Fondamenti di Analisi dei Dati

from data analysis to predictive techniques

Prof. Antonino Furnari (antonino.furnari@unict.it)
Corso di Studi in Informatica
Dip. di Matematica e Informatica
Università di Catania

Università di Catania

Classification Task and K-Nearest Neighbor

Classification models predict discrete class labels from input data. Unlike regression which outputs continuous values, classification assigns observations to predefined categories—from spam detection to medical diagnosis.

Classification Task Definition

Mathematical Framework

A classification model is a function

$$h : \mathbb{R}^n \to \{0, \ldots, M-1\}$$

that maps numerical input vectors to Mdiscrete class labels.

Given training data $D = \{(x_i, y_i)\}_{i=1}^{N}$, we minimize empirical risk using a loss function that counts incorrect predictions. For example:

$$L(\hat{y}, y) = \begin{cases} 1 & \text{if} \quad \hat{y} \neq y \\ 0 & \text{if} \quad \hat{y} = y \end{cases}$$

Empirical Risk in this case:

$$R_{emp}(h) = \frac{1}{N} \sum_{i=1}^{N} L(\hat{y}_i, y_i) = \frac{\text{number of incorrect predictions}}{N}$$

Key Applications

- Spam email detection (binary classification)
- Social media content categorization
- Object recognition in images (multi-class)
- Medical diagnosis and screening

Evaluation Measures: Accuracy & Error Rate

Evaluation measures are critical tools for machine learning. They guide model training by providing feedback on performance, help tune hyperparameters for optimal results, and ultimately assess how well an algorithm generalizes to unseen data.

$$Y_{TE} = \left\{ y^{(i)} | \left(\mathbf{x}^{(i)}, y^{(i)} \right) \in TE \right\}_i$$

$$\widehat{Y}_{TE} = \left\{ h\left(\mathbf{x}^{(i)} \right) | \left(\mathbf{x}^{(i)}, y^{(i)} \right) \in TE \right\}_i$$

Accuracy

Accuracy quantifies the percentage of test examples for which our algorithm has predicted the correct label. It's a straightforward metric showing overall correctness.

$$Accuracy\left(Y_{TE}, \widehat{Y}_{TE} \right) = \frac{\left| \left\{ y^{(i)} : y^{(i)} = \hat{y}^{(i)} \right\} \right|}{|Y_{TE}|}$$

For example, if a model correctly predicts 70 out of 100 test examples, its accuracy is 0.7 or 70%.

Error Rate

The error rate is the proportion of incorrect predictions. It's directly related to accuracy and represents the complement of correctness.

$$ErrorRate\left(Y_{TE}, \widehat{Y}_{TE} \right) = \frac{\left| \left\{ y^{(i)} : y^{(i)} \neq \hat{y}^{(i)} \right\} \right|}{|Y_{TE}|} = 1 - Accuracy\left(Y_{TE}, \widehat{Y}_{TE} \right)$$

Using the previous example, an accuracy of 70% implies an error rate of 30%.

The Challenge of Imbalanced Datasets

Consider a dataset with 10,000 entries:

- 500 from Class 0
- 9,500 from Class 1

A naive classifier that simply predicts "Class 1" for every input ($f(x) = 1$) achieves an accuracy of 95% (9,500 correct predictions out of 10,000).

While numerically high, this model is practically useless as it fails to identify any instances of Class 0. **This highlights how accuracy alone can be misleading in scenarios with imbalanced class distributions.**

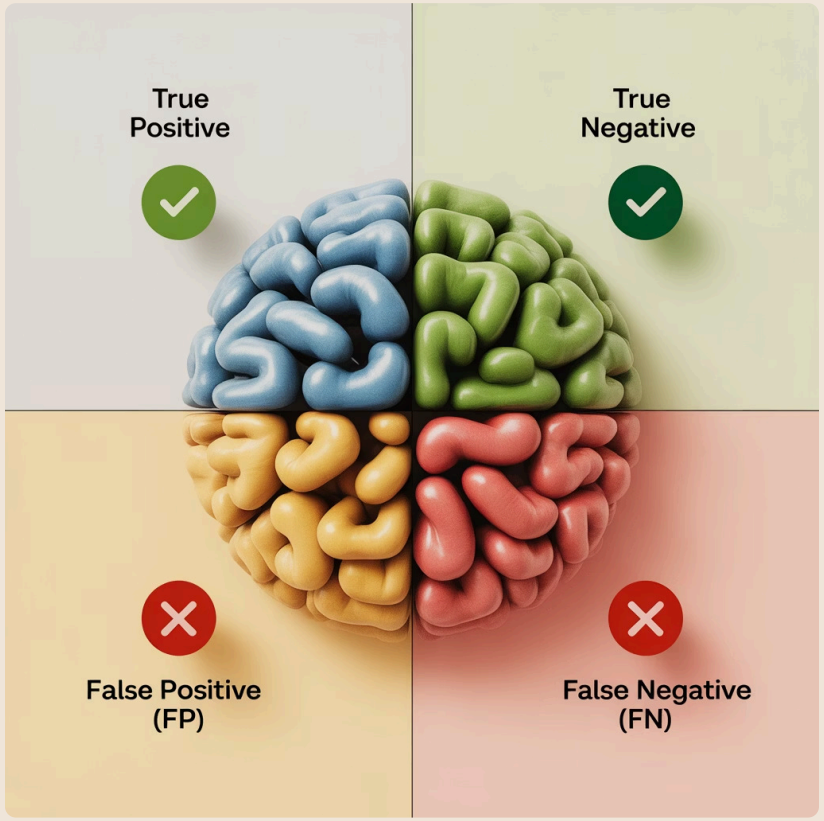
Understanding Error Types

Classification errors fall into two categories that reveal different failure modes:

- **False Positives (Type 1):** Classifying a negative example as positive
- **False Negatives (Type 2):** Classifying a positive example as negative

Correct predictions include:

- **True Positives:** Correctly identified positive examples
- **True Negatives:** Correctly identified negative examples



The Confusion Matrix

The confusion matrix provides a complete diagnostic view of classifier performance, showing exactly where predictions succeed and fail.

Confusion Matrix

CONFUSION MATRIX		PREDICTED LABELS	
		POSITIVE	NEGATIVE
TRUE LABELS	POSITIVE	TP	FN
	NEGATIVE	FP	TN

A strong classifier has large numbers on the diagonal (TP and TN) and small numbers off-diagonal (FP and FN).

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

Spam Detector Example

CONFUSION MATRIX		PREDICTED LABELS	
		SPAM	LEGITIMATE
TRUE LABELS	SPAM	40	10
	LEGITIMATE	20	30

Accuracy: 70%. The matrix reveals 20 legitimate emails wrongly flagged as spam.

$$Accuracy = \frac{40 + 30}{40 + 10 + 20 + 30} = \frac{70}{100} = 0.7$$

Imbalanced Dataset

CONFUSION MATRIX		PREDICTED LABELS	
		POSITIVE	NEGATIVE
TRUE LABELS	POSITIVE	9500	0
	NEGATIVE	500	0

$$Accuracy = \frac{9500}{9500 + 500} = 0.95$$

Even if we get a good accuracy, a glance at the confusion matrix reveal that predictions are imbalanced.

Precision and Recall

- **Precision** measures **how many of the examples which have been classified as positives were actually positives**
- **Recall** measures **how many of the examples which are positives, have been correctly classified as positives**

CONFUSION MATRIX		PREDICTED LABELS	
		POSITIVE	NEGATIVE
TRUE LABELS	POSITIVE	TP	FN
	NEGATIVE	FP	TN

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

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

Precision vs Recall Trade-offs

When to Prioritize Precision

Spam Filter: If an email is marked spam, it must truly be spam. False positives (blocking legitimate emails) are unacceptable.

Accept lower recall—some spam getting through is tolerable if we never block important messages.

When to Prioritize Recall

Medical Screening: If a patient has a disease, we must detect it. False negatives (missing sick patients) are dangerous.

Accept lower precision—false positives can be caught by follow-up tests, but missing a diagnosis is catastrophic.

Spam Detector Example

CONFUSION MATRIX		PREDICTED LABELS	
		SPAM	LEGITIMATE
TRUE LABELS	SPAM	40	10
	LEGITIMATE	20	30

We can compute the following precision and recall values:

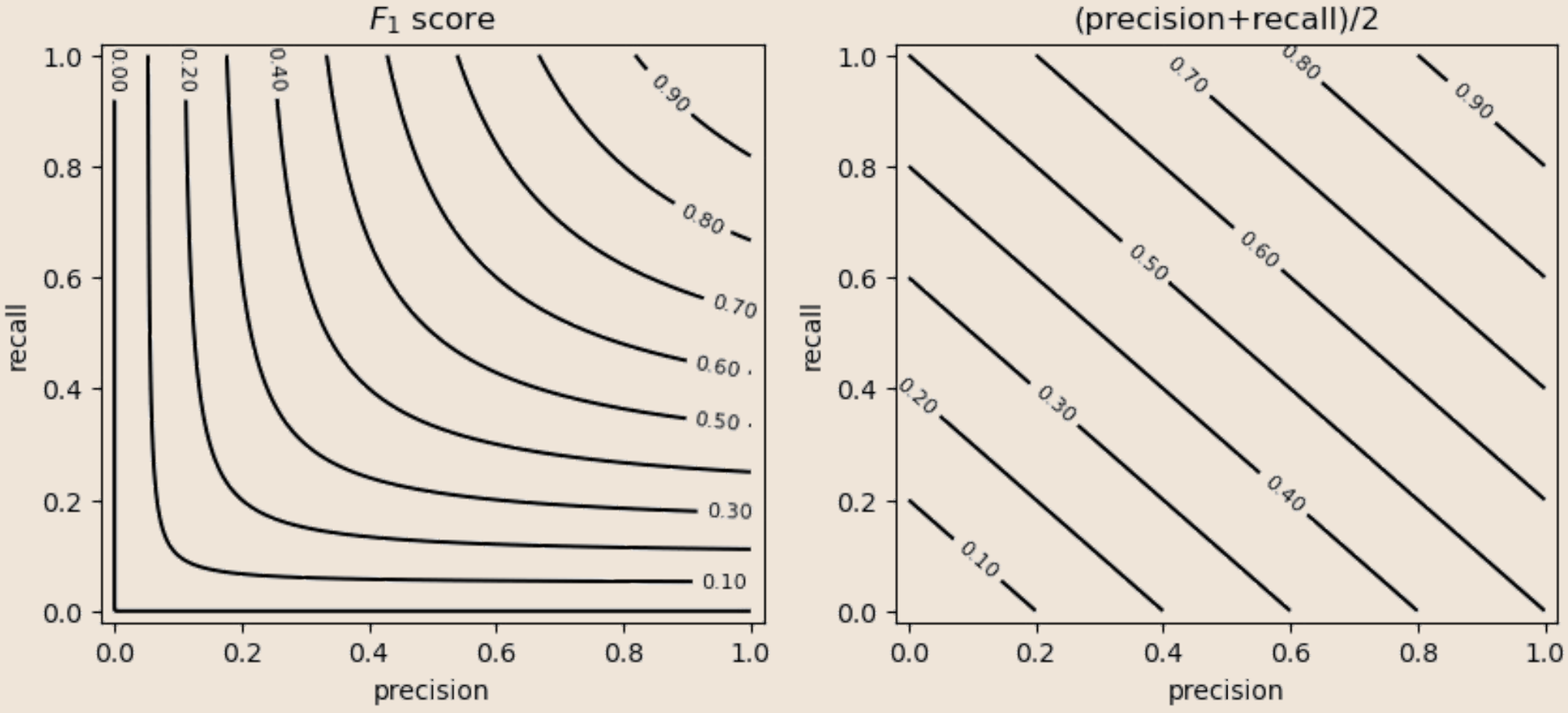
- $Precision = \frac{40}{20+30} = 0.8$
- $Recall = \frac{40}{40+20} = 0.67$

F_1 Score: Balancing Precision and Recall

The F_1 score provides a single metric that requires both high precision and high recall through the harmonic mean:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The harmonic mean penalizes imbalanced scores. To achieve a high F_1, both precision and recall must be strong—you cannot "cheat" by maximizing just one.



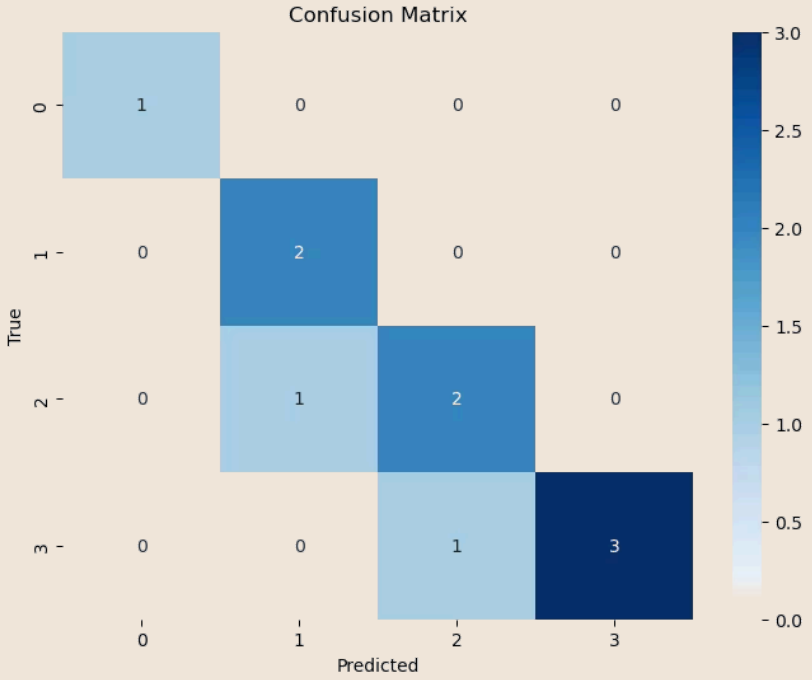
Spam Detector Example

CONFUSION MATRIX		PREDICTED LABELS	
		SPAM	LEGITIMATE
TRUE LABELS	SPAM	40	10
	LEGITIMATE	20	30

We can compute the following precision and recall values:

- $Precision = \frac{40}{20+30} = 0.8$
- $Recall = \frac{40}{40+20} = 0.67$

$$F_1 = 2\frac{precision \cdot recall}{recall + recall} = \frac{1.072}{1.47} = 0.72$$



Multi-Class Confusion Matrices

The confusion matrix generalizes to M classes as an M×M grid where element C_{ij} shows samples of class i classified as class j.

