EEN1083/EEN1085 Data analysis and machine learning I

Ali Intizar

Semester 1 2024/2025

Ollscoil Chathair Bhaile Átha Cliath
Dublin City University

Supervised Learning

Ollscoil Chathair Bhaile Átha Cliath
Dublin City University

Outline

Introduction

Machine Learning

What is Machine Learning?

“Machine Learning is a subfield of artificial intelligence which develops algorithms that can learn from training data and then make predictions on new unseen data.”

Why Machine Learning?

- Many real-world problems can’t be solved analytical.
- Rule based programming is not scalable for large scale unseen data.
- Large amount of data, can we train machines to learn from data?

Types of Machine Learning

Supervised Machine Learning

Supervised learning is a machine learning approach that’s defined by its use of labelled data sets. These data sets are designed to train or “supervise” algorithms into classifying data or predicting outcomes accurately. Using labelled inputs and outputs, the model can measure its accuracy and learn over time[1].

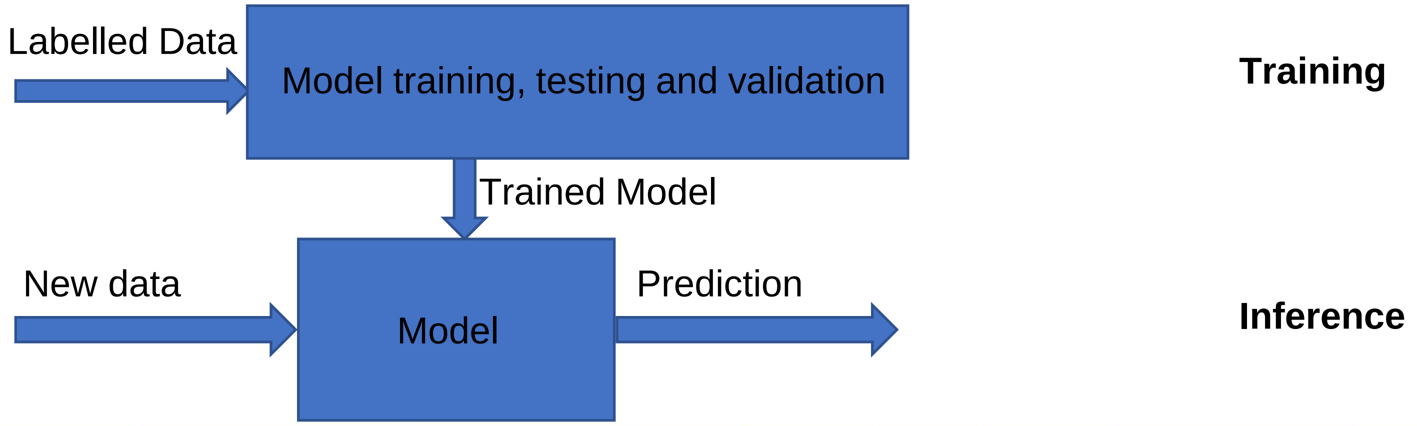
Un-supervised Machine Learning

Unsupervised learning uses machine learning algorithms to analyse and cluster unlabelled data sets. These algorithms discover hidden patterns in data without the need for human intervention (hence, they are “unsupervised”).[1]

https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning

Machine Learning

- Machine Learning develops a model that takes in a new(unseen) data points and predicts a value for output.
- Model learns from training data to be accurate in its predictions.
- Models can be also seen as parametrised functions and machine learning finds best value for parameters.



The supervised learning problem

Given pairs of training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ produce a function $\hat{y} = f(\mathbf{x})$ such that $f(\mathbf{x})$ **generalizes well** to previously unseen data.

- The $\mathbf{x}_i$'s are the features or inputs. Often $\mathbf{x}_i \in \mathrm{R}^D$
- The y_i's are the targets
- The function $f(\mathbf{x})$ is the decision function
- $\hat{y}$ is the prediction of $f(\mathbf{x})$

Will write $\mathbf{x}_i$ to mean training example i and x_j to mean feature j of some training example. x_{ij} is feature j of example i.

Example 1

Training set consists of emails marked as spam or not spam.

- $\mathbf{x}_i$ is the feature vector corresponding to email i.
- x_{ij} is number of occurrences of word j in email i in the training set. E.g. with a vocabulary size of 1000, the $\mathbf{x}_i \in \mathrm{R}^{1000}$.
- The target variable $y_i \in \{0, 1\}$ depending on whether the email is spam or not spam.

Our objective is to use the examples from training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ to come up with a function $f(\mathbf{x})$ that can map a feature representation of an email to $\{0, 1\}$.

We would also like our function to work well on data outside the training set (unseen data).

Example 1

Training set:

doc	class x_1	meeting x_2	data x_3	casino x_4	won x_5	viagra x_6	join x_7	... ...	spam? y
$\mathbf{x}_1$	3	0	3	0	0	0	0	...	0
$\mathbf{x}_2$	0	2	1	0	0	0	0	...	0
$\mathbf{x}_3$	0	0	3	0	0	5	0	...	1
$\mathbf{x}_4$	2	0	0	0	0	0	0	...	0
$\mathbf{x}_5$	1	0	3	2	2	0	2	...	1

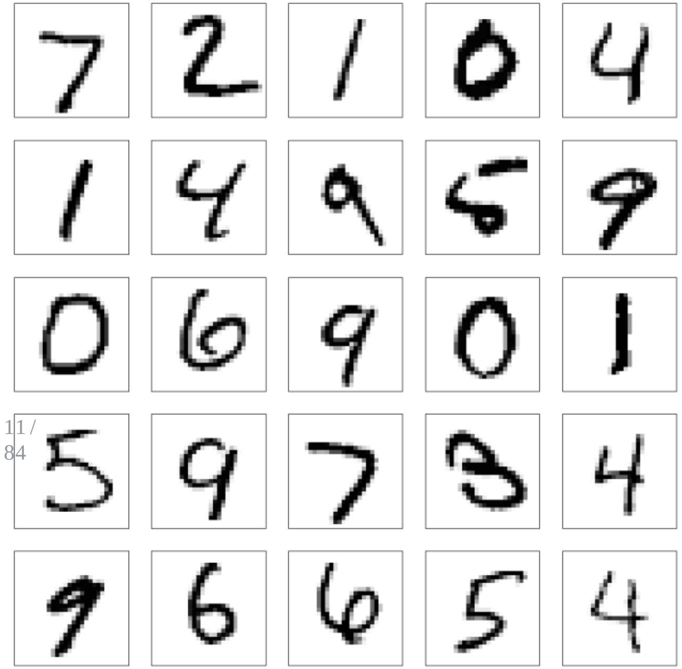
This is an example of a **binary classification** task.

Example 2

Produce a classifier to map from pixels to the digit.

- If images are grayscale and 28×28 pixels in size, then $\mathbf{x}_i \in \mathrm{R}^{784}$
- $y_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Example of a **multi-class classification** task.



Example 3

Features $\mathbf{x}$ could be:

- Age
- Level of education (secondary school, degree, Ph.D)
- Years of experience
- Industry

Target variable y is salary. E.g. $y_i = 50000$ EUR

Regression problem: target y value is continuous.

$$f : \mathbb{R}^D \to \mathbb{R}$$

More examples

- feature is time of day, target is light level
- feature is light level, target is time of day
- features are measurements from sensors (temp, humidity, brightness, etc.), target is $\{\text{rain}, \text{no rain}\}$
- features are vectors of image pixels, target is $\{\text{cat}, \text{dog}, \text{car}, \text{person}, \ldots\}$
- features are recorded audio fragments, targets are words

Terminology

The training set: $T = \{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in X, \, y_i \in Y\}$

- $\mathbf{x}_i$ are the inputs. Often vectors of $D-$dimensional real numbers $\mathbf{x}_i \in \mathrm{R}^D$
- y_i is the output or target variable.
- $(\mathbf{x}_i, y_i)$ is a training example.
- X is the domain (set)
- Y is the range (set)

E.g.

$$\mathbf{x}_1 = [\ 1\ \ 3\ \ \ 0\ \ \ 2\ \ \ 8\ \ 6\ ]^T$$

$$y_1 = 1$$

Terminology

$$\mathbf{x}_1 = [\,1 \;\; 3 \;\; 0 \;\; 2 \;\; 8 \;\; 6\,]^T$$

$$y_1 = 1$$

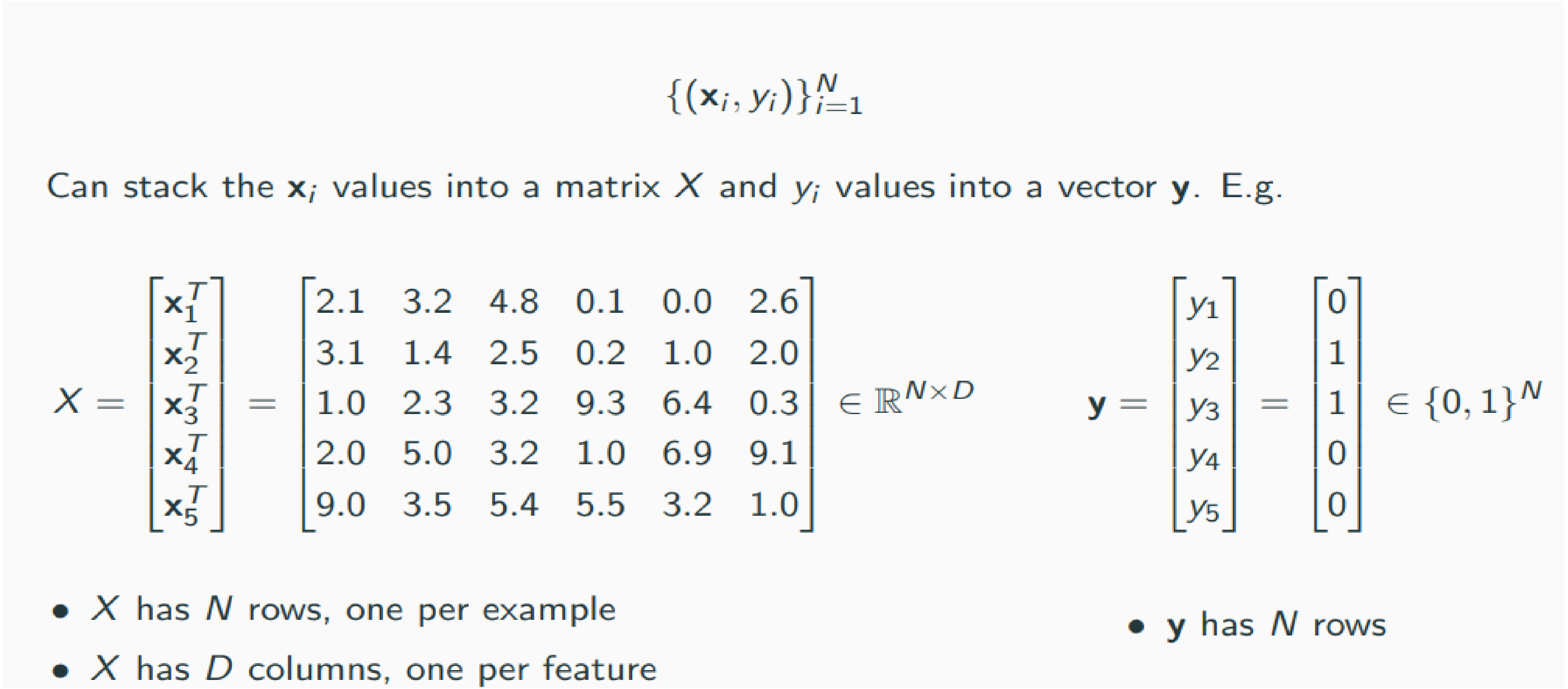
The x_j values are known as:

