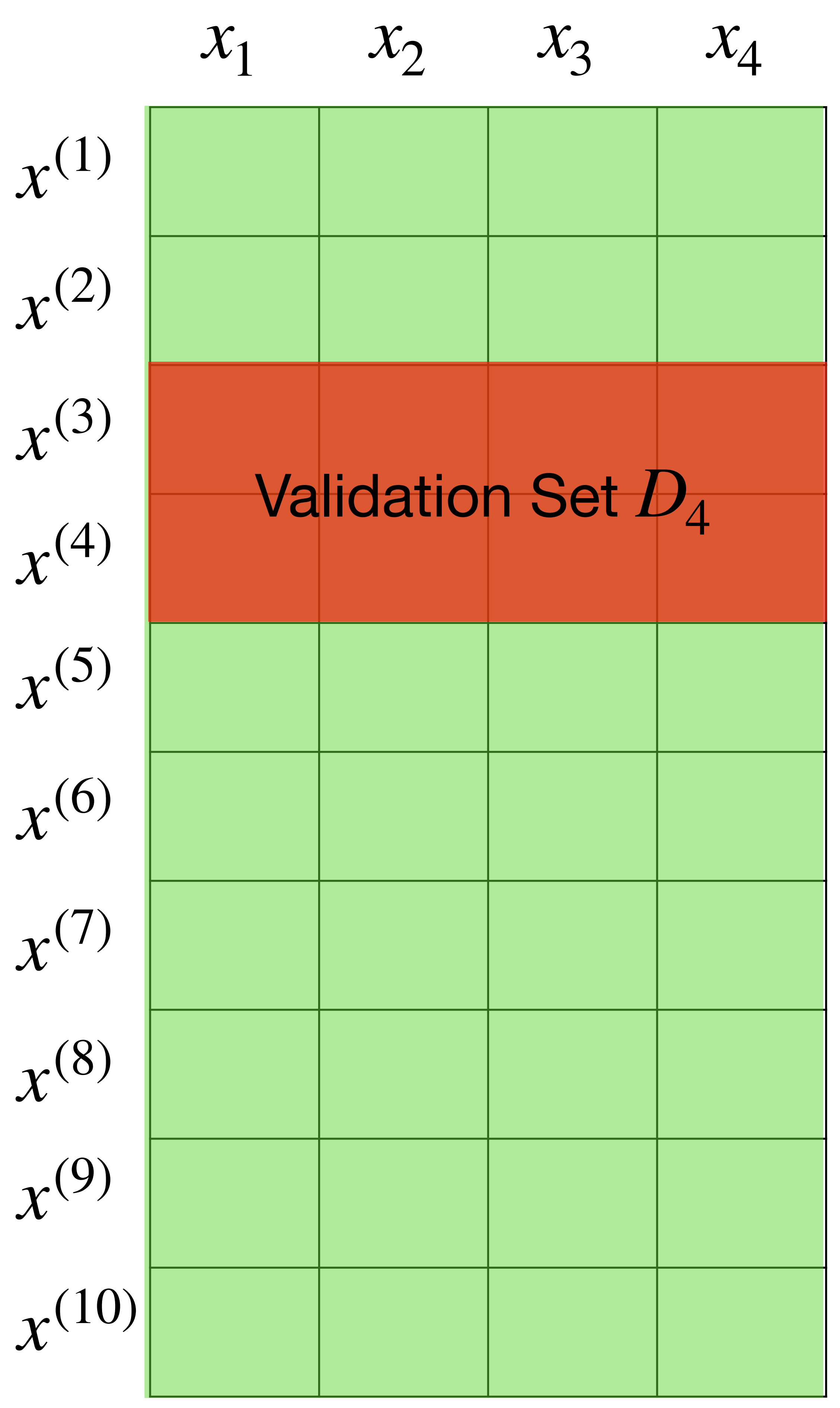
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$CV_4 = \frac{1}{D_4} \sum_{D_4} \ell(y_{D_4}, f_\theta(D_4))$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

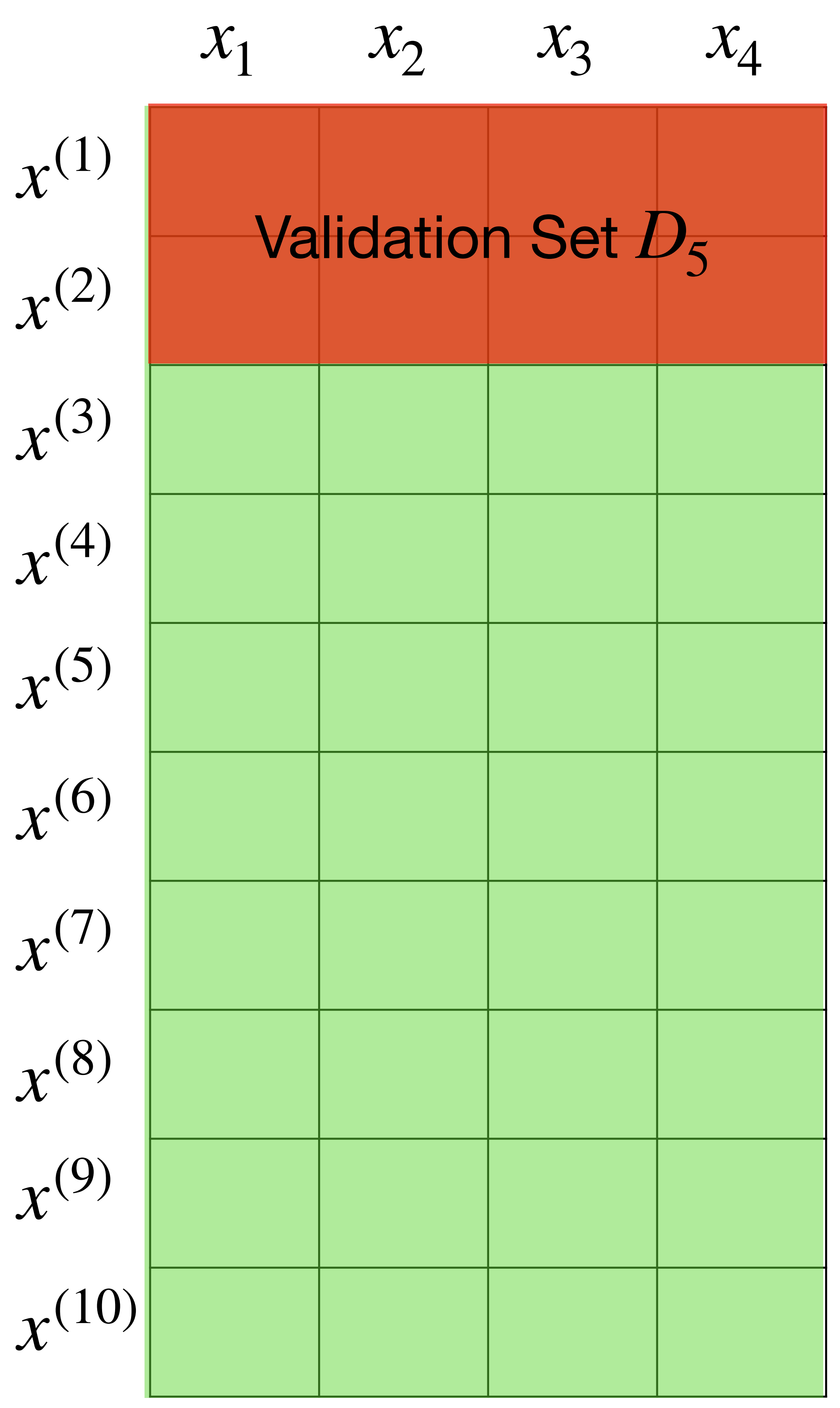
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

$$CV_5 = \frac{1}{D_5} \sum_{D_5} \ell(y_{D_5}, f_\theta(D_5))$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

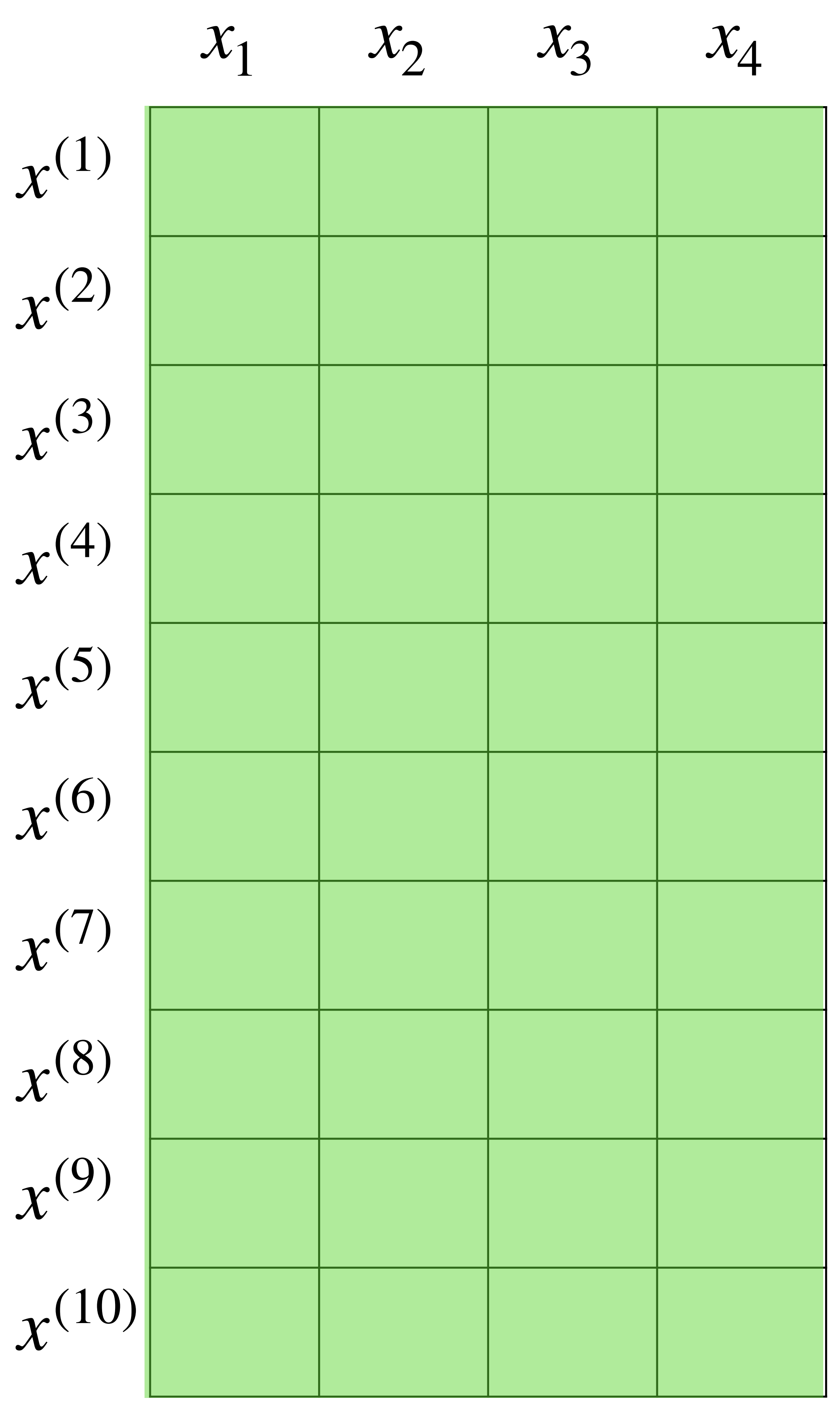
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let’s say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

