Introduction to Machine Learning

Related reading: chapters 7 and 10 of the textbook

What is learning?

Learning is the ability to improve one’s behaviour based on experience.

- The range of behaviours is expanded: the agent can do more.
- The accuracy on tasks is improved: the agent can do things better.
- The speed is improved: the agent can do things faster.

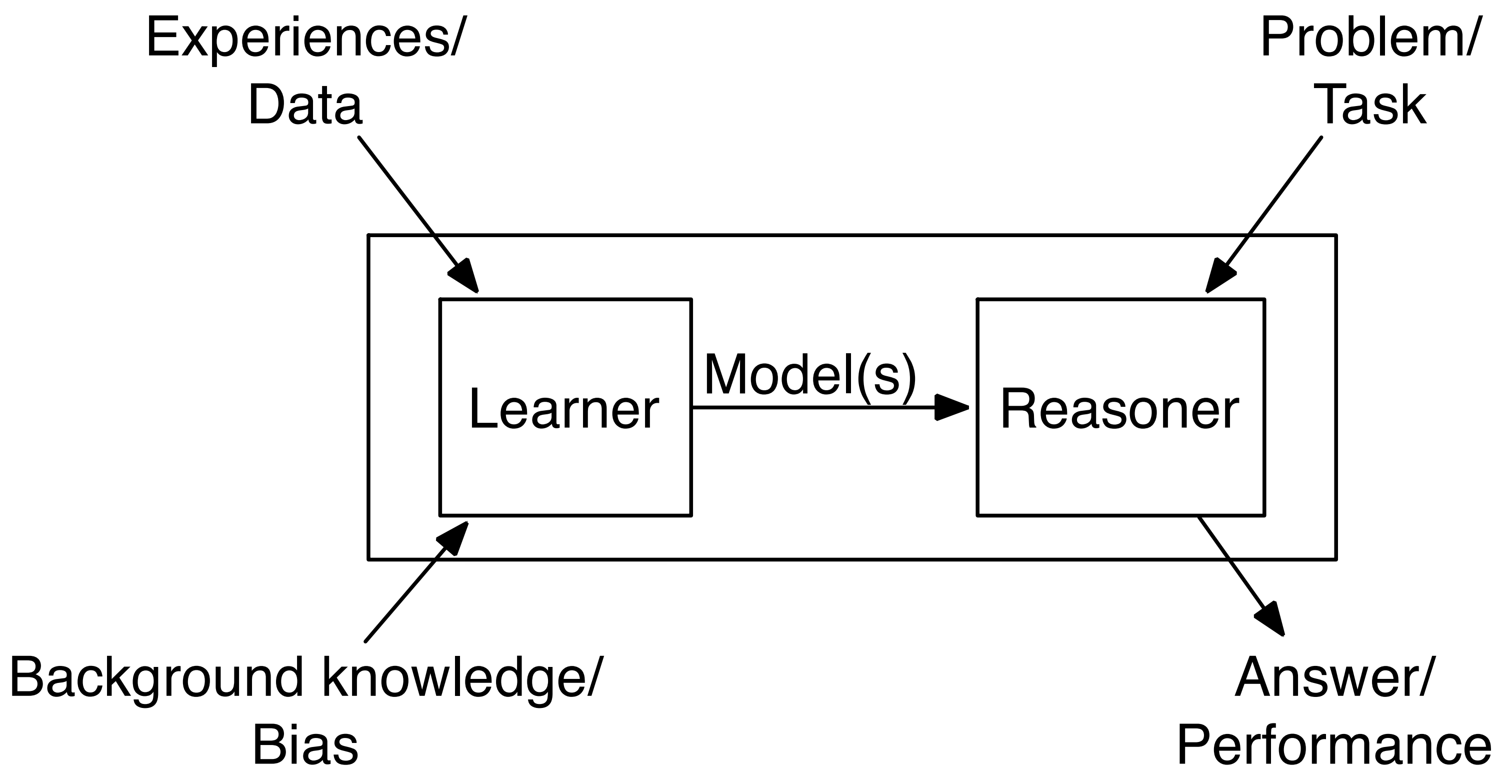
Components of a learning problem

The following components are part of any learning problem:

- Task: the behaviour or task that’s being improved. For example: classification, acting in an environment
- Data: the experiences that are being used to improve performance in the task.
- Measure of improvement: how can the improvement be measured?

  Example: increasing accuracy in prediction, new skills that were not present initially, improved speed.

Learning architecture



Supervised Learning

Given:

- a set of input attributes (features) $X_1, X_2, \ldots, X_n$;
- a target attribute (feature) Y (a class label or a real value);
- a set of training examples (instances) where the value of input and the target variables are given;

automatically **build a predictive model** that takes a new instance (where only the values for the input features are given) and returns (predicts) the value for the target feature for the given instance.

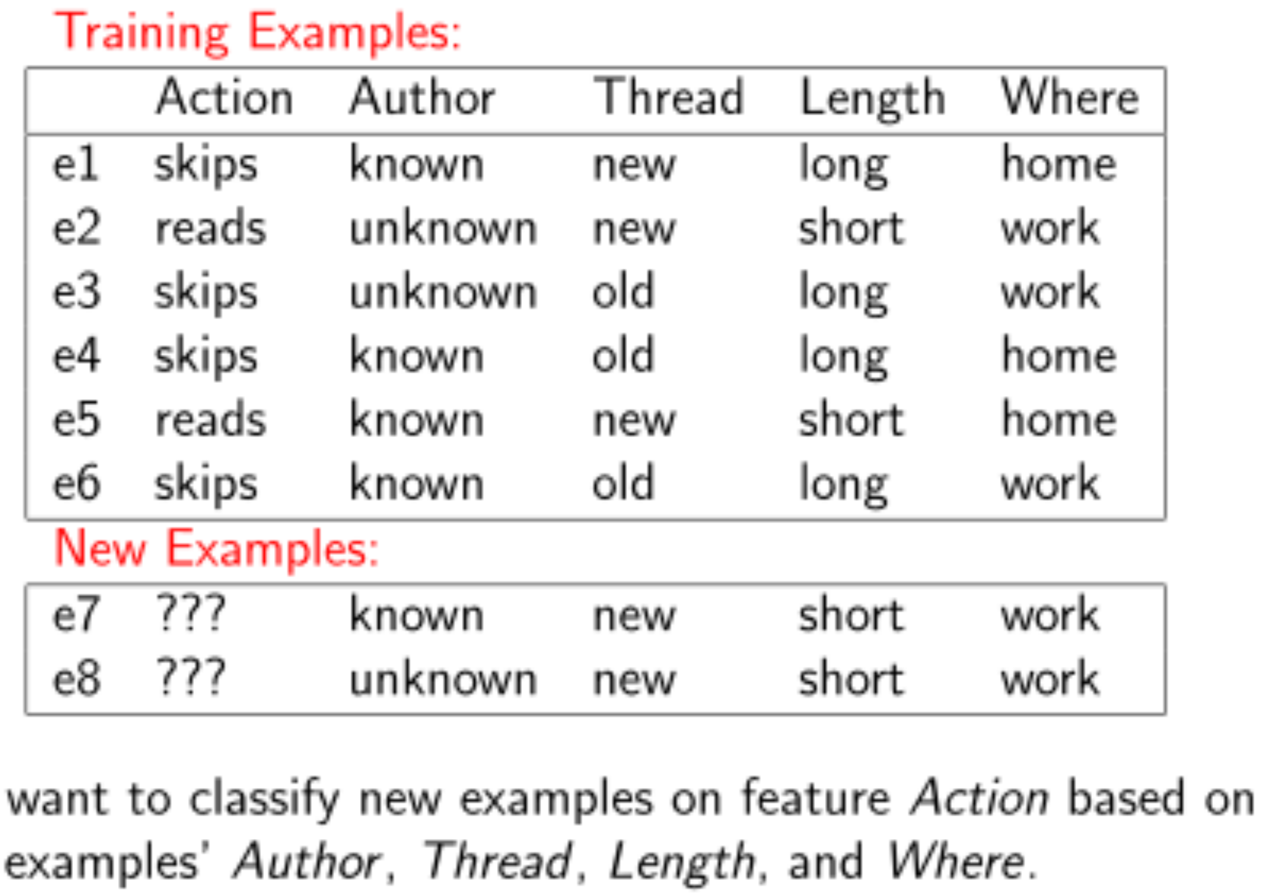
Note: the terms *feature*, *attribute*, and *(random) variable* are used with (more or less) the same meaning in this context.