- Features
- Variates
- Attributes
- Predictors

The y values are known as

- Targets
- Outputs
- Covariates

Terminology



$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

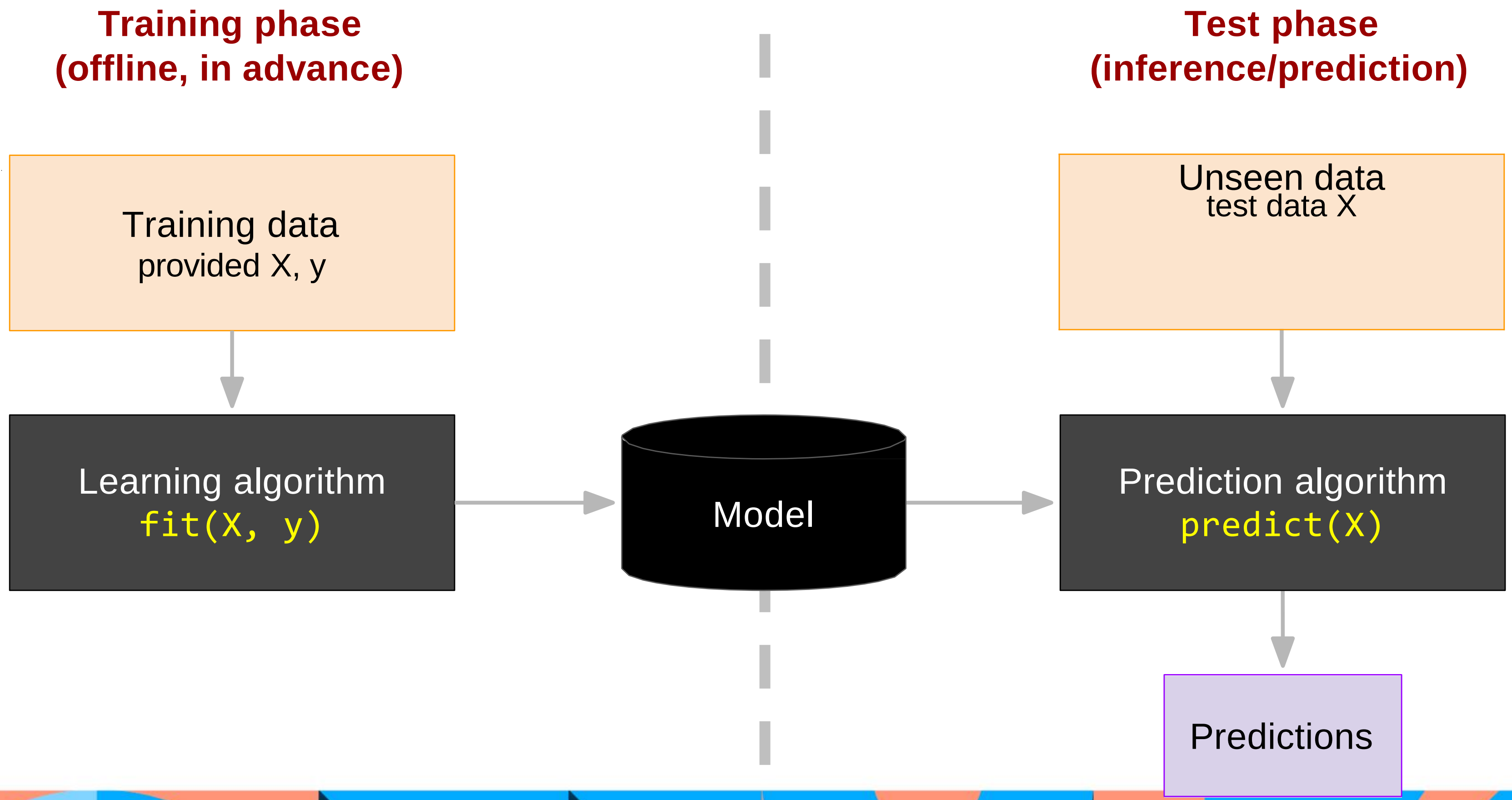
Can stack the $\mathbf{x}_i$ values into a matrix X and y_i values into a vector $\mathbf{y}$. E.g.

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \hline \mathbf{x}_2^T \\ \hline \mathbf{x}_3^T \\ \hline \mathbf{x}_4^T \\ \hline \mathbf{x}_5^T \end{bmatrix} = \begin{bmatrix} 2.1 & 3.2 & 4.8 & 0.1 & 0.0 & 2.6 \\ 3.1 & 1.4 & 2.5 & 0.2 & 1.0 & 2.0 \\ 1.0 & 2.3 & 3.2 & 9.3 & 6.4 & 0.3 \\ 2.0 & 5.0 & 3.2 & 1.0 & 6.9 & 9.1 \\ 9.0 & 3.5 & 5.4 & 5.5 & 3.2 & 1.0 \end{bmatrix} \in \mathbb{R}^{N \times D} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \hline y_2 \\ \hline y_3 \\ \hline y_4 \\ \hline y_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \in \{0, 1\}^N$$

- X has N rows, one per example
- X has D columns, one per feature

- $\mathbf{y}$ has N rows

Black box abstraction of supervised learning



Two types of problems

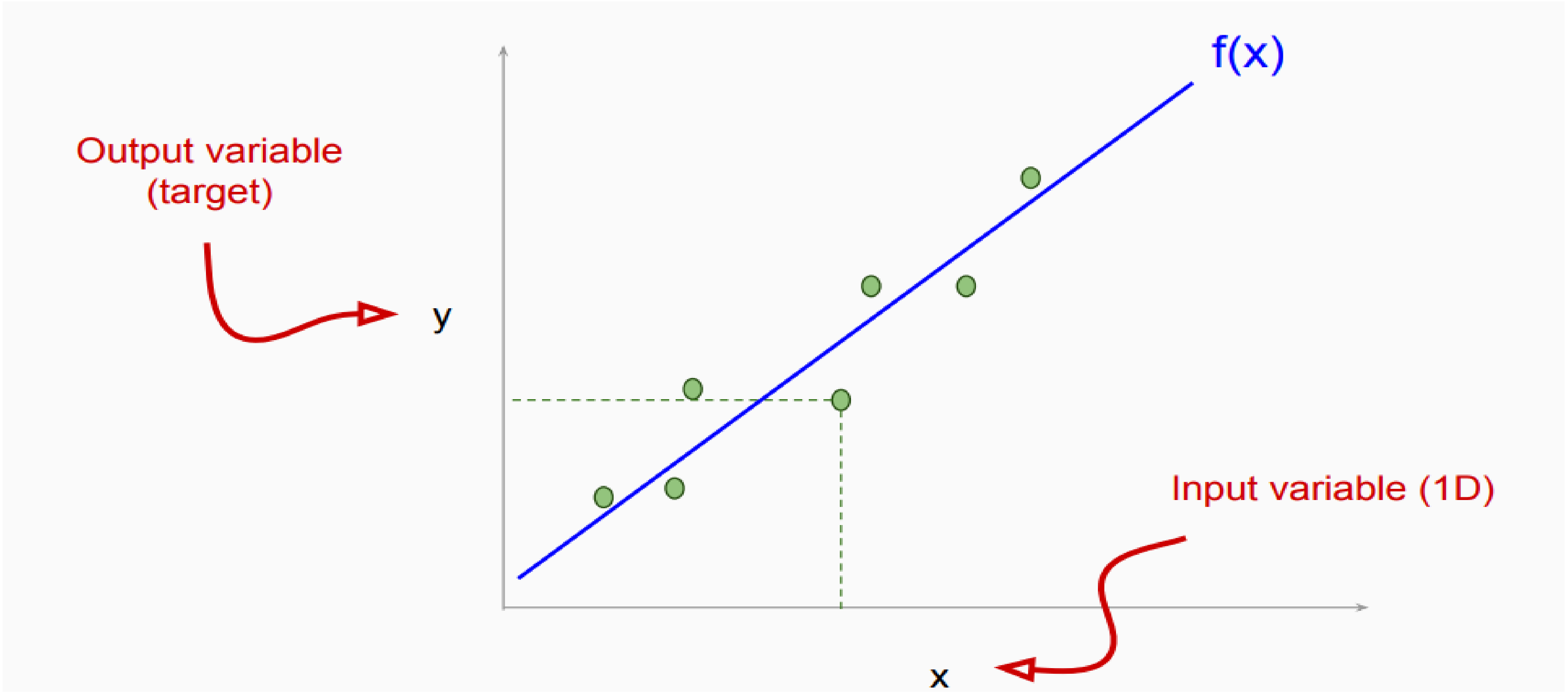
Regression: when the output variable y is a real number ($y \in \mathrm{R}$)

- Predicting house price from location, number of bedrooms, etc.
- Predicting movie earnings from user reviews
- Predicting multiple real numbers (multivariate regression $\mathbf{y} \in \mathrm{R}^N$)