Mean CV Score:

$$\bar{CV} = \frac{1}{k}\sum_{i=1}^{k} CV_i$$

Algorithm

1. Shuffle the dataset randomly

2. Split data into k equally-sized folds (or partitions)

3. for each fold $i = 1, 2, \ldots, k$:

    3a. Use fold i as the validation set

    3b. Use the remaining $k - i$ folds as the training set

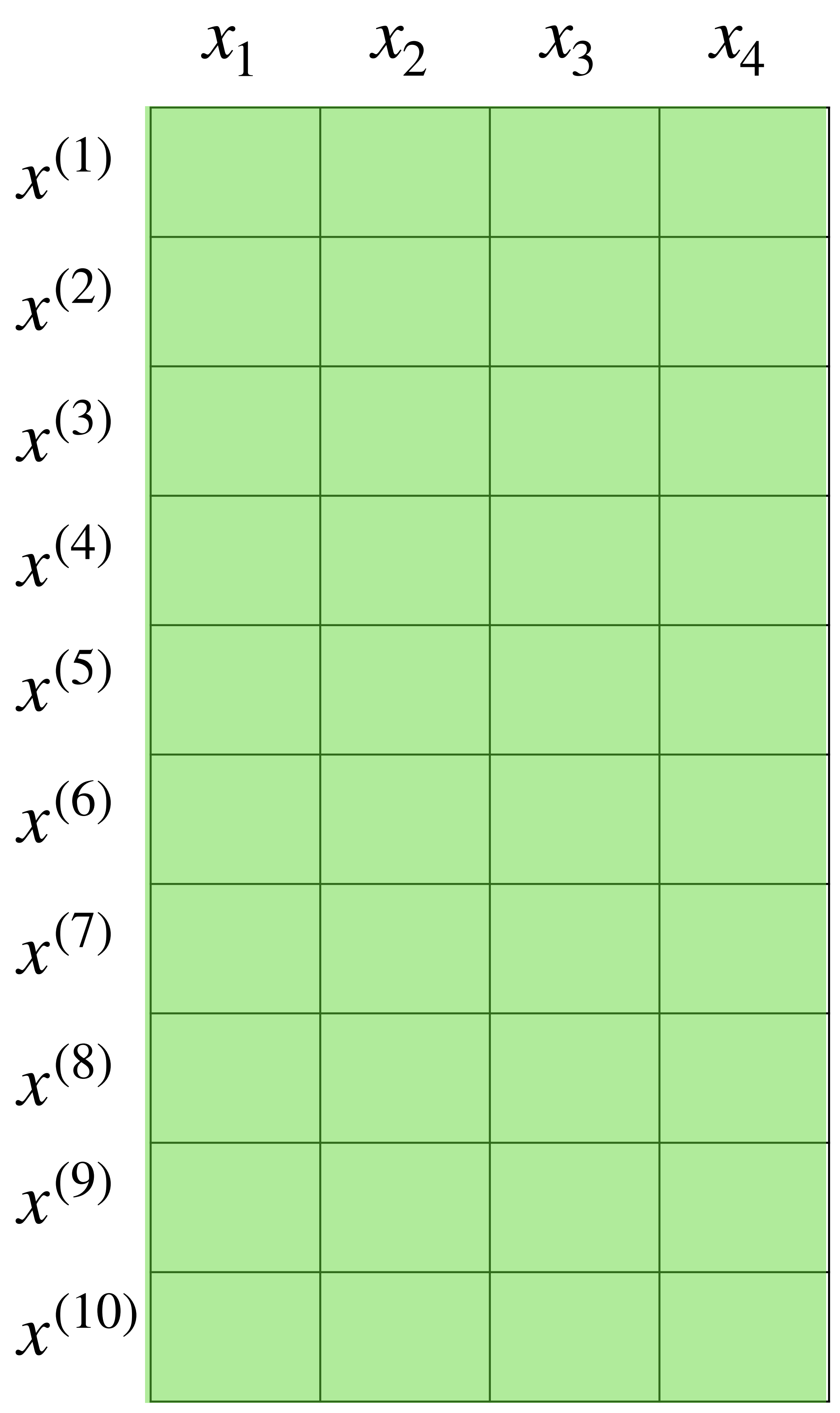
    3c. Train the model on the training set

    3d. Evaluate on the validation set, record performance metric

4. Aggregate the K performance estimates

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

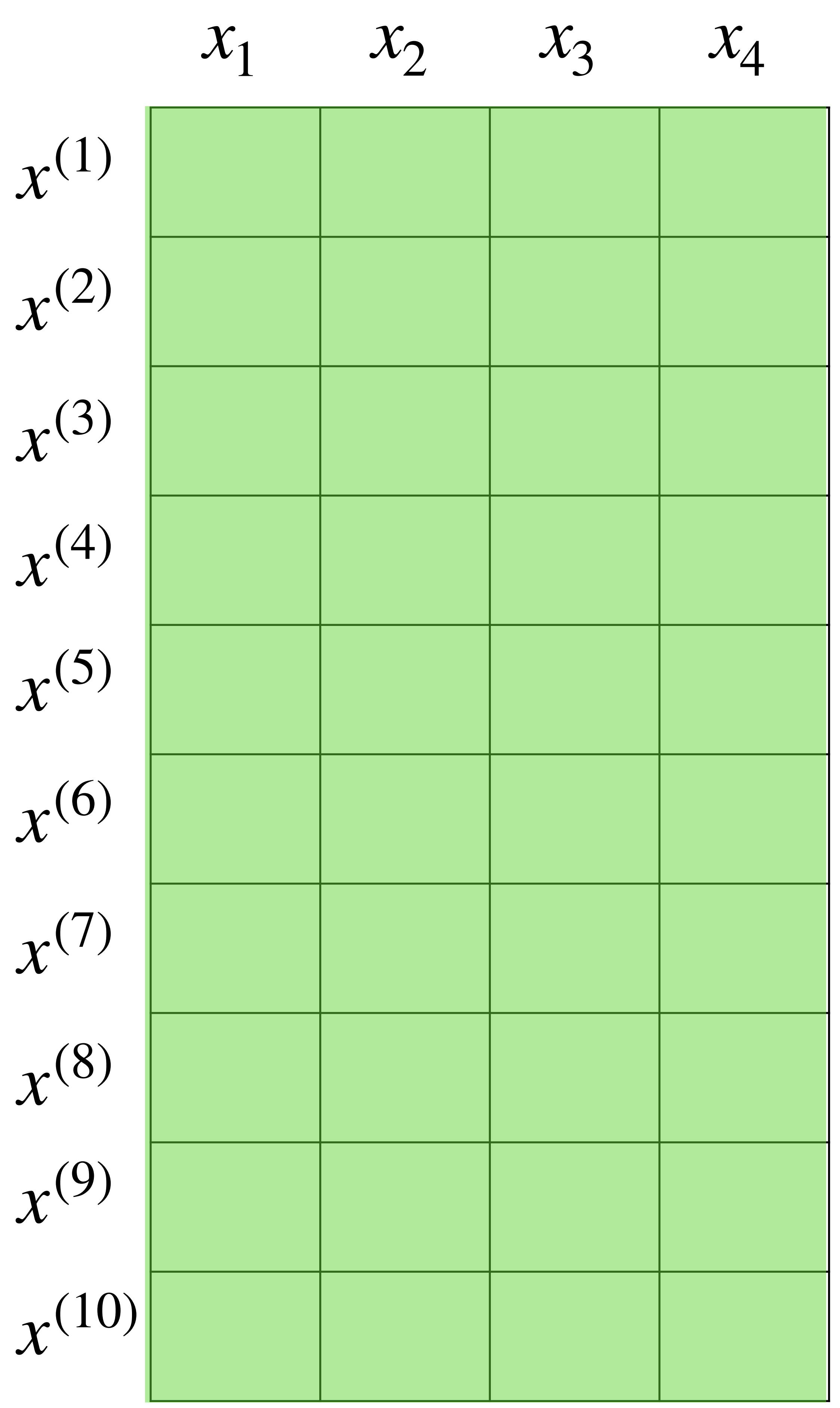
Mean CV Score:

$$\bar{CV} = \frac{1}{k}\sum_{i=1}^{k} CV_i$$

k-value	**Training Size**	**Properties**
k=2	50%	High Bias Low Variance Fast
k=5	80%	Good Balance Commonly Used
k=10	90%	Low Bias Commonly Used
k=m-1	m-1 samples	Low Bias Highest Variance Slow

k-Fold Cross Validation

Algorithm



Let's say we want to run $k = 5$-fold cross validation

Train on **8 rows**, test on **2 row**

Mean CV Score:

$$\bar{CV} = \frac{1}{k} \sum_{i=1}^{k} CV_i$$

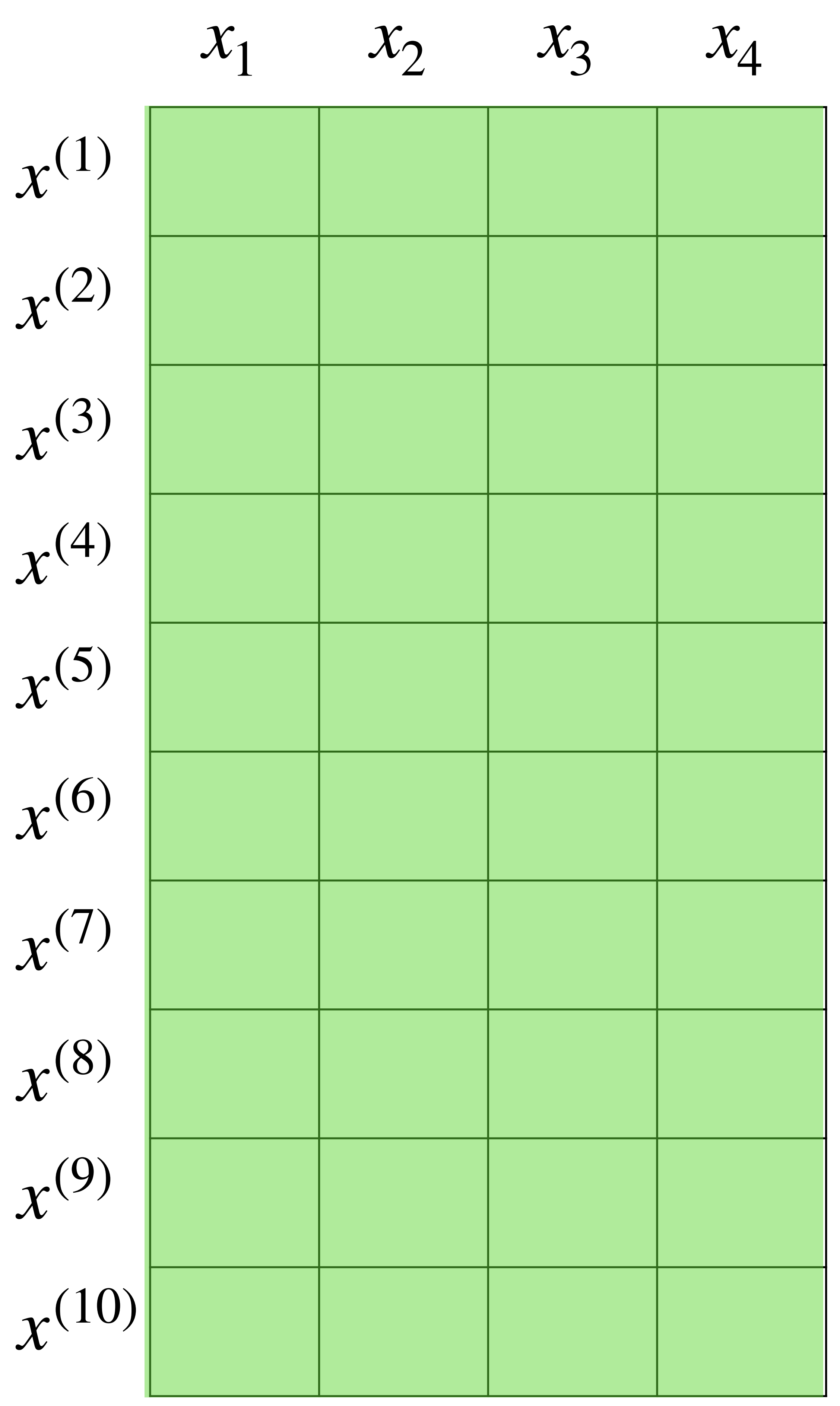
k-fold CV requires **training k models**.