Supervised Learning: Classification

In classification the target feature is discrete and represents a *class label*.

For instance in the following problem:

- Input attributes: Author, Thread, Length, Where
- Target attribute (class label): Action



Training Examples:

	Action	Author	Thread	Length	Where
e1	skips	known	new	long	home
e2	reads	unknown	new	short	work
e3	skips	unknown	old	long	work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

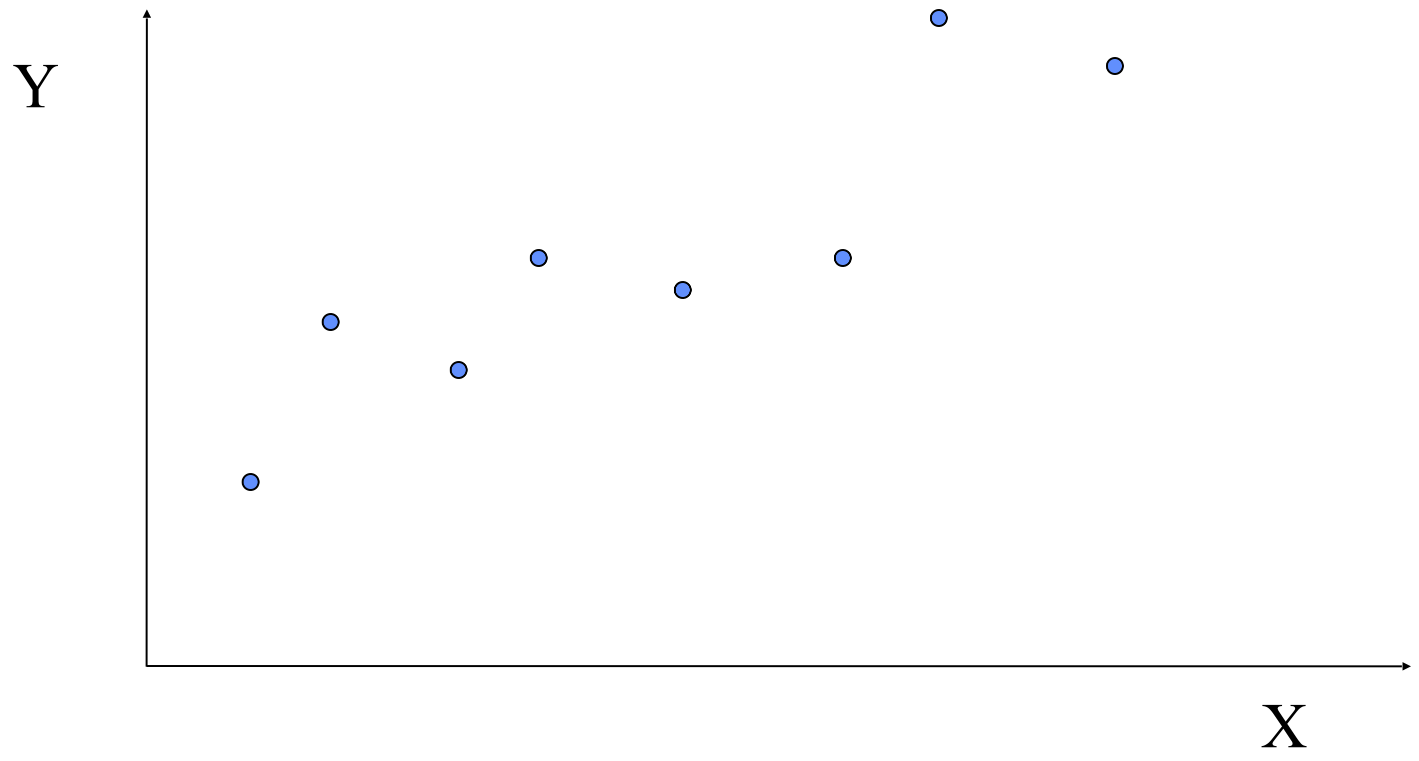
New Examples:

e7	???	known	new	short	work
e8	???	unknown	new	short	work

We want to classify new examples on feature *Action* based on the examples' *Author*, *Thread*, *Length*, and *Where*.

Supervised Learning: Regression

In regression the target feature is continuous.



Measuring Performance in Supervised Learning

- We measure how good or bad a model is with respect to a data set
- Common performance measures in classification:

$$\text{error} = \frac{\text{number of incorrectly classified (predicted) instances}}{\text{total number of instances}}$$

$$\text{accuracy} = \frac{\text{number of correctly classified (predicted) instances}}{\text{total number of instances}}$$

$$\text{accuracy} + \text{error} = 1$$

- Common performance measures in regression:
  - Mean Squared Error (MSE)
  - Mean Absolute Error (MAE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 . \qquad \text{MAE} = \frac{1}{n}\sum_{i=1}^{n}| Y_i - \hat{Y}_i | .$$

Example: accuracy and error

X1	X2	X3	Y (actual)	Y (predicted)
…	…	…	p	p
…	…	…	p	n
…	…	…	n	p
…	…	…	p	p
…	…	…	p	n
…	…	…	n	n
…	…	…	n	n

accuracy = 4/7
error = 3/7

Note: in binary classification problems (problems with two classes) we often call one class positive and the other negative.

Train and Test sets

A given set (multi-set) of examples is usually divided into:

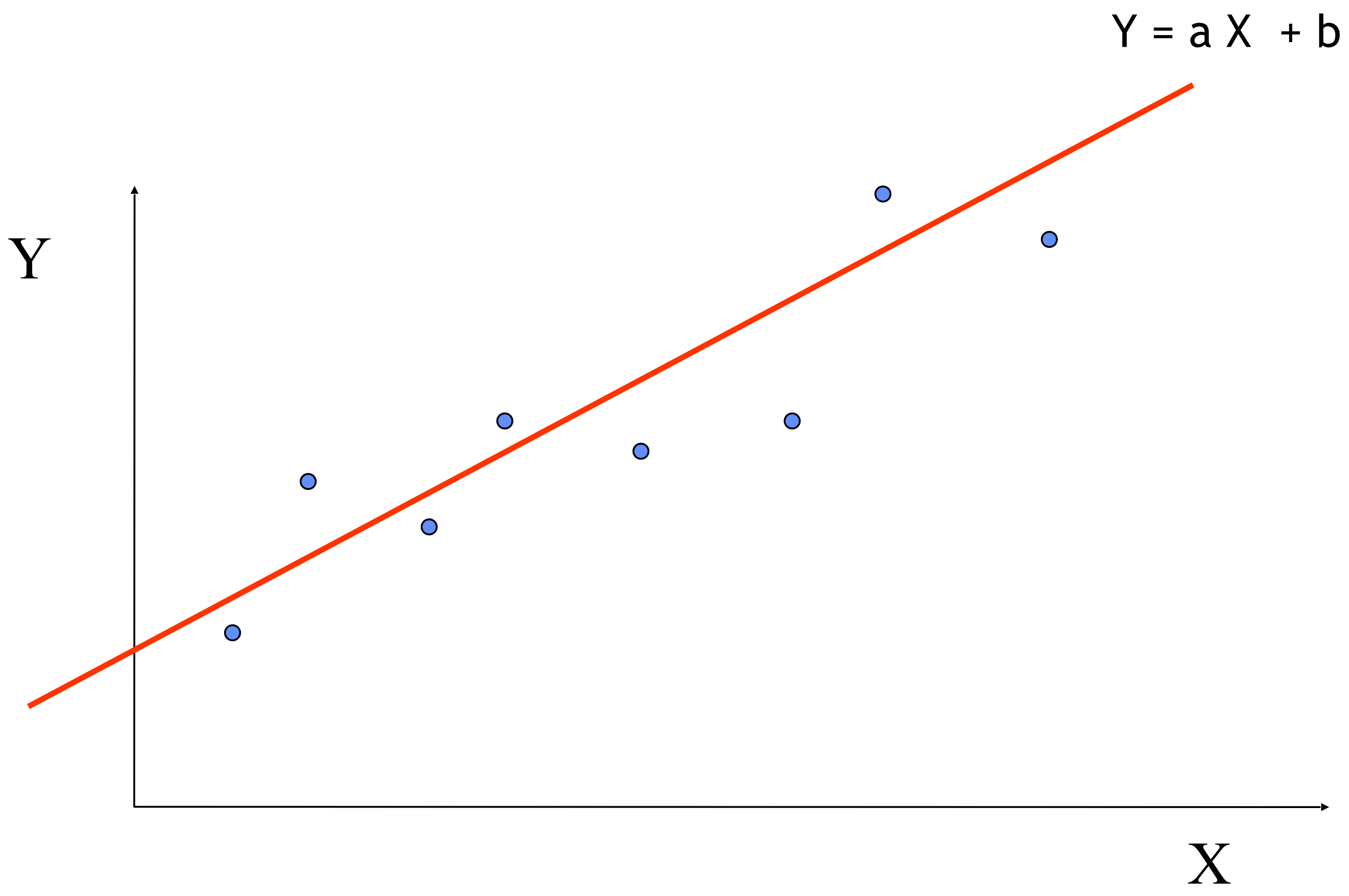
- training examples: that are used to train a model; and
- test examples that are used to evaluate the model.