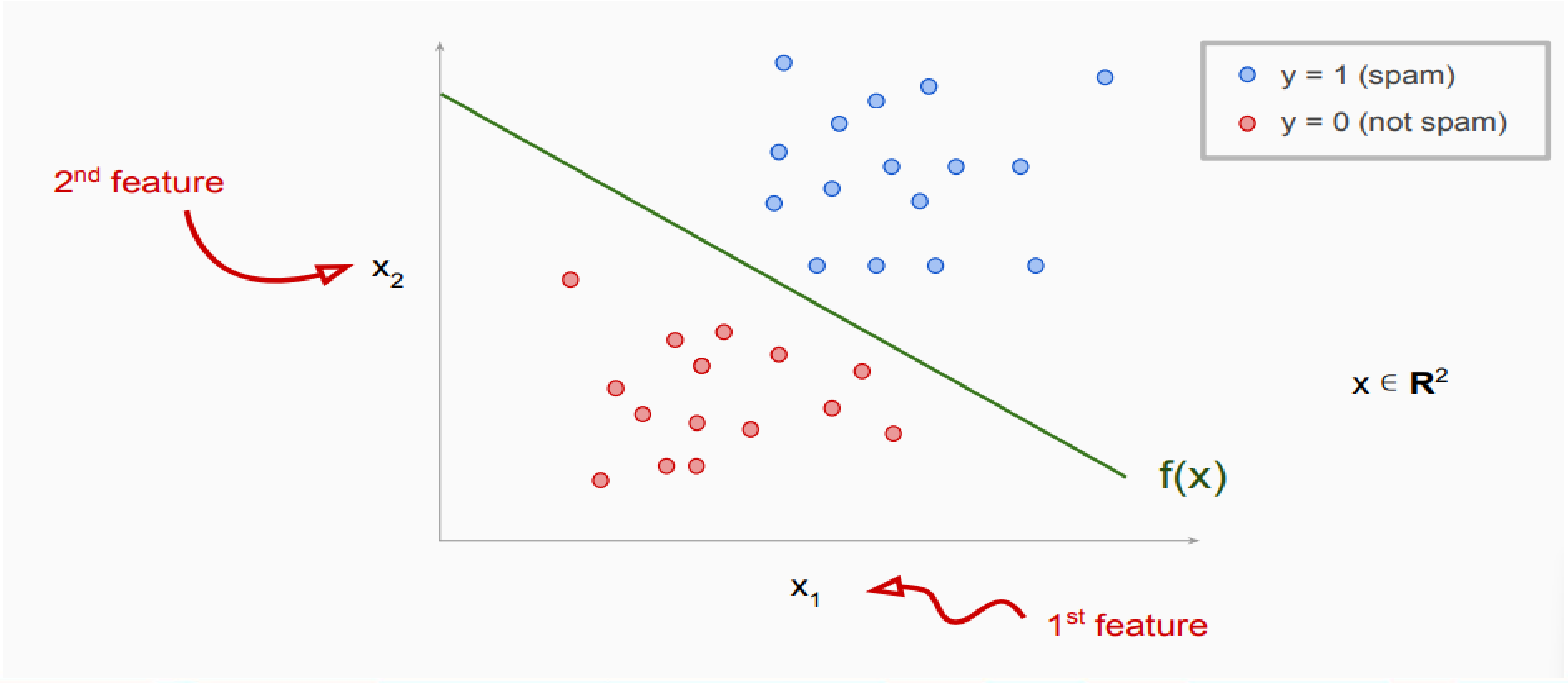
Classification: when the output variable y is a categorical variable

- Is an email spam or non spam (binary classification $y \in \{0, 1\}$)
- Is this an image of a $\{car, house, cat, dog, person\}$ (multiclass classification)
- Is the sentiment of this text $\{positive, negative\}$

Regression



Classification



Measuring error

Given some trained black box predictor $f(\mathbf{x})$ (classifier or regressor) trained on T, we would like some way to measure how accurate it is.

Two things we need:

- An error or accuracy **metric**
- A **dataset** to measure error against.

There are many metrics we can use. Some suitable for regression and others for classification.

Measuring error

Want to measure the distance between the prediction $\hat{y} = f(\mathbf{x})$ and the target y.

Standard metric is **mean squared error** (MSE) (aka Euclidean error).

Squared error for example i:

$$(y - f(\mathbf{x}))^2 = (y - \hat{y})^2$$

MSE for a dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$

$$E = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Measuring error

MSE not appropriate for classification, since y is a categorical variable.

One way to measure error is the **misclassification rate**:

$$E = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i)$$

where $\mathbb{1}(.)$ is the indicator function.

Note: accuracy is (1 - misclassification rate).

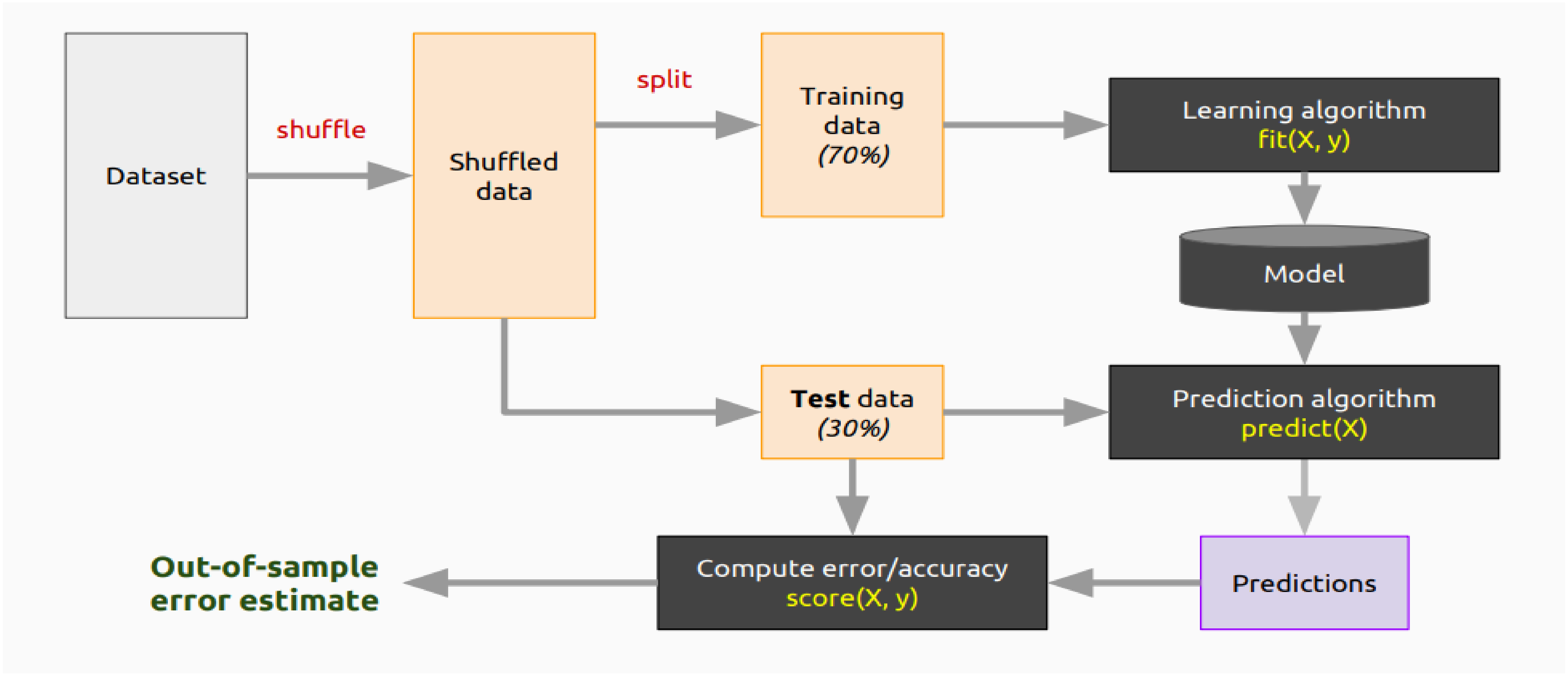
Measuring error

1. **Training error**: error on training set T
2. **Out-of-sample (OOS) error**: expected error on unseen data
3. **Test error**: error on a sample of unseen data (estimate of out of sample error)
4. **Generalization error**: (OOS error - training error)

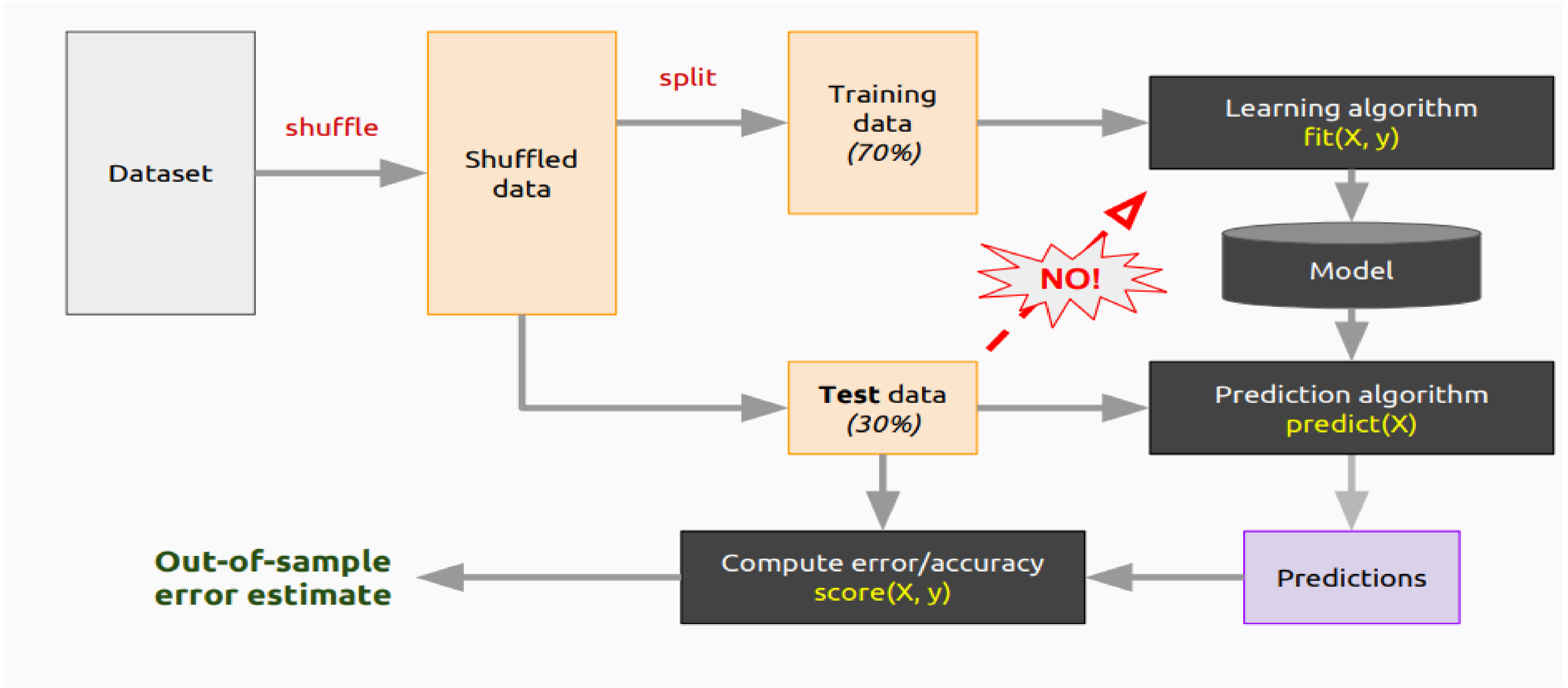
The objective of supervised learning is to **minimize OOS error**. Cannot measure this directly.

Essential to keep aside some test data to estimate the accuracy of your model on unseen data. Can use test error as a proxy for OOS error.

Train test splits


Dataset
shuffle
Shuffled data
split
Training data (70%)
Learning algorithm fit(X, y)
Model
Test data (30%)
Prediction algorithm predict(X)
Predictions
Compute error/accuracy score(X, y)
Out-of-sample error estimate

Train test splits


Dataset
shuffle
Shuffled data
split
Training data (70%)
Learning algorithm fit(X, y)
Model
NO!
Test data (30%)
Prediction algorithm predict(X)
Predictions
Compute error/accuracy score(X, y)
Out-of-sample error estimate

Train test splits

Why can we not just estimate accuracy on using the training set?

