Data Analysis and Machine Learning 4 (DAML)

Week 7: Model selection and evaluation

Elliot J. Crowley, 4th March 2024

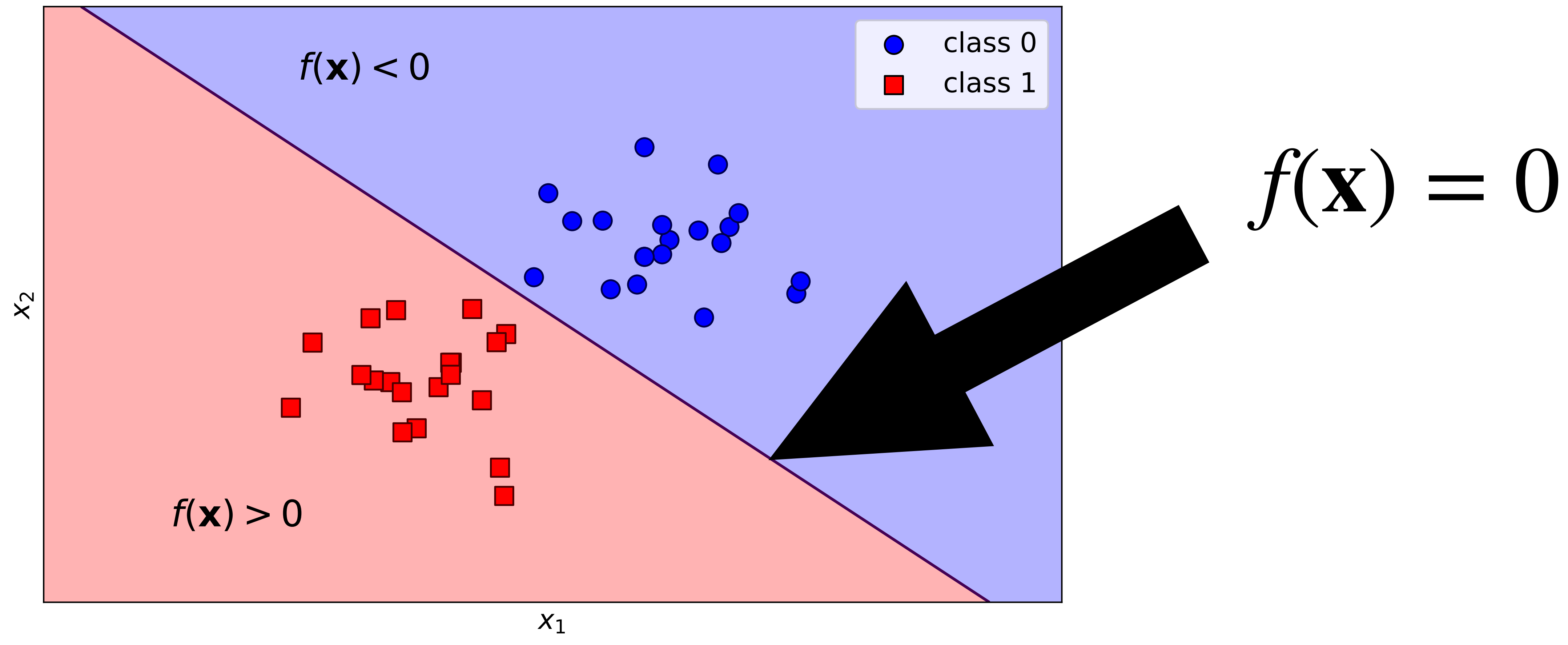
Recap

- We learnt that a linear classifier consisted of a linear model + threshold function

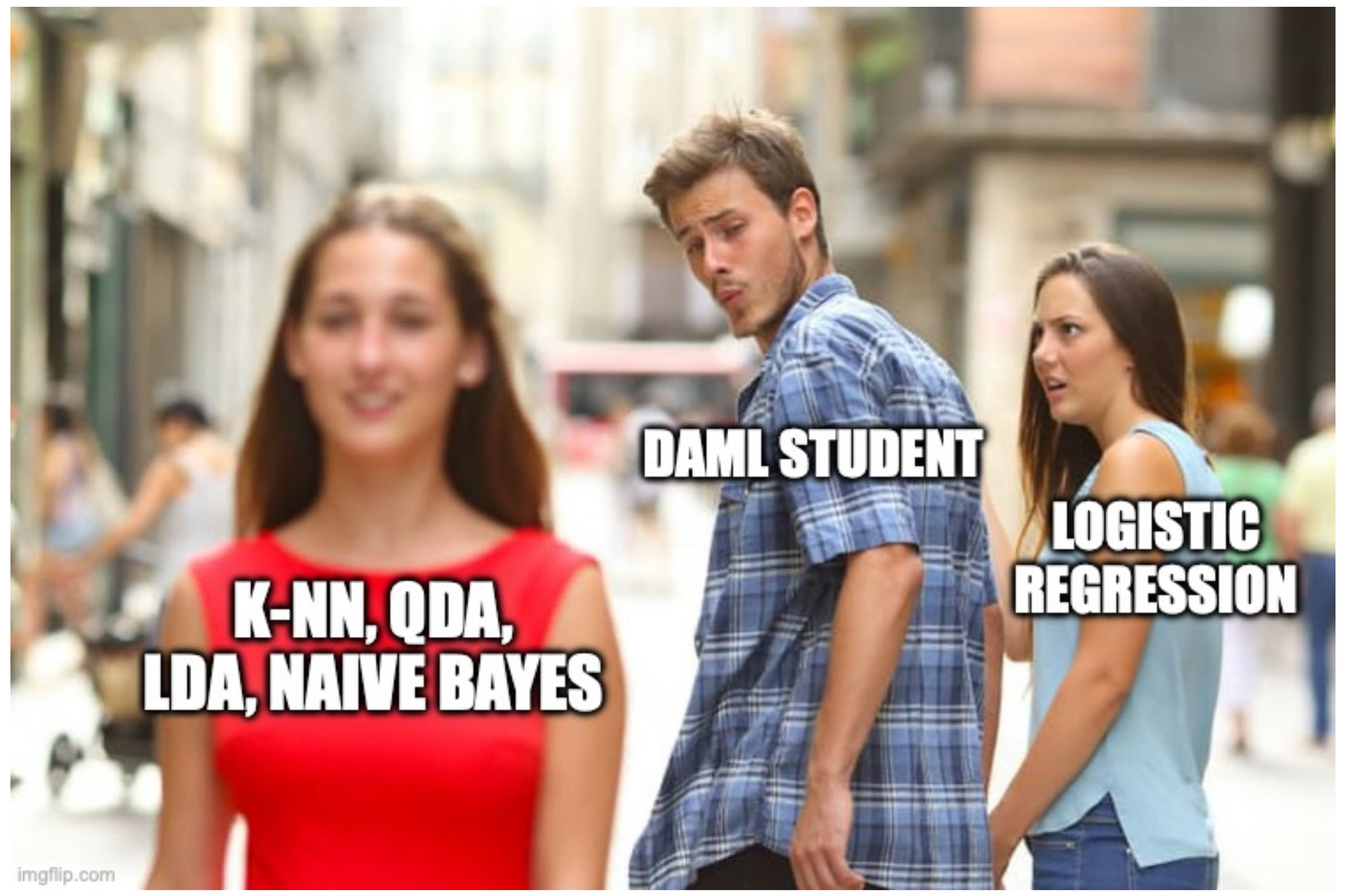
- We saw that this gave rise to a (linear) decision boundary

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

$$\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases}$$



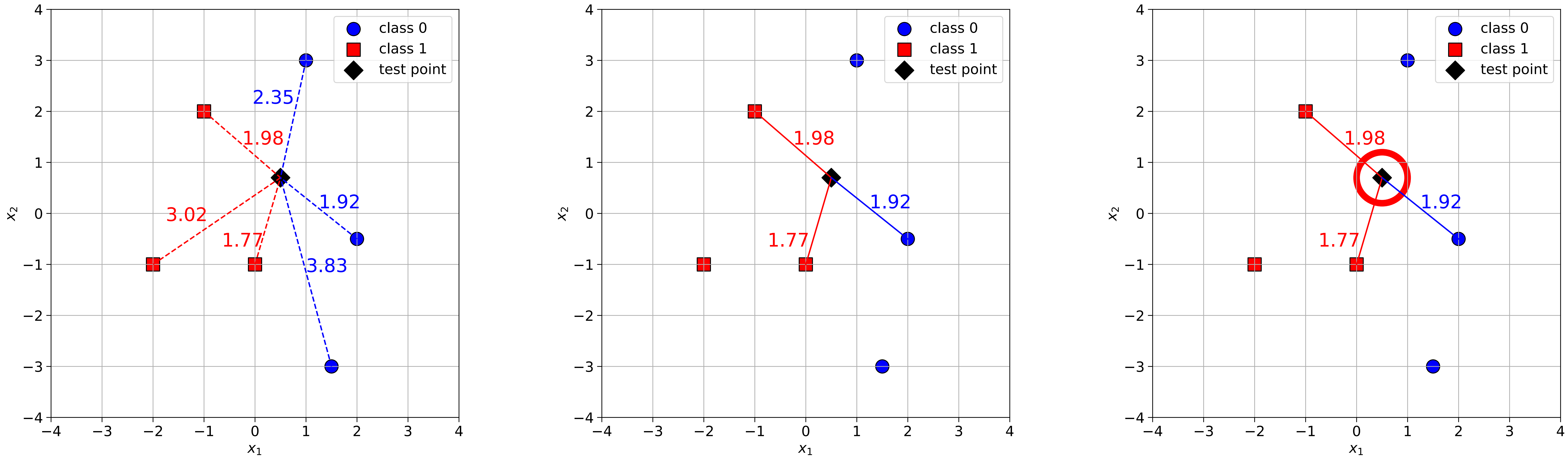
- We looked at different loss functions for fitting the model parameters



More classifiers

k-nearest neighbours (k-NN)

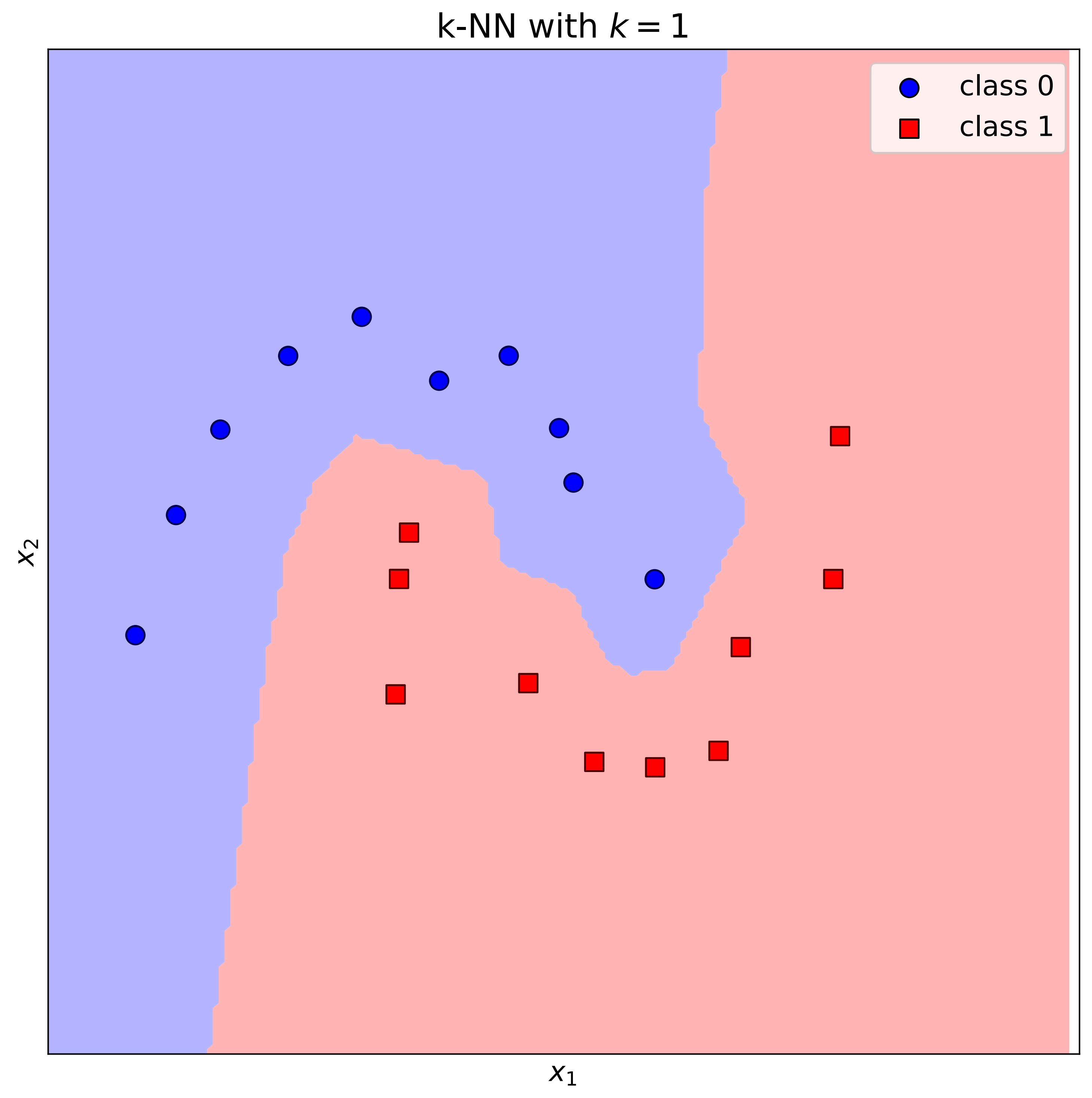
- A simple non-parametric model for classification with hyperparameter k
- Classify according to the mode class label of the k closest training points