These must be kept separate.

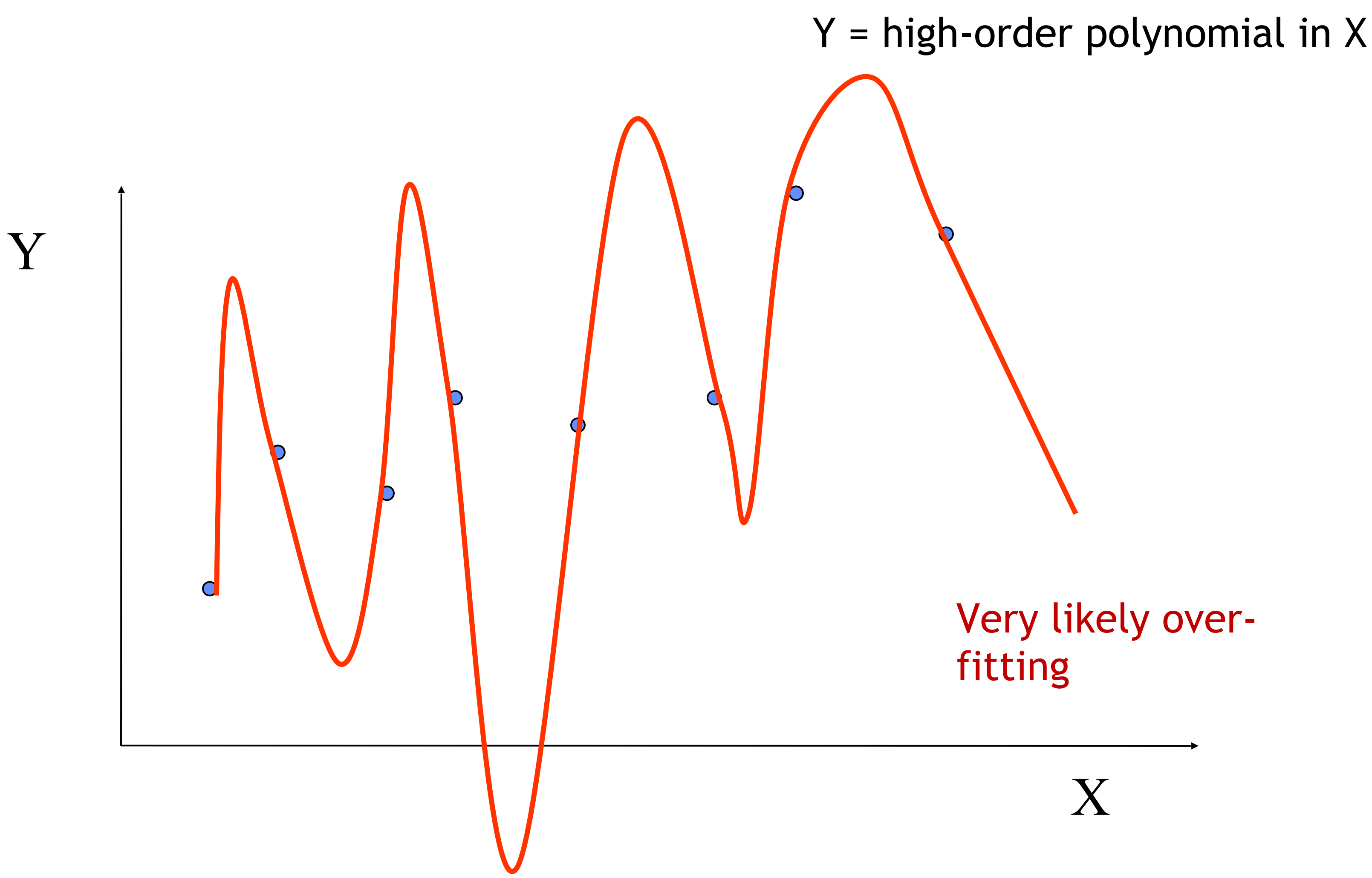
Why?

High performance (e.g. high accuracy) on training data does not necessary mean high performance on unseen data. Complex models can *overfit* the training data.