- It is easy to get 0 error on the training set.
- How?

Train test splits

Why can we not just estimate accuracy on using the training set?

- It is easy to get 0 error on the training set.
- How? **Just memorize the whole training set.**

Why is it important to shuffle the training data before splitting?

- Ensure the training set is **unbiased**
- Exception: time series data. (Why?)

How much test data do we need?

- Enough so that the estimate of OOS error is accurate.
- Central limit theorem. Variance of estimate goes to zero as $n \to \infty$
- Trade-off: want to keep as much data for training but still have enough to have low variance estimate of OOS.
- Rule of thumb: keep $10 - 30\%$ of data for test.

Why not use some of the test data to train?

- You have no way to estimate OOS error if you do
- Your model could easily **overfit** the test data and have poor generalization, you have no way of knowing without test data
- Model may fail in production

Best practices

Split your dataset into train and test at the very start

- Usually good practice to shuffle data (exception: time series)

Do not look at test data (data snooping)!

- Lock it away at the start to prevent contamination

NB: Never ever train on the test data!

- You have no way to estimate error if you do
- Model may fail in production

Nearest neighbours

Nearest neighbours

Simplest possible algorithm for “learning” from data. Learning by rote memorization.

Idea: simply memorize all the data.

Prediction: Find the **nearest** example in the dataset and simply output the memorized target value for that example.

Nearest neighbour classification

Learning algorithm (.fit)

Memorize the entire training set

$T = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$

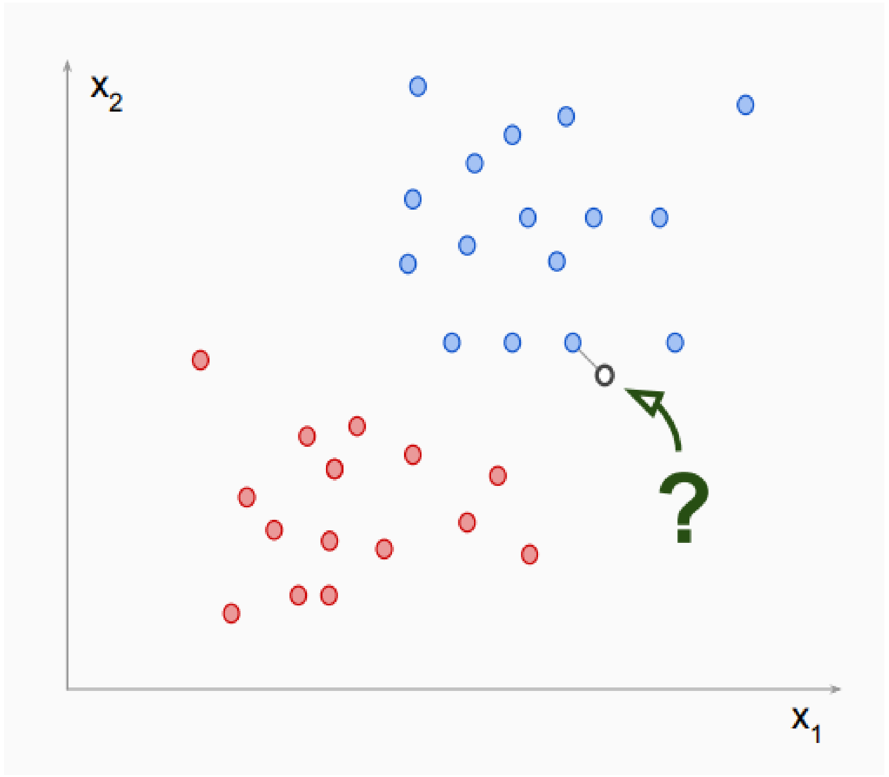
Prediction algorithm (.predict)

Find nearest $\mathbf{x}_i$ in training set and output the corresponding y value

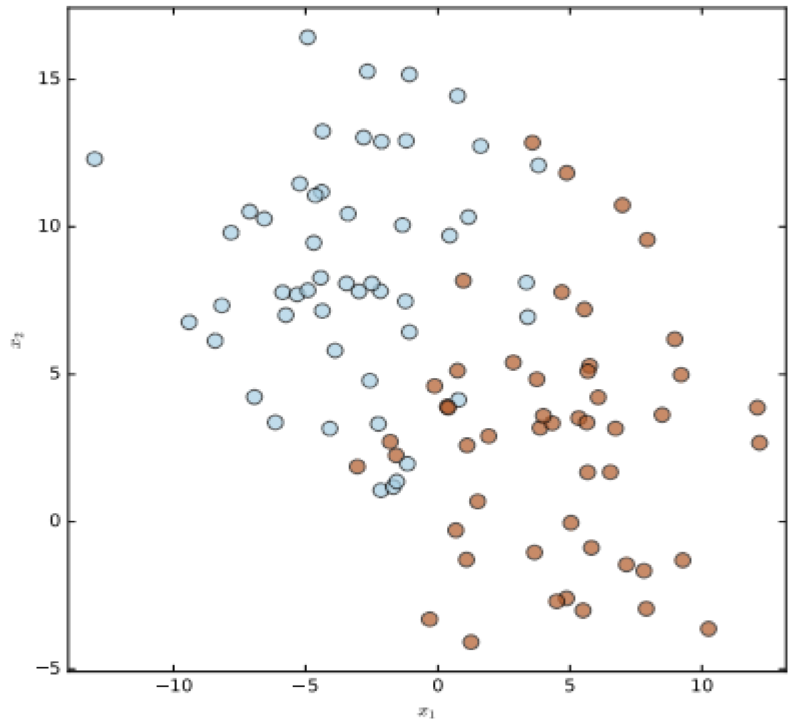
$$f(\mathbf{x}) = \underset{y_i \in T}{\arg\min}\, D(\mathbf{x}, \mathbf{x}_i)$$

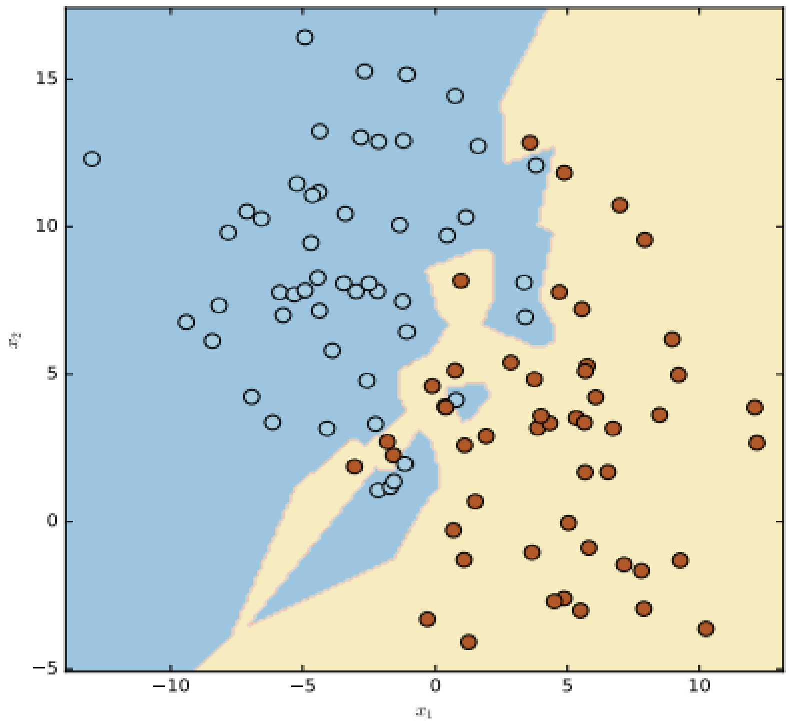
where D is a distance measure. E.g. Euclidean distance:

$$D(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_2$$

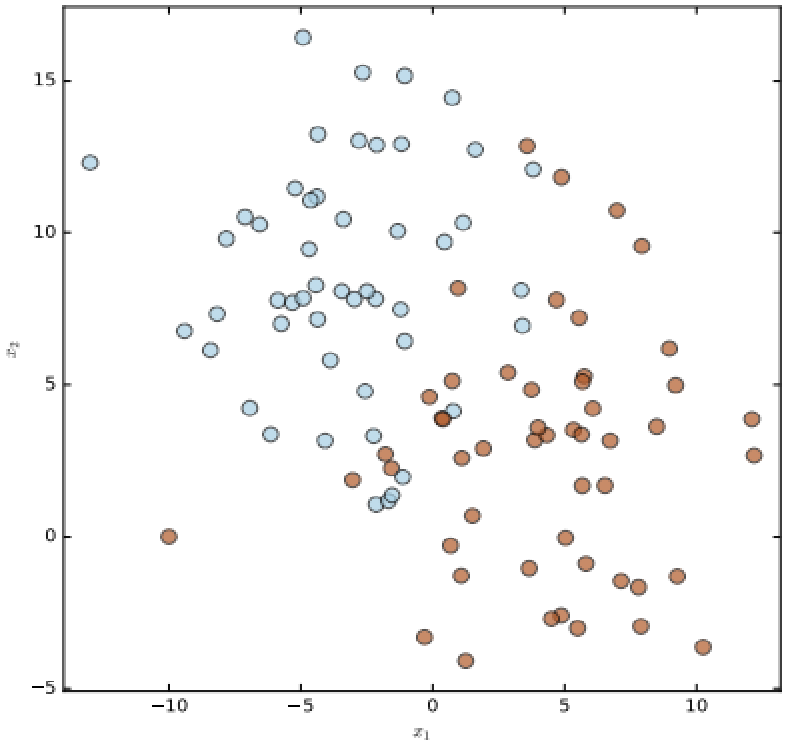


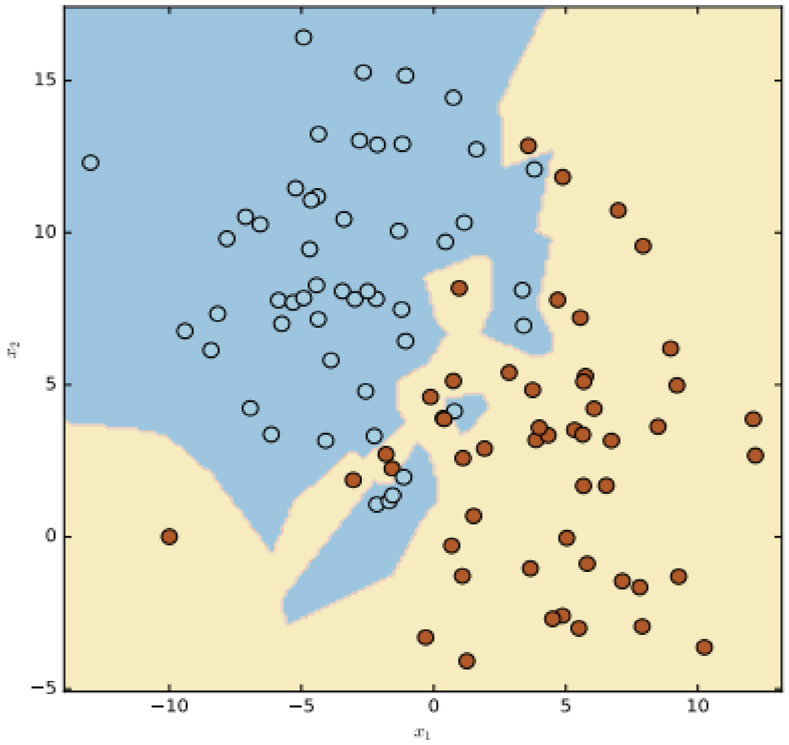
Nearest neighbour classification





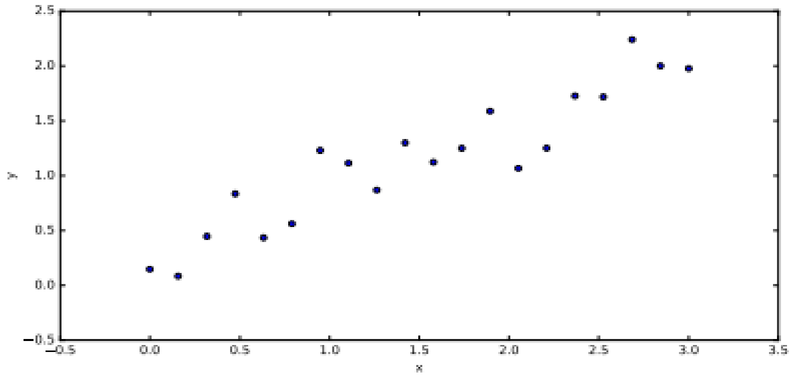
Nearest neighbour classification

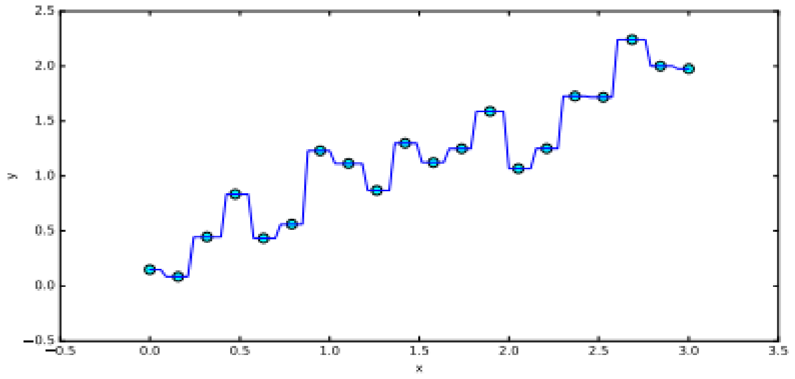




Nearest neighbour regression

- Data generated from $y = 0.8x + \text{noise}$
- Nearest neighbour regression fits data exactly.
- Fit function is very jagged
- Q: generalization outside of range of training data?





Advantages of nearest neighbours

Very fast learning algorithm!

- Just store all the training examples.
- Implementation is trivial.

Low classifier **bias**.

- Is able to always fit the training data exactly.

Can use any distance (or similarity) function. E.g. for bag of words representations of text documents, we can use *cosine similarity*.

$$\cos\theta = \frac{\mathbf{x}^T\mathbf{z}}{||\mathbf{x}||\,||\mathbf{z}||}$$

Problems with nearest neighbours

High **variance**

- Prediction function not robust to noise
- Small amount of noise can cause big changes to prediction function

Prone to **overfitting**:

- Always fits training examples exactly
- Fits any errors in the training set

Need to choose an appropriate distance measure:

- How to choose a good one for natural images? Audio?

Problems with nearest neighbours

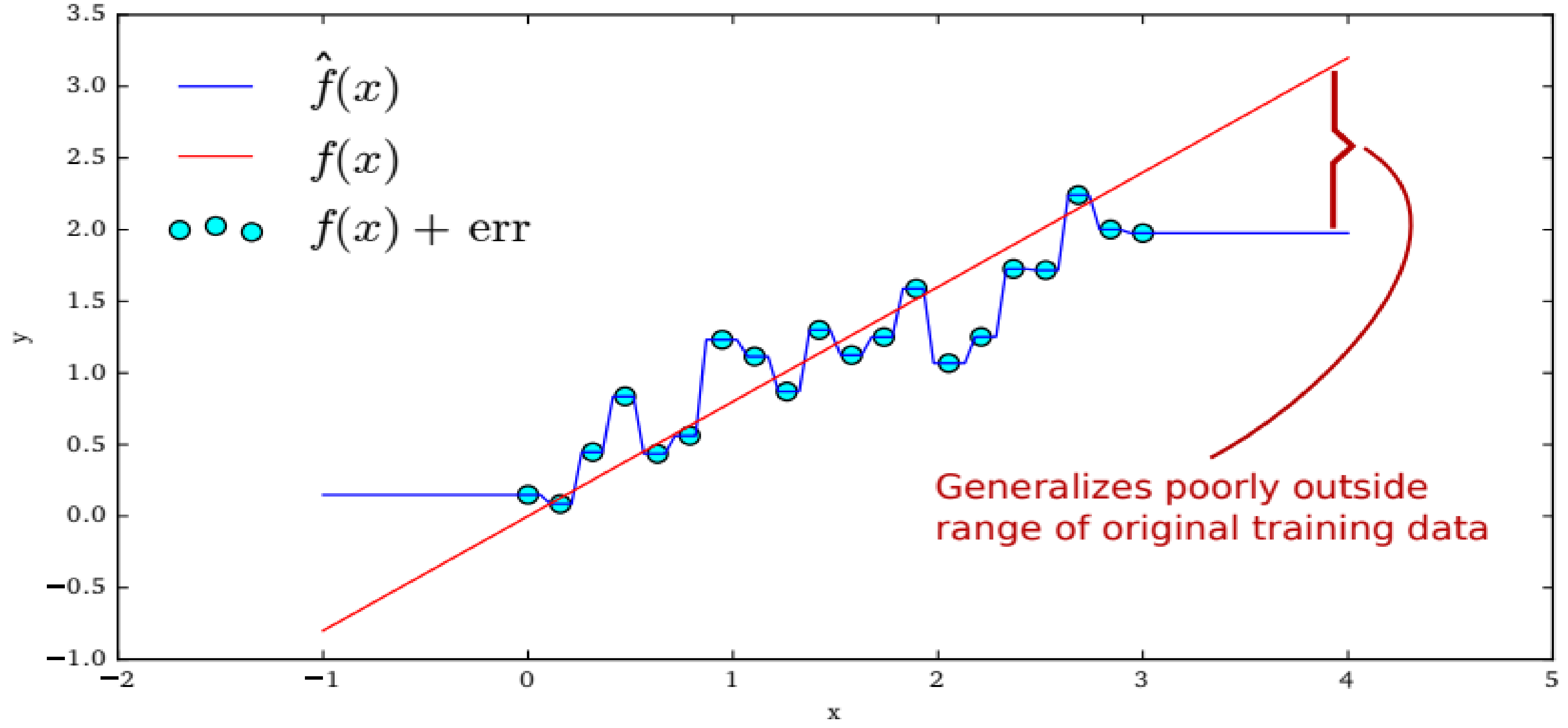
Need to memorize entire training set:

- Memory requirements grow with number of training examples.
- Predict time grows with number of training examples.

Problems with **generalization**

- How to classify examples that are far from what you have seen?
- In high dimensions, your nearest neighbour may be very different! (*curse of dimensionality*)

Nearest neighbours generalisation



Bias and variance

Sources of error

We’ve seen that the nearest neighbour predictors have **low bias** but **high variance**.

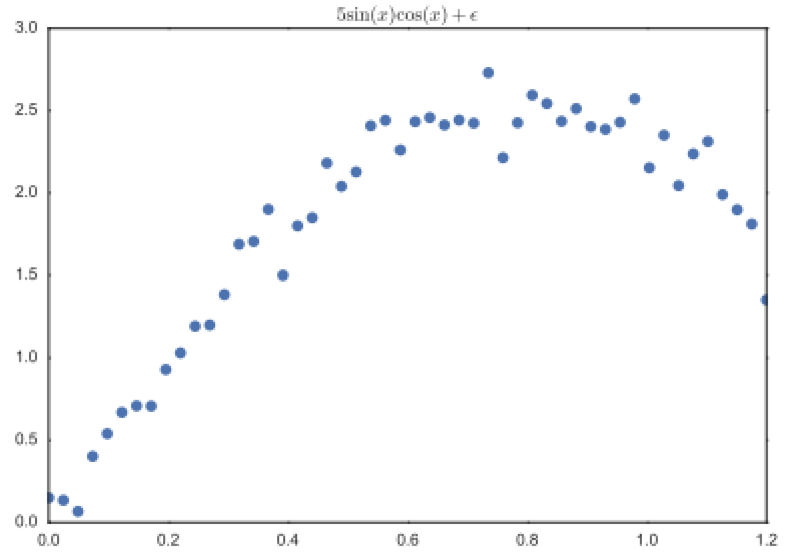
Also seen that nearest neighbours are susceptible to **overfitting**: fitting the **noise**.

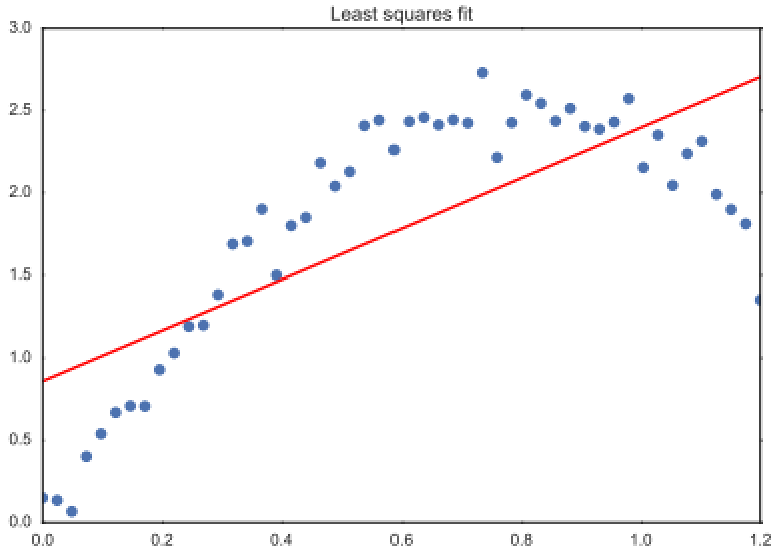
We will show that it is possible to **decompose** all sources of error into three components: bias, variance, and unavoidable noise.

This is called the **bias-variance** decomposition.

What is bias?