Strong diagonal values indicate correct classifications. Off-diagonal values reveal specific confusion patterns between classes.

For each class, compute separate precision, recall, and F_1 scores by treating it as a binary problem (class vs. all others).

Interpreting ROC Curves

ROC (Receiver Operating Characteristic) curves provide a powerful visual tool to evaluate the performance of **binary classifiers,** especially when they output a probability or confidence score. They allow us to assess a classifier's performance independently of the chosen classification threshold.

A classifier uses a confidence function, $c(\mathbf{x})$, which assigns a score to each input $\mathbf{x}$.

The final classification is determined by comparing this confidence score against a threshold θ:

$$h_\theta(\mathbf{x}) = [c(\mathbf{x}) \geq \theta]$$

As the threshold θ varies, so do the counts of True Positives (TP_θ), True Negatives (TN_θ), False Positives (FP_θ), and False Negatives (FN_θ).

Key Metrics: True Positive Rate (TPR) & False Positive Rate (FPR)

These rates quantify the classifier's performance at different thresholds:

The True Positive Rate (TPR), also known as **Recall** or Sensitivity, measures the proportion of actual positive cases that are correctly identified:

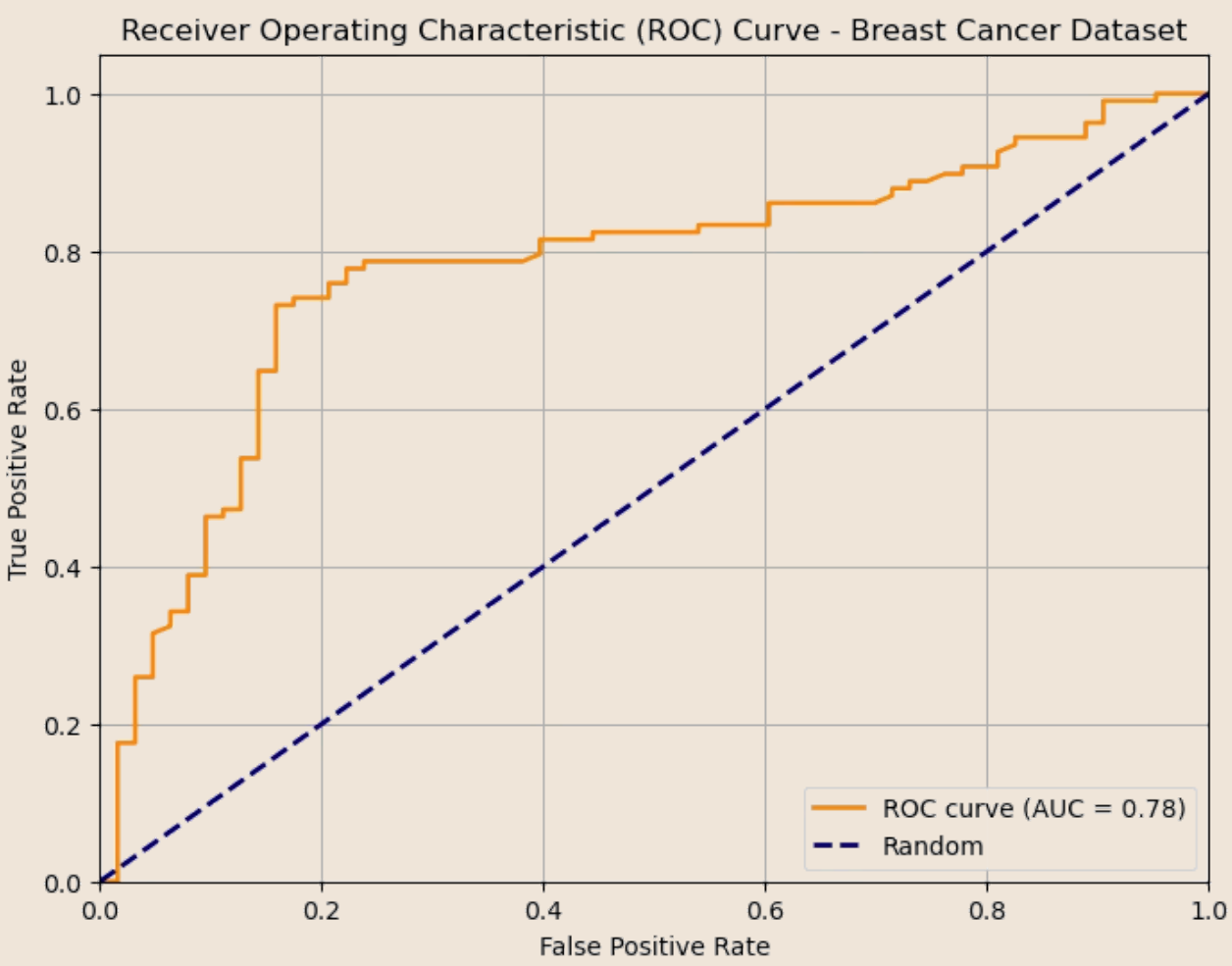
$$TPR_\theta = \frac{TP_\theta}{TP_\theta + FN_\theta}$$

The False Positive Rate (FPR) measures the proportion of actual negative cases that are incorrectly identified as positive:

$$FPR_\theta = \frac{FP_\theta}{FP_\theta + TN_\theta}$$

- **Low Thresholds:** If θ is very low, nearly all instances are classified as positive. Both TPR and FPR approach 1.
- **High Thresholds:** If θ is very high, nearly all instances are classified as negative. Both TPR and FPR approach 0.

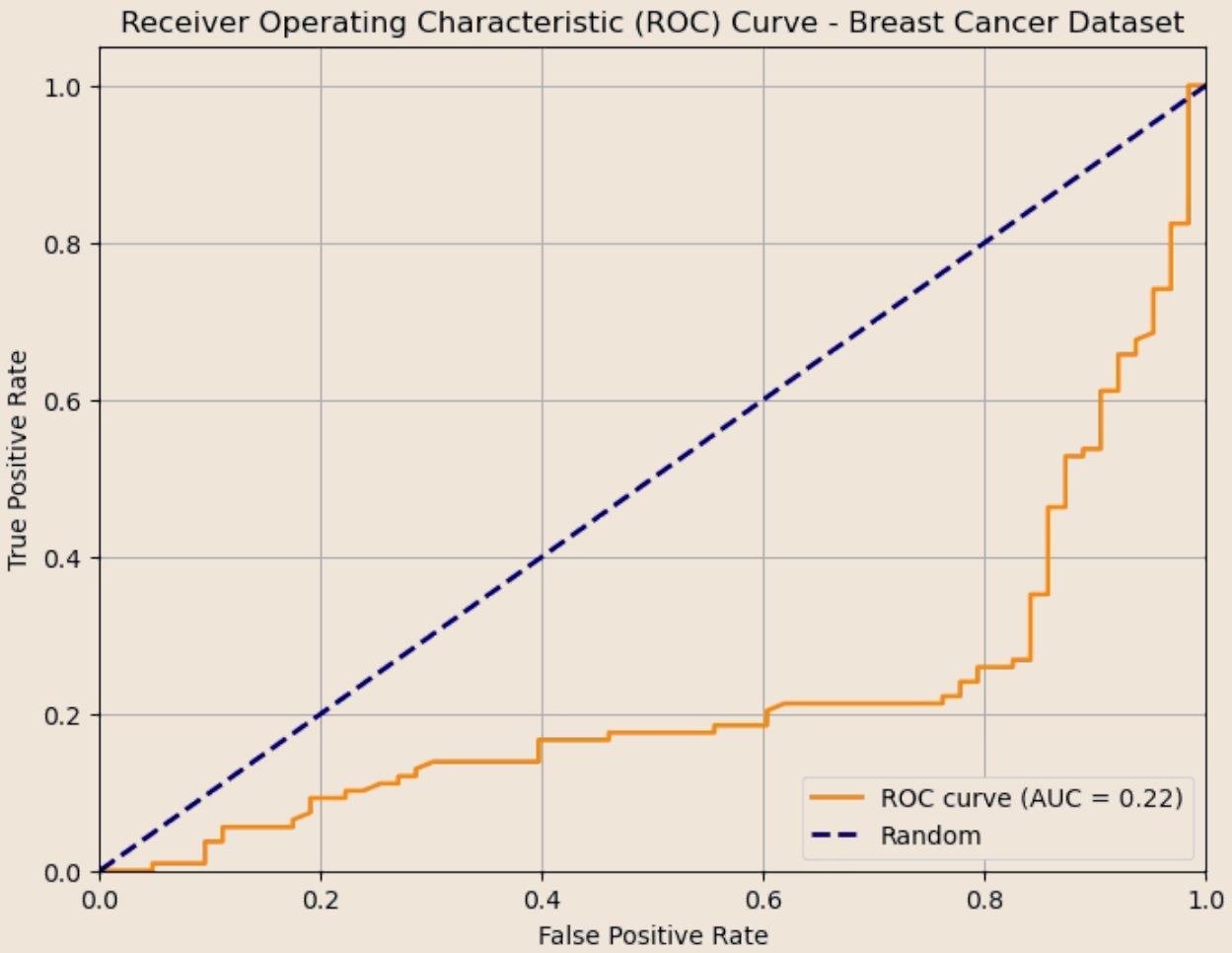
The ROC curve is created by plotting the TPR against the FPR for every possible threshold θ.



A good classifier's ROC curve will rise quickly towards the top-left corner (high TPR, low FPR), indicating better separation between classes across various thresholds.

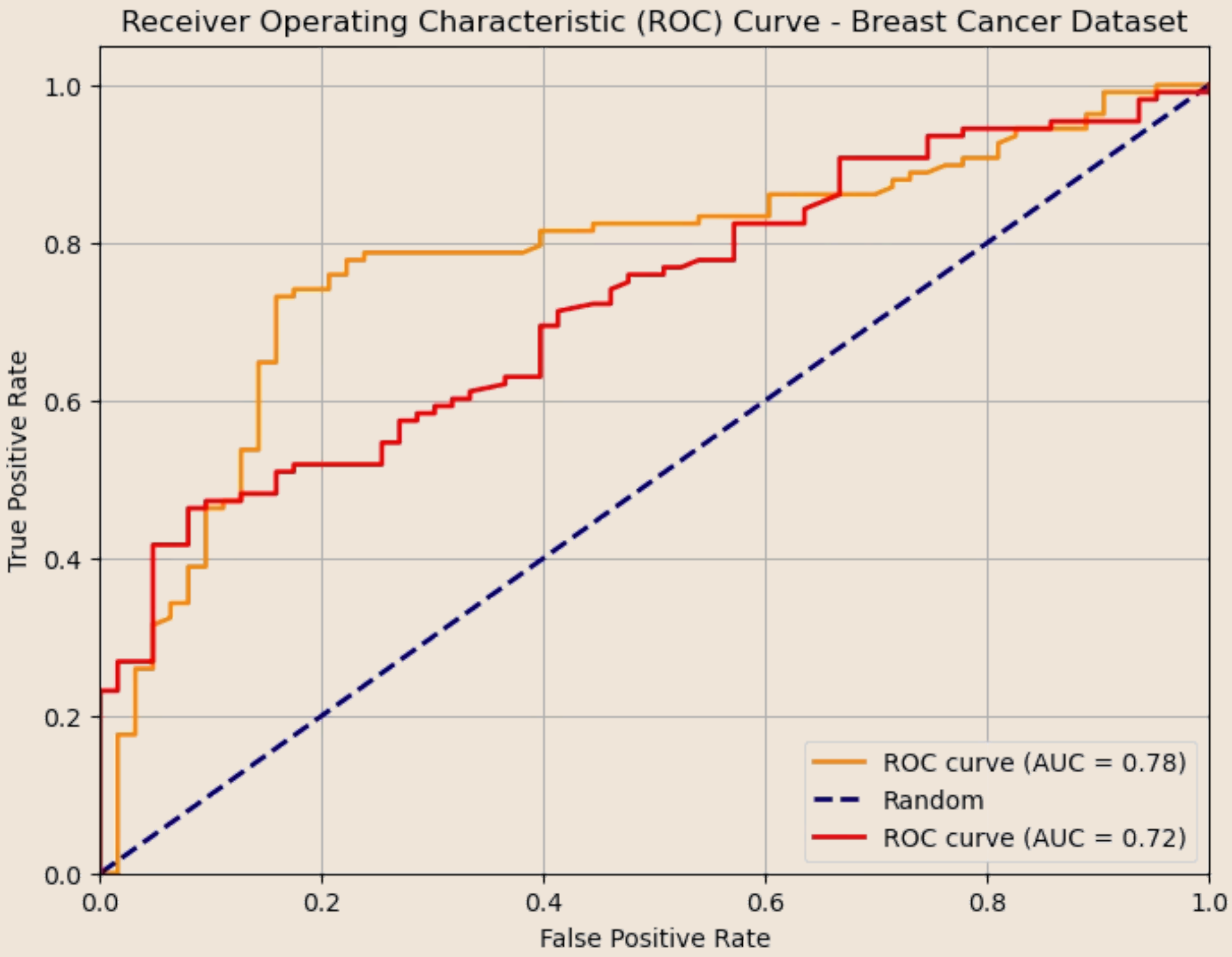
The **Area Under the Curve (AUC)** summarizes the ROC curve into a single value, representing the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. A higher AUC indicates better overall performance.