A Simple Model

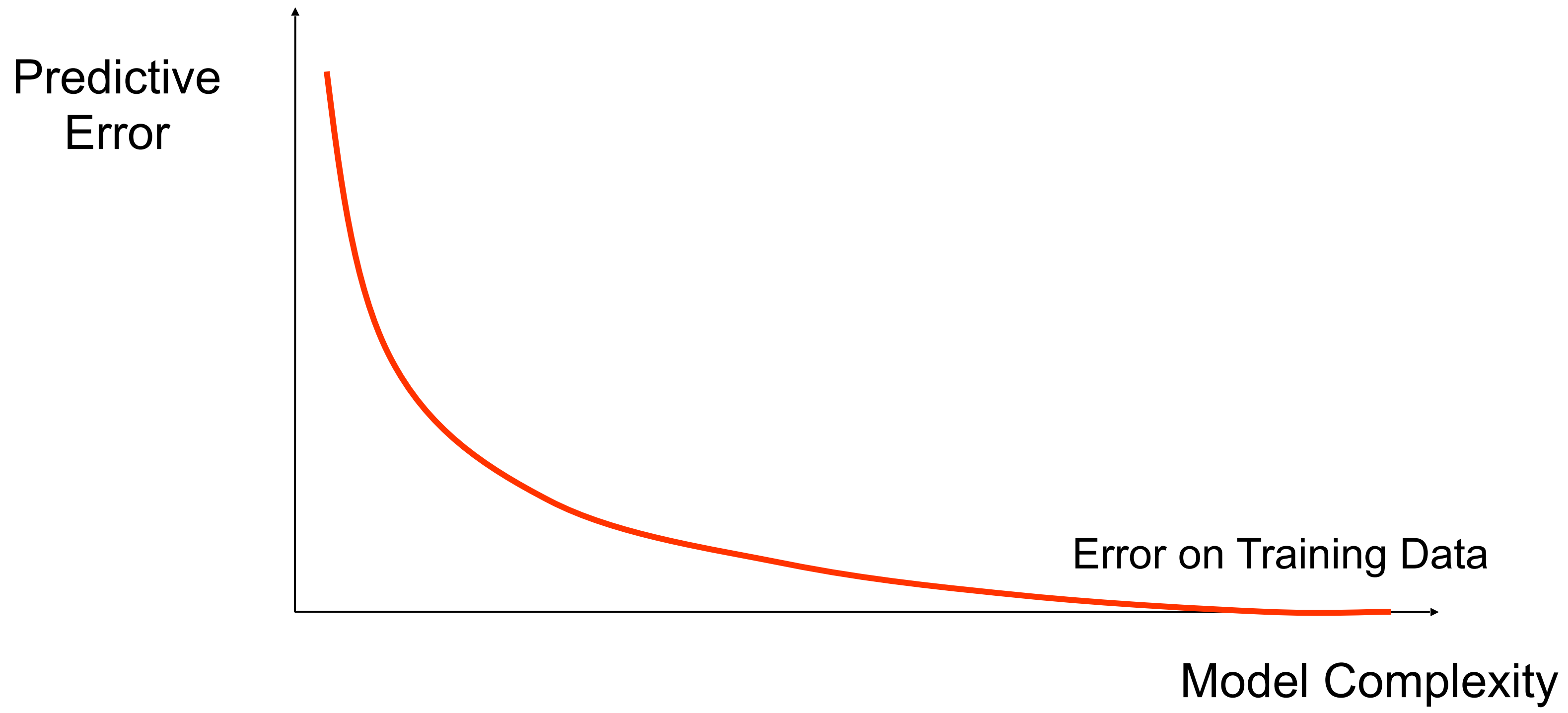


A Complex Model

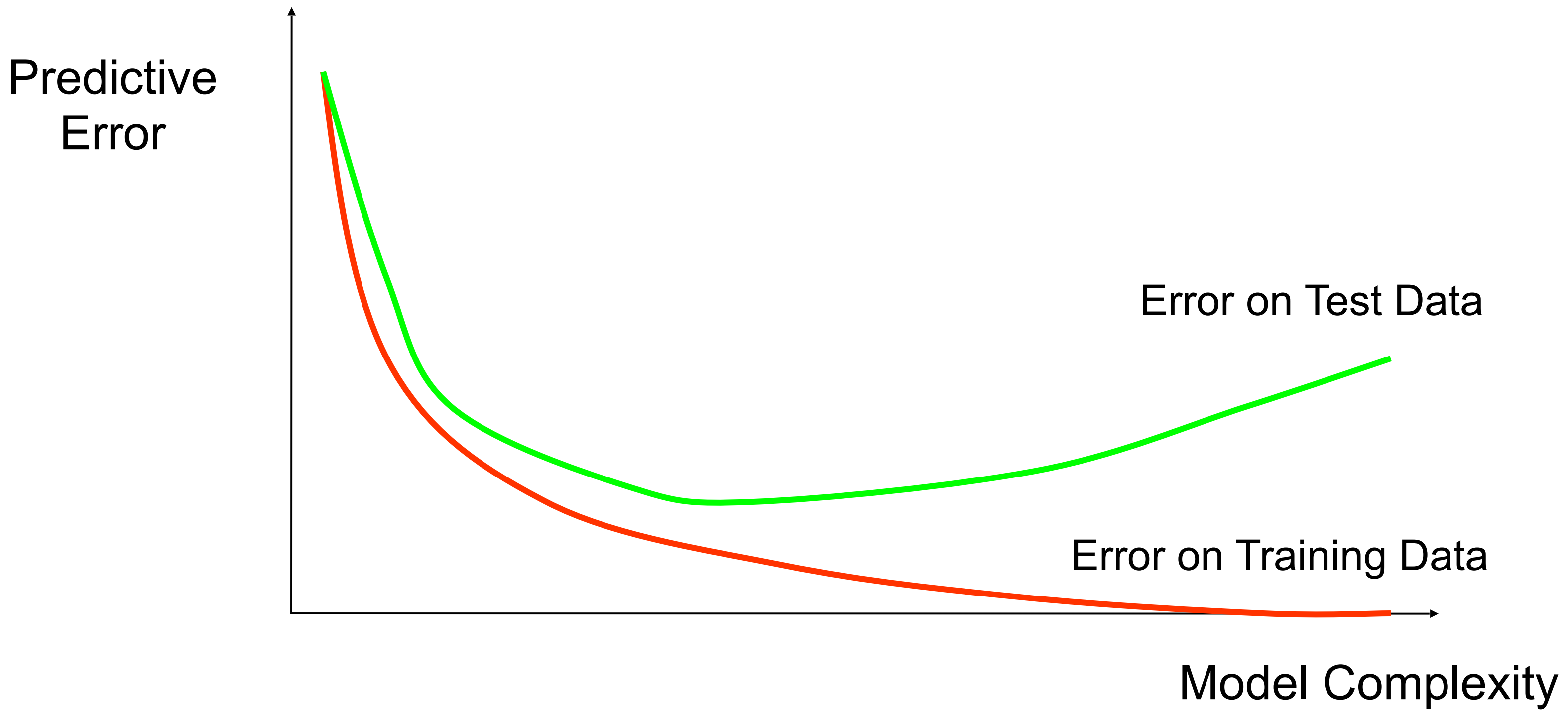


Very likely over-fitting

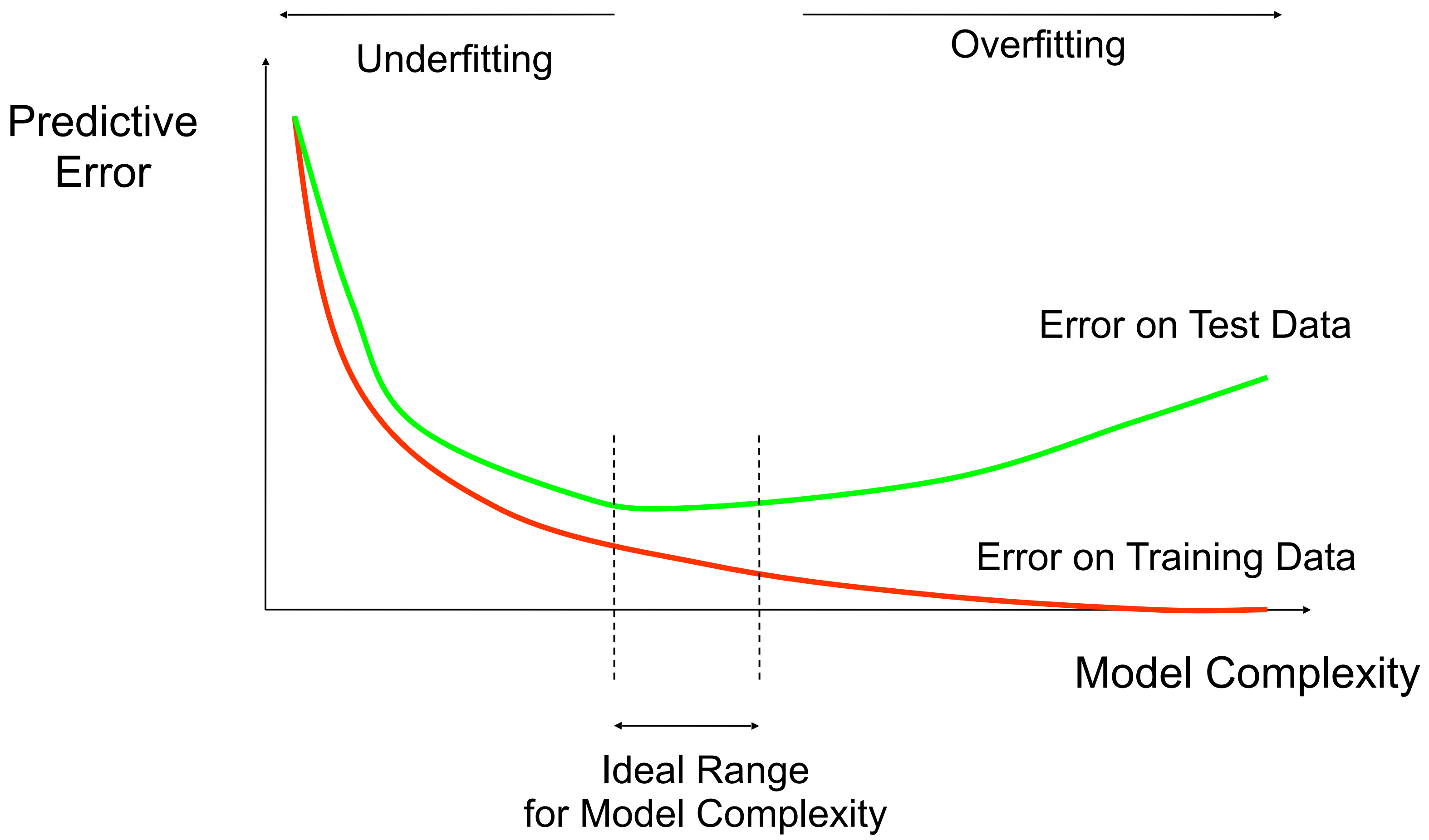
How Overfitting affects Prediction



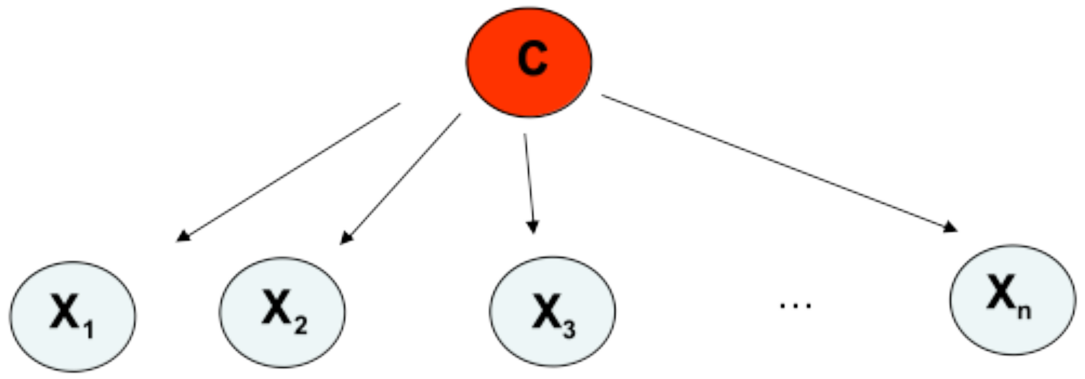
How Overfitting affects Prediction



How Overfitting affects Prediction



BNs Example: Naïve Bayes Models



$P(C \mid X_1,\ldots X_n) = \alpha \prod P(X_i \mid C)\, P(C) = \alpha\, P(X_1 \mid C)\, P(X_2 \mid C) \ldots P(X_n \mid C)\, P(C)$

Features X are conditionally independent given the ***class*** variable C

$P(C)$: Prior distribution of C, the class random variable
$P(X_i \mid C)$: Likelihood conditional distributions
$P(C \mid X_1,\ldots X_n)$: Posterior distribution

Widely used in machine learning
    e.g., spam email classification: X's = counts of words in emails

Conditional probabilities $P(X_i \mid C)$ can easily be estimated from labeled data.

Building a classifier

Determine whether a patient is susceptible to heart disease, given the following information:

- Whether they have a family history (true or false)
- Fasting blood sugar level (low or high)
- BMI (low, normal, high)

Classification (Heart disease): Yes or No

Given a set of data about past patients classification by experts, construct a classifier that will output the likely prediction (class) when given a new (unseen) patient (instance)

Sample dataset

History	BG	BMI	Heart Disease
true	low	high	Yes
true	low	normal	Yes
true	low	high	Yes
true	high	high	Yes
false	high	normal	Yes
true	low	normal	No
false	low	normal	No
true	low	low	No
false	low	high	No
false	high	low	No

Naïve Bayes

How to classify?

1. Find P(Class | an input vector) for different Classes.
2. Pick the class with the highest probability as the result of prediction.

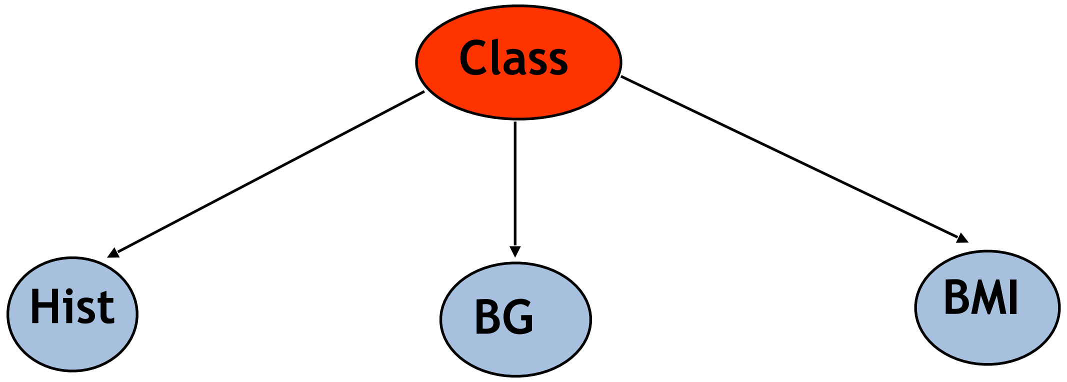
Problem: Hard to learn P(Class | Evidence)

- A lot of data is required; enough for each possible assignment of evidence
- For example P(Class=No | Hist = true, BG = high, BMI = low) needs lots of examples of (Hist = true, BG = high, BMI = low). Then count the fraction that are “no”. This is often infeasible, specially when there are many attributes.

A solution (compromise): assume input features are conditionally independent (naïve Bayes).

- Often the assumption is not entirely true but nevertheless yields reasonable performance.

Naïve Bayes Model



Input Features are conditionally independent given the ***Class*** variable.

P(Class | Hist, BG, BMI) = α x P(Hist|Class) x P(BG|Class) x P(BMI|Class) x P (Class)

P(Class): Prior distribution of Class (heart disease)
P(Input |Class): Likelihood conditional distributions
P(Class| Hist, BG, BMI): Posterior distribution of the class

Estimating CPTs from Data

	Class = Yes	Class = No
P(class)	5/10	5/10
P(history=true|class)	4/5	2/5
P(history=false|class)	1/5	3/5
P(BG=low|class)	3/5	4/5
P(BG=high|class)	2/5	1/5
P(BMI=low | class)	0/5	2/5
P(BMI=normal | class)	2/5	2/5
P(BMI=high | class)	3/5	1/5

Using the classifier

Classify a new case where:

- Hist = true
- BG = high
- BMI = low

P(Class | Hist=true, BG=high, BMI=low) = ?

P(Class = No| Hist=true, BG=high, BMI=low) =
α × P(Class=No) x P(Hist=true|Class=No) × P(BG=high|Class=No) × P(BMI=No|Class=No) =
$\alpha \times 0.5 \times 0.4 \times 0.2 \times 0.4 = \alpha \times 0.016$

P(Class = Yes| Hist=true, BG=high, BMI=low) =
α × P(Class=Yes) × P(Hist=true|Class=Yes) × P(BG=high|Class=Yes) × P(BMI=low|Class=Yes) =
$\alpha \times 0.5 \times 0.8 \times 0.4 \times 0 = \alpha \times 0$

After normalisation, the probabilities become 1 and 0.
Prediction: No
But the probabilities don’t seem realistic! Why?

Laplace smoothing

Problem: zero counts (in small data sets) lead to zero probabilities (i.e. impossible) which is too strong a claim based on only a small sample.

Solution: add a non-negative *pseudo-count* to the counts.

Let *count(constraints)* be the number of examples in the data set that satisfy the given constraints. For example:

- *count(A=a, B=b)* is the number of examples in the data set for which *A*=*a* and *B*=*b*
- *count(B=b)* is the number of examples in the data set for which *B*=*b*
- *count()* is the number of examples in the data set

Also let d*omain(A)* be the set of different values A can take and let *|domain(A)|* be the number of these different values. Then

$$P(A=a \mid B=b) \approx \frac{\text{count}(A=a, B=b) + \text{pseudocount}}{\sum_{a' \in \text{domain}(A)} (\text{count}(A=a', B=b) + \text{pseudocount})}$$

$$= \frac{\text{count}(A=a, B=b) + \text{pseudocount}}{\text{count}(B=b) + \text{pseudocount} \times |\text{domain}(A)|}$$

Estimating CPTs from Data

with smoothing (pseudo-count = 1)

	Class = Yes	Class = No
P(class)	(5+1)/(10+2) = 6/12	(5+1)/(10+2) = 6/12
P(history=true|class)	(4+1)/(5+2) = 5/7	(2+1)/(5+2) = 3/7
P(history=false|class)	(1+1)/(5+2) = 2/7	(3+1)/(5+2) = 4/7
P(BG=low|class)	(3+1)/(5+2) = 4/7	(4+1)/(5+2) = 5/7
P(BG=high|class)	(2+1)/(5+2) = 5/7	(1+1)/(5+2) = 5/7
P(BMI=low | class)	(0+1)/(5+3) = 1/8	(2+1)/(5+3) = 3/8
P(BMI=normal | class)	(2+1)/(5+3) = 3/8	(2+1)/(5+3) = 3/8
P(BMI=high | class)	(3+1)/(5+3) = 4/8	(1+1)/(5+3) = 2/8

Classification after smoothing

Classify a new case where:

- Hist = true
- BG = high
- BMI = low

P(Class | Hist=true, BG=high, BMI=low) = ?

P(Class = No| Hist=true, BG=high, BMI=low) =
α × P(Class=No) x P(Hist=true|Class=No) × P(BG=high|Class=No) × P(BMI=No|Class=No) =
α x 1/2 x 3/7 x 2/7 x 3/8 = α x 18/784

P(Class = Yes| Hist=true, BG=high, BMI=low) =
α × P(Class=Yes) × P(Hist=true|Class=Yes) × P(BG=high|Class=Yes) × P(BMI=low|Class=Yes) =
α x 1/2 x 5/7 x 3/7 x 1/8 = α x 15/784

Prediction: ‘No’

Same prediction but different confidence.

Parametric vs non-parametric models

- Parametric models are described with a set of parameters.
- With these models, learning means finding the optimal value of these parameters (e.g. linear regression, naive Bayes, neural networks).

- Non-parametric models are not characterised by parameters. A family of them is instance-based learning.
- Instance based learning is based on the memorisation of the dataset.
- A prediction is obtained by looking into the memorised examples.
- The cost of the learning process is 0, all the cost is in computing the prediction.
- This kind of learning is also known as lazy learning.

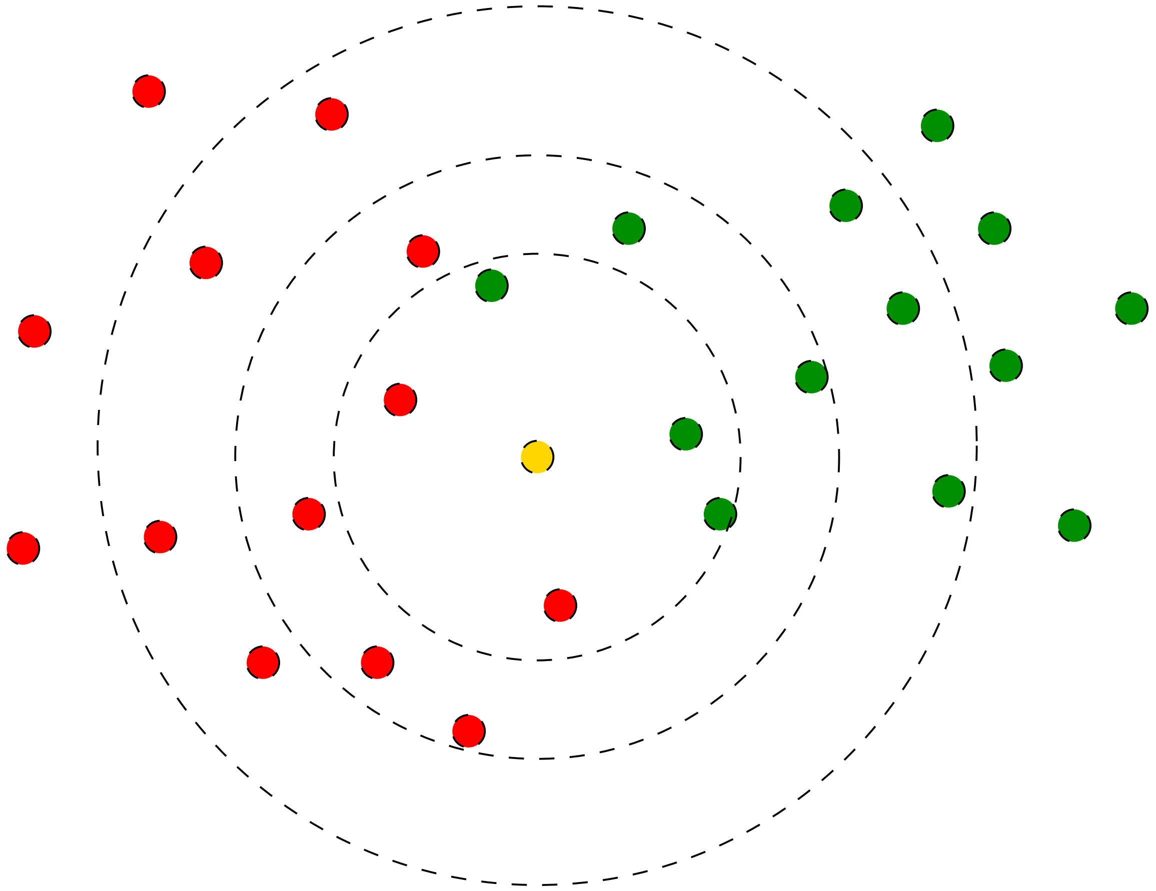
k-Nearest Neighbours (*k*-NN)

- **K-nearest neighbours** uses the local neighborhood to obtain a prediction
- The K memorized examples more similar to the one that is being classified are retrieved
- A distance function is needed to compare the examples similarity
  - Euclidean distance ($d(x_j, x_k) = \sqrt{\sum_i (x_{j,i} - x_{k,i})^2}$)
  - Mahnattan distance ($d(x_j, x_k) = \sum_i |x_{j,i} - x_{k,i}|$)
- This means that if we change the distance function, we change how examples are classified

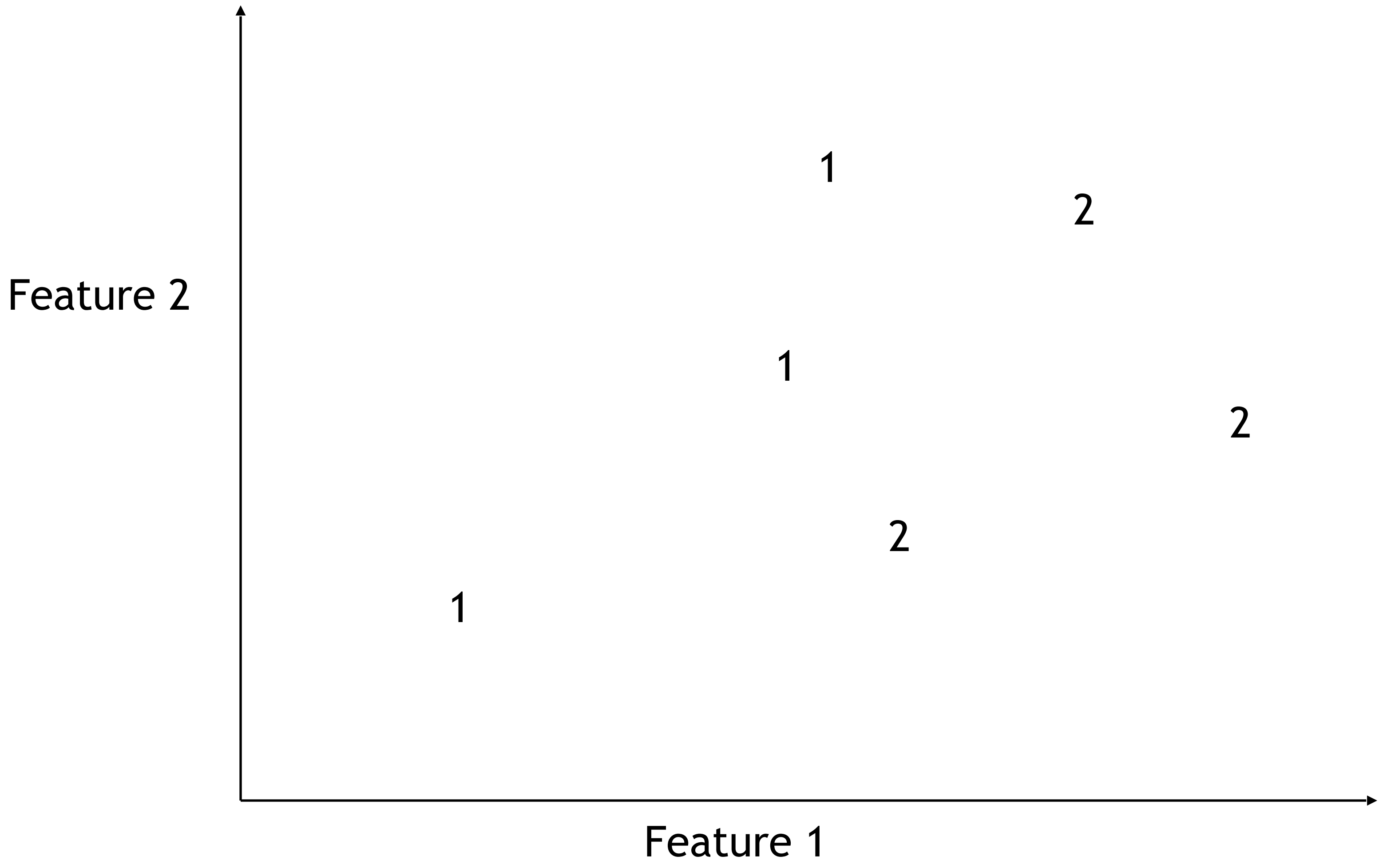
k-Nearest Neighbours - Algorithm

- Training: Store all the examples
- Prediction: $h(x_{new})$
  - Let be $x_1, \ldots, x_k$ the k more similar examples to x_{new}
  - $h(x_{new})=$ combine_predictions($x_1, \ldots, x_k$)
- The parameters of the algorithm are the number k of neighbours and the procedure for combining the predictions of the k examples
- The value of k has to be adjusted (crossvalidation)
  - We can overfit (k too low)
  - We can underfit (k too high)