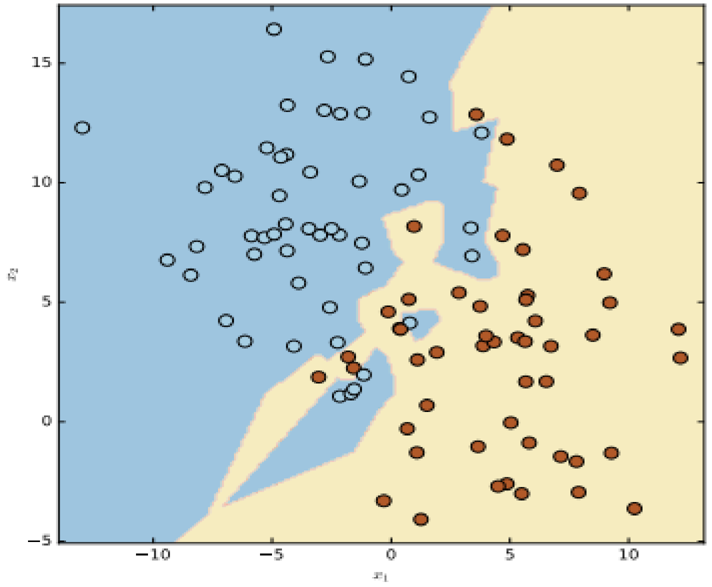
Error due to model rigidity: model not flexible enough to account for observations (underfitting). Often due to incorrect assumptions.

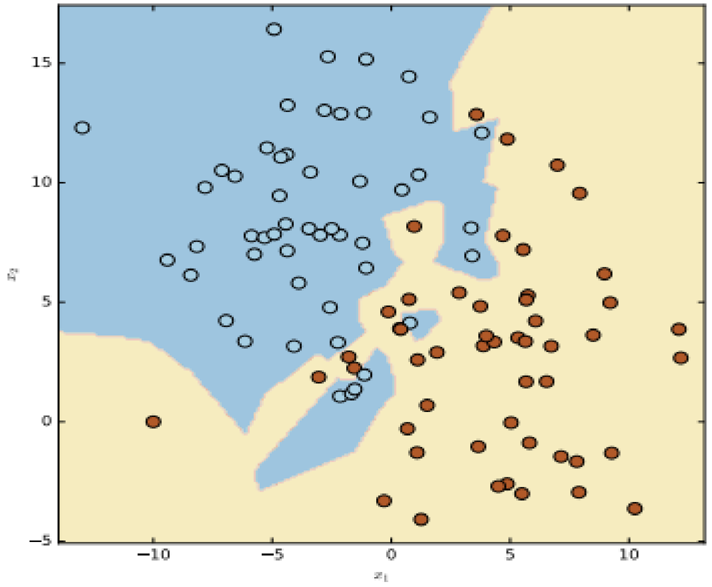




What is variance?

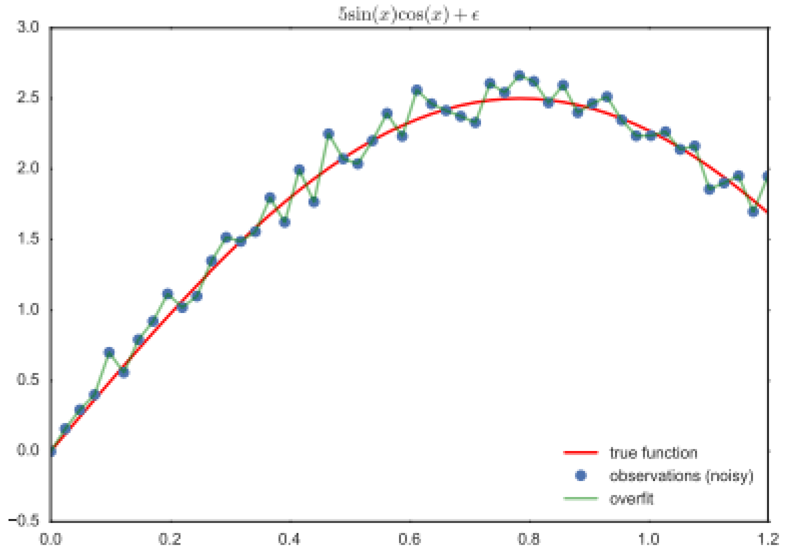
Error due to our model being sensitive to small perturbations in the training data (overfitting). Small changes in training data produce large changes in decision function.





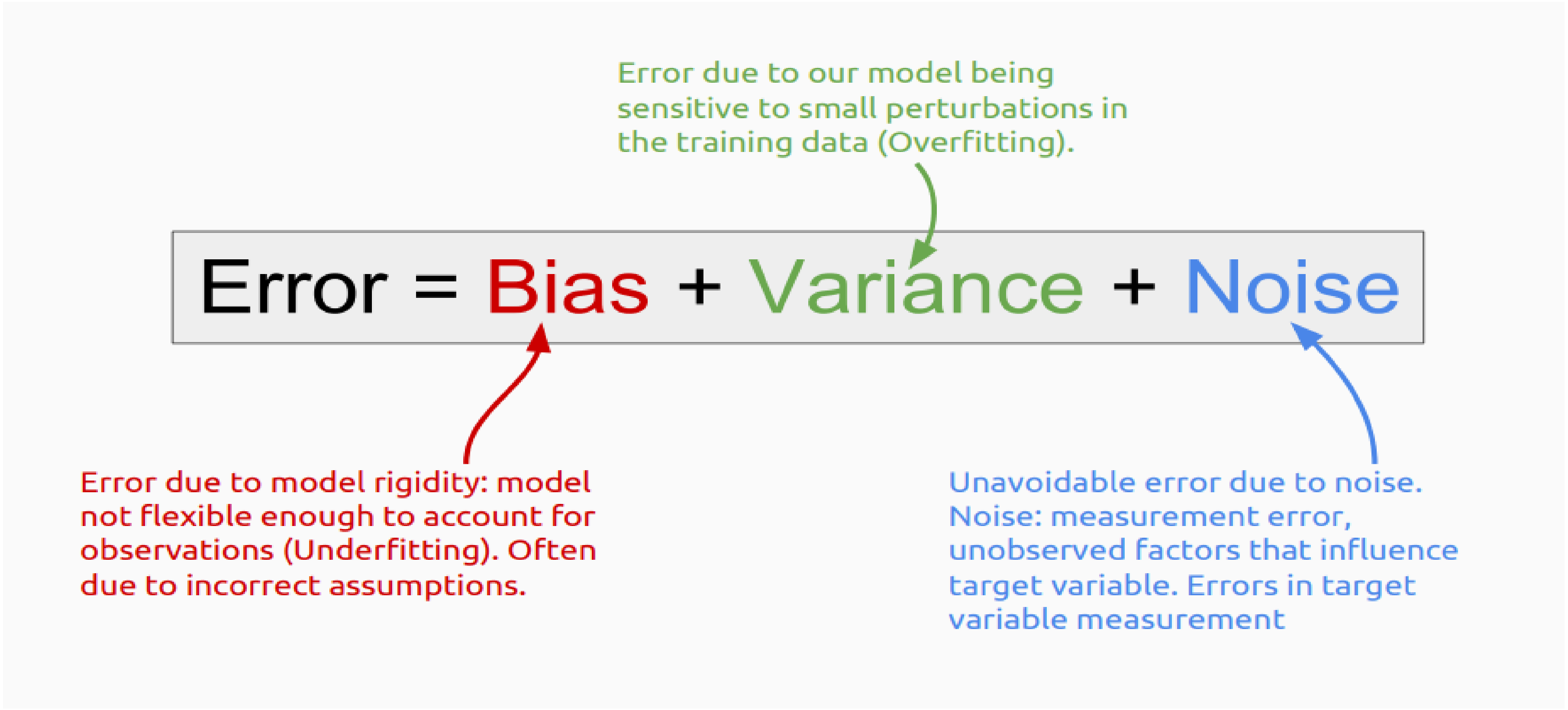
What is noise?

Unavoidable error. Noise: measurement error, unobserved factors that influence target variable. Errors in target variable measurement.



Fitting this kind of unavoidable error is also overfitting.

The bias-variance decomposition



Error due to our model being sensitive to small perturbations in the training data (Overfitting).

Error = Bias + Variance + Noise

Error due to model rigidity: model not flexible enough to account for observations (Underfitting). Often due to incorrect assumptions.

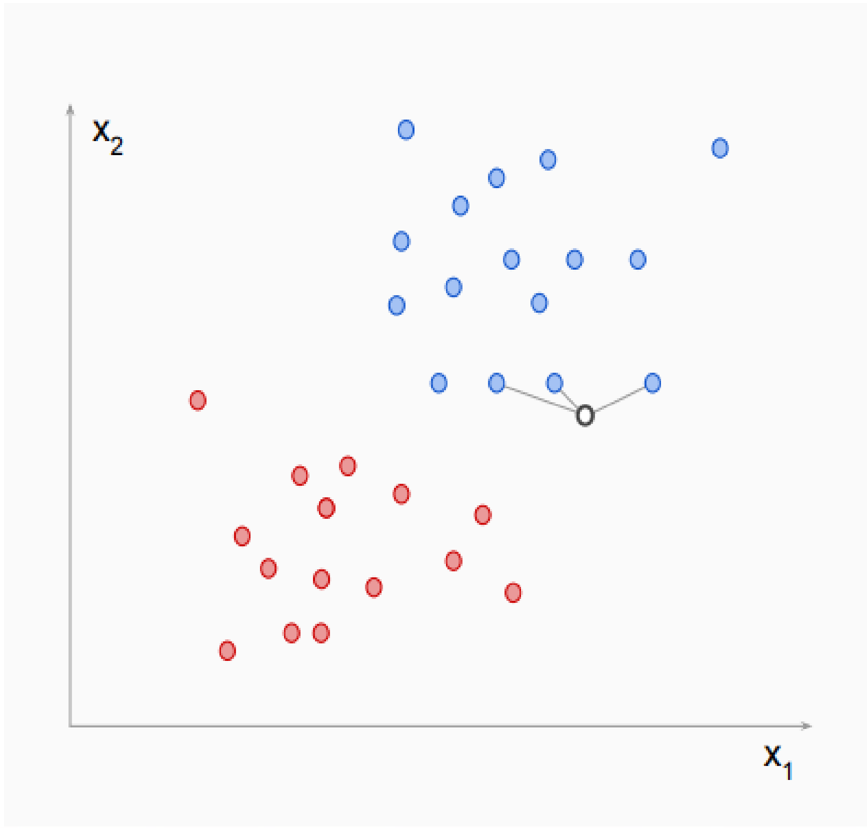
Unavoidable error due to noise. Noise: measurement error, unobserved factors that influence target variable. Errors in target variable measurement

