Overfitting and K-Nearest Neighbour

Dr. Xiaowei Huang

https://cgi.csc.liv.ac.uk/~xiaowei/

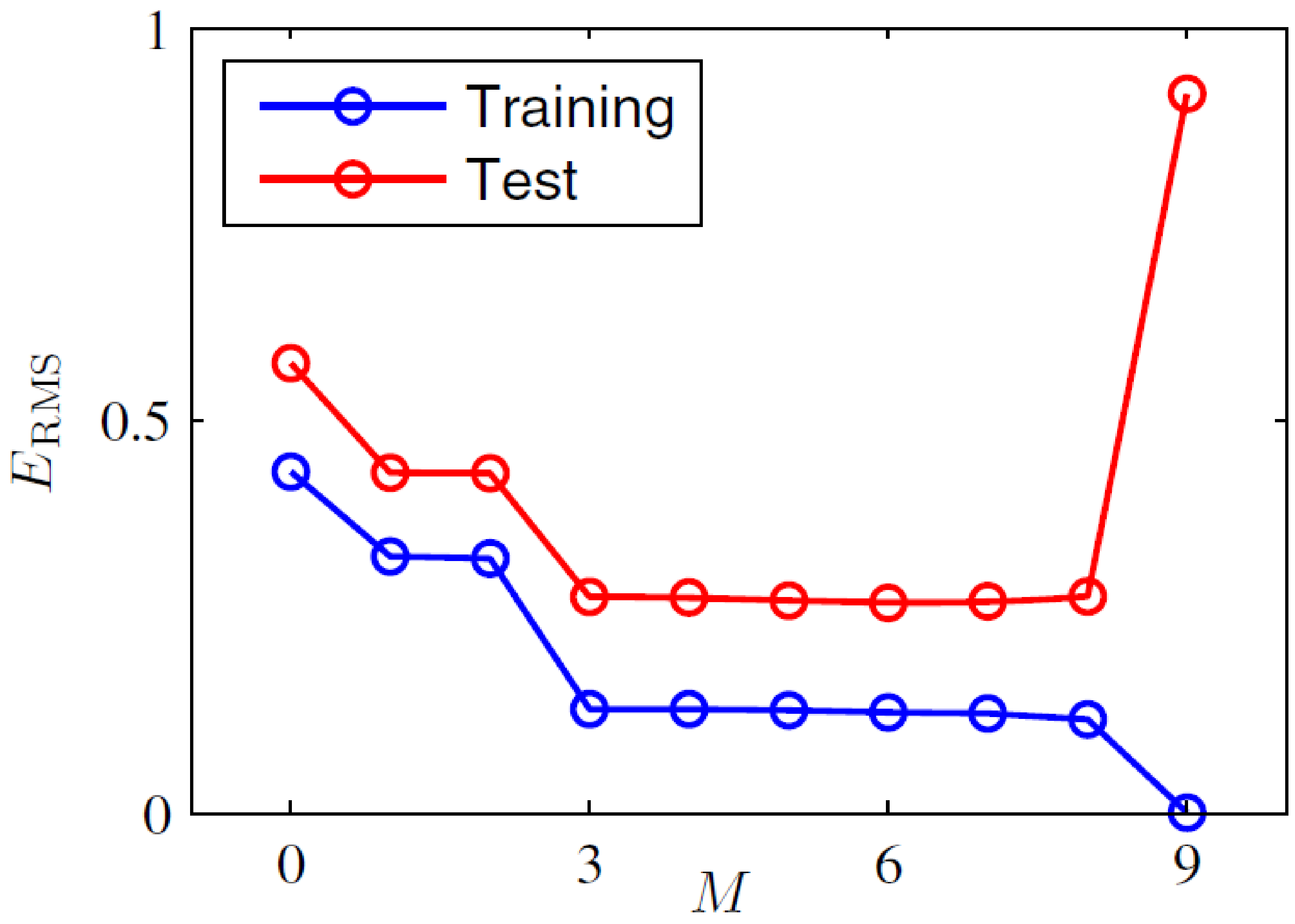
Up to now,

- Recap basic knowledge
- Decision tree learning
  - General algorithm
  - How to split
  - Identify the best feature to split
  - Stopping criteria
  - Accuracy

