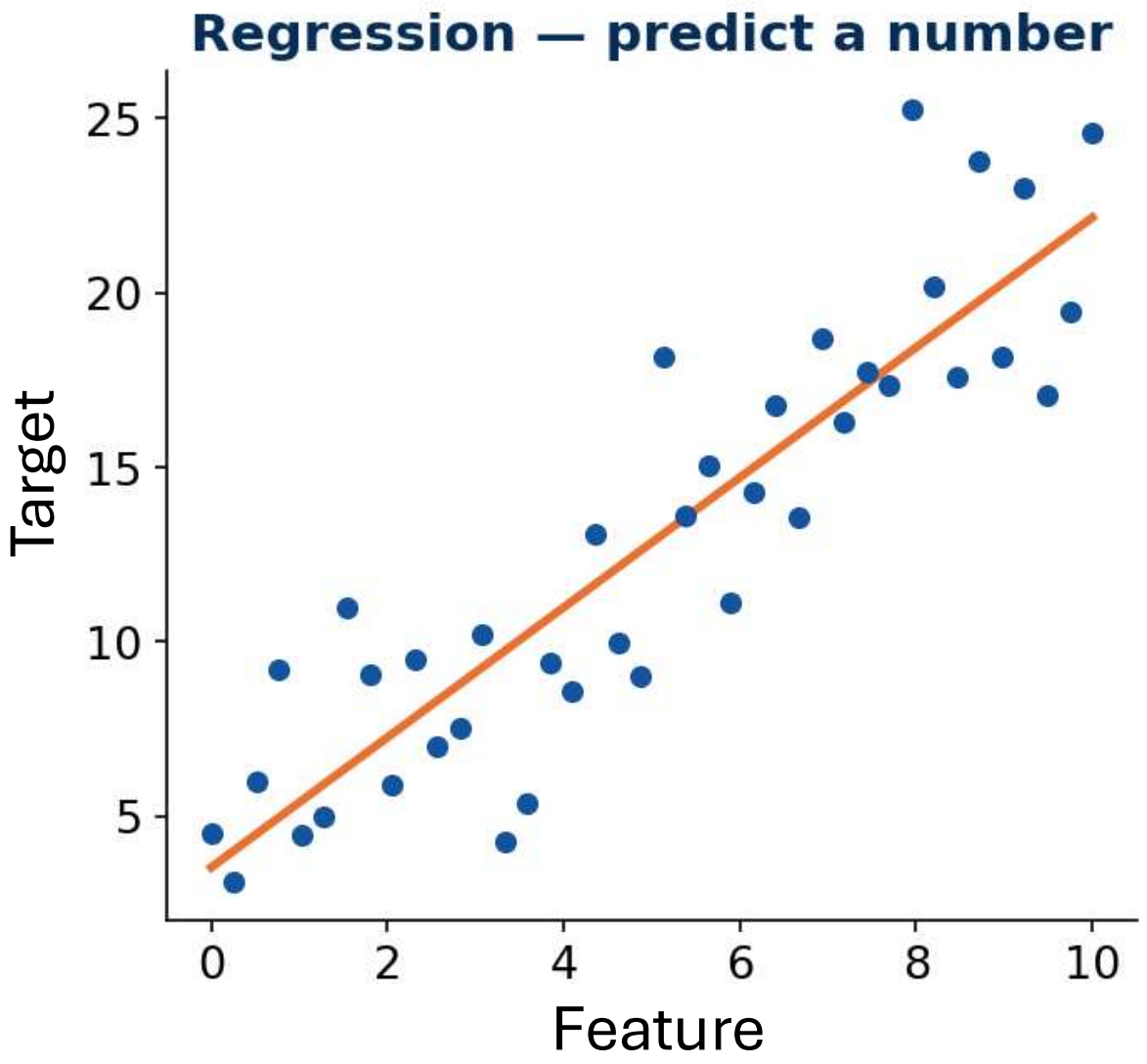
COM SCI M148, Lecture 10:
k-Nearest Neighbors (Classification)

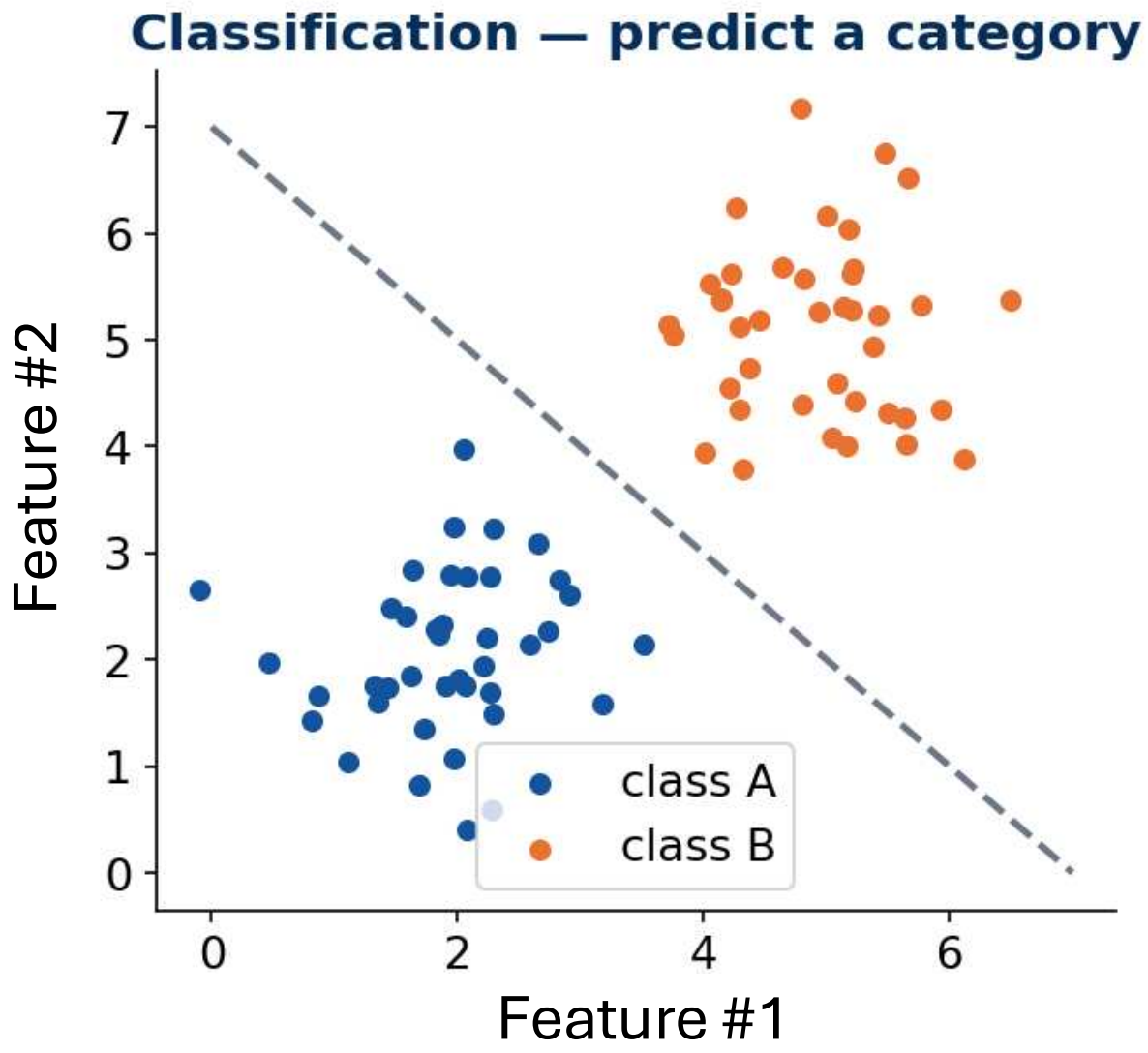
Prof. Nathan Szymanski
nszymanski@seas.ucla.edu
Department of Materials Science

UCLA

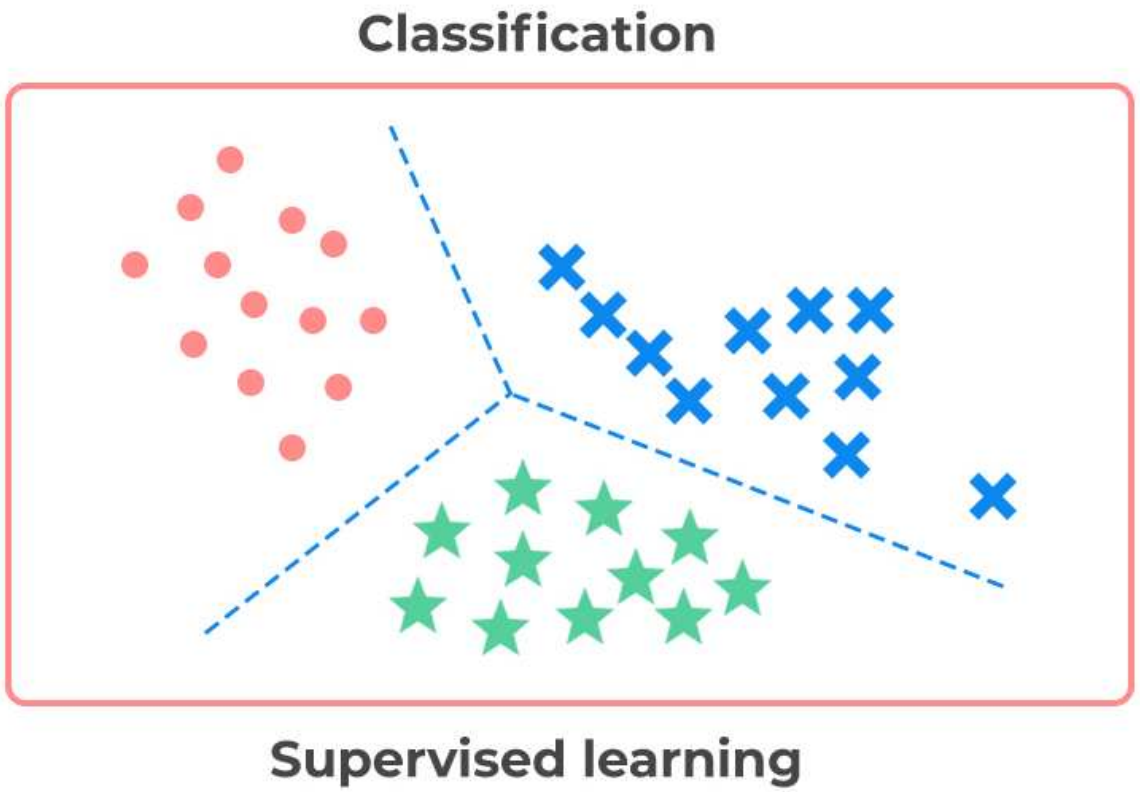
Samueli
School of Engineering

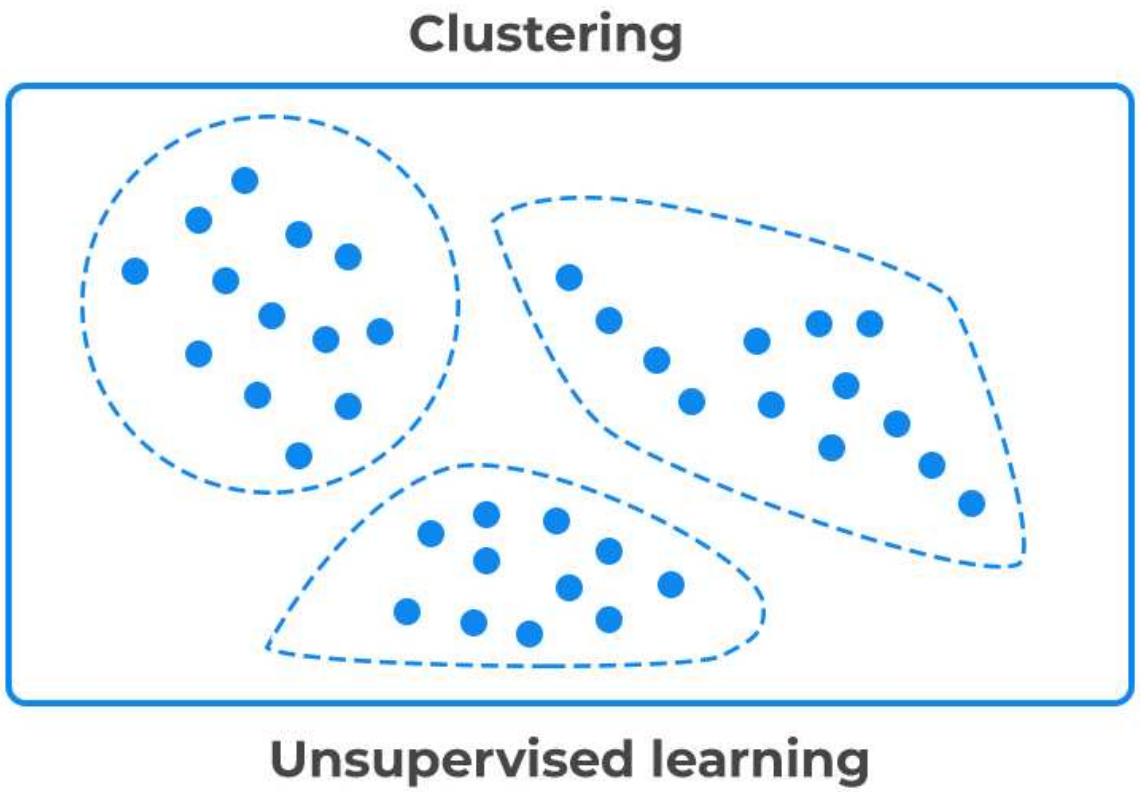
Regression *vs.* Classification





Supervised *vs.* Unsupervised learning

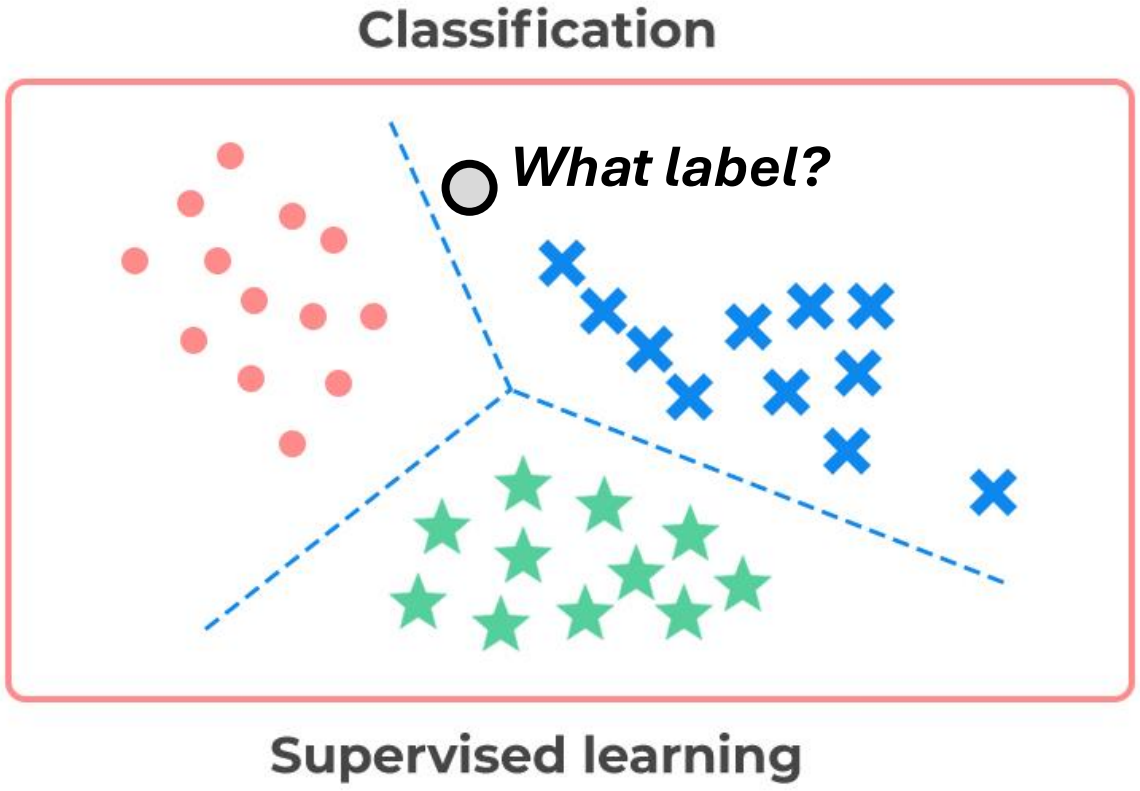




In this case, **we know the labels** for all datapoints

In this case, **we do *not* know the labels,** but we learn from the structure of the data

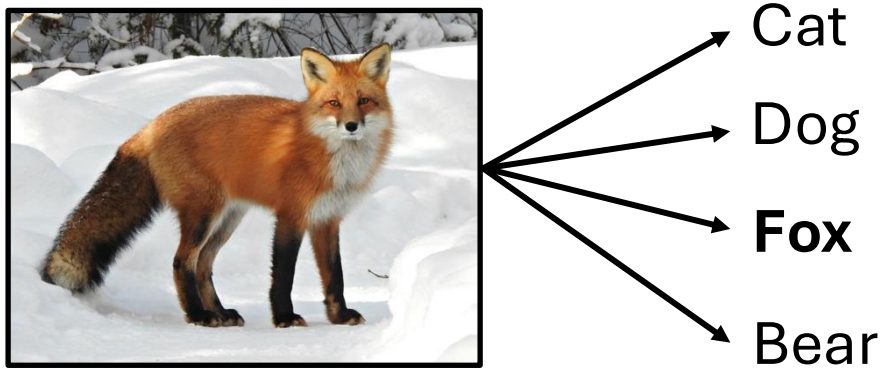
Supervised *vs.* Unsupervised learning



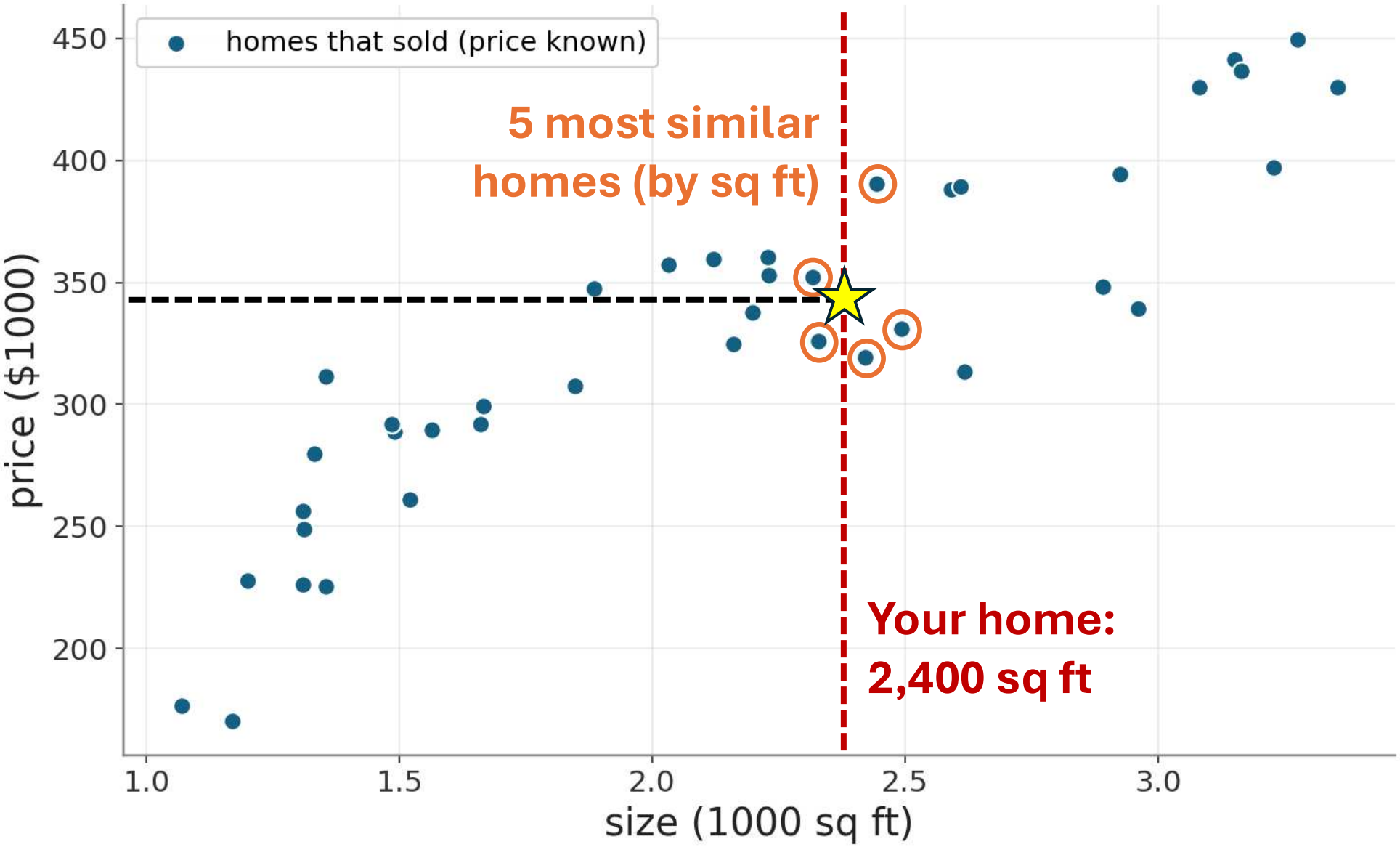
In this case, **we know the labels** for all datapoints

Similar to regression:
We train a model on known data and use it to predict labels on *new* data

Unlike regression:
We predict a discrete "category" (for example: cat, dog, fox, bear) instead of a continuous number



Today's focus: k-Nearest Neighbors



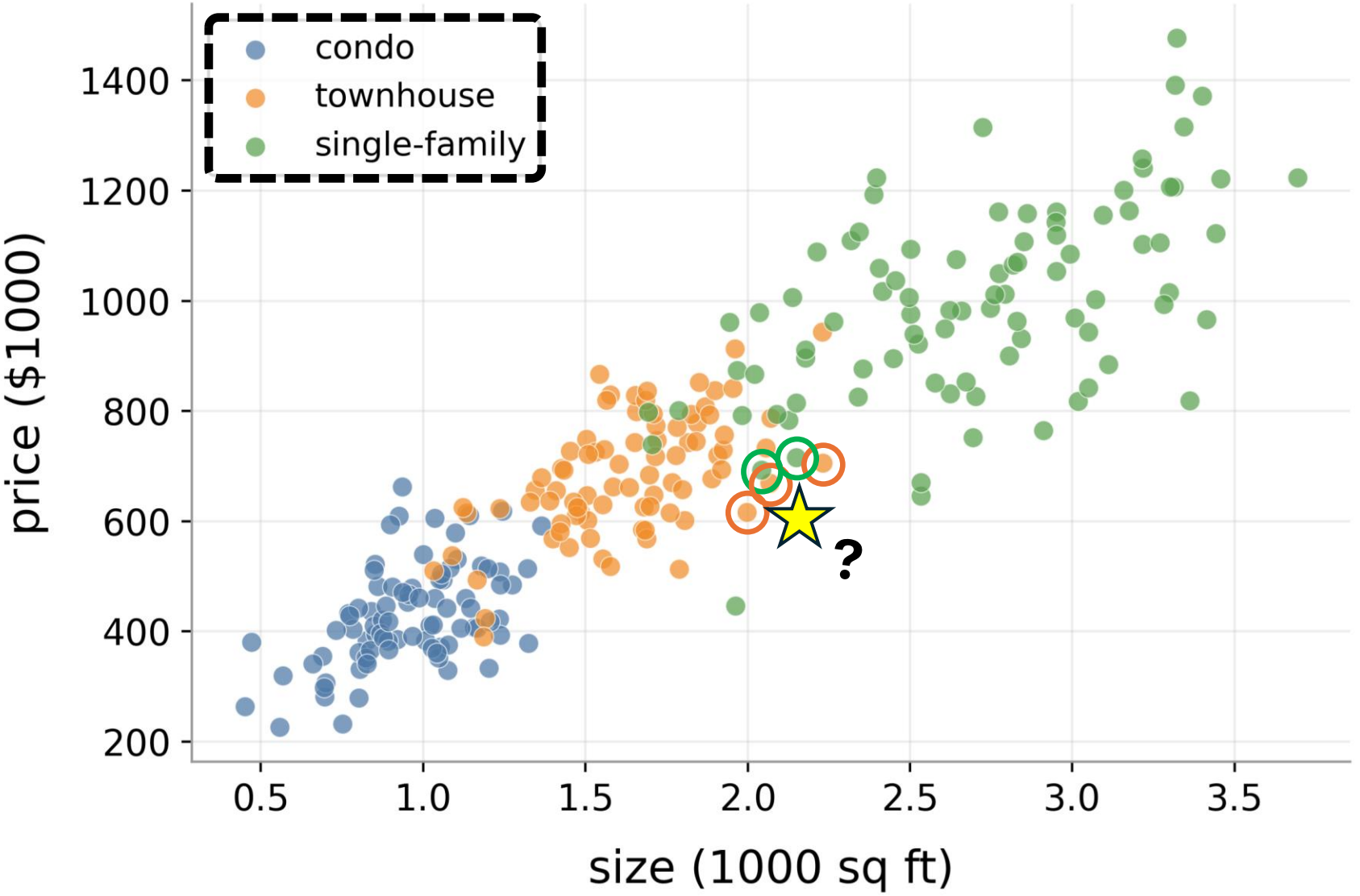
Size (1000 sq ft)	Sale price ($1000)
2.35	355
2.38	330
2.44	320
2.49	335
2.47	390

Average

$346k

This is *k*-Nearest Neighbors (*k* = 5) for *regression*

Today's focus: k-Nearest Neighbors



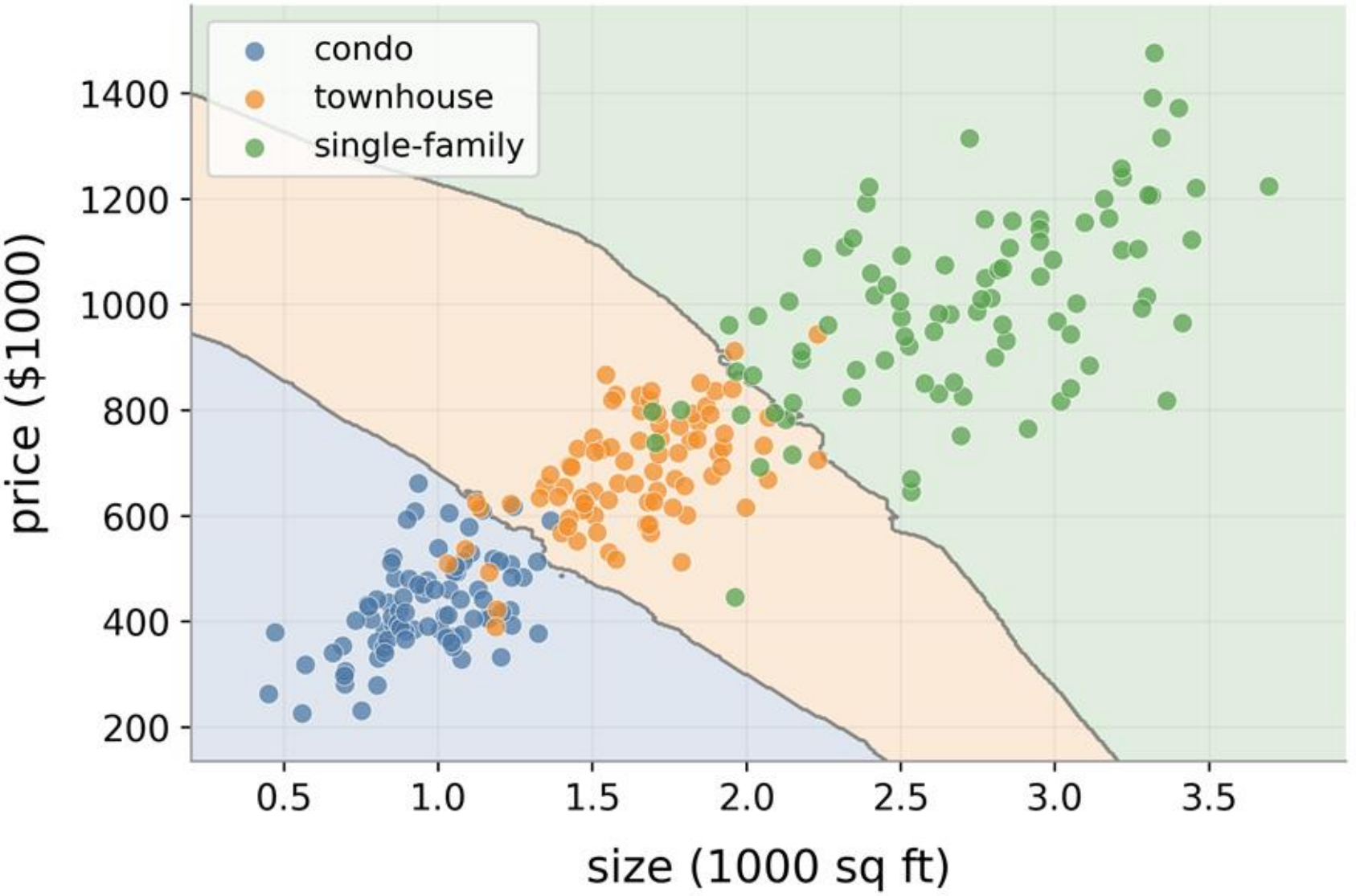
Size (1000 sq ft)	Sale price ($1000)	Home type
1.99	605	**Town.**
2.08	645	**Town.**
2.04	670	**Fam.**
2.15	710	**Fam.**
2.23	690	**Town.**

Vote

3/5 Townhouse

This is *k*-Nearest Neighbors (*k* = 5) for *classification*

Today's focus: k-Nearest Neighbors



We can repeat this process over a dense grid to create a **decision boundary** plot – which shows the regions that correspond to each label. Then classifying new points becomes nice & easy!