If training is expensive, smaller k is preferred.

k-value	**Training Size**	**Properties**
k=2	50%	High Bias Low Variance Fast
k=5	80%	Good Balance Commonly Used
k=10	90%	Low Bias Commonly Used
k=m-1	m-1 samples	Low Bias Highest Variance Slow

k-Fold Cross Validation

Variants

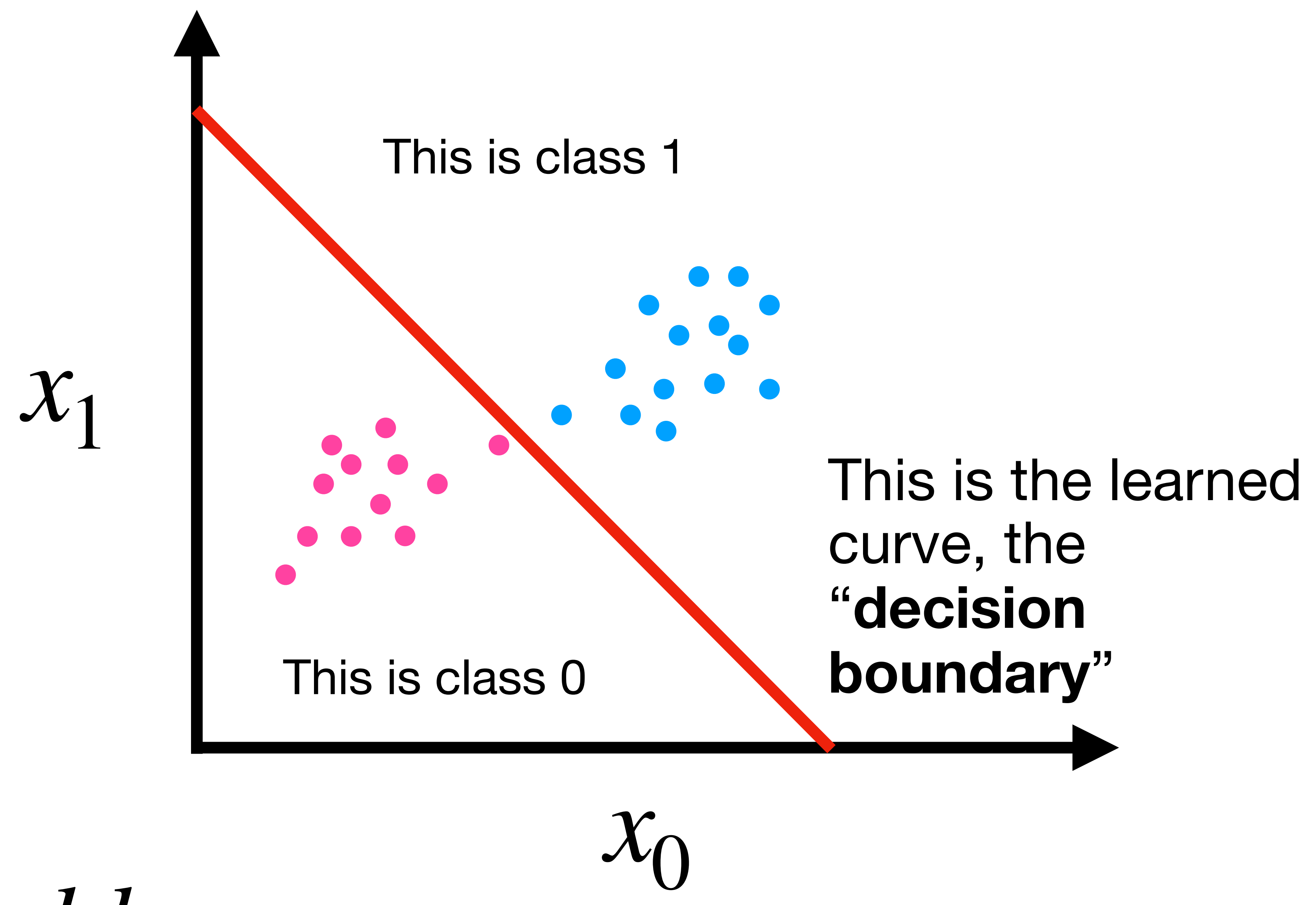


Stratified Cross-Validation

- The Problem with Random Splits
  - For imbalanced classification, random splits may create folds with different class distributions.
  - **One fold might have 40% positives while another has 20%, leading to unreliable estimates.**
  - Stratified sampling ensures each fold has approximately the same class distribution as the full dataset.
  - Algorithm:
    - Separate samples by class
    - For each class, distribute samples evenly across $k-$folds
    - Combine to form final folds

Classification

- Your classifier will output a probability value between 0 and 1
- Example:
  - $\mathbb{P}(cat \mid image_1) = 0.61$
  - $\mathbb{P}(cat \mid image_2) = 0.52$
- Practitioner needs to also set a **threshold**
  - $image_i$ is a cat if $\mathbb{P}(cat \mid image_i) \geq Threshold$



Classification Metrics

	Predicted Positive	Predicted Negative
Actual Positive		
Actual Negative		

Classification Metrics

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{\boxed{TP + FP}}$$

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{\boxed{TP + FN}}$$

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

Of all instances predicted as positive, what fraction actually are positive? Precision measures the **reliability of positive predictions**. High precision means **few false alarms.**

$$\text{Recall} = \frac{TP}{TP + FN}$$

When to care about precision?
When false positives are costly.

Examples include spam filtering (users hate losing important emails), recommendation systems (irrelevant recommendations erode trust), and legal contexts (wrongful accusations).

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision and Recall

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

Of all actual positive instances, **what fraction did we correctly identify**? Recall measures coverage of positive instances. High recall means **few missed positives**.

When to care about recall?
When false negatives are costly.

Examples include disease screening (missing a diagnosis can be fatal), security threats (missing an attack is catastrophic), and search engines (users want all relevant results).

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision vs Recall Tradeoff - F1 Score

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision and recall are **inherently in tension**.

Increasing the threshold for positive classification typically **increases precision but decreases recall**.

Decreasing the threshold has the opposite effect.

The optimal balance depends on the application's cost structure.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision vs Recall Tradeoff - F1 Score

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision vs Recall Tradeoff - F1 Score

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{False Positive Rate} = \frac{FP}{TN + FP}$$

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision vs Recall Tradeoff - F1 Score

Question:

How is this a tradeoff?
How would you increase/decrease the true positives?

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Metrics

Precision vs Recall Tradeoff - F1 Score

Question:

How is this a tradeoff?
How would you increase/decrease the true positives?

Answer: By changing the threshold

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

Metrics

Precision vs Recall Tradeoff - F1 Score

Question:

How is this a tradeoff?
How would you increase/decrease the true positives?

Answer: By changing the threshold

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- Cat in image if $\mathbb{P}(cat \mid image_i) \geq 0$
  - Precision goes 🟥, Recall goes 🟥

Metrics

Precision vs Recall Tradeoff - F1 Score

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

Question:

How is this a tradeoff?
How would you increase/decrease the true positives?

Answer: By changing the threshold

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- Cat in image if $\mathbb{P}(cat \,|\, image_i) \geq 0$
  - Precision goes down, Recall goes up

Metrics

Precision vs Recall Tradeoff - F1 Score

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

Question:

How is this a tradeoff?
How would you increase/decrease the true positives?

Answer: By changing the threshold

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- Cat in image if $\mathbb{P}(cat \mid image_i) \geq 0.999$
  - Precision goes 🟥 Recall goes 🟥

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

Metrics

Precision vs Recall Tradeoff - F1 Score

Question:

How is this a tradeoff?
How would you increase/decrease the true positives?