ROC Curves - Notable Case



Any curve which is systematically below this line indicates a classifier which is doing thresholding in the wrong way (i.e. we should invert the sign of the thresholding). For instance, this is the curve of the same classifier when we invert the sign of thresholding.

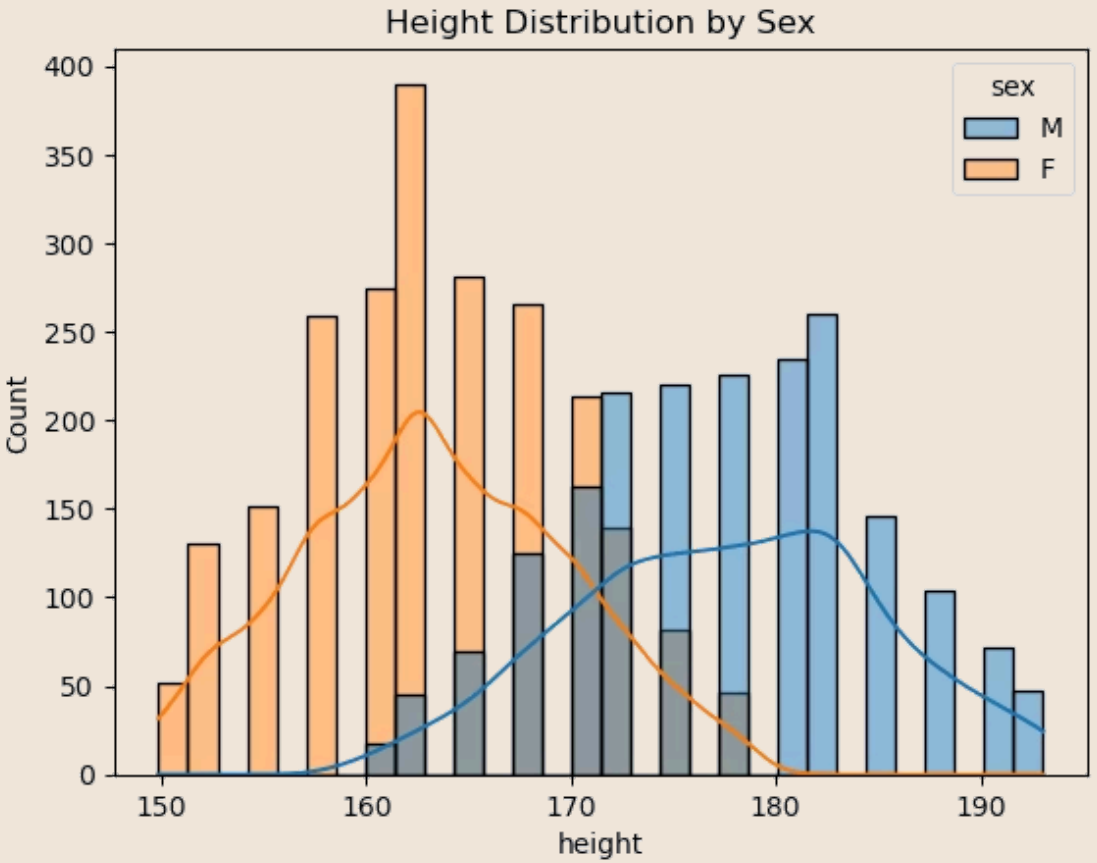
ROC Curve to Compare Models



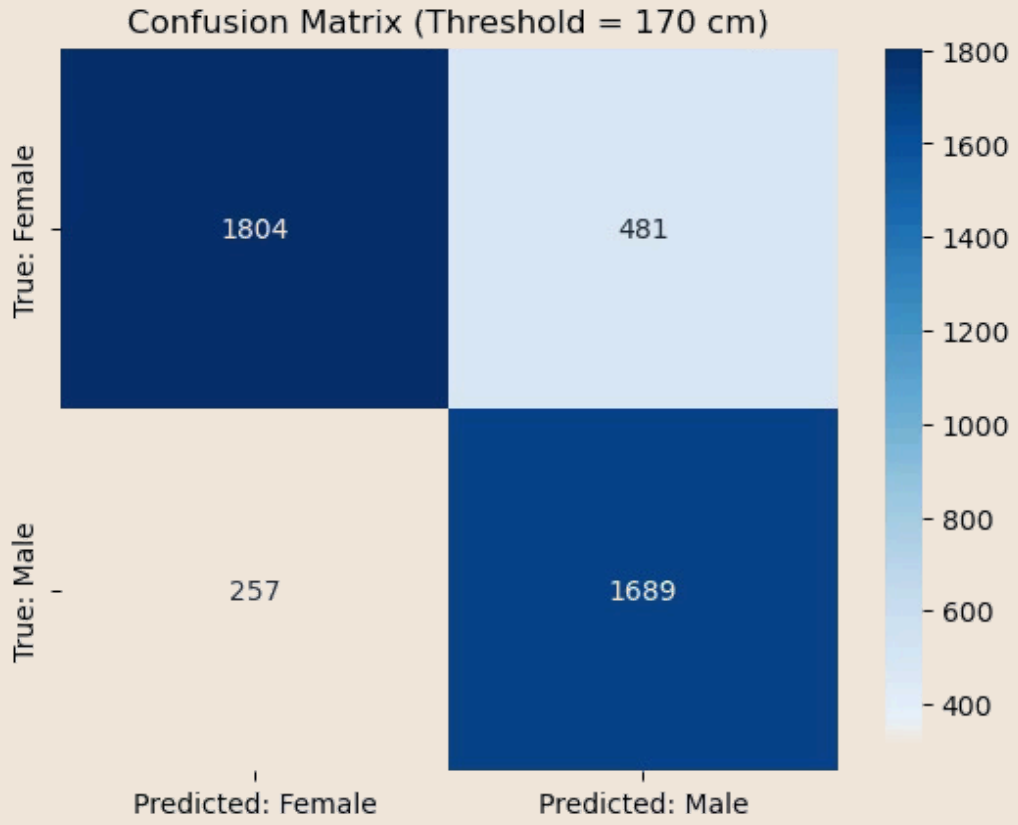
Threshold Classification Example

A simple height-based classifier demonstrates core evaluation concepts. Using a dataset of heights and weights, we predict biological sex with a single threshold.

> **If a person's** `height` **is above a certain *threshold*, we predict 'Male'.**

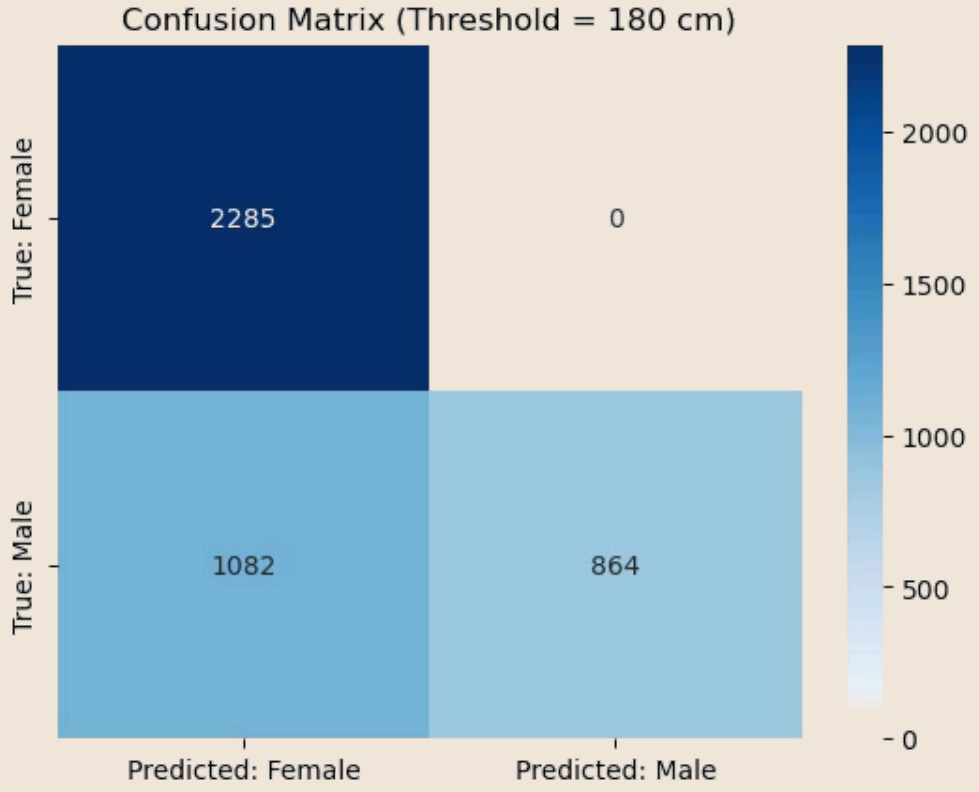


Threshold = 170cm



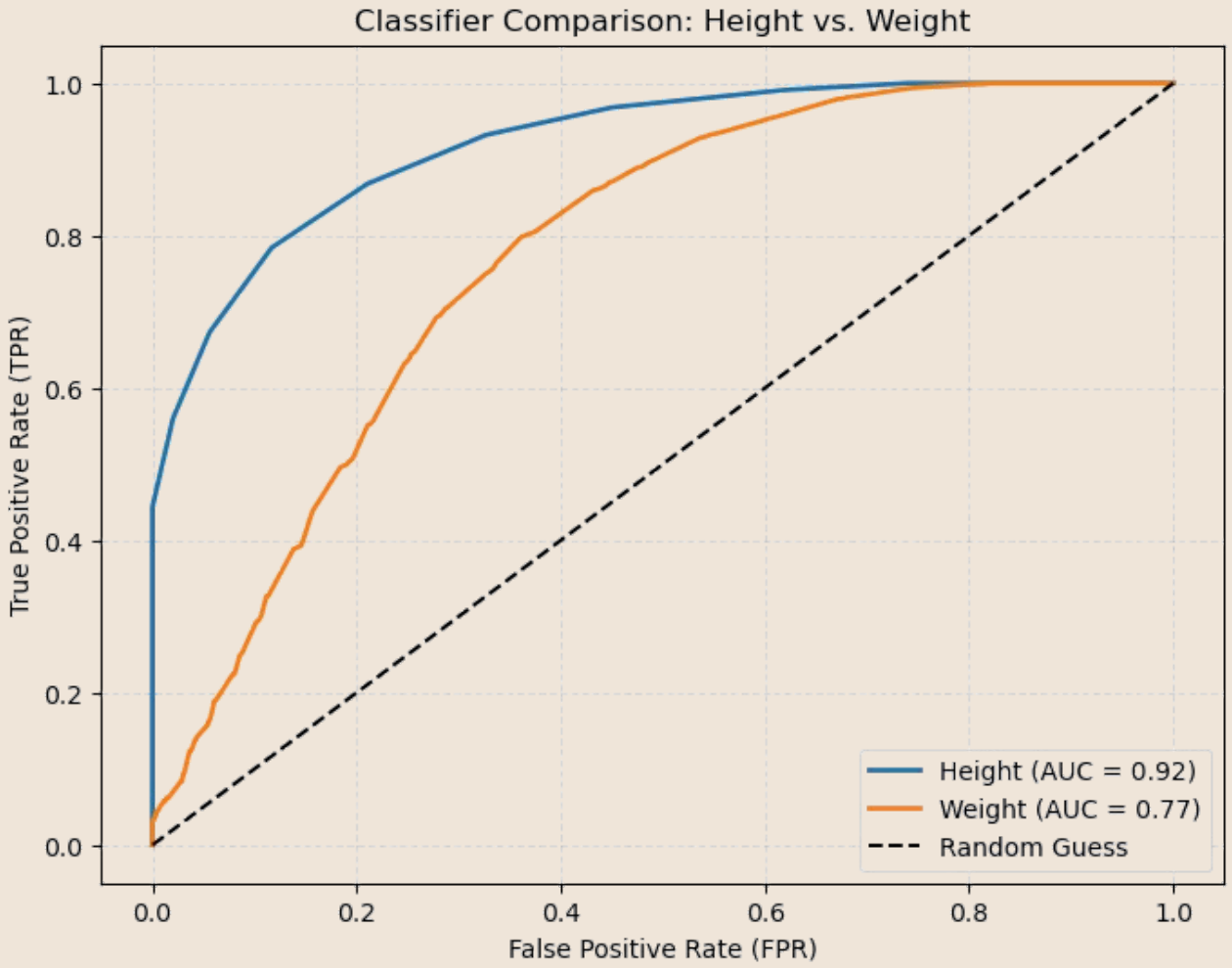
	precision	recall	f1-score	support
False	0.88	0.79	0.83	2285
True	0.78	0.87	0.82	1946
accuracy			0.83	4231
macro avg	0.83	0.83	0.83	4231
weighted avg	0.83	0.83	0.83	4231

Threshold = 180cm



	precision	recall	f1-score	support
False	0.68	1.00	0.81	2285
True	1.00	0.44	0.61	1946
accuracy			0.74	4231
macro avg	0.84	0.72	0.71	4231
weighted avg	0.83	0.74	0.72	4231

ROC Curves



Comparing Height and Weight for a threshold-based classifier.