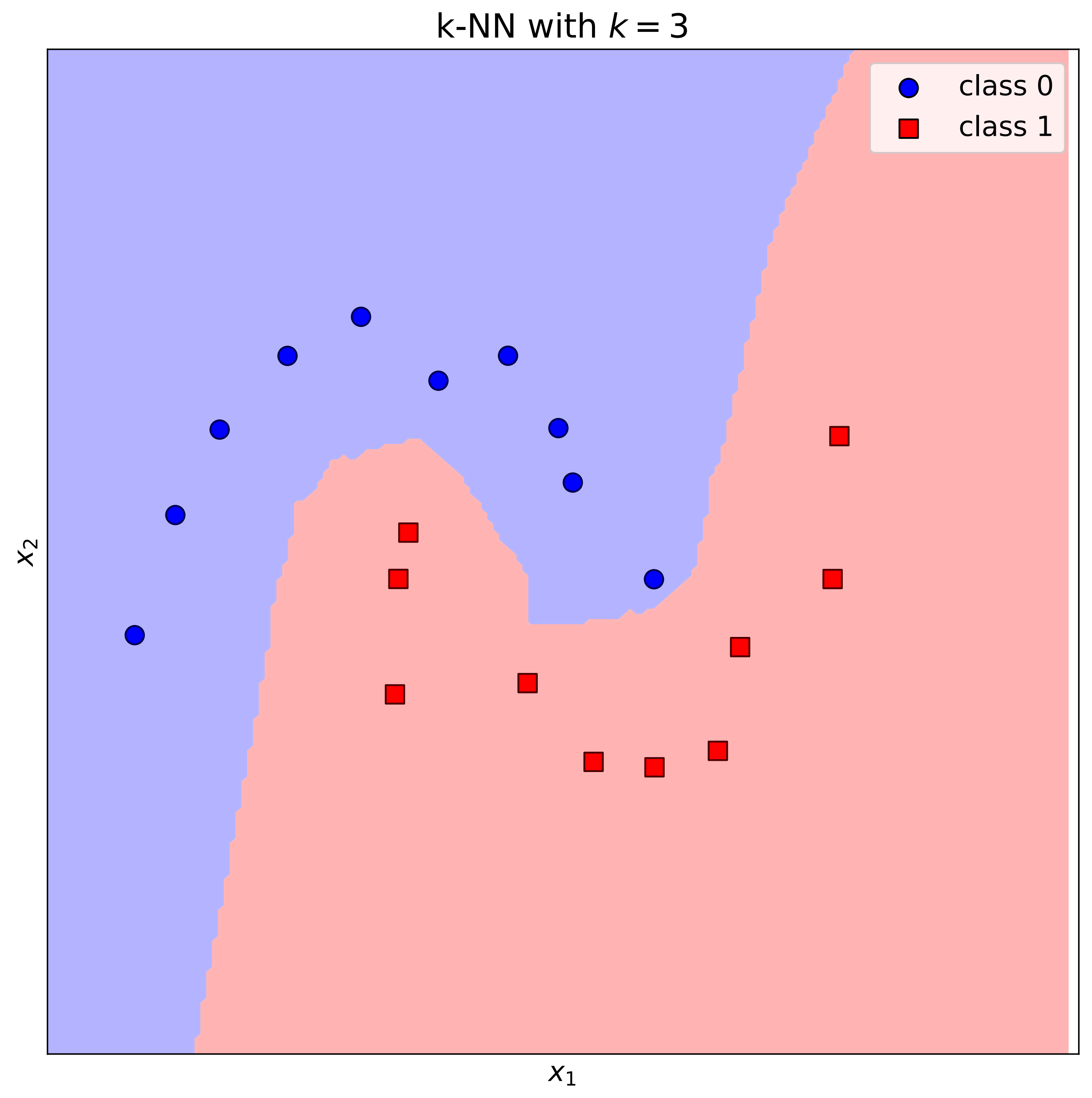


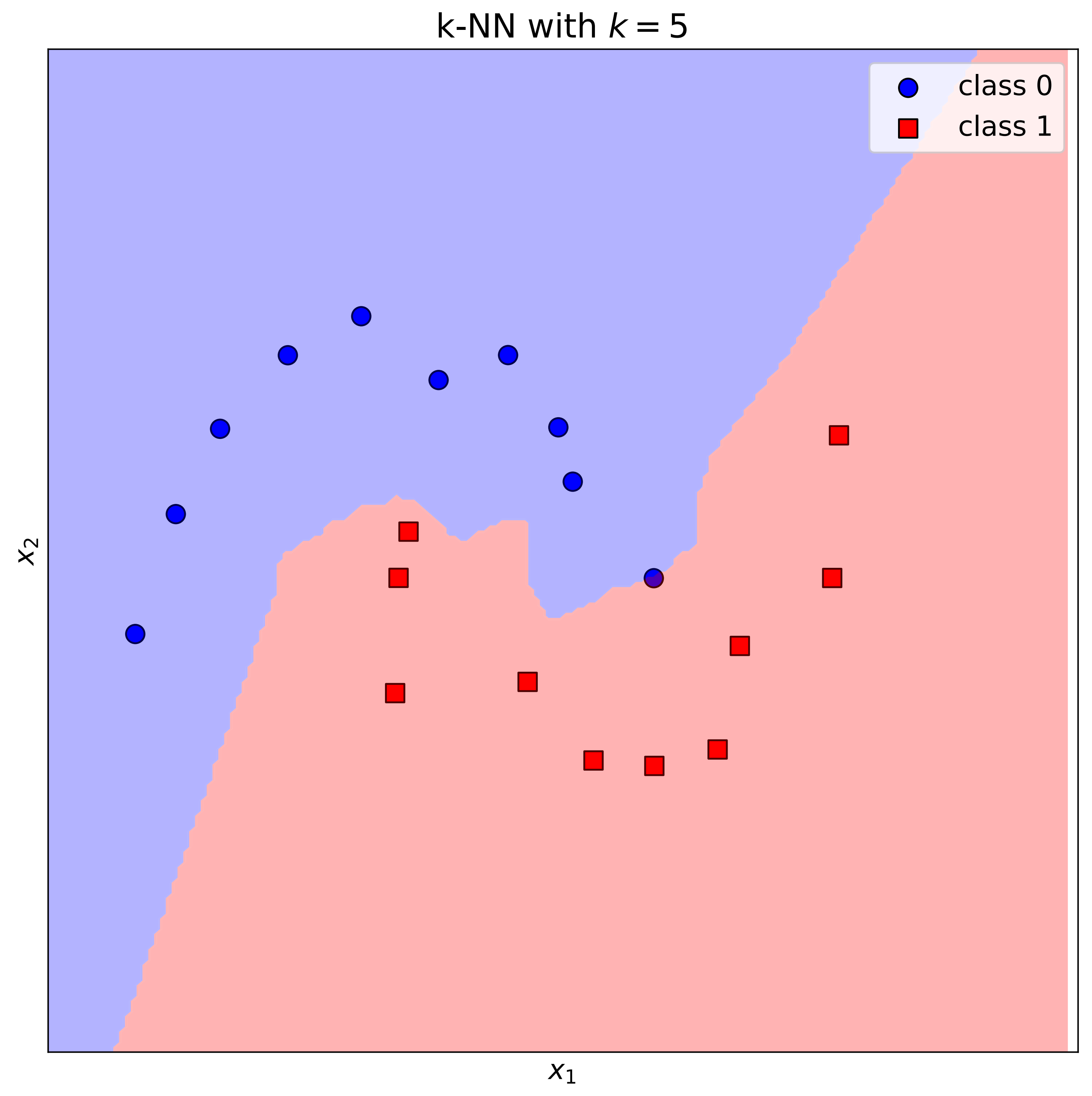
Here we are using $k = 3$

k-NN decision boundary

- Classify according to the mode class label of the k closest training points
- This gives a non-linear decision boundary