K nearest neighbours

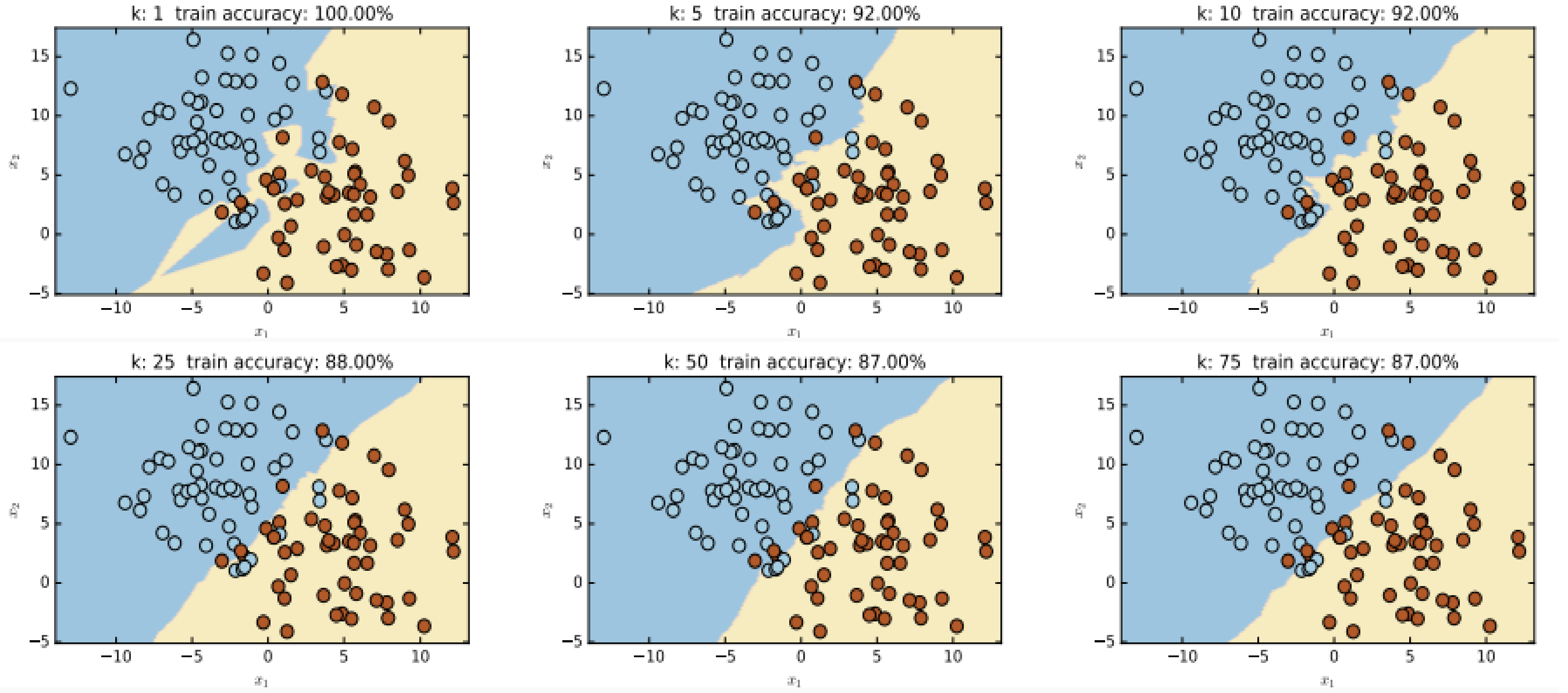
Regression: average y values over the K nearest neighbours

Classification: vote for the correct class with the K nearest neighbours

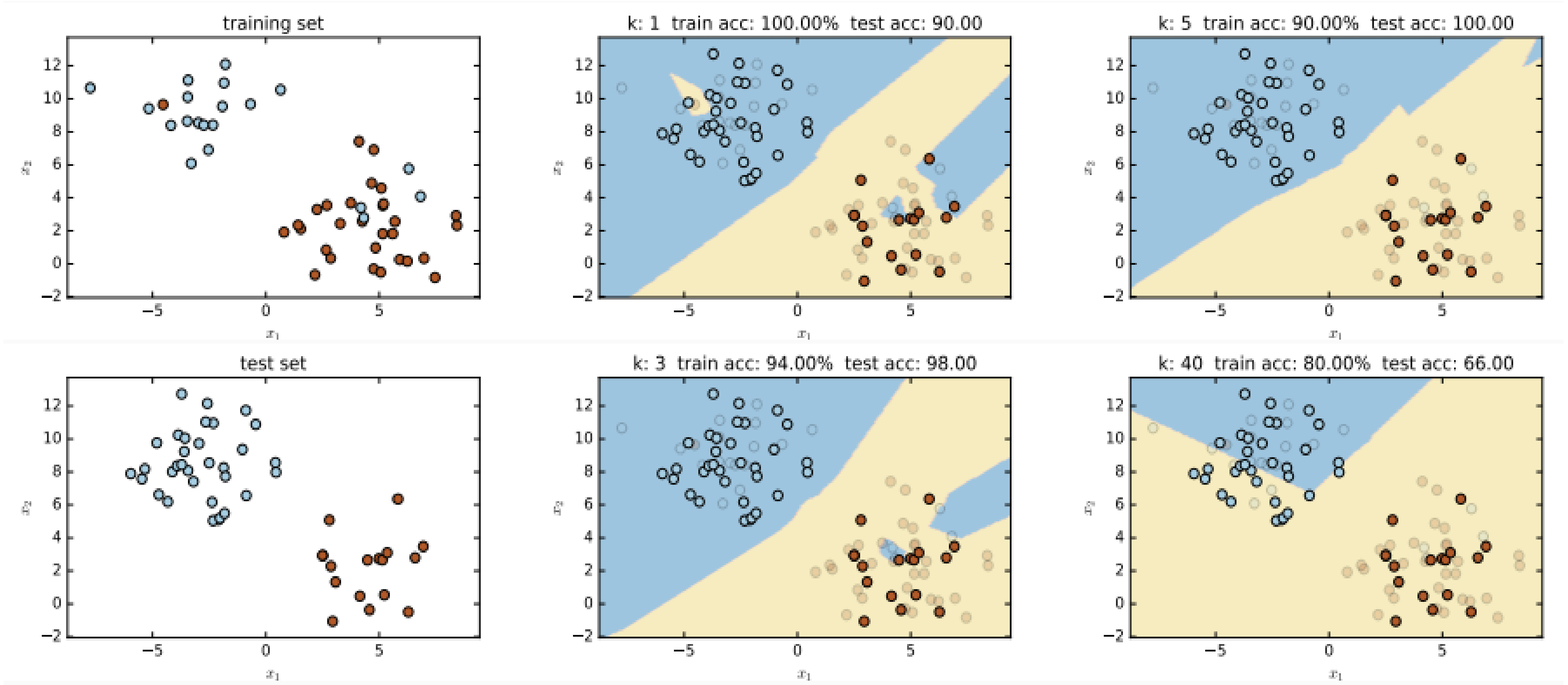
Individual training examples (and noise) now have a much smaller influence on the final model. Noise more likely to be "averaged out" by other neighbours.



K nearest neighbours

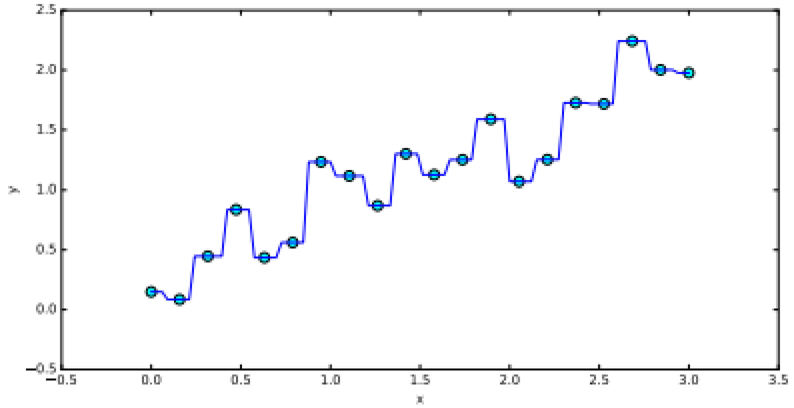


K nearest neighbours

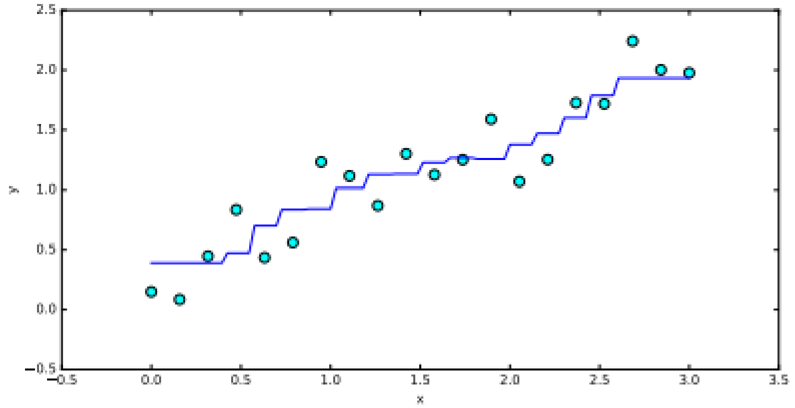


K nearest neighbours

k = 1



k = 5



K nearest neighbours

Larger K reduces the effect of noise on the final predictor: it **reduces the variance** of the predictor.

However, larger values of K **increase the bias** of the predictor.