Height is much better than weight, independent of the threshold.

Finding the Optimal Threshold

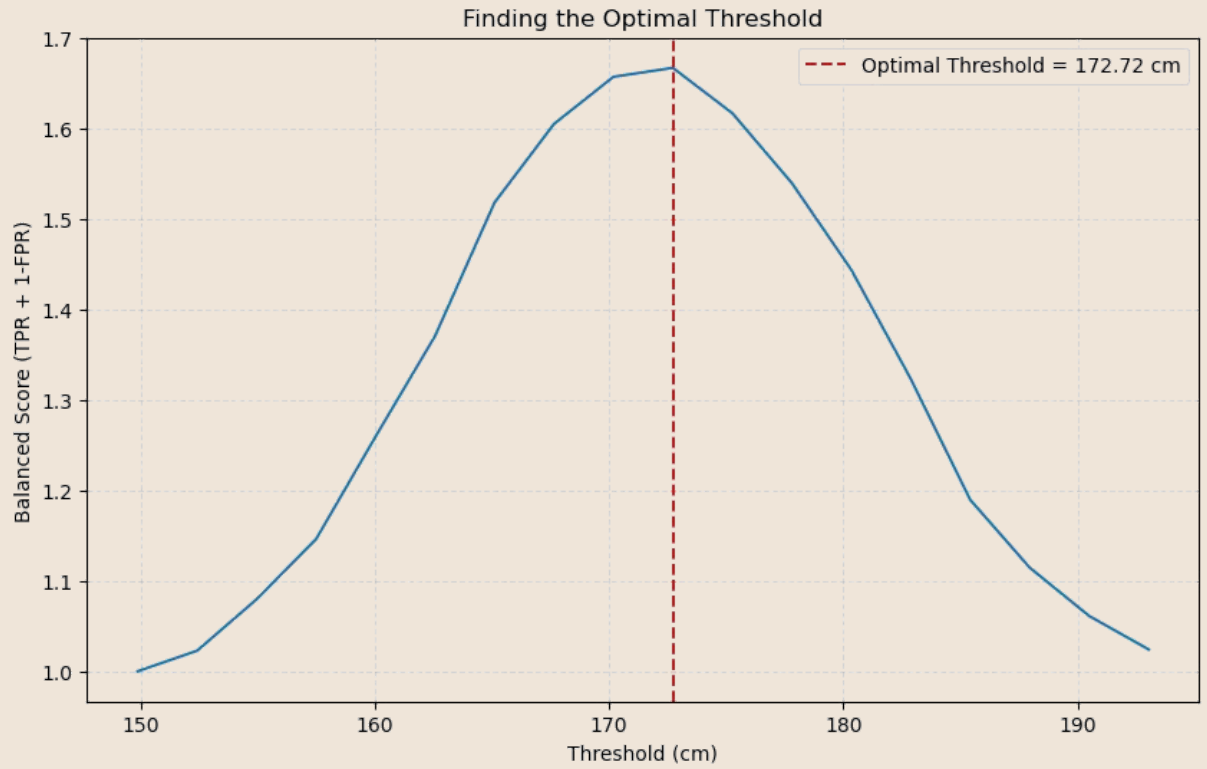
The ROC curve shows all possible trade-offs, but we must choose one threshold for deployment. The optimal choice depends on application requirements.

Youden's J Statistic

For balanced performance, maximize:

$$J = TPR + (1 - FPR)$$

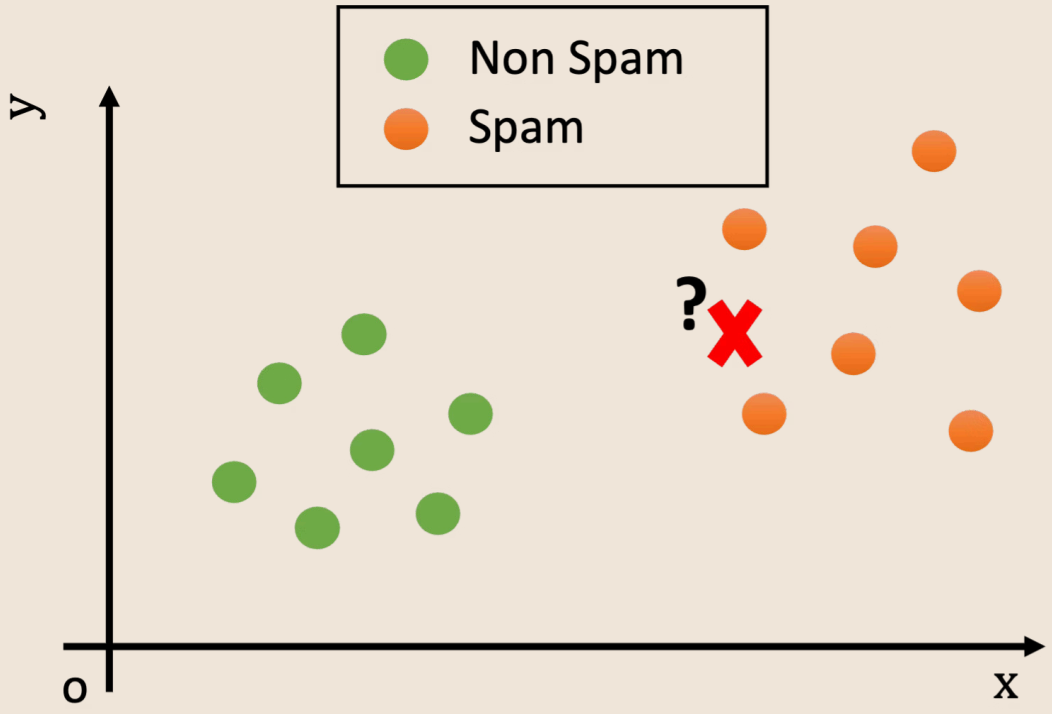
This finds the threshold closest to the ideal top-left corner of the ROC curve.



For the height classifier, the optimal threshold is 172.72 cm, achieving 78% TPR with only 12% FPR—a strong, balanced result.

```
--- Final Model at Optimal Threshold (172.72 cm) ---
True Positive Rate (TPR):  0.78 (We find 78% of all Males)
False Positive Rate (FPR): 0.12 (We mislabel only 12% of Females)
------------------------

Verifying with confusion_matrix:
TPR Check: 0.78
FPR Check: 0.12

Classification Report at Optimal Threshold:
              precision    recall  f1-score   support

       False       0.83      0.88      0.85      2285
        True       0.85      0.78      0.82      1946

    accuracy                           0.84      4231
   macro avg       0.84      0.83      0.84      4231
weighted avg       0.84      0.84      0.84      4231
```

Nearest Neighbor Algorithm (1-NN)



What's the class of the new (cross) sample?

We can measure how "similar" two examples $\mathbf{x}$ and $\mathbf{x}'$ are using a suitable distance function d:

$$d(\mathbf{x}, \mathbf{x}^{'})$$

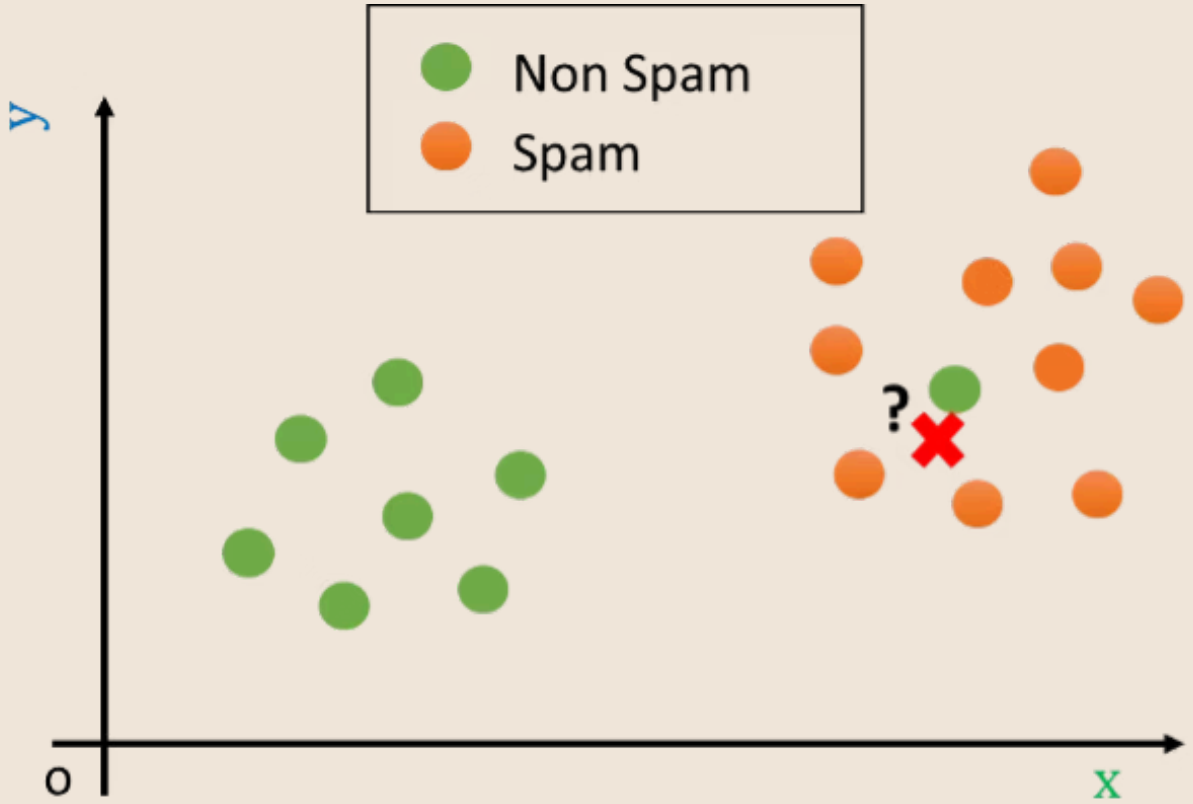
We expect similar examples to have a small distance. *A very common choice for $d$is the Euclidean distance*:

$$d\left(\mathbf{x},\mathbf{x}^{'}\right) = \left|\mathbf{x}-\mathbf{x}^{'}\right|_2 = \sqrt{\sum_{i=1}^{N}\left(x_i - x_i^{'}\right)^2}$$

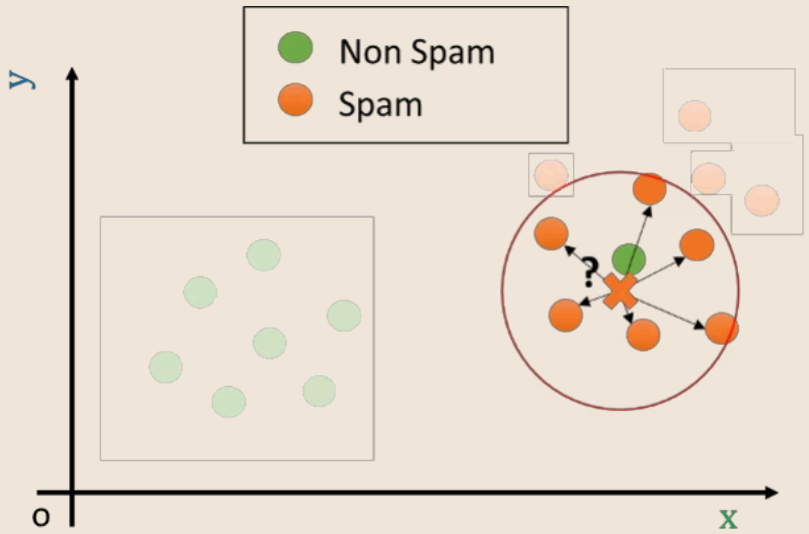
We can hence define a classifier which leverages our intuition **assigning to a test example $\mathbf{x}^{'}$ the class of the closest example in the training set**:

$$h\left(\mathbf{x}^{'}\right) = \arg_y \min d\left(\mathbf{x}^{'},\ \mathbf{x}\right) \mid (\mathbf{x}, y) \in TR$$

Outliers and 1-NN



- What if the dataset contains outliers?
- Using 1-NN here can bring suboptimal results.
- However, we know that the second nearest neighbor would already be a better choice!



K-Nearest Neighbor: Core Intuition

KNN classification relies on a simple principle: similar examples in feature space likely belong to the same class.

Measure Distance

Calculate Euclidean distance between test point and all training examples

Find Neighbors

Identify the K closest training examples

Majority Vote

Assign the most common class among the K neighbors

The algorithm assumes nearby points share characteristics. For emails, similar word frequencies suggest similar content and classification.

K-Nearest Neighbor: Addressing 1-NN Limitations

Instead of considering only the single closest point, the K-Nearest Neighbor (K-NN) algorithm evaluates a "neighborhood" of several points. By considering multiple neighbors, the algorithm becomes more resilient to individual noisy or mislabeled data points.