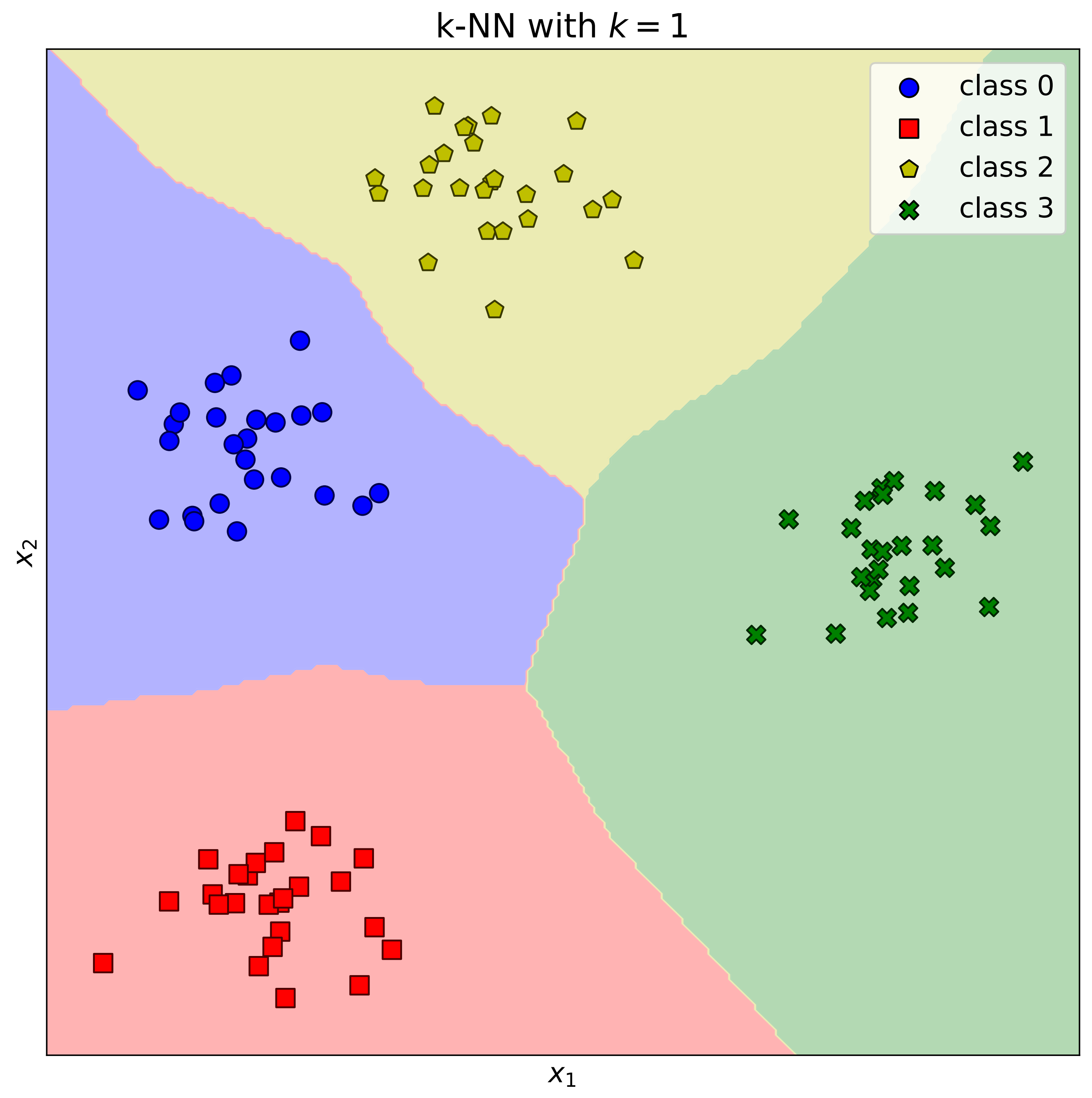


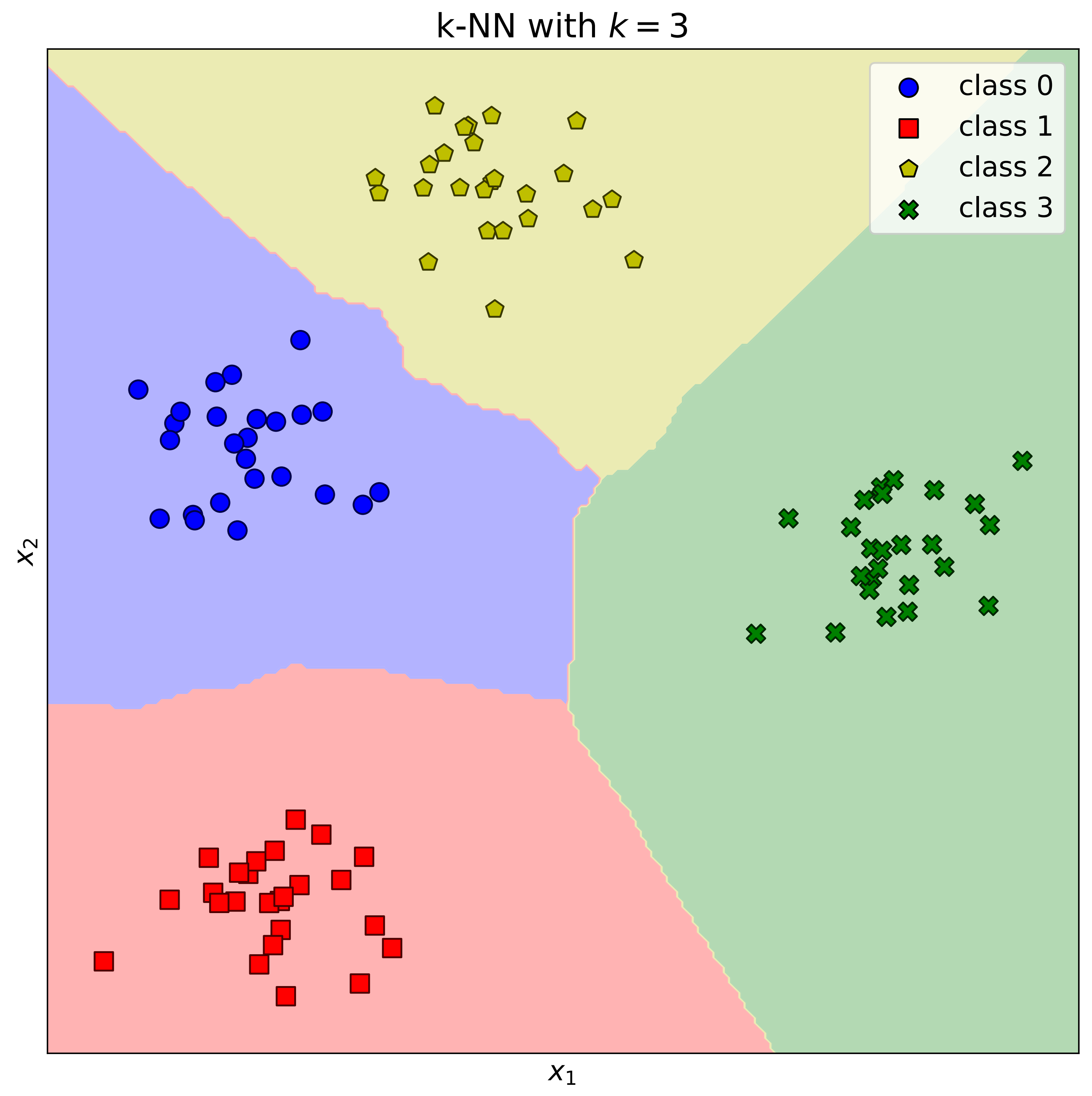


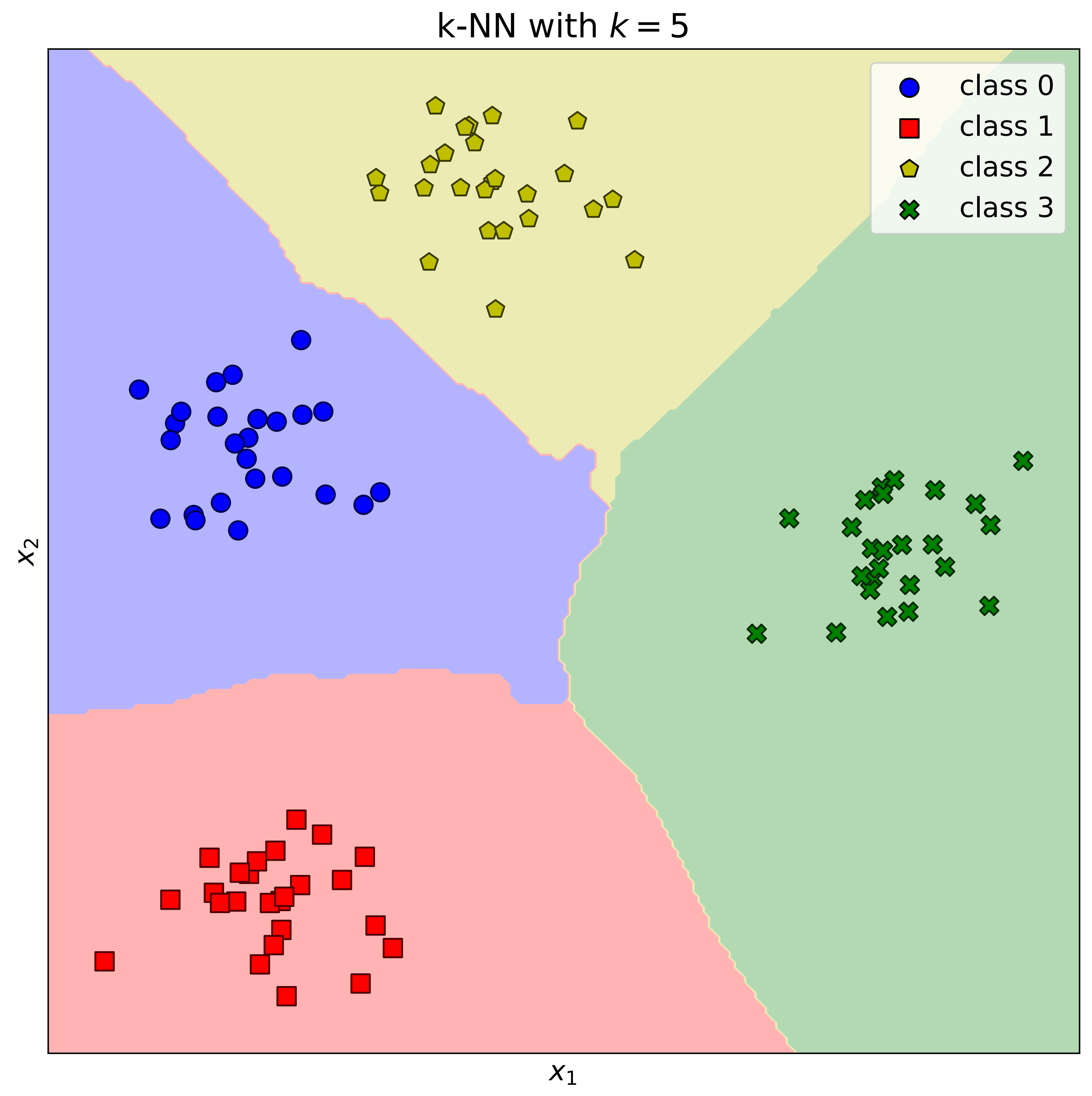


k-NN for multi-class classification

- Classify according to the mode class label of the k closest training points
- This can be naturally applied to multi-class problems







Quadratic discriminant analysis (QDA)

- Using Bayes rule, we can write the probability that a class label is c given $\mathbf{x}$ as

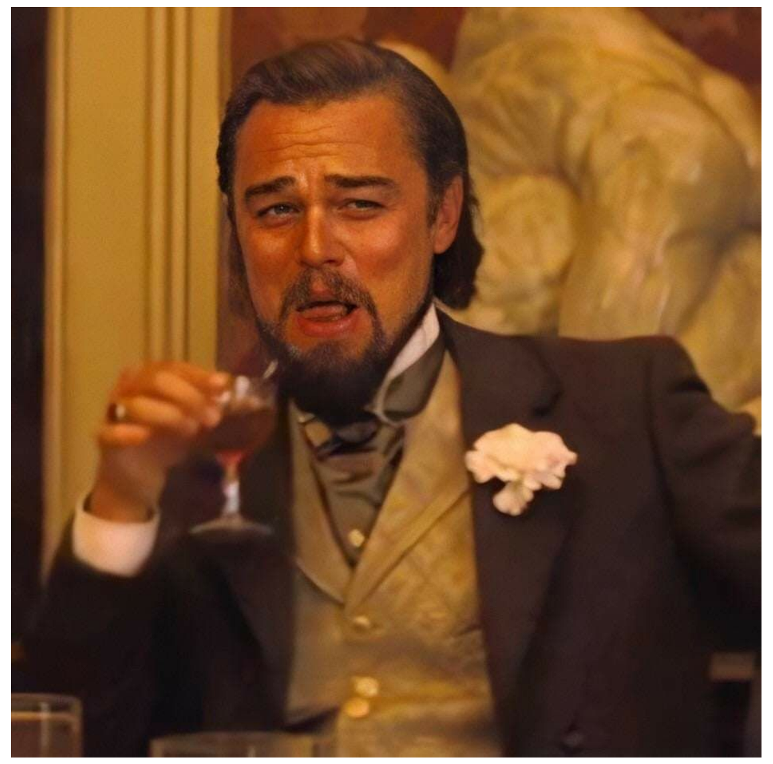
$$p(y = c \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = c)p(y = c)}{p(\mathbf{x})}$$

- $p(\mathbf{x})$ is independent of c so we can ignore it and classify according to $\underset{c}{\operatorname{argmax}}\, p(\mathbf{x} \mid y = c)p(y = c)$

- $p(y = c)$ is just the fraction of our training data in class c

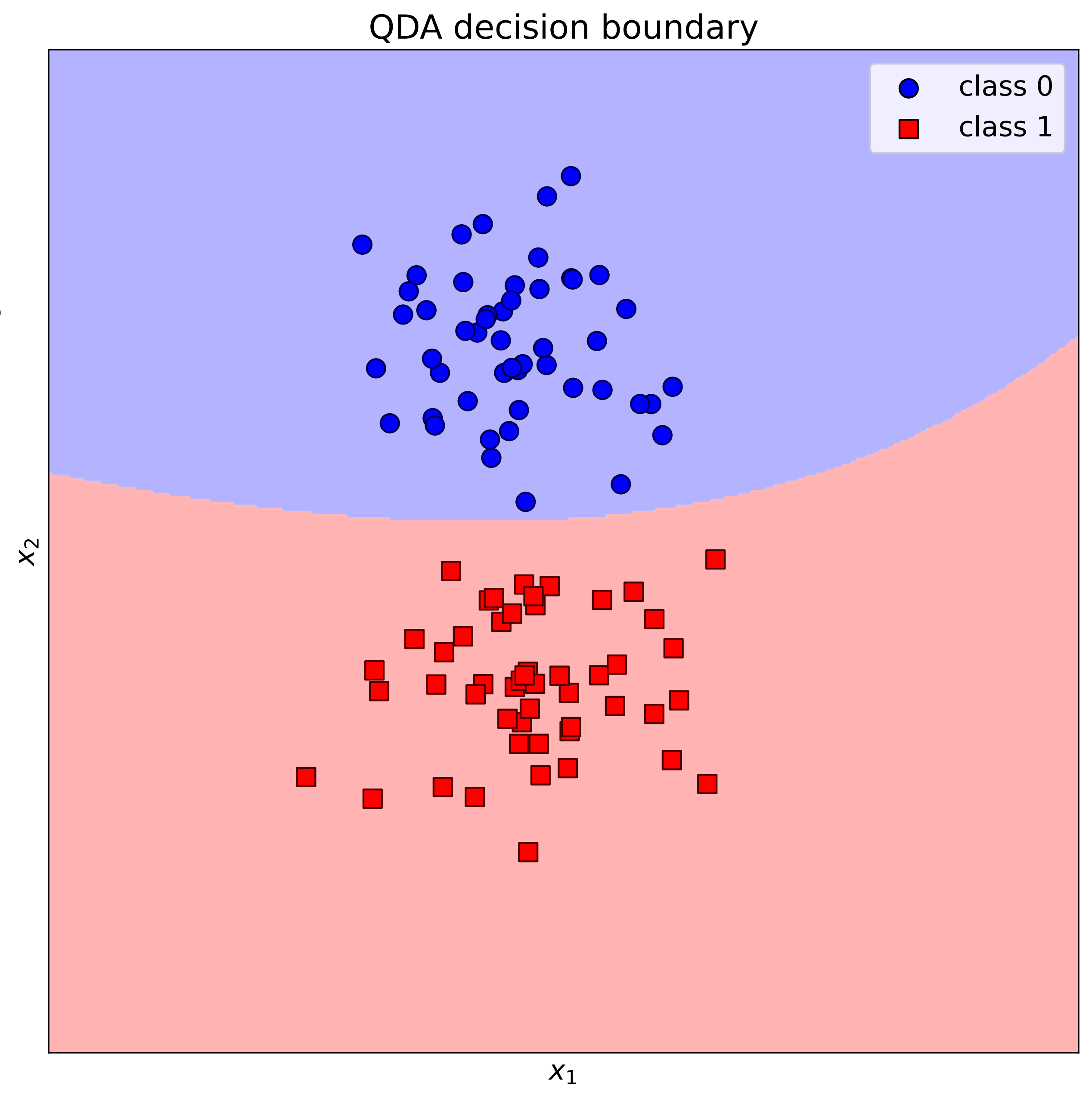
- We can make the **normal** assumption $p(\mathbf{x} \mid y = c) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$

This is a multivariate Gaussian where the mean is a vector and the covariance is a matrix. You'll look at these in detail in Lecture 9

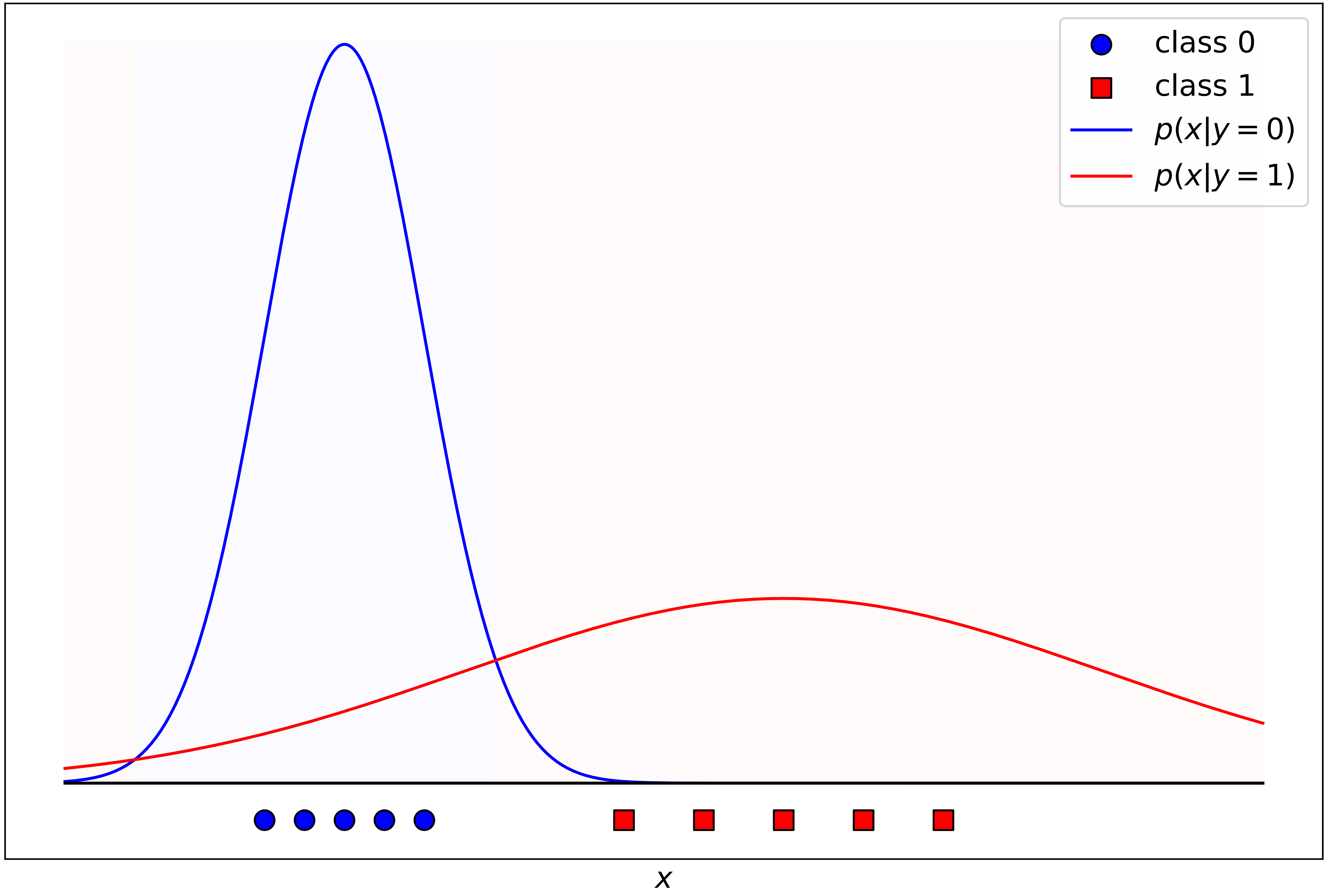


QDA continued

- We can use maximum likelihood estimation (MLE) to compute $\boldsymbol{\mu}_c \in \mathbb{R}^D$ and $\boldsymbol{\Sigma}_c \in \mathbb{R}^{D \times D}$ for each class from the training data
- This is just the mean and covariance of the points in each class!
- This gives us a quadratic decision boundary between classes
- This is a **generative classifier** as we can sample from $p(\mathbf{x} \mid y = c) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ to generate points for class c

