Northeastern University
Khoury College of
Computer Sciences

$k-$Nearest Neighbors

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

Monday | February 2, 2026

Today’s Outline

- **Metrics**
- k-Nearest Neighbors

Updates

$$\hat{y} = \theta_0^{x_0} + \theta_1^{x_1}$$

$$\hat{y} = \theta_0 \phi(x_0) + \theta_1 \phi(x_1)$$

$\sin(x_0)$ $\quad e^{x_1}$ $\quad \phi(x)$

- Office hour timings
- Topic based office hours
- Homework 2 is out - due Friday Feb 15th 11:59PM
- Homework 1 - answers on Wednesday
- Final exam dates
- Functions that can be considered linear regression

Metrics

- An obvious metric is **accuracy**

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Data Points}}$$

- Say you have a cat classifier with 1000 images. Your classifier gets 797 out of 1000 images correct

$$Accuracy = \frac{797}{1000} = 79\,\%$$

Metrics

- But, accuracy does not tell the whole picture
- Especially when data is skewed
  - For example, if your training data is of size 1000 images
    - 900 of them are of dogs
    - 100 of them are cats
  - **Question**: Is accuracy a good metric in this case?

Metrics

Confusion Matrix

1000

	Predicted Positive	Predicted Negative
Actual Positive	True Positive. = 500	False Negative.
Actual Negative	False Positive.	True Negative.

Metrics

Confusion Matrix

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

Metrics

Confusion Matrix

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

Metrics

Confusion Matrix

$$\text{Accuracy} = \frac{TP + TN}{\text{Everything}},$$

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

Metrics

Accuracy

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

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

The proportion of correct predictions.

Simple and intuitive, but misleading for imbalanced data.

A classifier that always predicts the majority class achieves high accuracy on imbalanced datasets while being useless.

Metrics

Precision and Recall

Precision →

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

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

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

	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

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

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

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

Metrics

Precision and Recall

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

→ Criminal trials.

→ expensive.

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

→ disease ✓

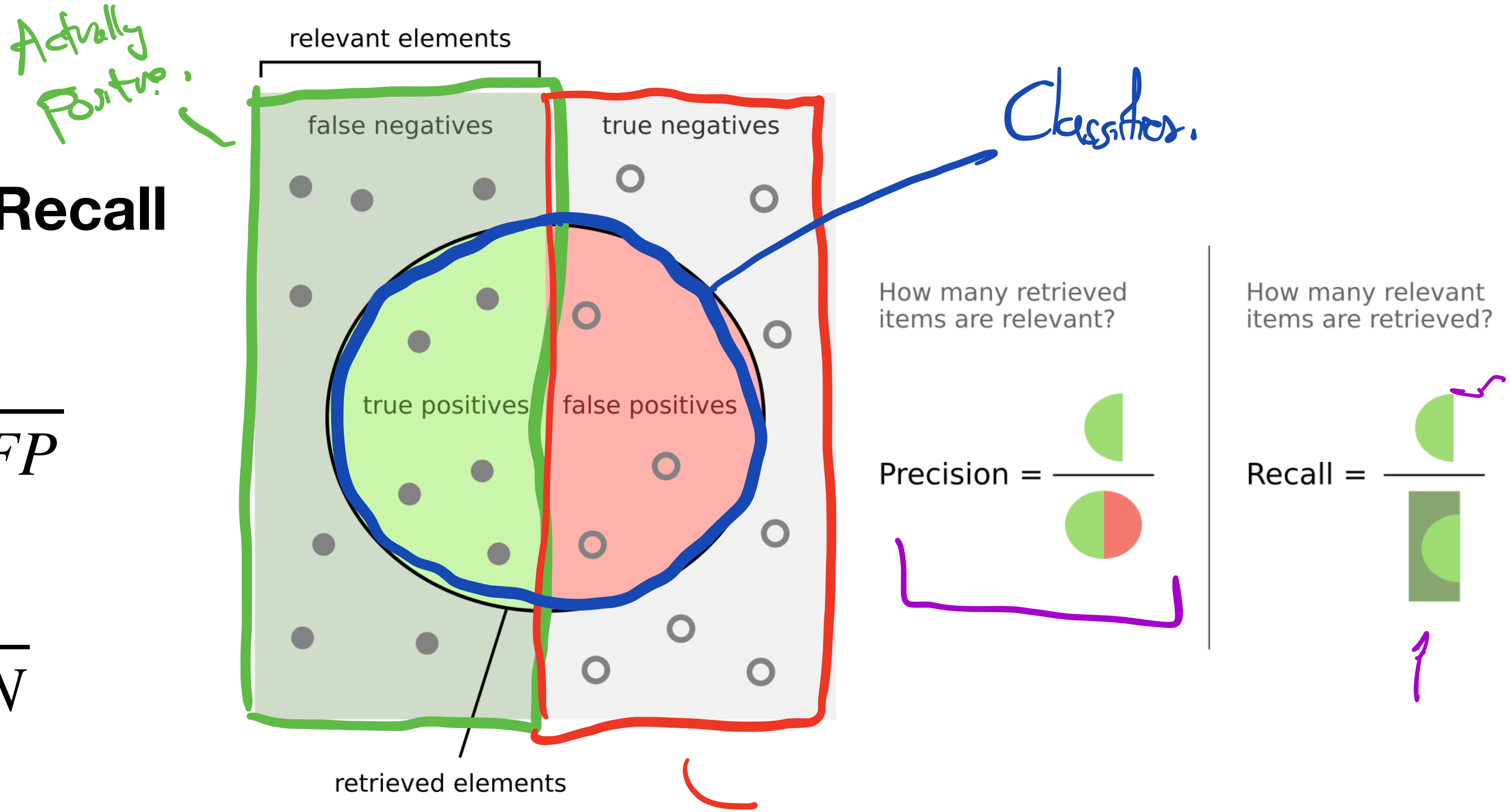
	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

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

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

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

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).

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

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

Metrics

Precision and Recall

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

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

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

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

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

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

The harmonic mean of precision and recall

F1 score is high only when **both** precision and recall are high

	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

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

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

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

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

	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

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

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

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

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

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

Same denominator

	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

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

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

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

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

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

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

Same denominator

FPR = 1 - Specificity

Cat — 0
dogs — 1

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)

Recall is the True Positive Rate

Metrics

Why so many metrics?

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

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

Assume your training data looks like this:

1000 rows
10 rows are spam emails → 1
990 are legitimate emails → 0

Scenario 1:

Classifier always outputs legit
What is the accuracy?

Metrics

Why so many metrics?

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

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

Assume your training data looks like this:

1000 rows
10 rows are spam emails
990 are legitimate emails

Scenario 1:

Classifier always outputs legit
What is the accuracy?

$$Acc = \frac{0 + 990}{0 + 990 + 0 + 10} = 99\,\%$$

Metrics

Why so many metrics?

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

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

Assume your training data looks like this:

1000 rows
10 rows are spam emails
990 are legitimate emails

Scenario 1:

Classifier always outputs legit
What is the accuracy?

$$Acc = \frac{0 + 990}{0 + 990 + 0 + 10} = 99\,\%$$

Scenario 2:

Classifier predicts **one** spam email as spam, and rest as legitimate
What is the precision?

Metrics

Why so many metrics?

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

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

Assume your training data looks like this:

1000 rows
10 rows are spam emails
990 are legitimate emails

Scenario 1:

Classifier always outputs legit
What is the accuracy?

$$Acc = \frac{0 + 990}{0 + 990 + 0 + 10} = 99\,\%$$

Scenario 2:

Classifier predicts **one** spam email as spam, and rest as legitimate
What is the precision?

$$Precision = \frac{1}{1 + 0} = 100\,\%$$

Metrics

Why so many metrics?

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

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

Assume your training data looks like this:

1000 rows
10 rows are spam emails
990 are legitimate emails

Scenario 1:

Classifier always outputs legit
What is the accuracy?

$$Acc = \frac{0 + 990}{0 + 990 + 0 + 10} = 99\,\%$$

Scenario 2:

Classifier predicts **one** spam email as spam, and rest as legitimate
What is the precision?

$$Precision = \frac{1}{1 + 0} = 100\,\%$$

Scenario 3:

Classifier always outputs spam
What is the recall?

Metrics

Why so many metrics?

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

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

Assume your training data looks like this:

1000 rows
10 rows are spam emails
990 are legitimate emails

Scenario 1:

Classifier always outputs legit
What is the accuracy?

$$Acc = \frac{0 + 990}{0 + 990 + 0 + 10} = 99\,\%$$

Scenario 2:

Classifier predicts **one** spam email as spam, and rest as legitimate
What is the precision?

$$Precision = \frac{1}{1 + 0} = 100\,\%$$

Scenario 3:

Classifier always outputs spam
What is the recall?

$$Recall = \frac{10}{10 + 0} = 100\,\%$$

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

P(spam | email) = 0.73 (0.9)

P(spam | other email) = 0.45

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)

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 $\mathbb{P}(cat \mid image_i) \geq 0$ – cat
  - Precision goes , Recall goes

Threshold

Metrics

Precision vs Recall Tradeoff - F1 Score

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

90cats

Question:

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

Answer: By changing the threshold

90

	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 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

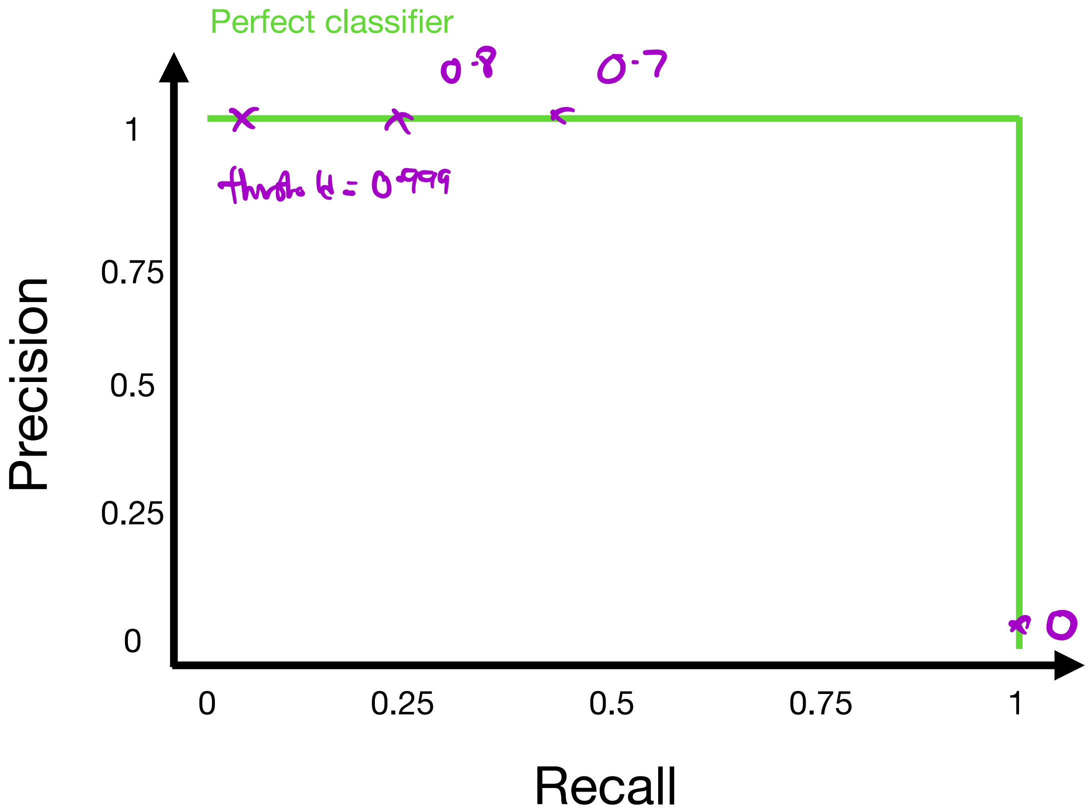
	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

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

0.5

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.

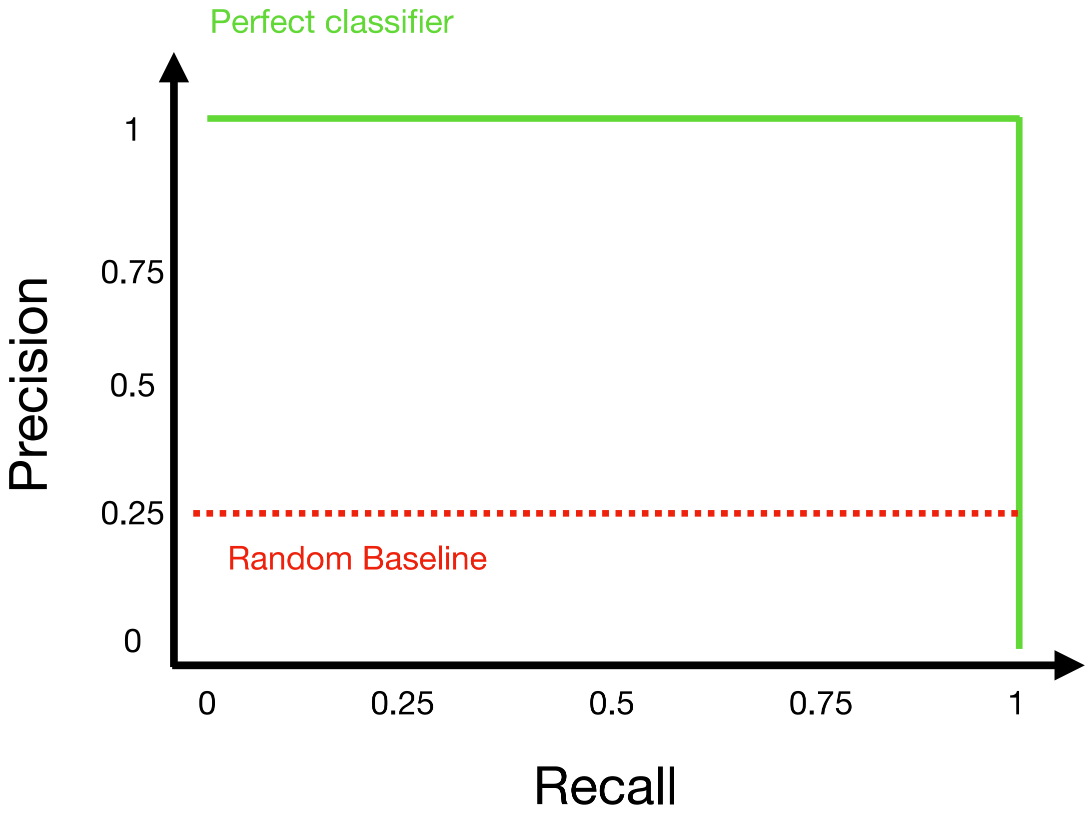
Metrics

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:

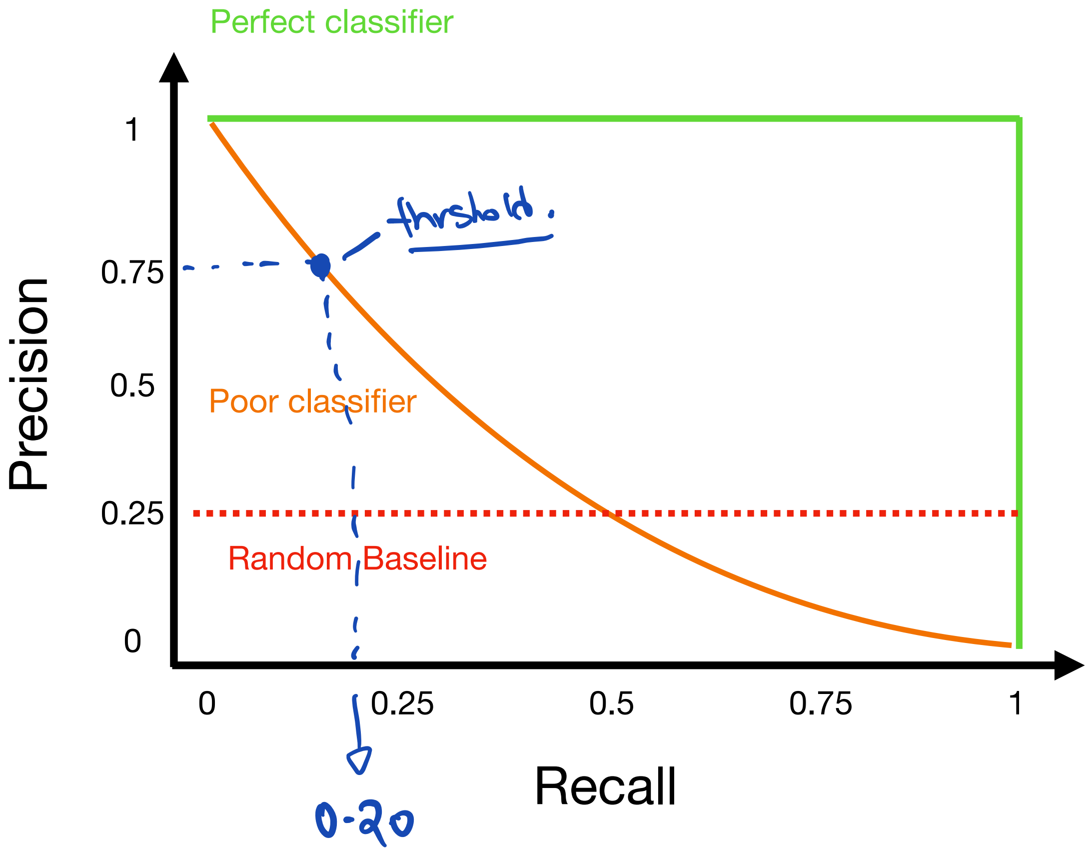
Horizontal line at the proportion of positives (25% here).

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)

P(spam|email1) = 0.75 → spam
P(spam|email2) = 0.59 → Not spam
P(spam|email3) = 0.63 → spam.

Random classifier:

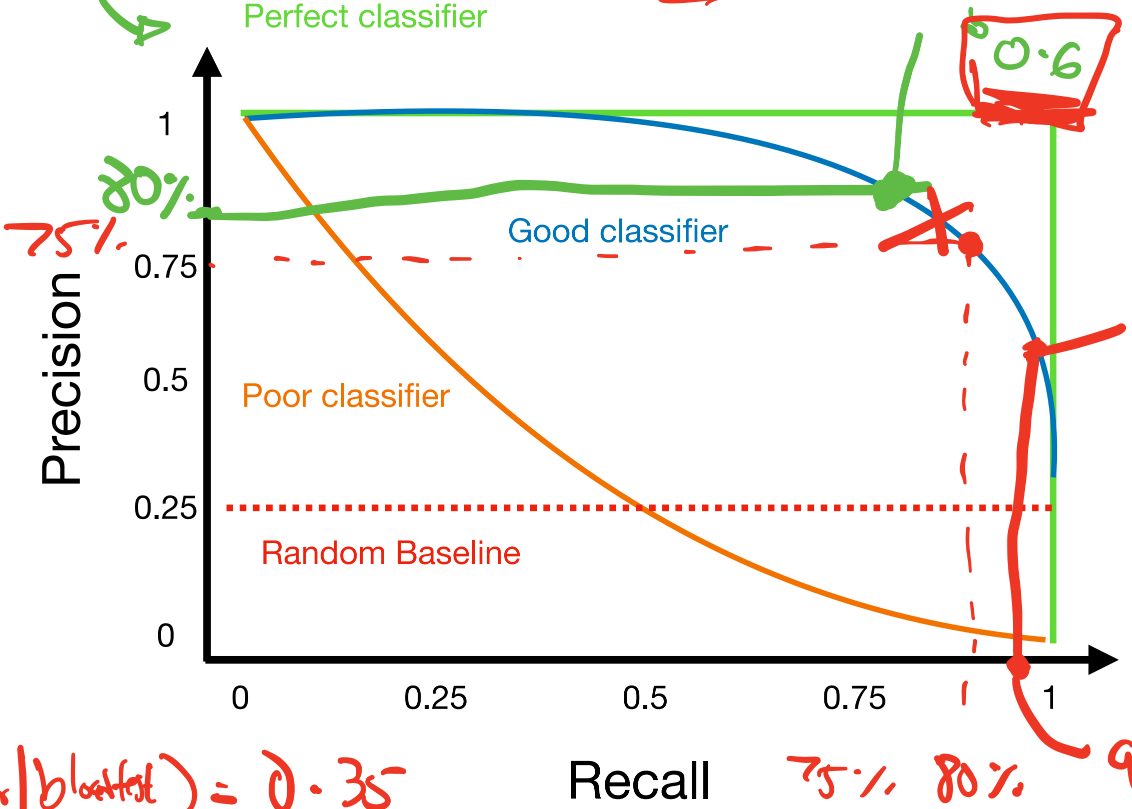
Horizontal line at the proportion of positives (25% here).

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.



P(cancer|bloodtest) = 0.35

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.

0.25

Good classifier:

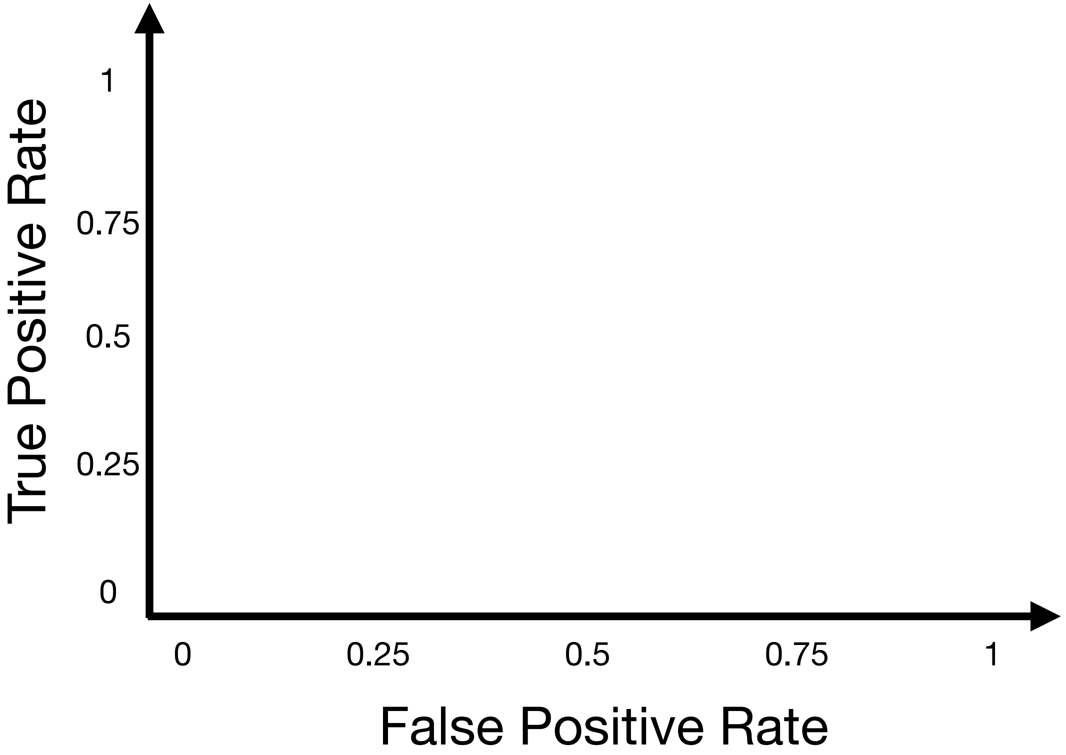
High precision maintained until high recall.

The curve hugs the top-right corner.

Metrics

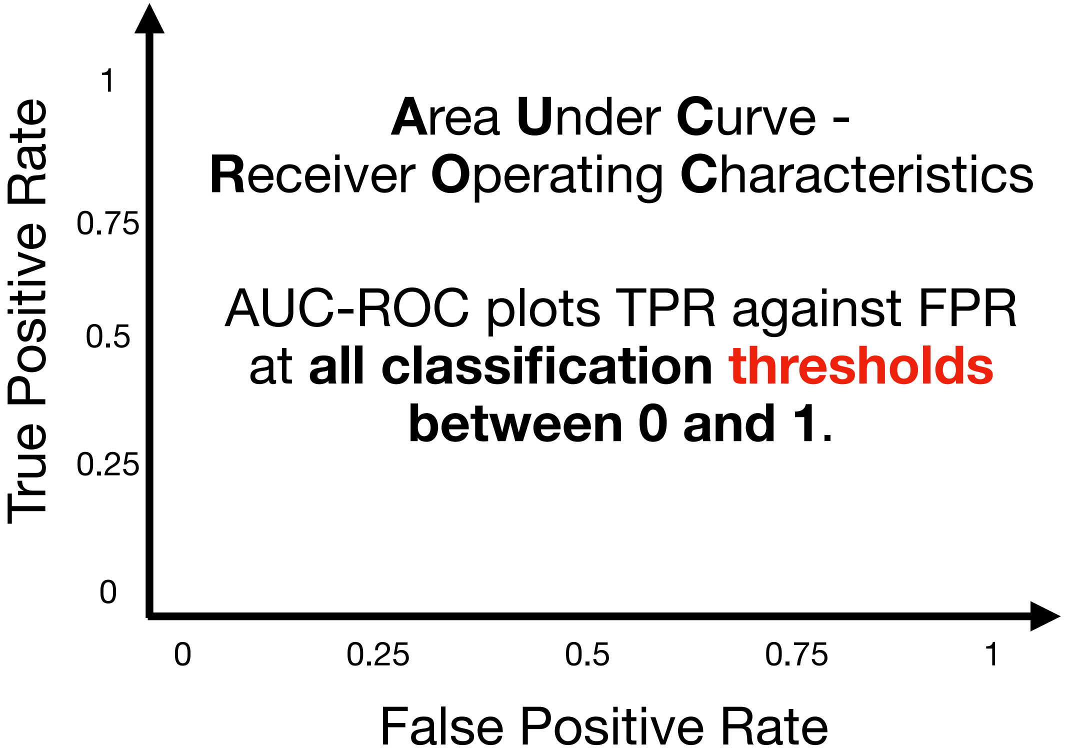
AUC-ROC Curve

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



Metrics
AUC-ROC Curve

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



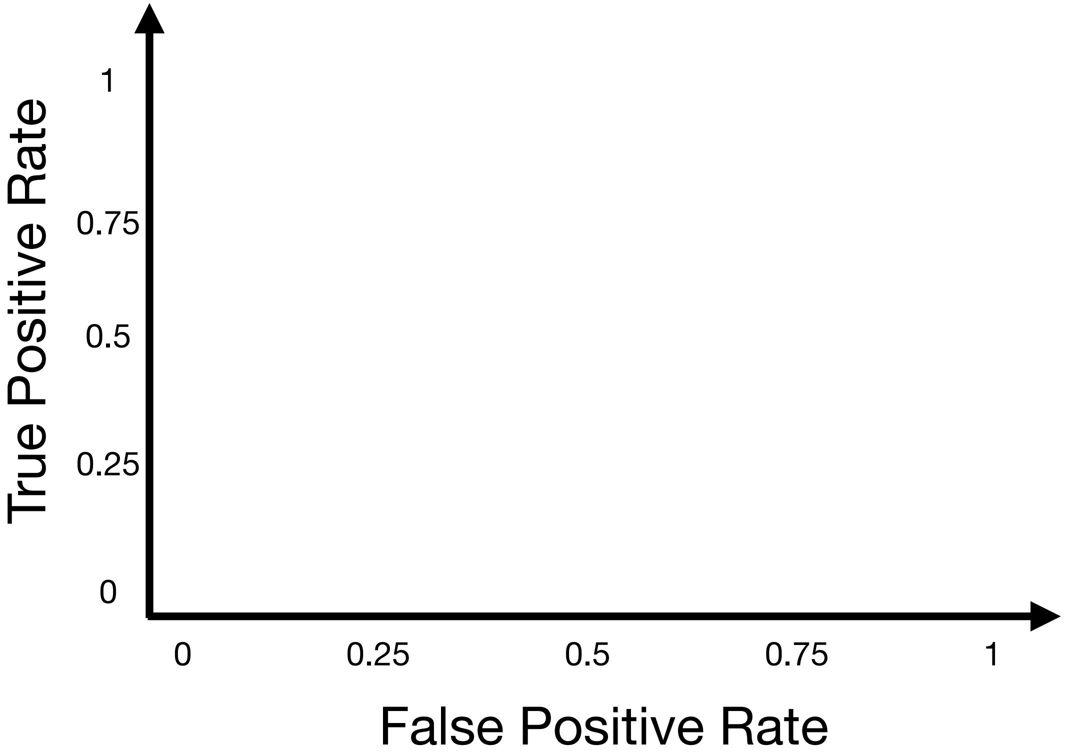
Area **U**nder **C**urve -
Receiver **O**perating **C**haracteristics

AUC-ROC plots TPR against FPR at **all classification thresholds between 0 and 1**.

Metrics

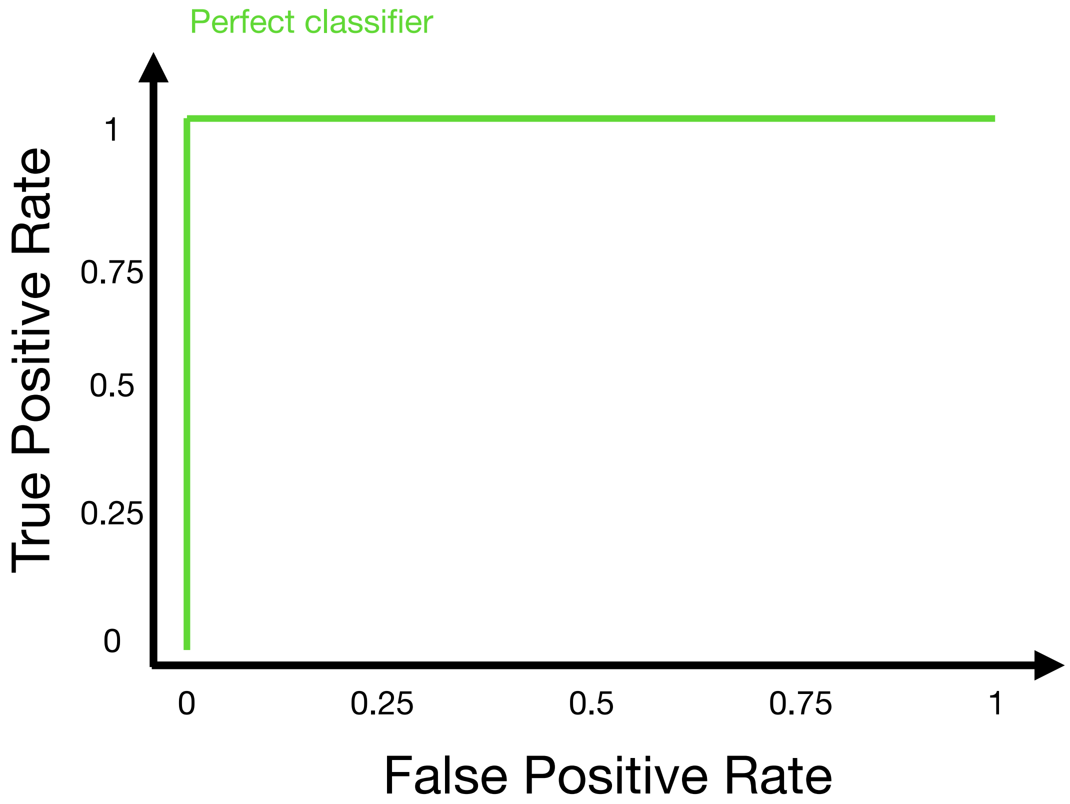
AUC-ROC Curve

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



Metrics
AUC-ROC Curve

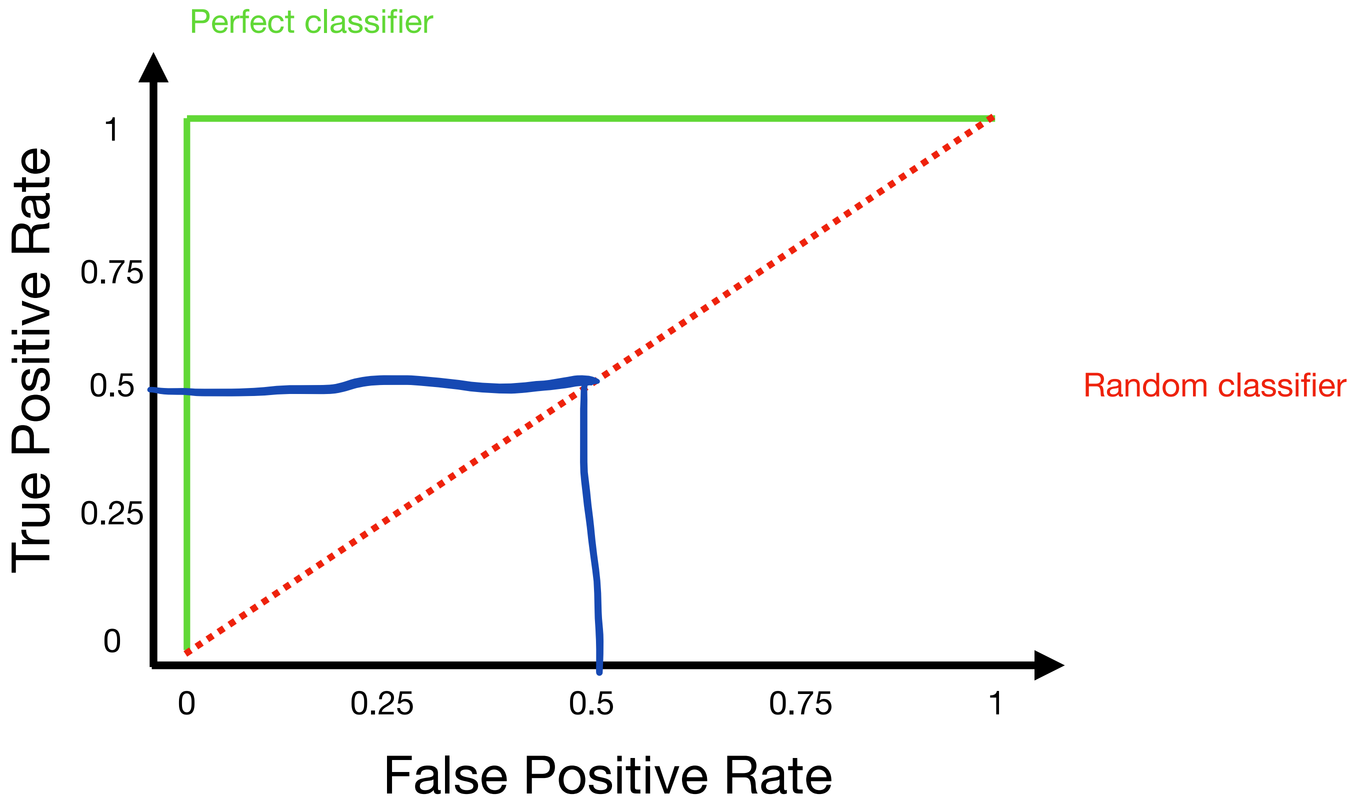
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

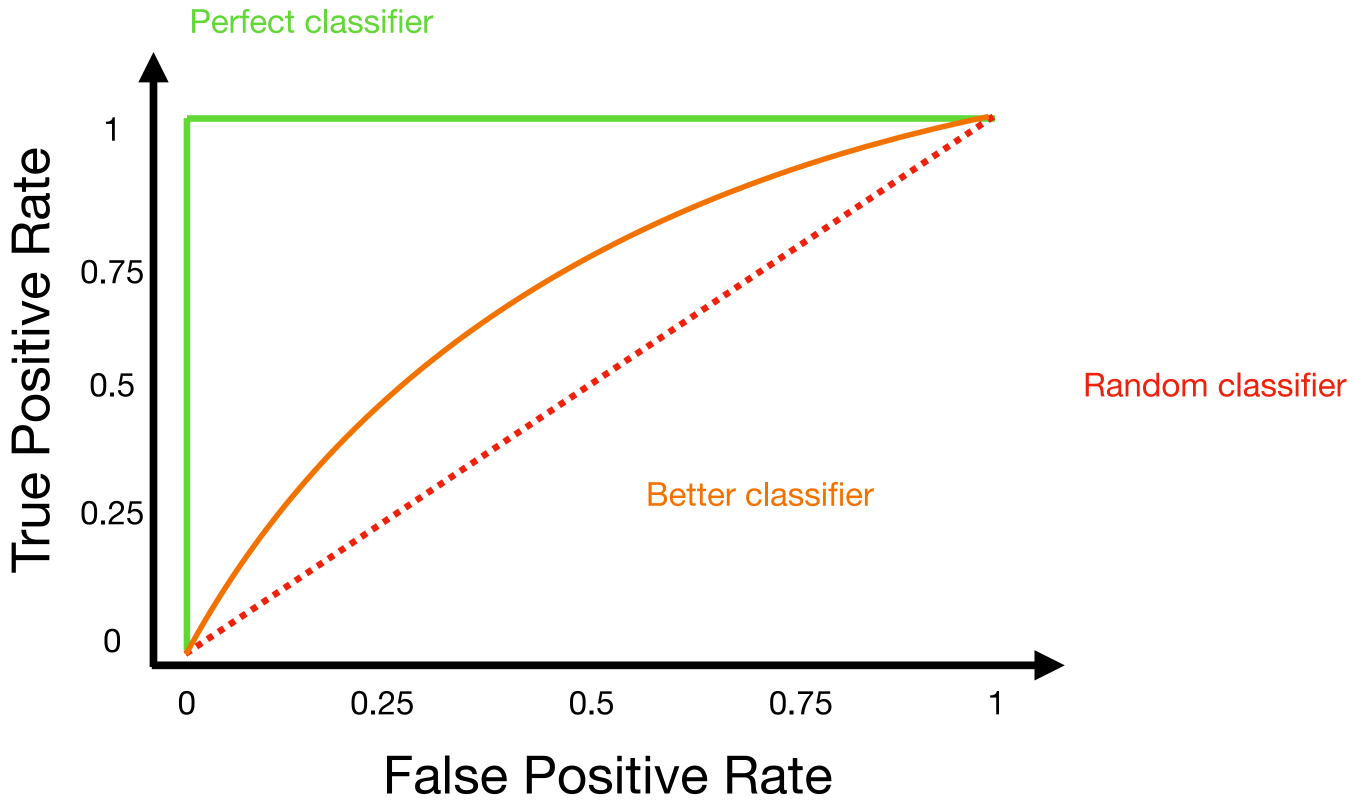
AUC-ROC Curve

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



Metrics
AUC-ROC Curve

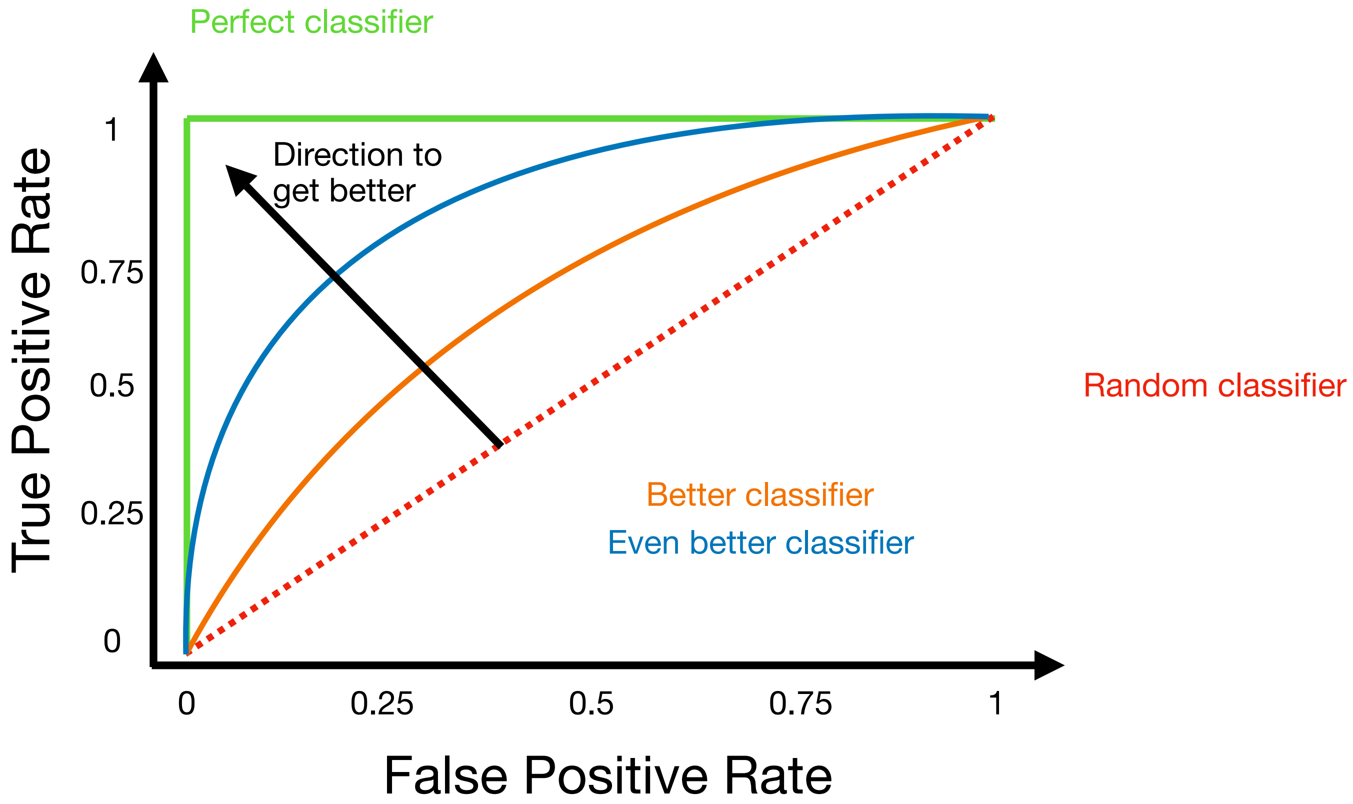
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

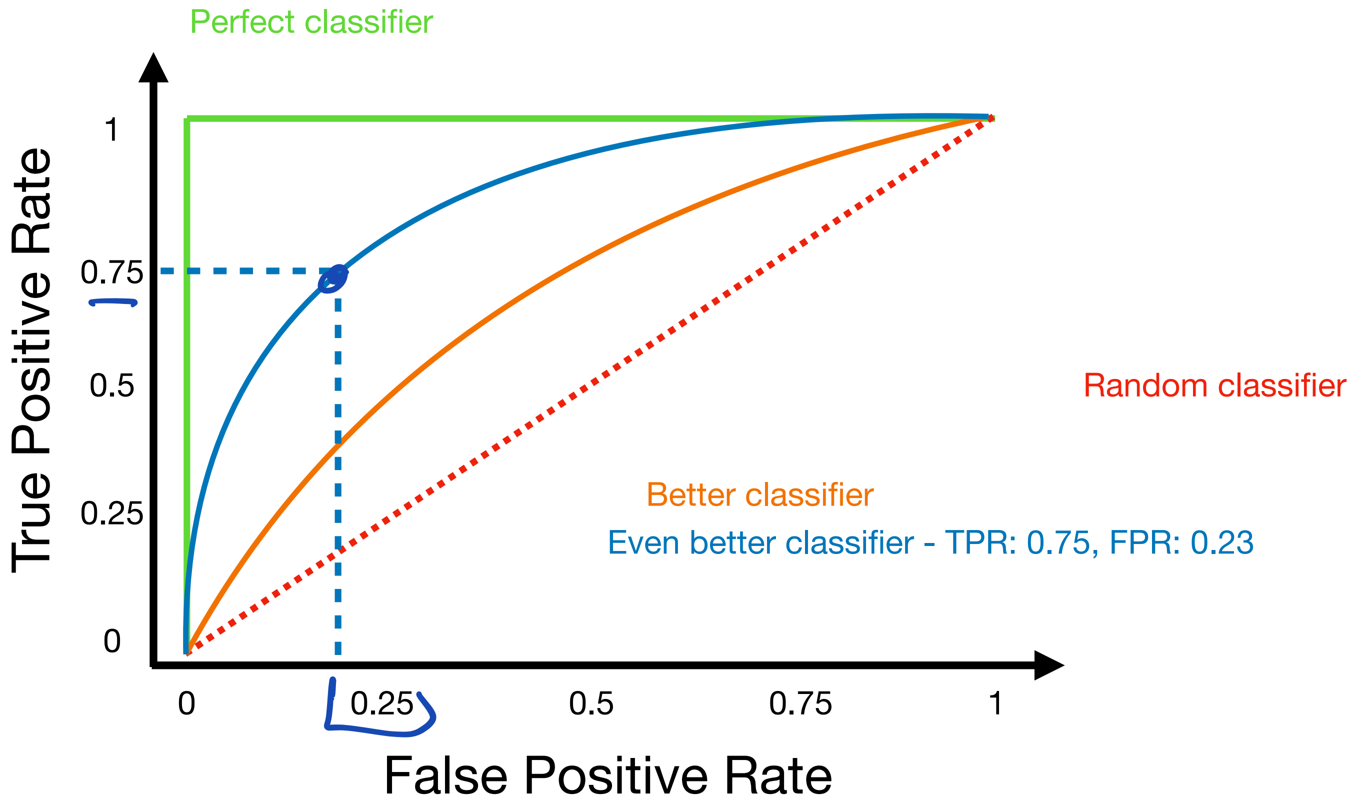
AUC-ROC Curve

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



Metrics
AUC-ROC Curve

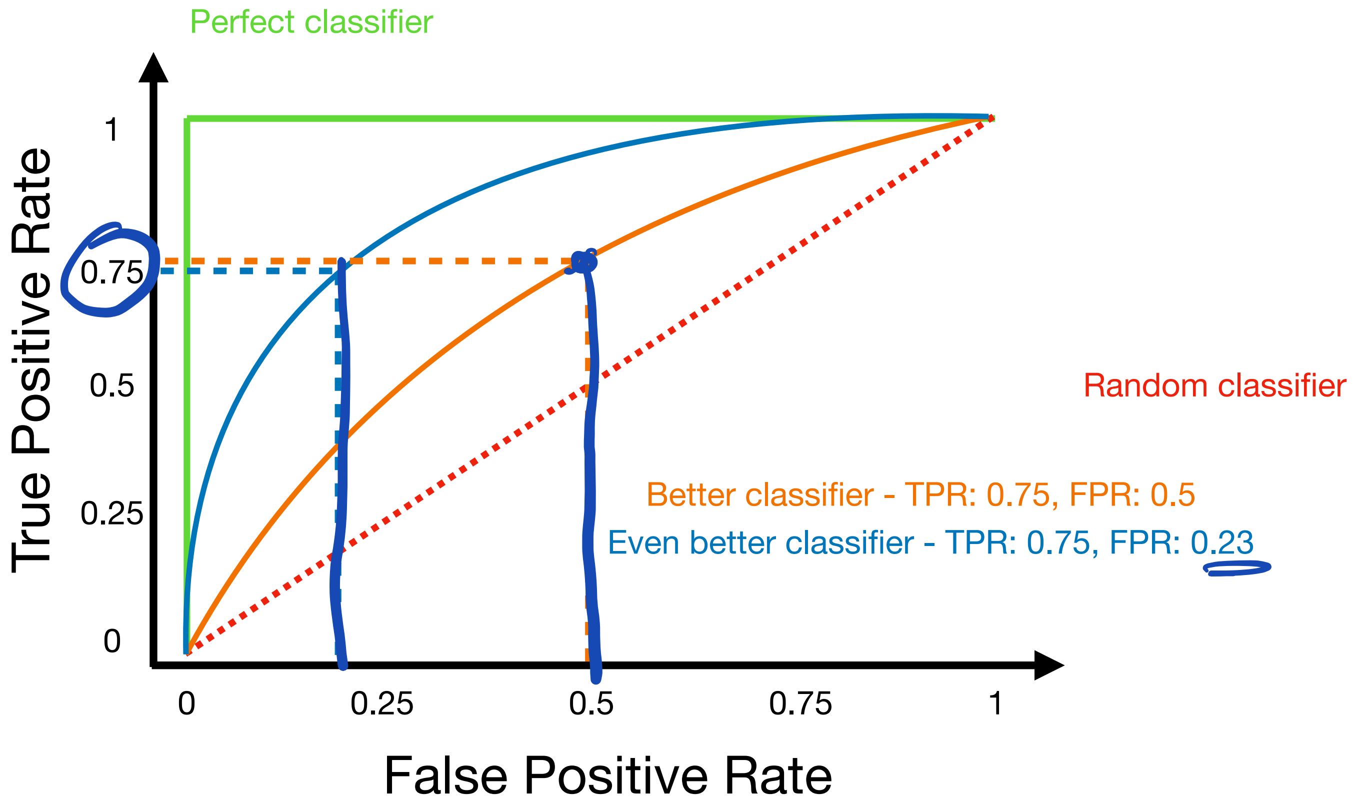
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$



Metrics

AUC-ROC Curve

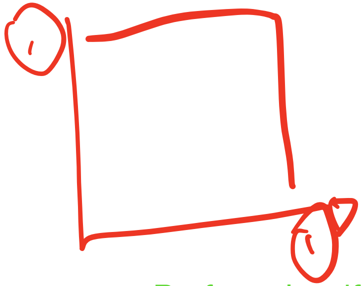
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$

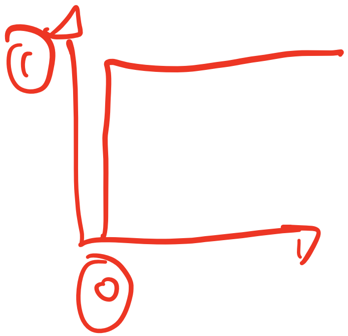


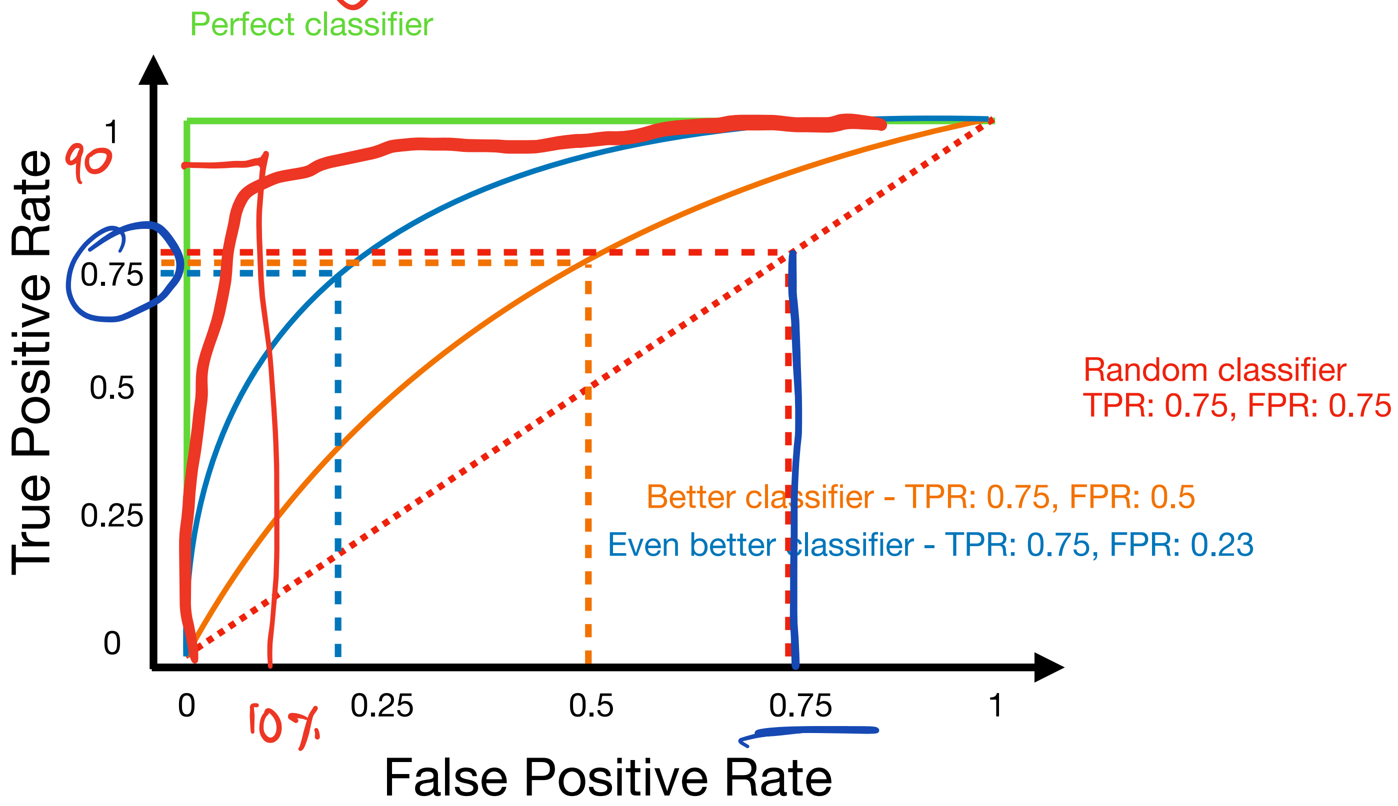
Metrics

AUC-ROC Curve

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



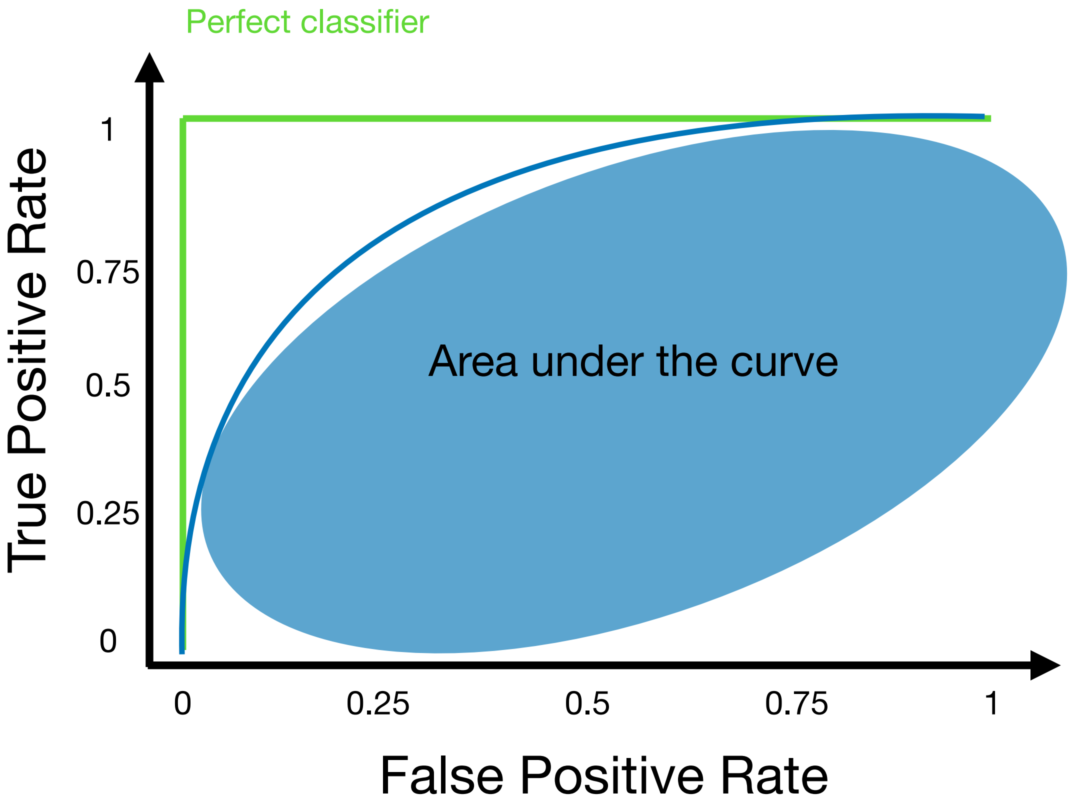




Metrics

AUC-ROC Curve

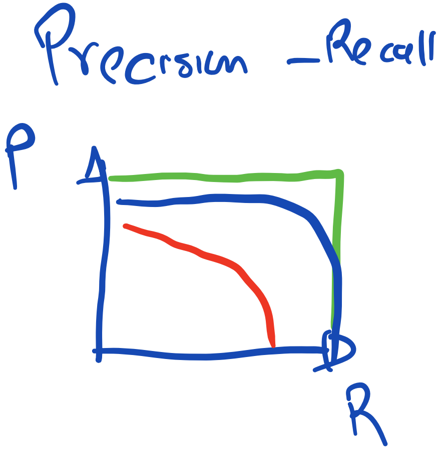
$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$

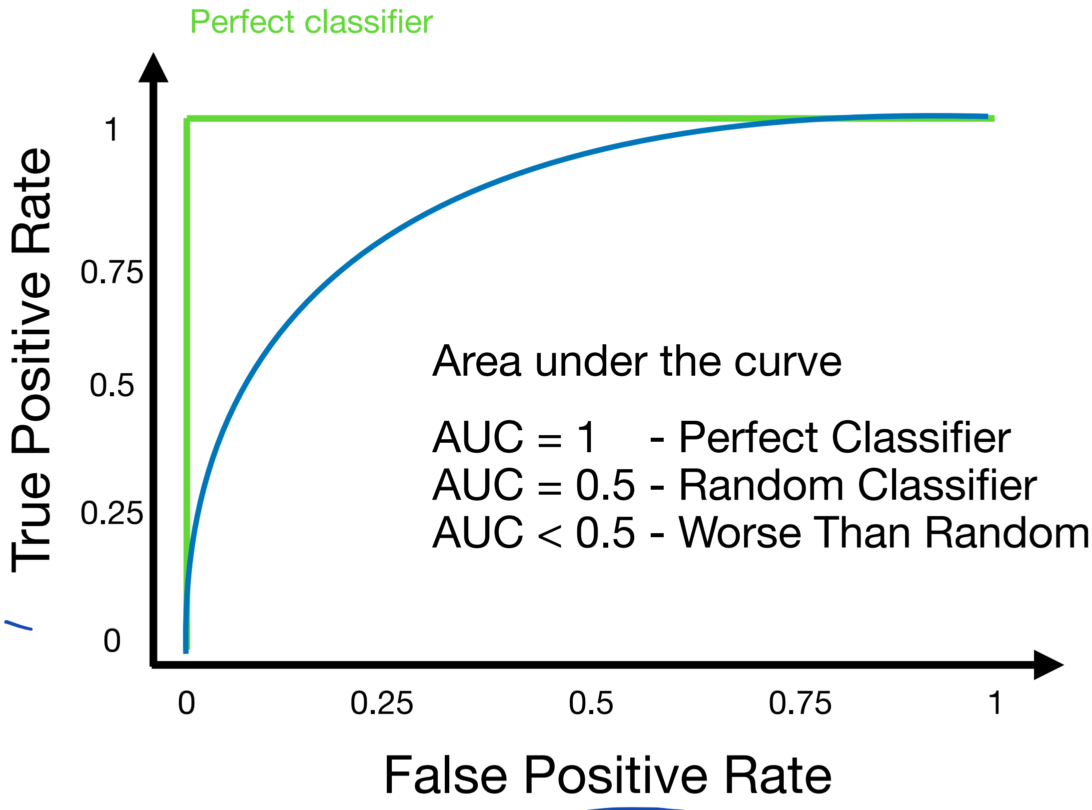


Metrics

AUC-ROC Curve

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

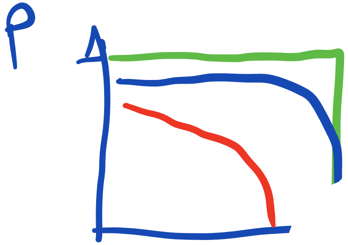


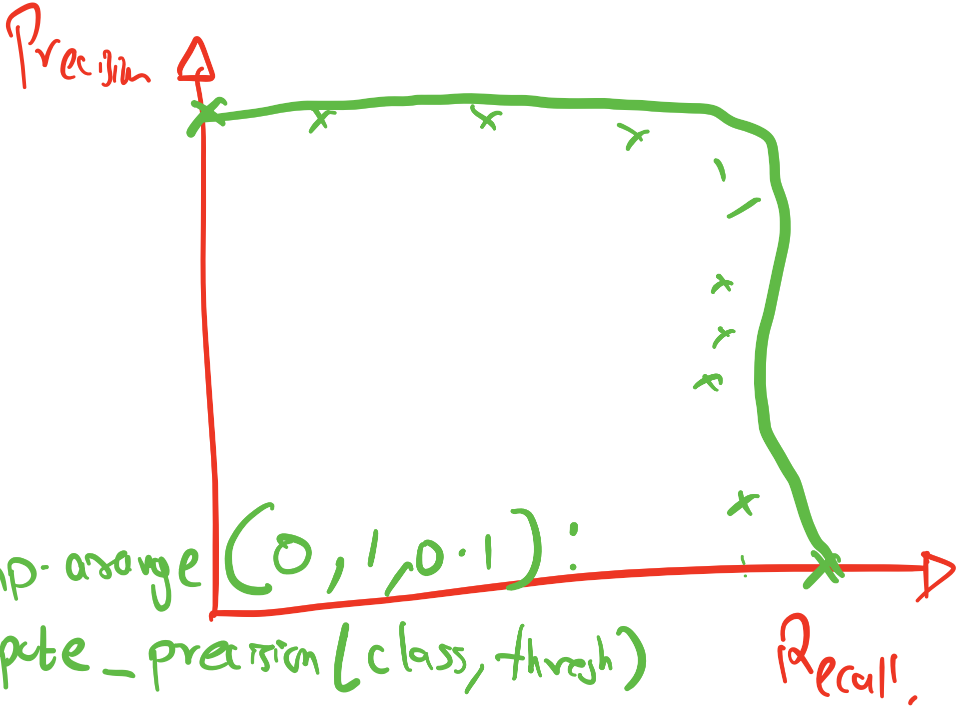


Metrics

AUC-ROC Curve

Precision - Recall





threshold : 0 → r - 1, P - 0

thre = 0.1 — r - 0.9, P - 0.10

0.25 — r - 0.9, P - 0.3

0.99 — r - 0, P - 1

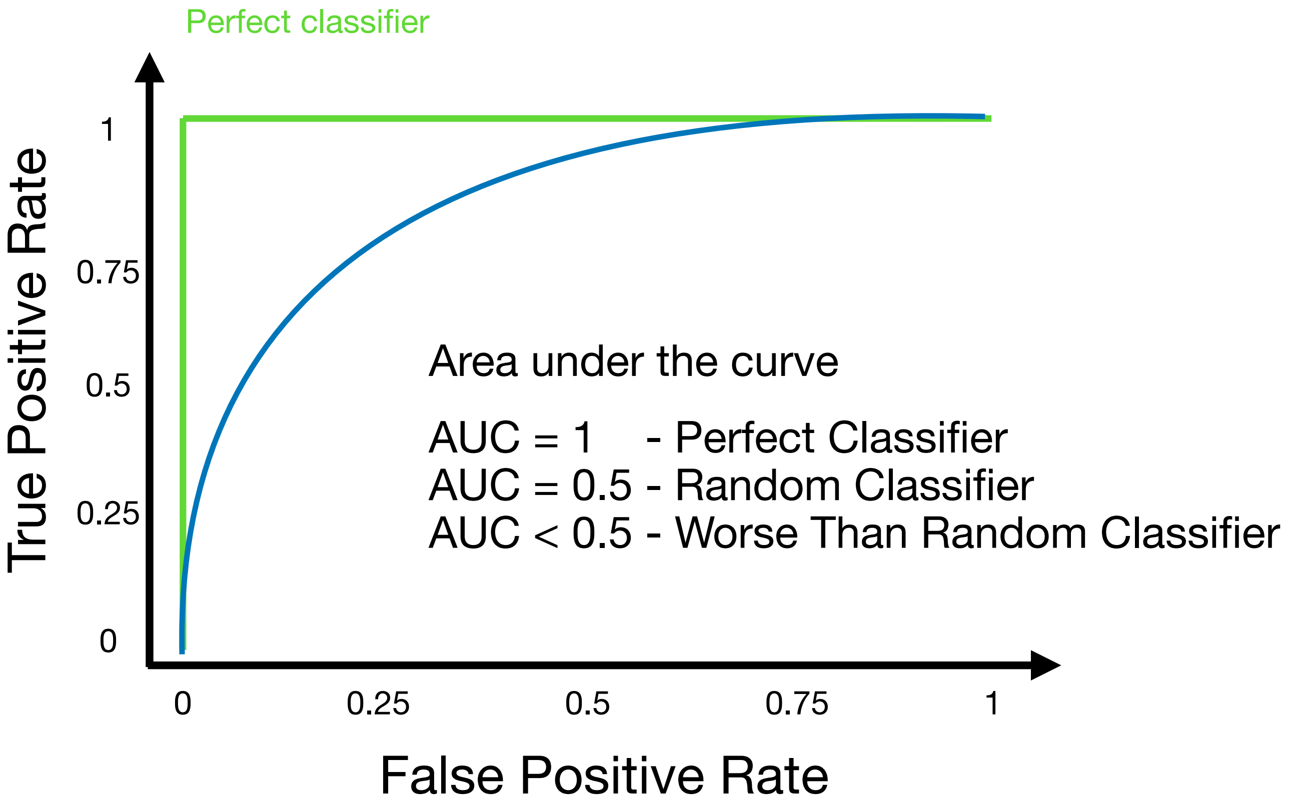
```
for threshold in np.arange(0, 1, 0.1):
    y = compute_precision(class, thresh)
    x = compute_recall(class, thresh)
plot(x, y)
```

Metrics

AUC-ROC Curve

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

Intuition:
AUC equals the probability that a randomly chosen positive instance is ranked **ranked higher** than a randomly chosen negative instance by the classifier's scores.



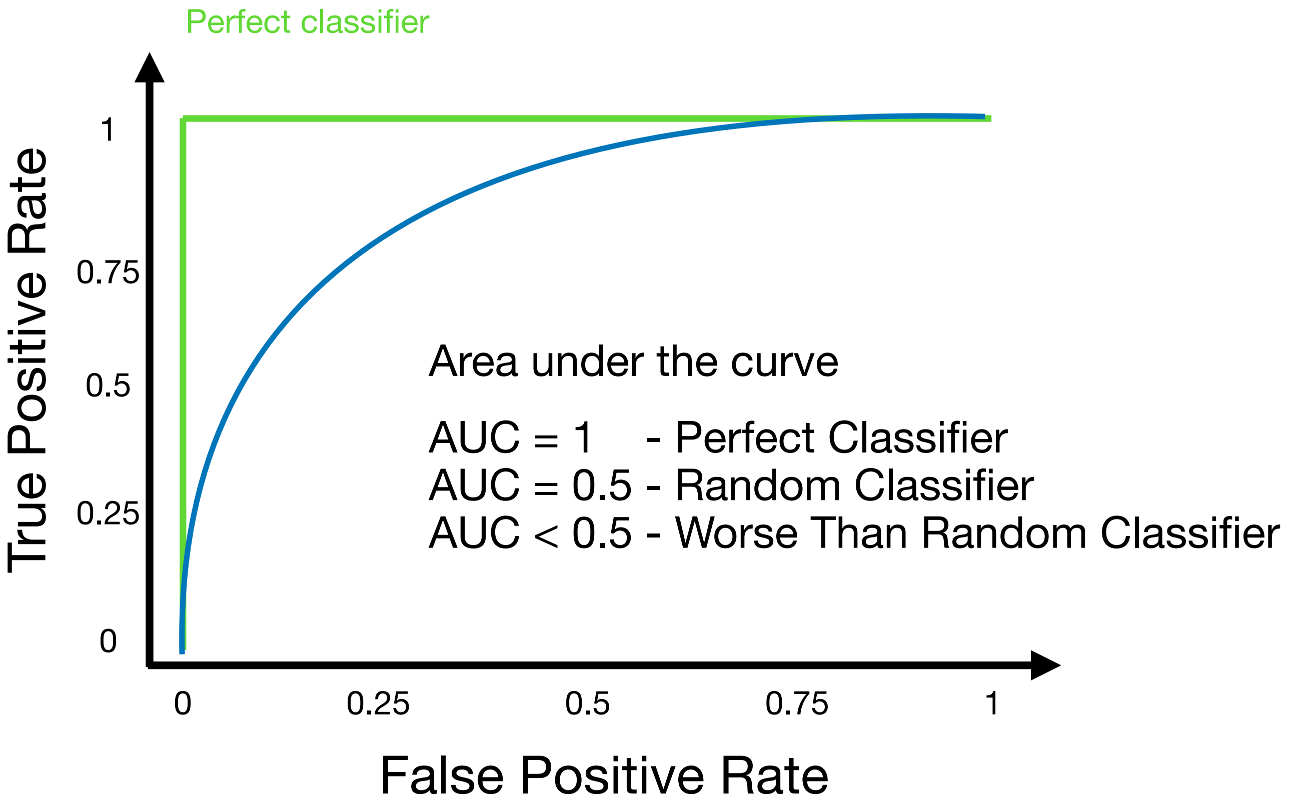
Metrics

AUC-ROC Curve

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

Intuition:
AUC equals the probability that a randomly chosen positive instance is ranked **ranked higher** than a randomly chosen negative instance by the classifier's scores.

Limitation:
ROC curves can be overly optimistic for **highly imbalanced datasets** because the **FPR denominator** is dominated by the large TN count.



Today’s Outline

- Metrics
- **k-Nearest Neighbors**

k-Nearest Neighbors

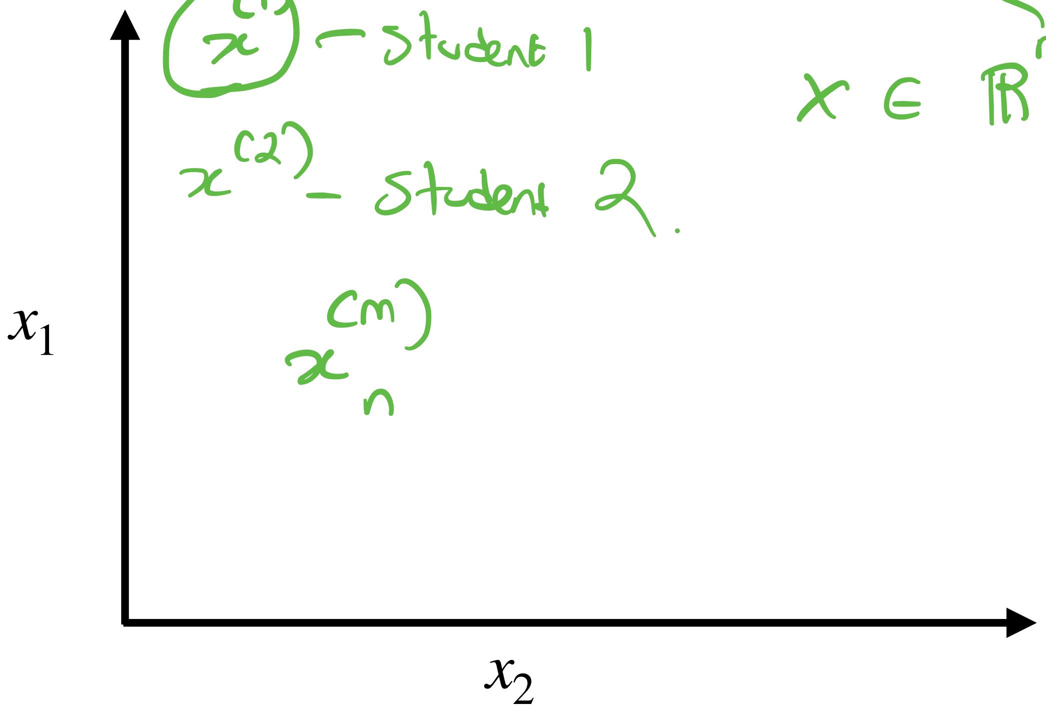
- KNN is a **non-parametric**, instance-based (lazy) learning algorithm.
- It makes no assumptions about the underlying data distribution and stores all training instances **rather than learning explicit parameters**.

k-Nearest Neighbors

- KNN is a **non-parametric**, instance-based (lazy) learning algorithm.
- It makes no assumptions about the underlying data distribution and stores all training instances **rather than learning explicit parameters**.
- **Key Idea**:
  - Similar instances have similar labels.
  - To classify a new point, find the **K training instances closest to it** and let them vote

k-Nearest Neighbors

	High School GPA x_1	SAT Scores x_2	Get Into College? y
$x^{(1)}$	3.6	1500	★ = 1
$x^{(2)}$	2.7	950	◆ = 0
$x^{(3)}$	3.7	1300	★ = 1
$x^{(4)}$	3.2	1550	★ = 1
$x^{(5)}$	3.2	1000	◆ = 0
$x^{(new)}$	**3.2**	**1250**	**?**



x_1 x_2 x_3 # training data # features

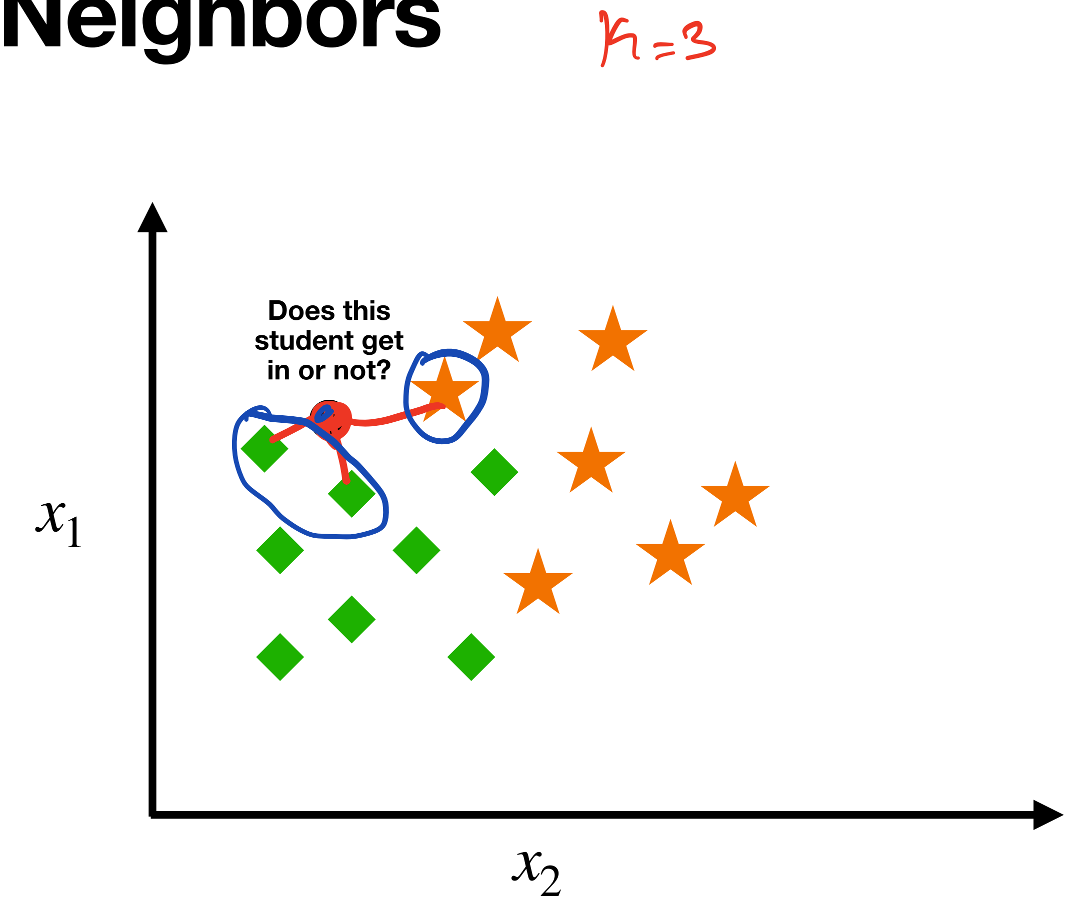
$x^{(1)}$ – student 1

$x^{(2)}$ – student 2.

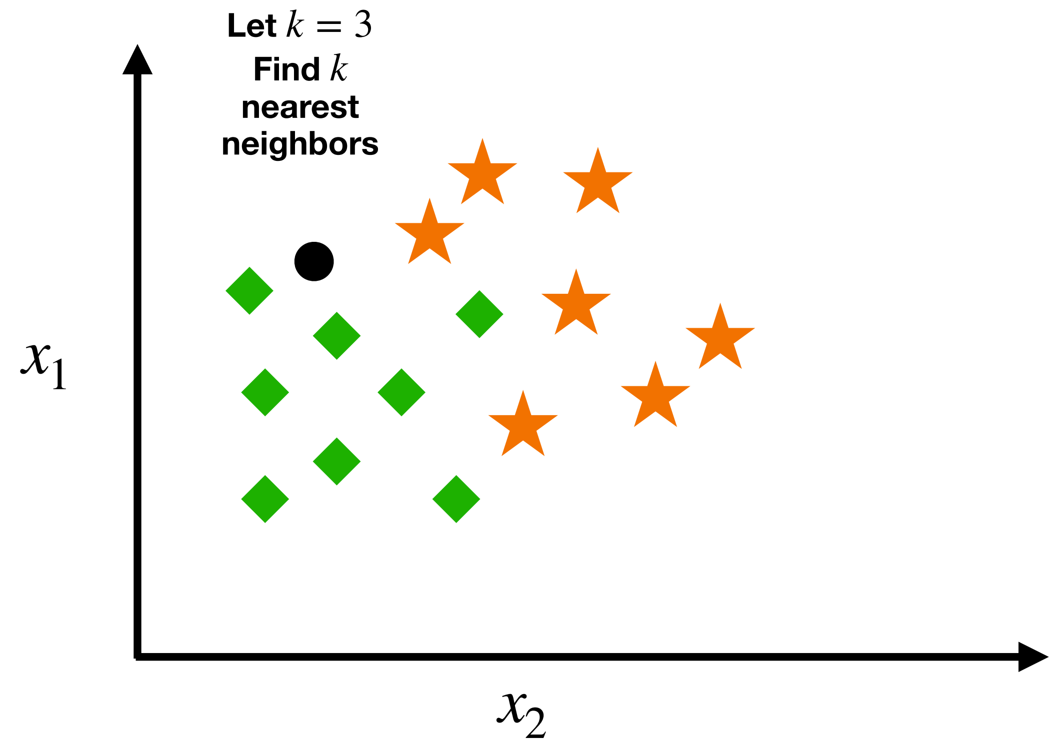
$x_n^{(m)}$

$X \in \mathbb{R}^{m \times n}$

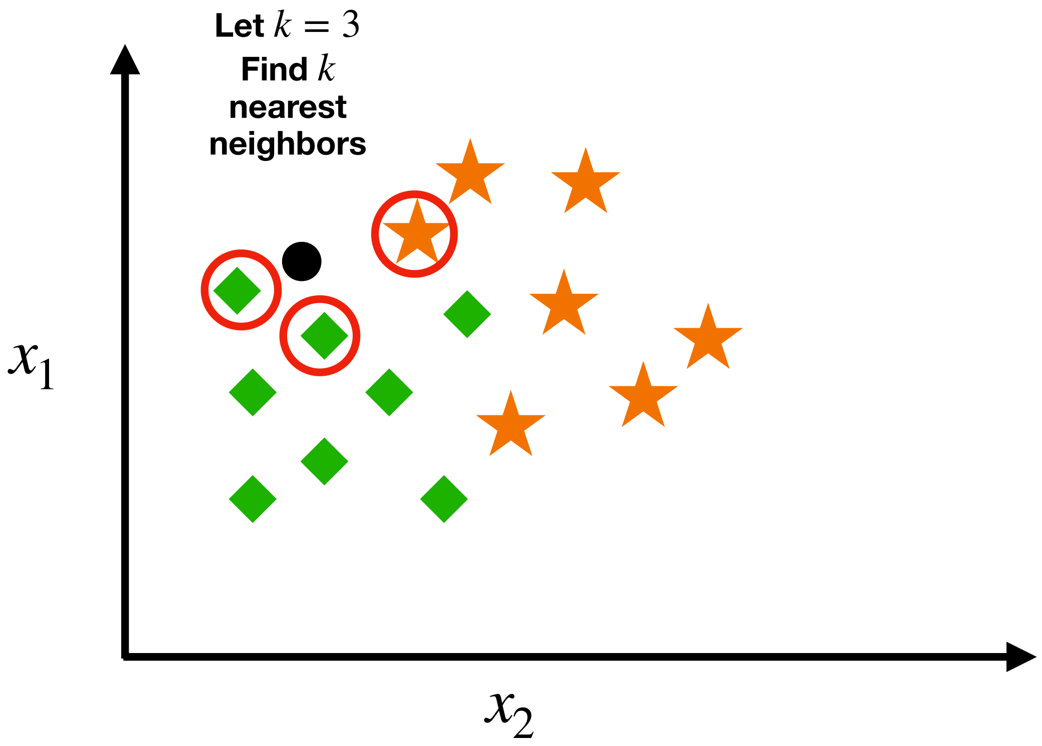
k-Nearest Neighbors



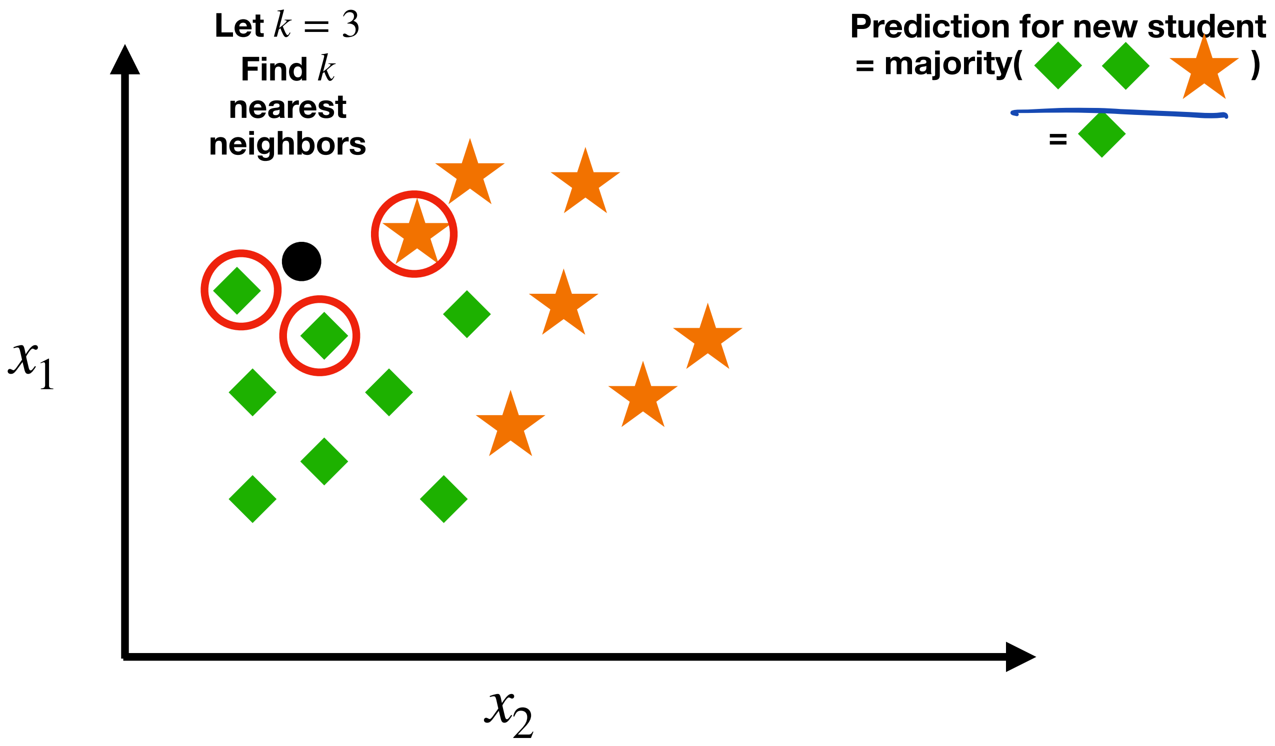
k-Nearest Neighbors



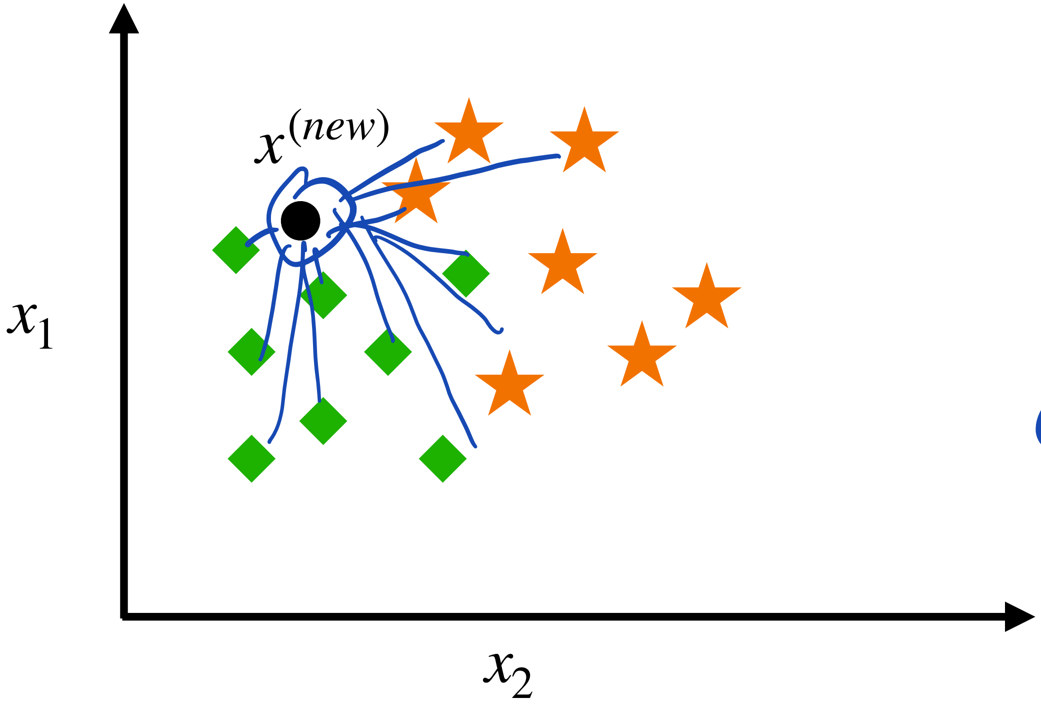
k-Nearest Neighbors



k-Nearest Neighbors



k-Nearest Neighbors



Algorithm:

Training Phase:
Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

Prediction/Testing Phase:
1. Compute distance from new point $x^{(new)}$ to every other point in the training data
2. Select the top $k-$nearest neighbors
3. For classification, return majority class amongst top k
4. For regression, return mean or median of the values of the $k-$neighbors

k-Nearest Neighbors

	High School GPA x_1	SAT Scores x_2	Get Into College? y
$x^{(1)}$	3.6	1500	★ = 1
$x^{(2)}$	2.7	950	◆ = 0
$x^{(3)}$	3.7	1300	★ = 1
$x^{(4)}$	3.2	1550	★ = 1
$x^{(5)}$	3.2	1000	◆ = 0
$x^{(new)}$	[illegible]	[illegible]	?

0

1500

Algorithm:

<u>Training Phase:</u>

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

<u>Prediction/Testing Phase:</u>

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

2. Select the top $k-$nearest neighbors

3. For classification, return <u>majority class</u> amongst top k

4. For regression, return <u>mean or median</u> of the values of the $k-$neighbors

k-Nearest Neighbors

Euclidean Distance (L_2 Norm):

$$d(x^{(new)}, x^{(i)}) = \sqrt{\sum_{j=0}^{n} (x_j^{(new)} - x_j^{(i)})^2}$$

Manhattan Distance (L_1 Norm):

$$d(x^{(new)}, x^{(i)}) = \sum_{j=0}^{n} | x_j^{(new)} - x_j^{(i)} |$$

Cosine Similarity:

$$sim(x^{(new)}, x^{(i)}) = \frac{x^{(new)} \cdot x^{(i)}}{\|x^{(new)}\| \|x^{(i)}\|}$$

$$(distance = 1 - sim(x^{(new)}, x^{(i)}))$$

The choice of the distance metric fundamentally affects which points are being considered "neighbors"

Algorithm:

Training Phase:

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

Prediction/Testing Phase:

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

2. Select the top $k-$nearest neighbors

3. For classification, return majority class amongst top k

4. For regression, return mean or median of the values of the $k-$neighbors

k-Nearest Neighbors

Euclidean Distance (L_2 Norm):

Most common choice, but sensitive to **feature scales**

$$d(x^{(new)}, x^{(i)}) = \sqrt{\sum_{j=0}^{n} (x_j^{(new)} - x_j^{(i)})^2}$$

Manhattan Distance (L_1 Norm):

$$d(x^{(new)}, x^{(i)}) = \sum_{j=0}^{n} |x_j^{(new)} - x_j^{(i)}|$$

Cosine Similarity:

$$sim(x^{(new)}, x^{(i)}) = \frac{x^{(new)} \cdot x^{(i)}}{||x^{(new)}||\,||x^{(i)}||}$$

$$(distance = 1 - sim(x^{(new)}, x^{(i)}))$$

Algorithm:

Training Phase:

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

Prediction/Testing Phase:

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

2. Select the top $k-$nearest neighbors

3. For classification, return majority class amongst top k

4. For regression, return mean or median of the values of the $k-$neighbors

k-Nearest Neighbors

Euclidean Distance (L_2 Norm):

$$d(x^{(new)}, x^{(i)}) = \sqrt{\sum_{j=0}^{n} (x_j^{(new)} - x_j^{(i)})^2}$$

Manhattan Distance (L_1 Norm):

$$d(x^{(new)}, x^{(i)}) = \sum_{j=0}^{n} | x_j^{(new)} - x_j^{(i)} |$$

Cosine Similarity:

$$sim(x^{(new)}, x^{(i)}) = \frac{x^{(new)} \cdot x^{(i)}}{\|x^{(new)}\| \|x^{(i)}\|}$$

$$(distance = 1 - sim(x^{(new)}, x^{(i)}))$$

Sum of **absolute** differences.
More **robust to outliers** than Euclidean.
Appropriate when features represent fundamentally different quantities.

Algorithm:

Training Phase:

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

Prediction/Testing Phase:

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

2. Select the top $k-$nearest neighbors

3. For classification, return majority class amongst top k

4. For regression, return mean or median of the values of the $k-$neighbors

k-Nearest Neighbors

Euclidean Distance (L_2 Norm):

$$d(x^{(new)}, x^{(i)}) = \sqrt{\sum_{j=0}^{n} (x_j^{(new)} - x_j^{(i)})^2}$$

Manhattan Distance (L_1 Norm):

$$d(x^{(new)}, x^{(i)}) = \sum_{j=0}^{n} |x_j^{(new)} - x_j^{(i)}|$$

Cosine Similarity:

$$sim(x^{(new)}, x^{(i)}) = \frac{x^{(new)} \cdot x^{(i)}}{\|x^{(new)}\| \|x^{(i)}\|}$$

$$(distance = 1 - sim(x^{(new)}, x^{(i)}))$$

Measures **angle** between vectors, **ignoring magnitude**.
Useful for text data and high-dimensional sparse vectors.

Algorithm:

Training Phase:

Store all training instances (x_{train}, y_{train})
No computation required. We are not learning any parameters

Prediction/Testing Phase:

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

2. Select the top $k-$nearest neighbors

3. For classification, return majority class amongst top k

4. For regression, return mean or median of the values of the $k-$neighbors

k-Nearest Neighbors

Hamming Distance:

$$d(x^{(new)}, x^{(i)}) = \sum_{j=0}^{n} 1 \cdot (x_j^{(new)} \neq x_j^{(i)})$$

Hamming distance is a metric for comparing sequences of equal length - it counts the **number of positions where corresponding elements differ**.

Example:

$d([red, small, round], [red, large, round]) = 1$

$d([cat, young, male], [dog, old, female]) = 3$

$d(ACGT, ACTT) = 1$

Use Cases:

Categorical features: When features are categorical (say state a person lives in), Euclidean distance is meaningless.

Hamming distance treats each feature as equal - either it matches or it doesn't.

Algorithm:

<u>Training Phase:</u>

Store all training instances (x_{train}, y_{train})

No computation required. We are not learning any parameters

<u>Prediction/Testing Phase:</u>

1. Compute **distance** from new point $x^{(new)}$ to every other point in the training data

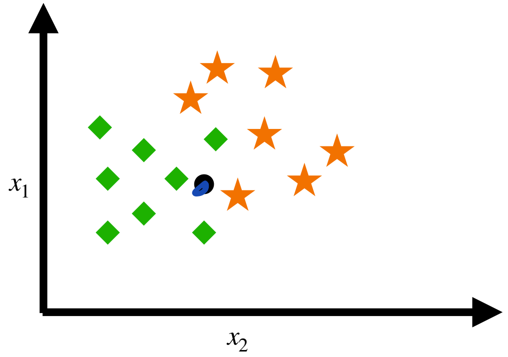
2. Select the top $k-$nearest neighbors

3. For classification, return <u>majority class</u> amongst top k

4. For regression, return <u>mean or median</u> of the values of the $k-$neighbors

k-Nearest Neighbors

Choosing k



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

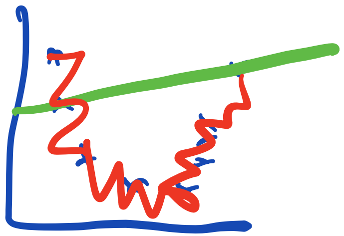
Small k (e.g. $k = 1$)

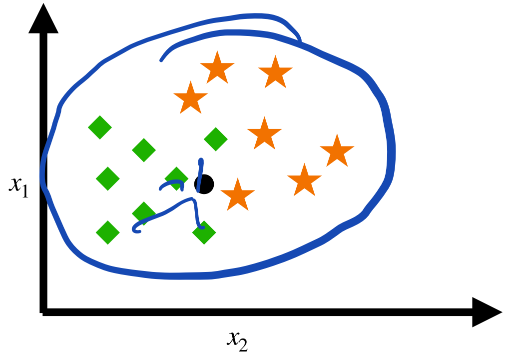
- High variance, low bias
- Decision boundary is highly irregular
- Very sensitive to noise and outliers
- Prone to overfitting, but can capture fine grained structure

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

k-Nearest Neighbors

Choosing k





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

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

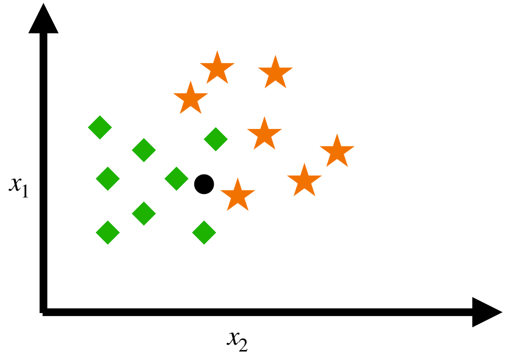
- High variance, low bias
- Decision boundary is highly irregular
- Very sensitive to noise and outliers
- Prone to overfitting, but can capture fine grained structure

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

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

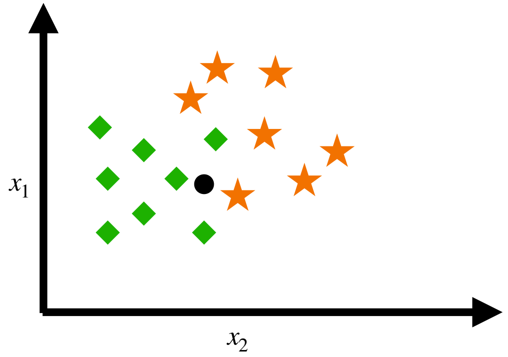
Practical Tips

- 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

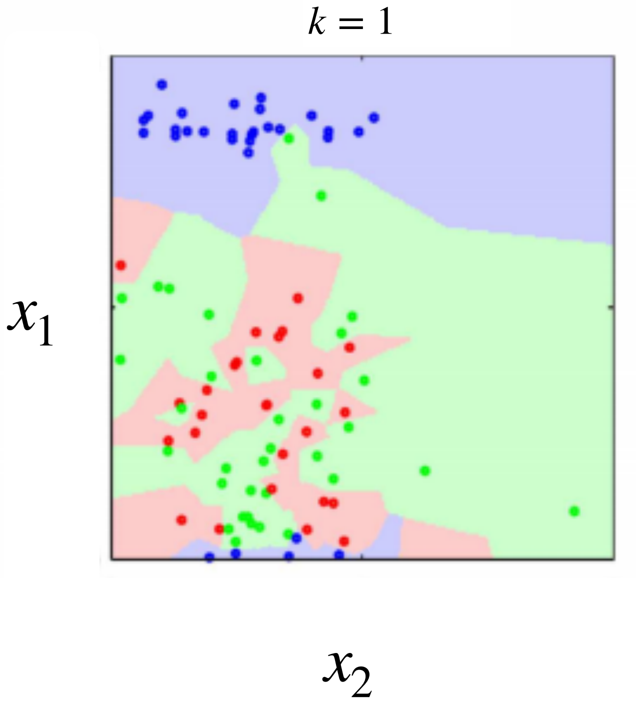
Large k (e.g. $k = m$)

- 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

k-Nearest Neighbors
Choosing k

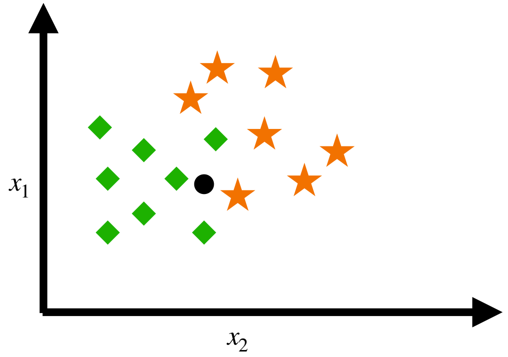


- k is the primary hyper-parameter controlling the bias-variance tradeoff

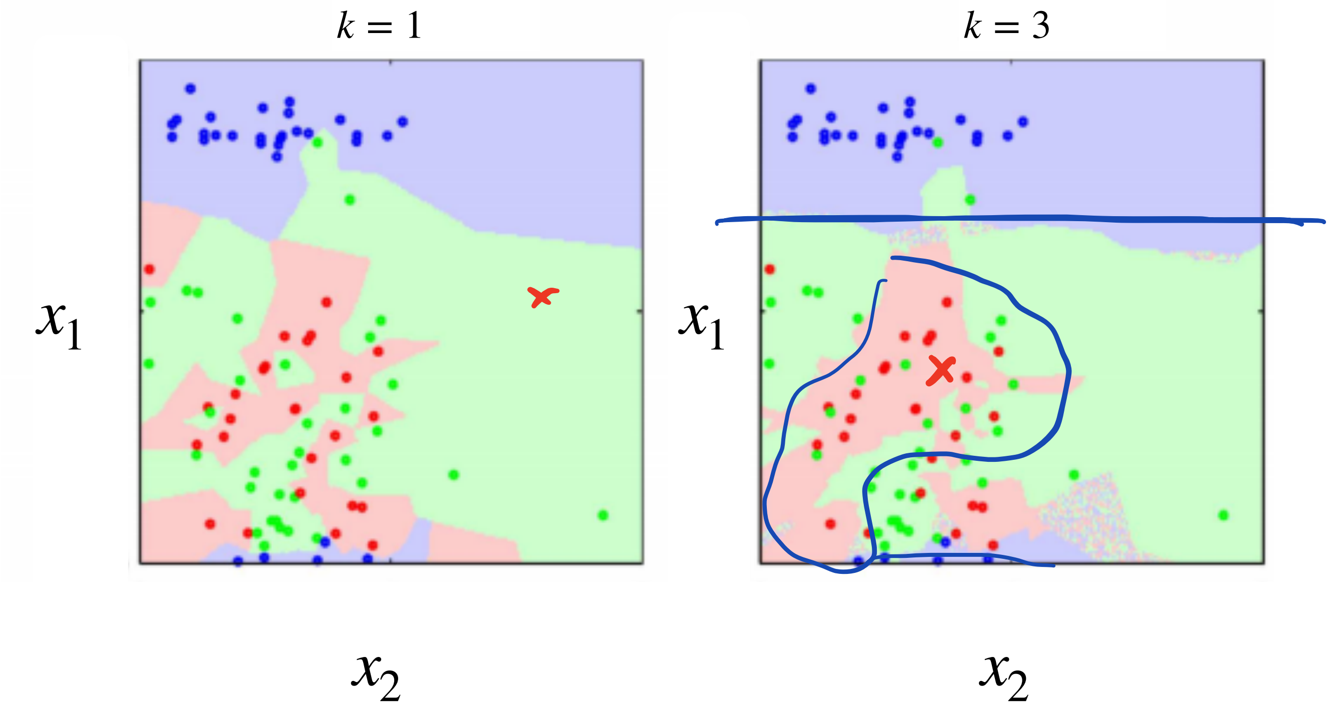


k-Nearest Neighbors

Choosing k

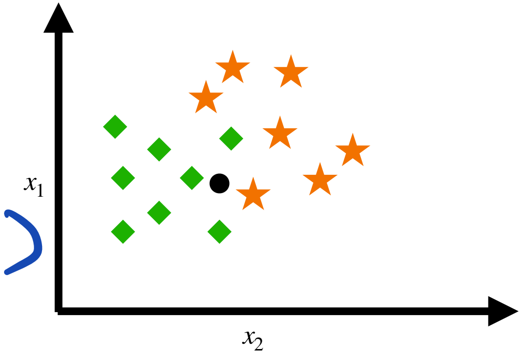


- k is the primary hyper-parameter controlling the bias-variance tradeoff

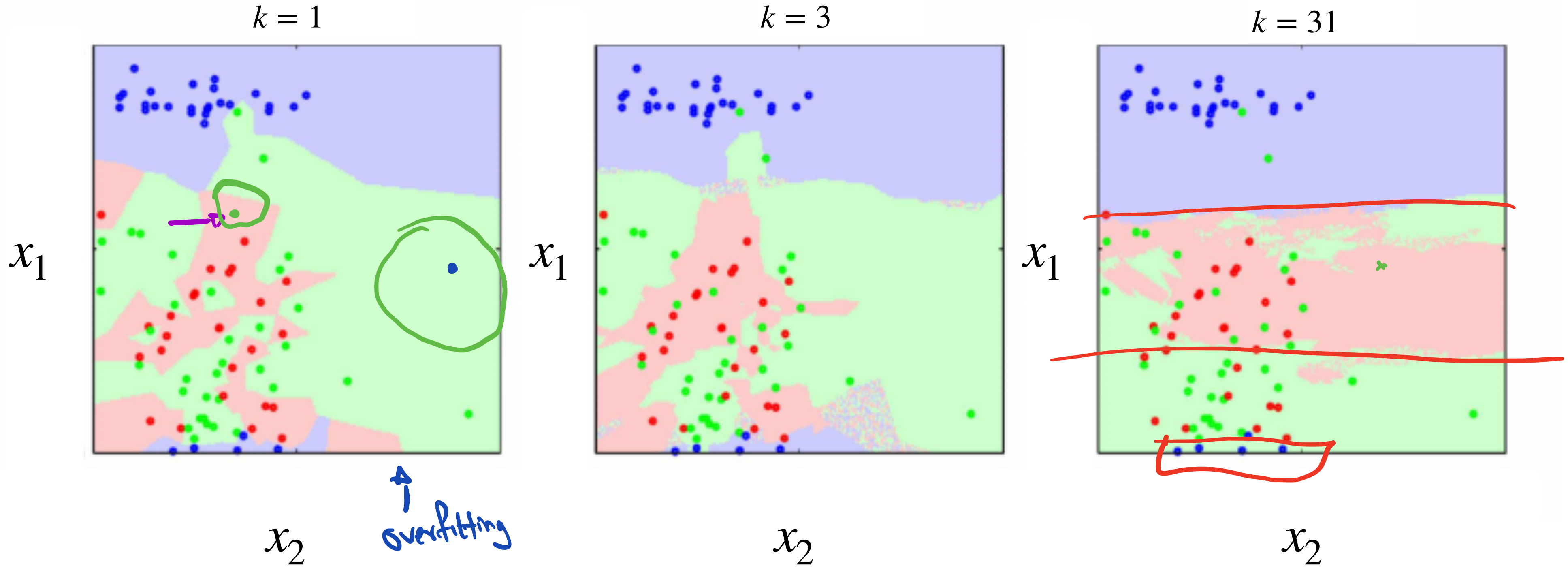


k-Nearest Neighbors

Choosing k



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



k-Nearest Neighbors

Choosing k - Cross-validation

- Why Not Just Use Training Error?
  - A model that memorizes the training data achieves zero training error but fails on new data.
  - Training error is a biased (optimistic) estimate of true generalization performance.
  - We need to estimate how well our model will perform on **unseen data**.

k-Nearest Neighbors

Choosing k - Cross-validation

- Naive Solution - Train/Test Split
  - Split data into training set (say 80%) and test set (20%).
  - Train on training set, evaluate on test set.
  - Issues:
    - Wastes data - 20% of precious labeled data is never used for training
    - High variance - Performance estimate depends heavily on **which points land in the test set**
    - No hyperparameter tuning: If we use the test set to select hyperparameters, we're overfitting to the test set - (using **validation set** is a possible fix for this issue)

k-Nearest Neighbors

Choosing k - Cross-validation

- Naive Solution - Train/Test Split
  - **Data Leakage Issue**
    - If we repeatedly evaluate on the test set while tuning hyperparameters, information about the test set **leaks** into our model selection process.
    - The test error becomes optimistically biased - no longer a valid estimate of **generalization**

k-Nearest Neighbors

Choosing k - Cross-validation

- **Solution!**
- Use cross-validation
  - Use **all data** for both training and validation
  - Get **reliable performance estimates** with uncertainty quantification
  - Select hyperparameters **without contaminating the final test set**

k-Fold Cross Validation

Algorithm

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

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

	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

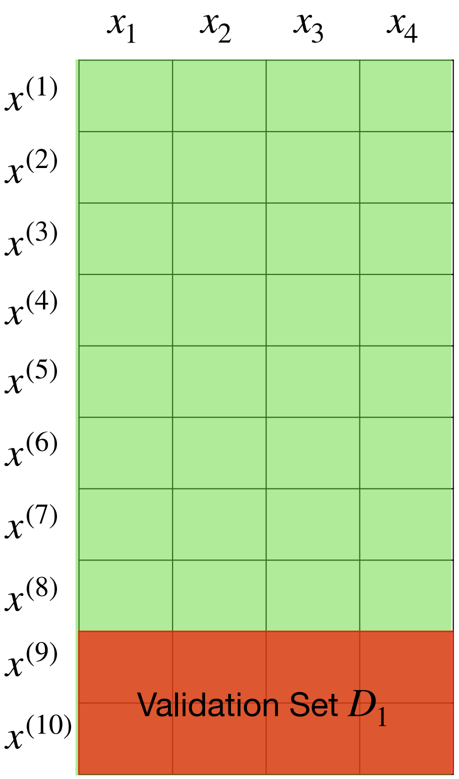
    3c. Train the model on the training set

    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

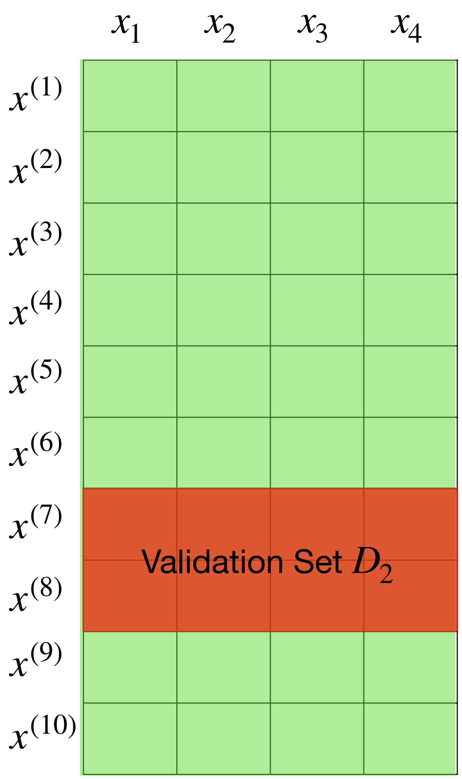
    3c. Train the model on the training set

    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

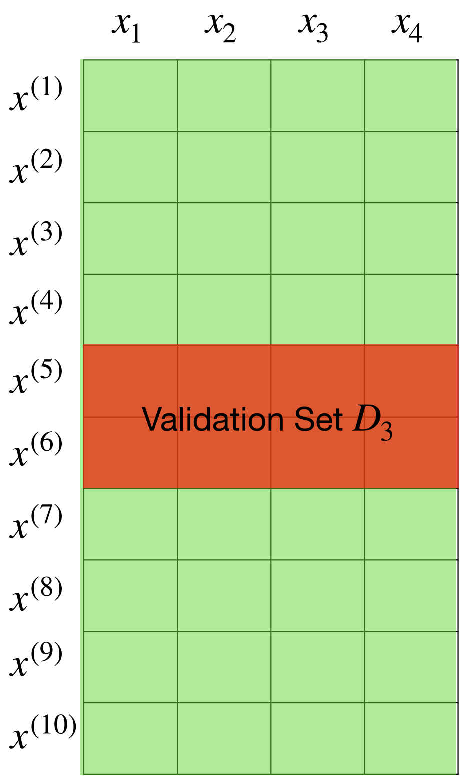
   3c. Train the model on the training set

   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

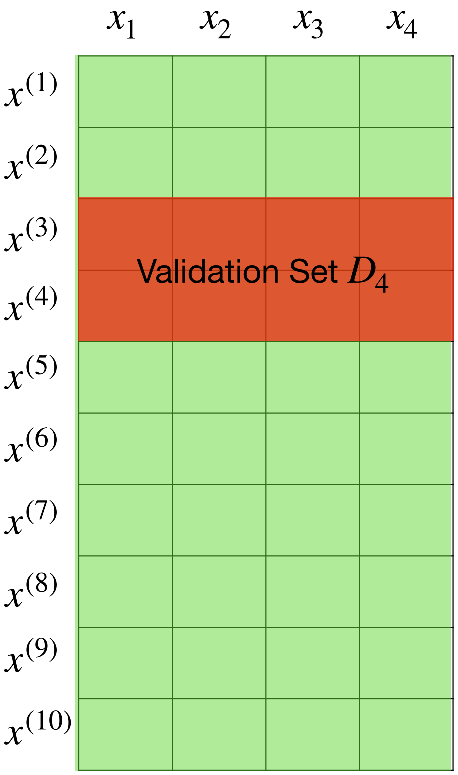
    3c. Train the model on the training set

    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

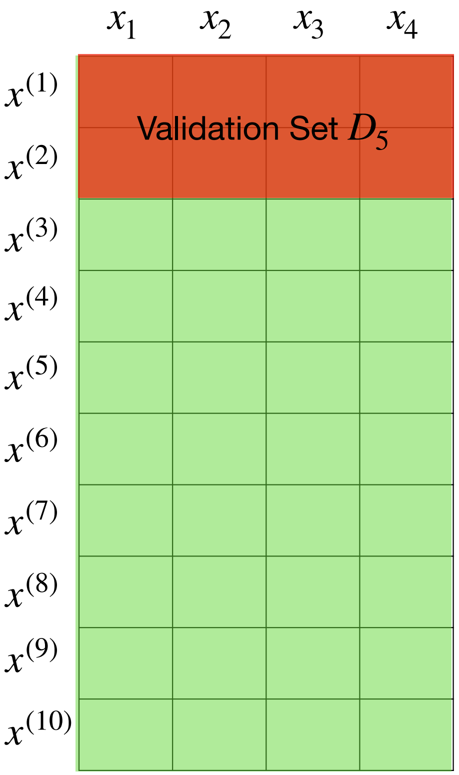
   3c. Train the model on the training set

   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

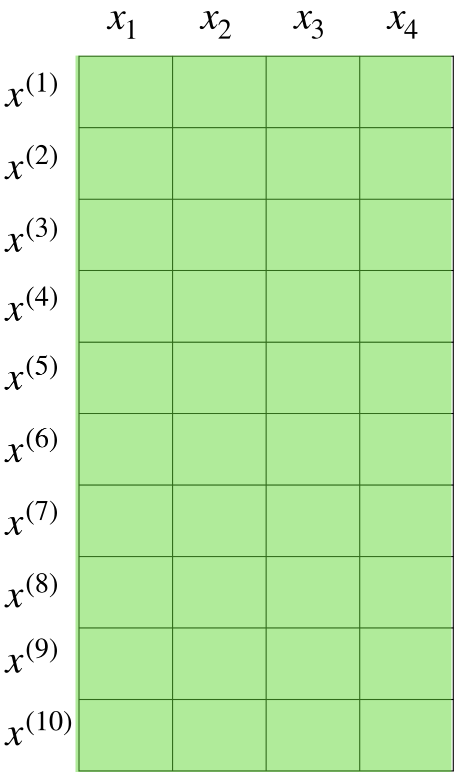
    3c. Train the model on the training set

    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

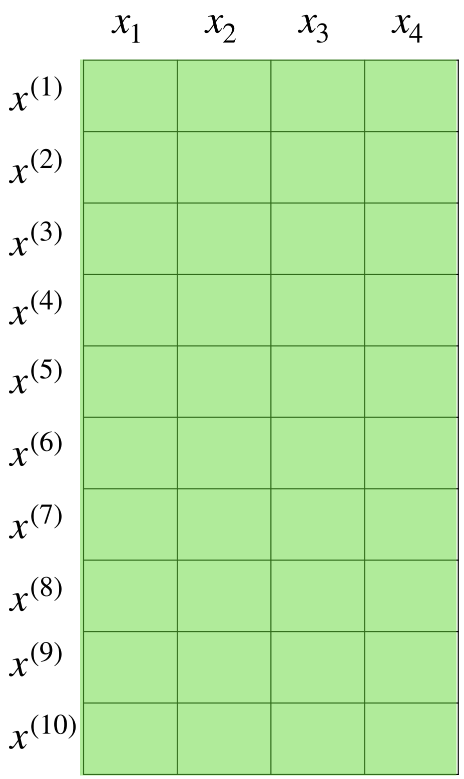
   3c. Train the model on the training set

   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**

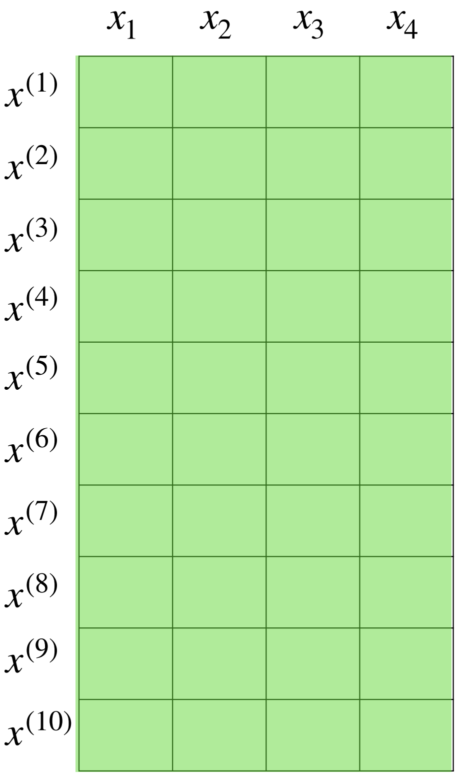
Mean CV Score:

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

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:

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

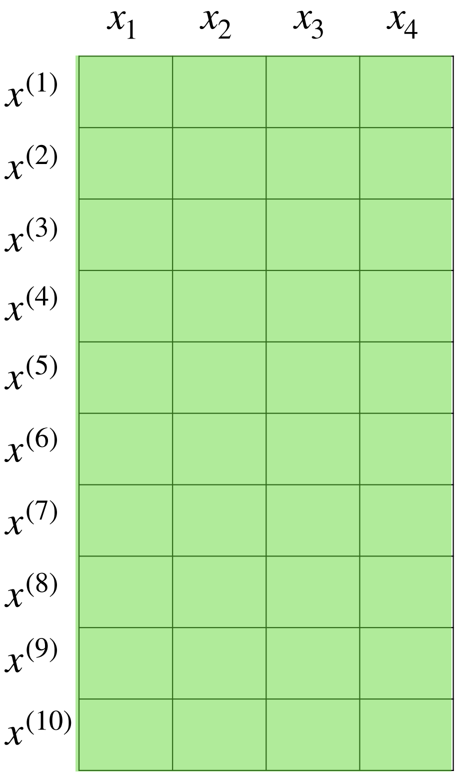
k-fold CV requires **training k models.**

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

Back to k-Nearest Neighbors

Practical Issues

- Feature Scaling
- Curse of Dimensionality
- Space and computational complexity

Back to k-Nearest Neighbors

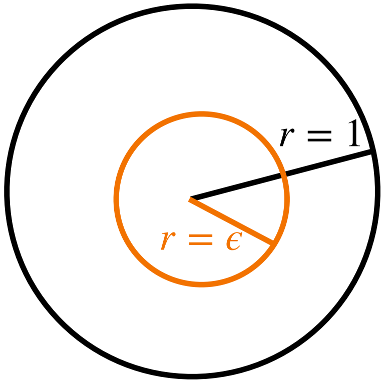
Practical Issues - Feature Scaling

- KNN is **highly sensitive** to feature scales because distance metrics are dominated by features with larger ranges.
- Example:
    - If feature A ranges from 0-1 and feature B ranges from 0-1000
    - Euclidean distance is almost **entirely determined by feature B**.
    - Solution: Always normalize or standardize features before applying kNN.

Back to k-Nearest Neighbors

Practical Issues - Curse of Dimensionality

- KNN suffers severely in high-dimensional spaces:
  - Distance concentration: As dimensionality increases, **distances between points become increasingly similar**.
  - **The ratio of nearest to farthest neighbor approaches 1**, making the concept of “nearest” meaningless.



Back to k-Nearest Neighbors

Practical Issues - Curse of Dimensionality

- KNN suffers severely in high-dimensional spaces:
  - Distance concentration: As dimensionality increases, **distances between points become increasingly similar**.
  - **The ratio of nearest to farthest neighbor approaches 1**, making the concept of “nearest” meaningless.
- Sparsity: The volume of space grows exponentially with dimension. To maintain the same density of points, training set size must grow exponentially.
- Irrelevant features: In high dimensions, many features may be irrelevant, adding noise to distance calculations.
- Mitigation strategies:
  - Dimensionality reduction (PCA, feature selection)
  - **Feature weighting** based on relevance
  - Consider other algorithms for $d > 20$

Back to k-Nearest Neighbors

Practical Issues - Computational Complexity

- Training: $O(1)$ - just store the data
- Prediction (naive):
  - $O(nm)$ per query, where m is training set size and n is dimensionality.
  - Must compute distance to all m points.
- Prediction (optimized) - Data structures can accelerate nearest neighbor search:
  - KD-trees: $O(nlogm)$ average case for low dimensions, but degrades to $O(nm)$ in high dimensions
  - Ball trees: Better for high dimensions than KD-trees
  - Locality-sensitive hashing (LSH): Approximate nearest neighbors in $O(n)$ with preprocessing
- Space complexity: $O(nm)$ to store the training data.

Back to k-Nearest Neighbors

Practical Issues

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

Back to 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

Next Class

- Logistic Regression
  - Brush up on conditional probability, Bayes' Theorem and Bernoulli Distribution