Defining the K-Neighborhood

Rather than a fixed radius, which can lead to neighborhoods with varying numbers of points (or even zero in sparse areas), K-NN defines a neighborhood by the **K closest data points**. This ensures a consistent number of votes for each classification decision.

$$N_K(\mathbf{x}') = N(\mathbf{x}', R_K(\mathbf{x}'))$$

$$R_K(\mathbf{x}') = \sup\{r : |N(\mathbf{x}', r) \setminus \{\mathbf{x}'\}| \leq K\}$$

For a given test point $\mathbf{x}'$, the K-neighborhood $N_K(\mathbf{x}')$ consists of the K training points closest to $\mathbf{x}'$.

K-NN Classification Algorithm

The K-NN classifier assigns the class that is most frequent among these K neighbors:

$$h\left(\mathbf{x}'\right) = mode\{y|\left(\mathbf{x}, y\right) \in N\left(\mathbf{x}'; TR,\ K\right)\}$$

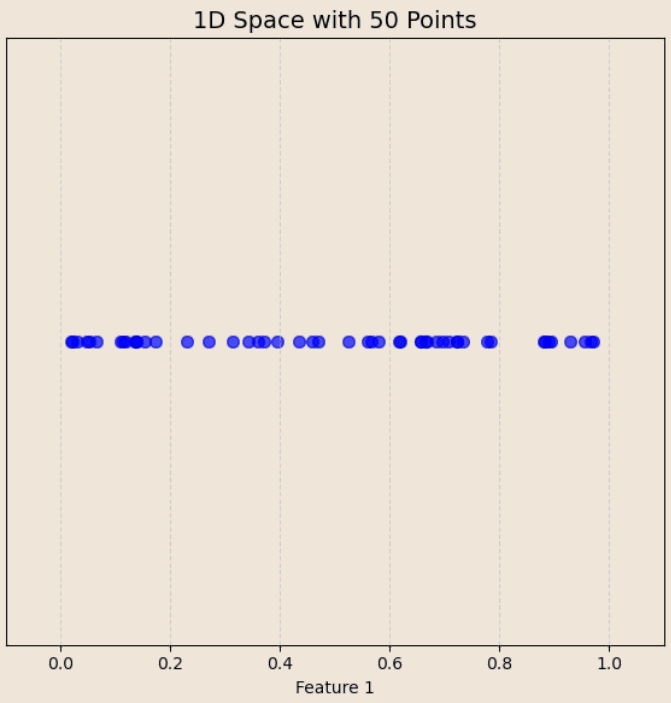
Here, `mode` returns the most frequent class label in the set of K neighbors.

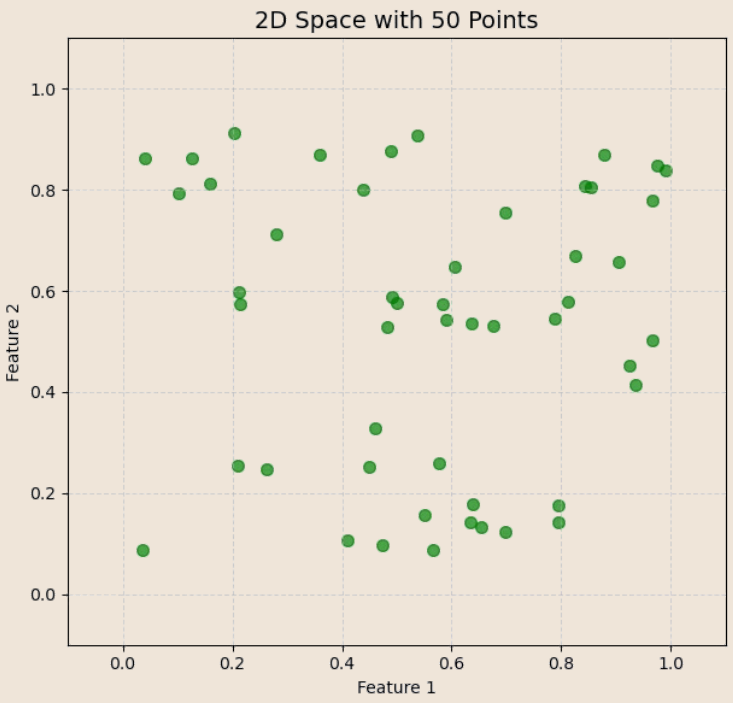
> **The Role of K:** The parameter K is a **hyperparameter** of the algorithm. Choosing an optimal K is crucial and is typically done through techniques like **cross-validation** or by evaluating performance on a **validation set**. A 1-NN algorithm is simply a special case where $K = 1$.

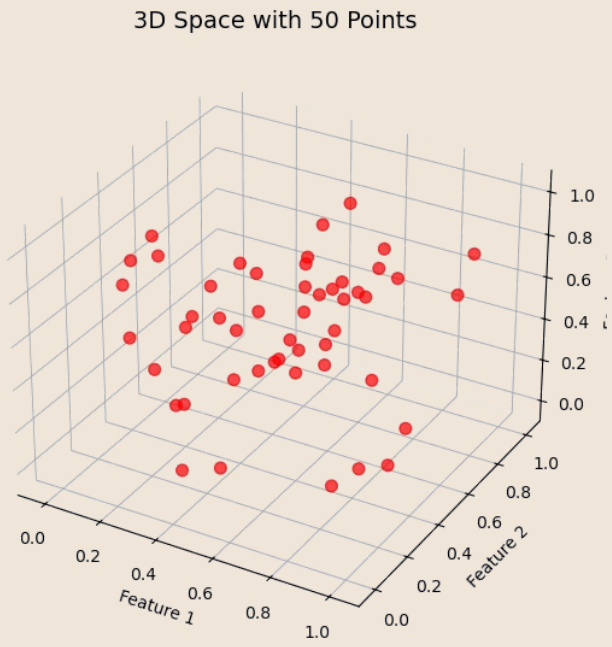
The Curse of Dimensionality

KNN's Fatal Flaw

As features increase, data becomes exponentially sparse and distance becomes meaningless. This is KNN's greatest weakness.







50

Data Points

Same number across dimensions

1000x

Volume Growth

Space expands exponentially

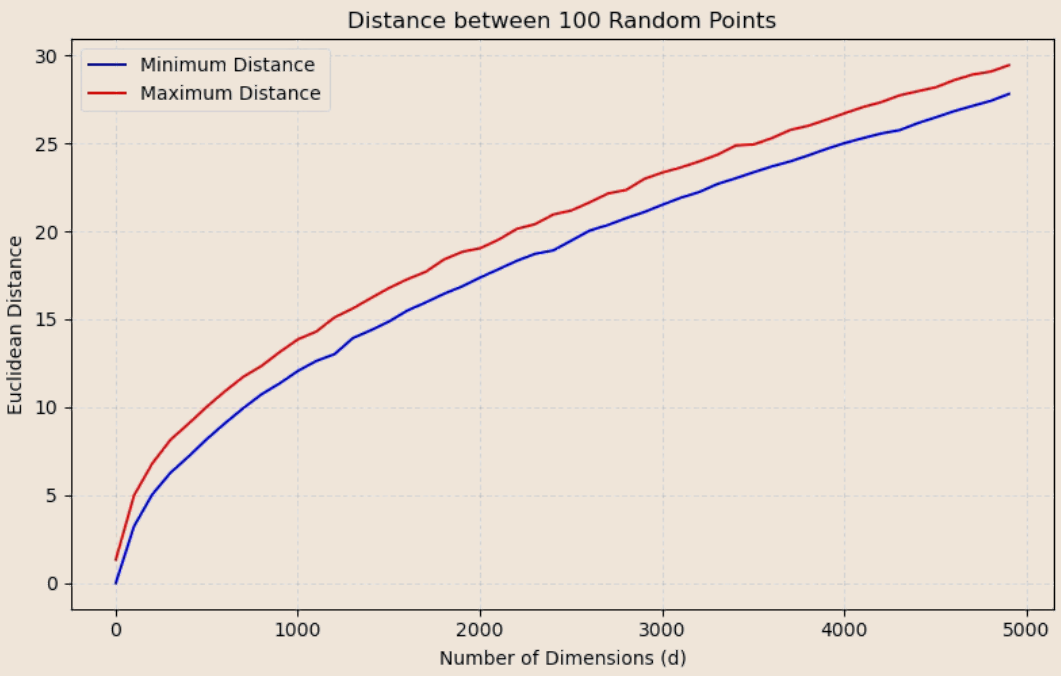
~0

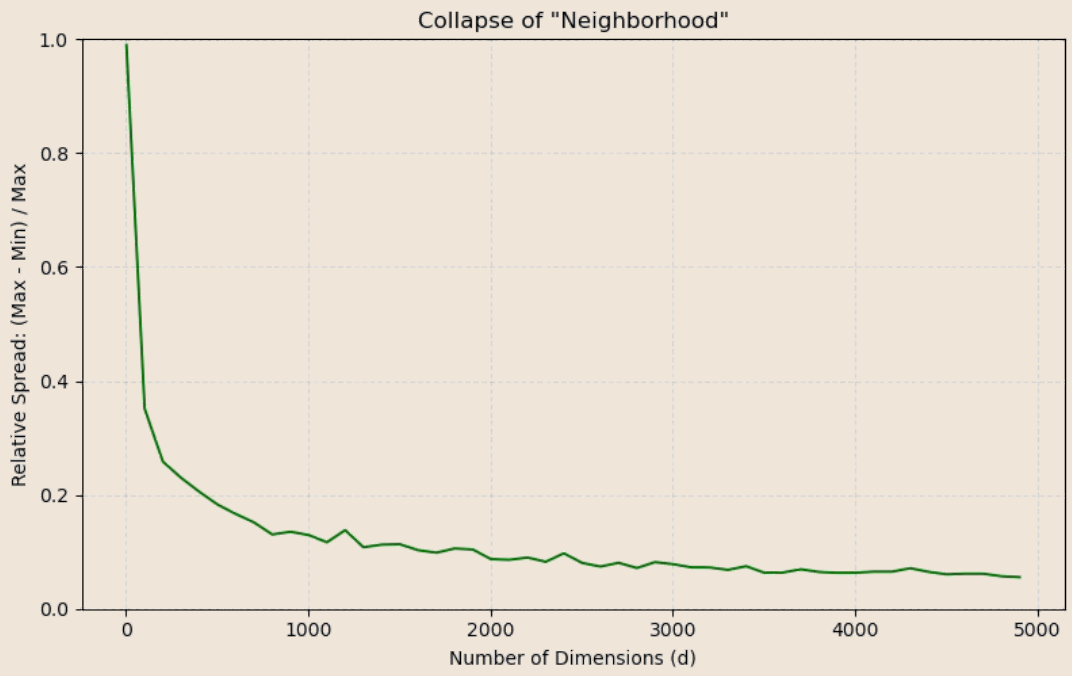
Distance Contrast

Min and max distances converge

In high dimensions, your "nearest" neighbor is almost as far as your "farthest" neighbor. The concept of "nearness" collapses, and KNN predictions become random.

Illustration of the Curse of Dimensionality

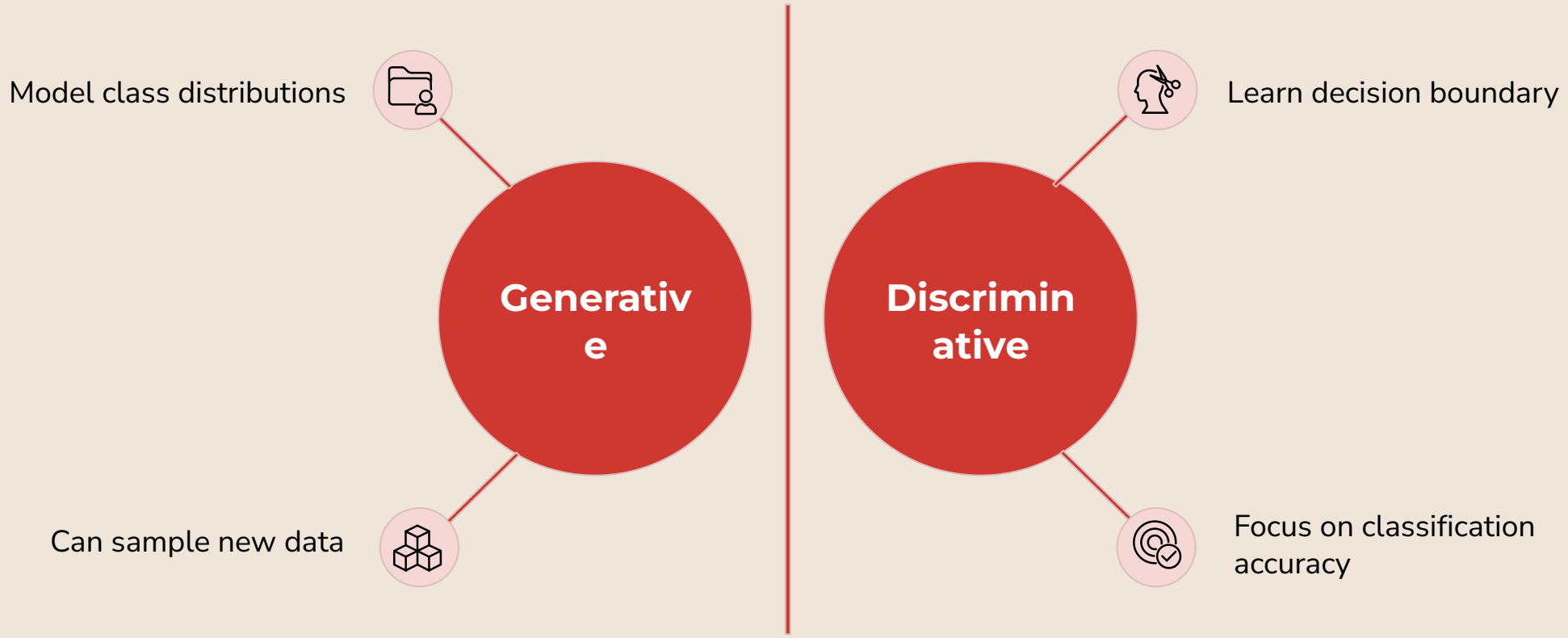




Practical Limit: KNN works excellently for 2-10 features but fails for high-dimensional data (images: 10,000+ features, text: 5,000+ features) without dimensionality reduction.