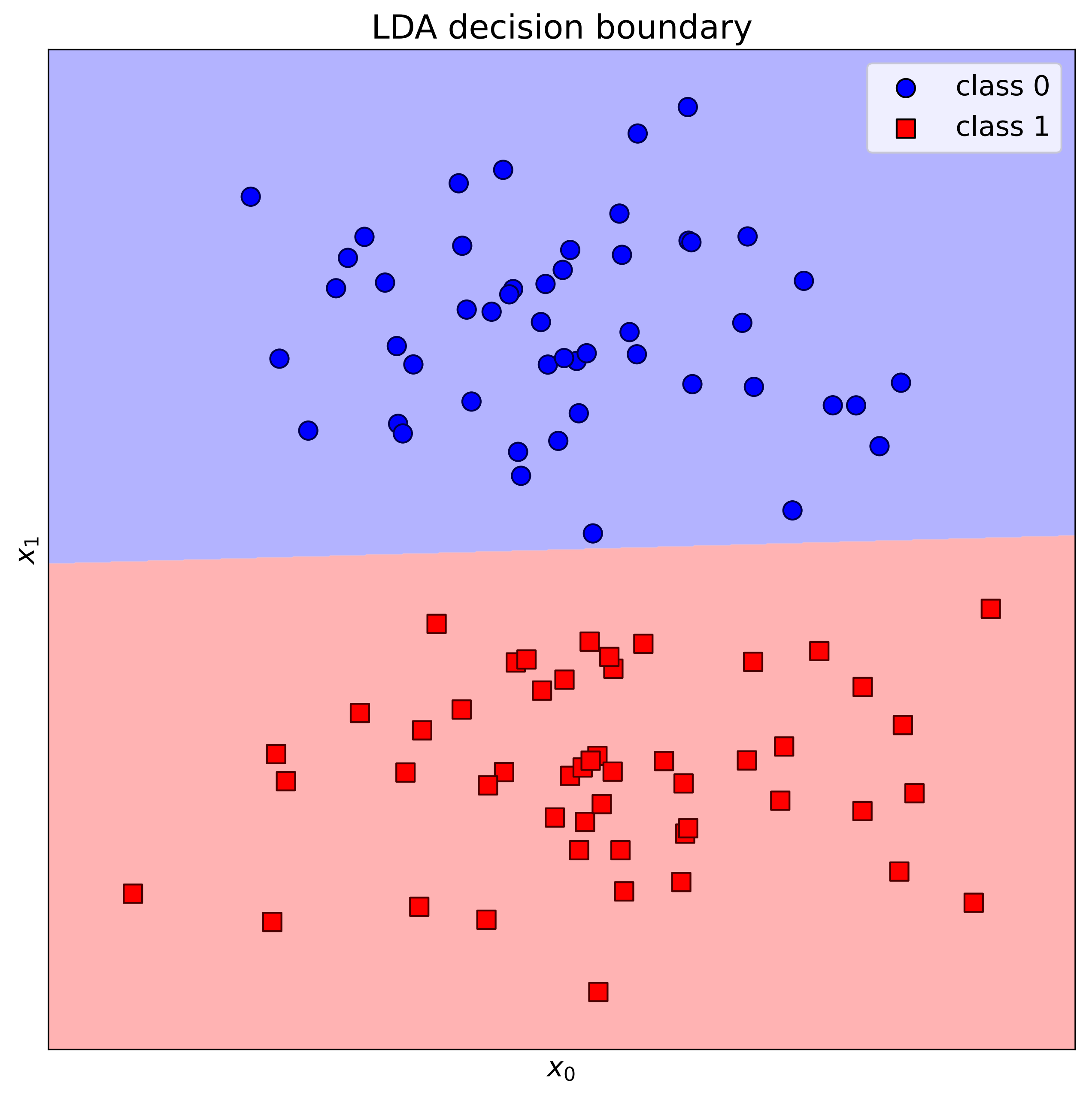


1D QDA example



Linear discriminant analysis (LDA)

- In QDA we classified according to $\underset{c}{\operatorname{argmax}}\, p(\mathbf{x} \mid y = c)p(y = c)$ where $p(\mathbf{x} \mid y = c) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$
- Let's make the simplifying assumption that $\boldsymbol{\Sigma}_c = \boldsymbol{\Sigma}$ for all classes
- If we perform MLE we now get a classifier with a linear decision boundary



Gaussian Naive Bayes classifier

- In QDA $p(\mathbf{x} | y = c) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ where $\boldsymbol{\Sigma}_c \in \mathbb{R}^{D \times D}$
- For large D the $\boldsymbol{\Sigma}_c$ matrices will be extremely large
- Gaussian Naive Bayes is like QDA except we assume that each $\boldsymbol{\Sigma}_c$ is diagonal
- This means that we are assuming features are independent of each other for a given class
- This lets us write $p(\mathbf{x} | y = c) = \prod_d p(x_d | y = c) = \prod_d \mathcal{N}(x_d; \mu_c, \sigma_c)$

Naive Bayes in general

- In Gaussian Naive Bayes $p(\mathbf{x}|y=c) = \prod_d p(x_d|y=c) = \prod_d \mathcal{N}(x_d;\mu_c,\sigma_c)$
- We are assuming each feature (conditioned on class) is normally distributed
- In **<insert distribution name here>** Naive Bayes we assume each $p(x_d|y=c)$ is **<insert distribution name here>** distributed
- Multinomial Naive Bayes is suitable for text classification on bag-of-words features

I like sausage	I hate sausage	sausage sausage
$[1 \quad 1 \quad 1 \quad 0]^\top$	$[1 \quad 0 \quad 1 \quad 1]^\top$	$[0 \quad 0 \quad 2 \quad 0]^\top$

No free lunch

- In terms of classification you know about Logistic regression, Perceptrons, Support Vector Machines, k-nearest neighbours, QDA, LDA, and Naive Bayes

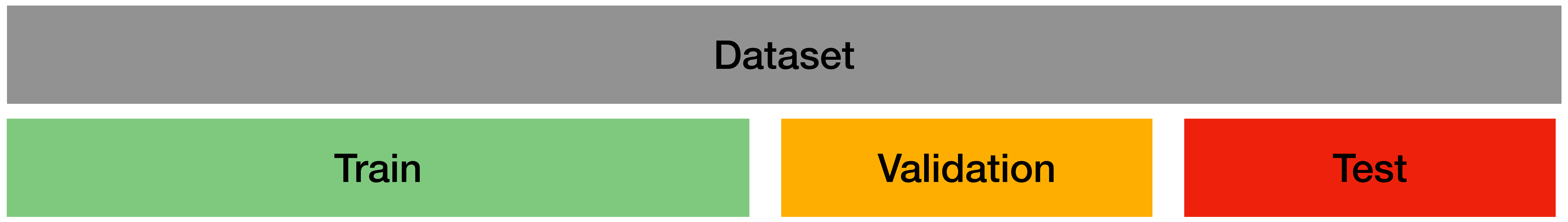
- You are given some data and you have to solve a classification task. Which *model type* do you pick? Which is the best?

- Unfortunately… **there is no universal best model! There is no free lunch!**

Model selection

Dataset splits

- Given a dataset and a task (e.g. classification) in machine learning we typically divide our dataset into a **training set,** a **validation set,** and a **test set**
- **The training set is used for training models**
- **The validation set is used for model selection**
- **The test set is used for a FINAL EVALUATION OF A CHOSEN MODEL**



Warning!

- The whole point of ML is to learn a model that will work well on new data
- The purpose of the test set is to give you a unbiased estimate of how well your final chosen model works on new data before you deploy it
- **Never train on test data**
- **Never perform model selection on test data**
- Set aside your test set at the start and **don't look at it** until you have selected a final model

You should evaluate on the test set as little as possible, ideally only once!

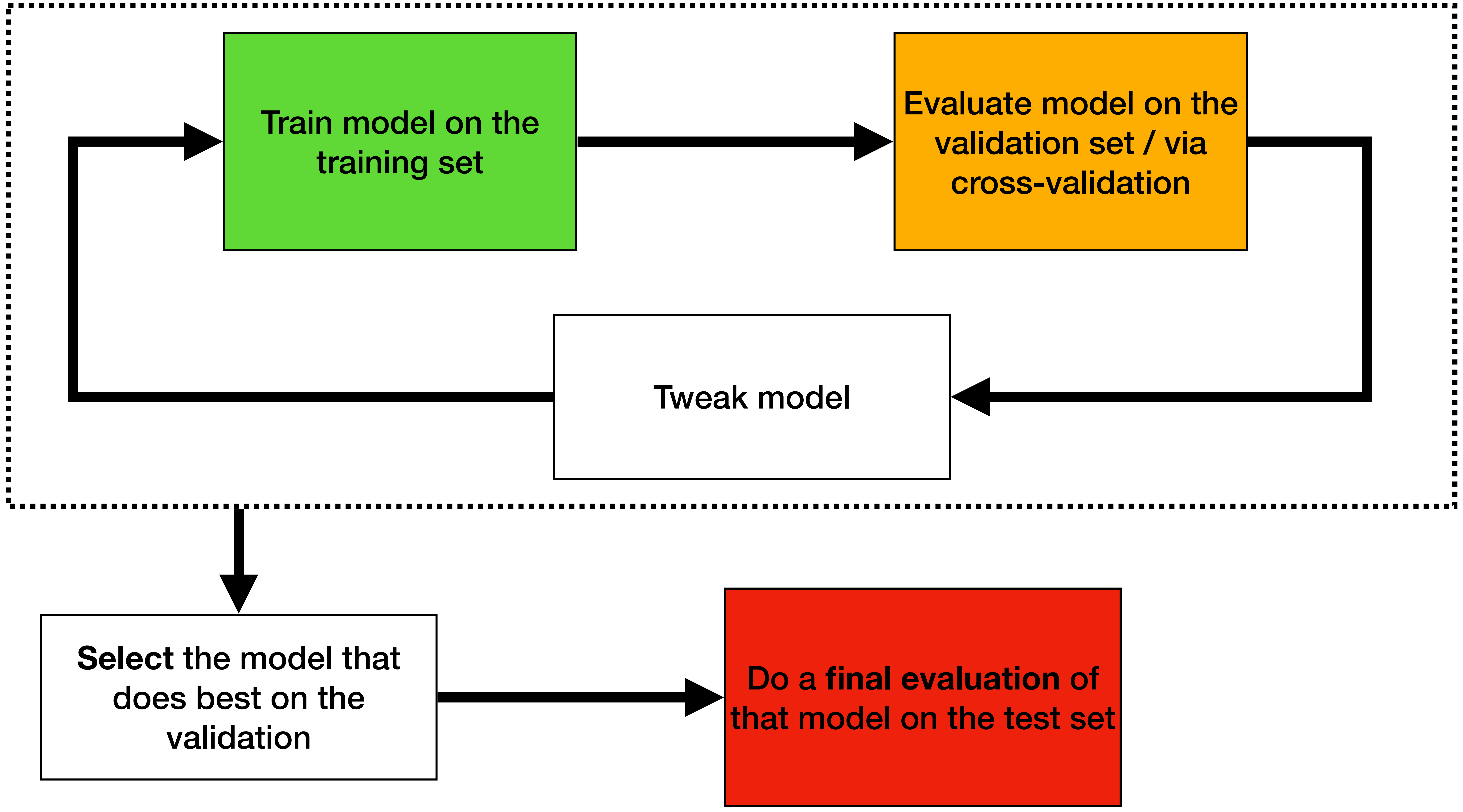
Some pitfalls to be aware of

- Watch out for duplicates! The same point could end up in both train and test
- Consider a medical task where you have data points associated with different patients — points from the same patient mustn't be in both train and test
- Your data points might be measured at certain times — train points should occur **before** test points (and with a sensible gap in time between the two)
- Don't use test data to compute e.g. statistics, PCA etc. (more on this later)
- Don't use features that were measured **after** your targets

Model selection

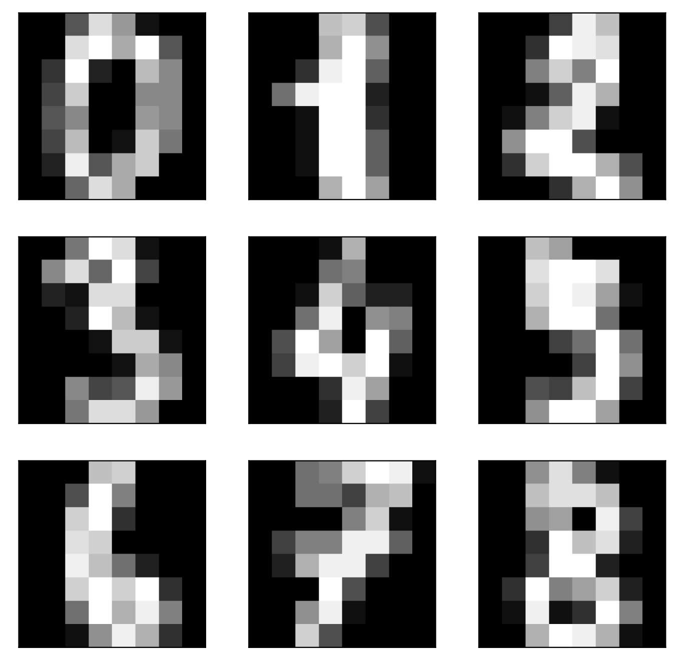
- Model selection is the problem of finding the best model for a given task
- A model has some **model type** (e.g. SVM, k-NN) and **hyperparameters** (e.g. regularisation strength)
- We use the training set to train models for different types of model and different hyperparameters
- We use a dedicated validation set (or perform cross-validation) to evaluate those models and select the best one
- What is "best"? This depends on your desiderata. We will usually assume it is the model that maximises some score e.g. accuracy for classification

A general ML workflow


Train model on the training set
Evaluate model on the validation set / via cross-validation
Tweak model
Select the model that does best on the validation
Do a **final evaluation** of that model on the test set

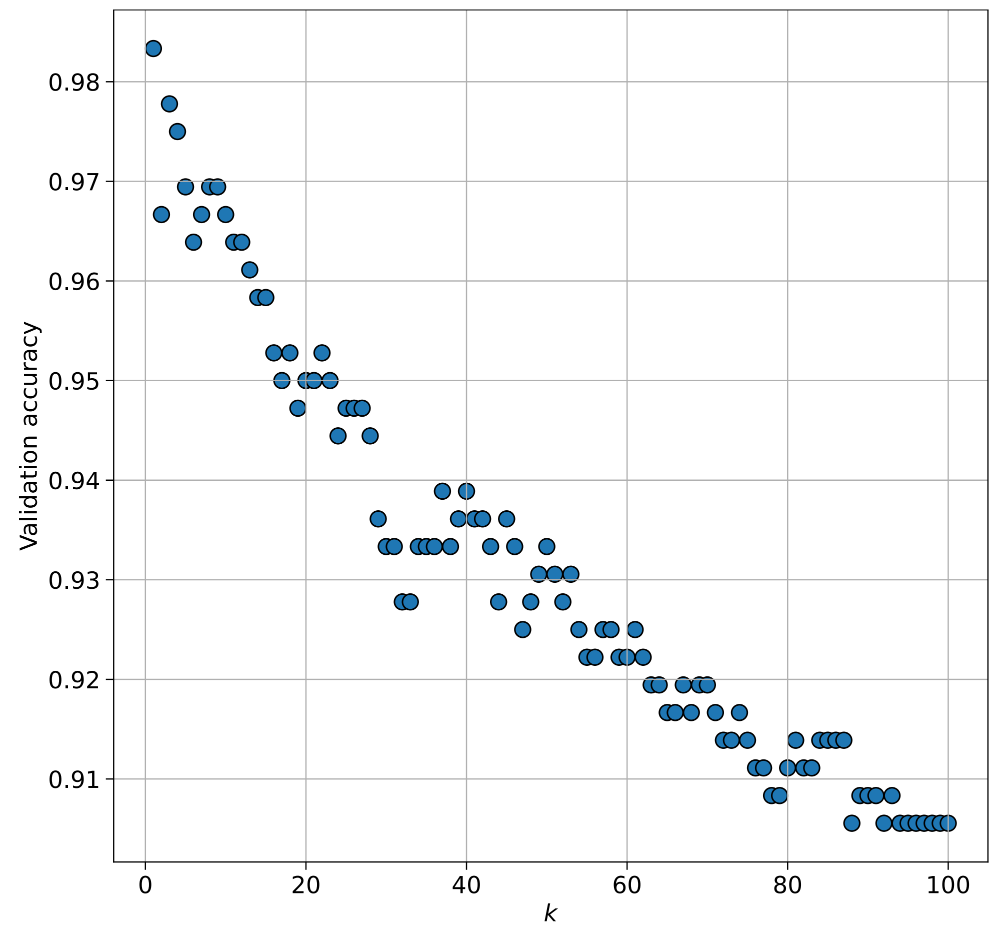
Inspiration: https://developers.google.com/machine-learning/crash-course/validation/another-partition