KNN for Classification: Step-by-step

Given some query (a new datapoint): what is its label?

Remember: KNN is "lazy" – we don't need to fit any model!

1. *Standardize* all the features (KNN is distance-based!)

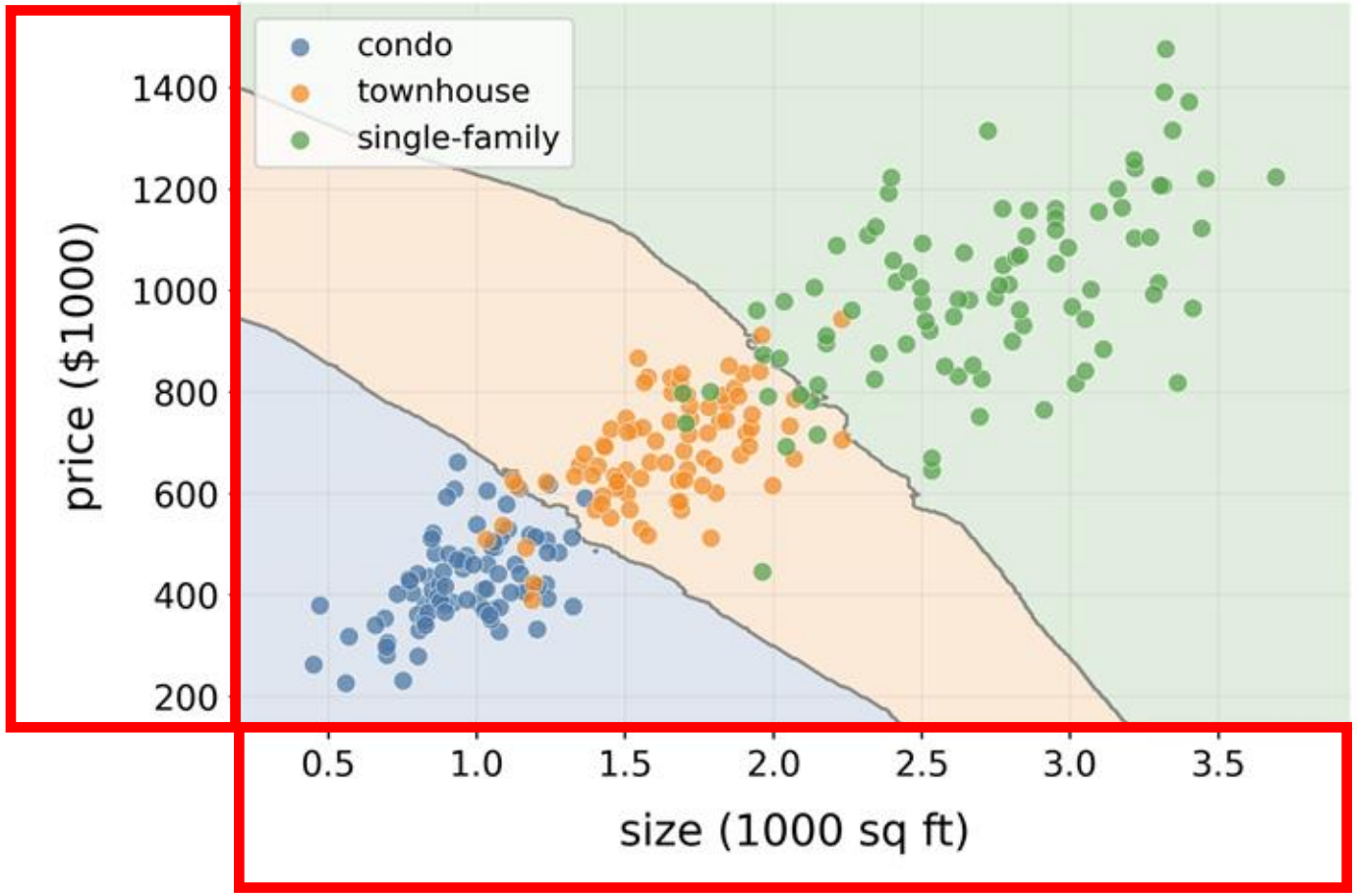
2. Compute the distance from the query to every point

3. Keep only the *k* nearest neighbors (smallest distances)

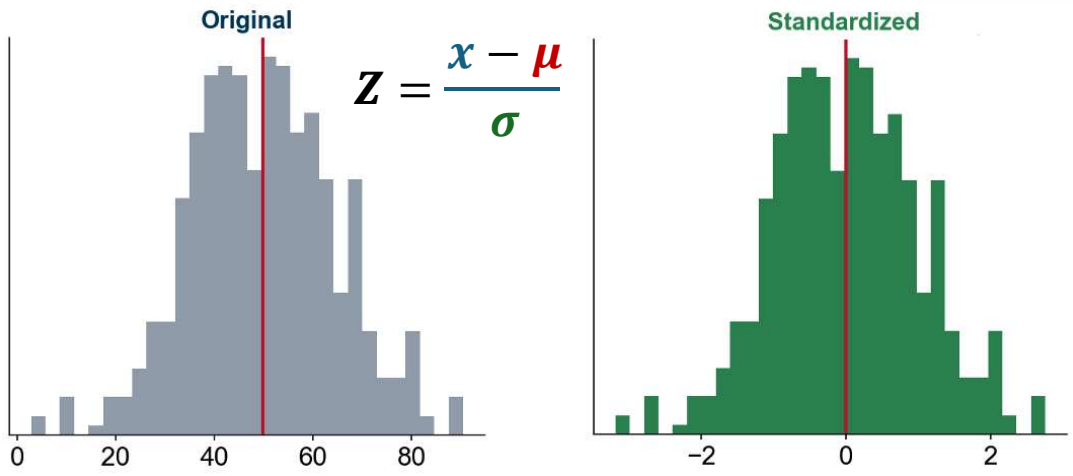
4. Predict the *majority* label among them

KNN for Classification: Step-by-step

1. *Standardize* all the features (KNN is distance-based!)

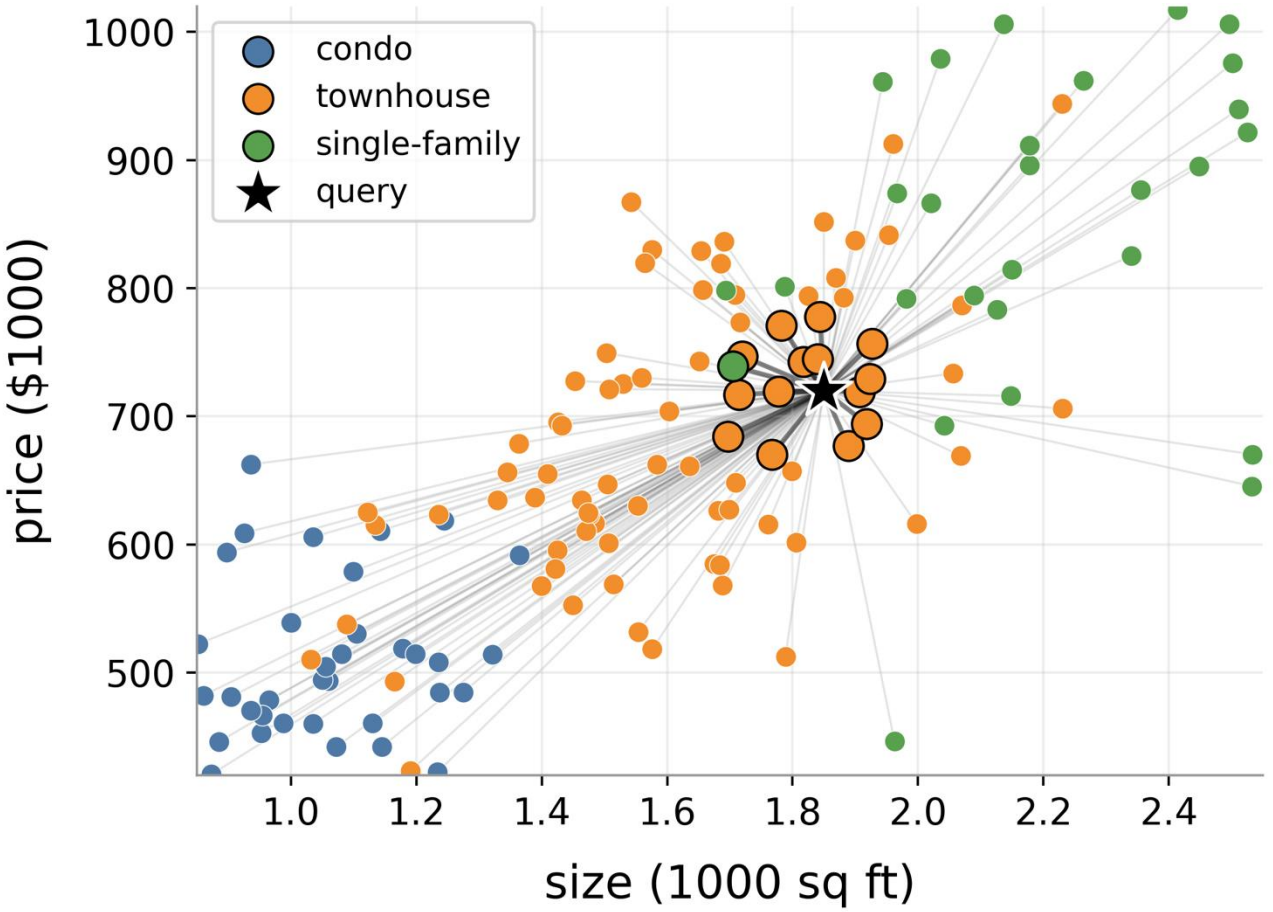


- These two features are ***not directly comparable***
- Without standardization, price would dominate the distances



KNN for Classification: Step-by-step

2. Compute the *distance* from the query to every point



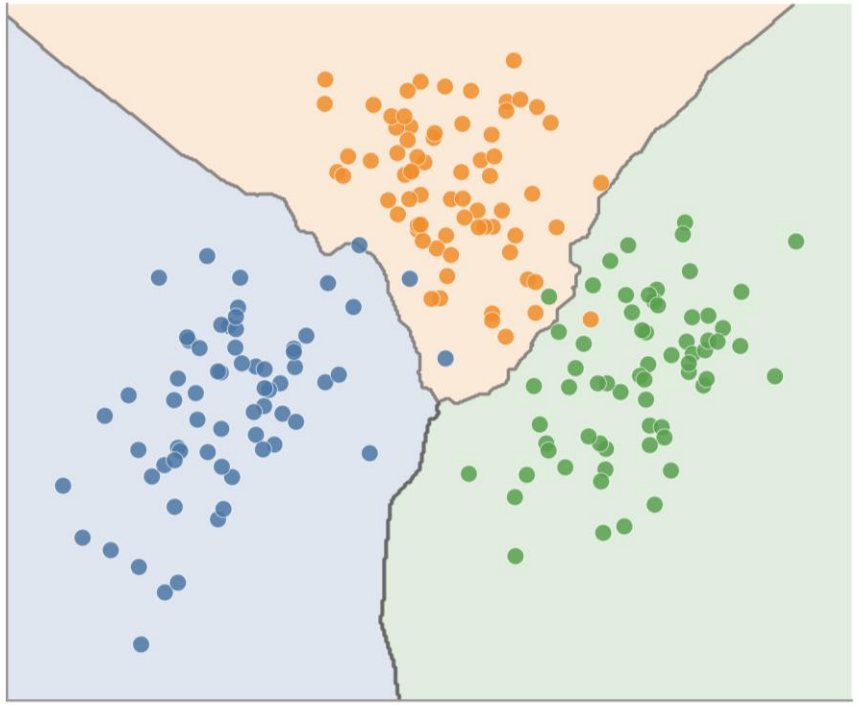
Euclidean distance is the most common:

$$D(x', x) = \sqrt{\sum_d |x'_d - x_d|^2}$$

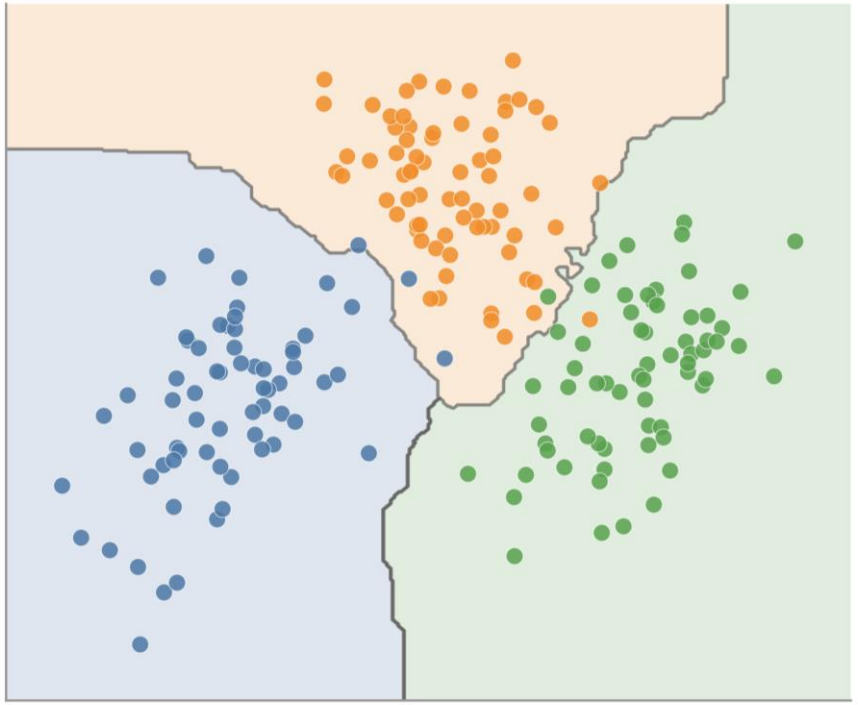
But remember…It's not the only distance measure! We may also use Manhattan distance.

The effects of the distance measure

Euclidean distance



Manhattan distance

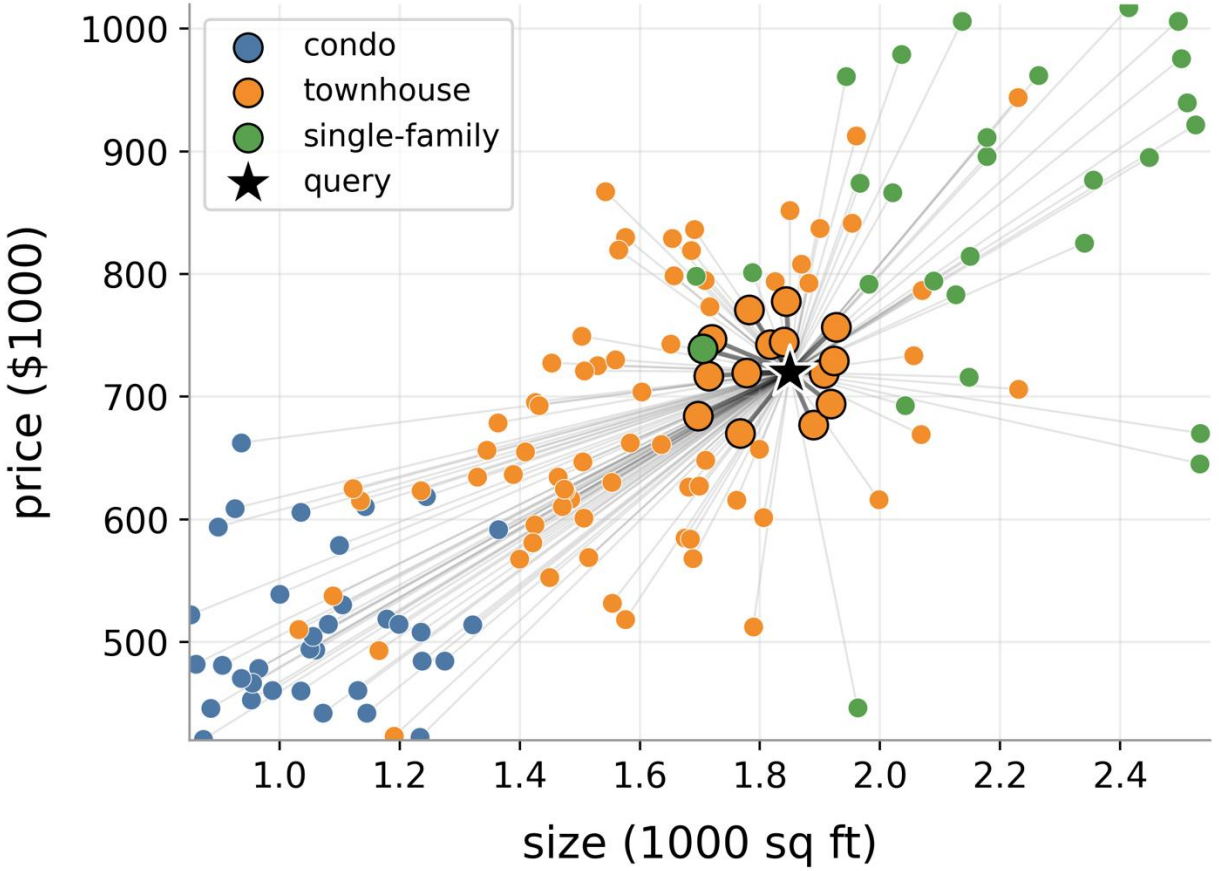


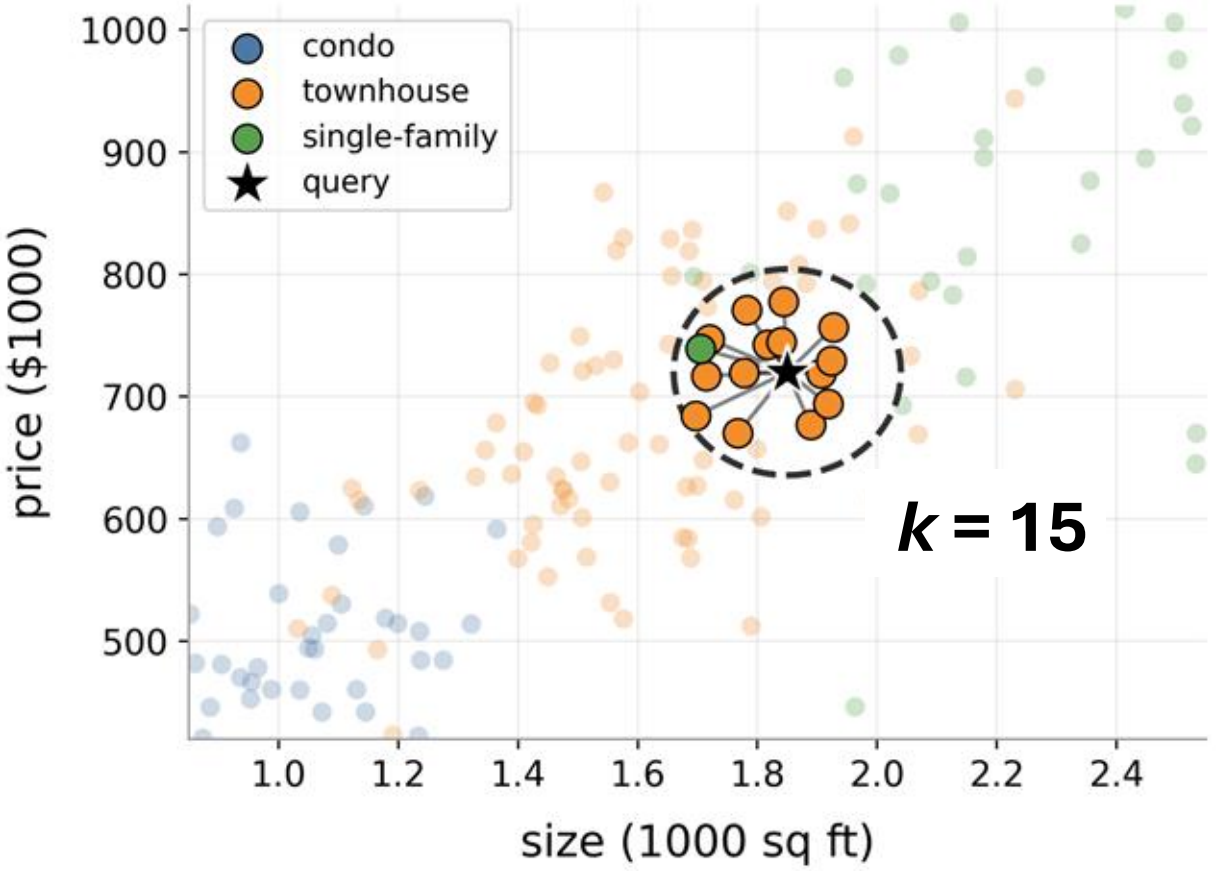
Euclidean distance tends to produce smoother, more rounded decision boundaries. This is the more common choice.

Manhattan distance tends to give more “blocky” decision boundaries. This is more often chosen if you are working in high dimensions.

KNN for Classification: Step-by-step

3. Keep only the *k* nearest neighbors (smallest distances)





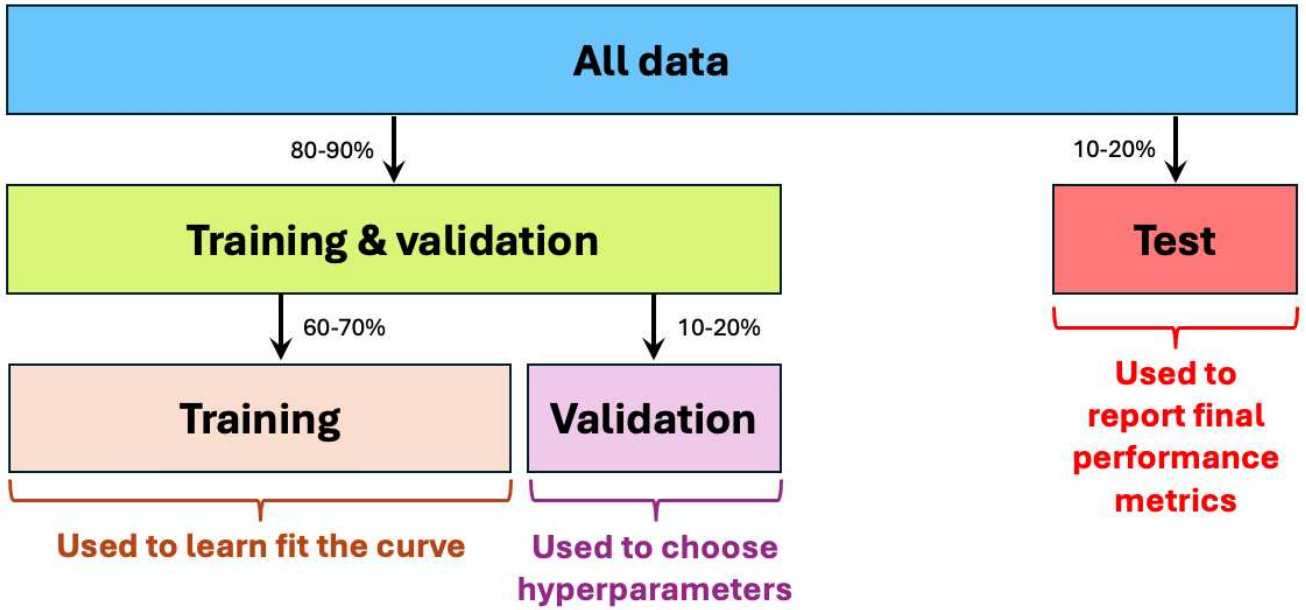
How to choose the number of neighbors (k)

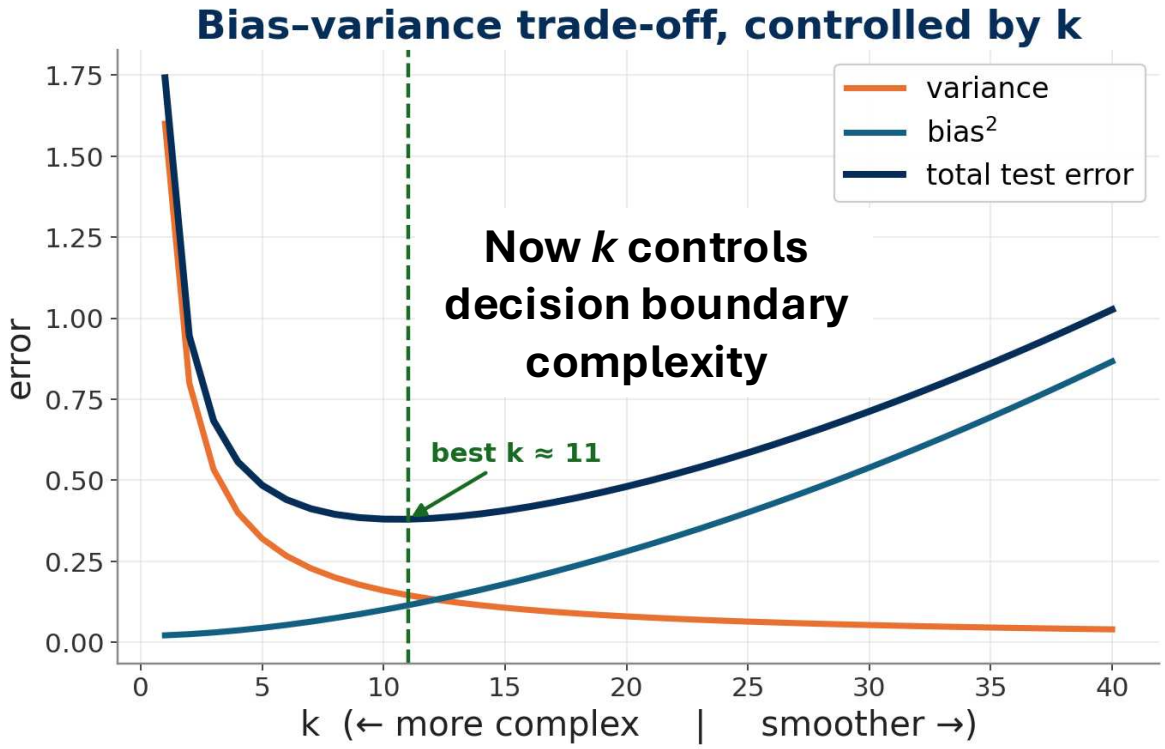
Small k
Flexible decision boundaries,
Sensitive to noise
Low bias, high variance

k is a **hyperparameter** to be tuned!

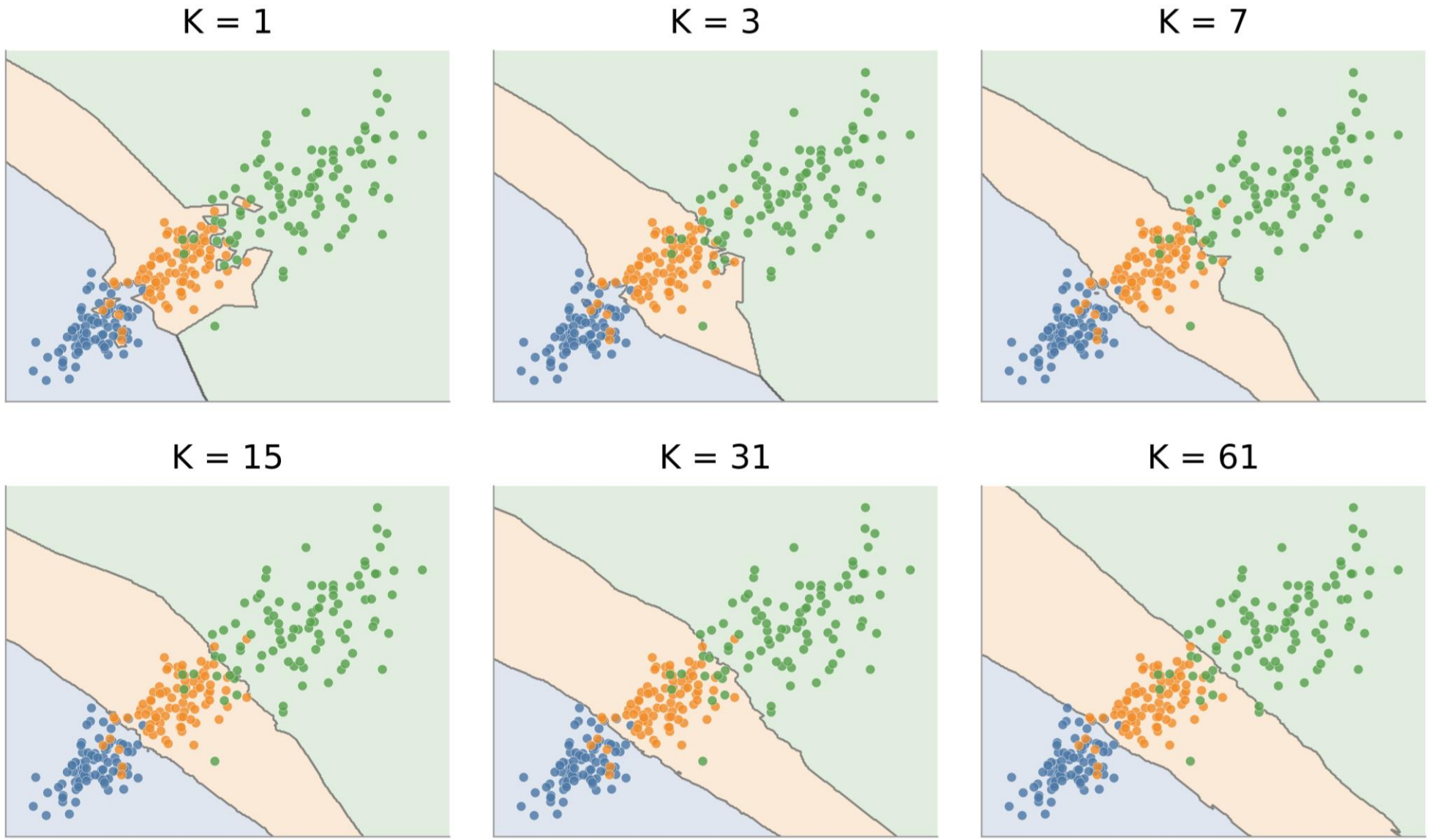
Large k
Smoother decision boundaries,
Can miss local structures
High bias, low variance

Remember we choose hyperparameters by **minimizing error on the validation set**





k has a big effect on decision boundaries!