KNN: A Discriminative Classifier

Understanding the fundamental distinction between generative and discriminative classifiers is crucial in machine learning. This concept clarifies how models like K-Nearest Neighbors approach the problem of classification.



Generative Classifiers

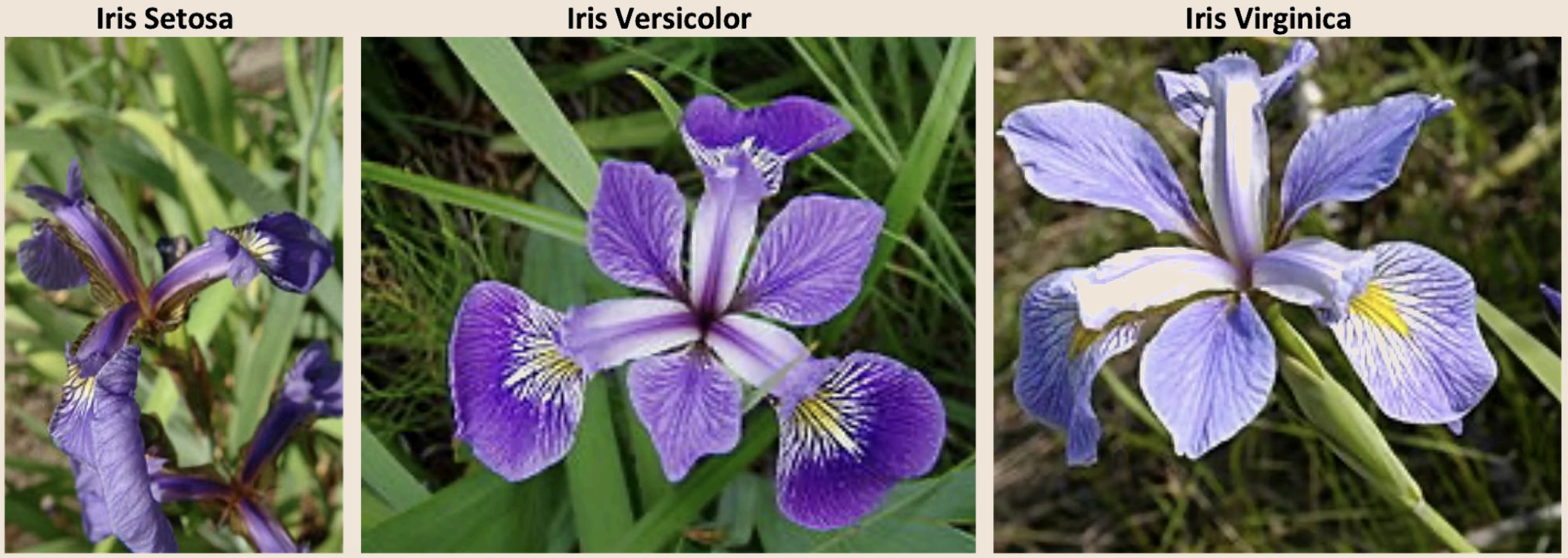
- **What they do:** Learn the underlying probability distribution of each class. They build a "story" for what each class looks like.
- **The Question it Answers:** "What does Class A look like?"
- **How it Classifies:** Use knowledge of class distributions (e.g., via Bayes' theorem) to calculate the probability of a class given new data, $P(y|X)$.
- **Example:** Naive Bayes is a classic generative model.

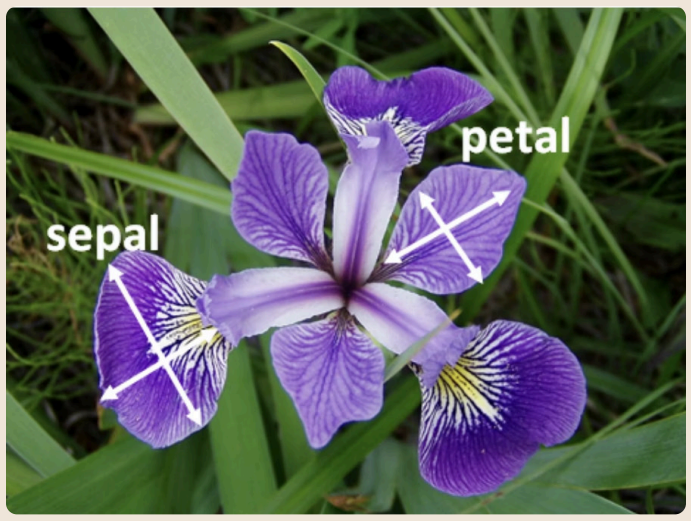
Discriminative Classifiers

- **What they do:** Ignore the underlying distribution of individual classes. They only focus on finding the decision boundary that separates one class from another.
- **The Question it Answers:** "What is the difference between Class A and Class B?"
- **How it Classifies:** Directly learn the function $h(x)$ or the boundary $P(y|X)$ to distinguish between classes.
- **Example: K-Nearest Neighbors** and Logistic Regression are classic discriminative models.

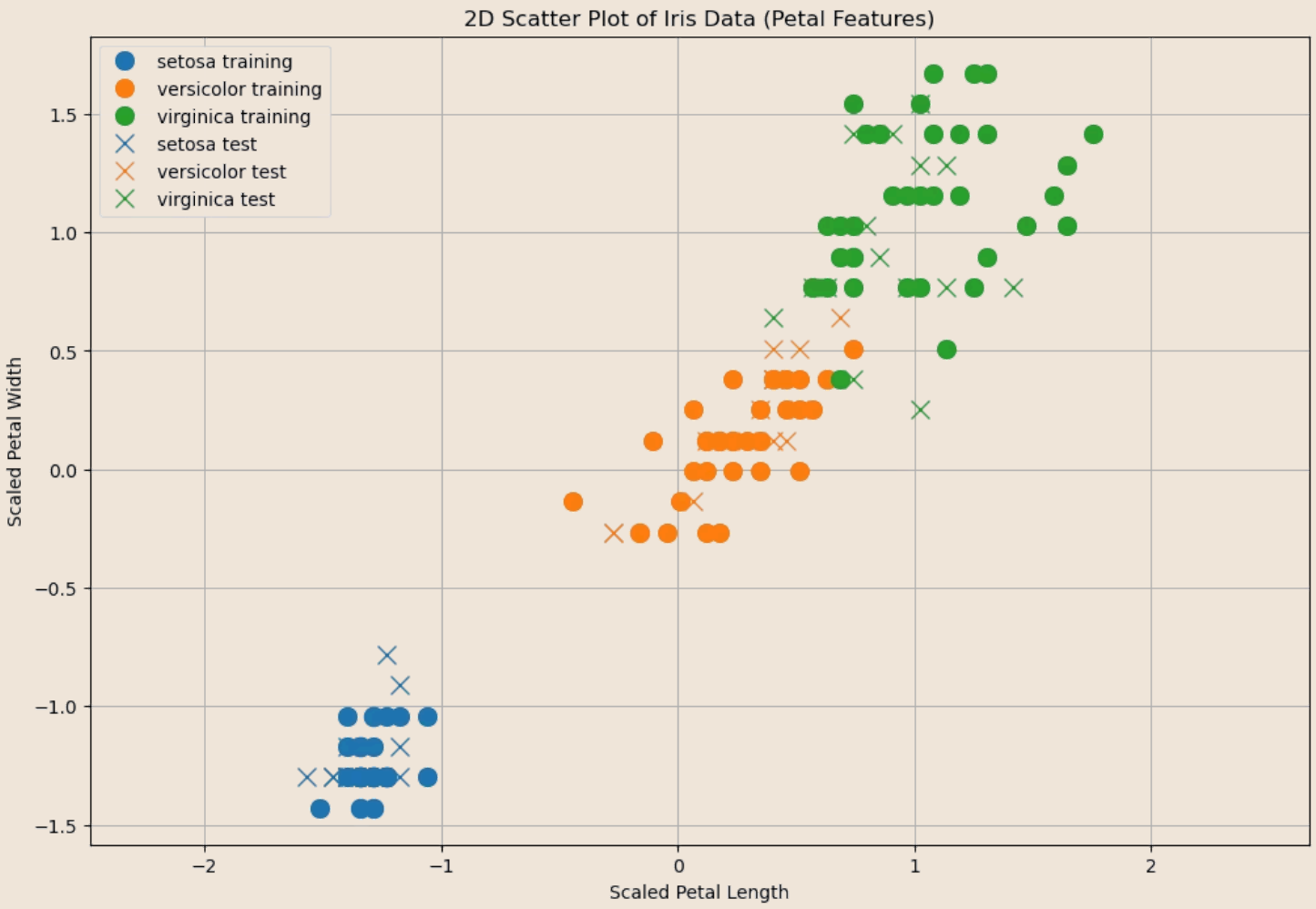
KNN on the Iris Dataset

The Fisher Iris dataset is the "hello world" of classification: 150 flowers from 3 species, measured by 4 features (sepal/petal length and width).



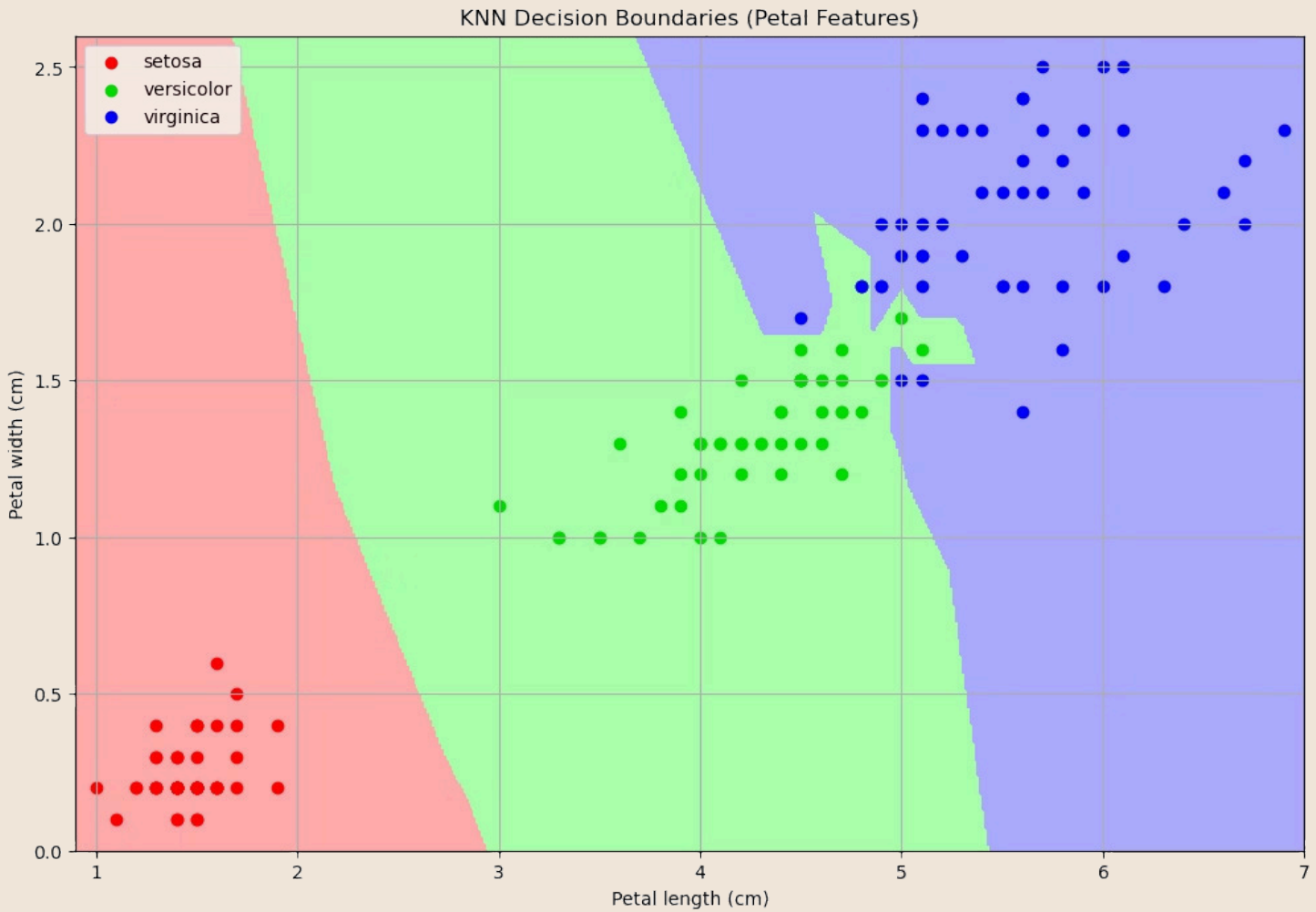


	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa



Using only two features,, we can visualize the 2D decision space and understand how KNN creates classification boundaries.