Today’s Topics

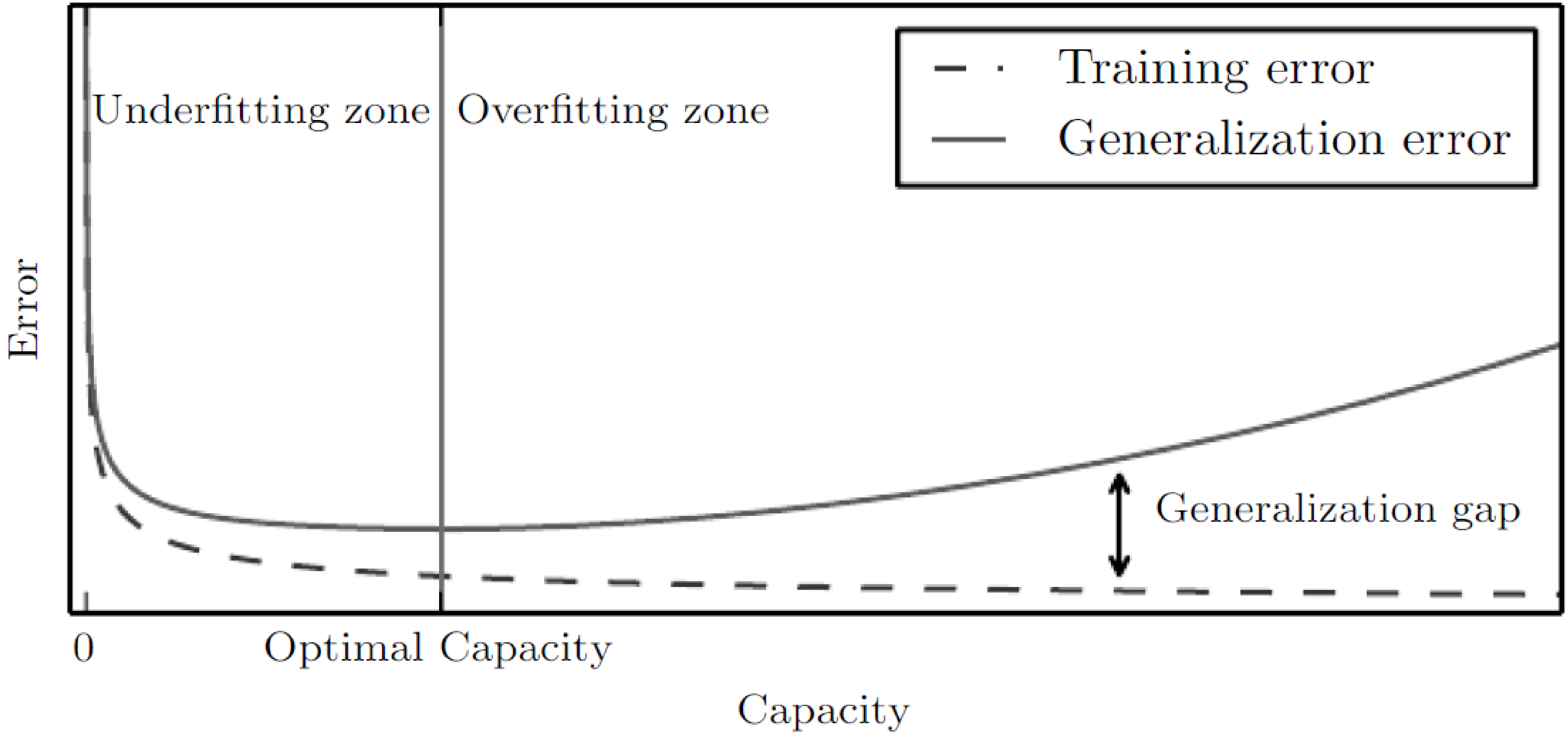
- Overfitting
- k-NN classification

Example: regression using polynomial

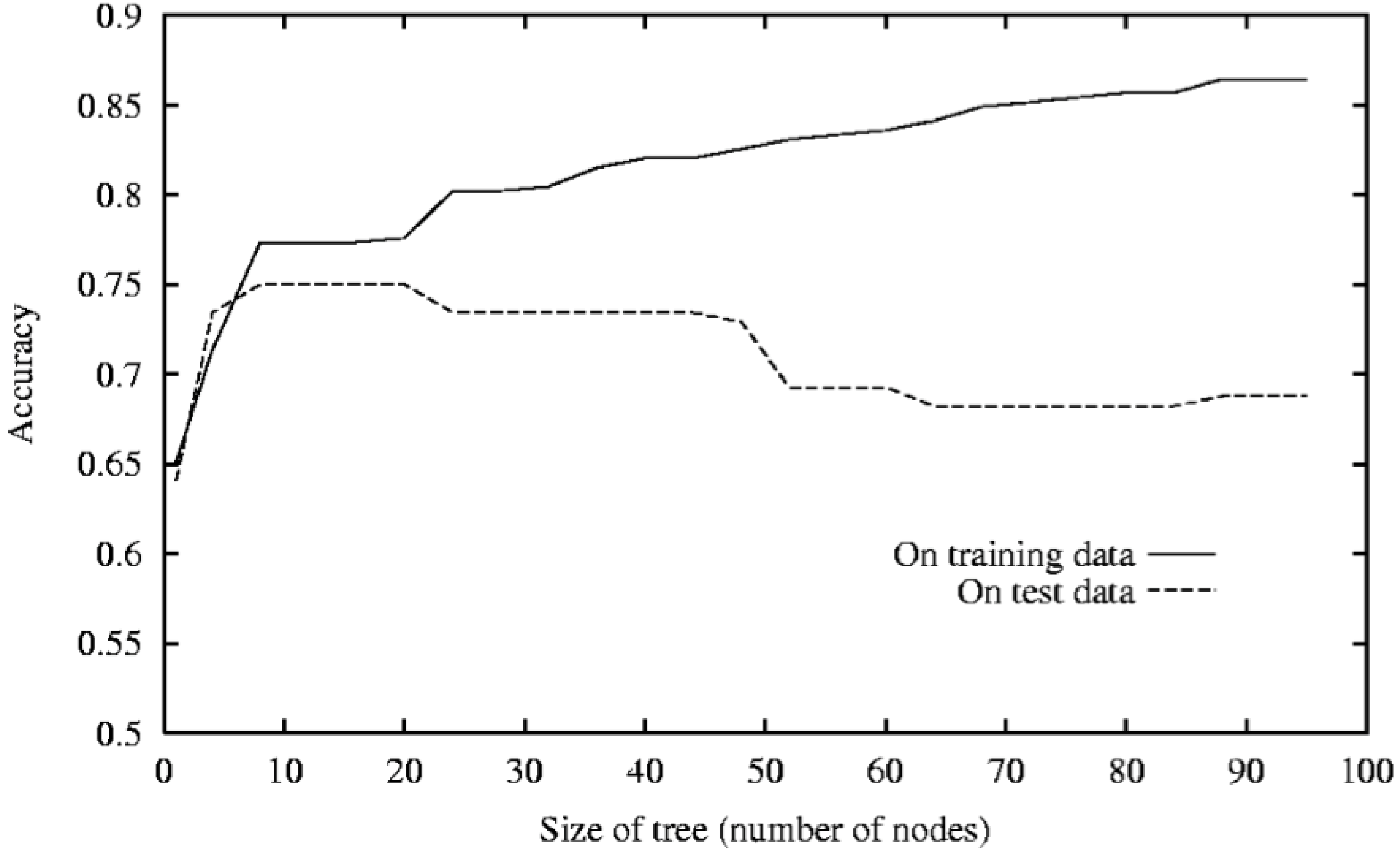


RMS: root mean square, i.e., the square root of the mean square

General phenomenon



Overfitting in decision trees



Prevent overfitting

- cause: training error and expected error are different
  - there may be noise in the training data
  - training data is of limited size, resulting in difference from the true distribution
  - larger the hypothesis class, easier to find a hypothesis that fits the difference between the training data and the true distribution
- prevent overfitting:
  - cleaner training data help!
  - more training data help!
  - throwing away unnecessary hypotheses helps! (Occam's Razor)

Overfitting in Decision Tree

Overfitting

- consider error of model M over
  - training data: $Error(D_{training}, M)$
  - entire distribution of data: $Error(D_{true}, M)$
- model $M \in H$ *overfits* the training data if there is an alternative model $M' \in H$ such that

$$Error(D_{training}, M) < Error(D_{training}, M')$$

Perform better on training dataset

$$Error(D_{true}, M) > Error(D_{true}, M')$$

Perform worse on true distribution

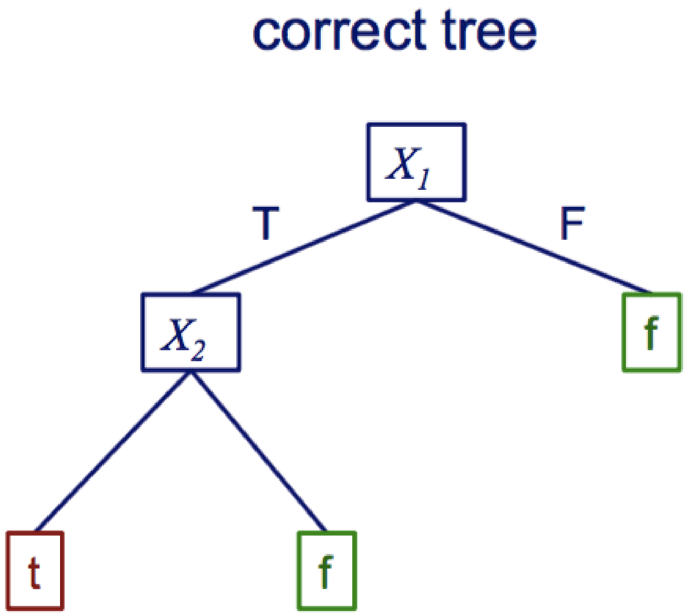
Example 1: overfitting with noisy data

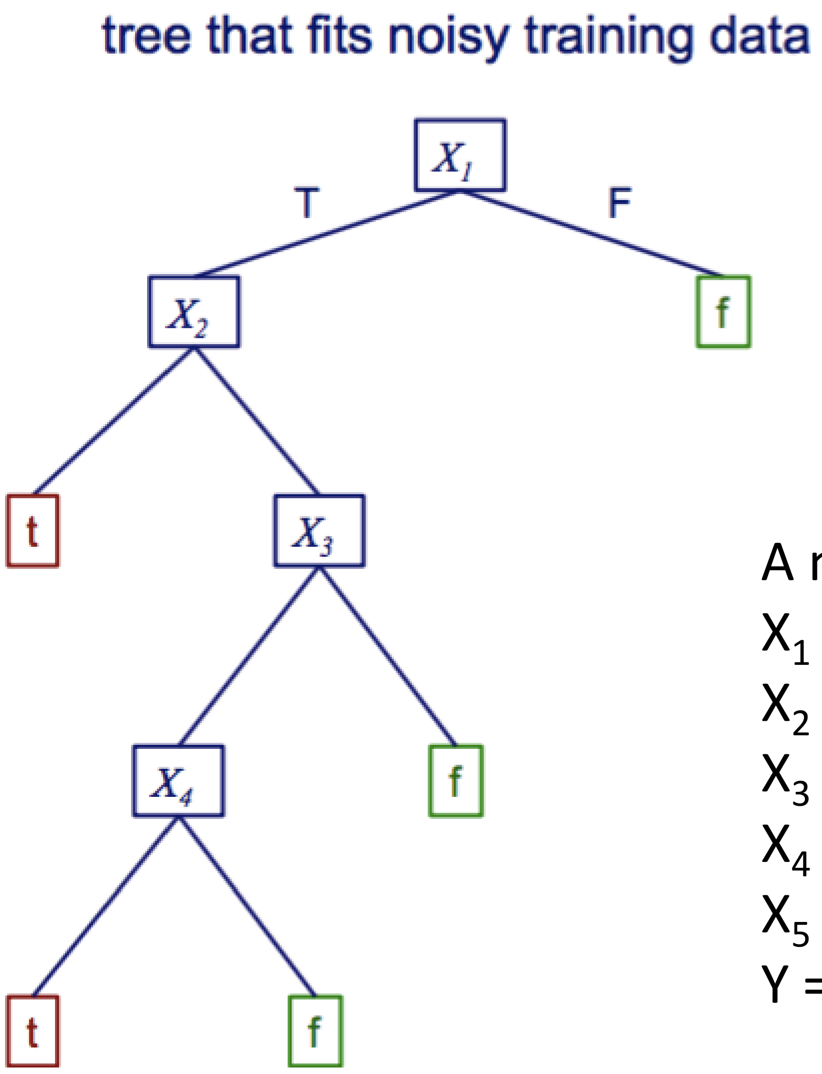
- suppose
  - the target concept is $Y = X_1 \wedge X_2$
  - there is noise in some feature values
  - we're given the following training set

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	f	f	t	...	t
t	f	t	t	f	...	t
t	f	f	t	f	...	f
t	f	t	f	f	...	f
f	t	t	f	t	...	f

noisy value

Example 1: overfitting with noisy data





$$Y = X_1 \wedge X_2$$

A noisy data sample:

X_1 = t

X_2 = f

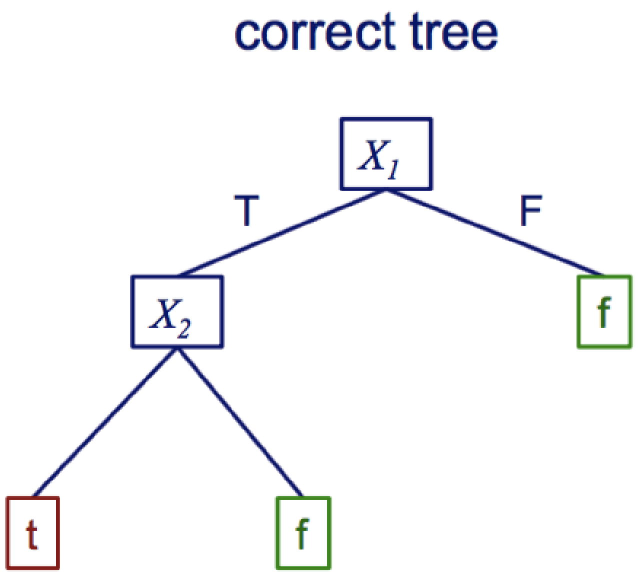
X_3 = t

X_4 = t

X_5 = f

Y = t

Example 1: overfitting with noisy data



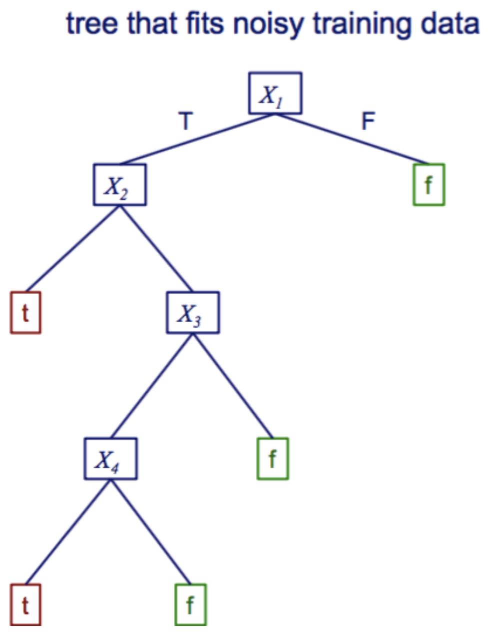
$$Y = X_1 \wedge X_2$$

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	f	f	t	...	t
t	f	t	t	f	...	t
t	f	f	t	f	...	f
t	f	t	f	f	...	f
f	t	t	f	t	...	f

noisy value

- What is the accuracy?
  - Accuracy($D_{training}$,M) = 5/6
  - Accuracy(D_{true},M) = 100%

Example 1: overfitting with noisy data



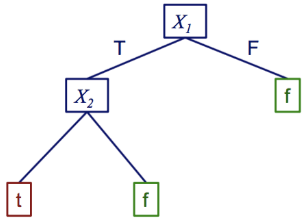
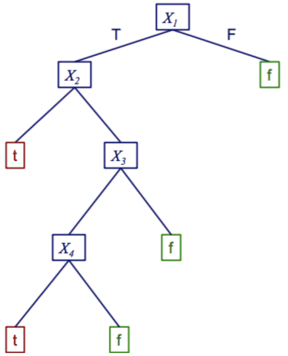
$$Y = X_1 \wedge X_2$$

X_1	X_2	X_3	X_4	X_5	…	Y
t	t	t	t	t	…	t
t	t	f	f	t	…	t
t	f	t	t	f	…	t
t	f	f	t	f	…	f
t	f	t	f	f	…	f
f	t	t	f	t	…	f

noisy value

- What is the accuracy?
  - Accuracy($D_{training}$,M) = 100%
  - Accuracy(D_{true},M) < 100%

Example 1: overfitting with noisy data

		Training set accuracy	True accuracy
M_1	correct tree 	5/6	100%
M_2	tree that fits noisy training data 	100%	< 100 %

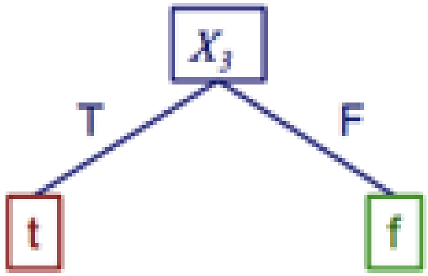
M_2 is overfitting!

Example 2: overfitting with noise-free data

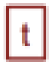
- suppose
  - the target concept is $Y = X_1 \wedge X_2$
  - $P(X_3 = t) = 0.5$ for both classes
  - $P(Y = t) = 0.66$
  - we're given the following training set

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	t	f	t	...	t
t	t	t	t	f	...	t
t	f	f	t	f	...	f
f	t	f	f	t	...	f

Example 2: overfitting with noise-free data



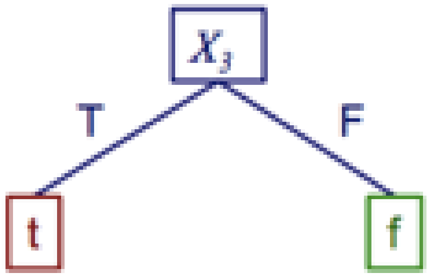
M_1



M_2

X_1	X_2	X_3	X_4	X_5	…	Y
t	t	t	t	t	…	t
t	t	t	f	t	…	t
t	t	t	t	f	…	t
t	f	f	t	f	…	f
f	t	f	f	t	…	f

Example 2: overfitting with noise-free data



$$Y = X_1 \wedge X_2$$

$P(X_3 = t) = 0.5$
$P(Y=t) = 0.66$

- What is the accuracy?
  - Accuracy($D_{training}$,M) = 100%
  - Accuracy(D_{true},M) = 50%

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	t	f	t	...	t
t	t	t	t	f	...	t
t	f	f	t	f	...	f
f	t	f	f	t	...	f

Example 2: overfitting with noise-free data

$$Y = X_1 \wedge X_2$$

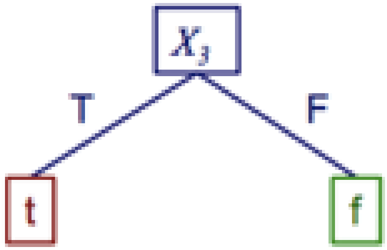
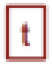
$\boxed{t}$

$P(X_3 = t) = 0.5$
$P(Y{=}t) = 0.66$

- What is the accuracy?
  - $\text{Accuracy}(D_{\text{training}}, M) = 60\%$
  - $\text{Accuracy}(D_{\text{true}}, M) = 66\%$

X_1	X_2	X_3	X_4	X_5	...	Y
t	t	t	t	t	...	t
t	t	t	f	t	...	t
t	t	t	t	f	...	t
t	f	f	t	f	...	f
f	t	f	f	t	...	f

Example 2: overfitting with noise-free data

		Training set accuracy	True accuracy	
M_1		100%	50%	M_1 is overfitting!
M_2		60%	66%	

- because the training set is a limited sample, there might be (combinations of) features that are correlated with the target concept by chance

Avoiding overfitting in DT learning

- two general strategies to avoid overfitting
  - *1. early stopping*: stop if further splitting not justified by a statistical test
    - Quinlan's original approach in ID3
  - *2. post-pruning*: grow a large tree, then prune back some nodes
    - more robust to myopia of greedy tree learning

Nearest-neighbor classification

Nearest-neighbor classification

- learning stage
  - given a training set ($\mathbf{x}^{(1)}$, $y^{(1)}$) ... ($\mathbf{x}^{(m)}$, $y^{(m)}$), do nothing
    - (it's sometimes called a *lazy learner*)

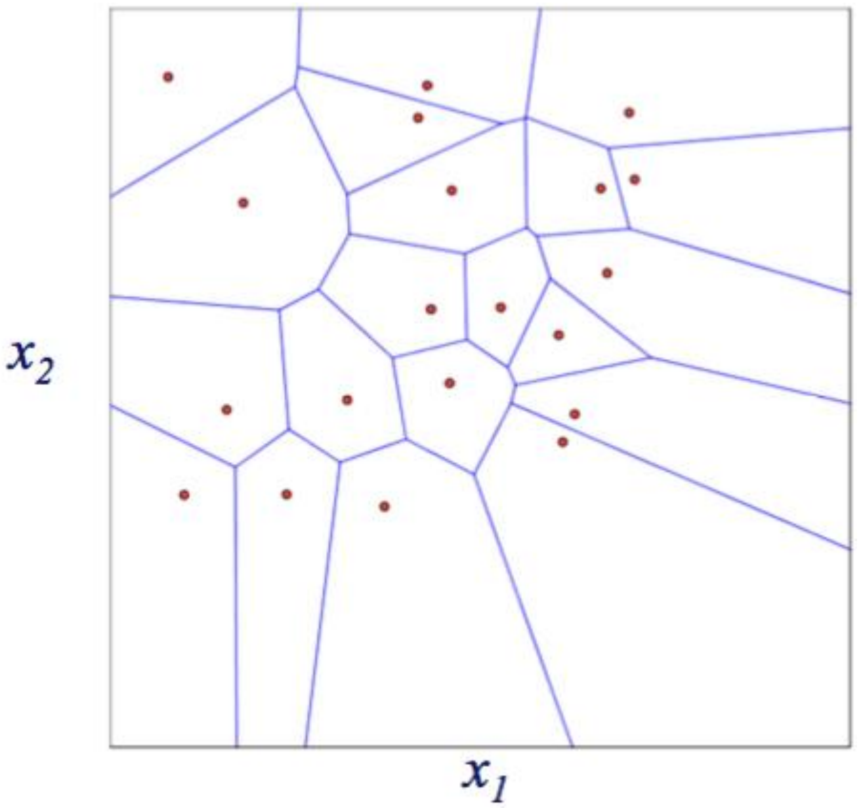
- classification stage
  - **given**: an instance $x^{(q)}$ to classify
  - find the training-set instance $x^{(i)}$ that is most similar to $x^{(q)}$
  - return the class value $y^{(i)}$

Nearest Neighbor

- **When to Consider**
  - Less than 20 attributes per instance
  - Lots of training data
- **Advantages**
  - Training is very fast
  - Learn complex target functions
  - Do not lose information
- **Disadvantages**
  - Slow at query time
  - Easily fooled by irrelevant attributes

The decision regions for nearest-neighbor classification

- Voronoi diagram: each polyhedron indicates the region of feature space that is in the nearest neighborhood of each training instance



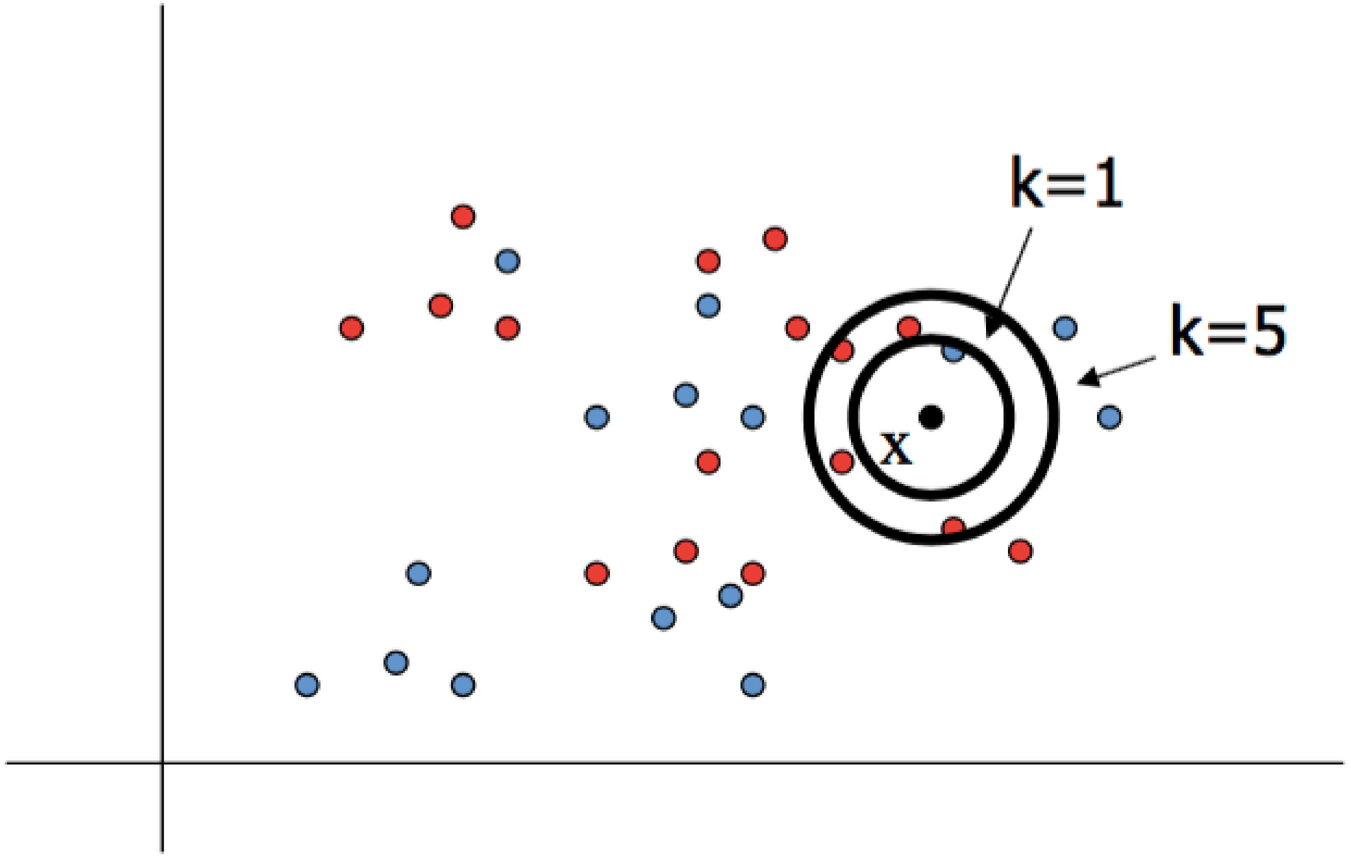
k-nearest-neighbor classification

- classification task
  - **given**: an instance $x^{(q)}$ to classify
  - find the k training-set instances $(\mathbf{x}^{(1)}, y^{(1)})\ldots(x^{(k)}, y^{(k)})$ that are the most similar to $x^{(q)}$
  - return the class value

$$\hat{y} \leftarrow \underset{v \in \text{values}(Y)}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, y^{(i)}) \qquad \delta(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

  - (i.e. return the class that have the most number of instances in the k training instances

To classify a new input vector x, examine the k-closest training data points to x and assign the object to the most frequently occurring class



How can we determine similarity/distance

- suppose all features are discrete
  - Hamming distance (or L^0 norm): count the number of features for which two instances differ

- Example: X = (Weekday, Happy?, Weather) Y = AttendLecture?
  - D : in the table
  - New instance: <Friday, No, Rain>
  - Distances = {2, 3, 1, 2}
  - For 1-nn, which instances should be selected?
  - For 2-nn, which instances should be selected?
  - For 3-nn, which instances should be selected?

v1	**v2**	**v3**	**y**
Wed	Yes	Rain	No
Wed	Yes	Sunny	Yes
Thu	No	Rain	Yes
Fri	Yes	Sunny	No

New datum

Fri	No	Rain	

How can we determine similarity/distance

- Example: X = (Weekday, Happy?, Weather) Y = AttendLecture?
  - New instance: <Friday, No, Rain>
  - For 3-nn, selected instances: {(<Wed, Yes, Rain>, No), (<Thu, No, Rain>, Yes), (<Fri, Yes, Sunny>, No)}
- Classification:

$$\hat{y} \leftarrow \underset{v \in \text{values}(Y)}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, y^{(i)})$$

  - v = Yes. $\sum_{i=1}^{k} \delta(v, y^{(i)}) = 0 + 1 + 0 = 1$
  - v = No. $\sum_{i=1}^{k} \delta(v, y^{(i)}) = 1 + 0 + 1 = 2$

So, which class this new instance should be in?

How can we determine similarity/distance

- suppose all features are continuous
  - Euclidean distance:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sqrt{\sum_f \left(x_f^{(i)} - x_f^{(j)}\right)^2}$$

where $x_f^{(i)}$ represents the f -th feature of $x^{(i)}$

  - Manhattan distance:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_f \left|x_f^{(i)} - x_f^{(j)}\right|$$

Recall the difference and similarity with L^p norm

How can we determine similarity/distance

- Example: X = (Height, Weight, RunningSpeed) Y = SoccerPlayer?
  - D: in the table
  - New instance: <185, 91, 13.0>
  - Suppose that Euclidean distance is used.
  - Is this person a soccer player?

v1	**v2**	**v3**	**y**
182	87	11.3	No
189	92	12.3	Yes
178	79	10.6	Yes
183	90	12.7	No

New datum

185	91	13.0	

How can we determine similarity/distance

- if we have a mix of discrete/continuous features:

$$d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sum_f \begin{cases} \left| x_f^{(i)} - x_f^{(j)} \right| \text{ if } f \text{ is continuous} \\ 1 - \delta\left(x_f^{(i)}, x_f^{(i)}\right) \text{if } f \text{ is discrete} \end{cases}$$

- typically want to apply to continuous features some type of normalization (values range 0 to 1) or standardization (values distributed according to standard normal)
- many other possible distance functions we could use ...

Standardizing numeric features

- given the training set D, determine the mean and stddev for feature x_i

$$\mu_i = \frac{1}{|D|}\sum_{d=1}^{|D|} x_i^{(d)} \qquad \sigma_i = \sqrt{\frac{1}{|D|}\sum_{d=1}^{|D|}\left(x_i^{(d)} - \mu_i\right)^2}$$

- standardize each value of feature x_i as follows

$$\hat{x}_i^{(d)} = \frac{x_i^{(d)} - \mu_i}{\sigma_i}$$

- do the same for test instances, using the same μ and σ derived from the *training* data