k-nearest prediction

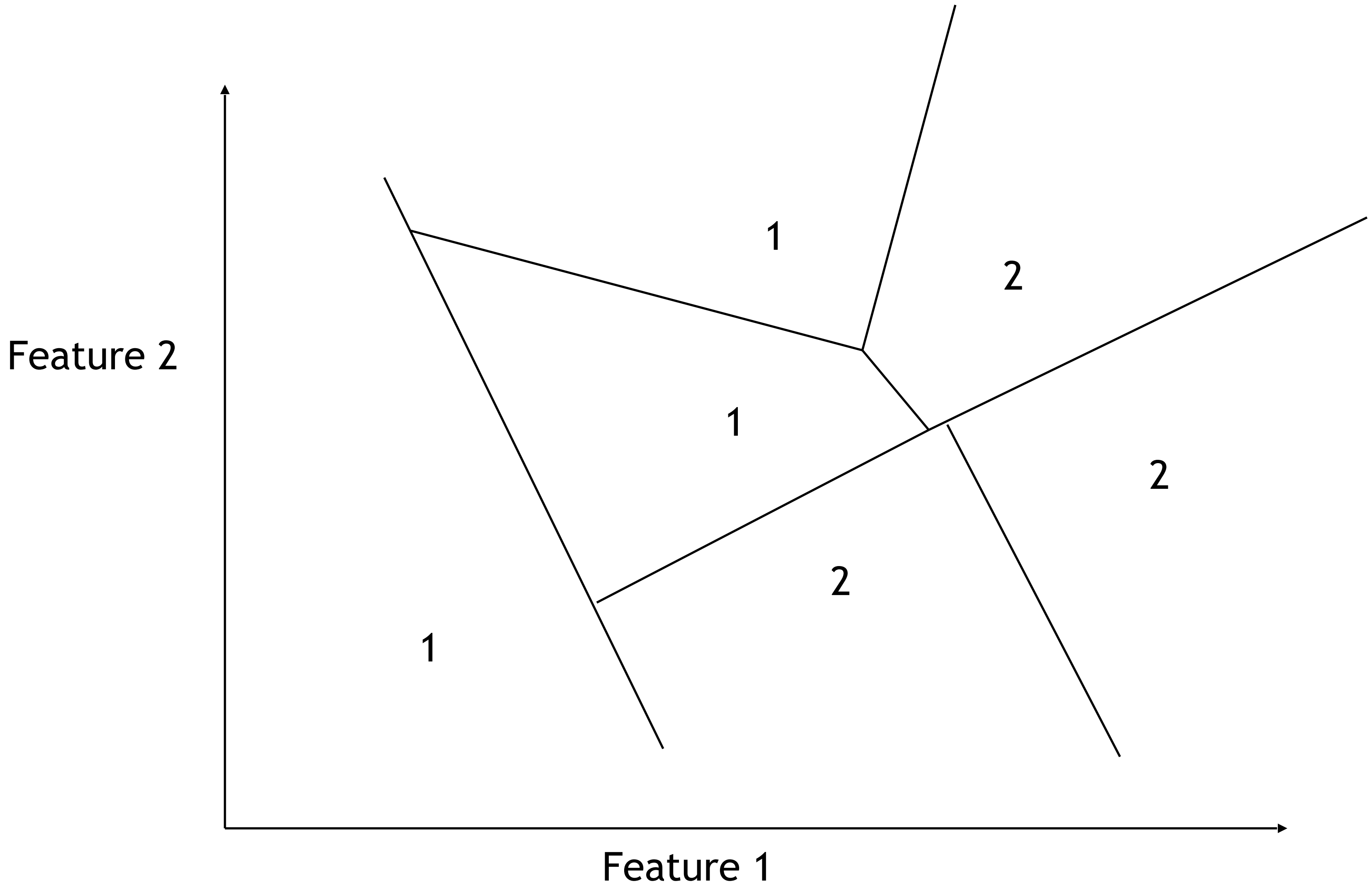


Geometric Interpretation of Nearest Neighbour



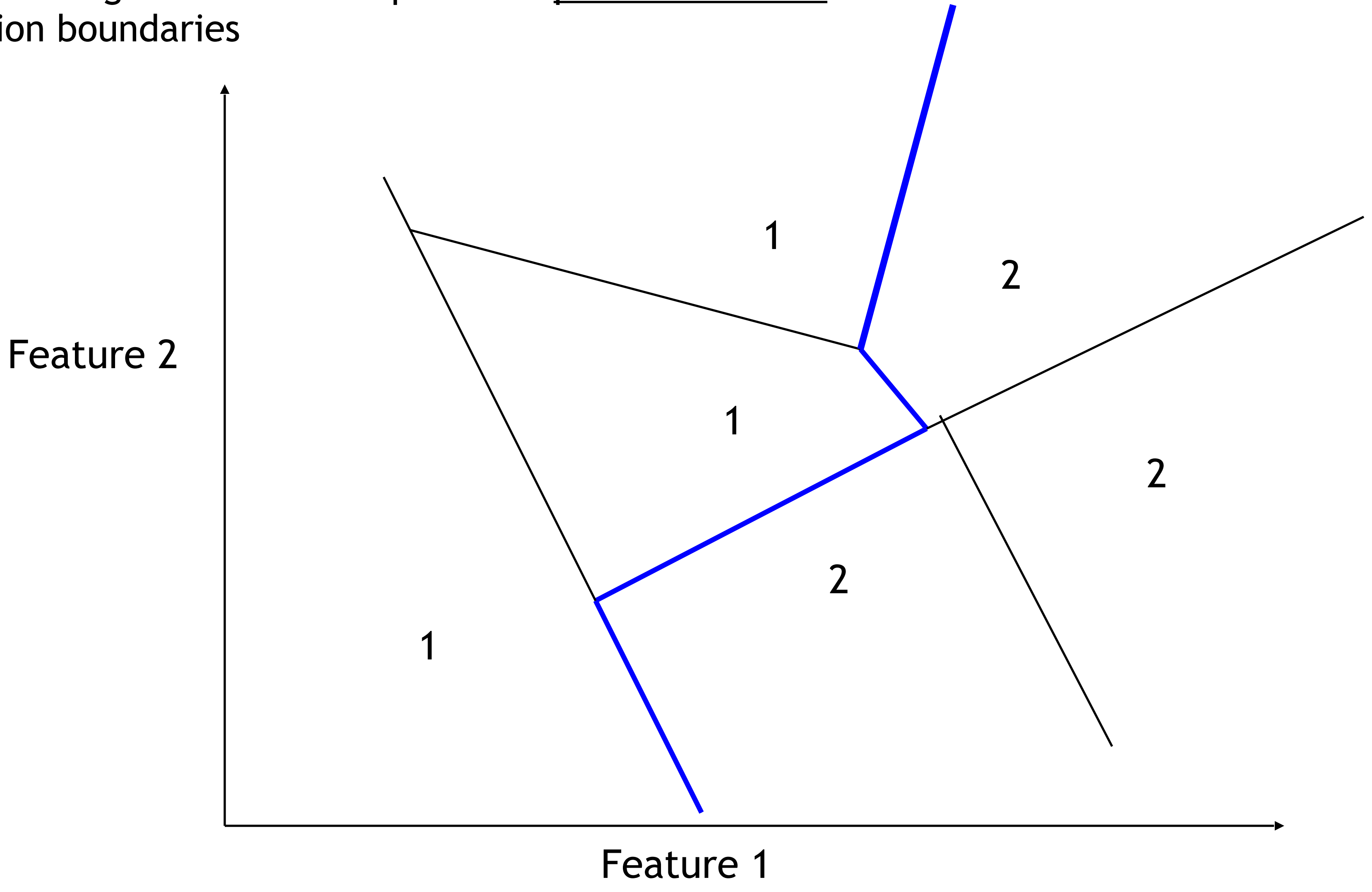
Regions for Nearest Neighbours

Each data point defines a “cell” of space that is closest to it. All points within that cell are assigned that class

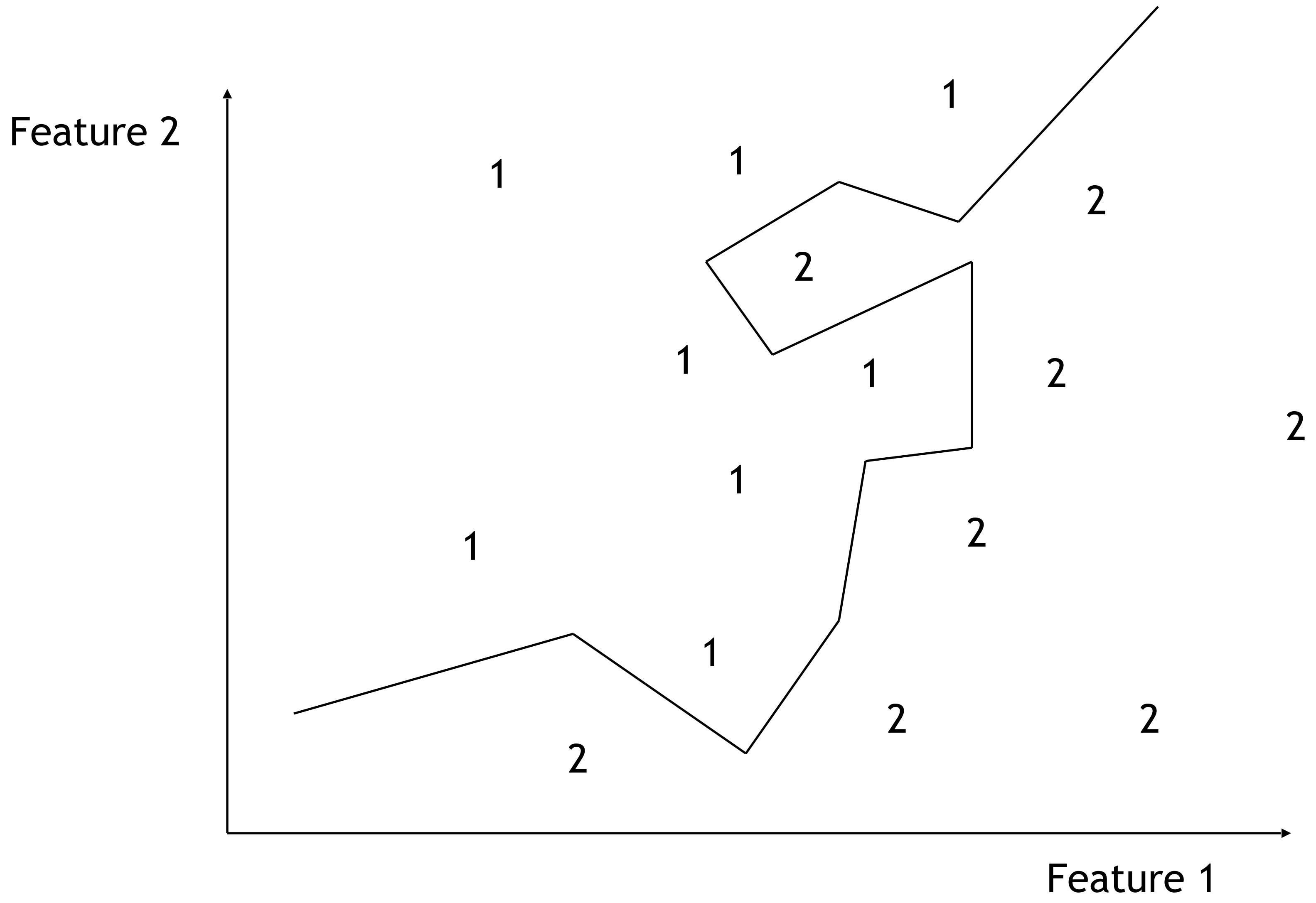


Decision Boundary

- Overall decision boundary = union of cell boundaries where class decision is different on each side
- Nearest-neighbour classifier produces <u>piecewise linear</u> decision boundaries



- More points means More Complex Decision Boundary



Linear function

A linear function of features $X_1, \ldots, X_n$ is a function of the form:

$$f^{\overline{w}}(X_1, \ldots, X_n) = w_0 + w_1 X_1 + \cdots + w_n X_n$$

Invent a new feature X_0 which has value 1, to make it not a special case.

$$f^{\overline{w}}(X_1, \ldots, X_n) = \sum_{i=0}^{n} w_i X_i$$

Linear regression

- Aim: predict feature Y from features $X_1, \ldots, X_n$.
- A feature is a function of an example.
  $X_i(e)$ is the value of feature X_i on example e.
- Linear regression: predict a linear function of the input features.

$$\begin{aligned}\widehat{Y}^{\overline{w}}(e) &= w_0 + w_1 X_1(e) + \cdots + w_n X_n(e) \\ &= \sum_{i=0}^{n} w_i X_i(e)\ ,\end{aligned}$$

$\widehat{Y}^{\overline{w}}(e)$ is the predicted value for Y on example e.
It depends on the weights $\overline{w}$.

Sum of squares

The sum of squares error on examples E for target Y is:

$$\begin{aligned} SSE(E,\overline{w}) &= \sum_{e\in E}(Y(e) - \widehat{Y}^{\overline{w}}(e))^2 \\ &= \sum_{e\in E}\left(Y(e) - \sum_{i=0}^{n} w_i * X_i(e)\right)^2 . \end{aligned}$$

Goal: given examples E, find weights that minimize $SSE(E,\overline{w})$.

Finding weights that minimise error

- Find the minimum analytically.
  Effective when it can be done (e.g., for linear regression).
- Find the minimum iteratively.
  Works for larger classes of problems.
  Gradient descent:

  $$w_i \leftarrow w_i - \eta \frac{\partial}{\partial w_i} Error(E, \overline{w})$$

  η is the gradient descent step size, the learning rate.