Example

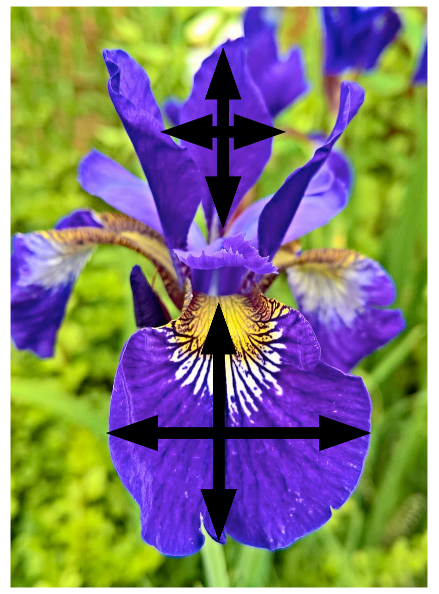


- Task: 10-way digit classification
- Dataset: 1797 vectorised images $\mathbf{x} \in \mathbb{R}^{64}$ labelled $y \in \mathbb{Z}^{+}_{<10}$

1. I split the dataset into 60/20/20 train/val/test
2. I choose to only consider k-NN models
3. I evaluate $k-$NN (which uses the training set) for $k = 1,...,100$ on the validation set
4. I **select** the model that gets highest accuracy on val $(k = 1)$
5. Then, I evaluate my final chosen model on the test set and get 98.8% accuracy



Example



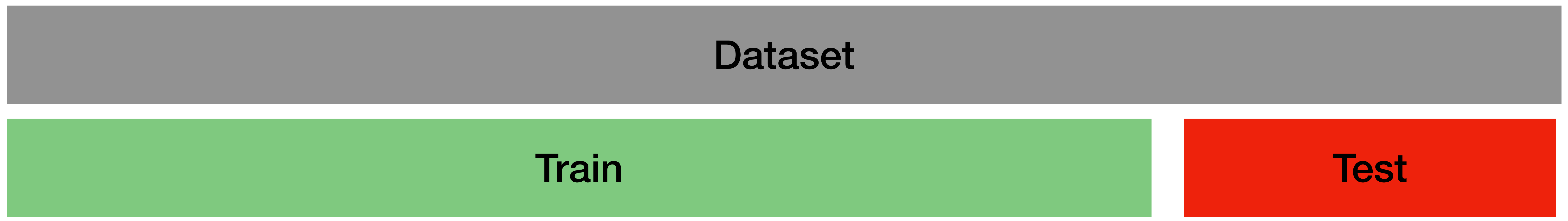
- Task: 3-way iris classification
- Dataset: 150 vectors of measurements $\mathbf{x} \in \mathbb{R}^4$ labelled $y \in \mathbb{Z}^+_{<3}$

1. I split the dataset into 50/25/25 train/val/test
2. I choose to consider 2 model types: LDA and logistic regression (with default hyperparameters)
3. I train a model for each model form on the training set
4. I evaluate these models on the validation set
5. I **select** logistic regression because it gets the highest accuracy on the validation set
6. Then, I evaluate my chosen model on the test set

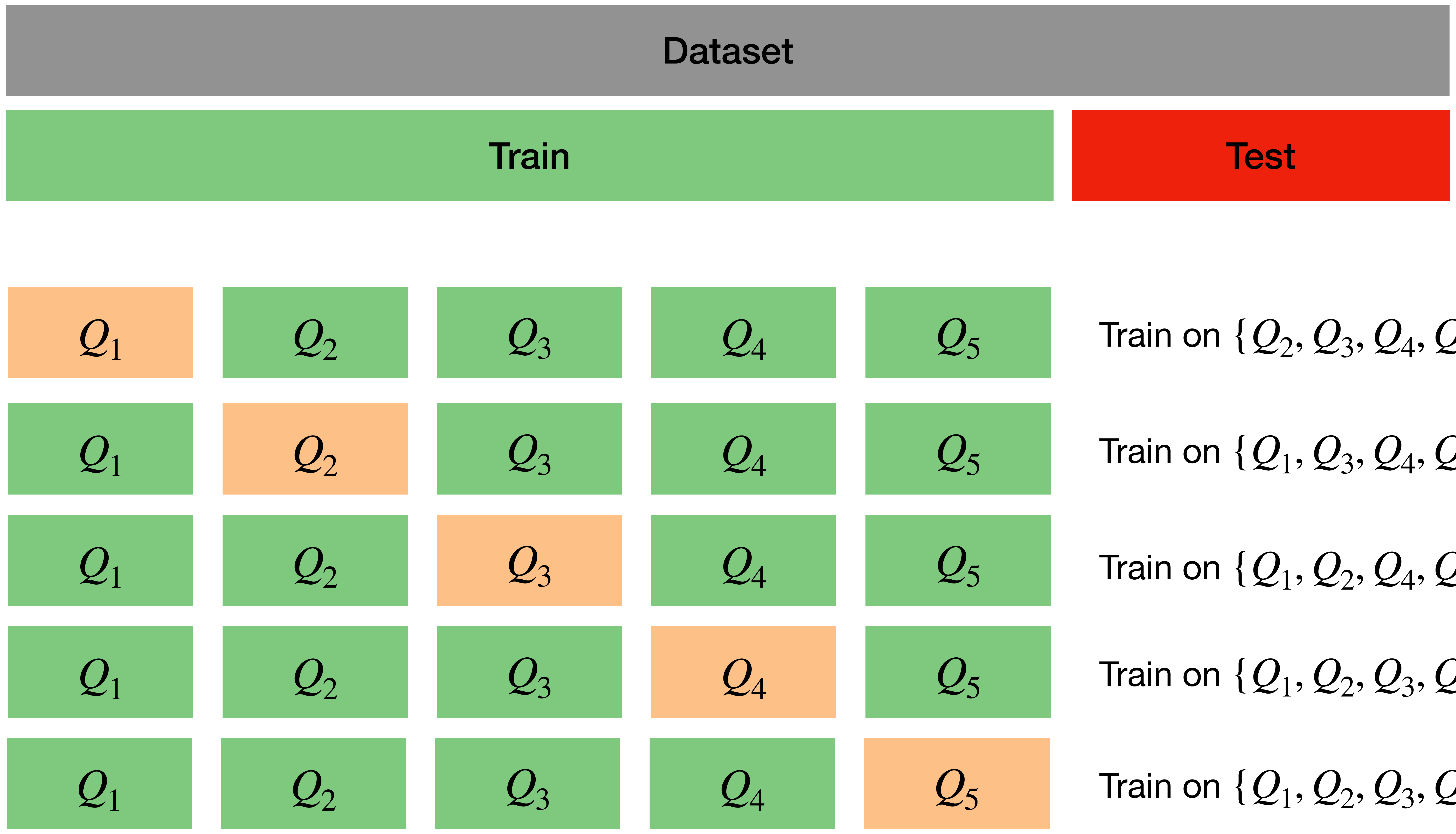
Typically, the chosen model is retrained on both train and validation to make the most of available data

Cross-validation

- We have been evaluating on a dedicated validation set for model selection
- This means the model we choose will be sensitive to the way the dataset was split up
- We can instead evaluate models through **cross-validation**
- This does not require us to have a dedicated validation set



k-fold cross validation ($k = 5$)


Dataset
Train
Test
Q_1 Q_2 Q_3 Q_4 Q_5
Q_1 Q_2 Q_3 Q_4 Q_5
Q_1 Q_2 Q_3 Q_4 Q_5
Q_1 Q_2 Q_3 Q_4 Q_5
Q_1 Q_2 Q_3 Q_4 Q_5

Train on $\{Q_2, Q_3, Q_4, Q_5\}$, evaluate on Q_1

Train on $\{Q_1, Q_3, Q_4, Q_5\}$, evaluate on Q_2

Train on $\{Q_1, Q_2, Q_4, Q_5\}$, evaluate on Q_3

Train on $\{Q_1, Q_2, Q_3, Q_5\}$, evaluate on Q_4

Train on $\{Q_1, Q_2, Q_3, Q_4\}$, evaluate on Q_5

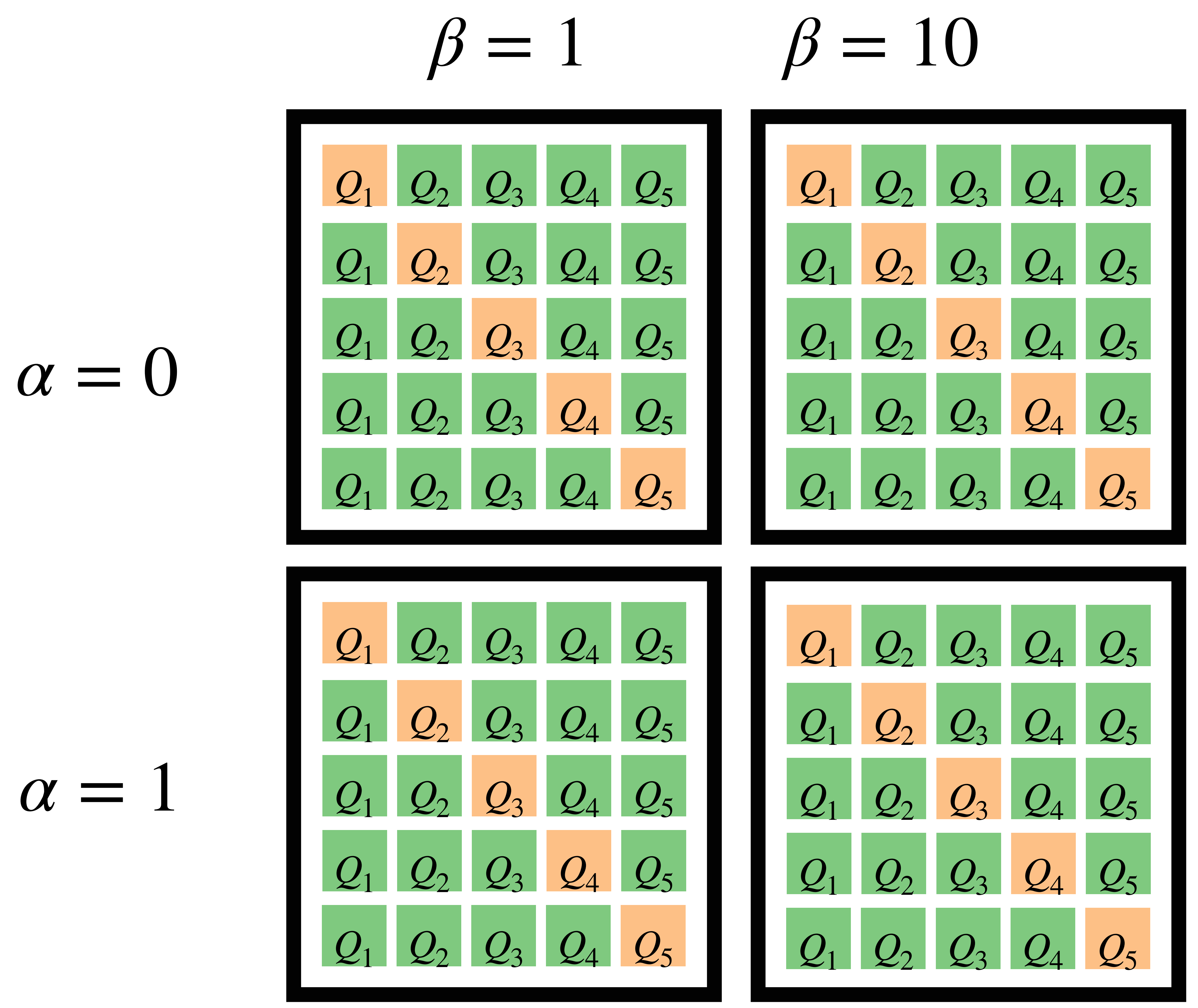
Then take average performance across Q_1, Q_2, Q_3, Q_4, Q_5

Inspiration: https://scikit-learn.org/stable/modules/cross_validation.html#multimetric-cross-validation