Answer: By changing the threshold

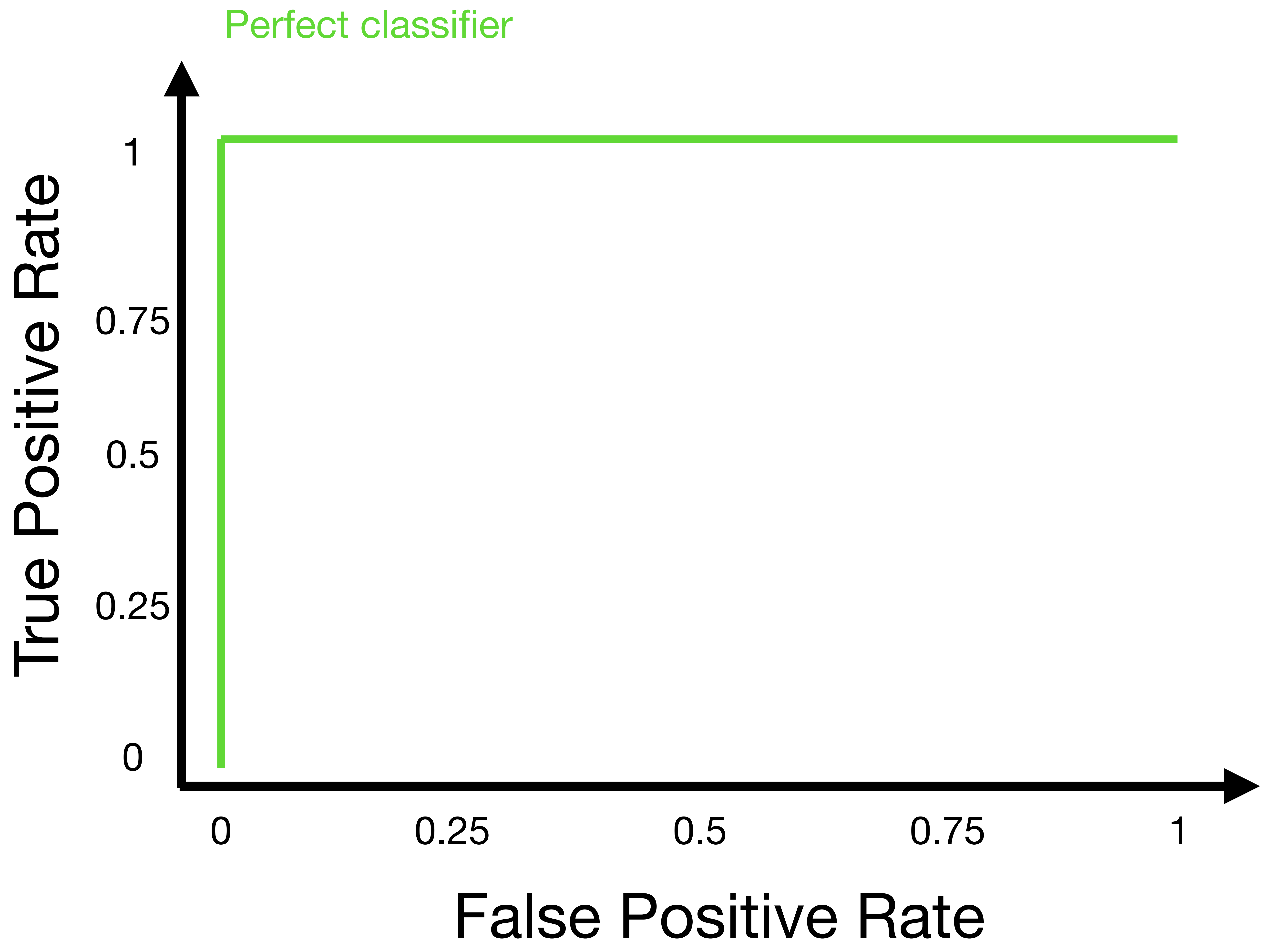
	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- Cat in image if $\mathbb{P}(cat \,|\, image_i) \geq 0.999$
    - Precision goes up, Recall goes down

Metrics

AUC-ROC Curve

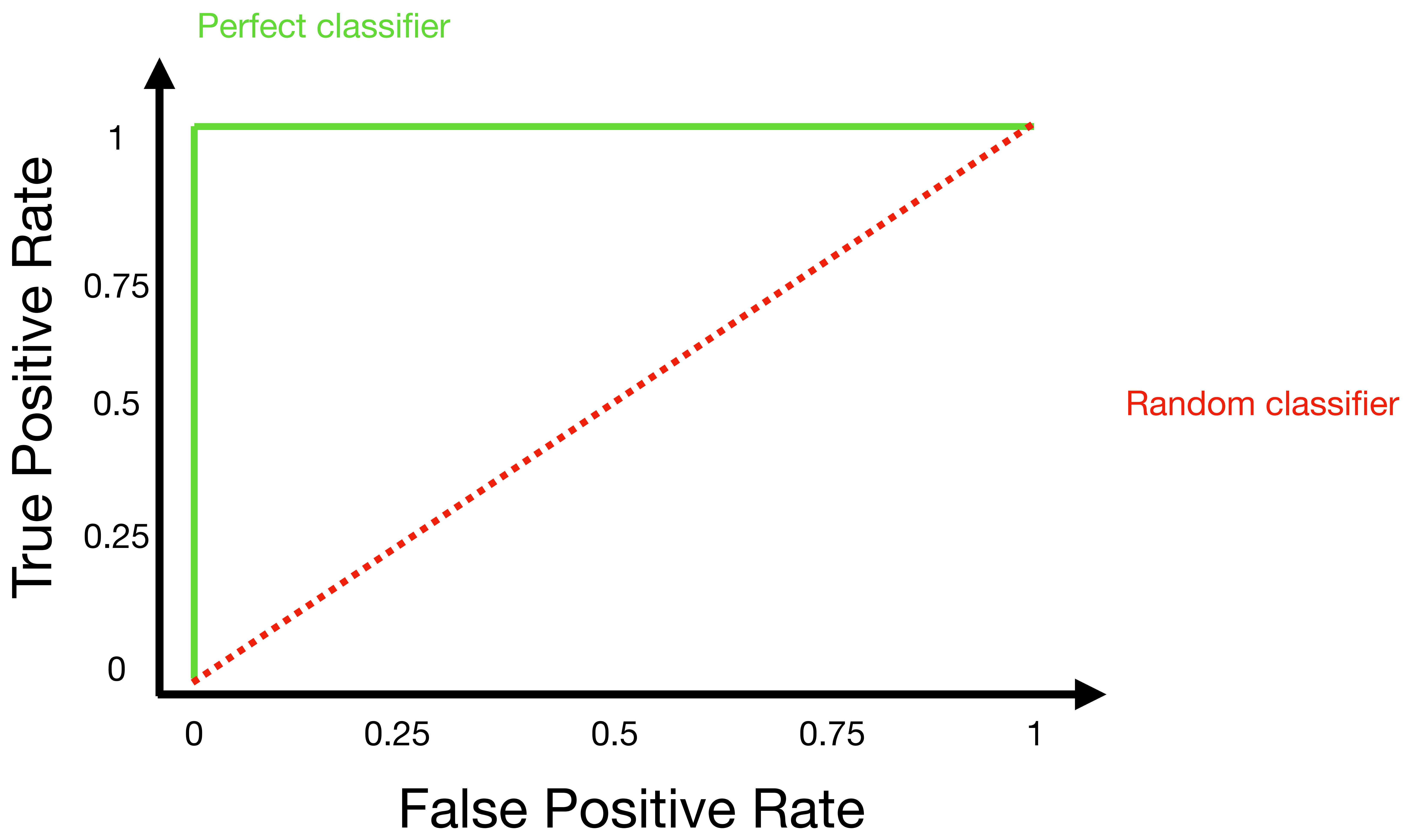
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics
AUC-ROC Curve

$$\text{TPR} = \frac{TP}{TP + FN}$$

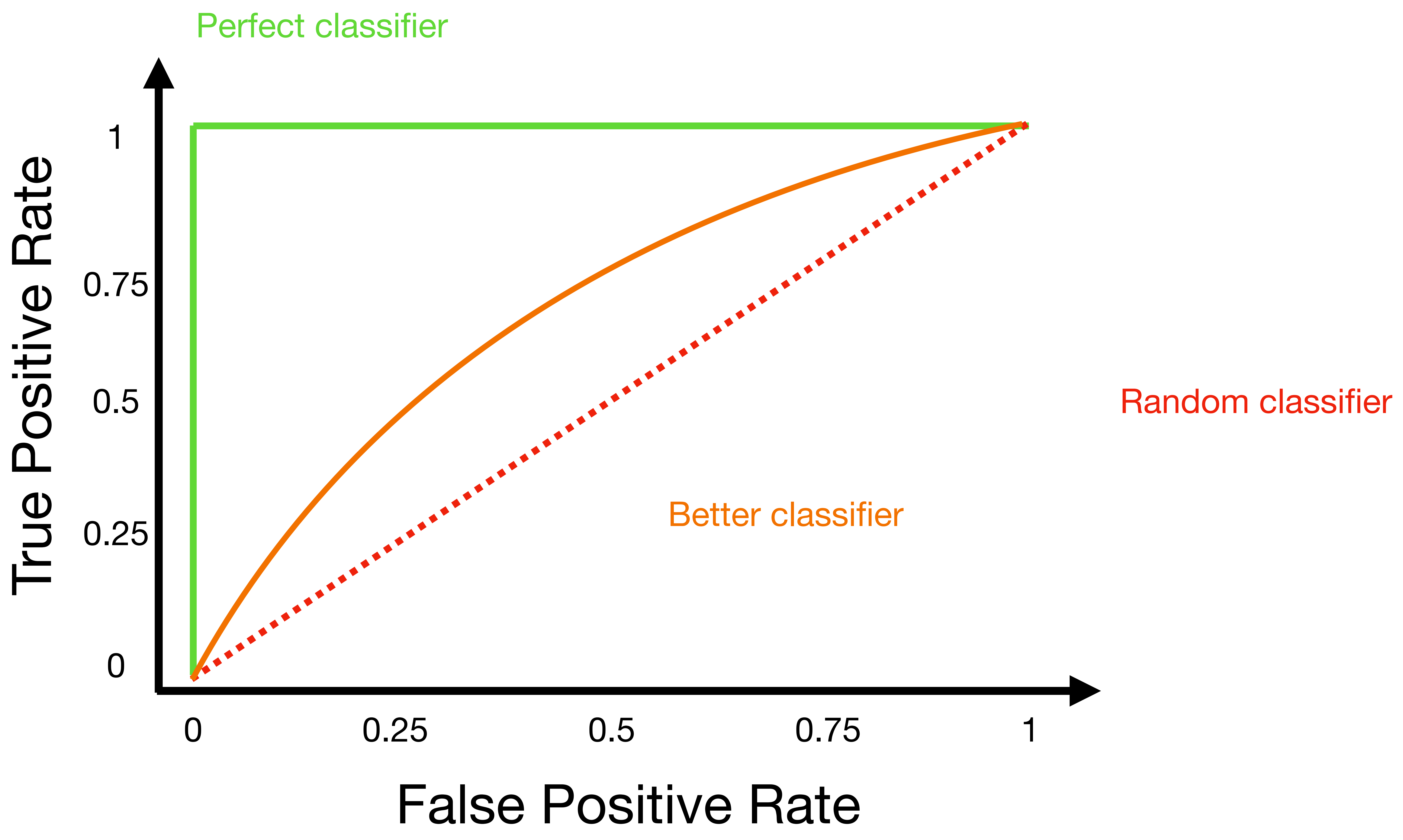
$$\text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