How do we define error for classification?

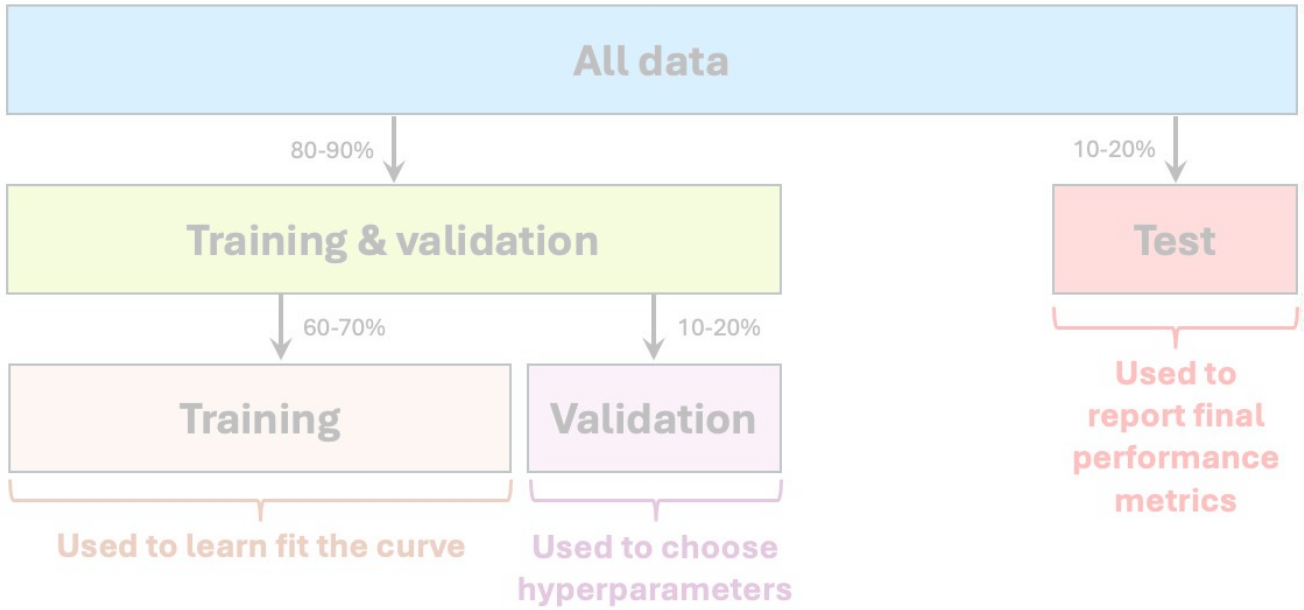
For **regression**, when comparing **continuous values**, we used:

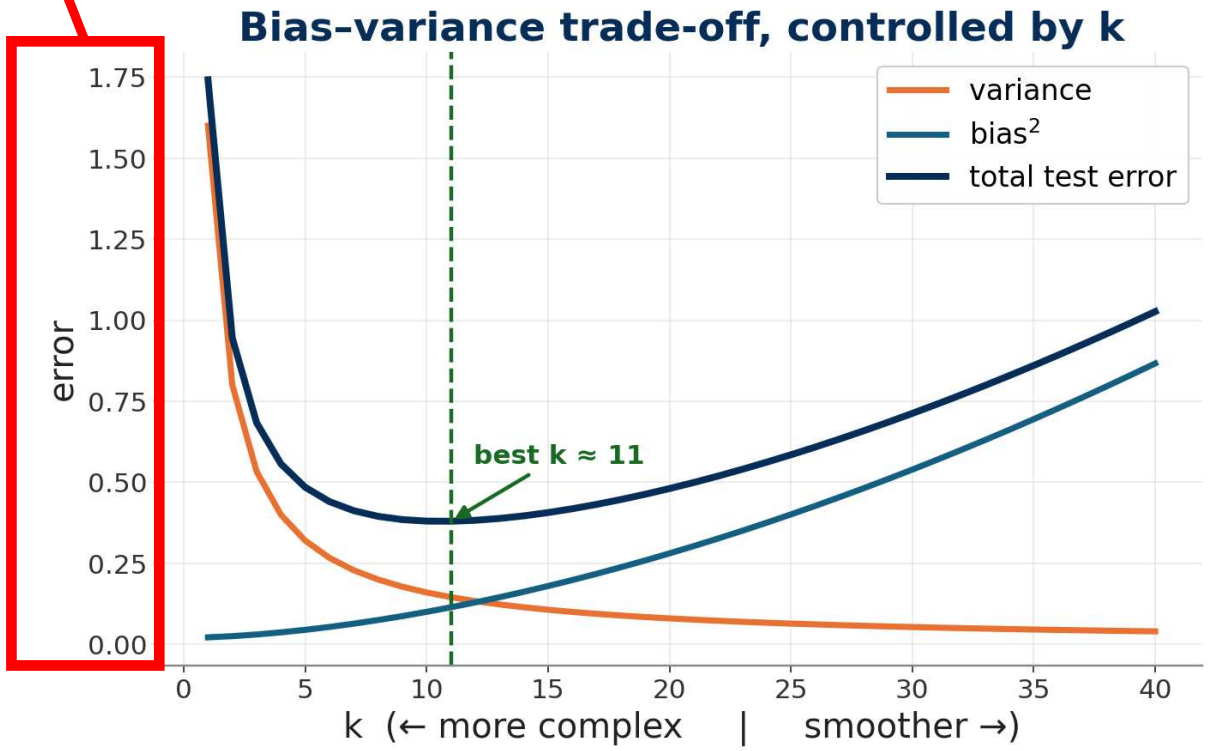
$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_{\text{actual}} - y_{\text{pred}}\right|^2$$

But how to quantify classification error?

Dog ≠ Cat

Remember we choose hyperparameters by **minimizing error on the validation set**



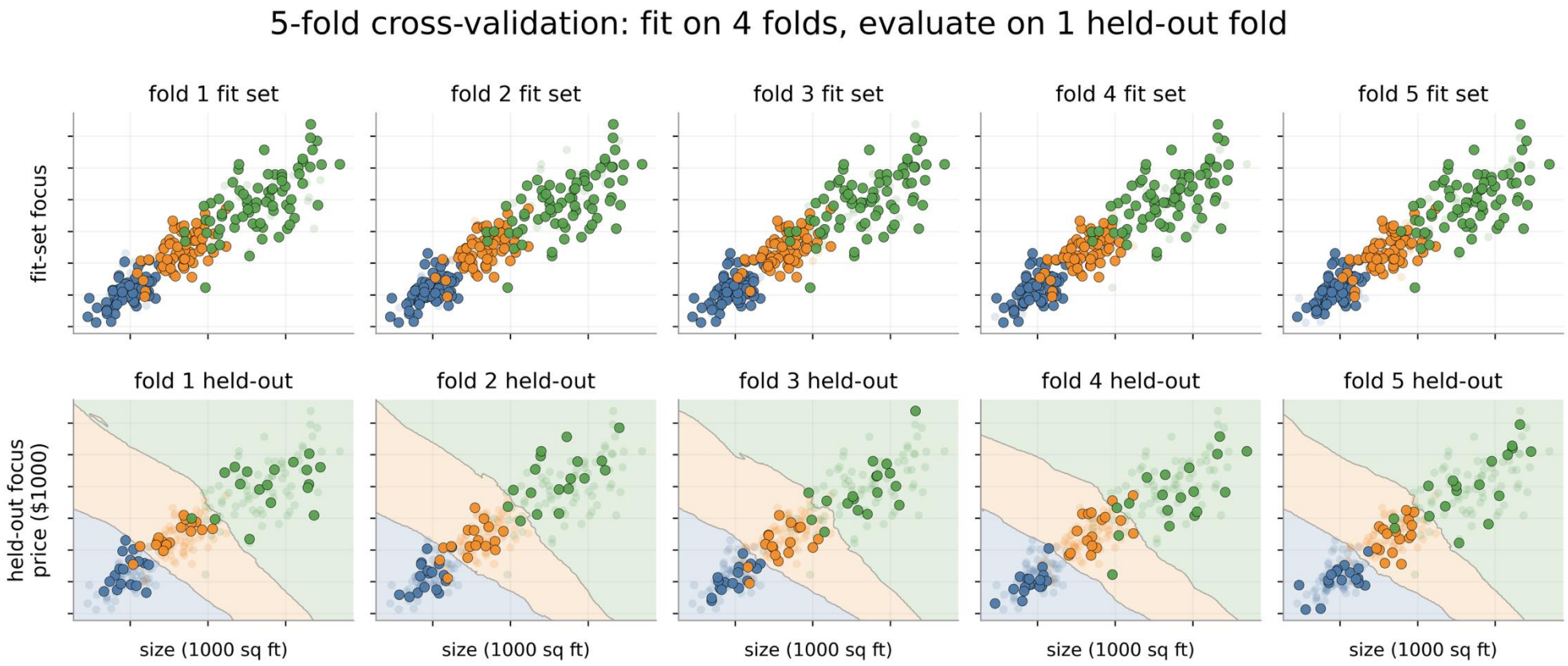


Classification error

$$\text{Error} = \frac{\#\ \text{Incorrect Classifications}}{\text{Total \# Samples}}$$

But how to quantify classification error?

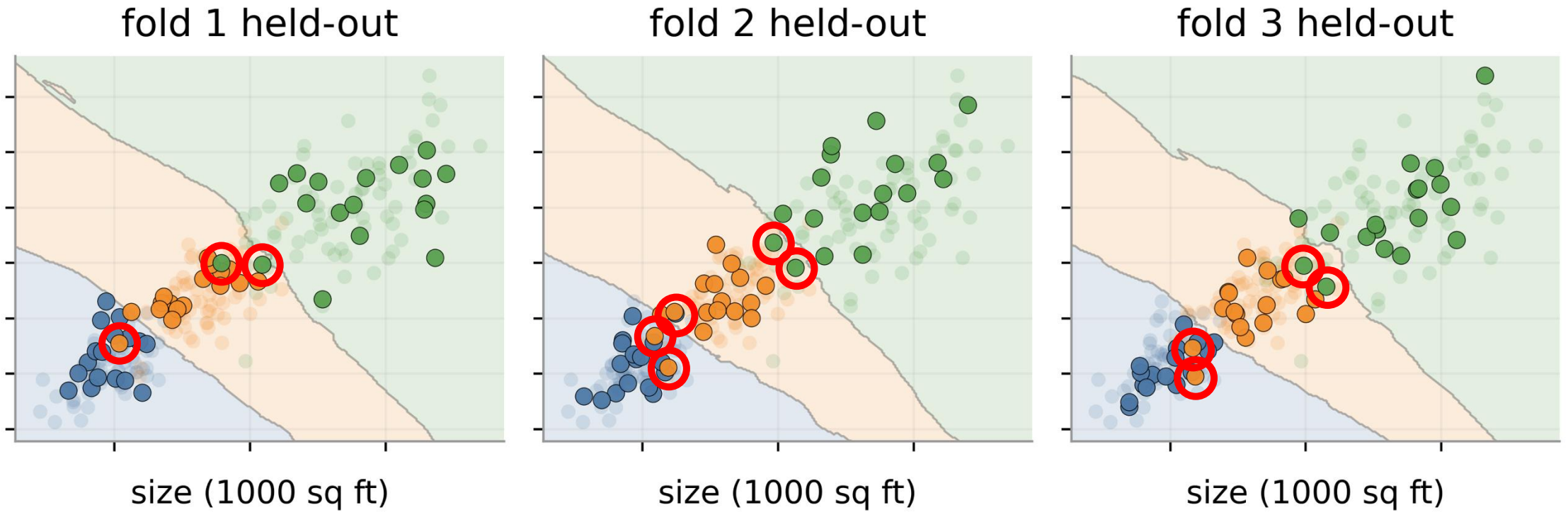
Dog ≠ Cat



Use these data to create decision boundaries

Use these to find classification error

Cross-validation to choose *k*



$$\frac{\textbf{3 Misclassifications}}{\textbf{52 Samples}} \approx 0.058 \text{ (5.8\% error)}$$

$$\frac{\textbf{5 Misclassifications}}{\textbf{52 Samples}} \approx 0.096 \text{ (9.6\% error)}$$

$$\frac{\textbf{4 Misclassifications}}{\textbf{52 Samples}} \approx 0.077 \text{ (7.7\% error)}$$

Cross-validation to choose *k*

It is also common to rewrite this as **classification accuracy: 100% - % error**

94.2% accuracy



$$\frac{\textbf{3 Misclassifications}}{\textbf{52 Samples}} \approx \textbf{0.058 (5.8\% error)}$$

90.4% accuracy



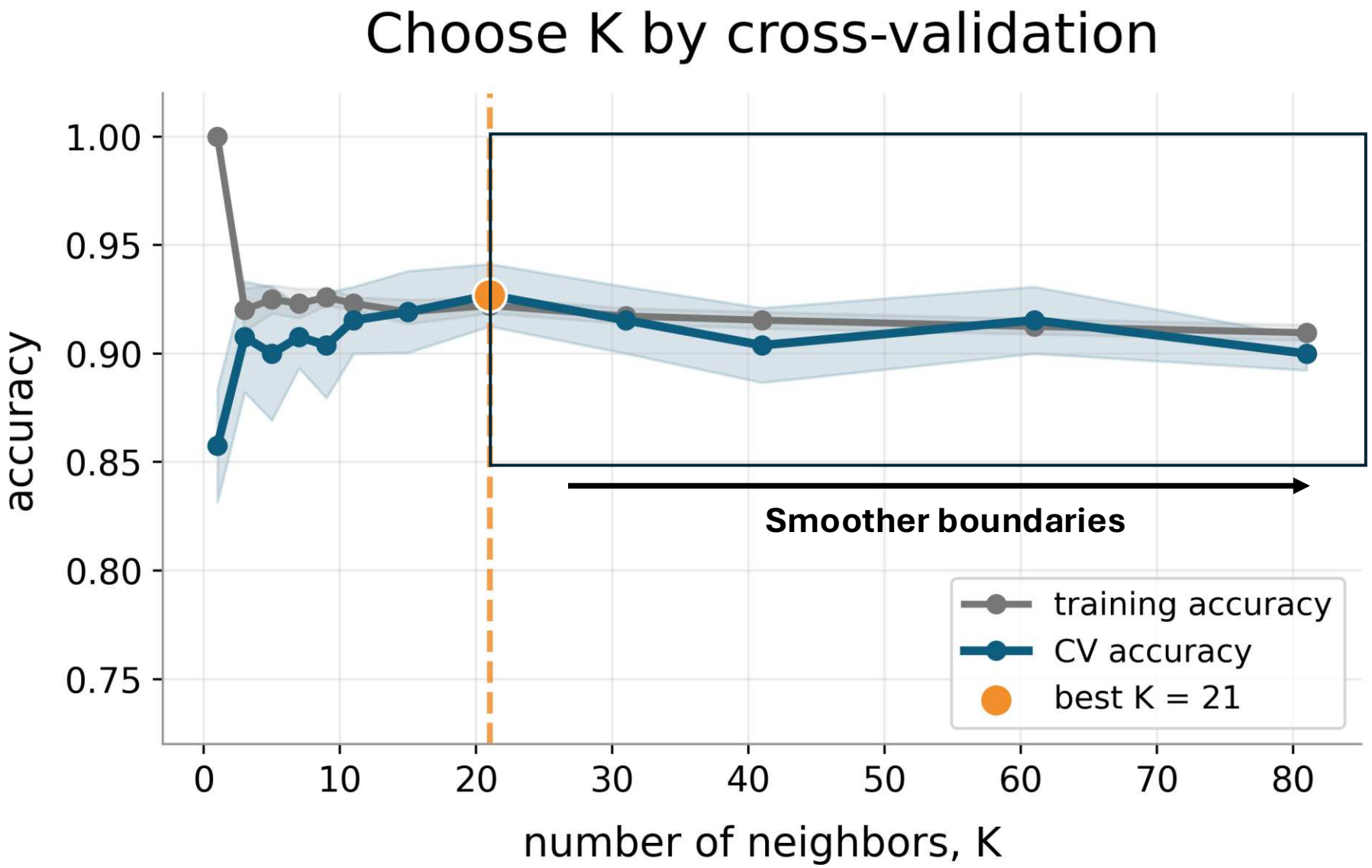
$$\frac{\textbf{5 Misclassifications}}{\textbf{52 Samples}} \approx \textbf{0.096 (9.6\% error)}$$

92.3% accuracy



$$\frac{\textbf{4 Misclassifications}}{\textbf{52 Samples}} \approx \textbf{0.077 (7.7\% error)}$$

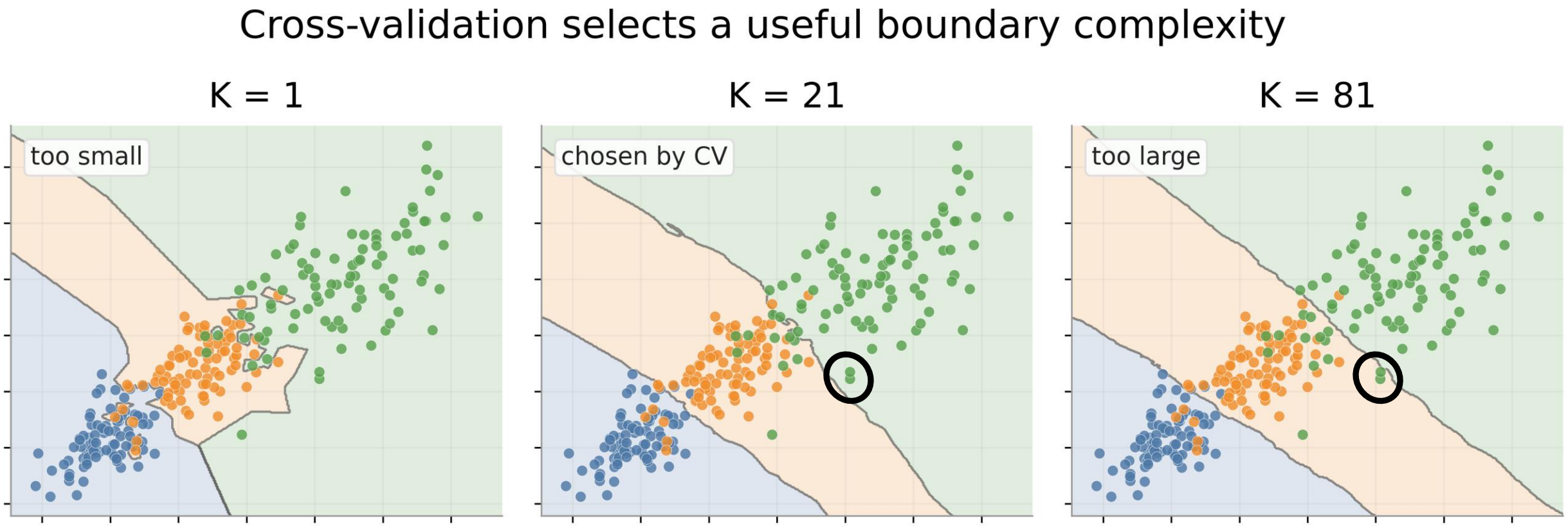
Cross-validation to choose *k*



Choose *k* where the cross-validation accuracy is highest (or close to highest)

The decrease in accuracy as we raise *k* is often quite subtle! But it still matters…we miss **local structure**

Cross-validation to choose *k*



Clearly not good…It tries to isolate noisy points

This looks reasonable at first glance, but it misses points near the boundary

KNN for Classification: Step-by-step

Given some query (a new datapoint): what is its label?

Remember: KNN is "lazy" – we don't need to fit any model!

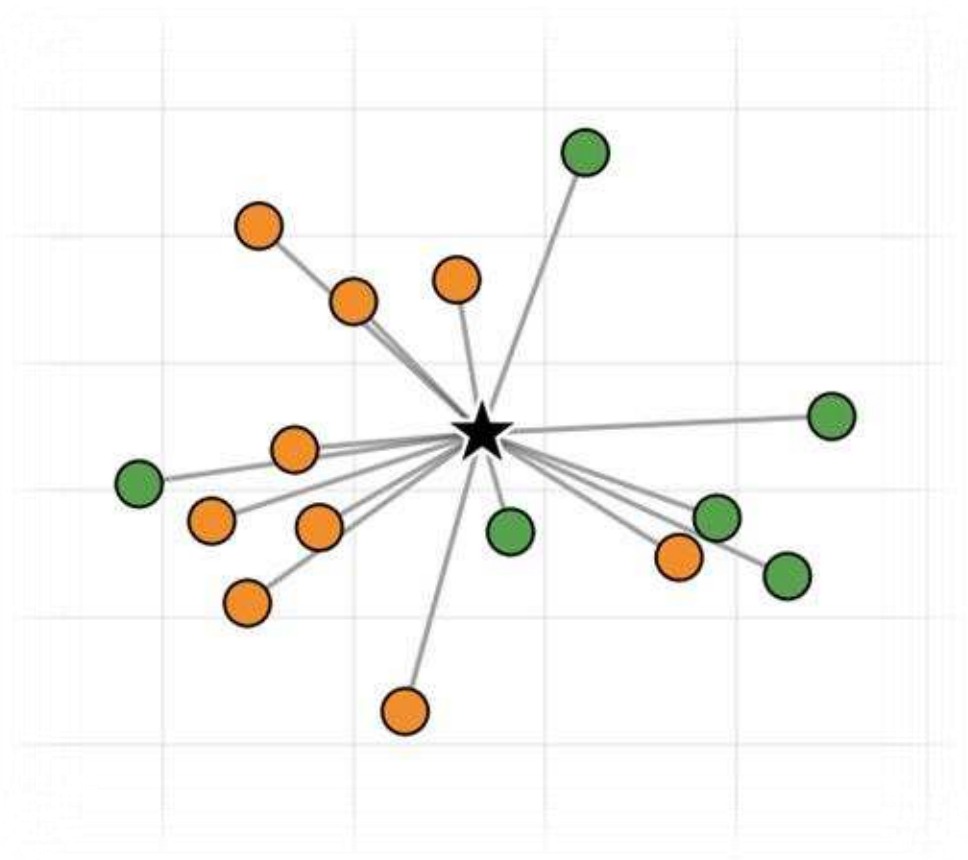
1. ***Standardize*** **all the features (KNN is distance-based!)**

2. **Compute the distance from the query to every point**

3. **Keep only the *k* nearest neighbors (smallest distances)**

4. Predict the *majority* label among them

How to label the query based on its neighbors



$$\hat{y}_{\text{query}} = \underset{c \in \{1,\dots,C\}}{\operatorname{argmax}} \sum_{i \in N_k(x_{\text{query}})} 1(y_i = c)$$

Possible labels (green/blue)

How many neighbors with each label

9 orange *vs.* 6 green

→ Label the query orange **with probability 60%**

(9/15)

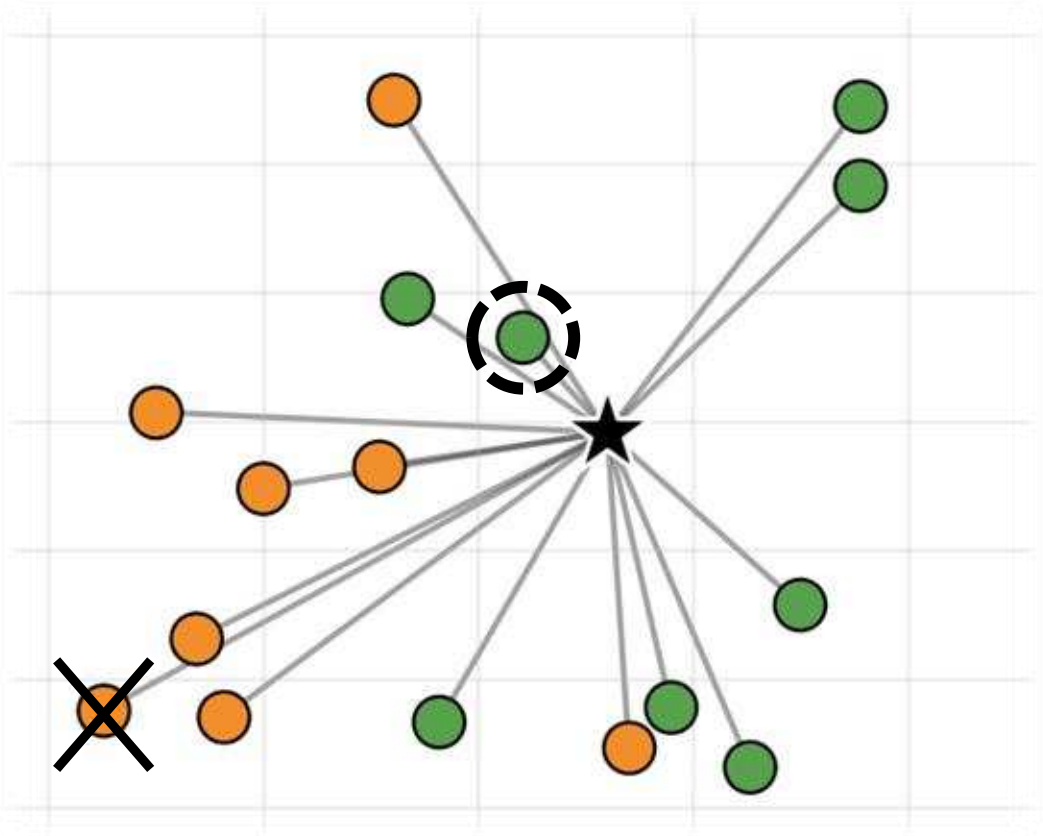
4. Predict the *majority* label among them

How to label the query based on its neighbors

What if there are an equal number of neighbors from each class?

Option 1: Reduce *k* by 1 (make it odd)
→ Drop the furthest point and recompute the label

Option 2: Assign label based on the nearest neighbor (tie-breaker)



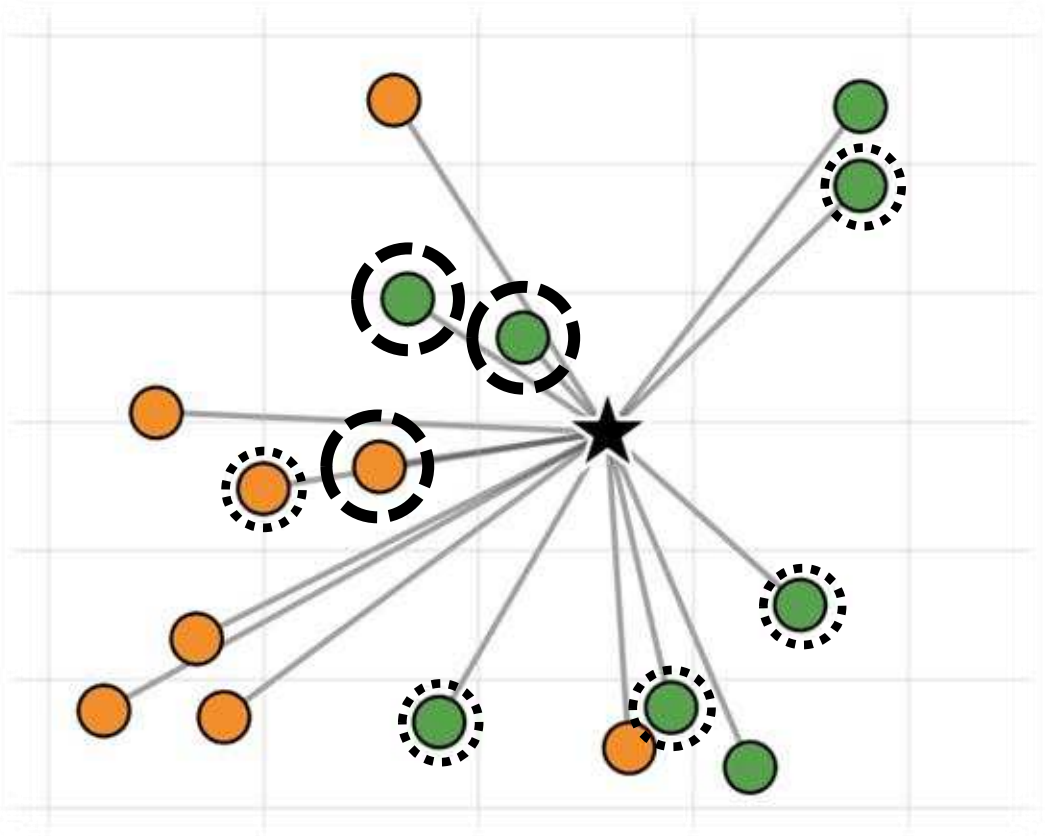
4. Predict the *majority* label among them

How to label the query based on its neighbors

Option 3: Distance-weighted vote

$$\sum_{i \in N_k(x_{\text{query}})} \frac{1(y_i = c)}{d(x_{\text{query}}, x_i)}$$

Closer neighbors (lower d) are given a higher weight in their votes



4. Predict the *majority* label among them

Class probability from each voting scheme

Choosing the majority vote

$$\hat{y}_{\text{query}} = \underset{c \in \{1,\ldots,C\}}{\operatorname{argmax}} \sum_{i \in N_k(x_{\text{query}})} 1(y_i = c)$$

$$P_{\text{majority}} = \frac{\text{number of majority votes}}{\text{total number of votes}}$$

Distance weighting

$$\sum_{i \in N_k(x_{\text{query}})} \frac{1(y_i = c)}{d(x_{\text{query}}, x_i)}$$

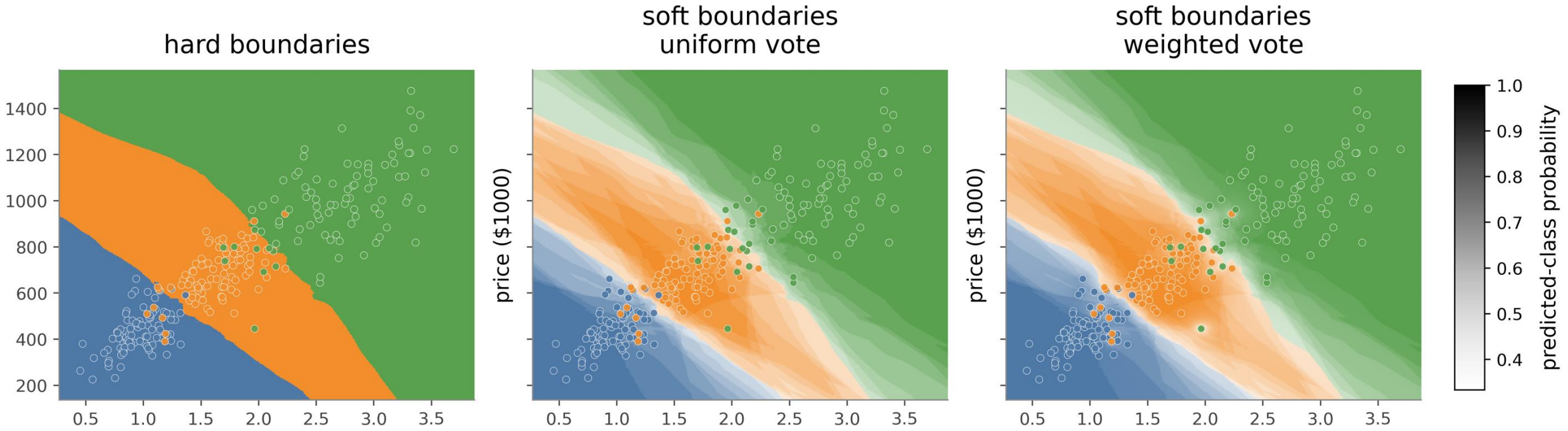
$$P_{\text{majority}} = \frac{\sum_{i \in N_k(x_{\text{query}})} w_i \cdot 1(y_i = c)}{\sum_{i \in N_k(x_{\text{query}})} w_i}$$

Where...

$$w_i = \frac{1}{d(x_{\text{query}}, x_i)}$$

$$P_{\text{majority}} = \frac{\textit{weighted votes from majority class}}{\textit{weighted votes from all classes}}$$

Probability surface = a “soft” map of classes



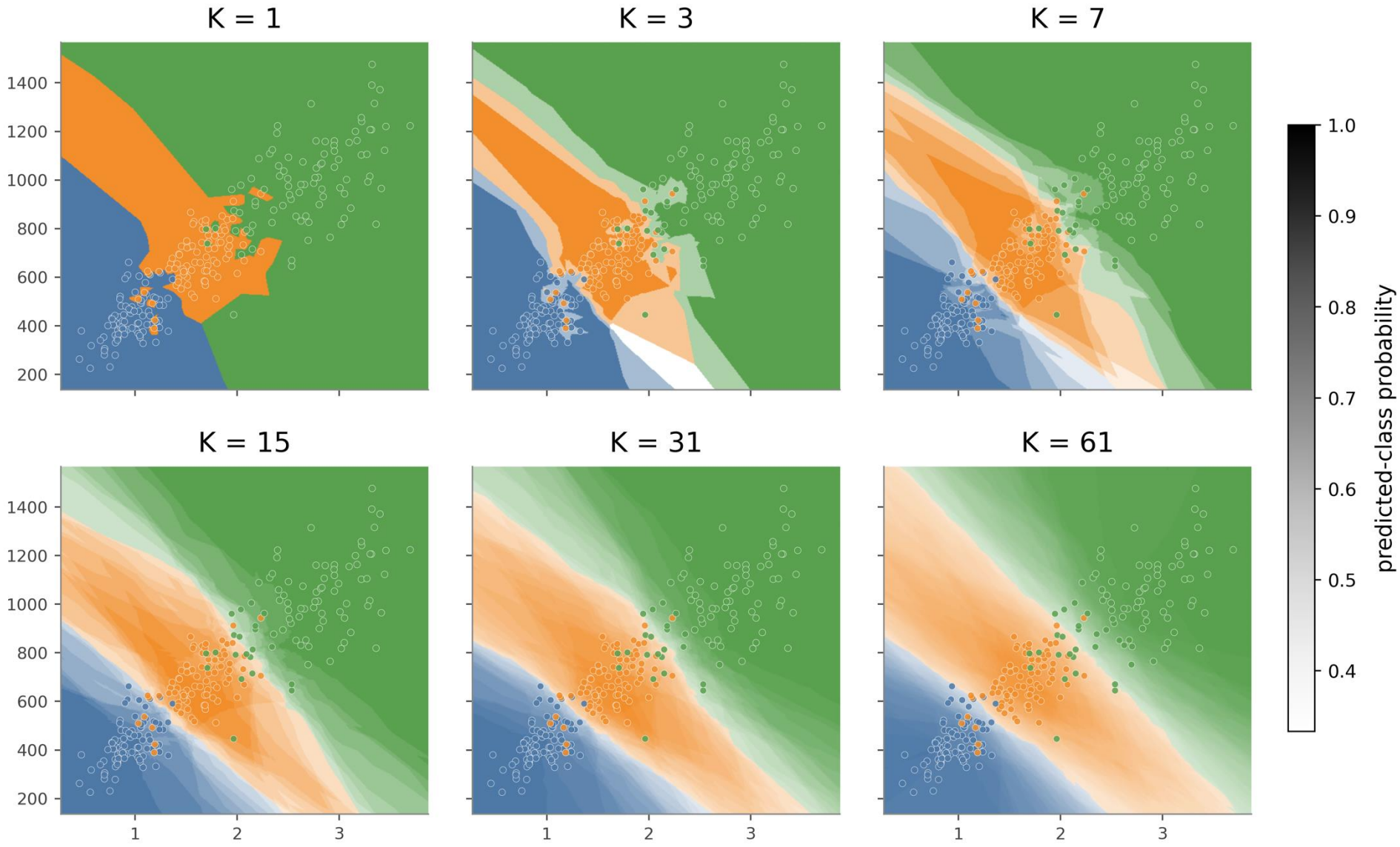
These are also known as ***probability surfaces.***

Often more instructive than hard decision boundaries!

It let’s us know whether our classification is *uncertain*.

Influence of *k* on the probability surface

Larger *k* leads to smoother probability surfaces, as more neighbors "vote" in each classification, and changes can be smaller (*e.g.*, 1/20 votes)

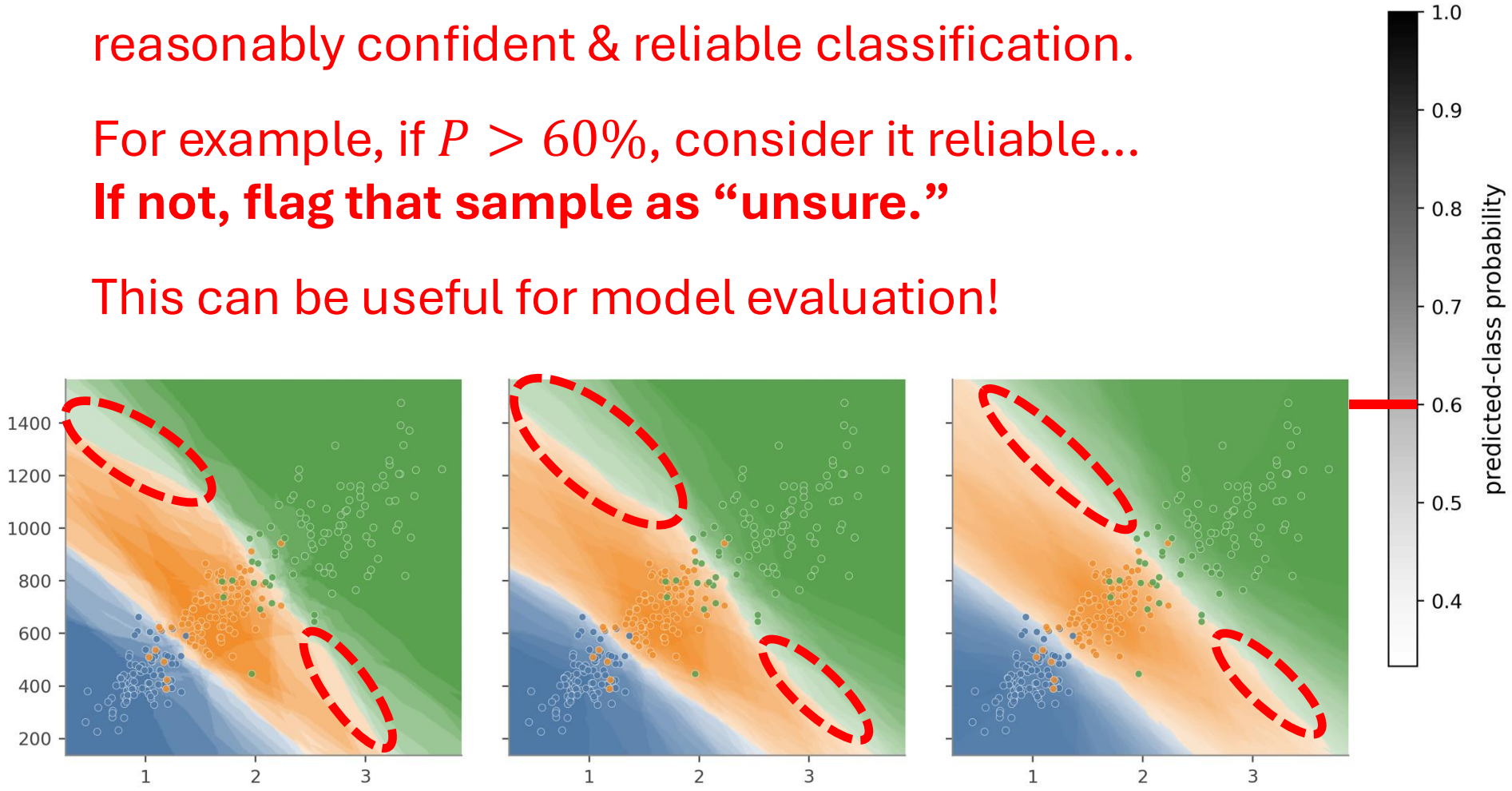


Probability thresholds

We often use a **threshold** on what we consider a reasonably confident & reliable classification.

For example, if $P > 60\%$, consider it reliable…
If not, flag that sample as "unsure."

This can be useful for model evaluation!



Questions?

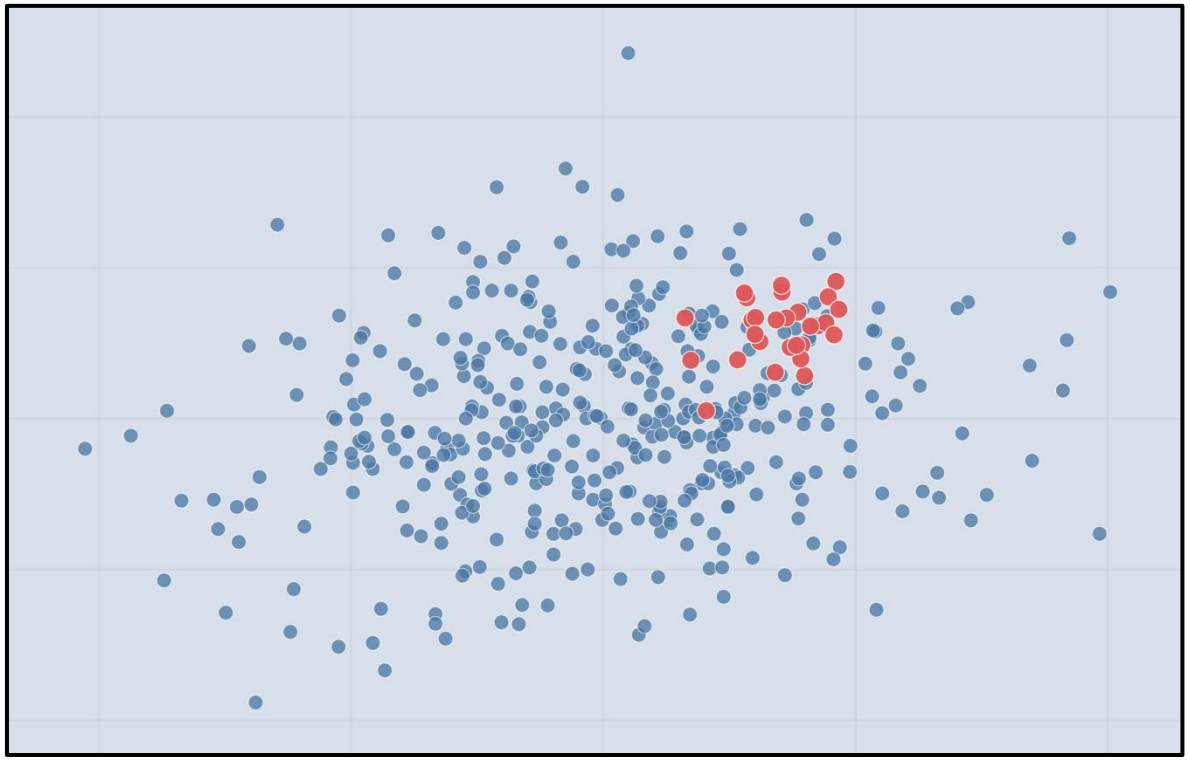
Evaluating a Classifier

Accuracy alone is not enough!

$$\% \text{ Error} = \left(\frac{\#\text{ Incorrect Classifications}}{\text{Total \# Samples}} \right) \cdot 100\%$$

$$\text{Accuracy} = 100 - \% \text{ Error}$$

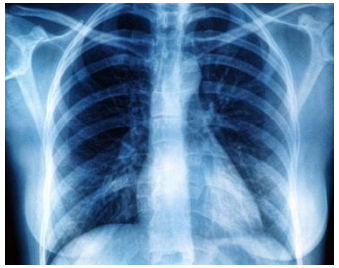
A classifier can have 99% accuracy yet still be useless! Let's see why...



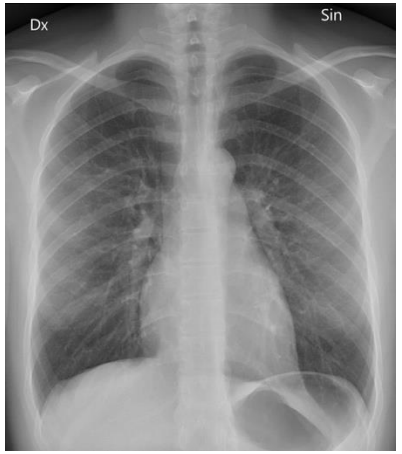
On this highly **imbalanced** dataset, our KNN classifier with $k = 100$ **achieves 94% accuracy**... By just predicting the same class over the entire feature space. No real insights gained!

Why do we care? Is this realistic?

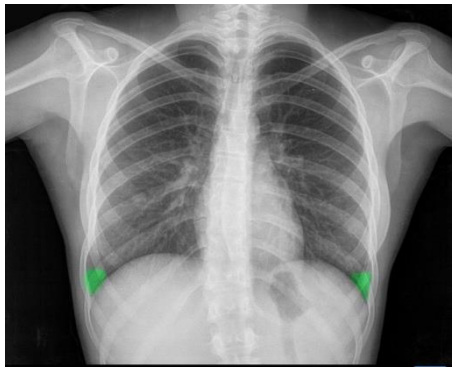
Accuracy alone is not enough!













Does ***not*** have lung cancer (99% of cases)

You can get **99% accuracy** by just predicting **nobody has cancer…**

Misunderstanding your model can have dire consequences 💀

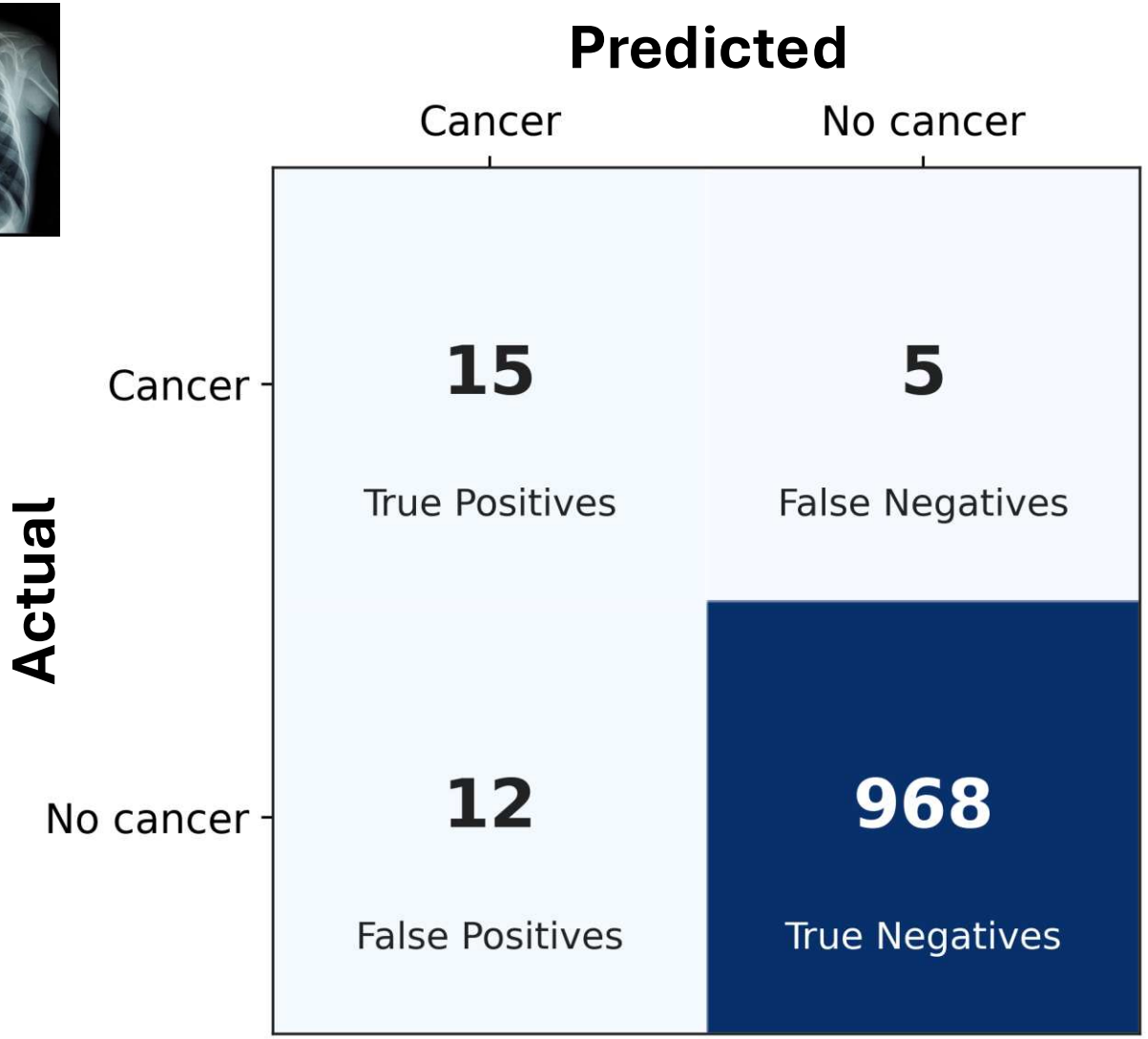
Does have lung cancer (1% of cases)

The confusion matrix

We can visualize model performance by using a **confusion matrix**

It contains 4 values:





The confusion matrix

We can visualize model performance by using a **confusion matrix**

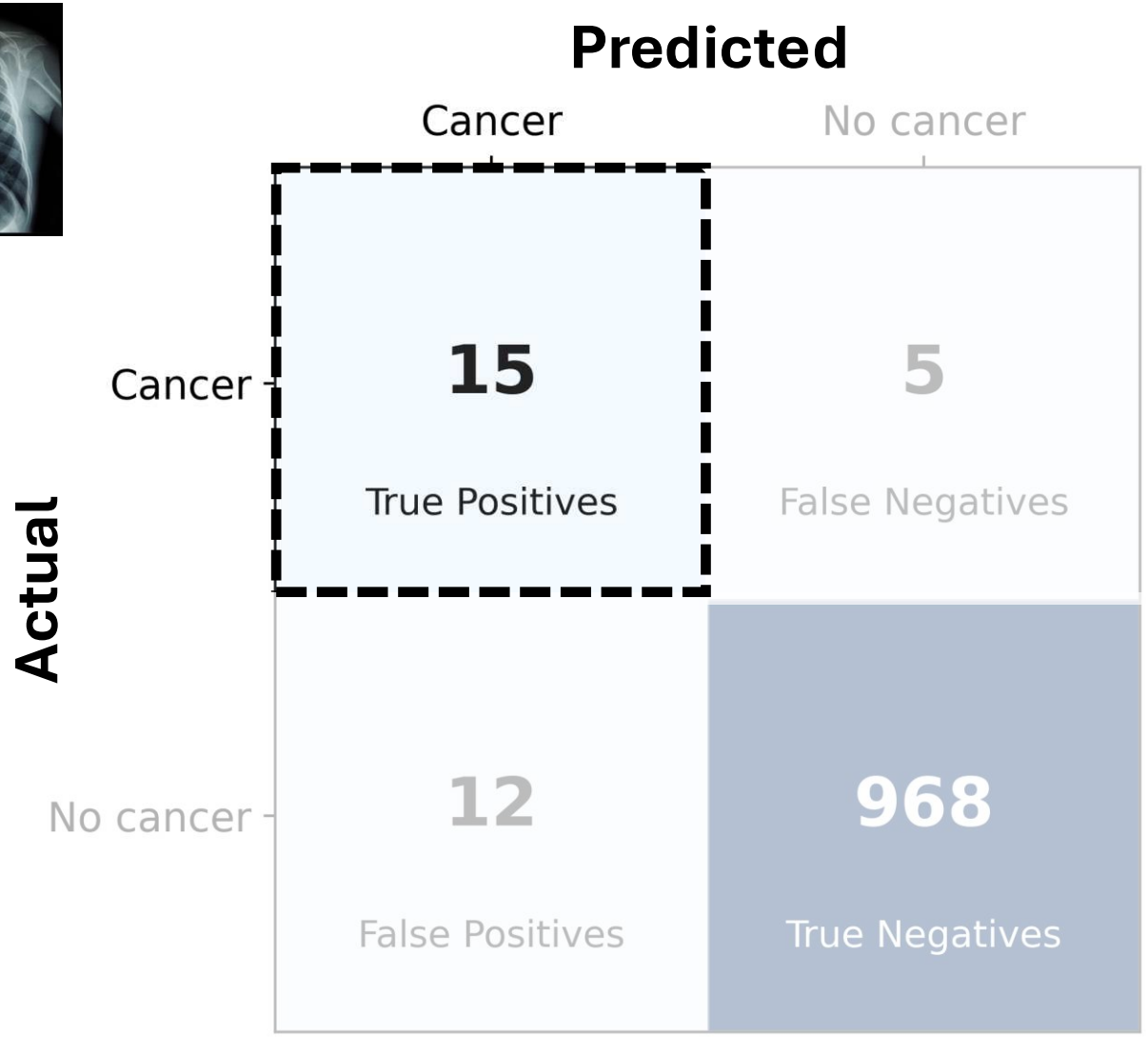


It contains 4 values:

True Positives

- Model ***predicts*** cancer
- Patient ***actually has*** cancer

Prediction and ground truth agree.
This is good! (though not for our patient)



The confusion matrix

We can visualize model performance by using a **confusion matrix**

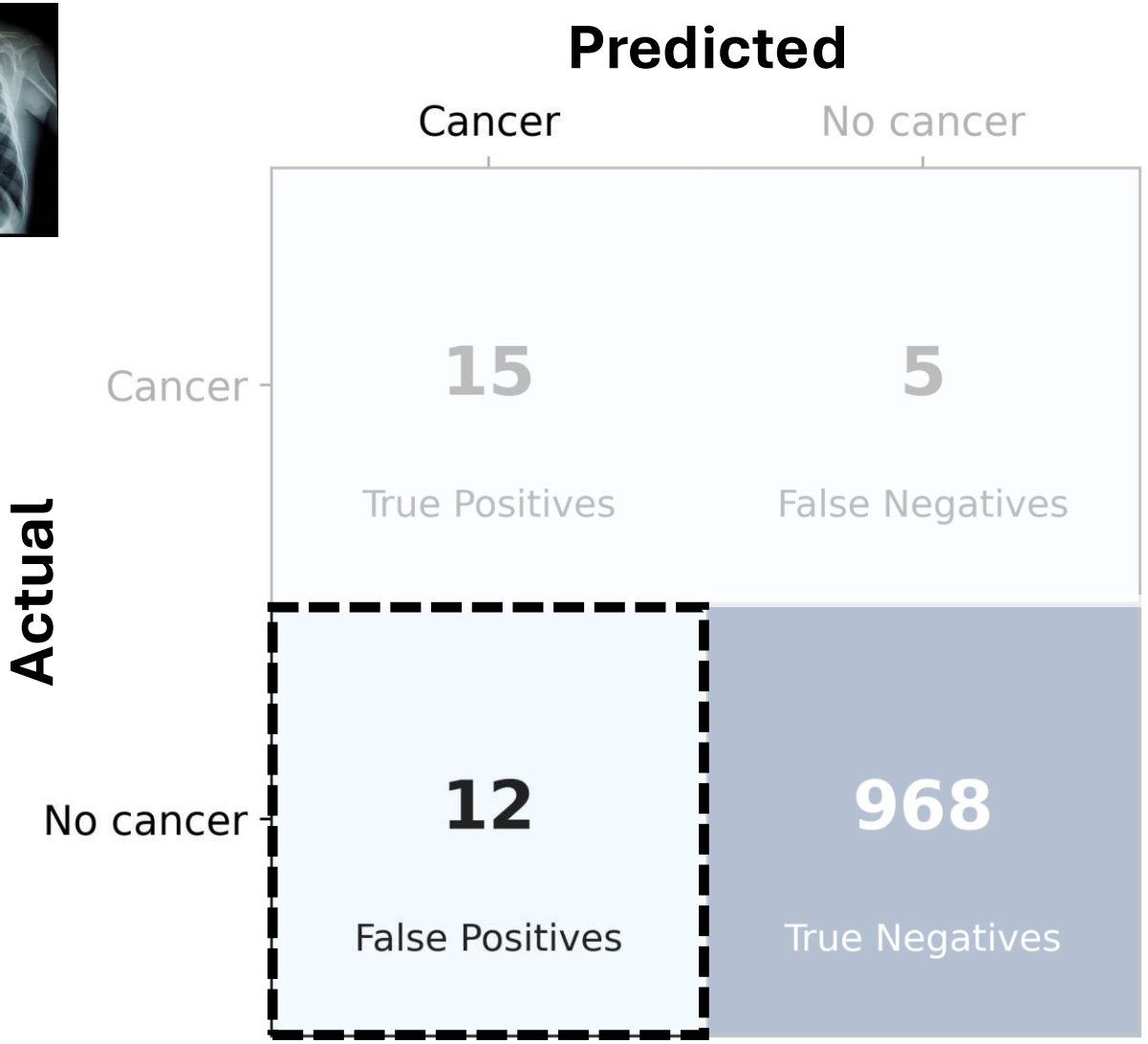


It contains 4 values:

False Positives

- ✅ Model ***predicts*** cancer
- ❌ Patient ***does not have*** cancer

Prediction and ground truth disagree. This is (sort of) bad.



The confusion matrix

We can visualize model performance by using a **confusion matrix**

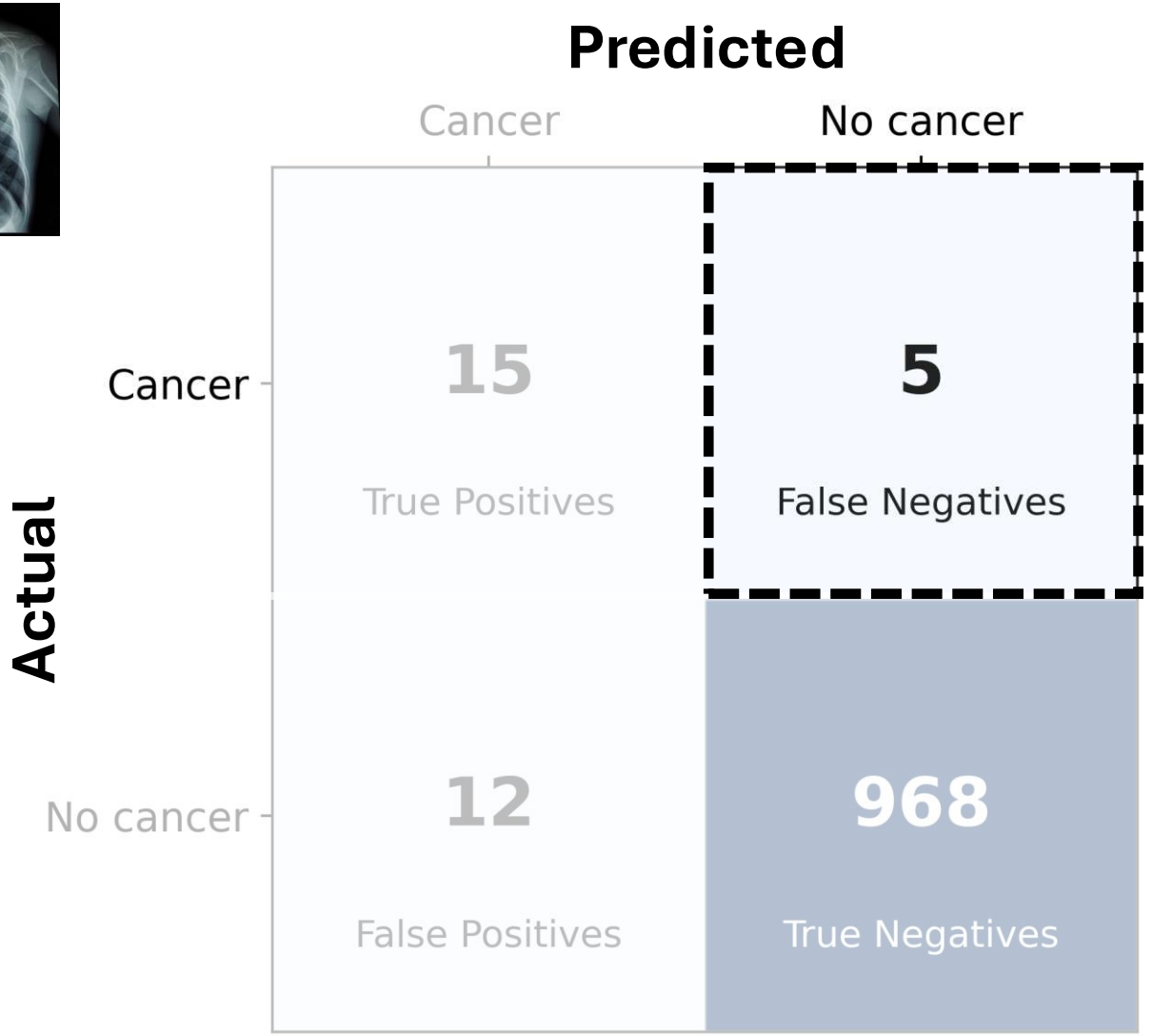


It contains 4 values:

False Negatives

- ✘ Model ***predicts no*** cancer
- ✔ Patient ***does have*** cancer

Prediction and ground truth disagree.
This is *very* bad in our cancer EX...



The confusion matrix

We can visualize model performance by using a **confusion matrix**

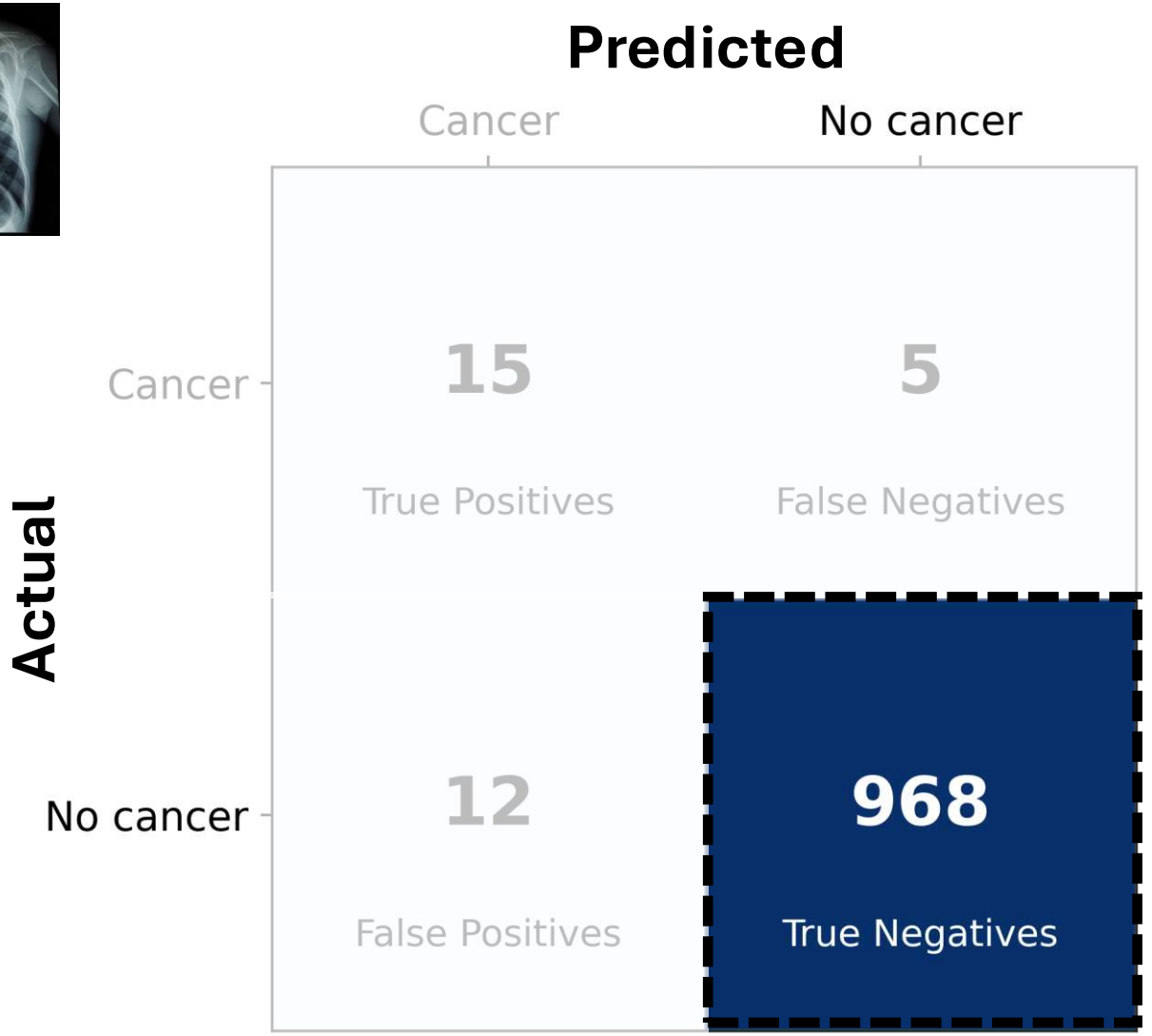


It contains 4 values:

True Negatives

- ✘ Model ***predicts no*** cancer
- ✘ Patient ***does not have*** cancer

Prediction and ground truth agree. This is good!



Metrics based on the confusion matrix

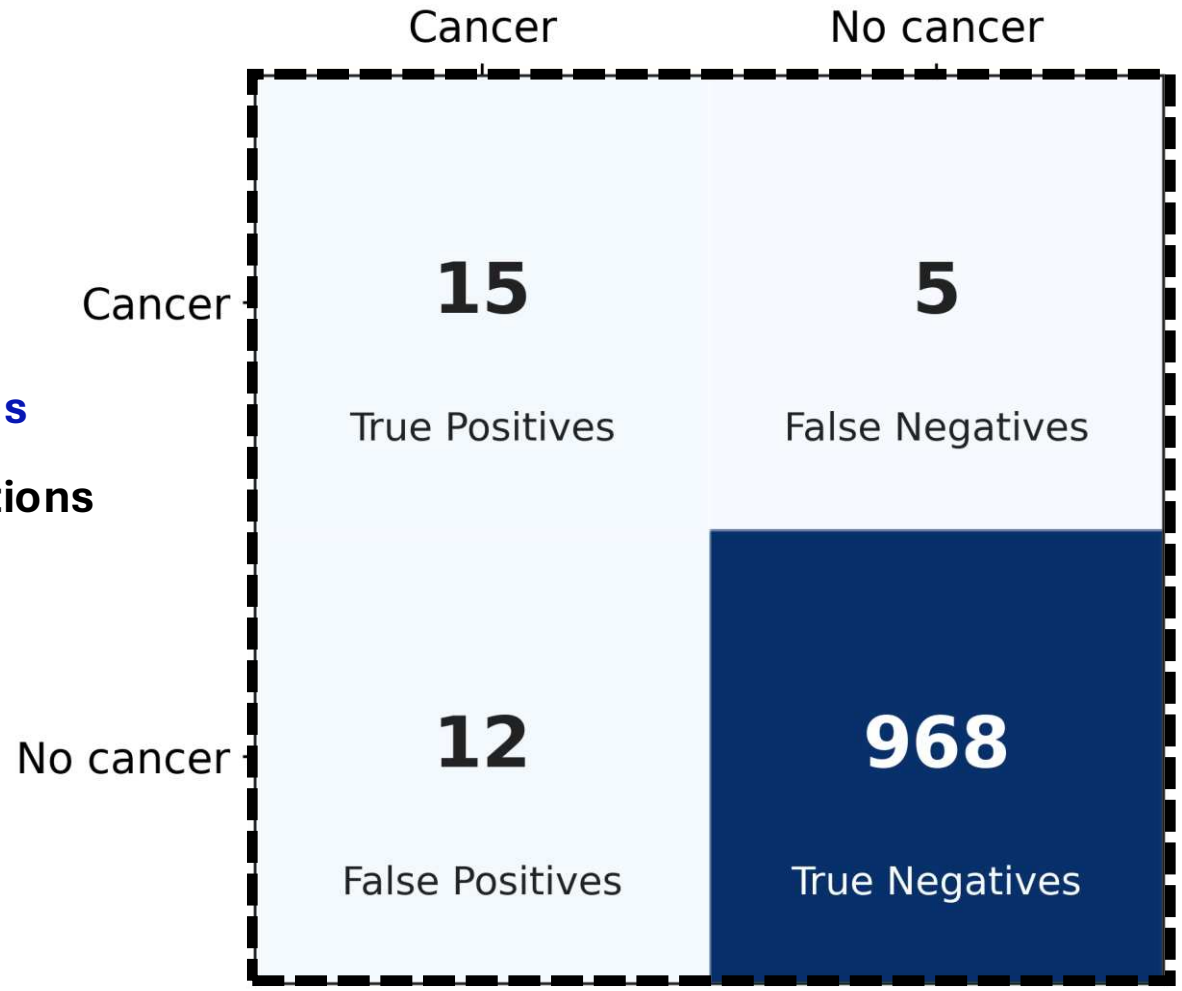
Reporting raw numbers from the confusion matrix can be a bit non-intuitive, since they depend on sample size…So let's compute some **metrics:**

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

Correct predictions

All predictions

This is okay if your dataset is balanced, but otherwise, **additional metrics** are needed – when in doubt, report all different types of metrics!



Metrics based on the confusion matrix

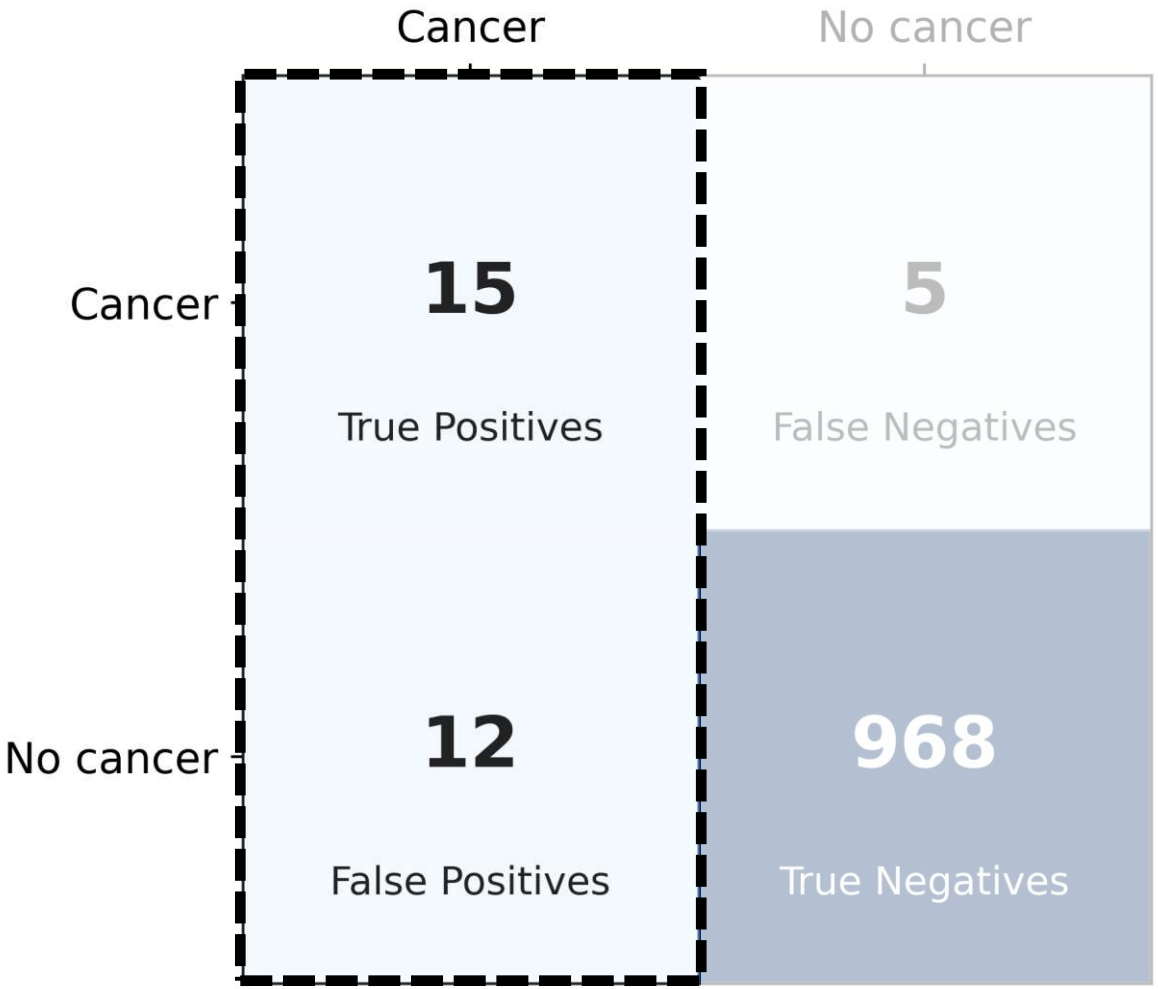
Reporting raw numbers from the confusion matrix can be a bit non-intuitive, since they depend on sample size…So let's compute some **metrics:**

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

Correctly predicted positives

All predicted positives

- Of everything we labeled positive, how much *really* is
- **Punishes false alarms (FP)**
- **A high precision means positives are trustworthy**



Metrics based on the confusion matrix

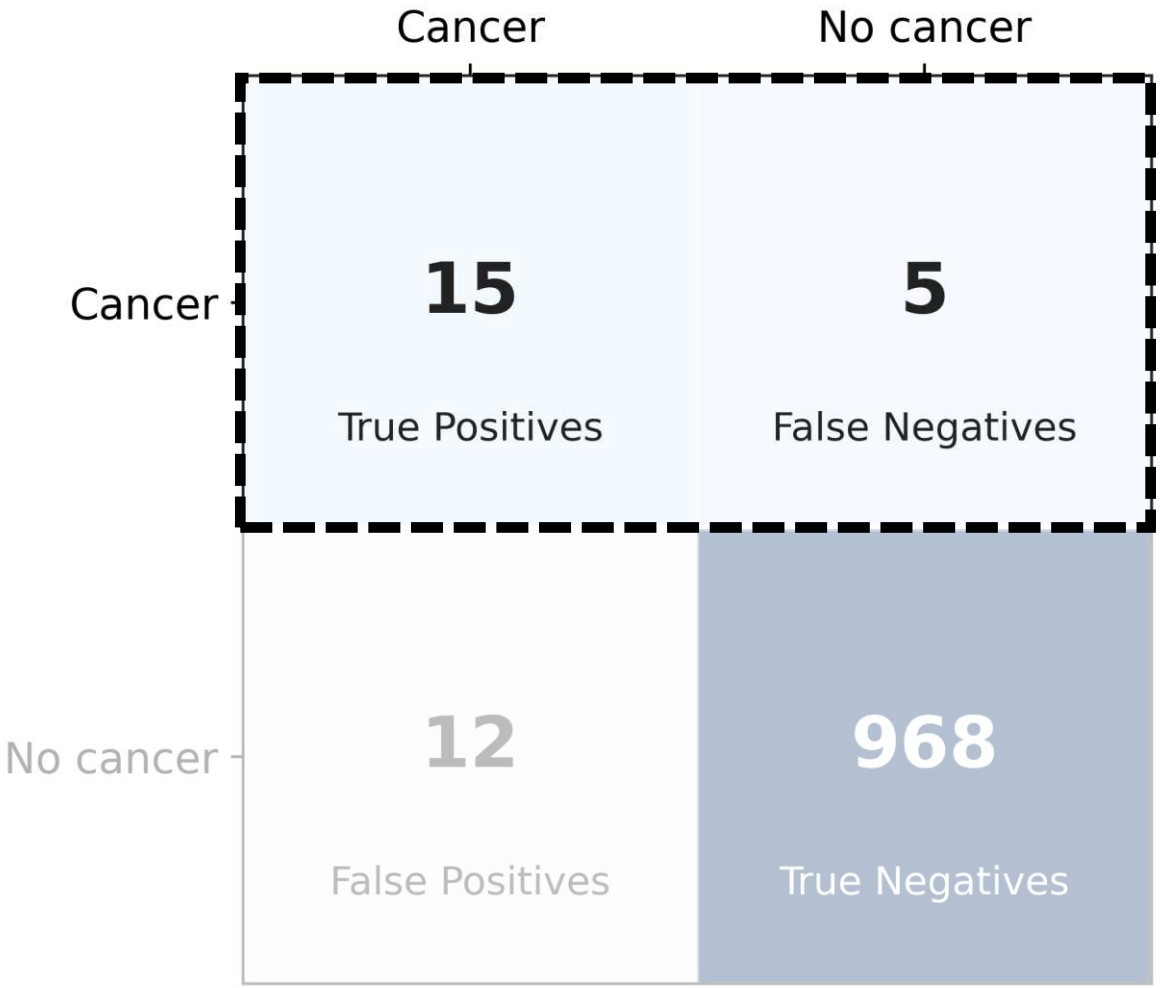
Reporting raw numbers from the confusion matrix can be a bit non-intuitive, since they depend on sample size…So let's compute some **metrics:**

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

Correctly predicted positives

All *actual* positives

- **Of all the *actual* positives, how many did we catch?**
- **Punishes false misses (FN)**
- **High recall → you can trust your model to catch actual positives**



Precision *vs.* Recall

Ideally, both precision and recall are as close to 1 as possible. But there is often a tradeoff…And **which one we prioritize depends entirely on the application**

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

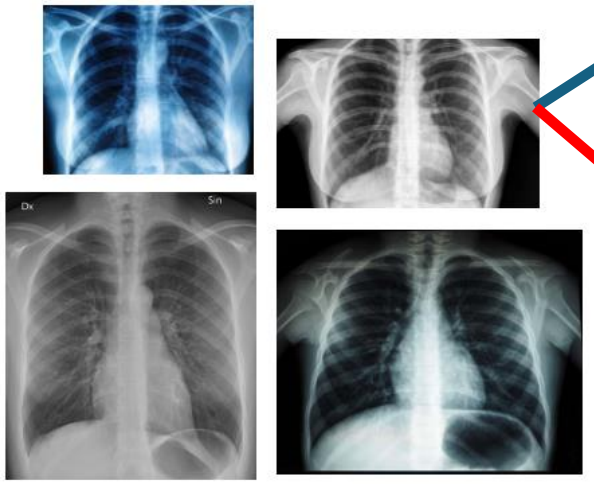
- Of everything we labeled positive, how much *really* is
- **Punishes false alarms (FP)**
- **A high precision means positives are trustworthy**

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

- Of all the *actual* positives, how many did we catch?
- **Punishes false misses (FN)**
- **High recall → you can trust your model to catch positives**

Precision *vs.* Recall

Ideally, both precision and recall are as close to 1 as possible. But there is often a tradeoff…And **which one we prioritize depends entirely on the application**



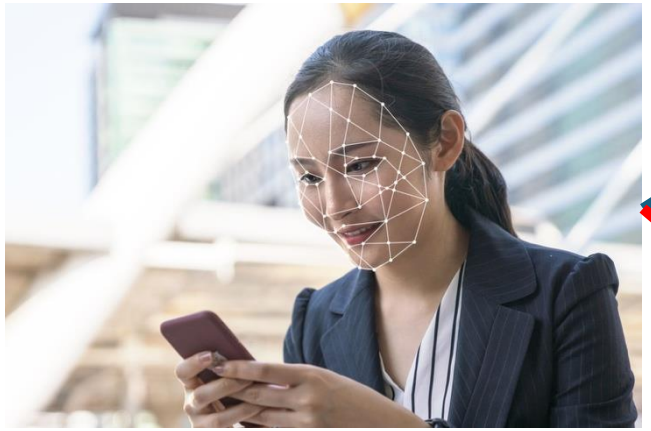
Has cancer

No cancer

Example #1:
Cancer screening

We really do not want any false negatives (missed cancer cases) so a **high recall** is most important

Example #2: Facial recognition



Is known user

Is not known user

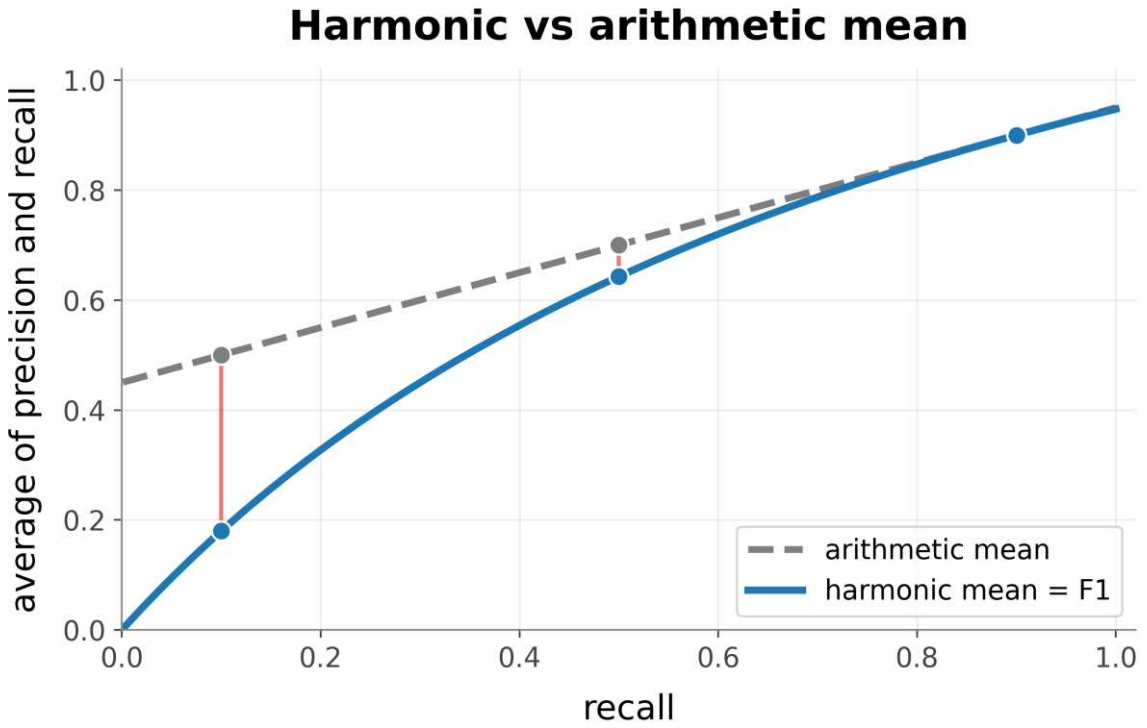
We don't want false positives (letting a stranger into your phone) so a **high precision** matters more

The F_1 score: balance precision and recall

If we just want a **reasonable balance** of precision and recall → **F_1 score:**

$$F_1 = 2\frac{(\text{Precision}) \cdot (\text{Recall})}{(\text{Precision}) + (\text{Recall})} = \frac{2 \cdot \text{TP}}{(2 \cdot \text{TP}) + \text{FP} + \text{FN}}$$

- This is one of the most commonly reported classification metrics.
- It is the **harmonic mean** of the precision and recall, which is high only when *both* metrics are high. It's a special kind of average that is **sensitive to small values.**



A worked example of classification metrics

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} = \frac{15 + 968}{15 + 12 + 968 + 5} \approx 0.98$$

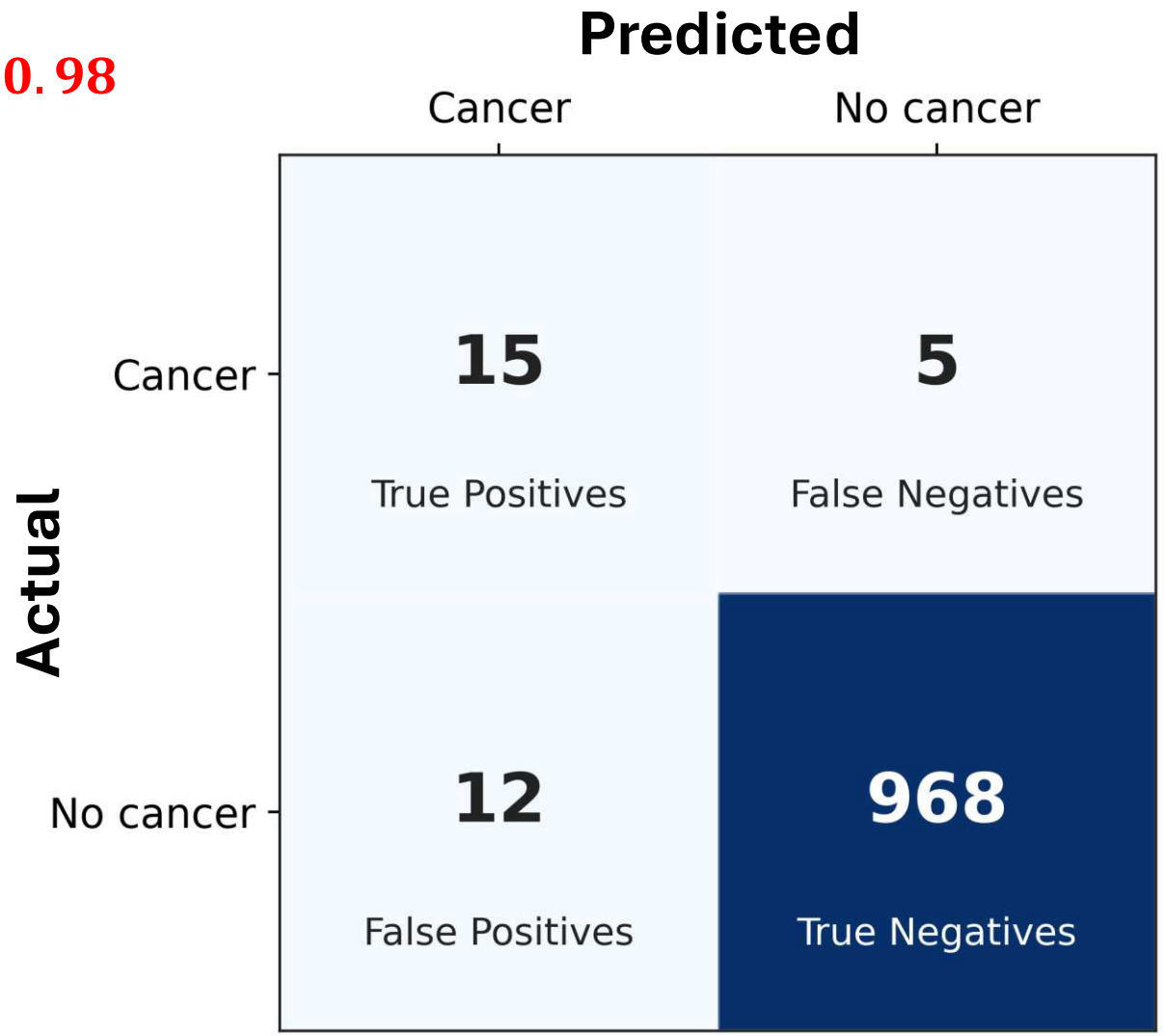
$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{15}{15 + 12} \approx 0.55$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{15}{15 + 5} \approx 0.75$$

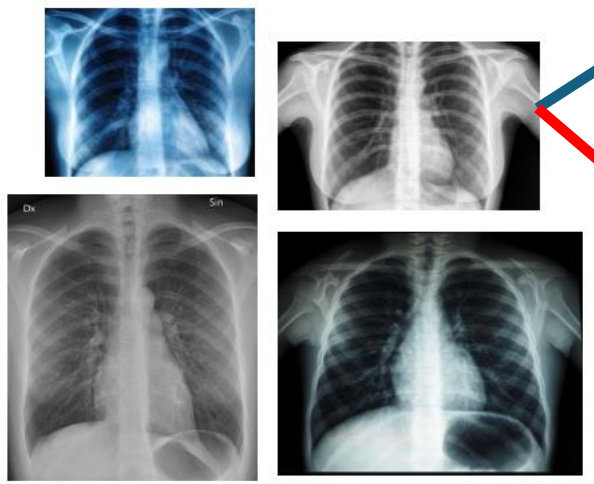
$$F_1 = 2\frac{(\text{Prec}) \cdot (\text{Recall})}{(\text{Prec}) + (\text{Recall})} = 2\frac{0.55 \cdot 0.75}{0.55 + 0.75}$$

$$F_1 \approx 0.63$$

You see this is much more informative than the accuracy!



Similar metrics apply to multiclass problems



Has cancer

No cancer

This is a **binary classification** problem. Our model has only two options (yes/no).

This is a **multiclass** problem.

Our model has more than options it may choose.



Cat

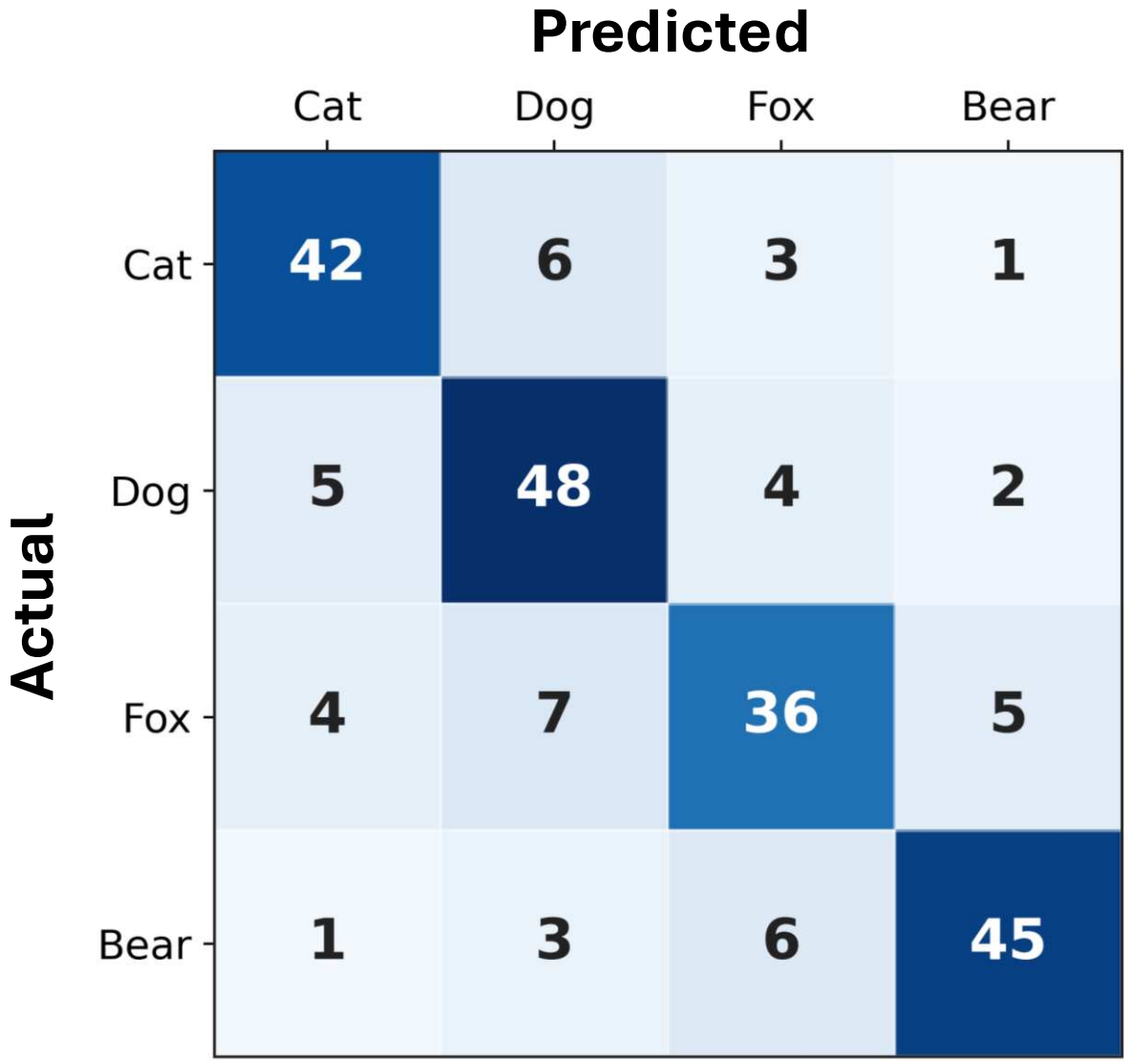
Dog

Fox

Bear

Similar metrics apply to multiclass problems

- Notice the **correct classifications** (where predicted = actual) lie along **the diagonal**
- From this matrix, we may compute precision, recall, and F_1 score on a per-class basis:
  - *e.g.*, treat "dog" as the positive class and everything else as the negative class **(dog | *not* dog)**
- We can also report one set of metrics averaged over all classes (next slide)



Similar metrics apply to multiclass problems

Class	n	Precision	Recall	F1 score
Cat	52	0.81	0.81	0.81
Dog	59	0.75	0.81	0.78
Fox	52	0.73	0.69	0.71
Bear	55	0.85	0.82	0.83

The **macro F_1 score*** is simply the (unweighted) average of the rightmost column:

$$F_{1,\text{macro}} = \frac{F_1^1 + F_1^2 + \cdots + F_1^K}{K} = \frac{0.81 + 0.78 + 0.71 + 0.83}{4} \approx 0.781$$

Use this score if every class is equally important (all get the same weight)

*There is also micro F_1 which we won't discuss…But it's calculated directly from TP, FP, and FN across samples/classes.

Similar metrics apply to multiclass problems

Class	n	Precision	Recall	F1 score
Cat	52	0.81	0.81	0.81
Dog	59	0.75	0.81	0.78
Fox	52	0.73	0.69	0.71
Bear	55	0.85	0.82	0.83

The **weighted F_1 score** is simply average score weighted by the number of samples:

$$F_{1,\text{weighted}} = \frac{n_1 F_1^1 + n_2 F_1^2 + \cdots + n_K F_1^K}{n_1 + n_2 + \cdots + n_K}$$

$$= \frac{52(0.81) + 59(0.78) + 52(0.71) + 55(0.83)}{52 + 59 + 52 + 55} \approx 0.783$$

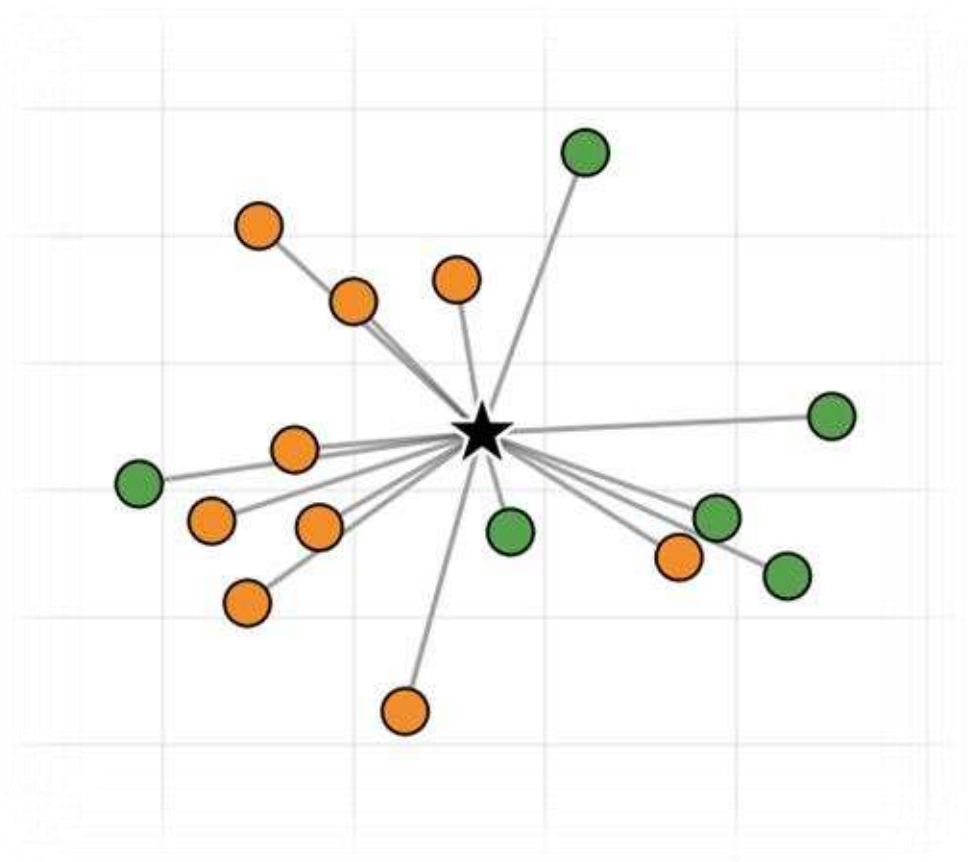
Use this score if your care about overall performance on your dataset, and your dataset is imbalanced (notice in this dataset, it doesn't differ much)

Questions?

It is important to understand classification metrics!
...They will show up time and time again...

Tuning the Precision *vs.* Recall Tradeoff

Each class has an associated probability



From KNN: 9 orange *vs.* 6 green neighbors

- 9/15 (60%) chance the query is green
- 6/15 (**40%**) chance the query is orange

Suppose instead that orange represents **positive cases that have cancer**

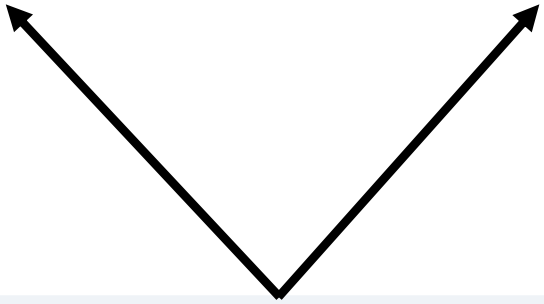
Whereas **green = negative (no cancer)**

It may be useful to define a **threshold** above which the query is labeled positive. For binary cases, this would default to 50%, but we can vary it!

The threshold dictates precision *vs.* recall

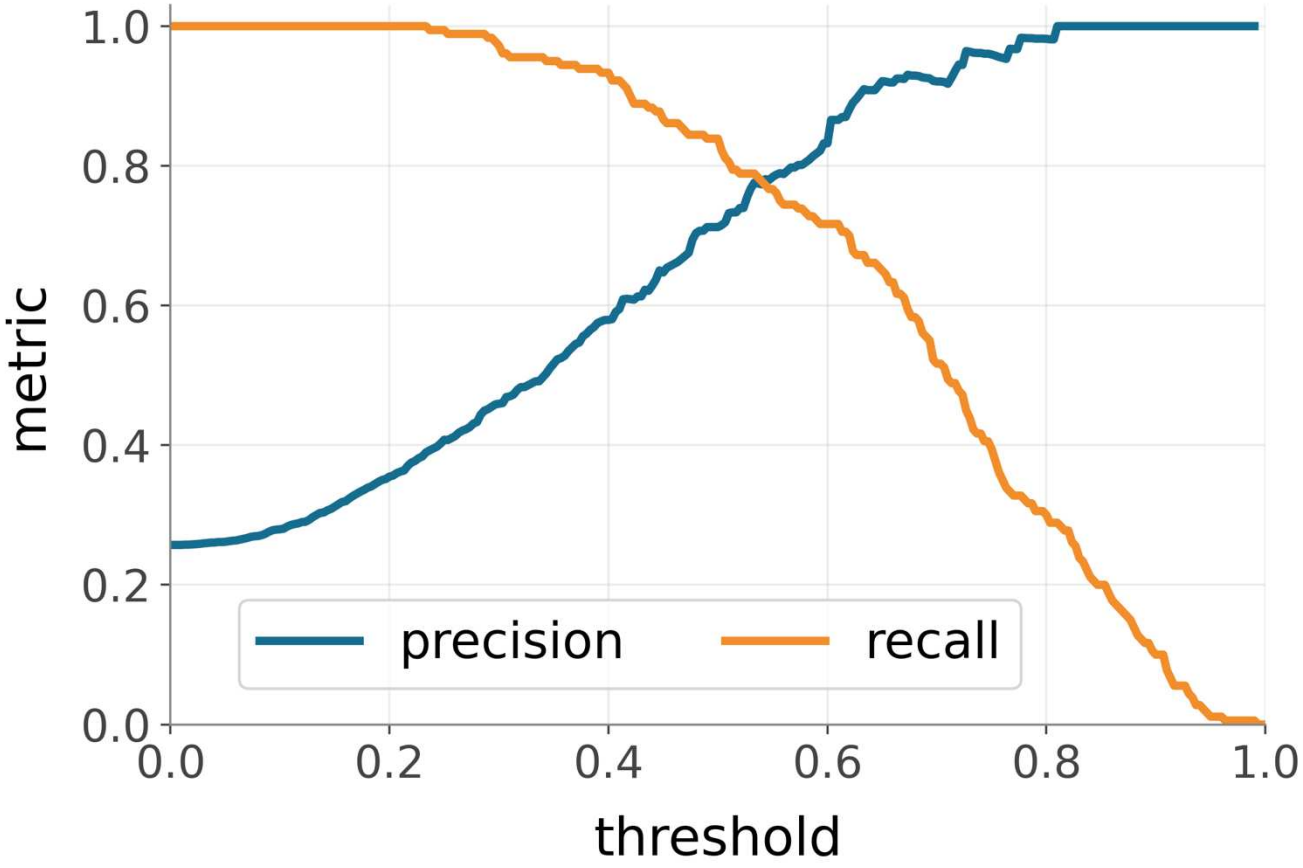
A **lower threshold** will lead to more false positives and less false negatives → Lower precision → Higher recall

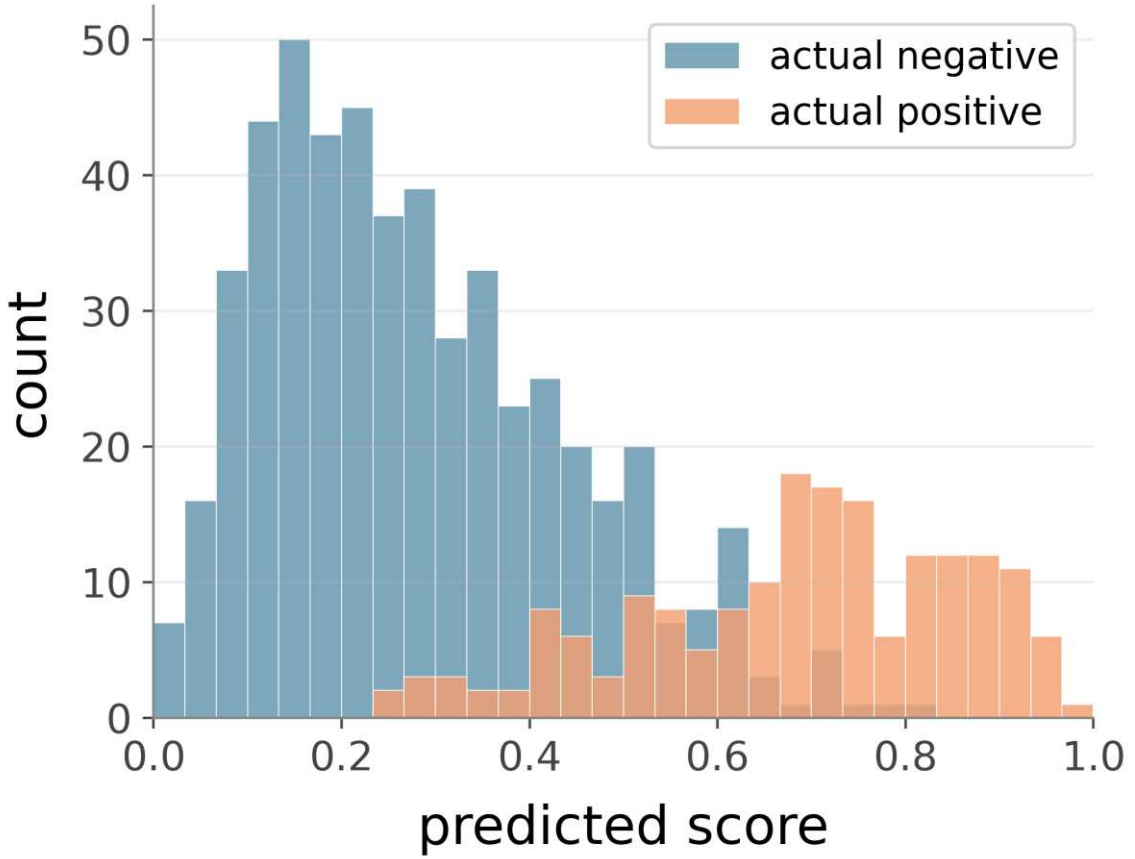
A **higher threshold** will lead to fewer false positives and more false negatives → Higher precision → Lower recall



It may be useful to define a **threshold** above which the query is labeled positive. For binary cases, this would default to 50%, but we can vary it!

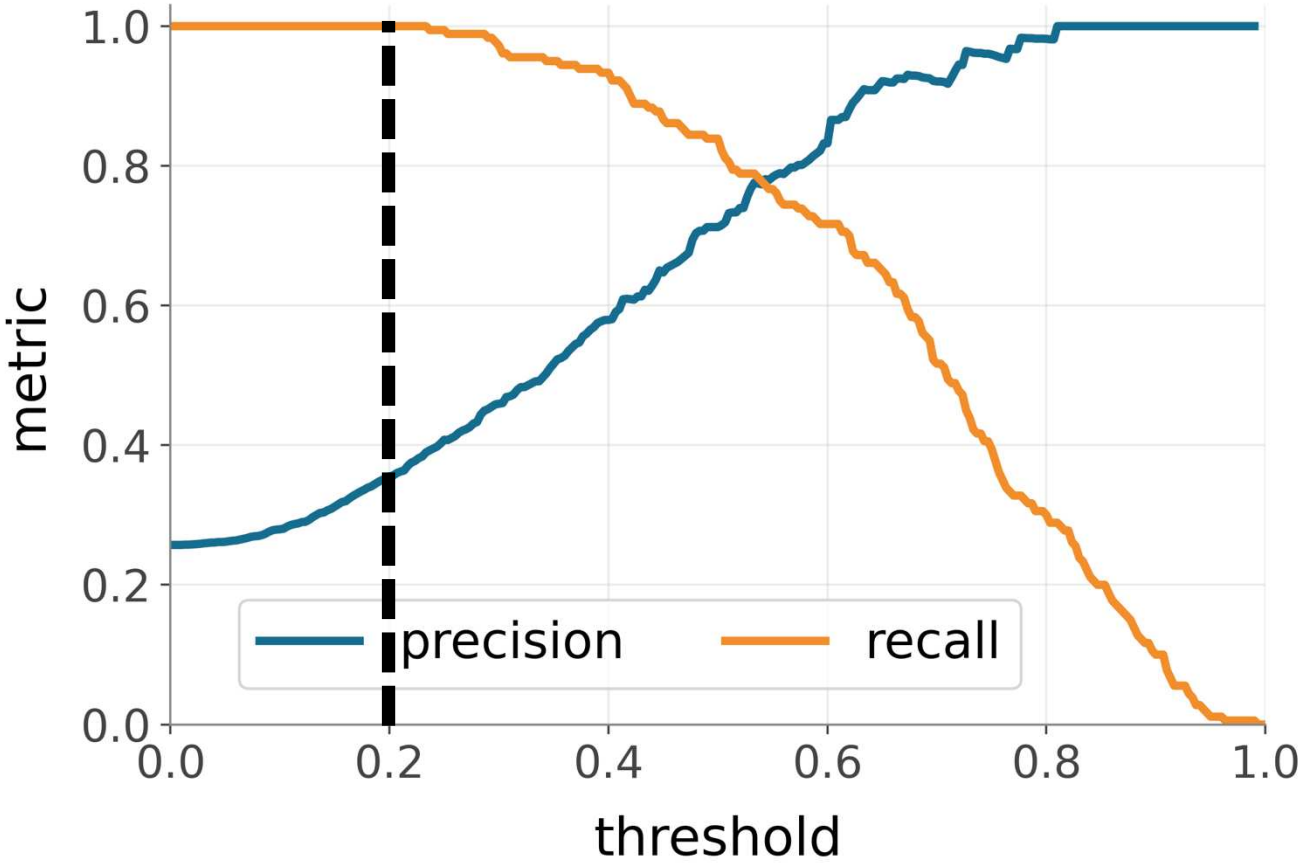
The threshold dictates precision *vs.* recall

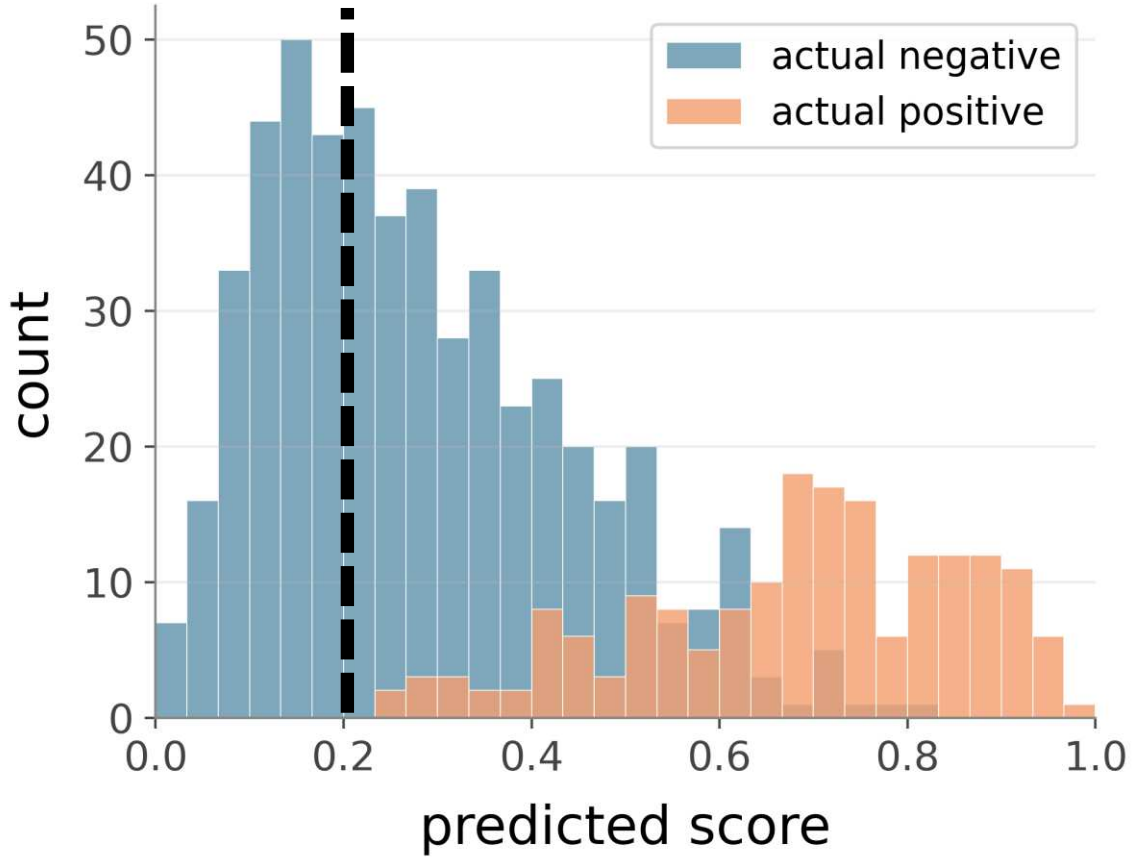




As we raise the classification threshold: ↑ precision but ↓ recall

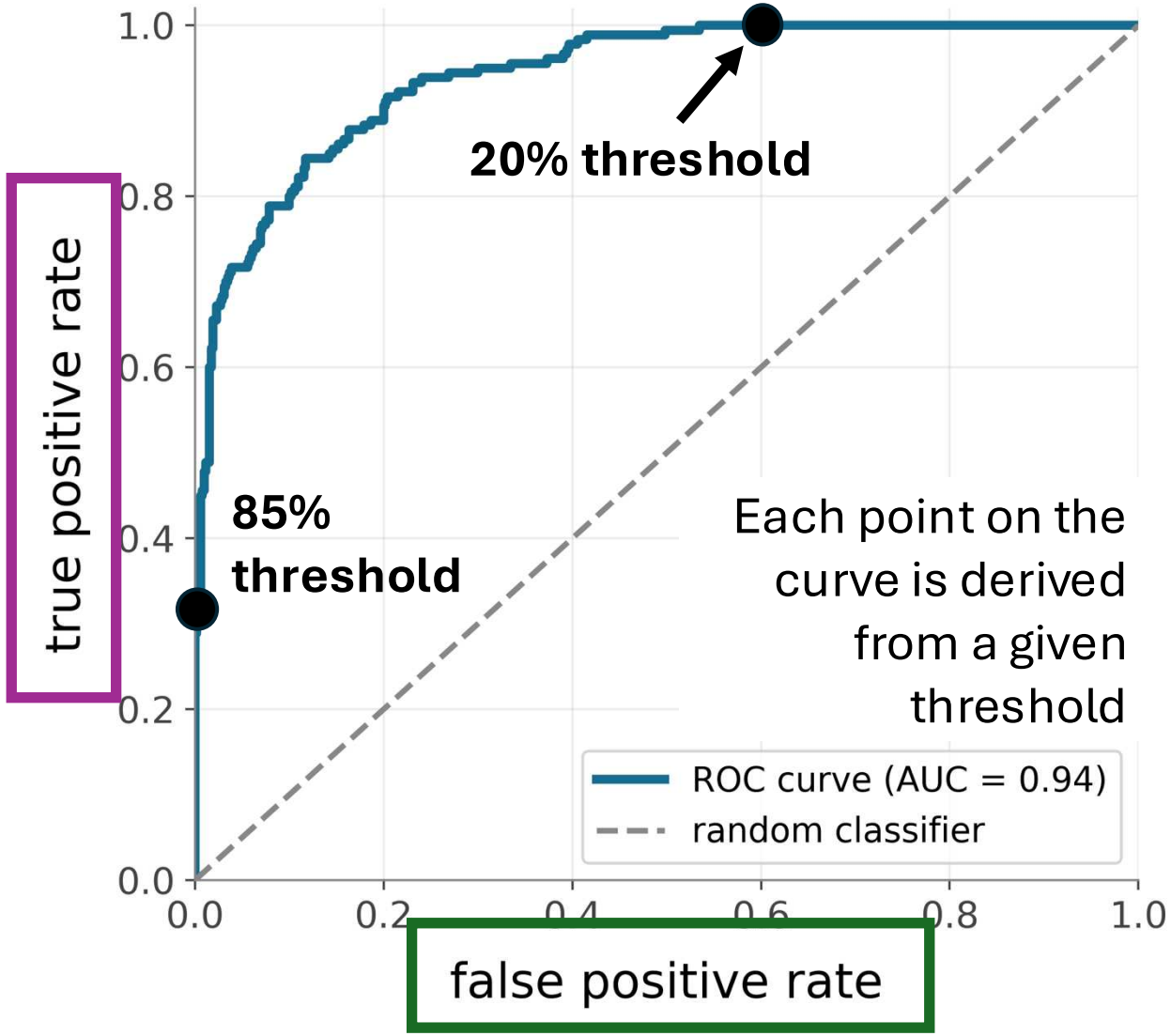
The threshold dictates precision *vs.* recall





We can “catch” all the actual positives by using a threshold of 20%, at the cost of low precision…Probably a good choice for cancer screening

Receiver Operating Characteristic (ROC) curve



The True Positive Rate:

$$\mathbf{TPR} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}}$$ (same as recall)

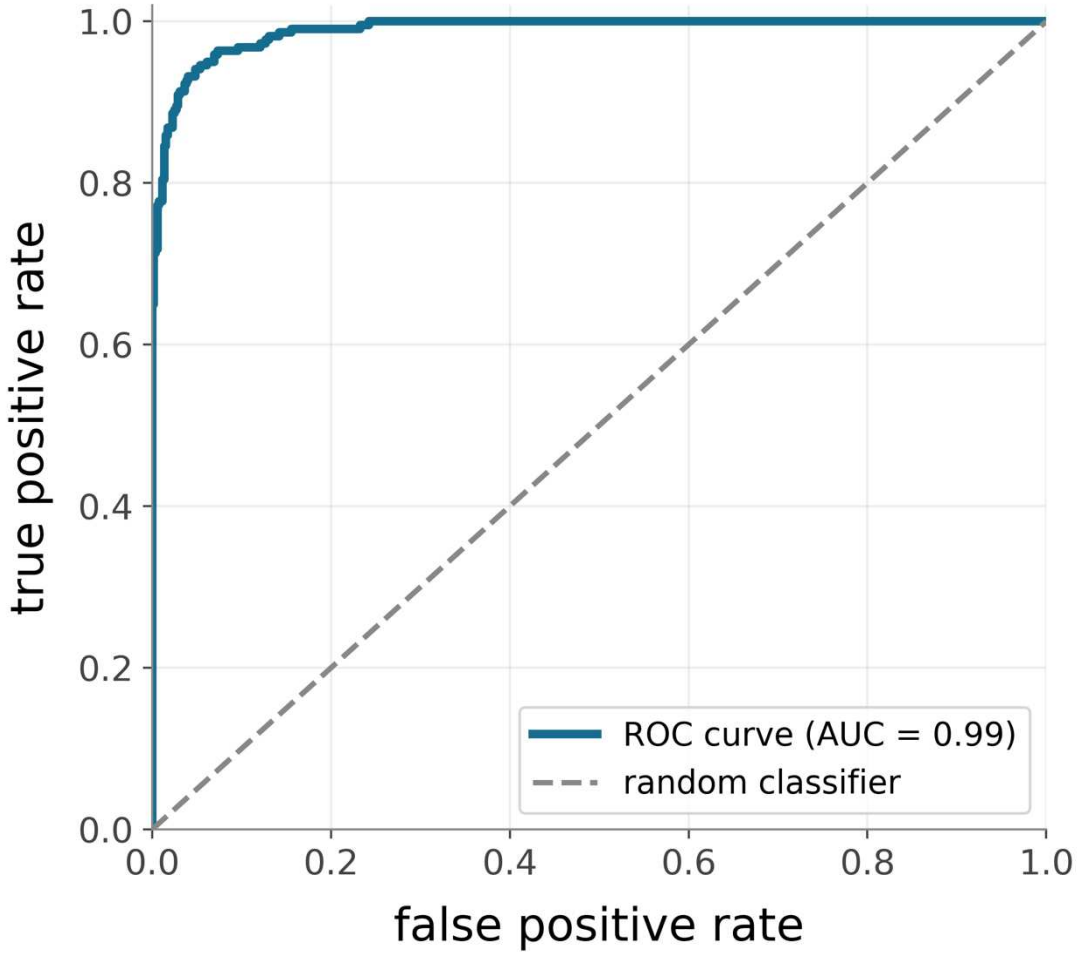
The False Positive Rate:

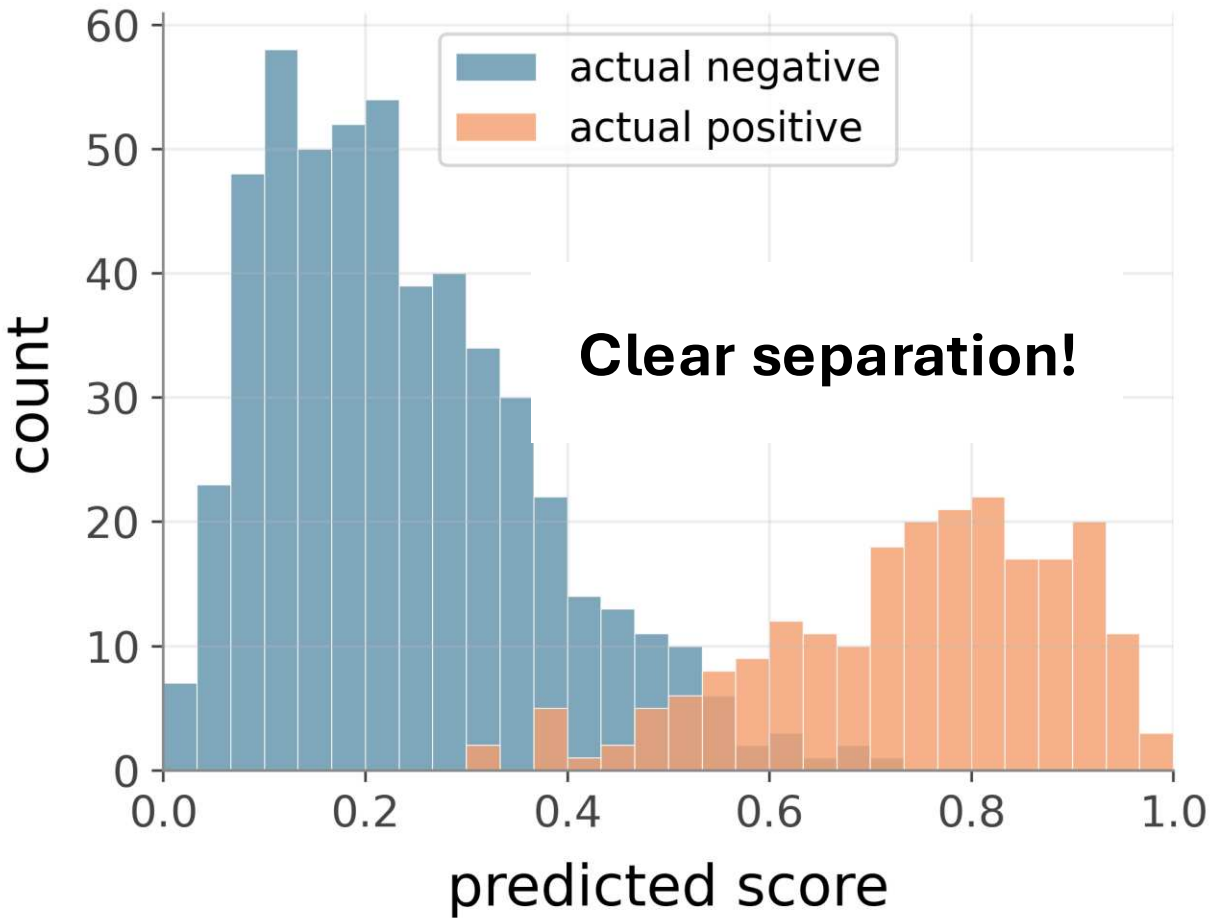
$$\mathbf{FPR} = \frac{\mathbf{FP}}{\mathbf{FP} + \mathbf{TN}}$$

Of all the negatives, how many did we incorrectly classify as positive?

We'd like to choose a threshold that makes both as close to 1 as possible (depending on our problem)

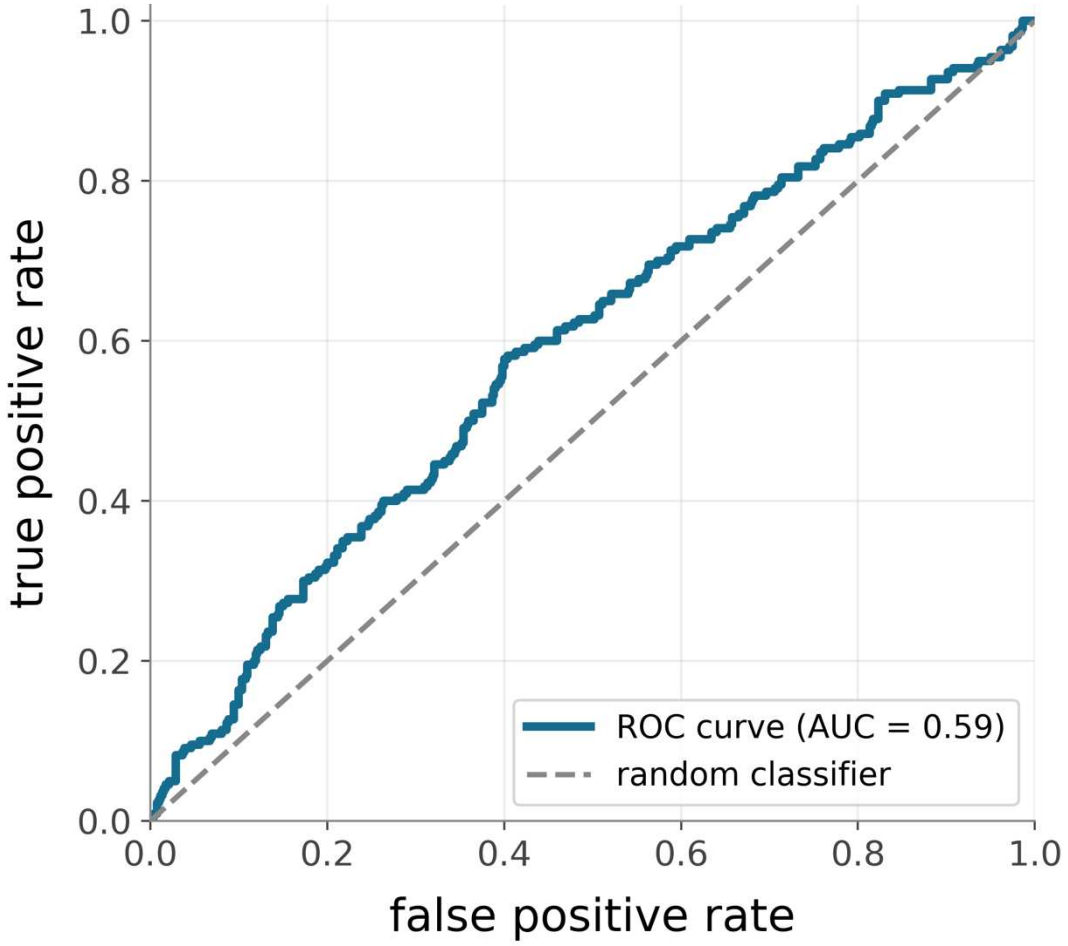
Good *vs.* bad classifiers based on the ROC

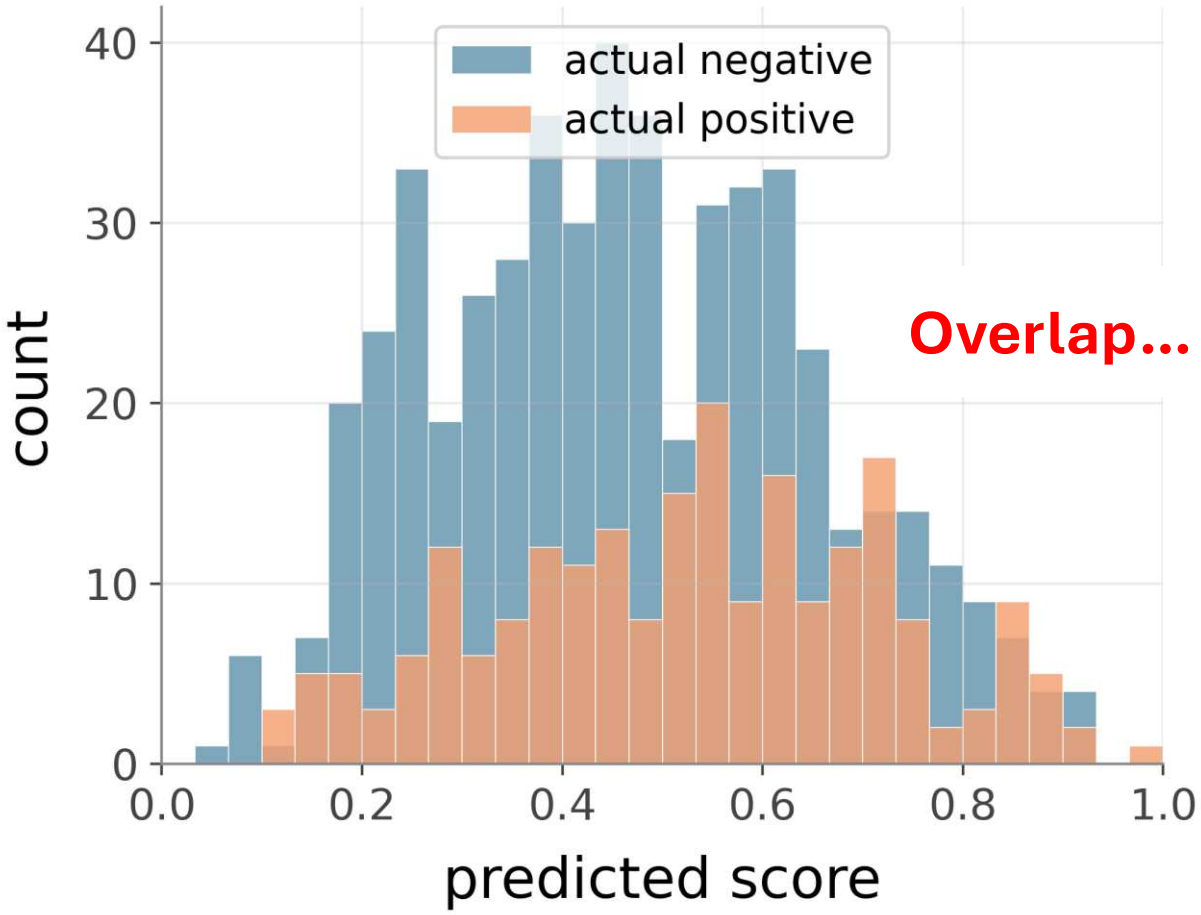




A *good* classifier has an ROC that rises very quickly as we decrease the threshold.
This means we can reach TPR ≈ 1 without raising FPR much.

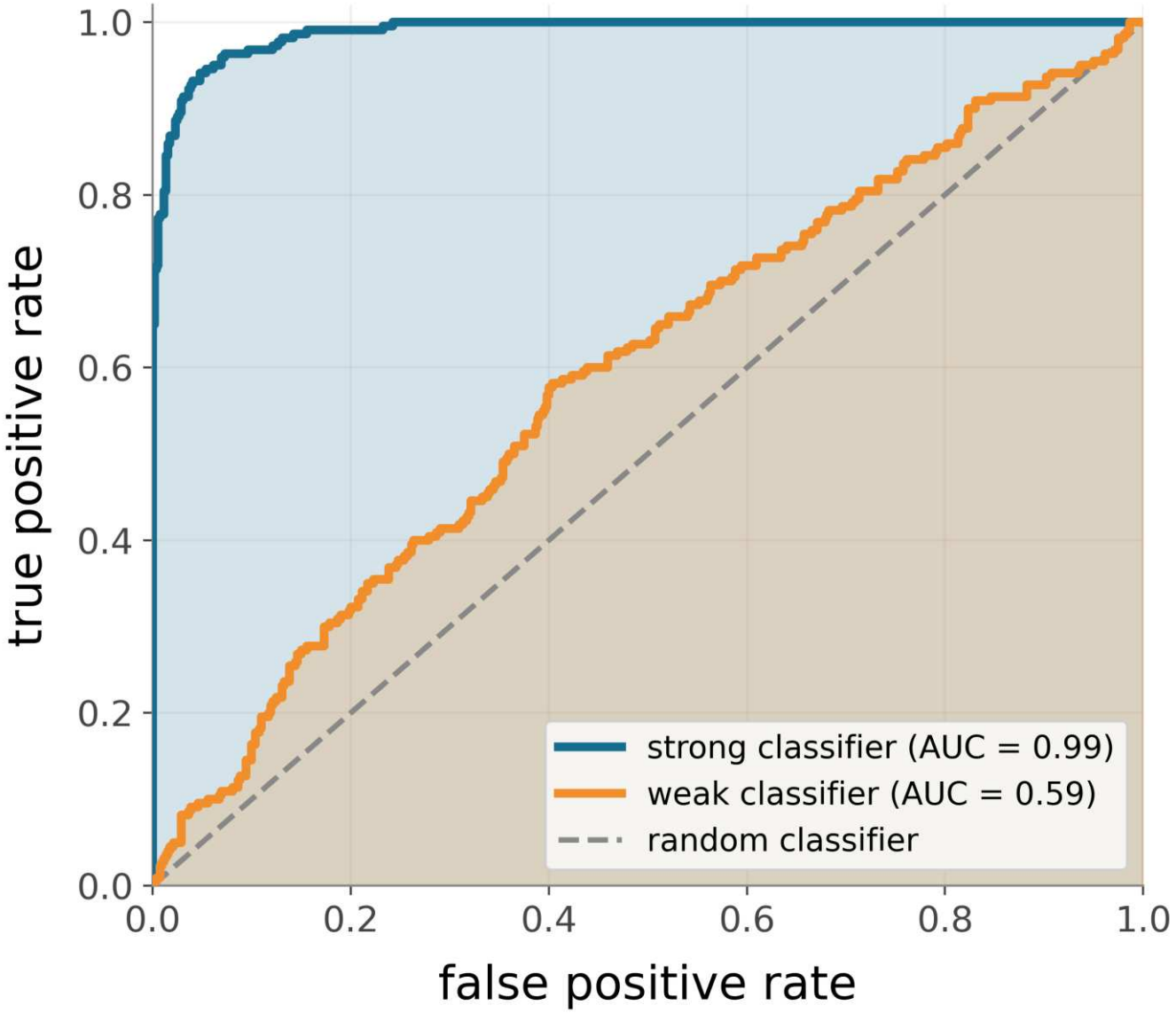
Good *vs.* bad classifiers based on the ROC





A *bad* classifier has an ROC that rises slowly as we decrease the threshold.
This means we have to raise FPR a lot to get decent TPR.

We can quantify this using AUC

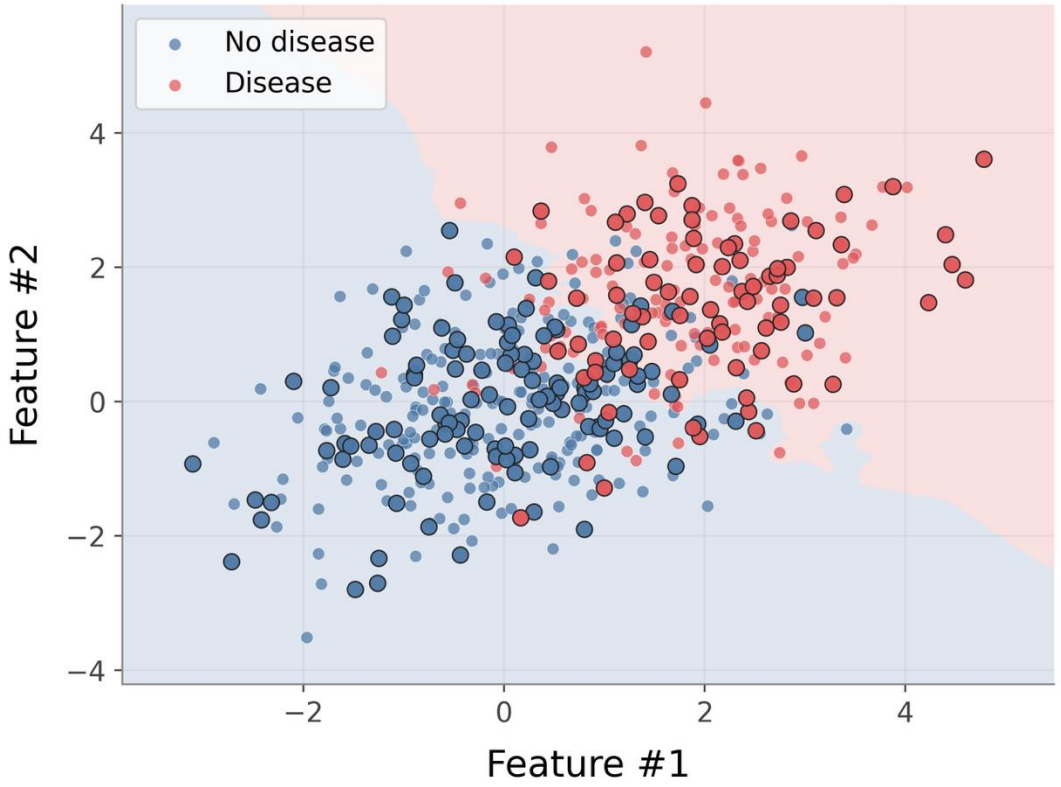


The **Area Under the ROC Curve (AUC)** is a single-number summary of a classifier's performance across all possible classification thresholds.

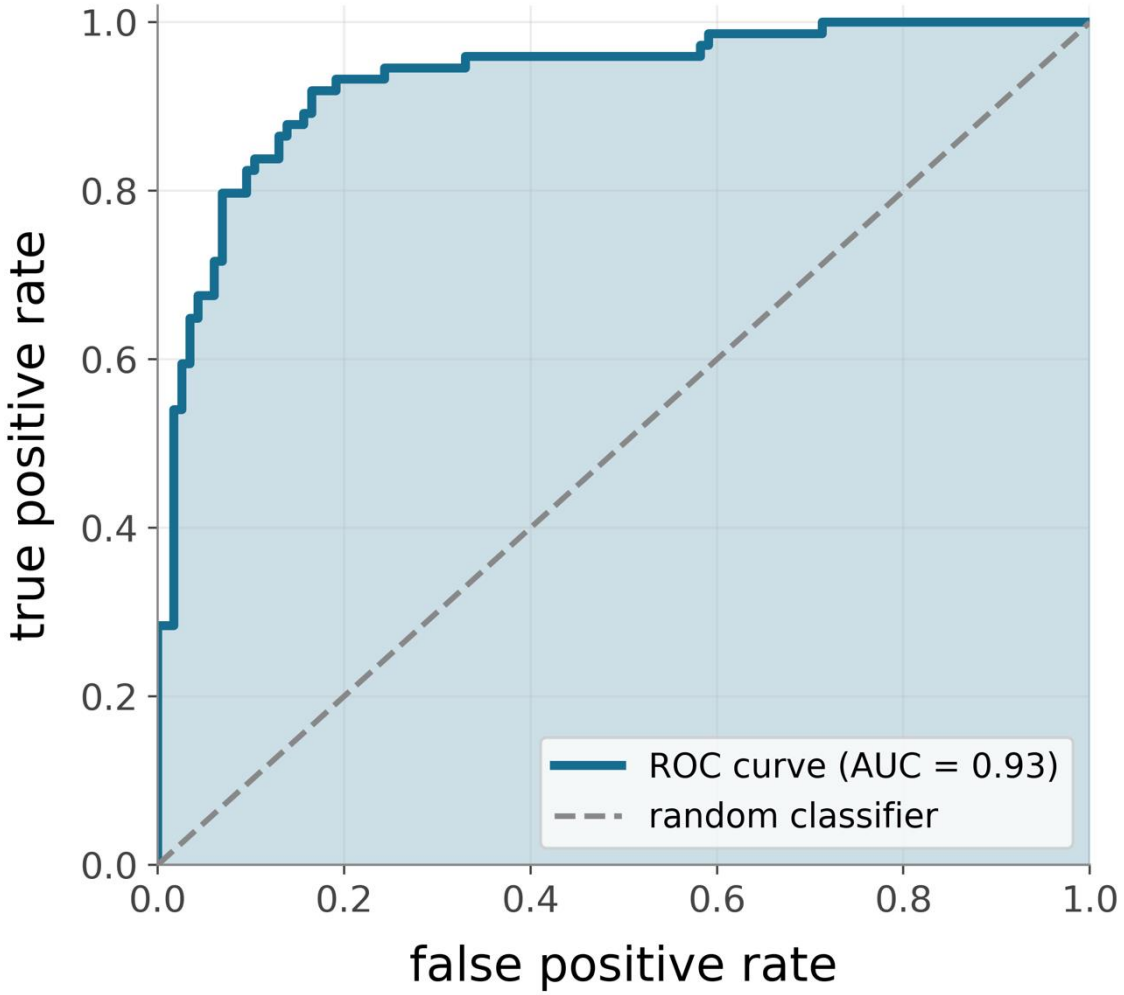
Larger AUC values (close to 1) indicate better discrimination between the positive and negative classes.

This is **threshold independent!** If you have bad accuracy but good AUC, it means you can choose a better threshold.

Complete evaluation of a KNN model

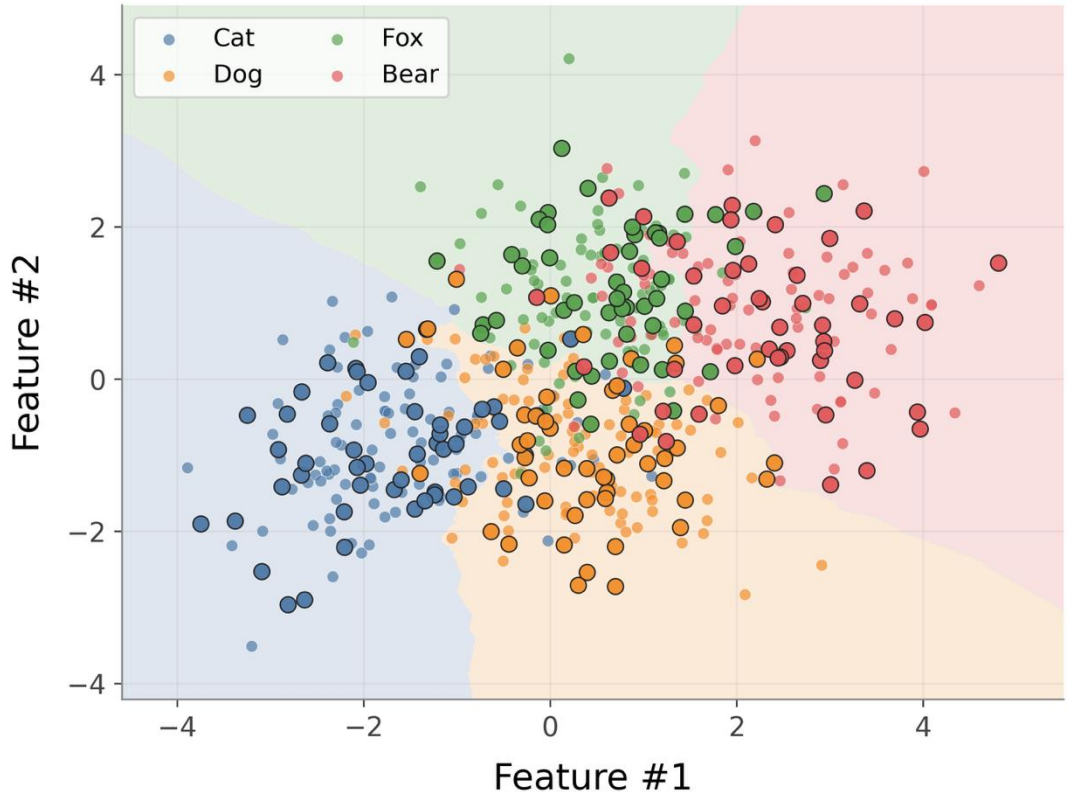


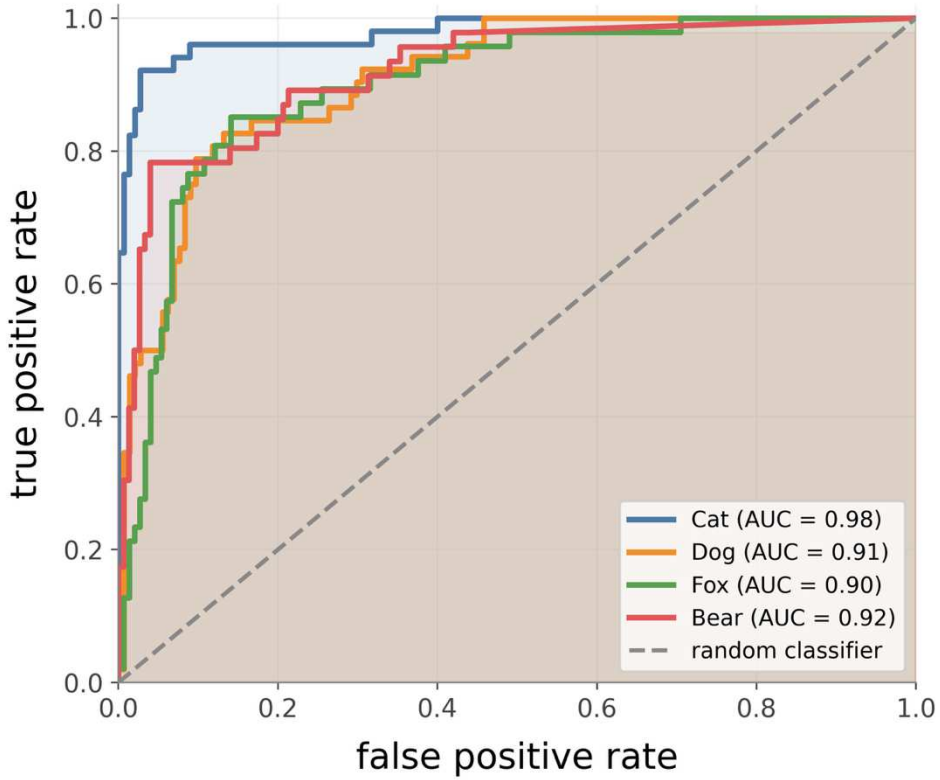
As always: report on a held-out (test) set



Metric	Value
Accuracy	0.88
Precision	0.88
Recall	0.80
F1 score	0.84

Similar results for a multiclass problem



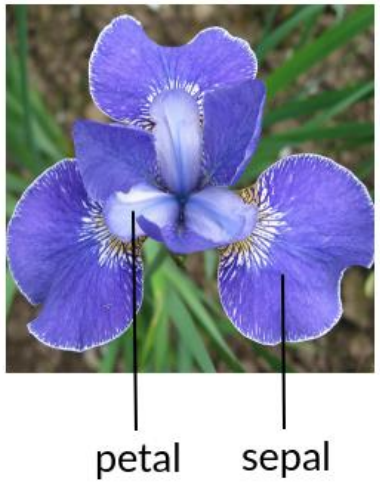


Class	n	Precision	Recall	F1 score
Cat	51	0.93	0.82	0.88
Dog	52	0.72	0.81	0.76
Fox	47	0.72	0.77	0.74
Bear	46	0.84	0.78	0.81

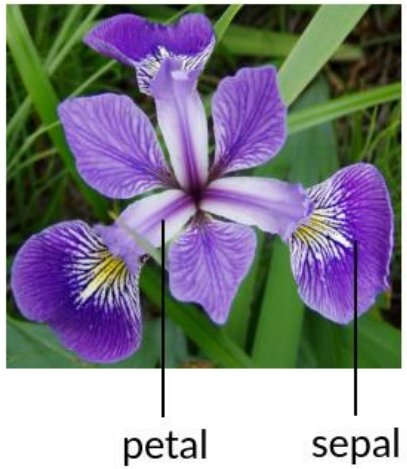
KNN classification on real data

EX #1: The Iris dataset

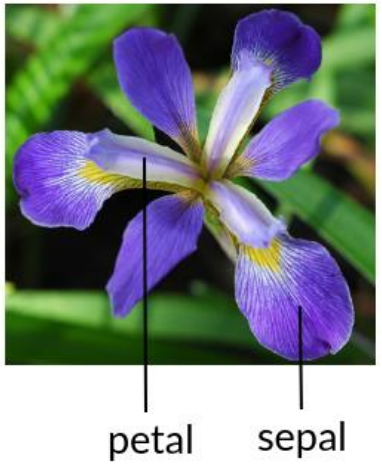
iris setosa



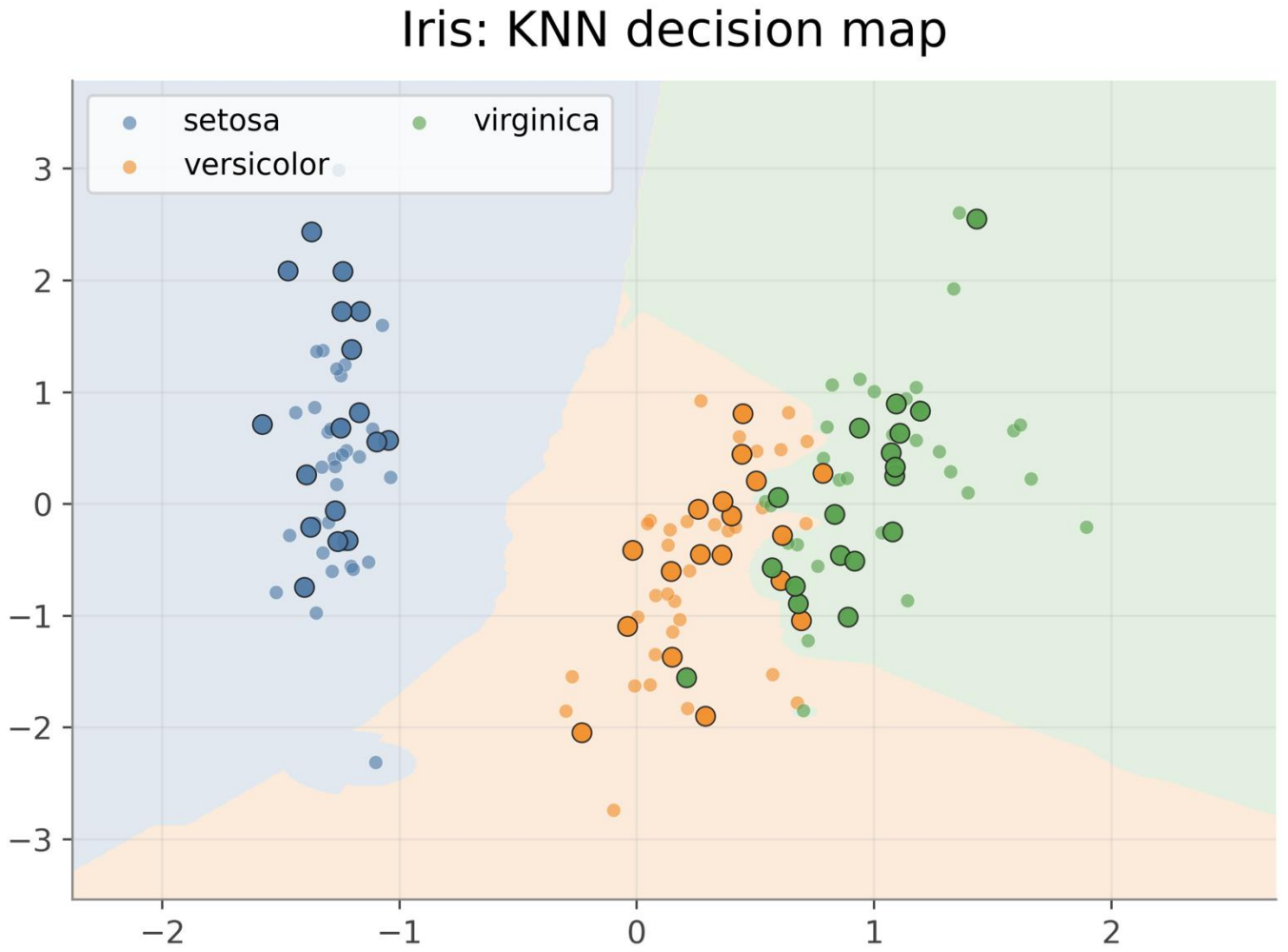
iris versicolor



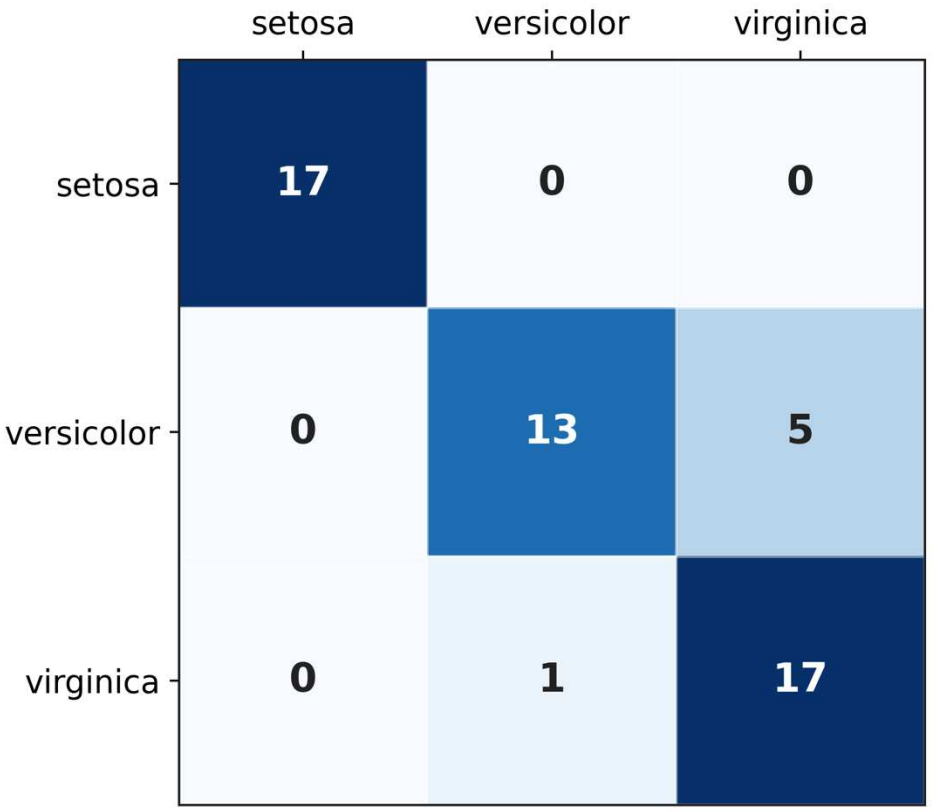
iris virginica

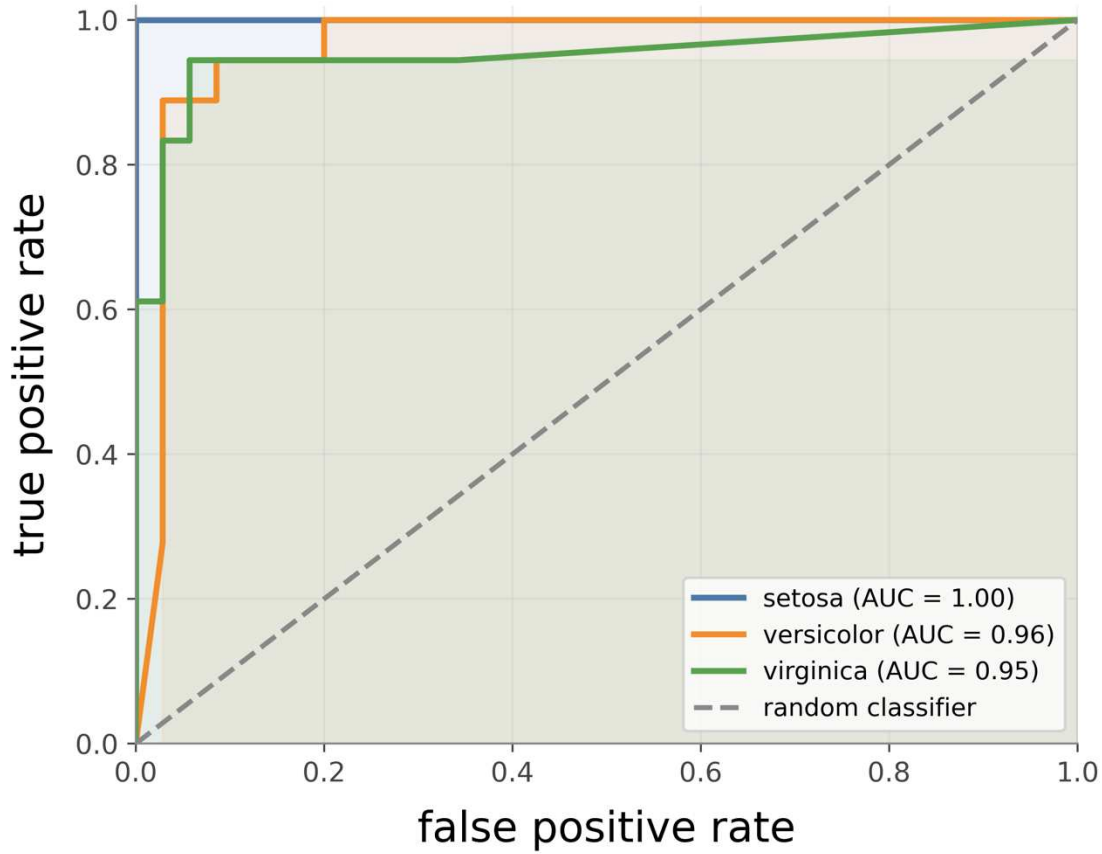


Run KNN on the petal features with $k = 9$



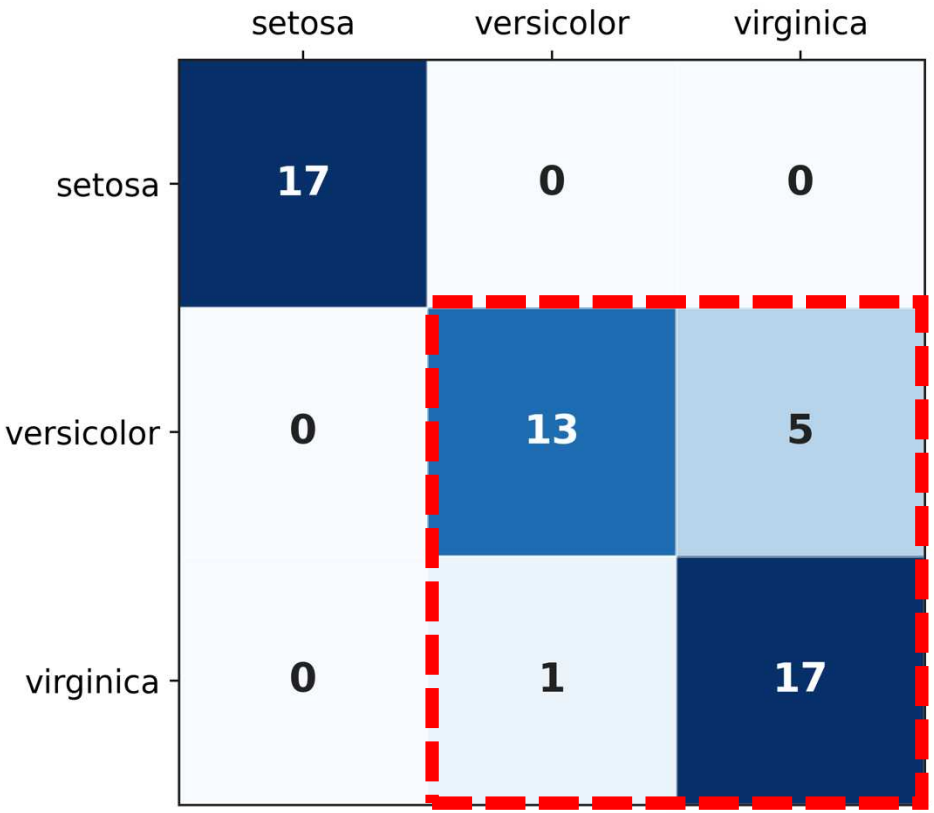
EX #1: The Iris dataset

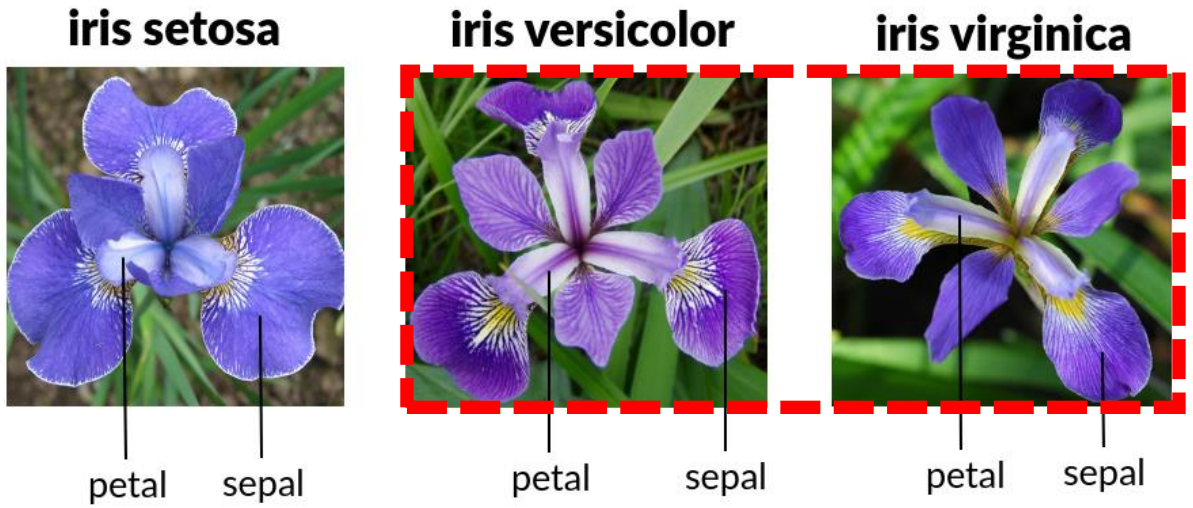




Class	n	Precision	Recall	F1 score
setosa	17	1.00	1.00	1.00
versicolor	18	0.93	0.72	0.81
virginica	18	0.77	0.94	0.85

EX #1: The Iris dataset

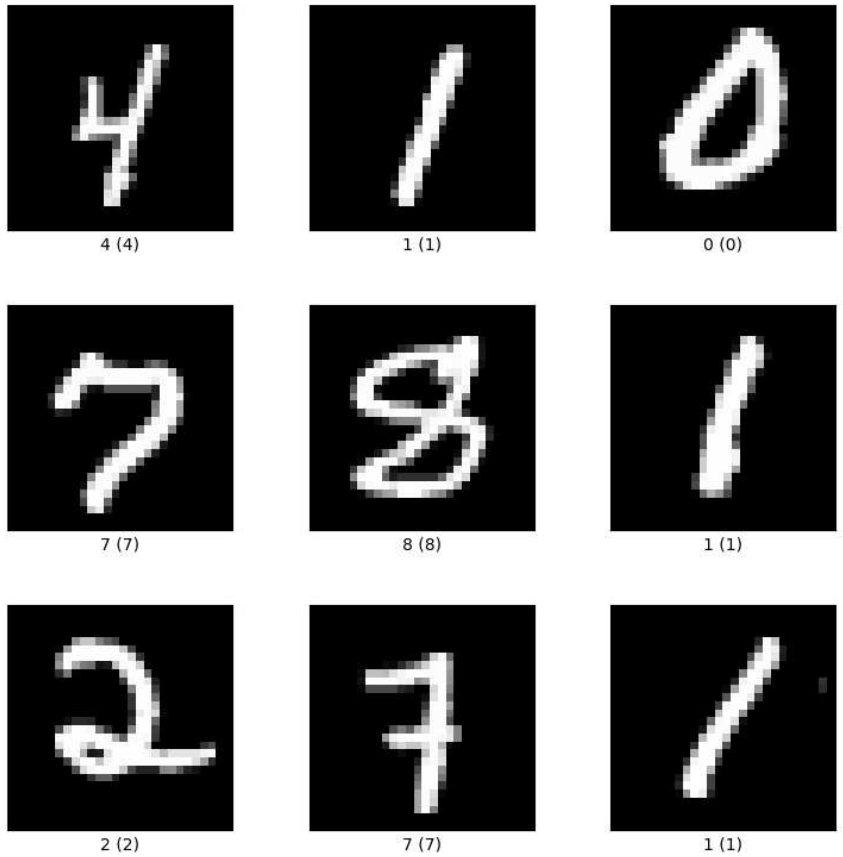




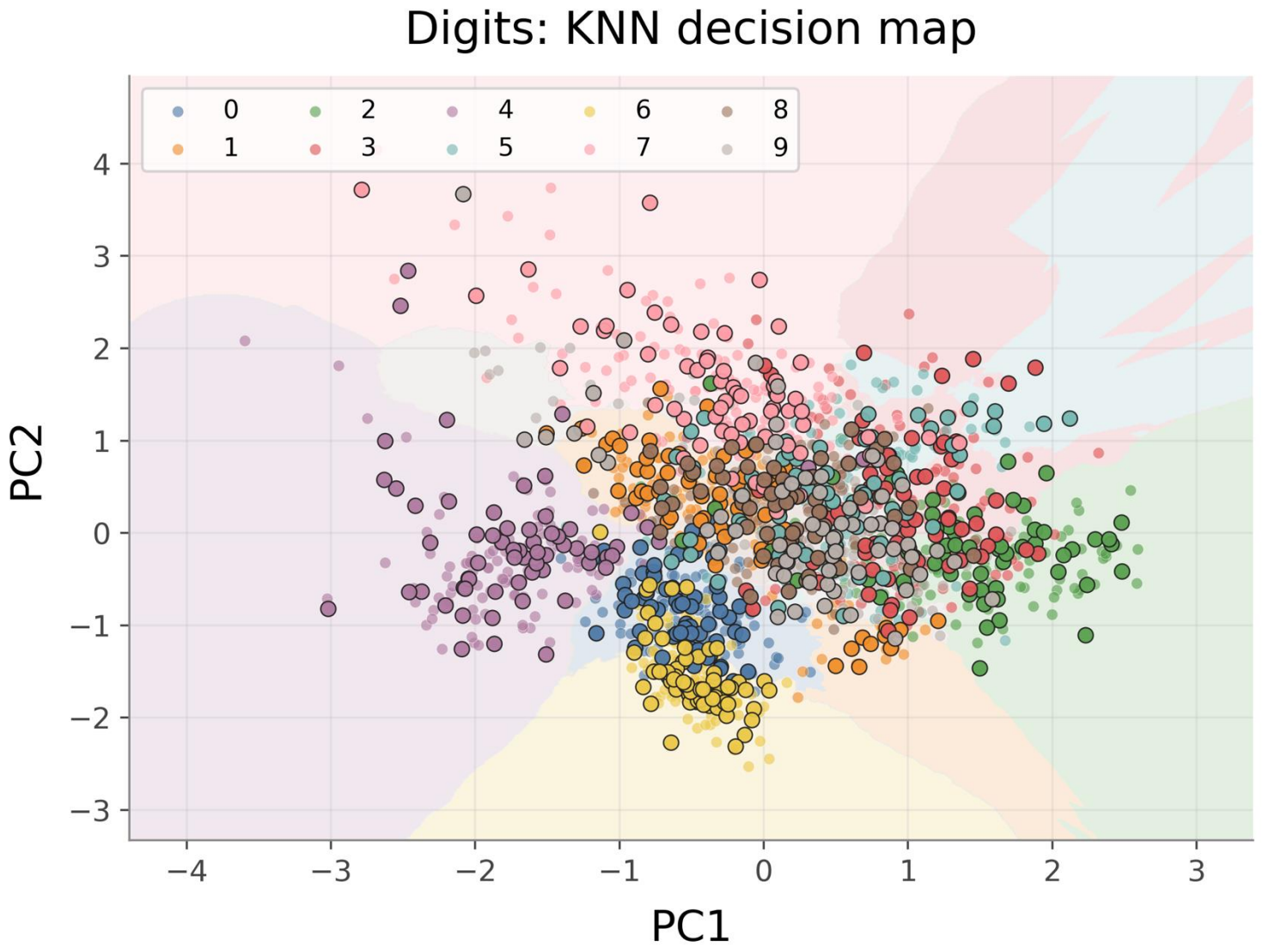
The confusion between these two is understandable!

Class	n	Precision	Recall	F1 score
setosa	17	1.00	1.00	1.00
versicolor	18	0.93	0.72	0.81
virginica	18	0.77	0.94	0.85

EX #2: The MNIST dataset

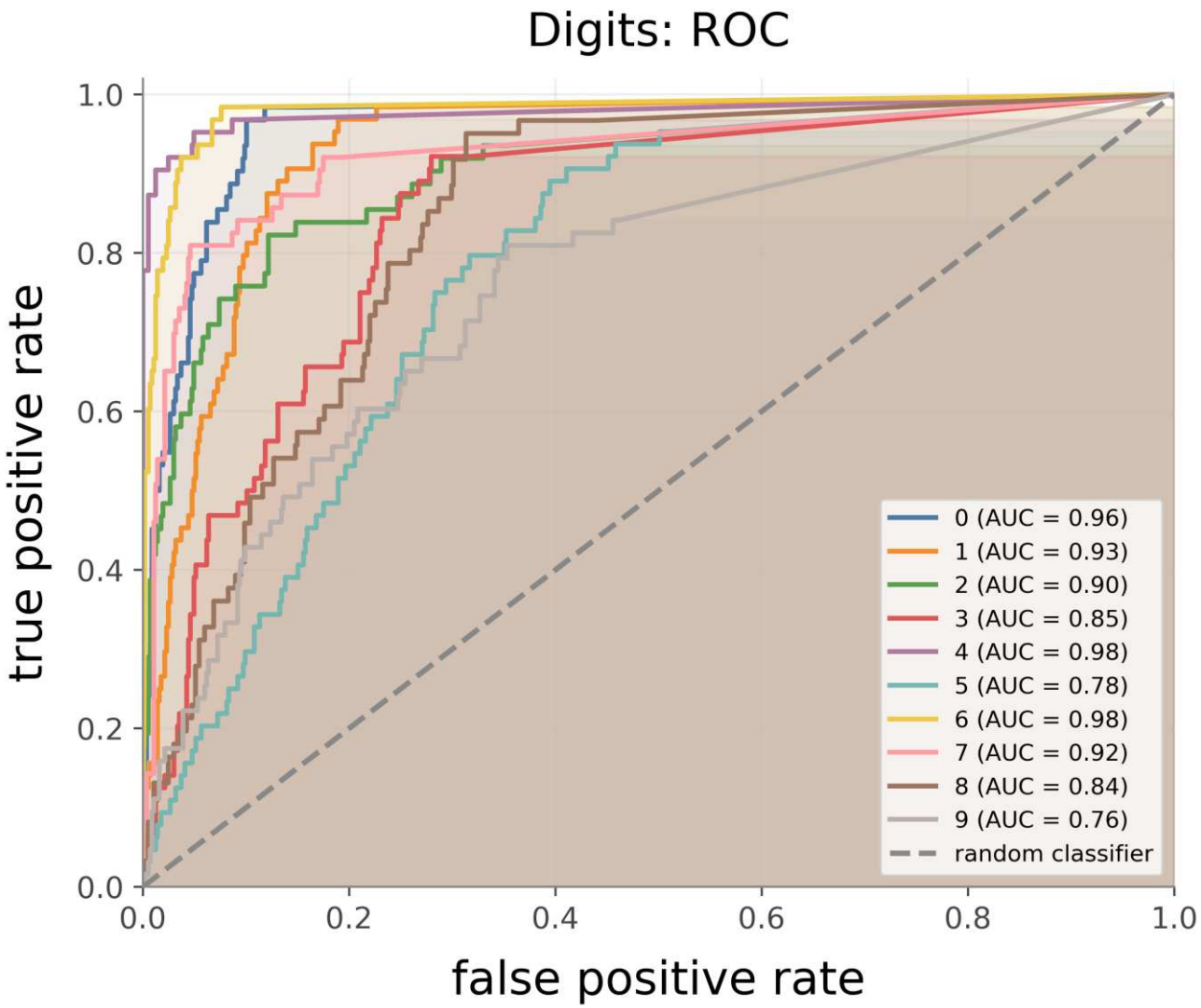


PCA followed by KNN with $k = 15$



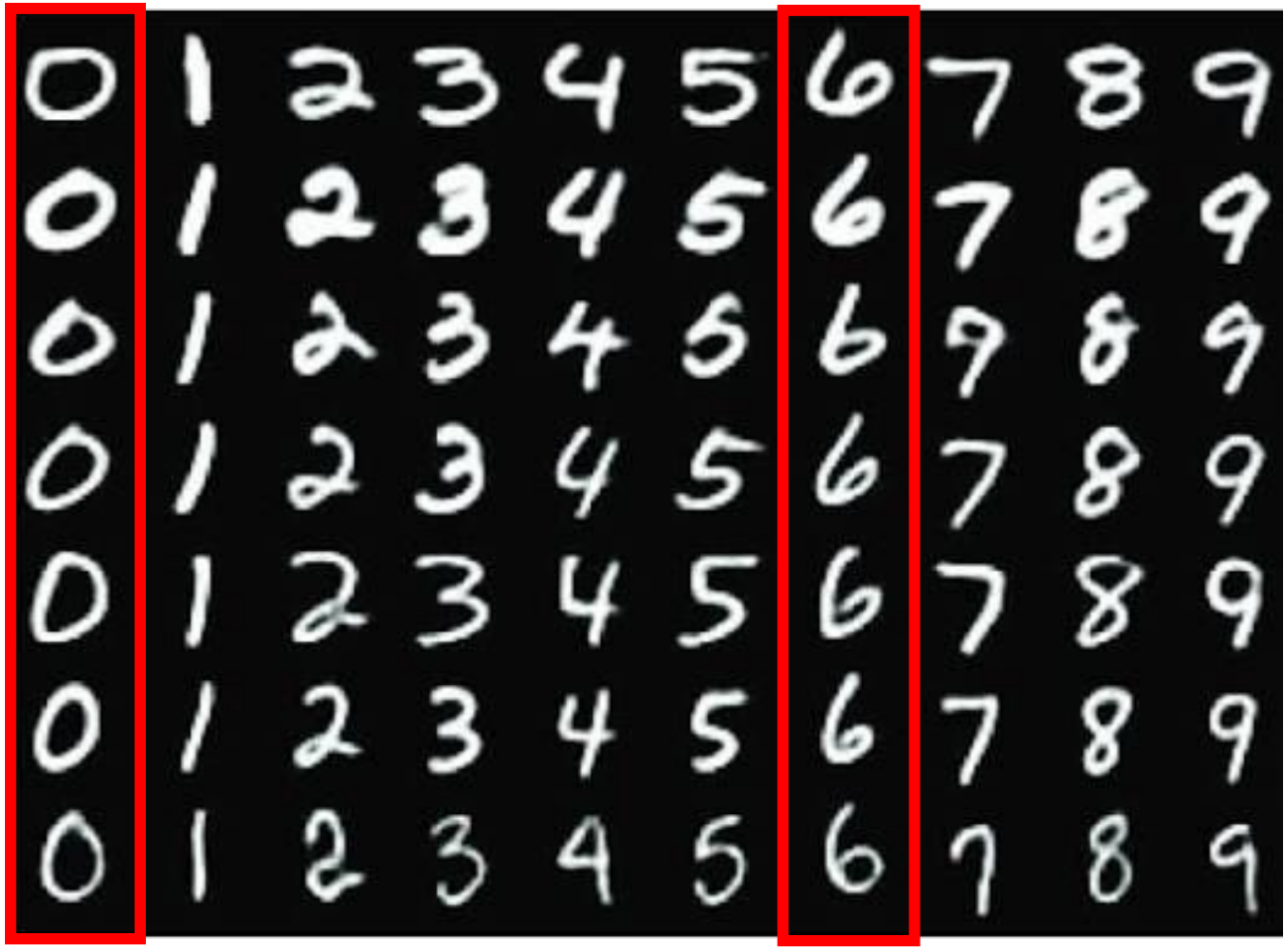
EX #2: The MNIST dataset

	0	1	2	3	4	5	6	7	8	9
0	41	2	0	0	2	0	15	0	2	0
1	2	40	4	0	0	1	0	3	12	2
2	0	2	29	19	0	3	0	1	1	7
3	2	2	7	26	0	11	0	3	2	11
4	2	1	0	0	54	1	0	2	1	2
5	2	8	1	13	0	15	0	5	10	10
6	9	0	0	0	1	0	53	0	0	0
7	0	7	0	2	0	3	0	47	2	2
8	1	15	0	5	0	12	0	1	17	10
9	3	6	4	5	0	9	0	7	7	22



EX #2: The MNIST dataset

	0	1	2	3	4	5	6	7	8	9
0	41	2	0	0	2	0	15	0	2	0
1	2	40	4	0	0	1	0	3	12	2
2	0	2	29	19	0	3	0	1	1	7
3	2	2	7	26	0	11	0	3	2	11
4	2	1	0	0	54	1	0	2	1	2
5	2	8	1	13	0	15	0	5	10	10
6	9	0	0	0	1	0	53	0	0	0
7	0	7	0	2	0	3	0	47	2	2
8	1	15	0	5	0	12	0	1	17	10
9	3	6	4	5	0	9	0	7	7	22



EX #2: The MNIST dataset

	0	1	2	3	4	5	6	7	8	9
0	41	2	0	0	2	0	15	0	2	0
1	2	40	4	0	0	1	0	3	12	2
2	0	2	29	19	0	3	0	1	1	7
3	2	2	7	26	0	11	0	3	2	11
4	2	1	0	0	54	1	0	2	1	2
5	2	8	1	13	0	15	0	5	10	10
6	9	0	0	0	1	0	53	0	0	0
7	0	7	0	2	0	3	0	47	2	2
8	1	15	0	5	0	12	0	1	17	10
9	3	6	4	5	0	9	0	7	7	22



In summary: Strengths/weaknesses of KNN

Strengths

- **Simple and intuitive** to implement. Results are interpretable
- **No training required** as it simply stores the training data
- **Can learn complex, nonlinear decision boundaries**

Weaknesses

- **Slow predictions:** must compare a new sample to all) training samples
- **Memory intensive:** stores the entire training dataset
- **Performance degrades in high dimensions** (may combine with PCA)
- **Choice of k is important** and can be difficult to nail down

Questions?

What to know for HW + Exams

- How to do KNN classification by hand
- Understand the influence of *k* on decision boundaries
- **Classification metrics and visuals:**
  - **Accuracy, precision, recall, F_1-score**
  - **Single-class *vs.* multiclass**
  - **Confusion matrices**
  - **ROC and AUC**