Grid search with k-fold cross validation

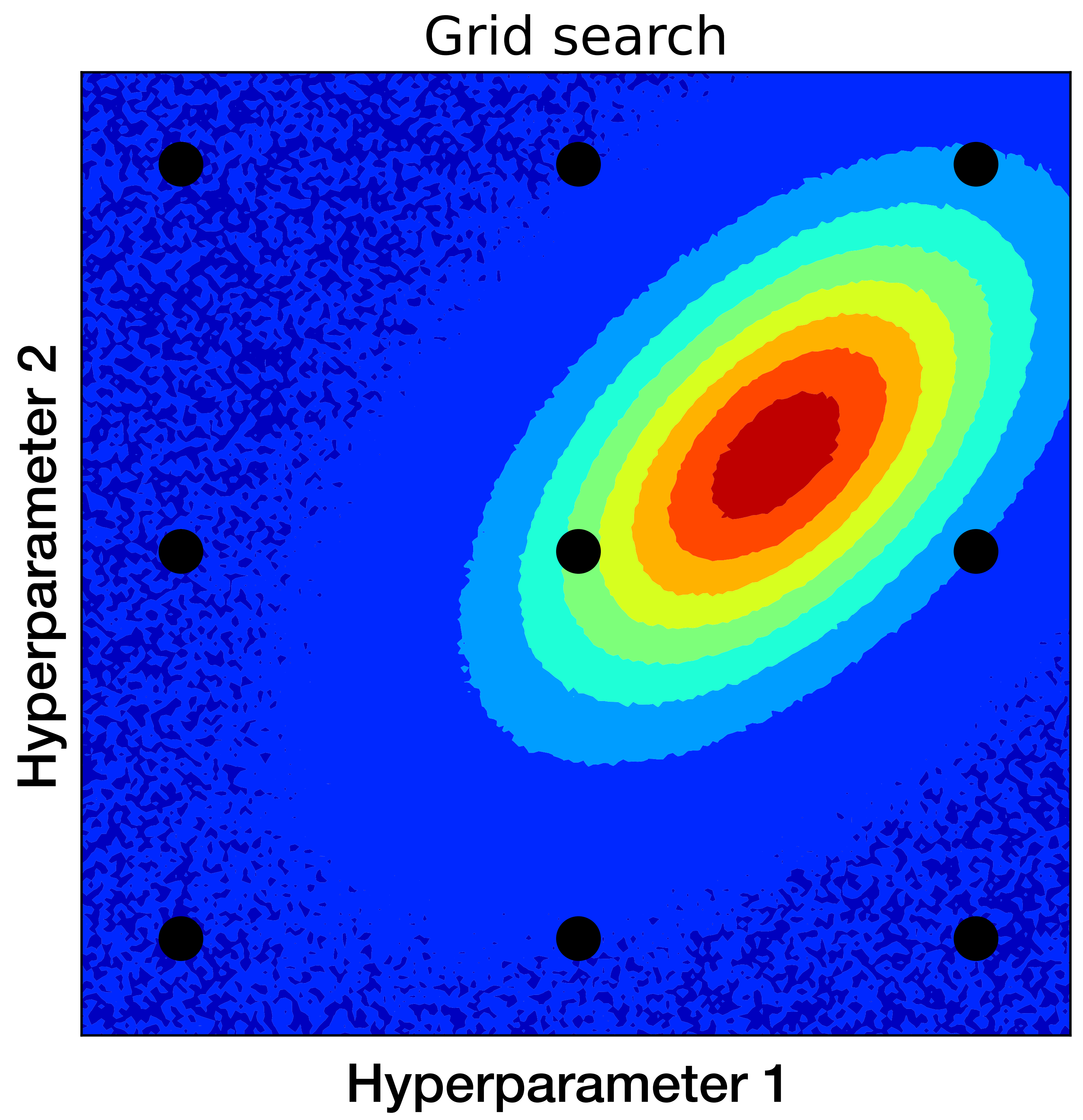
Imagine we have hyperparameters α and β.
Let's search over $\alpha = \{0,1\}$ and $\beta = \{1,10\}$

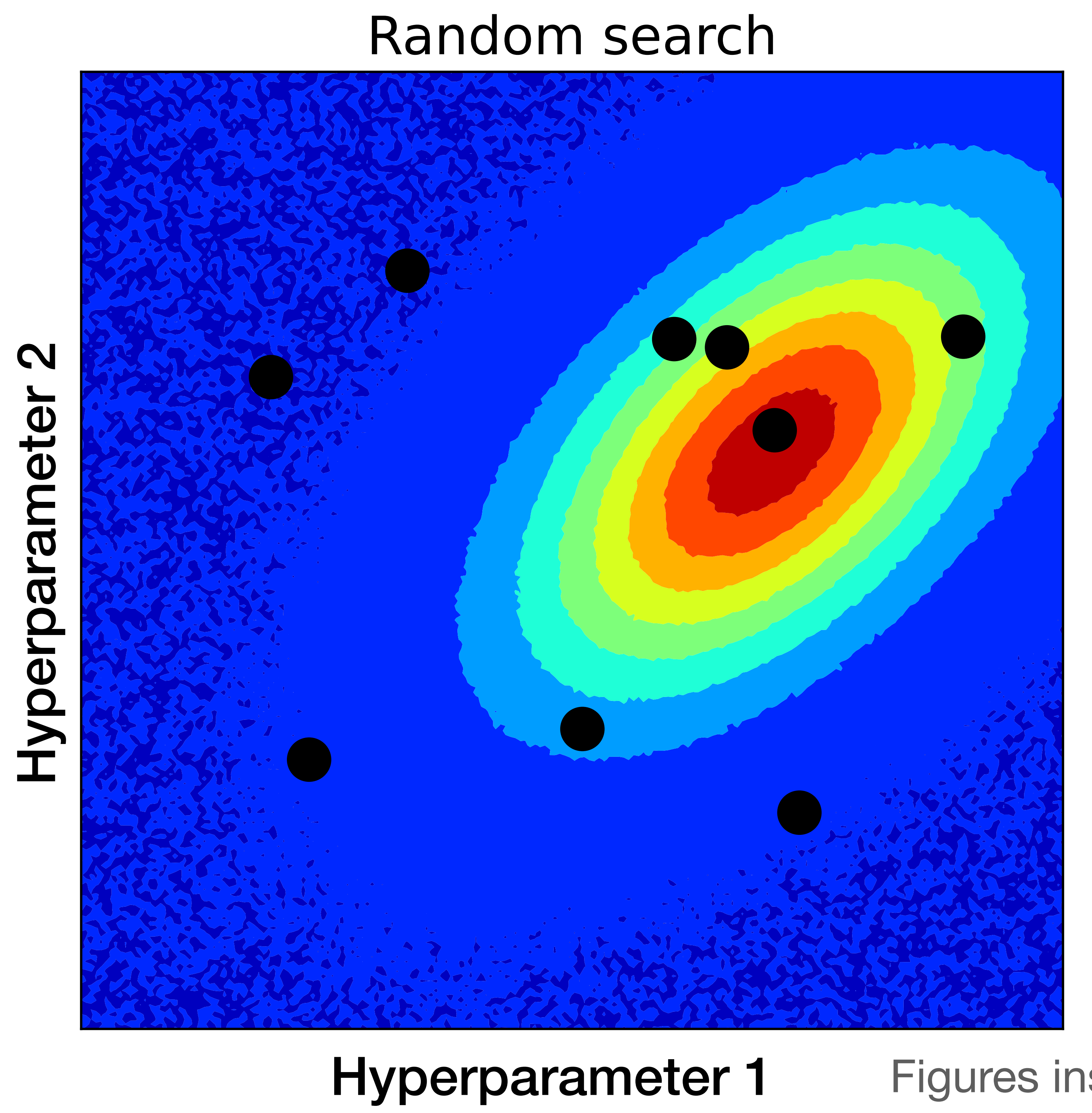
Cross-validation performance can be used in place of validation performance when doing a grid search



A note on grid search

- Grid search is an intuitive starting point for hyperparameter tuning
- But random search (and other schemes) work better in practice!





Figures inspired by Raschka et al.'s book

There are so many models to choose from

- Ideally, we'd try everything*: all model types + hyperparameter combinations
- But we don't have infinite compute, we need to be pragmatic!
- A reasonable strategy is to compare model types with default hyperparameters (on val/cross-val) …
- Then tune the hyperparameters of the most promising model type (on val/cross-val)

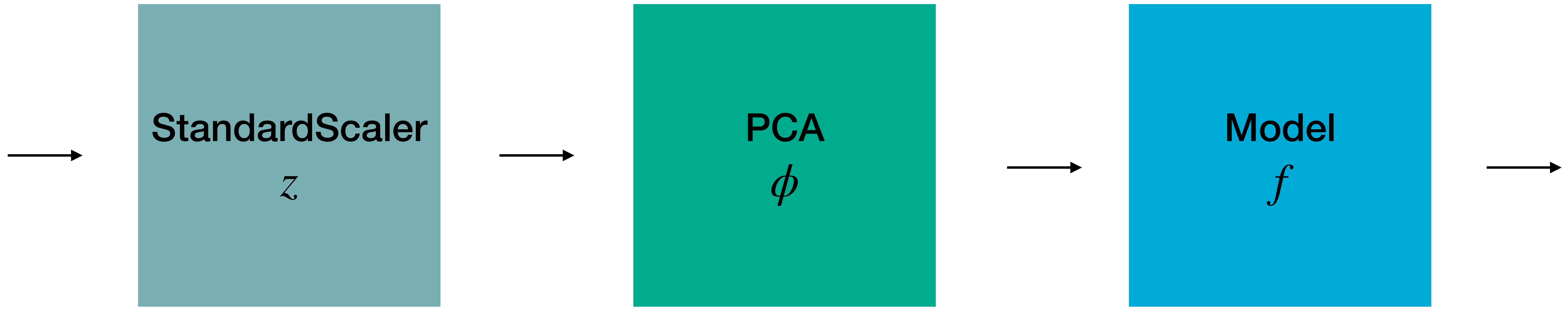
*Validation and test sets "wear out with repeated use" (see https://developers.google.com/machine-learning/crash-course/validation/another-partition at the bottom of the page)

We don't consider nested cross-validation in this course, but it's worth looking at if you have a moment: https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

But the model isn't the whole story…

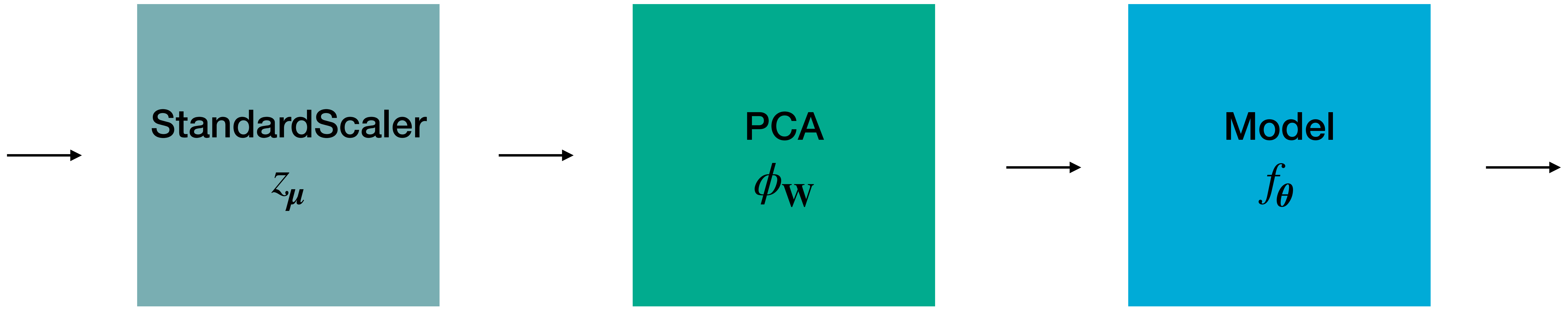
You might have a **pipeline** where, given an input $\mathbf{x}$…

- You normalise it with a standard scaler $z(\mathbf{x})$
- You then perform dimensionality reduction with PCA $\phi(z(\mathbf{x}))$
- You then put it through a model $f(\phi(z(\mathbf{x})))$



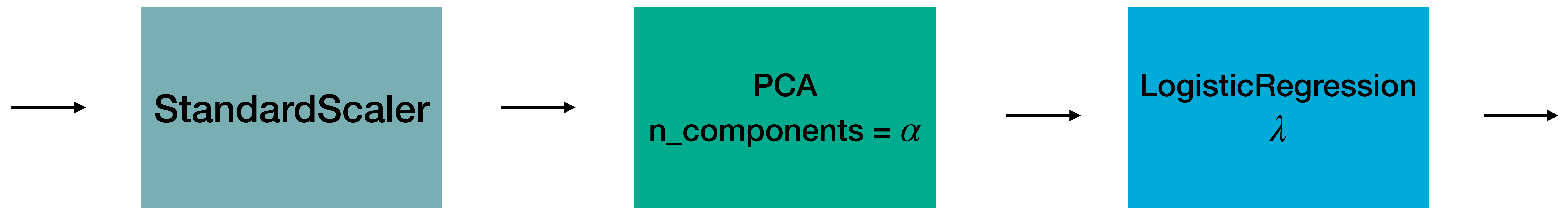
Data leakage and pipelines

- The scaler uses some statistics $\boldsymbol{\mu}$ so let's write $z_{\boldsymbol{\mu}}$
- PCA uses a matrix $\mathbf{W}$ so let's write $\phi_{\mathbf{W}}$
- The model has parameters $\boldsymbol{\theta}$ so let's write $f_{\boldsymbol{\theta}}$
- Test (or val) data must **<u>not</u>** be used to compute any of $\boldsymbol{\mu}, \mathbf{W}, \boldsymbol{\theta}$
- Using pipelines in sklearn helps prevent this data leakage