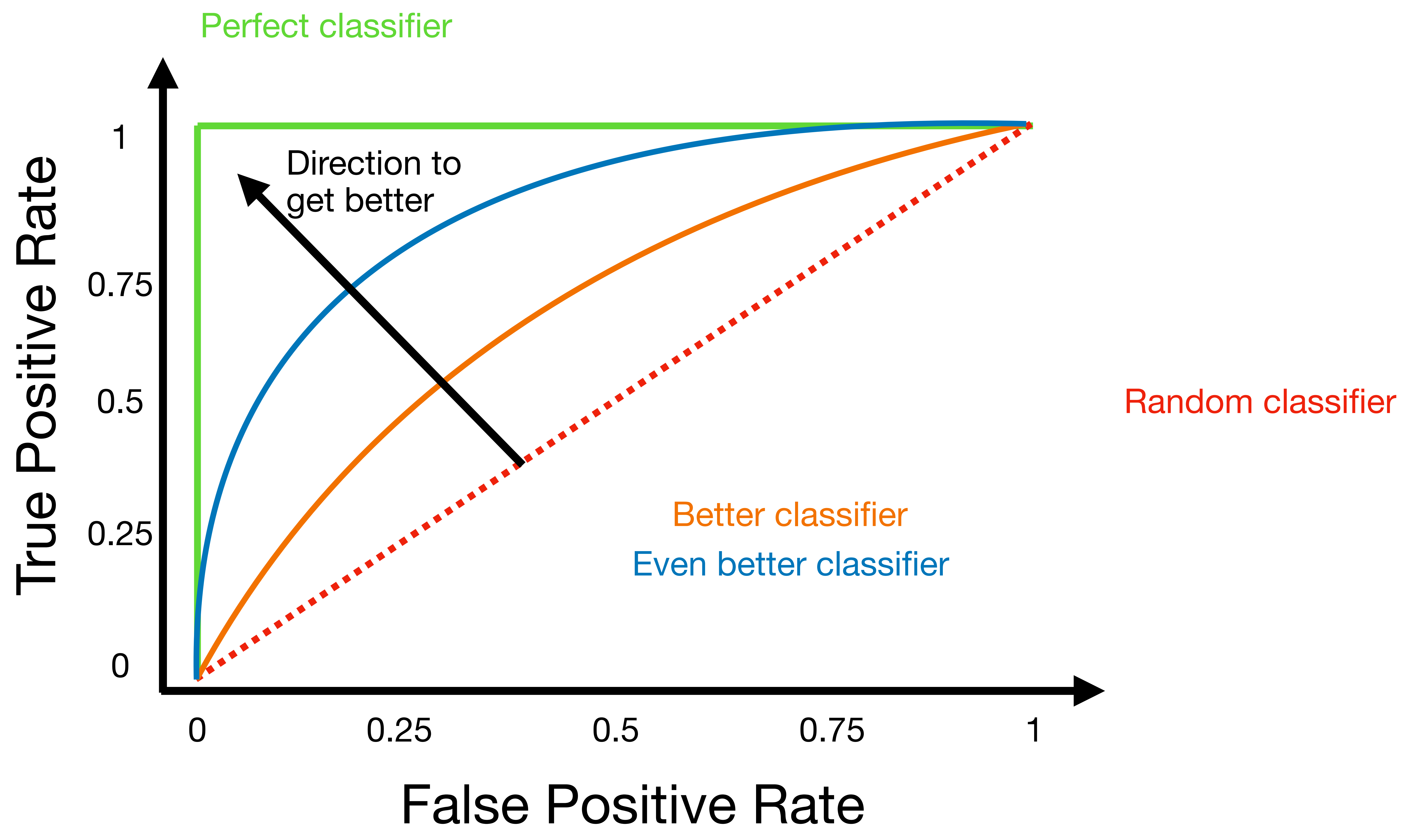
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

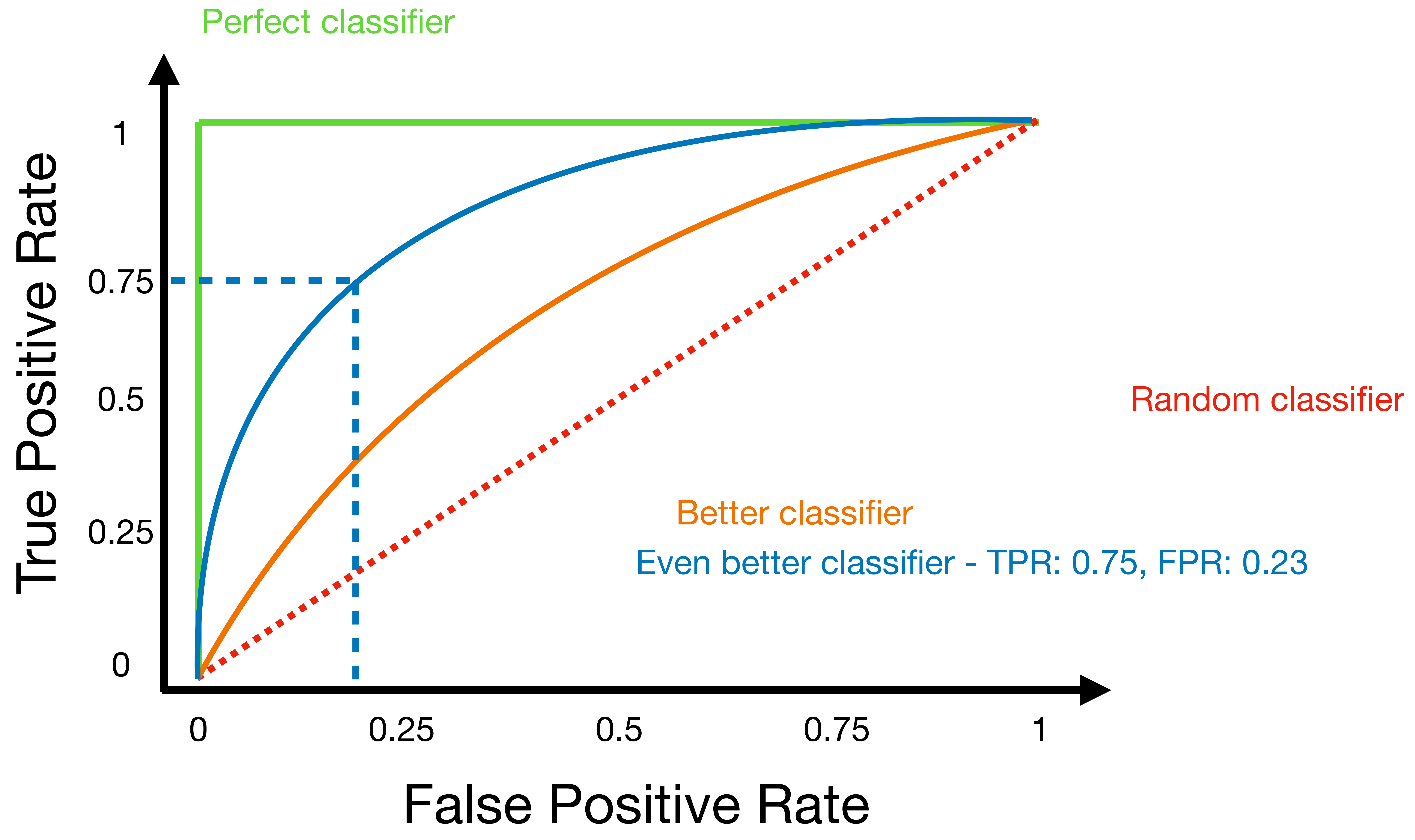
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

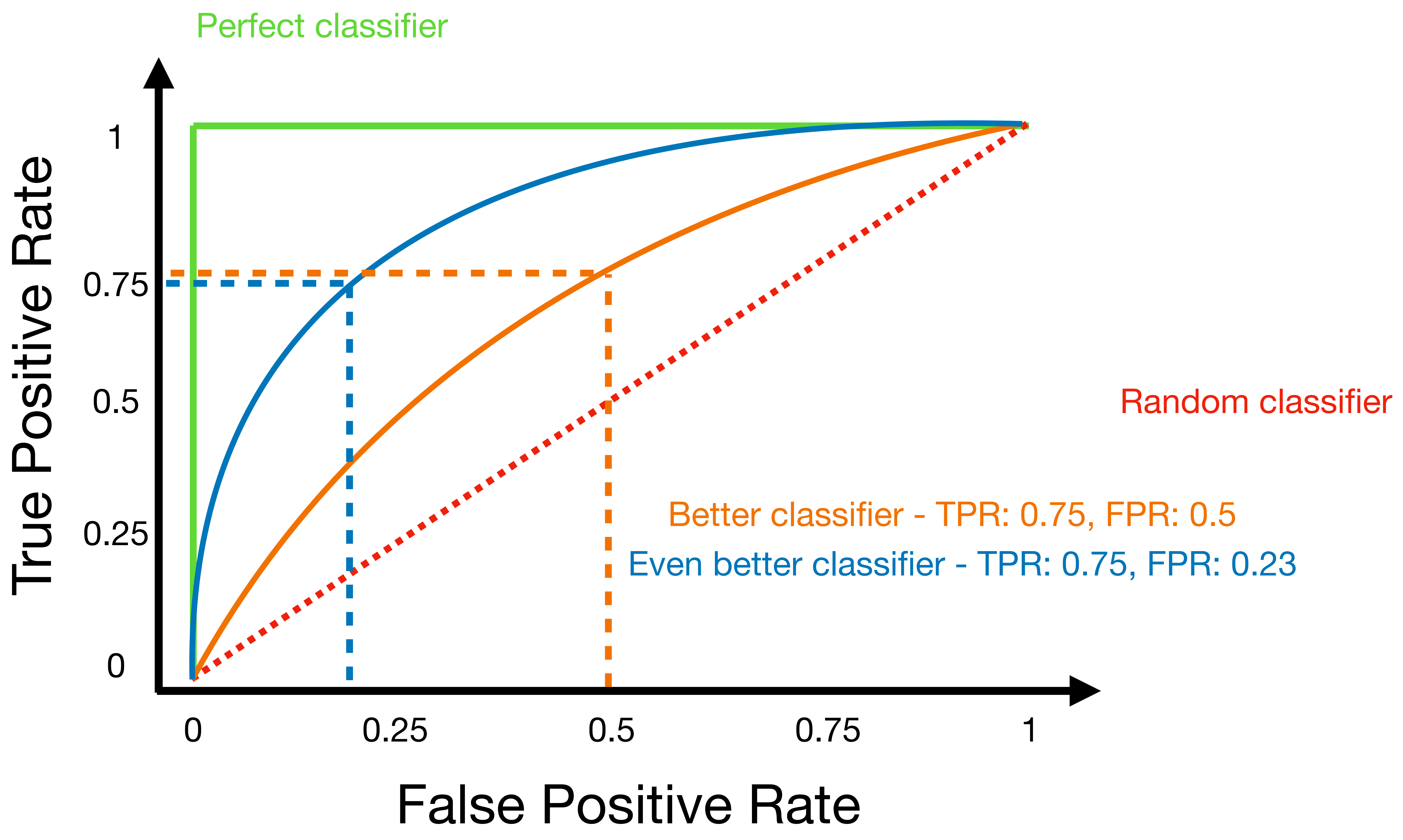
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

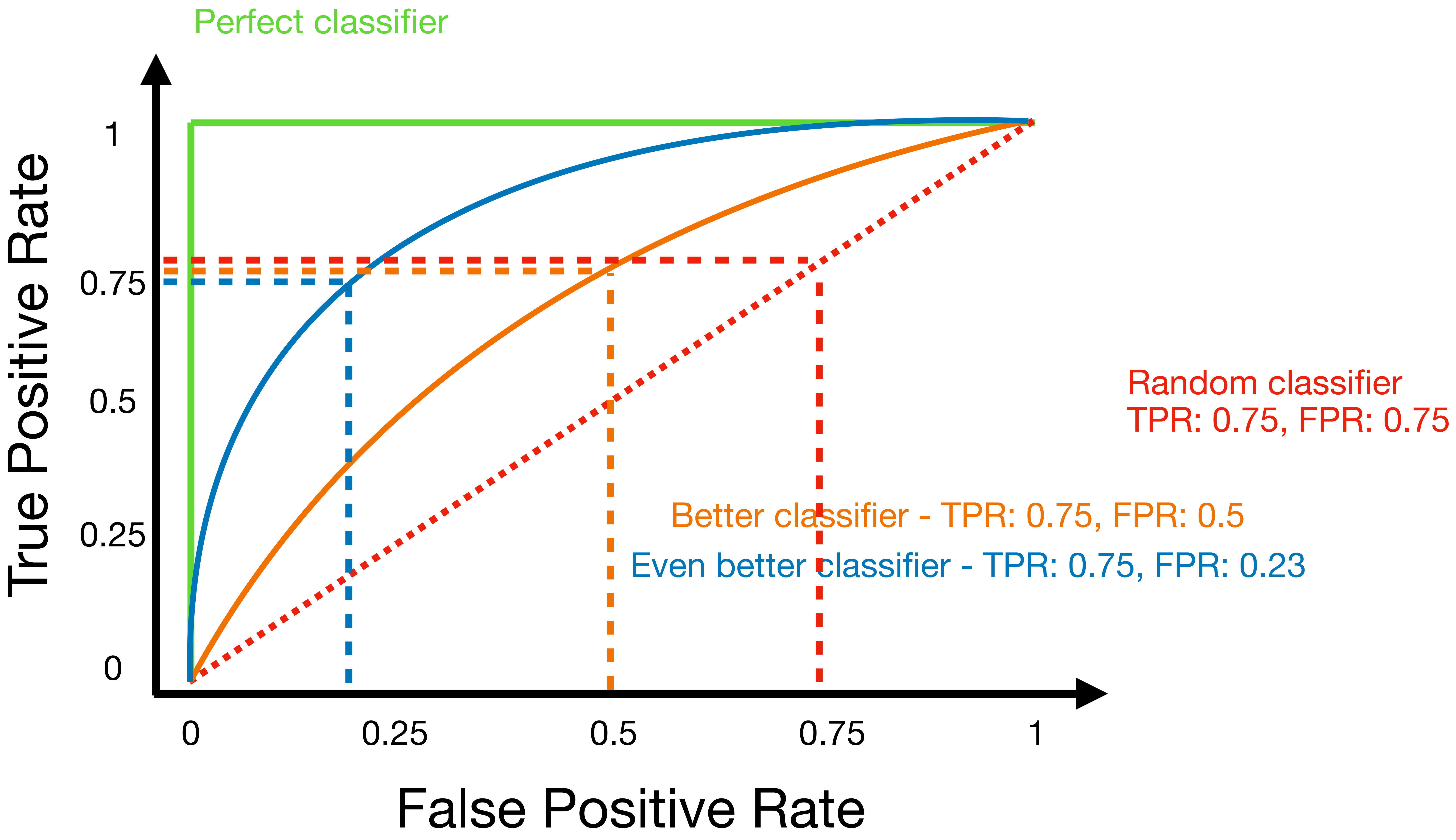
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

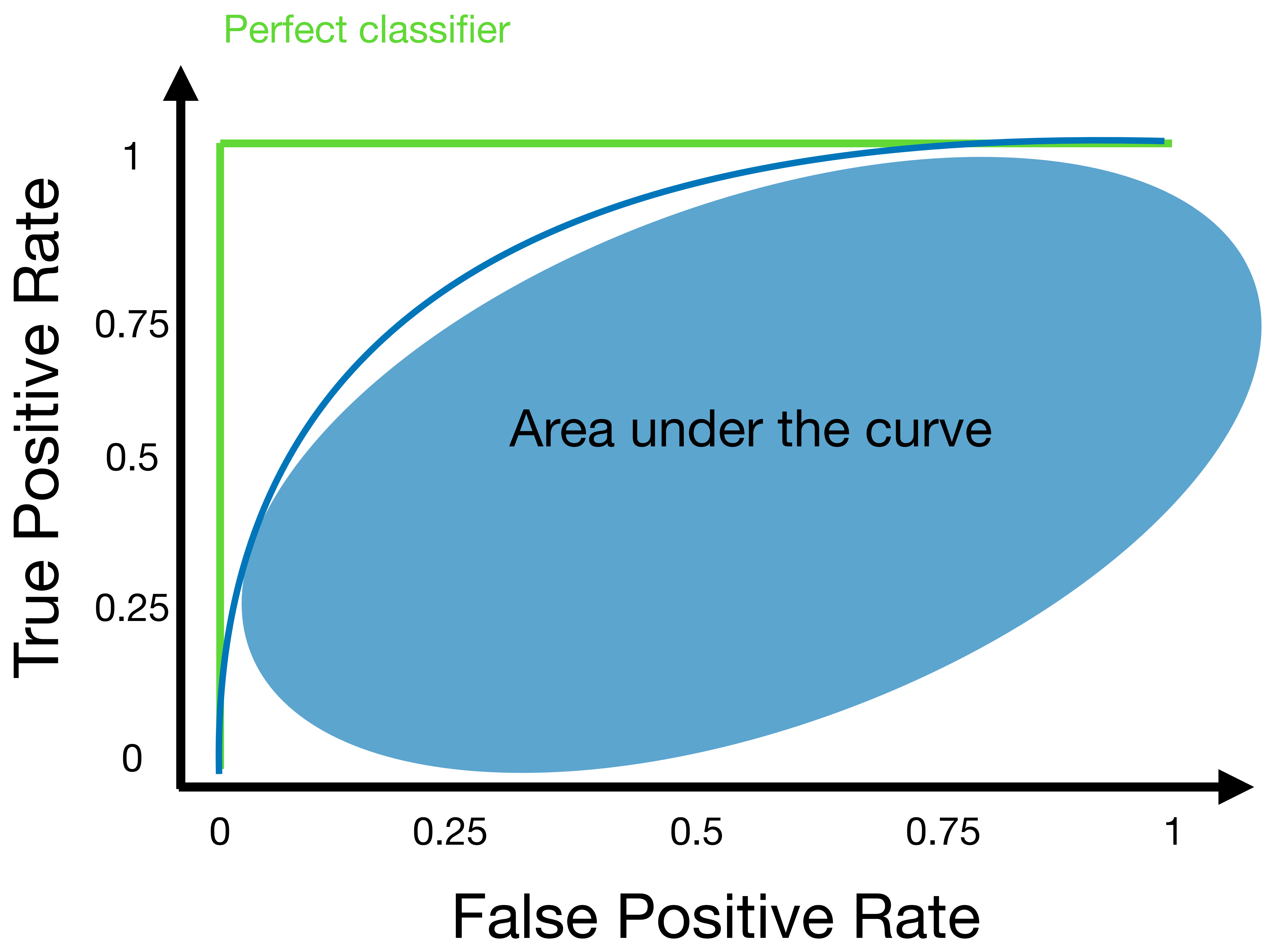
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

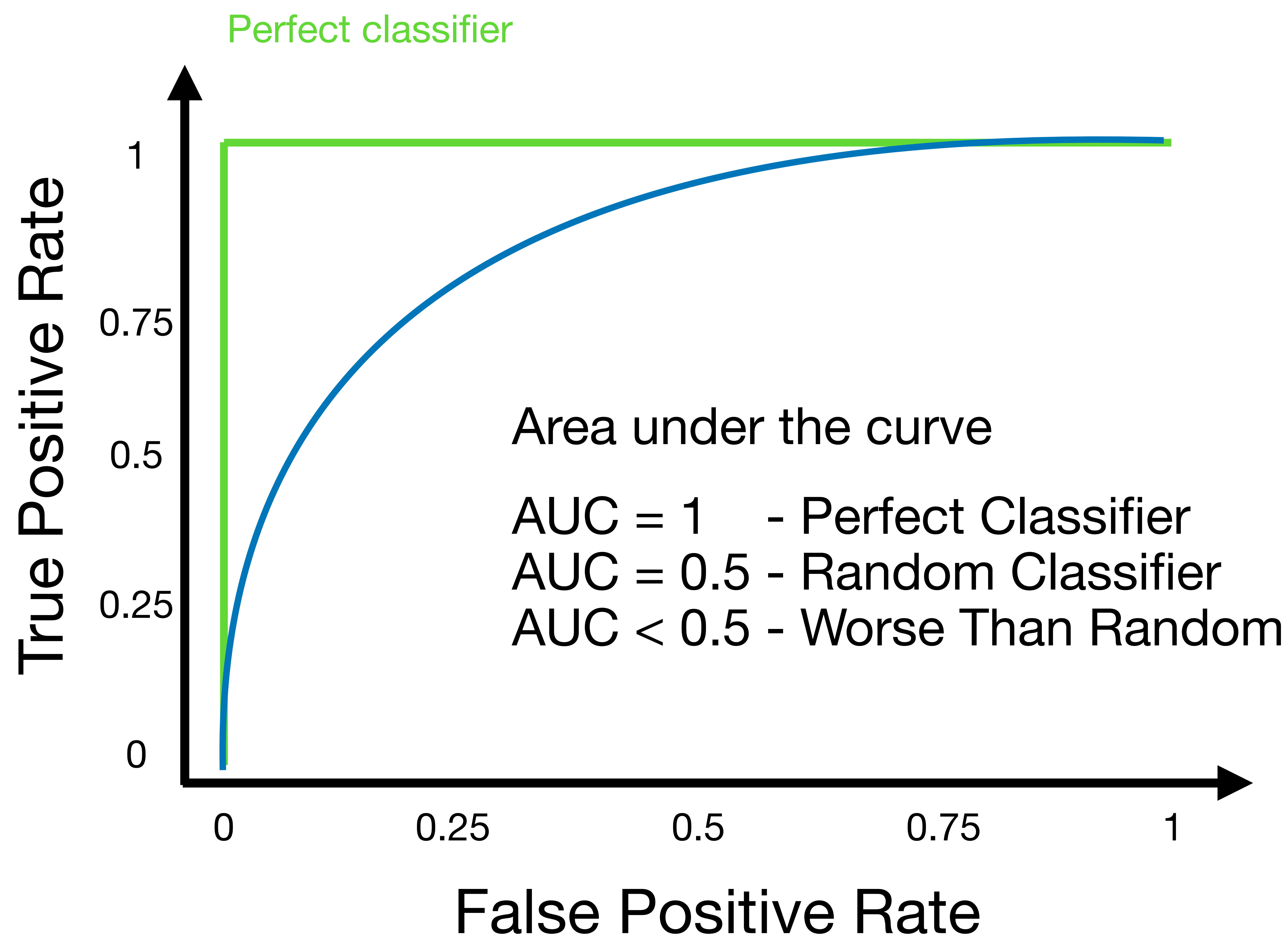
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$

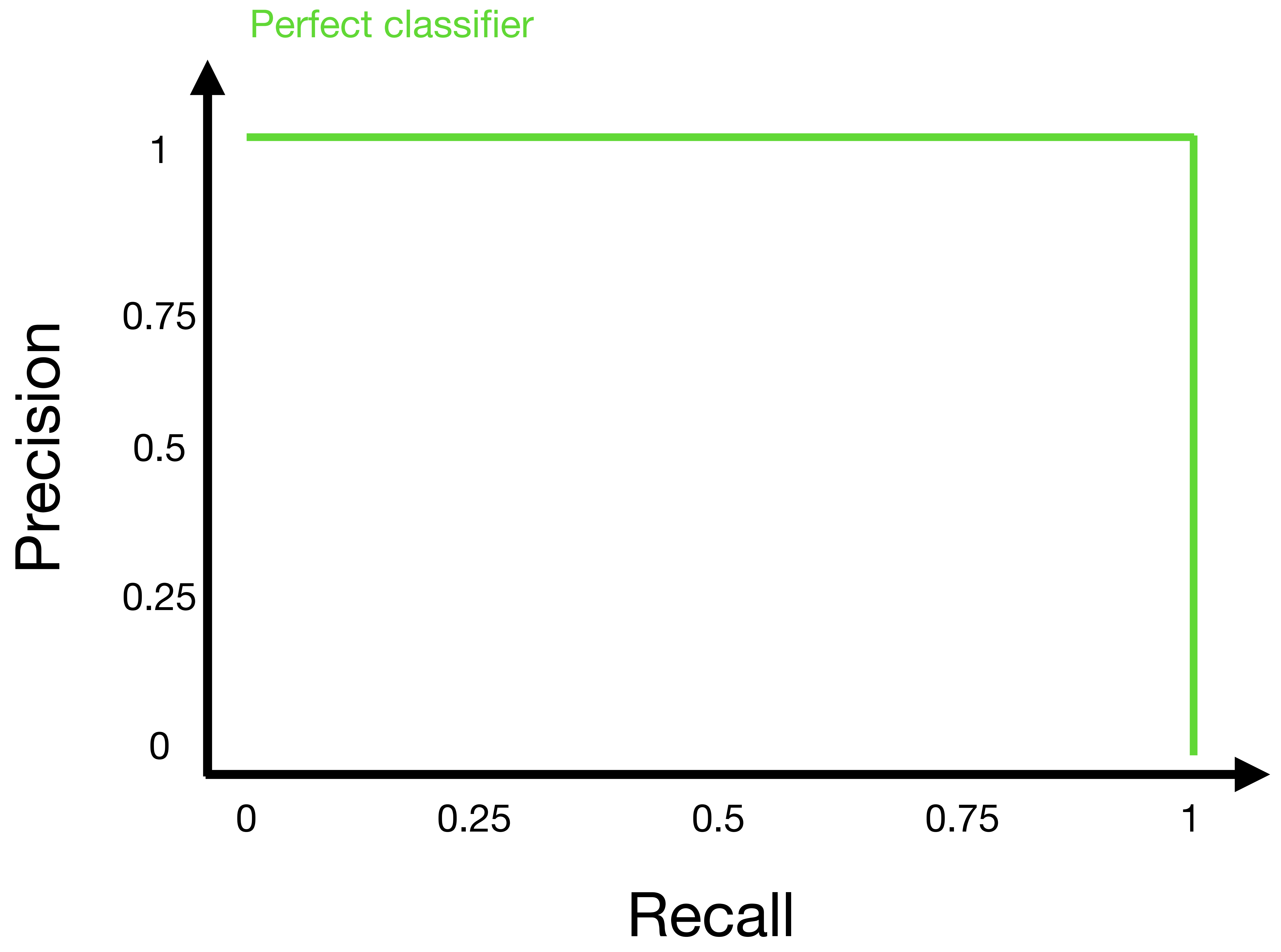


Area under the curve

AUC = 1 - Perfect Classifier
AUC = 0.5 - Random Classifier
AUC < 0.5 - Worse Than Random Classifier

Metrics

Area Under Precision-Recall Curve (AUP)



Perfect classifier:

Precision stays at 1.0 across all recall values. AUC-PR = 1.0.

Every positive prediction is correct, and all actual positives are found.

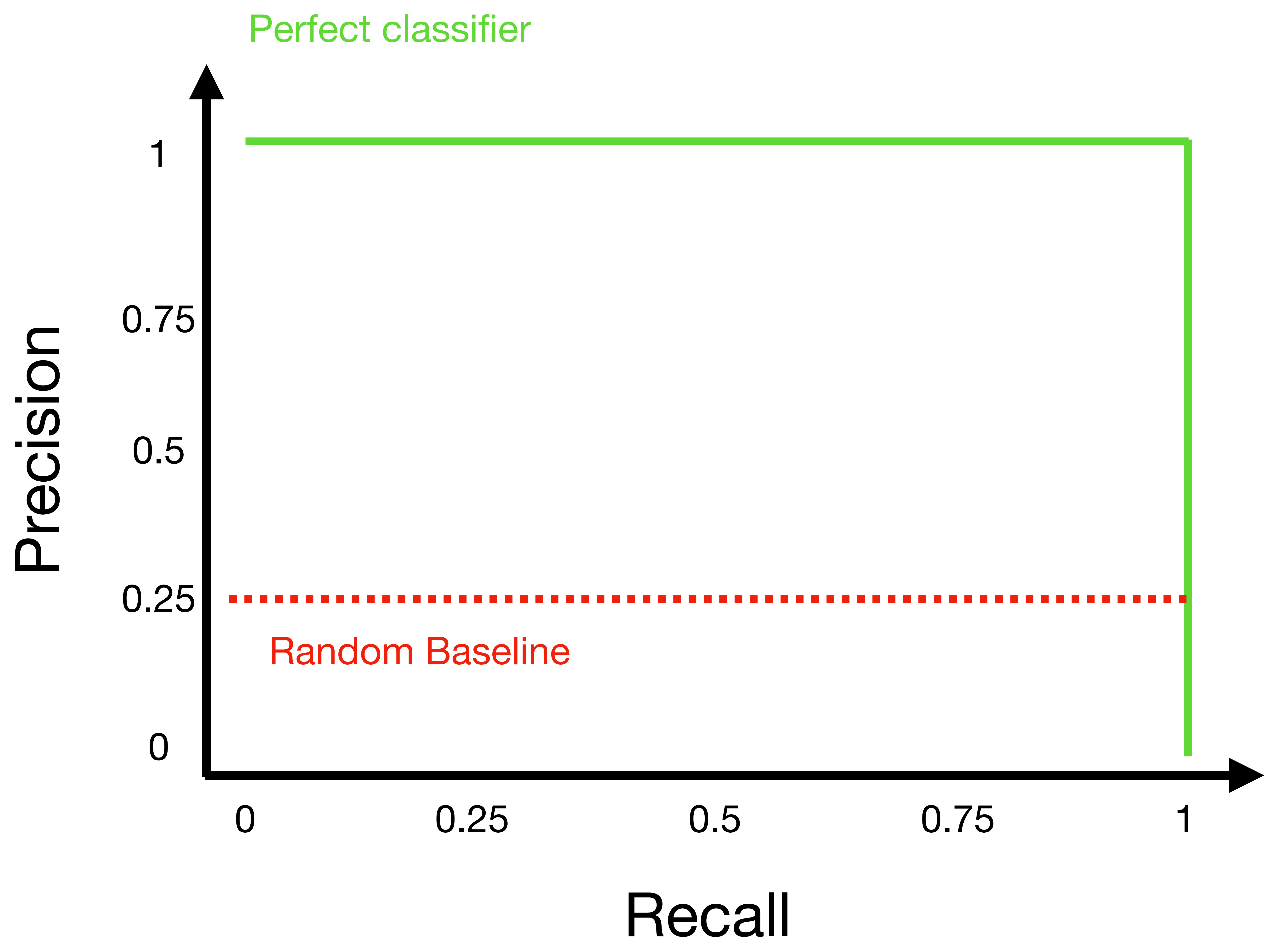
Metrics

Area Under Precision-Recall Curve (AUP)

Random classifier:

Horizontal line at the proportion of positives (25% here).

AUC-PR equals the class proportion. No predictive power.



Perfect classifier:

Precision stays at 1.0 across all recall values. AUC-PR = 1.0.

Every positive prediction is correct, and all actual positives are found.

Metrics

Area Under Precision-Recall Curve (AUP)

Random classifier:

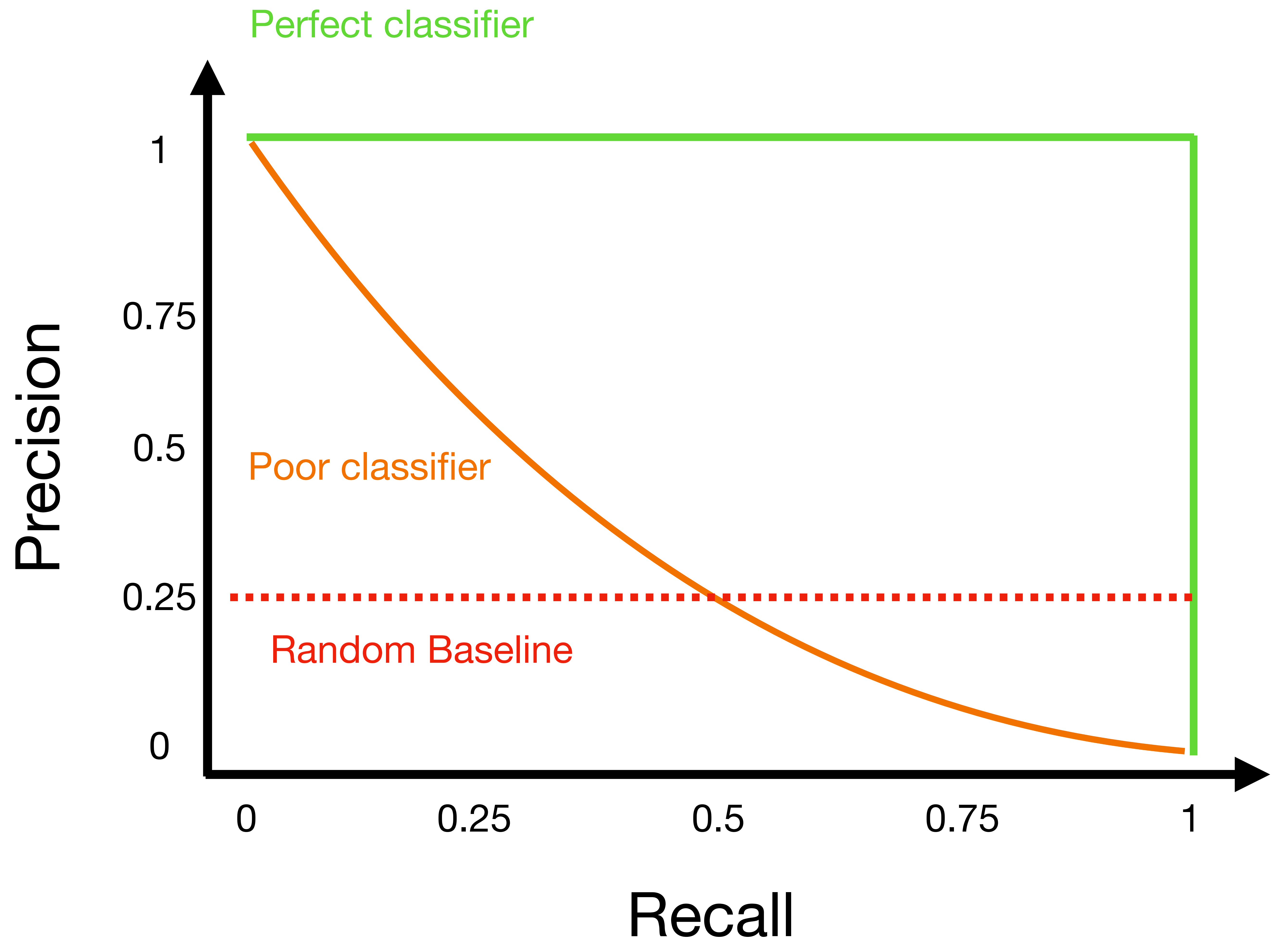
Horizontal line at the proportion of positives (25% here).

AUC-PR equals the class proportion. No predictive power.

Poor classifier:

Precision drops steadily as recall increases.

Still better than random, but significant tradeoff between precision and recall.



Perfect classifier:

Precision stays at 1.0 across all recall values. AUC-PR = 1.0.

Every positive prediction is correct, and all actual positives are found.

Metrics

Area Under Precision-Recall Curve (AUP)

Random classifier:

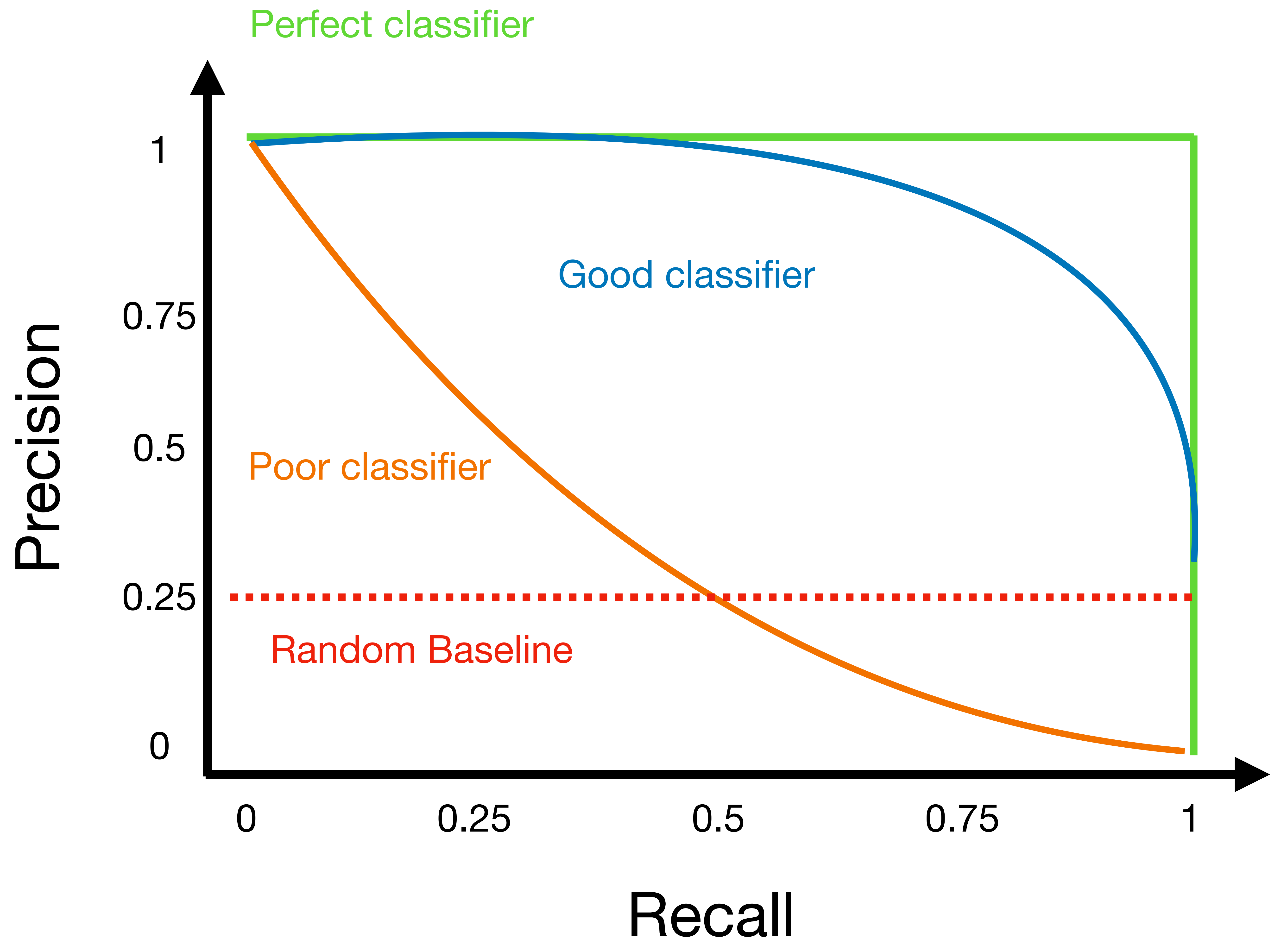
Horizontal line at the proportion of positives (25% here).

AUC-PR equals the class proportion. No predictive power.

Poor classifier:

Precision drops steadily as recall increases.

Still better than random, but significant tradeoff between precision and recall.



Perfect classifier:

Precision stays at 1.0 across all recall values. AUC-PR = 1.0.

Every positive prediction is correct, and all actual positives are found.

Good classifier:

High precision maintained until high recall.

The curve hugs the top-right corner.

Comparisons

1. Gradient Descent vs Closed Form

Gradient Descent

+ Linear increase in m (# training data) and n (# features)
+ Generally applicable to multiple models
+ Guaranteed to reach global optimum for convex functions and appropriate learning rate
- Need to choose learning rate α and stopping conditions
- Need to choose optimization method (Adam, RMSProp etc..)
- Might get stuck in local optima / saddle point
- Needs feature scaling

Closed Form

$$\theta = (X^T X)^{-1} X^T Y$$

+ No parameter tuning
+ Gives global optimum
- Not generally applicable to any learning algorithm
- Slow computation - scales with n^3 where n is number of features

2. Batch vs Mini-Batch vs Stochastic Gradient Descent

Batch
Pros:

Stable Convergence: No noise in gradient estimates means smooth, predictable progress toward the minimum

Guaranteed Descent: Each update is guaranteed to reduce the loss (with appropriate learning rate)

Simple learning rate selection: The lack of noise means you can often use larger learning rates without instability

Parallelizable Gradient Computation: The sum over all samples can be computed in parallel across multiple processors

Stochastic
Pros:

Fast Updates: Each parameter update is computationally cheap, allowing rapid initial progress.

Memory Efficient: Only one sample needs to be in memory at a time.

Escapes Local Minima: The inherent noise helps the algorithm escape shallow local minima and saddle points. <u>The stochasticity acts as implicit regularization</u>

Online Learning: Can naturally incorporate new data as it arrives - just perform an update on each new sample

Better Generalization: The noise can prevent overfitting to the training set.

2. Batch vs Mini-Batch vs Stochastic Gradient Descent

Batch Cons:

Computationally Expensive: For large datasets, computing the full gradient is very slow. A dataset with 10 million samples requires processing all 10 million before a single update.

Memory Intensive: The entire dataset must fit in memory.

Redundant Computation: Many datasets contain redundant or similar samples. BGD computes gradients for all of them even when a subset would provide nearly the same information.

Poor Escape From Local Minima: The **deterministic** nature means the algorithm follows the same path every time and can get permanently stuck in local minima or saddle points.

Slow for Online Learning: Cannot incorporate new data without reprocessing everything.

Stochastic Cons:

High Variance: Individual gradient estimates can be very noisy, causing erratic updates.

Unstable Convergence: The loss curve is noisy. The algorithm may step away from the minimum even when near it.

Requires Learning Rate Decay: To converge to a minimum (rather than oscillating around it), the **learning rate must decrease** over time, adding hyperparameters.

Poor Hardware Utilization: Modern GPUs are optimized for **parallel operations on batches,** not sequential single-sample operations. SGD fails to exploit this.

Sensitive to Sample Ordering: The order in which samples are presented can affect results, requiring careful shuffling.

2. Batch vs Mini-Batch vs Stochastic Gradient Descent

Mini-Batch

Variance Reduction: Averaging over B samples reduces gradient variance by a factor of B compared to pure SGD, while still maintaining some beneficial noise

Hardware Efficiency: GPUs perform matrix operations in parallel. A batch size of 64 is nearly as fast as a batch size of 1 on modern hardware, giving essentially 64× speedup over SGD

Memory-Computation Tradeoff: Batch size can be tuned to maximize GPU memory utilization without requiring the full dataset

Balances Exploration and Exploitation: Enough noise to escape poor regions, enough signal to make consistent progress.

3. k-Nearest Neighbors

Choosing k

- k is the primary hyper-parameter controlling the bias-variance tradeoff

Small k (e.g. $k = 1$)

- High variance, low bias
- Decision boundary is highly irregular
- Very sensitive to noise and outliers
- Prone to overfitting, but can capture fine grained structure

Practical Tips

- Start with $k = \sqrt{m}$
- Use cross-validation to select optimal k
- If k is odd, it avoids ties in binary classification
- k should be smaller than the smallest class size

Large k (e.g. $k = m$)

- High bias, low variance
- Decision boundary is very smooth
- Robust to noise, but may miss local patterns
- At the extreme of $k = m$, always predicts majority class

3. k-Nearest Neighbors

Pros

- Simple to understand and implement
- No training phase (fast to “train”)
- Naturally handles multi-class classification
- Non-parametric: makes no distributional assumptions
- Can capture arbitrarily complex decision boundaries
- Easily adapts to new training data (just add it)

Cons

- Slow prediction for large datasets
- High memory requirement (stores all training data)
- Sensitive to irrelevant features and feature scaling
- Struggles in high dimensions (curse of dimensionality)
- No interpretable model or feature importance
- Requires meaningful distance metric

3. k-Nearest Neighbors

When to use k-NN?

Use

- Small to medium datasets
- Low to moderate dimensionality ($n < 20$)
- Non-linear decision boundaries expected
- Data arrives incrementally (online learning)
- Quick baseline model needed

Don't Use

- Large datasets with real-time prediction requirements
- Very high-dimensional data
- Features have varying relevance
- Interpretability is required

4. LDA

Pros

- Simple, fast, closed-form solution
- No hyperparameters to tune
- Works well when assumptions approximately hold
- Provides probabilistic outputs
- Built-in dimensionality reduction
- Stable with small datasets

Cons

- Assumes Gaussian distributions
- Assumes shared covariance (linear boundaries only)
- Sensitive to outliers (affect mean and covariance estimates)
- Cannot capture non-linear relationships
- Fails if features are highly non-Gaussian

5. Classifiers

Comparison	**Logistic Regression**	**LDA**
Type	Discriminative	Generative
Assumption	Conditional Independence Between Rows of Data	Gaussian and shared covariance
Training	Gradient Descent	Closed Form
Data	Better with large data else risk overfitting	Works well across data sizes
Probabilities	Well calibrated	Well calibrated
Missing features	Requires pre-processing	Requires pre-processing