The Decision Boundary



A classifier f assigns a class to each input $\mathbf{x}$. Since $\mathbf{x}$ can be seen as a geometrical point in the n-dimensional space, it is generally interesting to see how the classification function works on a portion of the representation space.

This is done by generating a **classification map** or **decision boundary**, which is obtained computing the label that the classifier would assign to a dense grid of data points.

In the plot, points of different colors represent the data points belonging to the three classes of the dataset, while regions of different colors represent how that point in space would be classified by the model.

The Impact of K on Decision Boundaries

The hyperparameter K controls the bias-variance trade-off, dramatically affecting how the algorithm partitions feature space.

Small K

High variance, low bias. Over-segments space, sensitive to outliers. Overfitting.

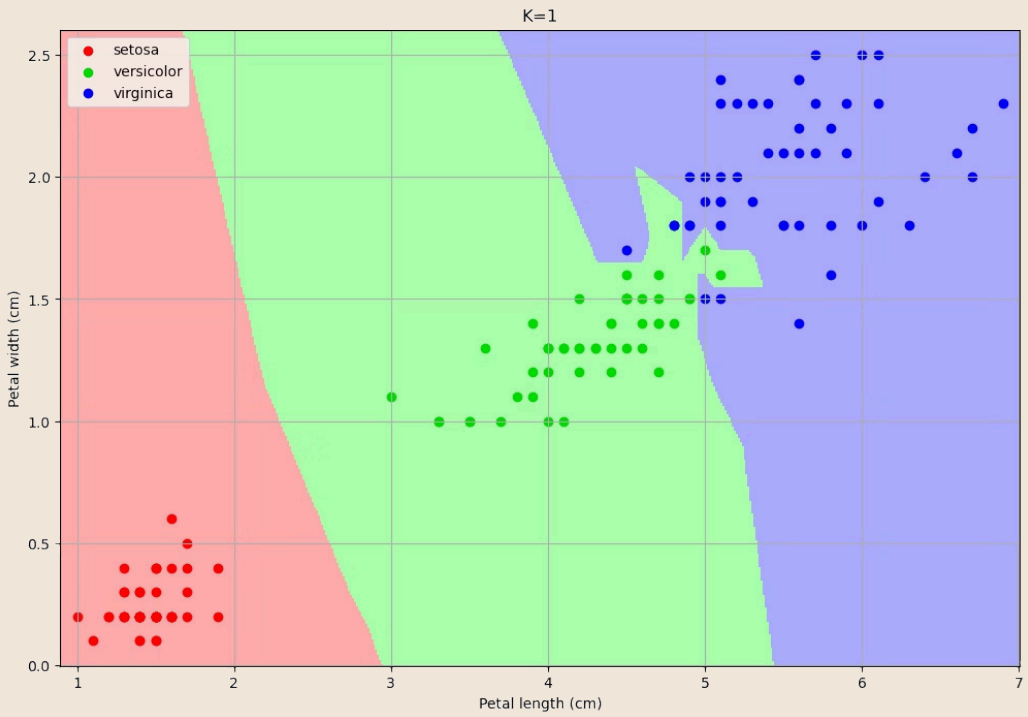
Large K

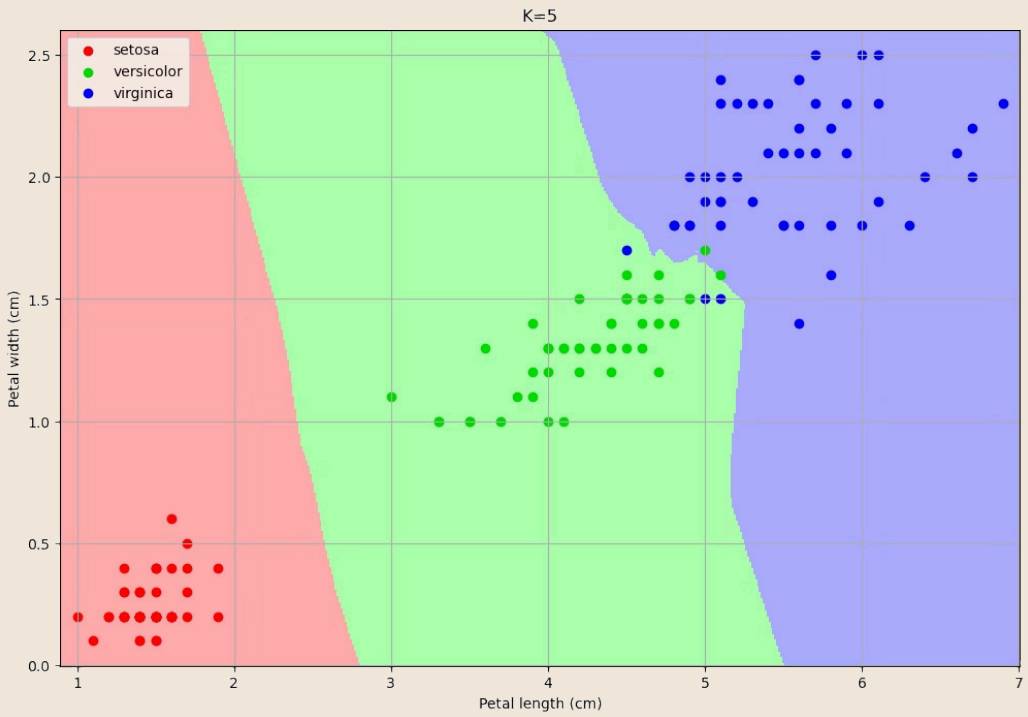
Low variance, high bias. Ignores local structure. Underfitting.

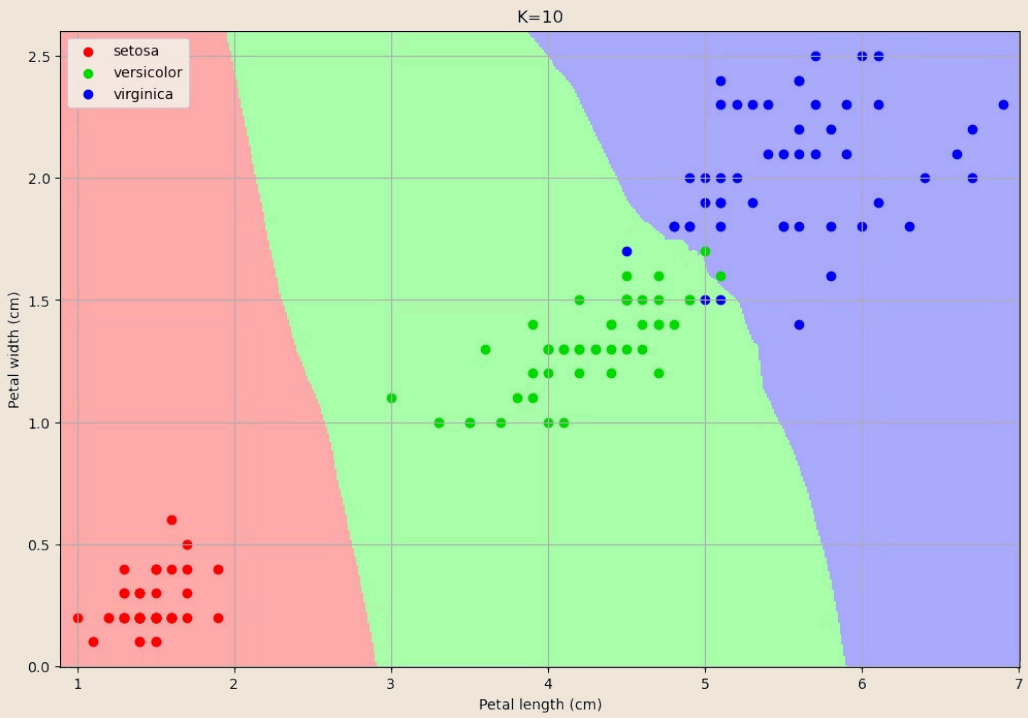
1 2 3

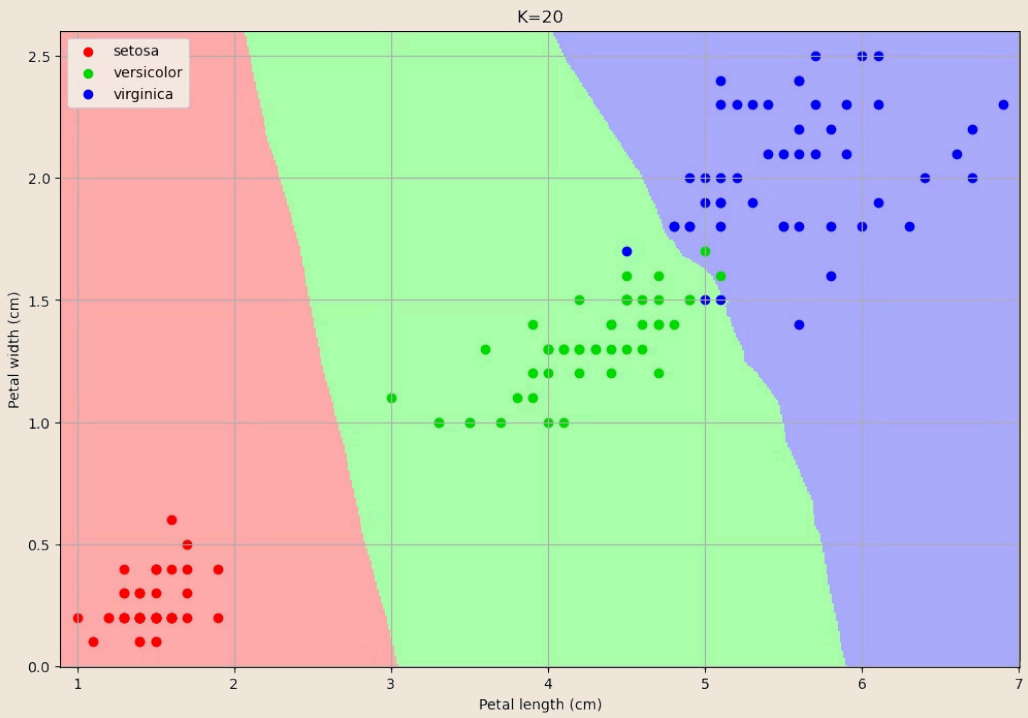
Medium K

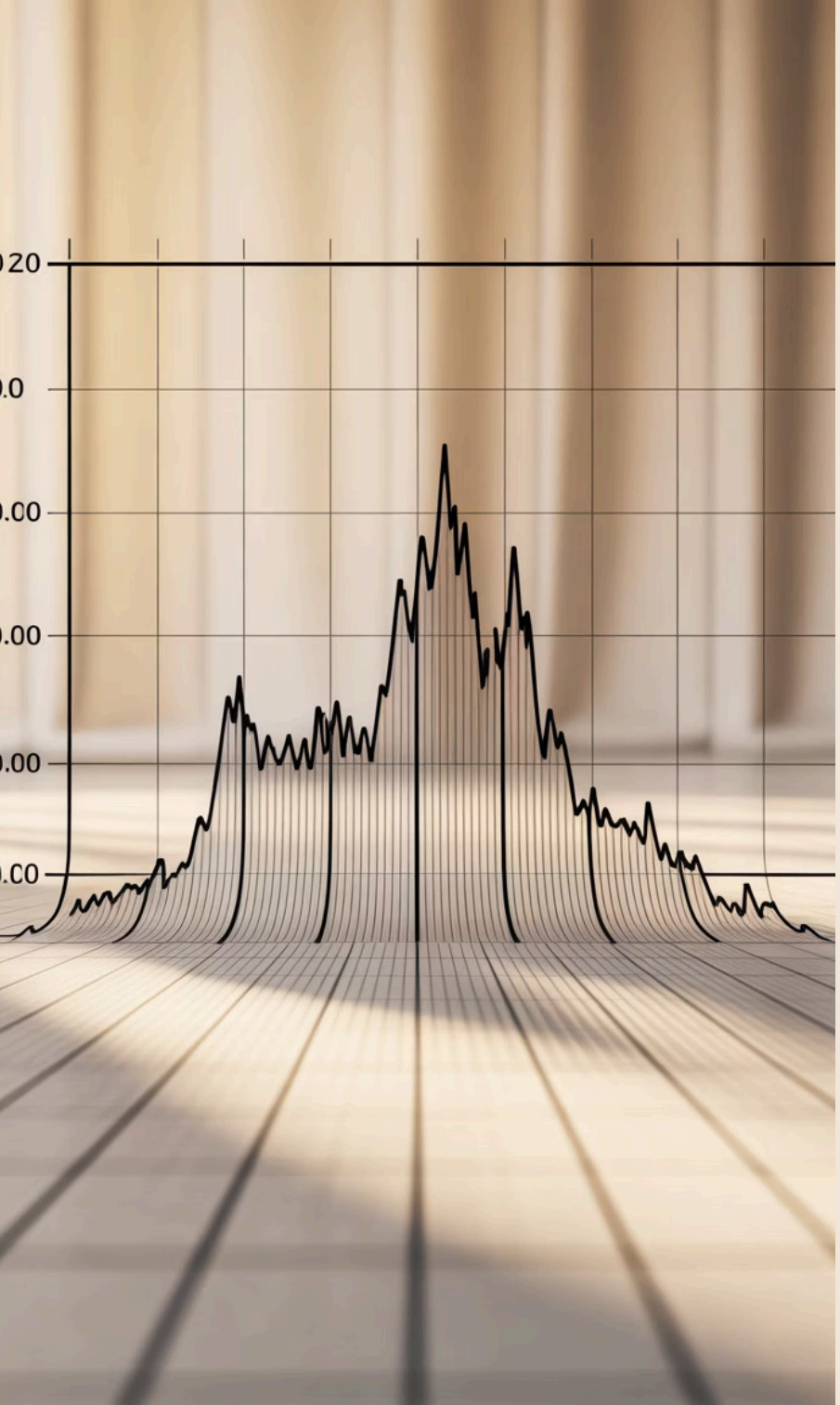
Balanced. Smooth regions while respecting local patterns. Optimal range.