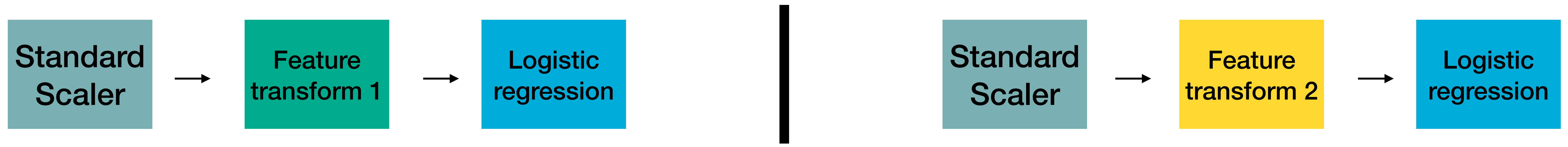
Pipelines

- We can (and should) tune hyperparameter combinations **within** pipelines



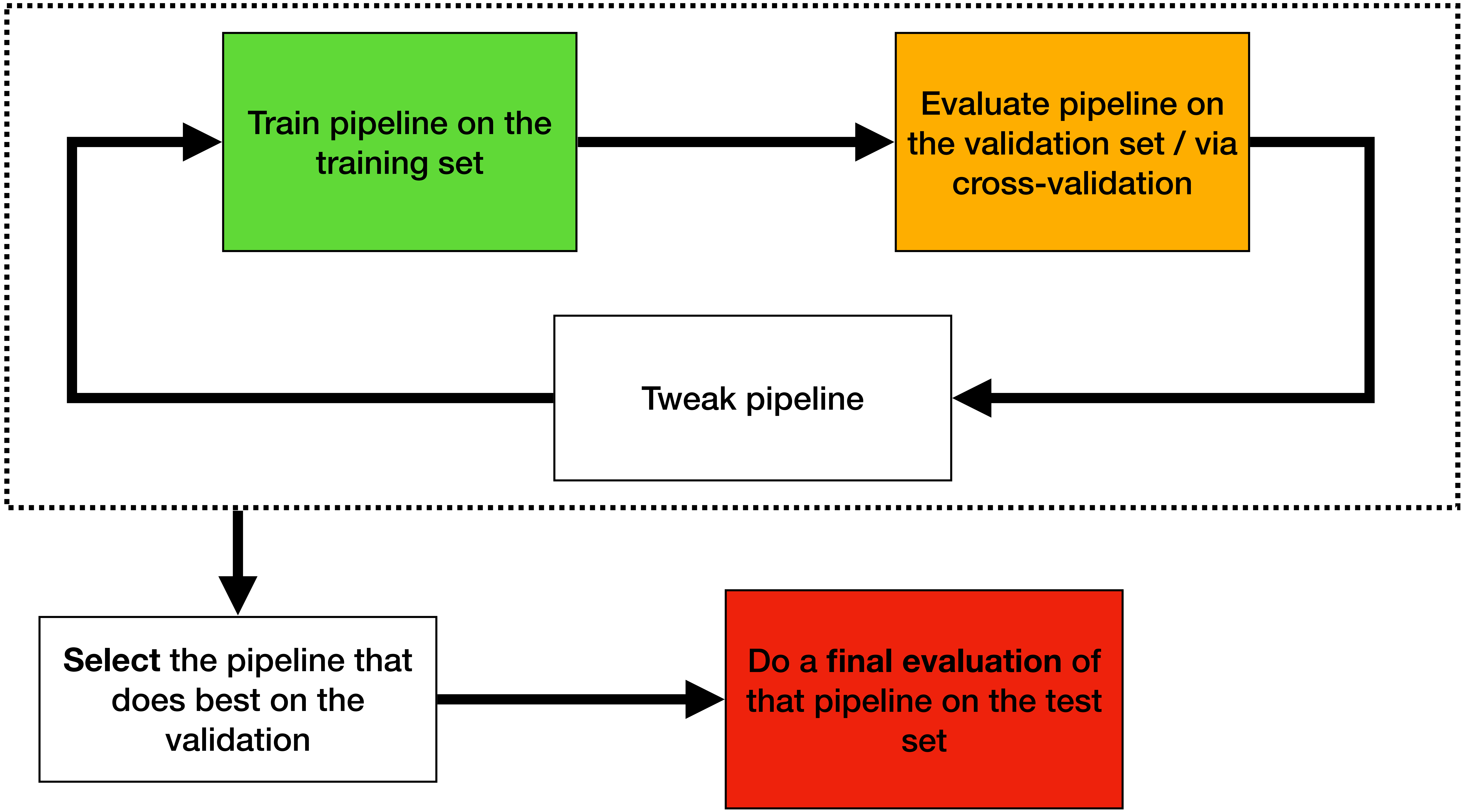
- e.g. above we could grid search across values of α and λ

- We can also compare **across** pipelines, swapping different parts



- This is all model selection so use validation / cross-val to find the best pipeline for your task

A general ML workflow


Train pipeline on the training set
Evaluate pipeline on the validation set / via cross-validation
Tweak pipeline
Select the pipeline that does best on the validation
Do a **final evaluation** of that pipeline on the test set

Inspiration: https://developers.google.com/machine-learning/crash-course/validation/another-partition

Model evaluation

Evaluating binary classifiers

- So far we have used accuracy as the de facto means to evaluate a classifier
- This is simply the fraction of correct predictions (error is 1 minus accuracy)
- If we have a binary classifier and consider class 1 to be the "positive class" and class 0 to be the "negative class" then we can write accuracy as:

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

		Predicted class	
		0	1
True class	0	TN	FP
	1	FN	TP

TP is # true positives
TN is # true negatives
FP is # false positives
FN is # false negatives

Confusion matrices

- A model predicts that a patient with cancer has cancer (TP)
- A model predicts that a patient with cancer doesn't have cancer (FN)
- A model predicts that a patient without cancer has cancer (FP)
- A model predicts that a patient without cancer doesn't have cancer (TN)

A

		Predicted class	
		0	1
True class	0	50	2
	1	10	50

B

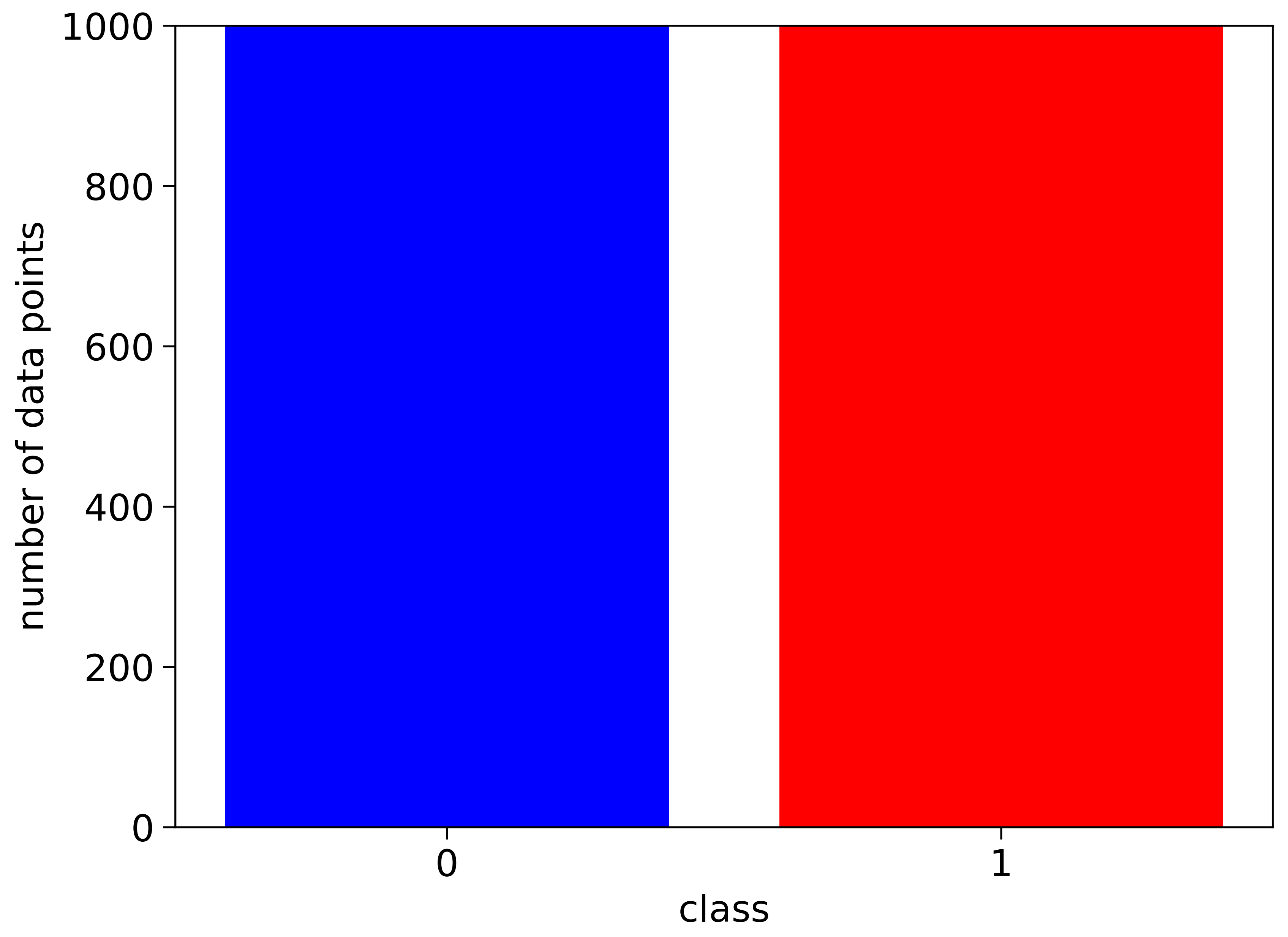
		Predicted class	
		0	1
True class	0	50	10
	1	2	50

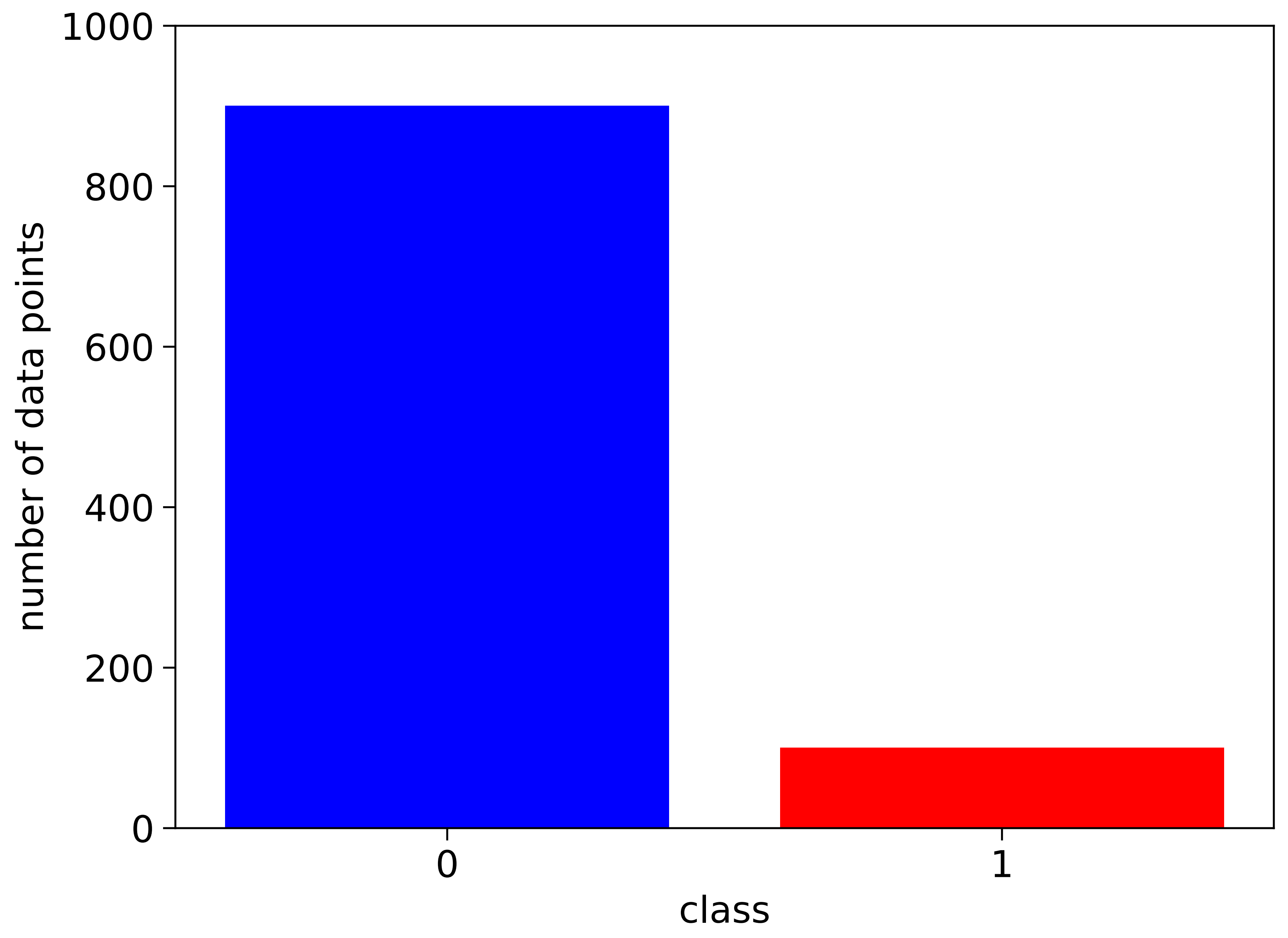
Both these classifiers have the same accuracy

One is much worse…

Class imbalances

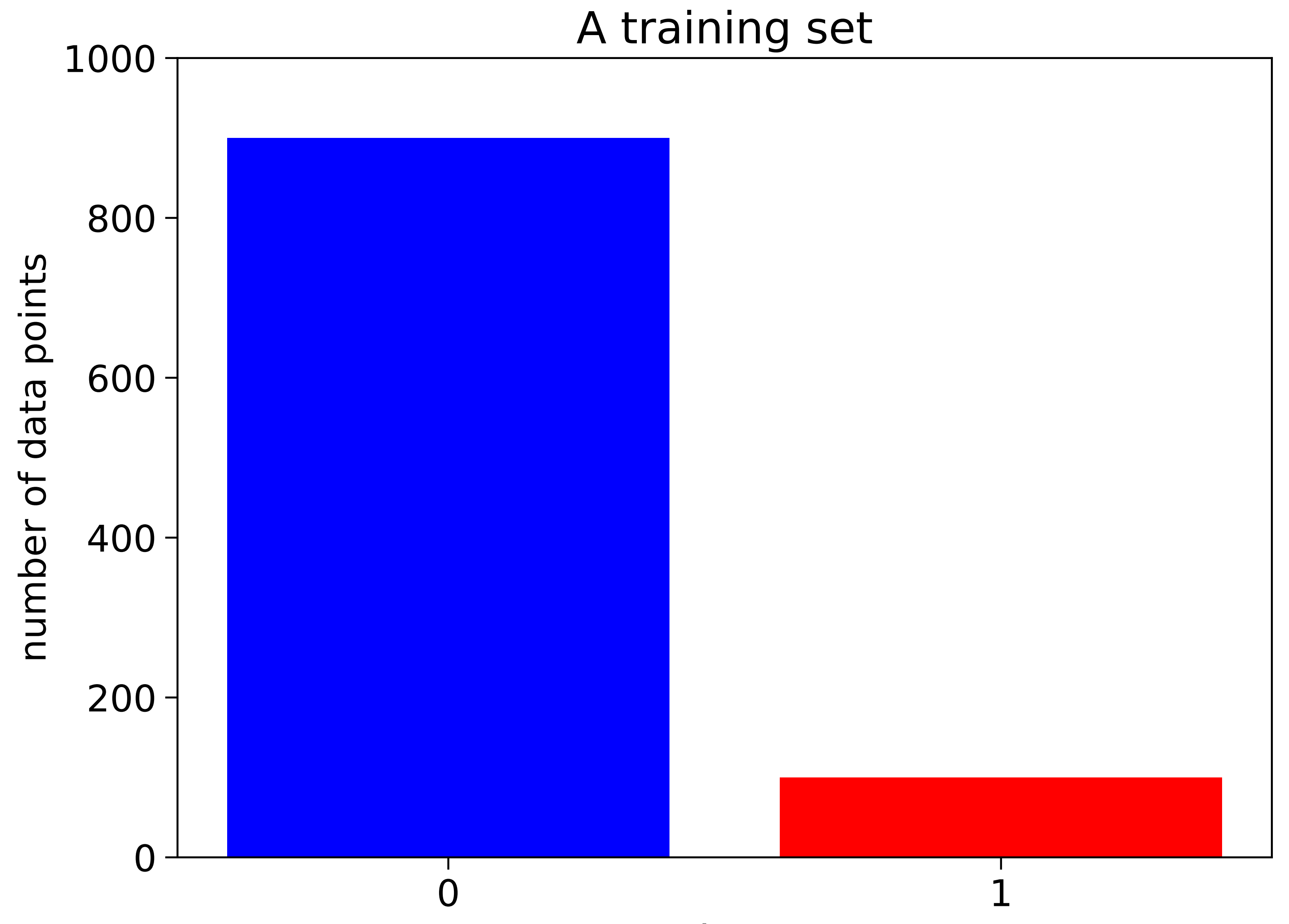
- Most of the datasets we've considered are balanced
- They have similar numbers of examples in each class
- What happens if e.g. class 1 is more rare?





Dummy classifiers

- Consider a training set that is 90% class 0 and 10% class 1
- Now consider a dummy classifier that always predicts class 0
- **It gets 90% accuracy despite having learnt nothing!**
- Always be aware of the dumbest baseline when evaluating models



Accounting for class imbalances

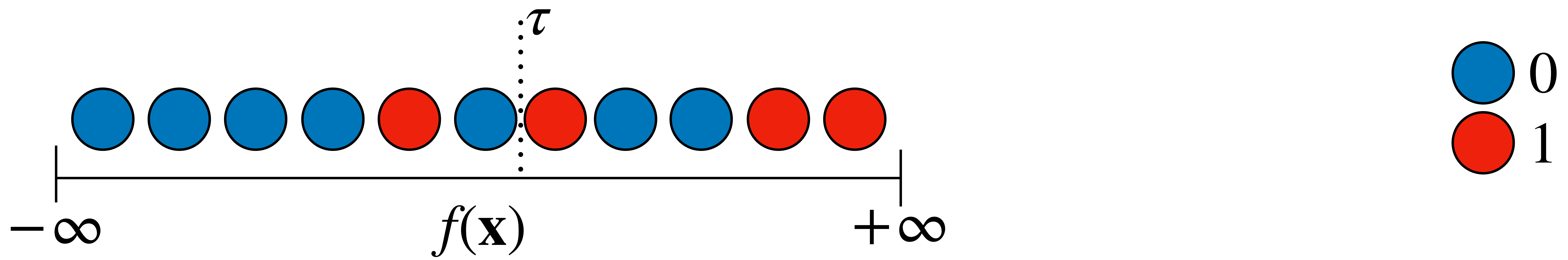
- Class imbalances are part of life
- Imagine diagnosing a rare disease like *lycanthropy*… hardly anyone has it!
- The number of positives P is much less than the number of negatives N
- Let's consider the **true positive rate** $TPR = \frac{TP}{P}$
- Let's also consider the **false positive rate** $FPR = \frac{FP}{N}$
- We want high TPR and low FPR for finding werewolves (and other things)

Receiver operating characteristic

- The predictions of a binary classifier are typically made according to

$$\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) \geq \tau \\ 0 & \text{if } f(\mathbf{x}) < \tau \end{cases}$$

- τ is usually 0 but we can calibrate it
- We need to reduce τ to increase our true positive rate
- **But** we need to increase τ to reduce our false positive rate

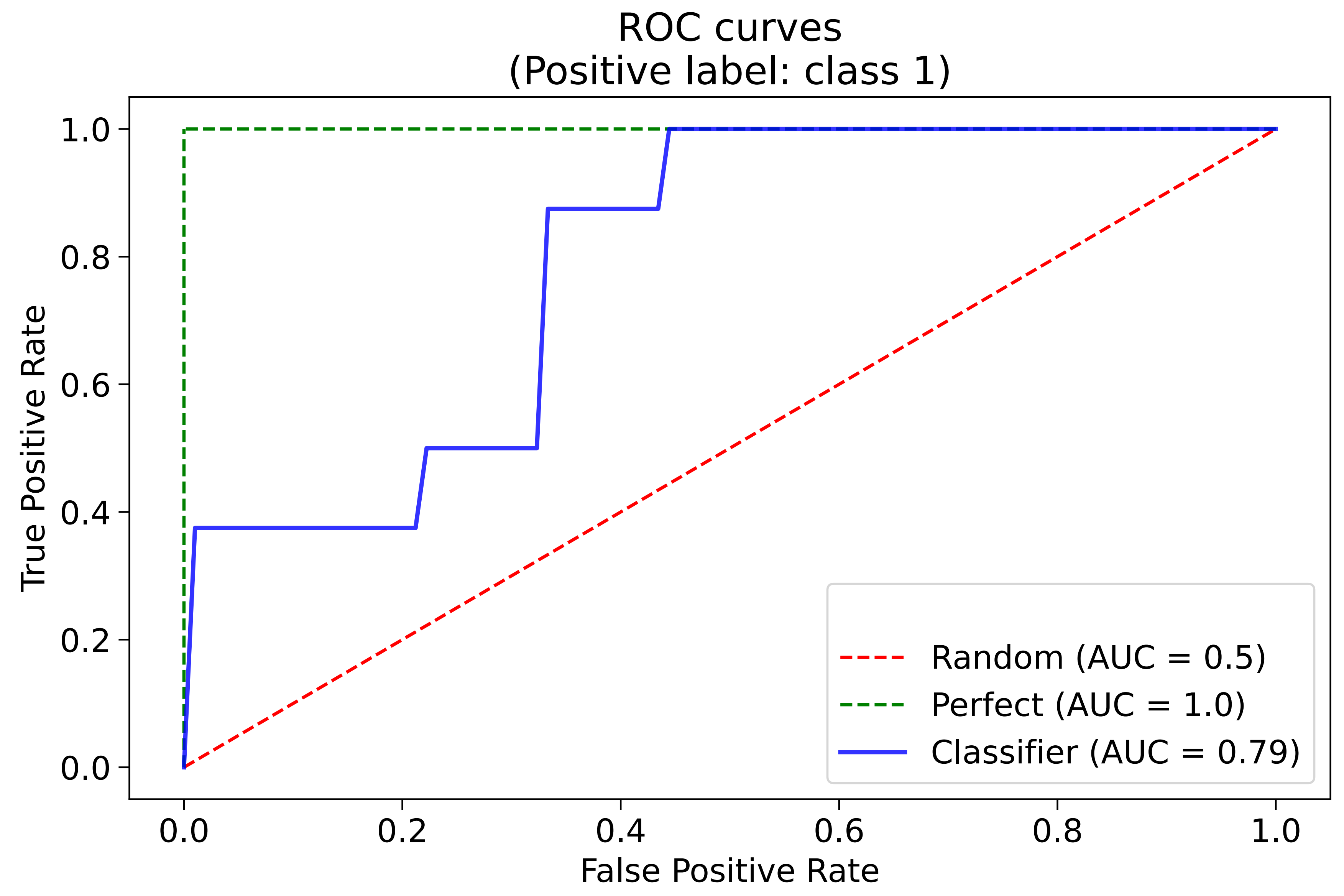


Inspiration for figure: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

Receiver operating characteristic (ROC) curves

- This is a plot of TPR against FPR for different thresholds τ
- A good classifier should hug the top-left corner of this plot
- We can therefore use the area under the curve (AUC) to summarise a classifier's performance when we care about FPR and TPR
- ROC curves are insensitive to class imbalance

A neat interpretation is that the AUC is the probability that given a randomly sampled positive and negative point, the positive point will have the higher classifier score (Credit: Joe Mellor)



Precision and recall for a fixed threshold

- The **recall** ($= TPR$) is the fraction of correctly classified +ve examples for a fixed threshold τ

$$REC_\tau = \frac{TP}{P} = \frac{TP}{TP + FN}$$

- The **precision** is the fraction of examples which were classified as +ve that were classified correctly for a fixed threshold τ

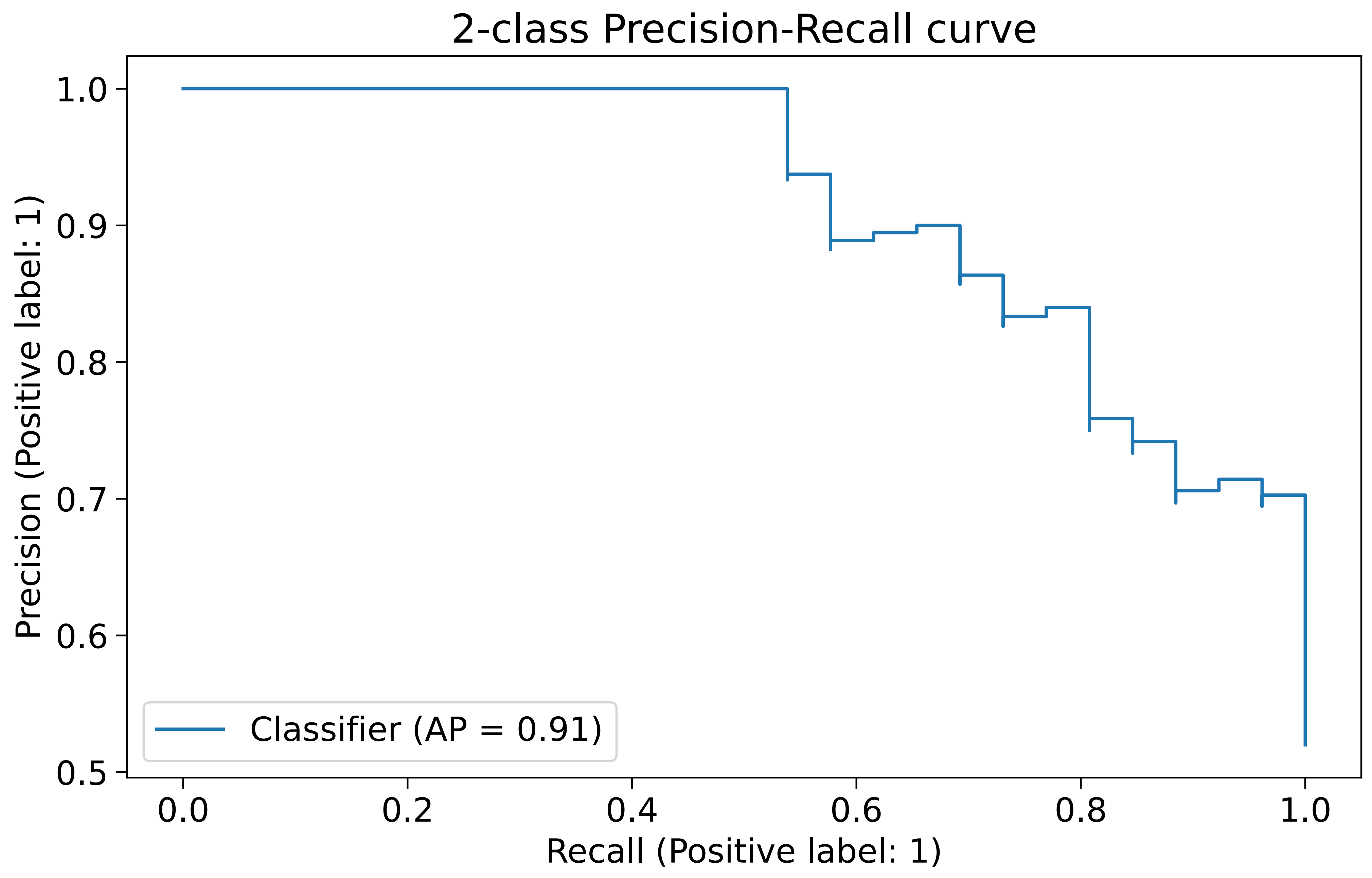
$$PRE_\tau = \frac{TP}{TP + FP}$$

- We want both high recall and precision but there is a balancing act

The $F1$ score combines precision and recall into a single number: $F1_\tau = 2\frac{PRE_\tau \times REC_\tau}{PRE_\tau + REC_\tau}$

PR curves

- Precision and recall can be plotted against each other as we vary τ
- The area under this curve is called **average precision (AP)** and summaries the combined precision-recall profile of a classifier across thresholds



Example from sklearn docs

**PR and ROC curves show roughly the same information.
See https://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf**

Multi-class classification

- We have seen a bunch of ways to score binary classifiers
- What happens if we have classes $0, 1, \ldots, K-1$ with $N_0, N_1, \ldots, N_{K-1}$ samples?
- We can compute a score per class $s_0, s_1, \ldots, s_{K-1}$ with that class as +ve and the rest as -ve and combine those scores in some manner
- The macro-average weights each **class** equally, and the weighted average weights classes according to how many samples there are in that class

$$s_{macro} = \frac{s_0 + s_1 + \ldots + s_{K-1}}{K} \qquad s_{weighted} = \frac{N_0 s_0 + N_1 s_1 + \ldots + N_{K-1} s_{K-1}}{N}$$

Evaluating regression models

- R^2 is the most common score for evaluating regression models
- We aren't going to consider scores other than R^2 and MSE for regression
- Please check out https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics to find out about other scores

Decisions, decisions

- Ultimately, your ML model gives you a **prediction**
- You have to make a **decision** on the basis of that prediction!
- That decision could be “do nothing”

Bayesian decision theory

- We have a binary classifier that predicts $p(y \mid \mathbf{x})$ where $y = 0/1$ is not-cancer/cancer
- Using this prediction, We can make a decision - we can **diagnose** a patient as having cancer $(a = 1)$ or not having cancer $(a = 0)$
- We can create a loss matrix where each element quantifies how **bad** action a is given the true label is y e.g. $\mathbf{L}_{0,1} = L(a = 0 \mid y = 1)$
- The **empirical risk** of taking action 1 is
$R(a = 1 \mid \mathbf{x}) = L(a = 1 \mid y = 1)p(y = 1 \mid \mathbf{x}) + L(a = 1 \mid y = 0)p(y = 0 \mid \mathbf{x})$
- The **empirical risk** of taking action 0 is
$R(a = 0 \mid \mathbf{x}) = L(a = 0 \mid y = 1)p(y = 1 \mid \mathbf{x}) + L(a = 0 \mid y = 0)p(y = 0 \mid \mathbf{x})$
- **Take the action with the least risk!**

$$\begin{array}{cccc} & & \multicolumn{2}{c}{\text{Action } a} \\ & & 0 & 1 \\ & 0 & \boxed{0} & \boxed{1} \\ \text{True label } y & & & \\ & 1 & \boxed{10} & \boxed{0} \end{array}$$

Hypothetical loss matrix $\mathbf{L}$

Bayesian decision theory continued

- If there are K classes and A possible actions then we have loss matrix $\mathbf{L} \in \mathbb{R}^{K \times A}$ and the empirical risk of action i is

$$R(a = i \,|\, \mathbf{x}) = \sum_j L(a = i \,|\, y = j) p(y = j \,|\, \mathbf{x})$$

- But you know how these models work. Can you always trust them?
- Do you really believe that they output reliable probabilities?
- Is this all a bit utilitarian?
- **Be careful**

Summary

- We have looked at the non-parametric k-NN classifier
- We have looked at some generative classifiers
- We have studied the purpose of training, validation, and test splits
- We have considered the problem of model selection
- We have looked at multiple ways to evaluate classifiers
- We saw how to make decisions on the basis of empirical risk