The bias-variance tradeoff

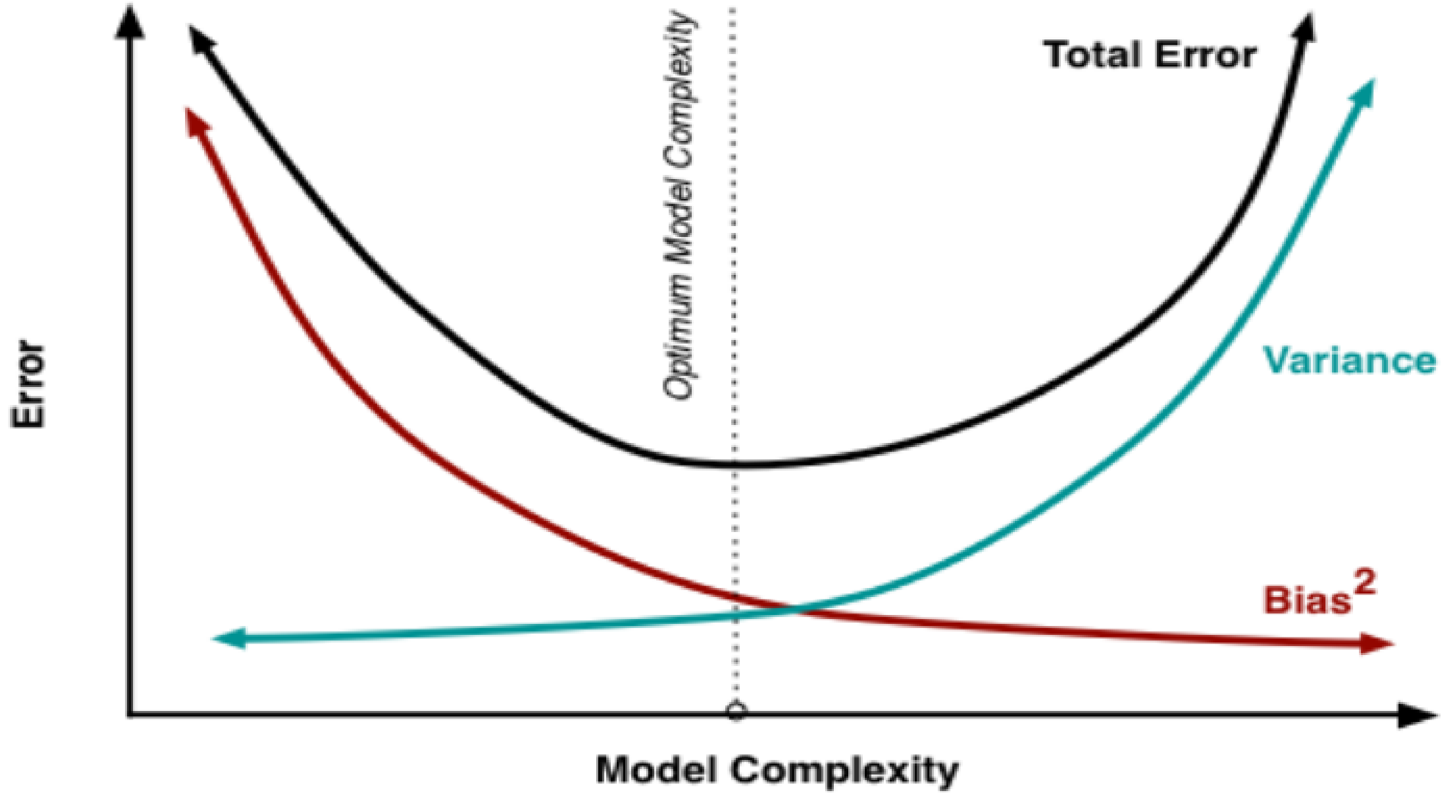
So there is a tradeoff:

- Too large K: out of sample error high due to bias
- Too small K: out of sample error high due to variance

This situation is typical in machine learning and is called the **bias-variance tradeoff**. In general:

- More flexible models (**more degrees of freedom**) have higher variance, but lower bias.
- Less flexible models (**fewer degrees of freedom**) have higher bias, but lower variance.

The bias-variance tradeoff



Choosing K

K nearest neighbours has a parameter (K) that must be selected somehow

K is known as a **hyperparameter** (to distinguish it from learned parameters, as we will see later in linear models)

Other models also have hyperparameters.

How should we select good values for such hyperparameters?

More generally, given a choice of models, how should we choose the one that we expect will generalize best?

Model selection

Model selection

- Choosing appropriate hyperparameters is called model selection.

- More generally: model selection is choosing between a set of candidate models (e.g. linear models, nearest neighbours). Hyperparameters give rise to new models.

- **Idea**: why don't we just try out a bunch of different hyperparameters, train models, test them on the test set, compute error, and use the hyperparameter that gives the lowest error?

Model selection

- Choosing appropriate hyperparameters is called model selection.

- More generally: model selection is choosing between a set of candidate models (e.g. linear models, nearest neighbours). Hyperparameters give rise to new models.

- **Idea**: why don't we just try out a bunch of different hyperparameters, train models, test them on the test set, compute error, and use the hyperparameter that gives the lowest error?

- **THIS IS A BAD IDEA!!!!!**

Model selection

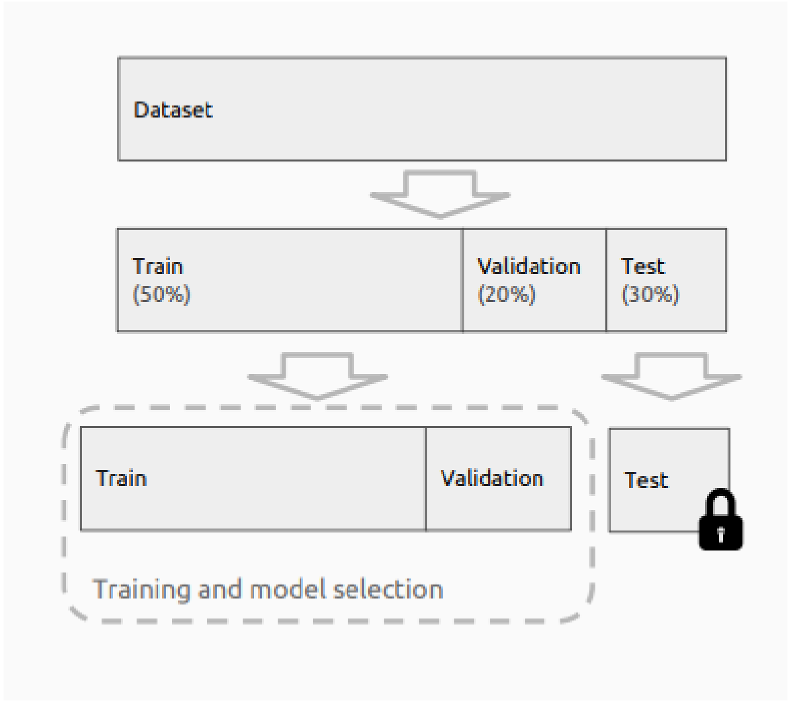
- Hyperparameters are just like any other value: selecting them to minimize error on the test set means you can **overfit the test set**!
- Form of data snooping.
- Means that test error is now a **biased estimate** of OOS error.
- You now need a new test set!
- What to do instead?

Validation sets

Split the data into 3 sets: training, validation, and test

Use the validation set to choose your model:

- Train with different hyperparameter values
- Compute error on validation set
- Select hyperparameters that give smallest error on validation set



Optional: when no more hyperparameters need to be tuned, retrain on (training + validation)

Finally: estimate OOS error on test set

Cross validation

Issue: using a validation set means you have less training data!

This may be problematic, especially when data is scarce.

We would like to use more of the training data for training, without having to split off a big chunk for validation.

Answer: cross-validation!

Cross validation

- **Idea**:

  1. Divide training data up into five chunks (folds).
  2. Train on chunk 1, 2, 3, 4. Compute validation error on 5.
  3. Now train on 1, 2, 3, 5. Compute validation error on 4.
  4. Now train on 1, 2, 4, 5. Compute validation error on 3.

- 5. ...
- 6. Average together errors.

- Choose model that minimizes average validation error.

K-fold cross validation

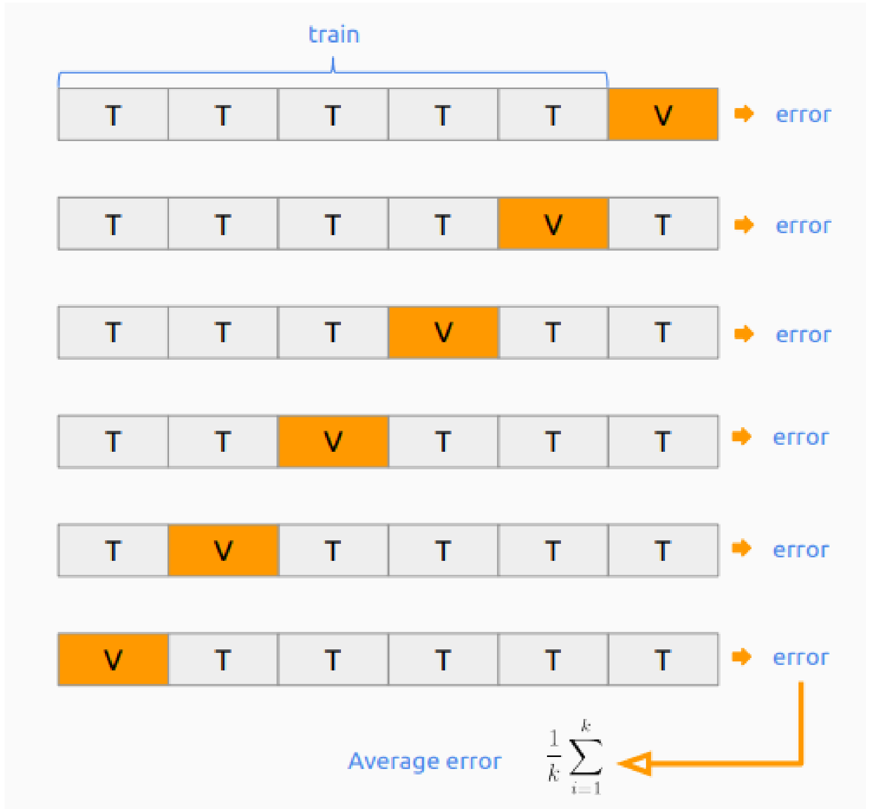
Divide training set up into k *folds*.

- K can be as small or large as you like
- Large k (smaller folds): more training data, slower
- Small k (larger folds): less training data, faster

Train on $(k-1)$ folds, validate on remaining fold

Repeat until all folds have been validated on

Average validation error over folds



Leave one out cross validation

Make folds as small as possible: 1 single training example per fold

Train on all data bar one example

Compute error on single “left out” example

Repeat $(N-1)$ times, leaving out a different example each time

Compute validation error as average of error on all N left out examples.

- **Advantages**
  - Make maximum use of the training data
  - Appropriate when there is very little training data available
- **Disadvantages**:
  - Need to run learning algorithm N times: may be very time consuming if learning is algorithm is slow

Model selection in practice

When you have **lots of data**: use a dedicated validation set.

When you have **less data** and more computing resources: use cross validation.

When you have **very little data** and want to make maximum use of it: use leave one out cross validation.

The no free lunch theorem

We’ve already introduced several machine learning algorithms.

Remainder of the module will introduce even more.

Why so many? Is there not one algorithm that will work reasonably well on all types of tasks?

No free lunch: **bias free learning is futile**.

You can’t do inference without making assumptions!

Different assumptions needed for different tasks.

The no free lunch theorem

No Free Lunch theorem (Wolpert, 1996): there is no one model that works best for every problem.

Assumptions of a great model for one problem may not hold for another; it is common in ML to try multiple models and select one that works best for a particular problem.

You typically cannot tell in advance which algorithm will work best: there is no **a-priori** distinction between them.

This is why **model selection** is so important: test several models and choose one with least validation error.

Summary

- Black box abstraction of supervised learning
- Regression and classification problems
- Types of error
- Training and test data splits
- Nearest neighbours
- Bias and variance
- K nearest neighbours
- Model selection
- The no free lunch theorem

Further reading

The elements of statistical learning:

- Intro to supervised learning, nearest neighbours: Chapter 1, 2
- Model selection, bias, variance: Chapter 7
- Linear and quadratic discriminant analysis: Section 4.3

Questions?

Resources

Alternate derivation of the bias-variance decomposition by Yasser Abu-Mostafa from Caltech. Video lecture on YouTube (**Highly recommended**): https://www.youtube.com/watch?v=zrEyxfl2-a8 . Starts approx 9 mins in.