Finding Optimal K with Cross-Validation

Cross-validation systematically tests different K values on held-out folds to find the optimal hyperparameter without touching the test set.

01

Split Data

Create train/test split. Lock away test set.

02

Cross-Validate

For each K, perform 5-fold CV on training data. Record mean accuracy.

03

Select Best K

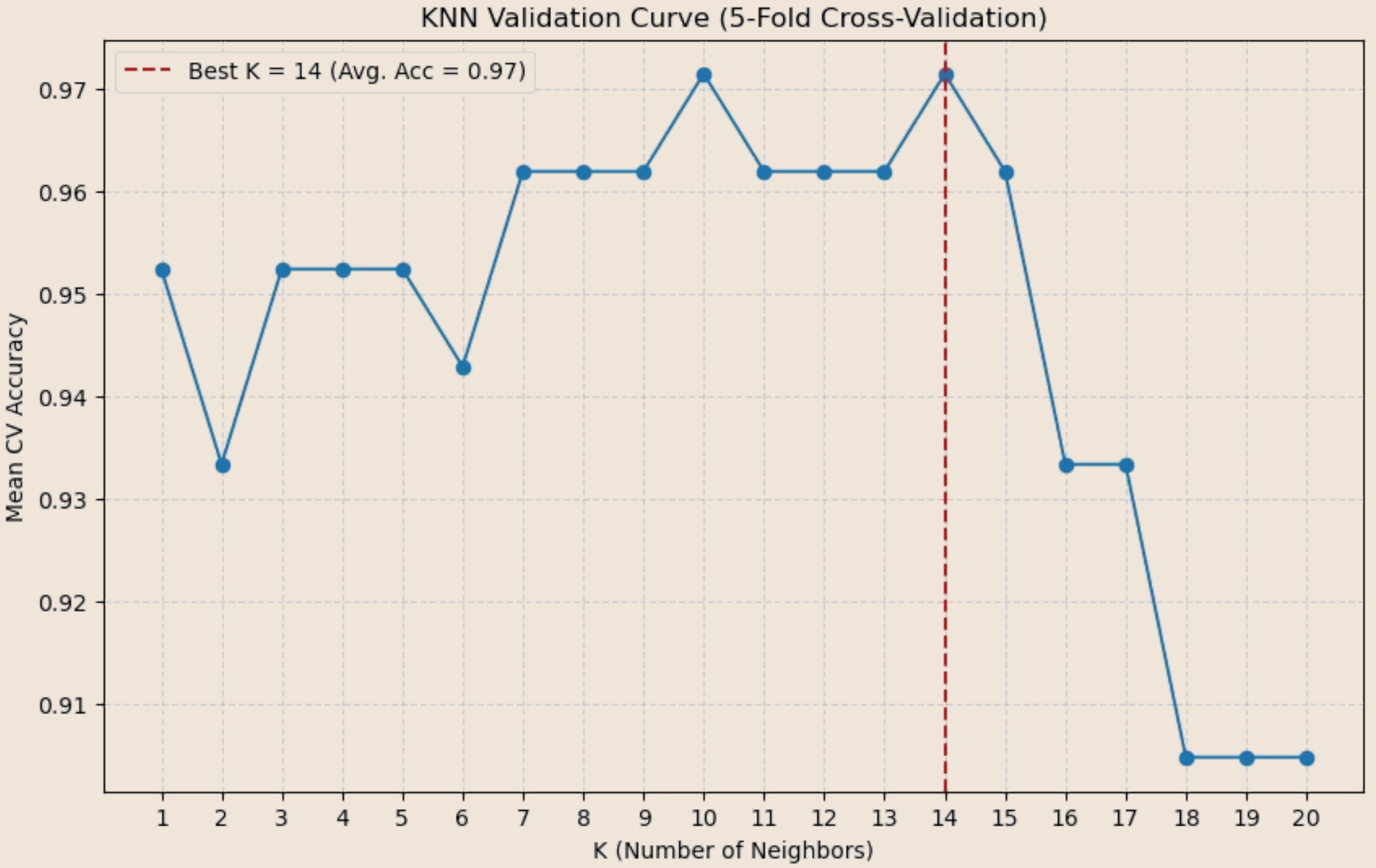
Choose K with highest CV accuracy (K=14 in our example).

04

Final Evaluation

Train on full training set with best K. Evaluate once on test set: 95.56% accuracy.

Finding the Best K by Cross-Validation



KNN for Regression

KNN extends naturally to regression by predicting the mean of K neighbors instead of the mode. The algorithm is identical except for the final aggregation step.

Classification vs Regression

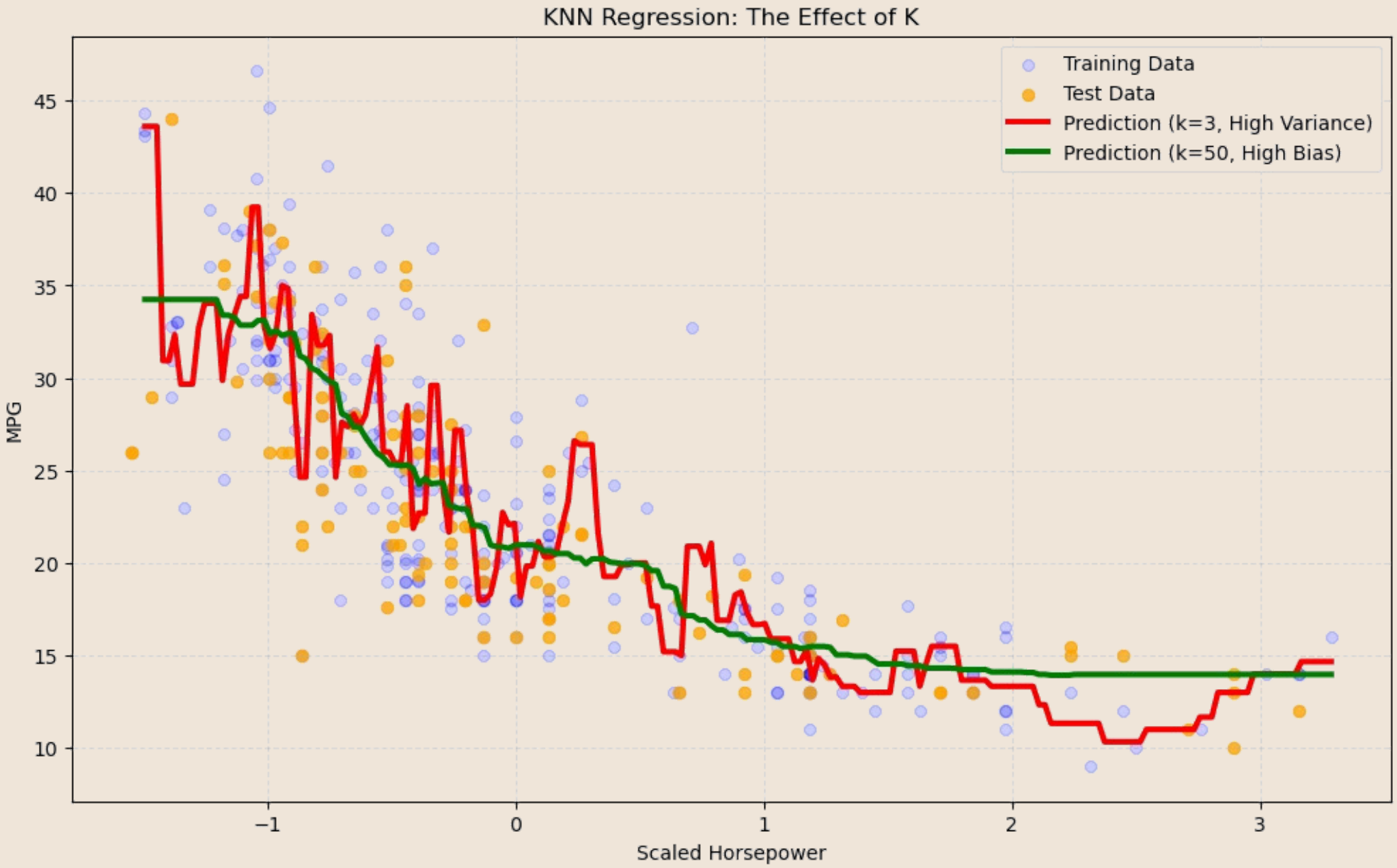
- **Classification:** Majority vote of K neighbors' labels
- **Regression:** Average of K neighbors' values

Same distance calculation, same neighbor selection, different aggregation.

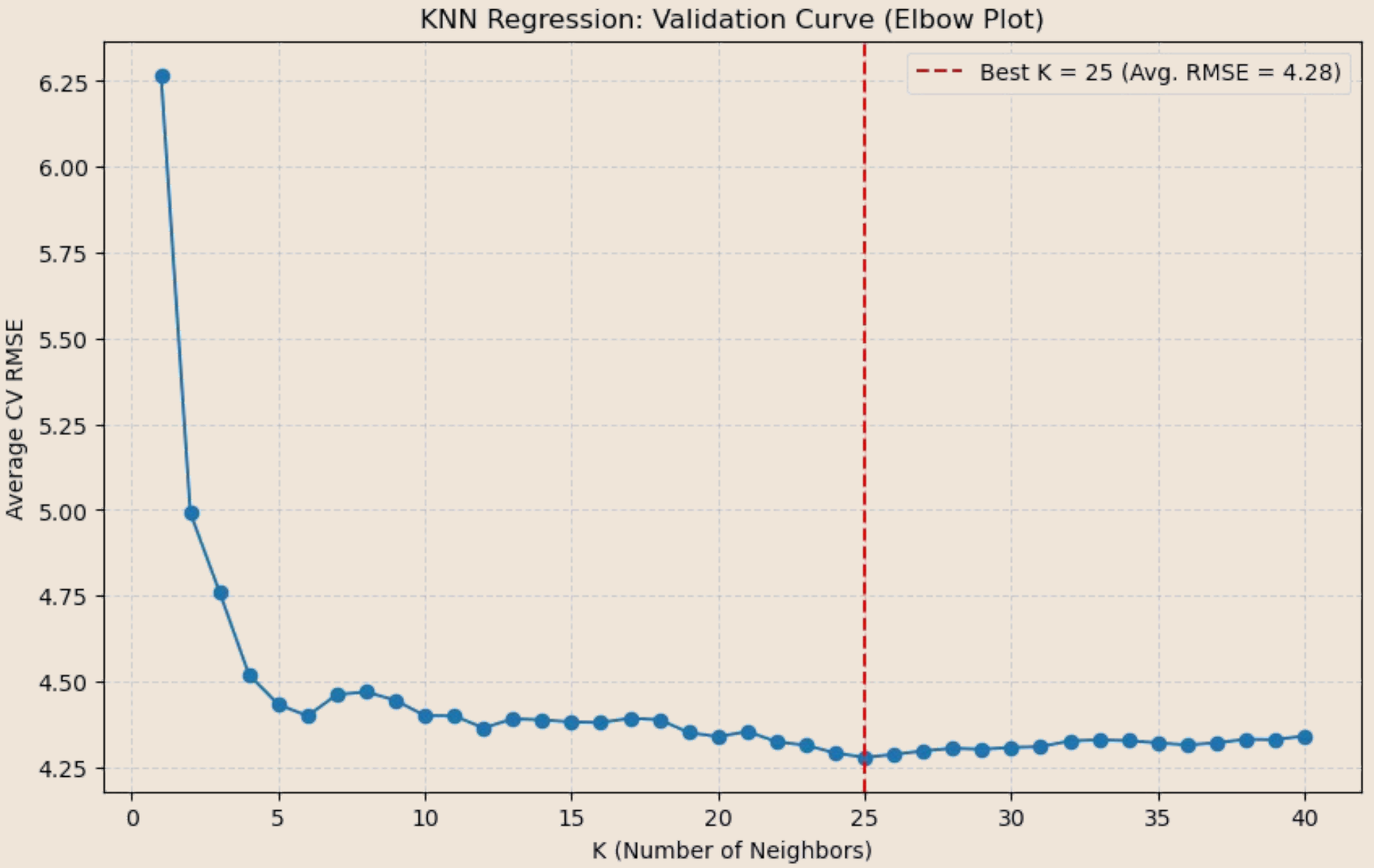
MPG Prediction Example

Predicting fuel efficiency from horsepower using KNN regression. Optimal K=25 found via cross-validation.

Final Test RMSE: 4.29 mpg



Finding the Best K for Regression



KNN in Python

Conclusions and Next Steps

We Have Explored:

- Classification Task
- Classification Measures
- Threshold-Based Classifier
- K-NN
- K-NN Regressor

In the next lectures, we will look at logistic regression

References

- Evaluation Measures for Classifcation: https://en.wikipedia.org/wiki/Precision_and_recall
- Nearest Neighbor: Section 2.5.2 of [1]

[1] Bishop, Christopher M. *Pattern recognition and machine learning*. springer, 2006. https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf