Introduction to Data Science

(Khoa học dữ liệu)

Khoat Than

Hanoi University of Science and Technology

khoattq@soict.hust.edu.vn

IT4142E, SOICT, HUST, 2019

Contents of the course

- Introduction to Data Science
- Data storage and processing
- Exploratory data analysis
- **Machine Learning**
- Big data analysis
- Visualization
- Natural language processing
- Computer vision
- Graph analysis
- Recommender system

Linear regression: introduction

- **Regression problem:** learn a function $y = f(\mathbf{x})$ from a given training data $\mathbf{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_M, y_M)\}$ such that $y_i \cong f(\mathbf{x}_i)$ for every i
  - Each observation of $\mathbf{x}$ is represented by a vector in an n-dimensional space, e.g., $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{in})^T$. Each dimension represents an *attribute/feature/variate*.
  - Bold characters denote vectors.
- Linear model: if $f(\mathbf{x})$ is assumed to be of linear form

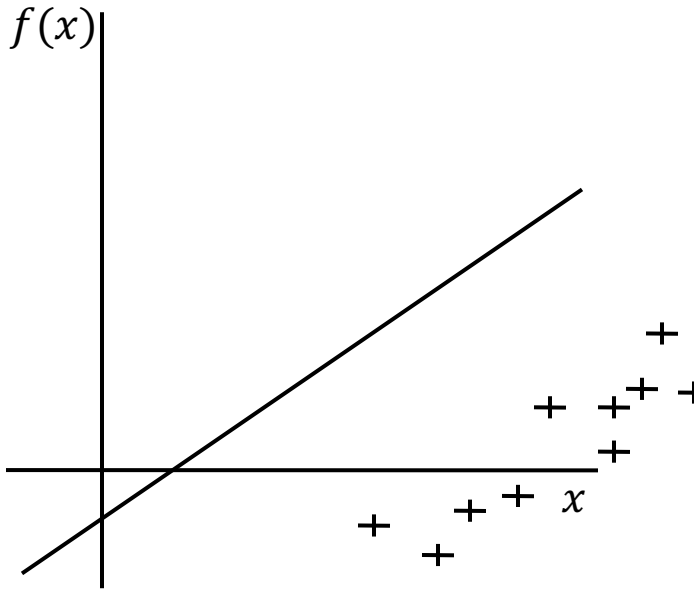
$$f(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_n x_n$$

  - $w_0, w_1, \ldots, w_n$ are the regression coefficients
- Note: learning a linear function is equivalent to learning the coefficient vector $\mathbf{w} = (w_0, w_1, \ldots, w_n)^T$.

Linear regression: example

- What is the best function?

0.13	-0.91
1.02	-0.17
3.17	1.61
-2.76	-3.31
1.44	0.18
5.28	3.36
-1.74	-2.46
7.93	5.56
...	...



Prediction

- For each observation $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$
  - The *true output*: c_x
    (but unknown for future data)
  - *Prediction* by our system/method:

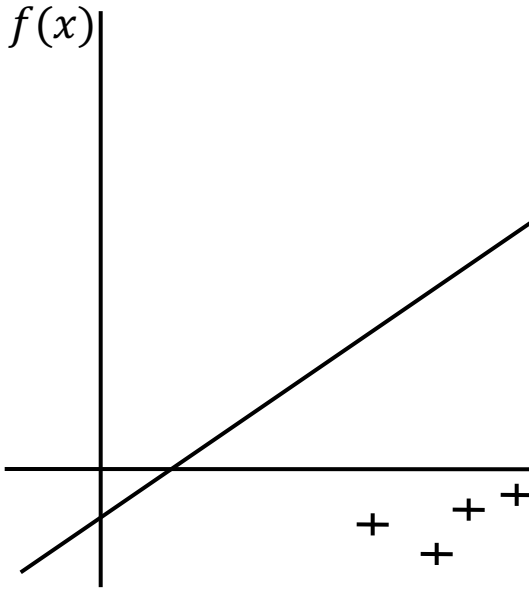
$$y_x = w_0 + w_1 x_1 + \ldots + w_n x_n$$

  - We often expect $y_x \cong c_x$.
- Prediction for future observation $\mathbf{z} = (z_1, z_2, \ldots, z_n)^T$
  - Use the previously learned function to make prediction

$$f(\mathbf{z}) = w_0 + w_1 z_1 + \ldots + w_n z_n$$

Learning a regression function

- Learning goal: *learn a function f* such that its prediction in the future is the best.*
  - The error of $|c_z - f(\mathbf{z})|$ is as small as possible for future **z**.
  - Generalization is best.
- Difficulty: infinite number of functions
  - How can we learn?
  - Is function f better than g?
- Use a measure to learn
  - *Loss function* or *generalization error* are often used to guide learning.



Loss function

- Definition:
  - The *error/loss* of the prediction for an observation $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$

$$r(\mathbf{x}) = [c_x - f^*(\mathbf{x})]^2 = (c_x - w_0 - w_1 x_1 - \ldots - w_n x_n)^2$$

  - The *expected loss* over the whole space:

$$E = \mathbf{E}_x[r(\mathbf{x})] = \mathbf{E}_x[c_x - f^*(\mathbf{x})]^2$$

($\mathbf{E}_x$ is the expectation over $\mathbf{x}$)

Cost, risk

- The goal of learning is to find f* that minimizes the expected loss:

$$f^* = \arg\min_{f \in \boldsymbol{H}} \boldsymbol{E}_x \left[r(\boldsymbol{x}) \right]$$

  - **H** is the space of linear functions.
- But, we cannot work directly with this problem during the learning phase. (why??)

Empirical loss

- We can only observe a set of training data **D** = {($\mathbf{x}_1$, y_1), ($\mathbf{x}_2$, y_2), …, ($\mathbf{x}_M$, y_M)}, and have to learn f from **D**.

- Empirical loss (residual sum of squares):

$$RSS(f) = \sum_{i=1}^{M} (y_i - f(\boldsymbol{x}_i))^2 = \sum_{i=1}^{M} (y_i - w_0 - w_1 x_{i1} - \cdots - w_n x_{in})^2$$

  - RSS/M is an approximation to $\mathbf{E}_x[r(\mathbf{x})]$.
- Many learning algorithms base on this RSS and its variants.

Methods: ordinary least squares (OLS)

- Given **D**, we find f* that minimizes RSS:

$$f^* = \arg\min_{f \in \boldsymbol{H}} RSS(f)$$

$$\Leftrightarrow \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{M} (y_i - w_0 - w_1 x_{i1} - \cdots - w_n x_{in})^2 \quad (1)$$

- This method is often known as *ordinary least squares (OLS)*.
- Find **w*** by taking the gradient of RSS and the solving the equation RSS'=0. We have:

$$\boldsymbol{w}^* = (\boldsymbol{A}^T \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{y}$$

  - Where **A** is the data matrix of size Mx(n+1), whose the i^{th} row is $\mathbf{A}_i = (1, x_{i1}, x_{i2}, \ldots, x_{in})$; $\mathbf{B}^{-1}$ is the inversion of matrix **B**; $\mathbf{y} = (y_1, y_2, \ldots, y_M)^T$.
  - Note: we assume that $\mathbf{A}^T\mathbf{A}$ is invertible.

Methods: OLS

- Input: $\mathbf{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_M, y_M)\}$
- Output: $\mathbf{w}^*$
- Learning: compute

$$\boldsymbol{w}^* = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{y}$$

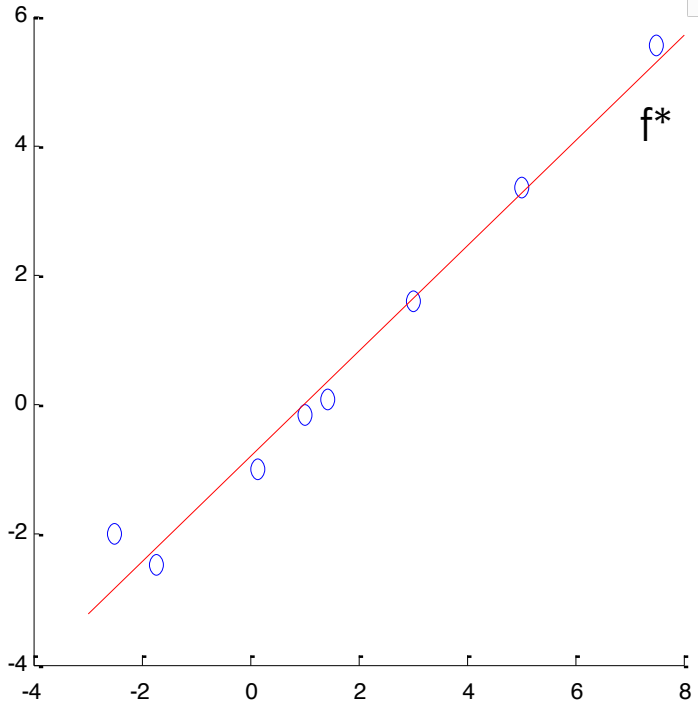
  - Where $\mathbf{A}$ is the data matrix of size Mx(n+1), whose the i^{th} row is $\mathbf{A}_i = (1, x_{i1}, x_{i2}, \ldots, x_{in})$; $\mathbf{B}^{-1}$ is the inversion of matrix $\mathbf{B}$; $\mathbf{y} = (y_1, y_2, \ldots, y_M)^T$.
  - Note: we assume that $\mathbf{A}^T\mathbf{A}$ is invertible.
- Prediction for a new $\mathbf{x}$:

$$y_x = w_0^* + w_1^* x_1 + \cdots + w_n^* x_n$$

Methods: OLS example

x	y
0.13	-1
1.02	-0.17
3	1.61
-2.5	-2
1.44	0.1
5	3.36
-1.74	-2.46
7.5	5.56

$f^*(x) = 0.81x - 0.78$



Methods: limitations of OLS

- OLS cannot work if $\mathbf{A}^T\mathbf{A}$ is not invertible
  - If some columns (attributes/features) of **A** are dependent, then **A** will be singular and therefore $\mathbf{A}^T\mathbf{A}$ is not invertible.
- OLS requires considerable computation due to the need of computing a matrix inversion.
  - Intractable for the very high dimensional problems.
- OLS very likely tends to overfitting, because the learning phase just focuses on minimizing errors on the training data.

Methods: Ridge regression (1)

- Given **D** = {($\mathbf{x}_1$, y_1), ($\mathbf{x}_2$, y_2), …, ($\mathbf{x}_M$, y_M)}, we solve for:

$$f^* = \arg\min_{f \in \boldsymbol{H}} RSS(f) + \lambda \|\boldsymbol{w}\|_2^2$$

$$\Leftrightarrow \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{M} (y_i - \boldsymbol{A_i w})^2 + \lambda \sum_{j=0}^{n} w_j^2 \qquad (2)$$

  - Where $\mathbf{A}_i$ = (1, x_{i1}, x_{i2}, …, x_{in}) is composed from $\mathbf{x}_i$; and λ is a regularization constant (λ> 0).

- The regularization/penalty term $\lambda \|\boldsymbol{w}\|_2^2$
  - Limits the magnitute/size of **w*** (i.e., reduces the search space for f*).
  - Helps us to trade off between *the fitting of f on* ***D*** and *its generalization* on future observations.

Methods: Ridge regression (2)

- We solve for **w*** by taking the gradient of the objective function in (2), and then zeroing it. Therefore we obtain:

$$\boldsymbol{w}^* = (\boldsymbol{A}^T\boldsymbol{A} + \lambda \boldsymbol{I}_{n+1})^{-1}\boldsymbol{A}^T\boldsymbol{y}$$

  - Where **A** is the data matrix of size Mx(n+1), whose the i^{th} row is $\mathbf{A}_i = (1, x_{i1}, x_{i2}, \ldots, x_{in})$; $\mathbf{B}^{-1}$ is the inversion of matrix **B**; $\mathbf{y} = (y_1, y_2, \ldots, y_M)^T$; $\mathbf{I}_{n+1}$ is the identity matrix of size n+1.
- Compared with OLS, Ridge can
  - Avoid the cases of singularity, unlike OLS. Hence Ridge always works.
  - Reduce overfitting.
  - But error in the training data might be greater than OLS.
- Note: *the predictiveness of Ridge depends heavily on the choice of the hyperparameter* λ.

Methods: Ridge regression (3)

- Input: **D** = {($\mathbf{x}_1$, y_1), ($\mathbf{x}_2$, y_2), …, ($\mathbf{x}_M$, y_M)} and λ>0
- Output: **w***
- Learning: compute

$$\boldsymbol{w}^* = (\boldsymbol{A}^T\boldsymbol{A} + \lambda \boldsymbol{I}_{n+1})^{-1}\boldsymbol{A}^T\boldsymbol{y}$$

- Prediction for a new **x**:

$$y_x = w_0^* + w_1^* x_1 + \cdots + w_n^* x_n$$

- Note: to avoid some negative effects of the magnitute of y on covariates **x**, one should remove w_0 from the penalty term in (2). In this case, the solution of **w*** should be modified slightly.

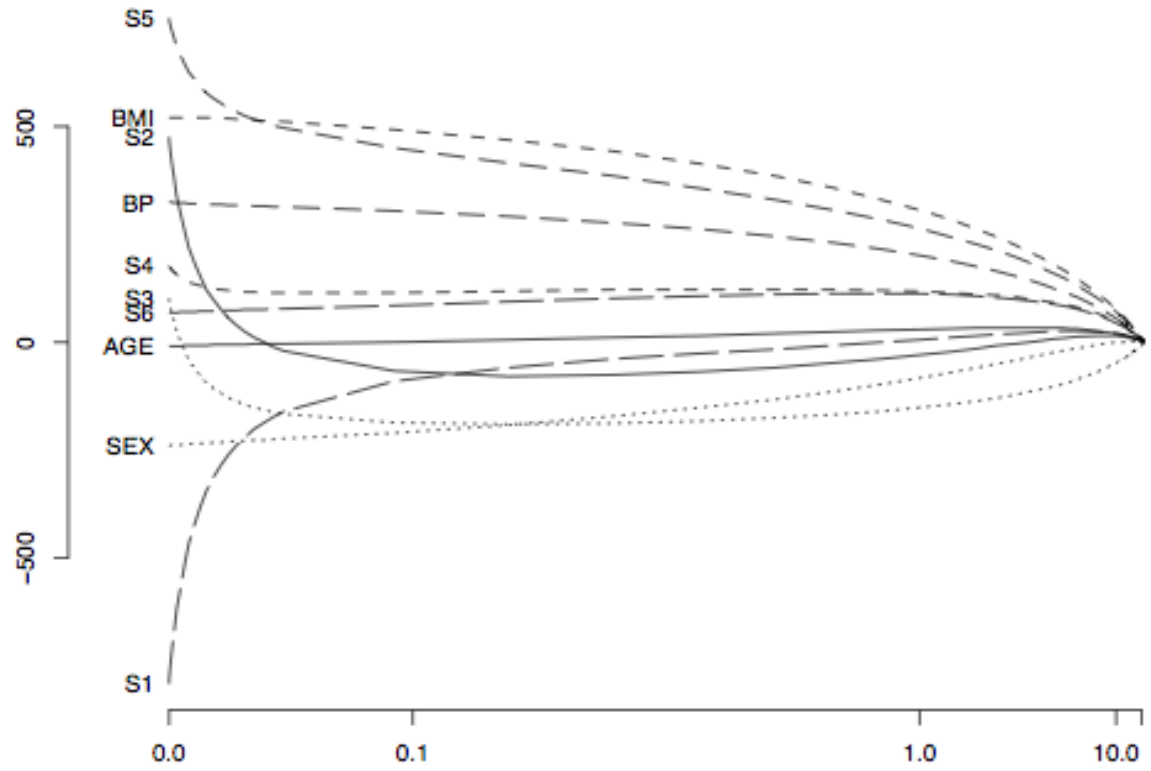
An example of using Ridge and OLS

- We used a training set **D** of 67 observations on prostate cancer, each was represented with 8 attributes. Ridge and OLS were learned from **D**, and then predicted 31 new observations.

w	Ordinary Least squares	Ridge
0	2.465	2.452
lcavol	0.680	0.420
lweight	0.263	0.238
age	−0.141	−0.152
lbph	0.210	0.002
svi	0.305	0.094
lcp	−0.288	−0.051
gleason	−0.021	0.232
pgg45	0.267	−0.056
Test RSS	**0.521**	**0.492**

Effects of in λ Ridge regression

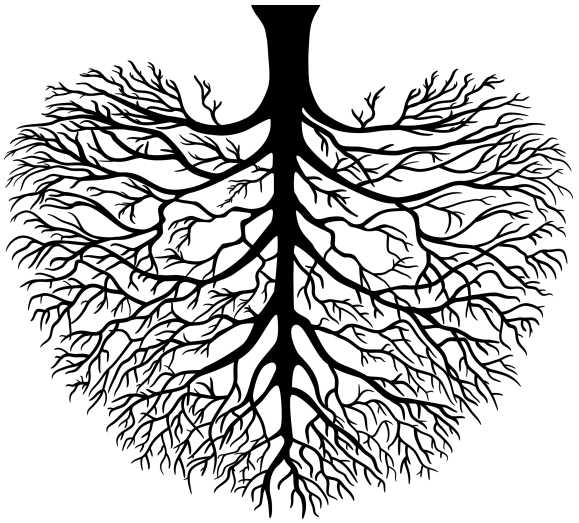
- **W*** = (w_0, S1, S2, S3, S4, S5, S6, AGE, SEX, BMI, BP) changes as the regularization constant λ changes.



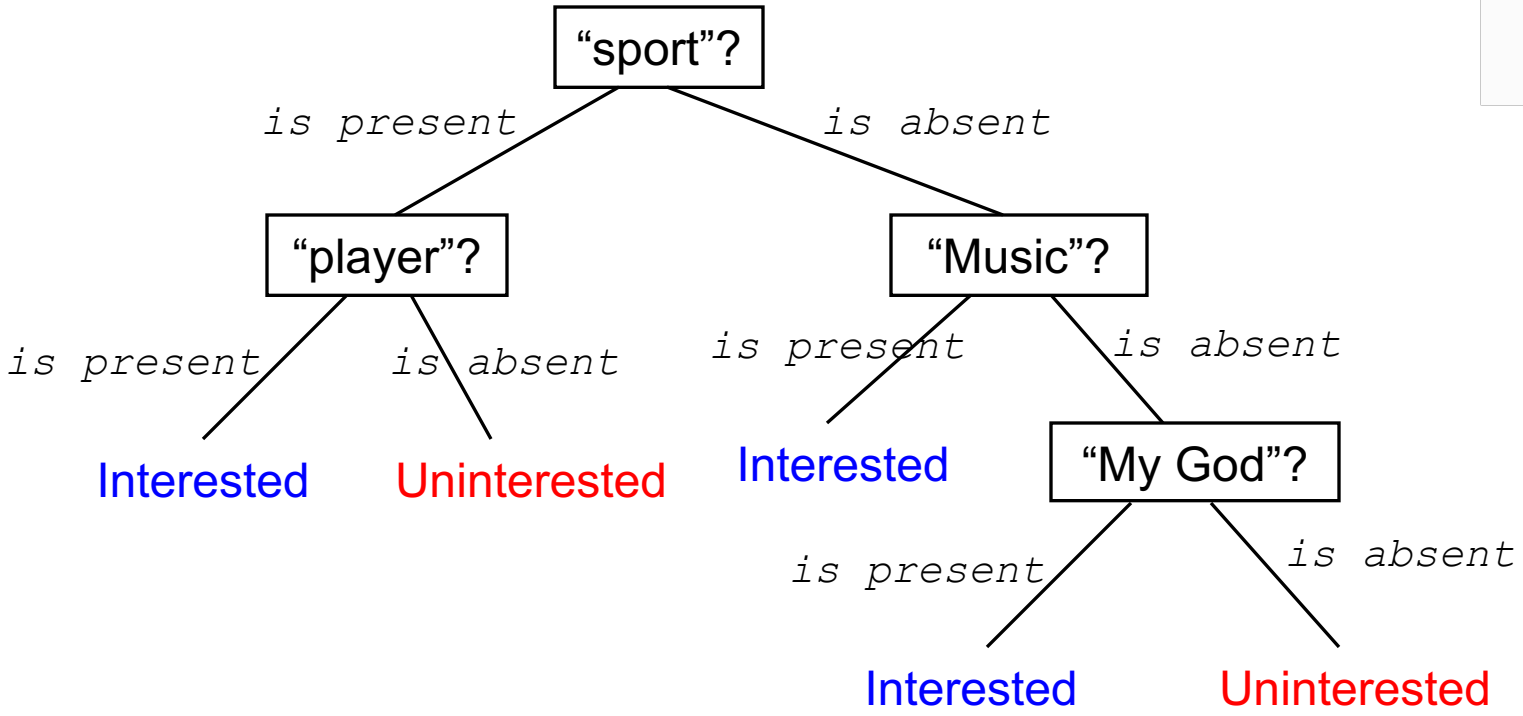
Classification

1. Decision tree

- Decision tree
  - To represent a function by using a tree.
- Each decision tree can be interpreted as a set of rules of the form: IF-THEN
- Decision trees have been used in many practical applications.

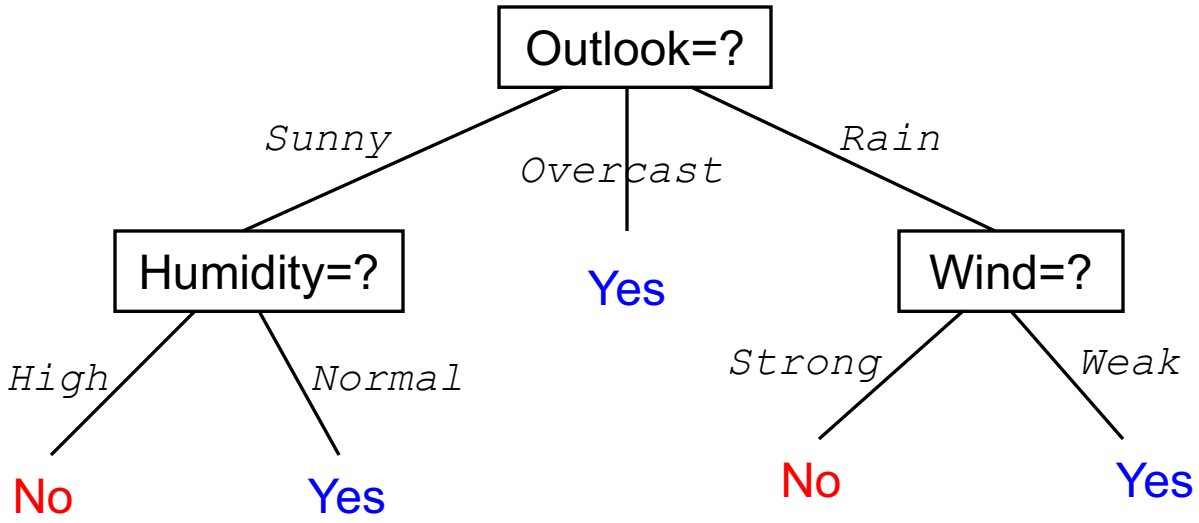


Examples of a decision tree (1)



- (…,"sport",…,"player",…) → Interested
- (…,"My God",…) → Interested
- (…,"sport",…) → Uninterested

Examples of a decision tree (2)



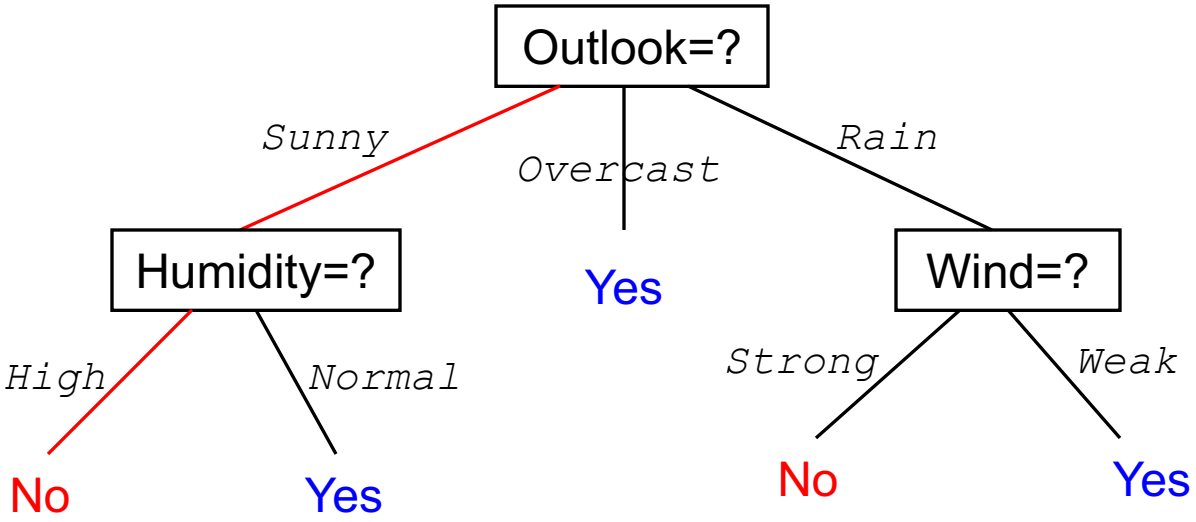
- (Outlook=`Overcast`, Temperature=`Hot`, Humidity=`High`, Wind=`Weak`) → Yes
- (Outlook=`Rain`, Temperature=`Mild`, Humidity=`High`, Wind=`Strong`) → No
- (Outlook=`Sunny`, Temperature=`Hot`, Humidity=`High`, Wind=`Strong`) → No

Tree representation (1)

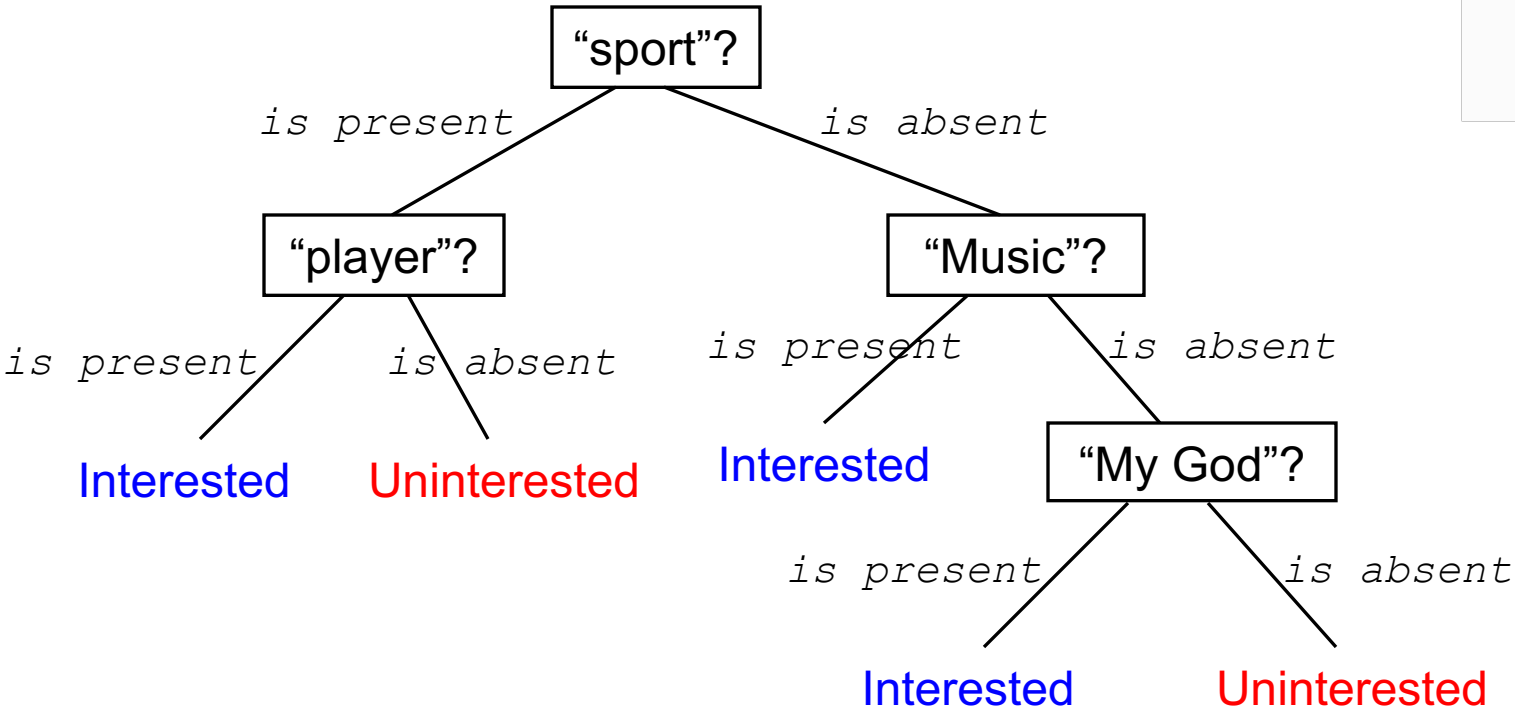
- *Each internal node* represents an attribute for testing the incoming data.
- Each *branch/subtree of a node* corresponds to a value of the attribute of that node.
- Each leaf node represents a class label.
- Once a tree has been learned, *we can predict the label for a new instance by using its attributes to travel from the root downto a leaf.*
  - The label of the leaf will be used to assign to the new instance.

Tree representation (2)

- Each path from the root to a leaf is a *conjunction/AND* of the attribute tests.
- A decision tree itself is a *disjunction/OR of those conjunctions*.



Representation by a disjunction



[("sport" is present) ∧ ("player" is present)] ∨

[("sport" is absent) ∧ ("Music" is present)] ∨

[("sport" is absent) ∧ ("Music" is absent) ∧ ("My God" is present)]

2. Learning a decision tree by ID3

- ID3 (Iterative Dichotomiser 3) is a greedy algorithm which was proposed by Ross Quinlan in 1986.
- It uses the top-down scheme.
- At each node N, select a test attribute A which can help us best do classification for the data in N.
  - *Generate a branch for each value of A, and then separate the data into its branches accordingly.*
- Grow the tree until:
  - *It classifies correctly all the training data; or*
  - *All the attributes are used.*
- Note: each attribute can only appear at most once in any path of the tree.

The ID3 algorithm

ID3_alg(*Training_Set, Class_Labels, Attributes*)

Generate the Root of the tree

If all of *Training_Set* belong to class c, then Return Root as leaf with label c

If *Attributes* is empty, then
    Return Root as leaf with label c = **Majority_Class_Label**(*Training_Set*)

A ← a set of *Attributes* that are best discriminative for *Training_Set*

Let A be the test attributes of Root

For each value v of A

    Generate a branch of Root which corresponds with v.

    Determine $Training\_Set_v$ = { x in *Training_Set* | x_A = v}

    If ($Training\_Set_v$ is empty) Then

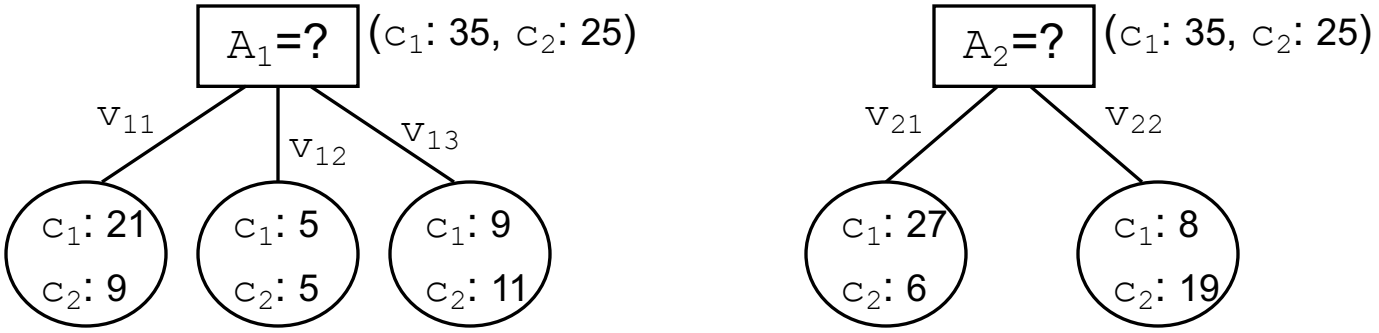
        Generate a leaf with class label c = **Majority_Class_Label**(*Training_Set*)

    Else
        Generate a subtree by **ID3_alg**($Training\_Set_v$, Class_Labels, Attributes \{A})

Return Root

How to choose the test attributes?

- At each node, how can we choose a set of test attributes?
  - These attributes should be *discriminative*, i.e., can help us classify well the data inside that node.
- How to know an attribute to be discriminative?
- Ex: assuming 2 classes in the data, which of A_1 and A_2 should be selected as the test attribute?



- ***Information gain*** can help.

Information gain: entropy

- Entropy measures the impurity/inhomogeneity of a set.
- Entropy of a set S with c classes can be defined as:

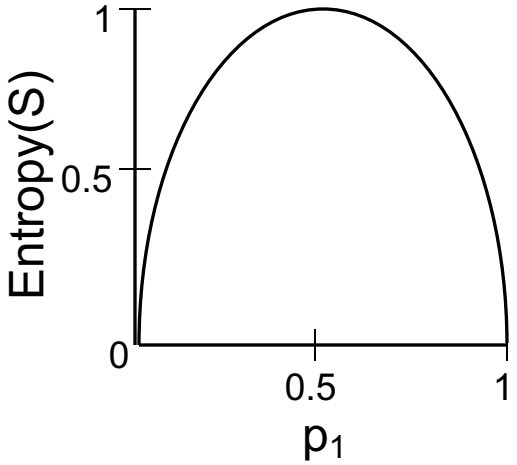
$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

  - Where p_i is the proportion of instances with class label i in S; and $0.\log_2 0 = 0$ as a convention; $p_1+p_2+\ldots+p_c = 1$
- For 2 classes: entropy(S) = $- p_1\log_2 p_1 - p_2\log_2 p_2$
- Meanings of entropy in Information Theory:
  - Entropy shows the *number of bits on average* to encode a class of S.
  - Entropy of a message measures the *average amount of information* contained in that message.
  - Entropy of a random variable x measures the *unpredictability* of x.

Information gain: entropy example

- S consists of 14 examples for which 9 belong to class c_1 and 5 belong to class c_2.
- So the entropy of S is:

  Entropy(S)
  $= -(9/14).\log_2(9/14) -(5/14).\log_2(5/14)$
  ≈ 0.94



- Entropy = 0 if all examples in S have the same label.
- Entropy = 1 if the two classes in S are equal in size.
- Otherwise, entropy will always belong to (0, 1).

Information gain

- *Information gain* of an attribute in S:
  - Measures the reduction of entropy if we divide S into subsets according to that attribute.
- Information gain of attribute A in S is defined as:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

  - Where Values(A) is the set of all values of A, and $S_v = \{\mathbf{x} \mid \mathbf{x} \text{ in } S, \text{ and } x_a = v\}$
- The **second term** in Gain(S,A) measures the *information remained* when S is divided into subsets according to the values of A.
- Meaning of Gain(S,A): *the average amount of information is lost when dividing S according to A.*

Information gain: example (1)

- A set S of observations about a person playing tennis.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

[Mitchell, 1997]

Information gain: example (2)

- What is Gain(S, Wind)?
- Wind has two values: Strong & Weak
- S = {9 examples with label Yes, 5 examples with label No}
- S_{Weak} = {6 examples with label Yes and 2 examples with label No, having Wind=Weak}
- S_{Strong} = {3 examples with label Yes, 3 examples with label No, having Wind=Strong}
- So:

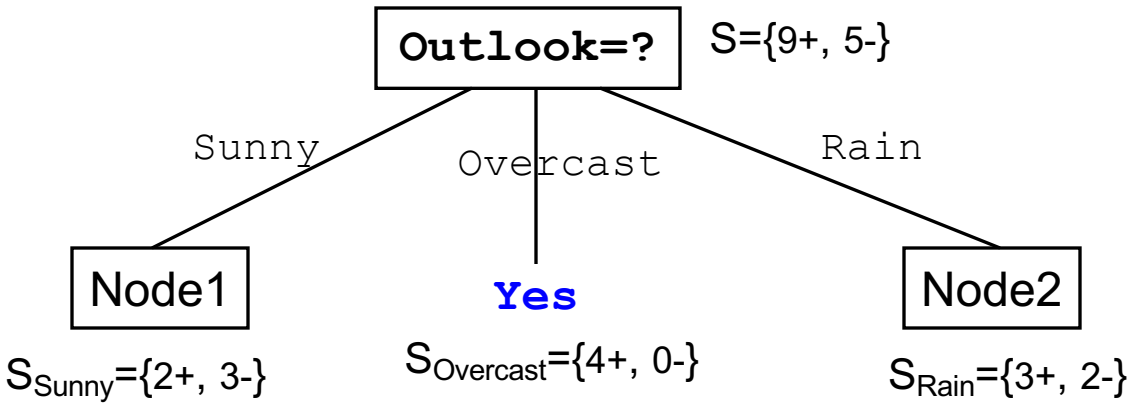
$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S) - \frac{8}{14} Entropy(S_{Weak}) - \frac{6}{14} Entropy(S_{Strong})$$

$$= 0.94 - \frac{8}{14} * 0.81 - \frac{6}{14} * 1 = 0.048$$

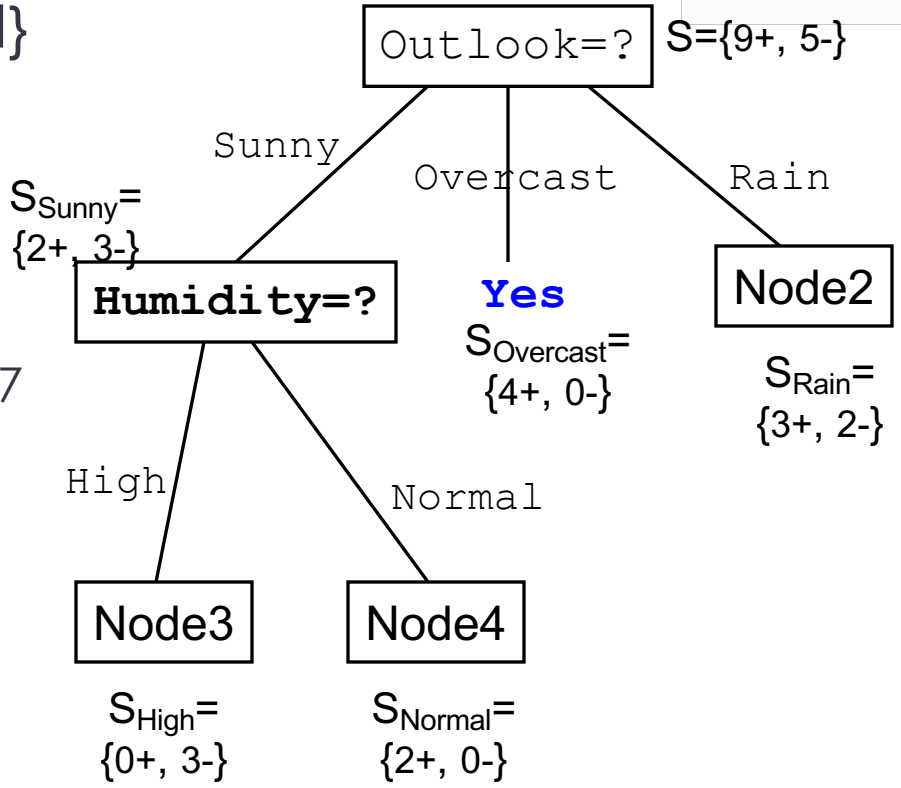
ID3: example (1)

- At the root, which one of {Outlook, Temperature, Humidity, Wind} should be the test attribute?
  - Gain(S, **Outlook**) = ... = **0.246**
  - Gain(S, Temperature) = ... = 0.029
  - Gain(S, Humidity) = ... = 0.151
  - Gain(S, Wind) = ... = 0.048
- So, Outlook is selected as the test attribute.



ID3: example (2)

- At Node1, which one of {Temperature, Humidity, Wind} should be the test attribute?
  - Note: Outlook is left out
  - Gain(S_{Sunny}, Wind) = ... = 0.019
  - Gain(S_{Sunny}, Temperature) =...= 0.57
  - Gain(S_{Sunny}, **Humidity**) = ... = **0.97**
- So, Humidity is selected to divide Node1.

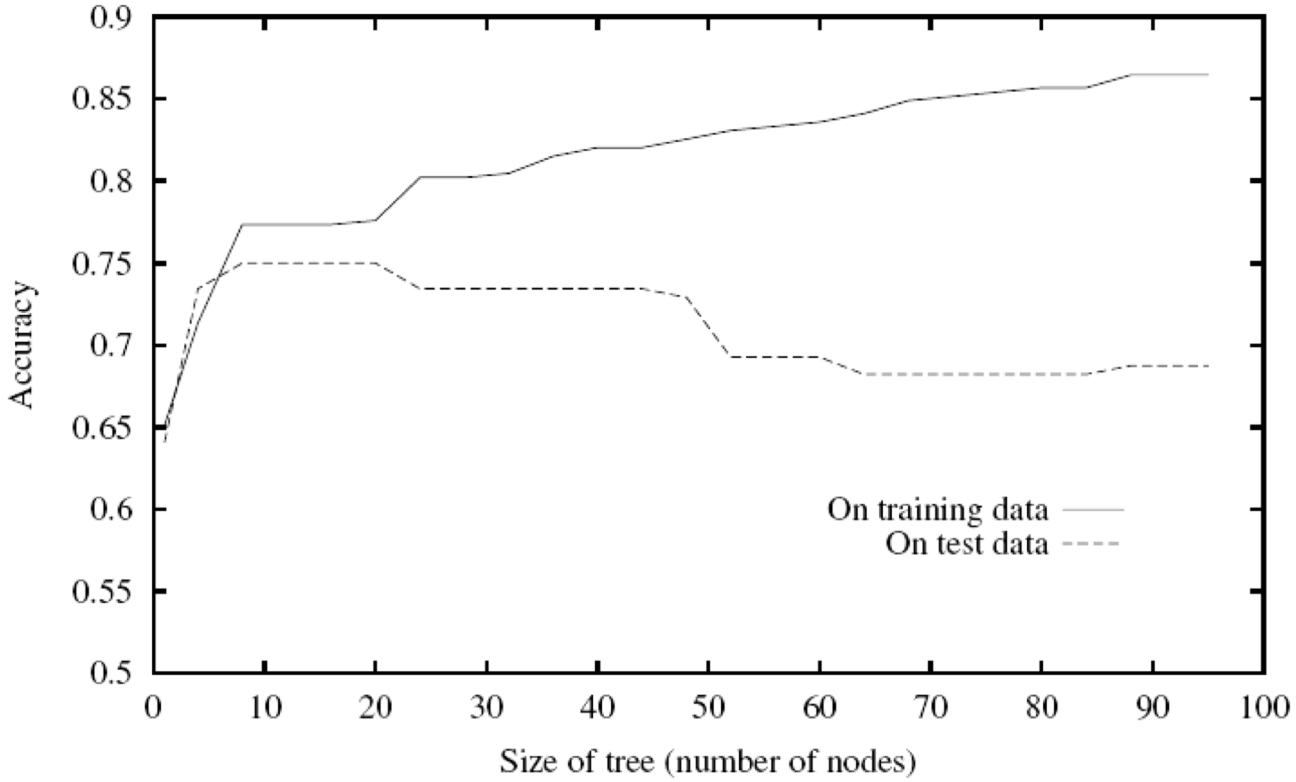


3. Some issues of ID3

- The learnt trees may overfit the training data.
- How to work with real attributes?
  - Many applications have real inputs.
- Is there any better measure than information gain?
- How to deal with missing values?
  - Missing-value is an inherent problem in many practical applications.
- How to enclose the cost of attributes in ID3?

Overfitting in ID3 (2)

- An example: continuing to grow the tree can improve the accuracy on the training data, but perform badly on the test data.



[*Mitchell, 1997*]

Overfitting: solutions

- 2 solutions:
  - *Stop learning early:* prevent the tree before it fits the training data perfectly.
  - *Prune the full tree:* grow the tree to its full size, and then post prune the tree.
- It is hard to decide when to stop learning.
- Post-pruning the tree empirically results in better performance. But
  - How to decide the good size of a tree?
  - When to stop pruning?
- We can use a validation set to do pruning, such as, *reduced-error pruning*, and *rule-post pruning*.

ID3: missing or real values

- How to work with real attributes?
  - Real attributes/features are popular in practice.
  - One way is to *discretization*, i.e., transforming a real attribute into a discrete one by dividing the domain of that attribute into a set of intervals.
    Ex: [0, 1] → {[0, 0.25); [0.25, 0.5); [0.5, 0.75); [0.75, 1]}
- How to deal with missing values?
  - Missing values are inherent in practical applications.
  - An observation **x** may not have a value x_A.
  - *Solution 1:* fill in x_A as the most popular value of A in the training data.
  - *Solution 2:* fill in x_A as the most popular value of A in the training data which belong to the same class with **x**.

5. Random forests

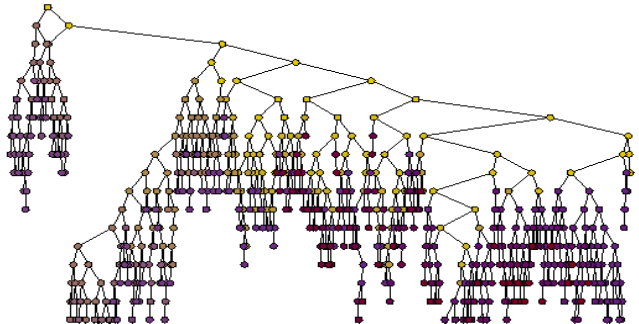
- Random forests (RF) is a method by Leo Breiman (2001) for both classification and regression.
- **Main idea:** prediction is based on combination of many decision trees, by *taking the average of all individual predictions.*
- Each tree in RF is simple but random.
- Each tree is grown differently, depending on the choices of the attributes and training data.

5. Random forests

- RF currently is one of the most popular and accurate methods [Fernández-Delgado et al., 2014]
  - It is also very general.
- RF can be implemented easily and efficiently.
- It can work with problems of very high dimensions, without overfitting ☺
- However, little is known about its theoretical properties ☹

5. RF: three basic ingredients

- **Randomization and no pruning:**
  - For each tree and at each node, we select randomly a subset of attributes.
  - Find the best split, and then grow appropriate subtrees.
  - Every tree will be grown to its largest size without pruning.
- **Combination:** each prediction later is made by taking the average of all predictions of individual trees.
- **Bagging:** the training set for each tree is generated by sampling (with replacement) from the original data.



5. RF: algorithm

- **Input**: training data D
- **Learning**: grow K trees as follows
  - Generate a training set D_i by sampling with replacement from D.
  - Learn the i^{th} tree from D_i:
    - At each node:
      - Select randomly a subset S of attributes.
      - Split the node into subtrees according to S.
    - Grow this tree upto its largest size without pruning.
- **Prediction**: taking the average of all predictions from the individual trees.

5. RF: practical performance

- RF is extensively compared with other methods
  - By Fernández-Delgado et al. (2014).
  - Using 55 different problems.
  - Using average accuracy (μ^P) as a measure.

No.	Classifier	μ^P	No.	Classifier	μ^P
1	rf_t	91.1	11	Bagging_LibSVM_w	89.9
2	parRF_t	91.1	12	RandomCommittee_w	89.9
3	svm_C	90.7	13	Bagging_RandomTree_w	89.8
4	RRF_t	90.6	14	MultiBoostAB_RandomTree_w	89.8
5	RRFglobal_t	90.6	15	MultiBoostAB_LibSVM_w	89.8
6	LibSVM_w	90.6	16	MultiBoostAB_PART_w	89.7
7	RotationForest_w	90.5	17	Bagging_PART_w	89.7
8	C5.0_t	90.5	18	AdaBoostM1_J48_w	89.5
9	rforest_R	90.3	19	Bagging_REPTree_w	89.5
10	treebag_t	90.2	20	MultiBoostAB_J48_w	89.4

Evaluation of analysis results

Assessing performance (1)

- *Theoretical evaluation:* study some theoretical properties of a method/model with some explicit mathematical proofs.
  - How many training instances are enough?
  - What is the expected accuracy of prediction?
  - Noise-resistance? …
- *Experimental evaluation:* observe the performance of a method in practical situations, using some datasets and a performance measure. Then make a summary from those experiments.
- We will discuss experimental evaluation in this lecture.

Assessing performance (2)

- ***Model assessment:*** *we need to evaluate the performance of a method/model, only based on a given observed dataset D.*
- Evaluation strategies:
  - To obtain a reliable assessment on performance.
- Evaluation measures:
  - To measure performance quantitatively.

2. Some evaluation techniques

- Hold-out
- Stratified sampling
- Repeated hold-out
- Cross-validation

Hold-out (random splitting)

- The observed dataset D is randomly splitted into 2 non-overlapping subsets:
  - D_{train}: used for training
  - D_{test}: used to test performance

- Note that:
  - No instance of D_{test} is used in the training phase.
  - No instance of D_{train} is used in the test phase.
- Popular split: $|D_{train}| = (2/3).|D|$, $|D_{test}| = (1/3).|D|$
- This technique is suitable when D is of large size.

Stratified sampling

- For small or imbalanced datasets, random splitting might result in a training dataset which are not representative.
  - A class in D_{train} might be empty or have few instances.
- *We should split D so that the class distribution in D_{train} is similar with that in D.*
- Stratified sampling fulfills this need:
  - *We randomly split each class of D into 2 parts:* one is for D_{train}, and the other is for D_{test}.
  - for each class:

- Note that this technique cannot be applied to regression and unsupervised learning.

Repeated hold-out

- We can do hold-out many times, and then take the average result.
  - Repeat hold-out n times. The i^{th} time will give a performance result p_i. The training data for each hold-out should be different from each other.
  - Take the average $p = \text{mean}(p_1, \ldots, p_n)$ as the final quality.
- Advantages?
- Limitations?

Cross-validation

- In repeated hold-out: there are overlapping between two training/testing datasets. It might be redundant.
- *K-fold cross-validation:*
  - *Split D into K equal parts which are non-overlapping.*
  - *Do K runs (folds): at each run, one part is used for testing and the remaining parts are used for training.*
  - *Take the average as the final quality from K individual runs.*

- Popular choices of K: 10 or 5
- It is useful to combine this technique with stratified sampling.
- This technique is suitable for small/average datasets.

3. Model selection

- An ML method often has a set of hyperparameters that require us to select suitable values a priori.
  - Ridge regression: λ
  - Random forest: number of trees
- How to choose a good value?
- ***Model selection:*** *given a dataset D, we need to choose a good setting of the hyperparameters in method (model) A such that the function learned by A generalizes well.*
- A validation set T_{valid} is often used to find a good setting.
  - It is a subset of D.
  - A good setting should help the learned function to predict well on T_{valid}.

Model selection: using hold-out

- Given an observed dataset D, we can **select** a good value for hyperparamerter λ as follows:
  - *Select a finite set S which contains all potential values of λ.*
  - *Select a performance measure P.*
  - *Randomly split D into 2 non-overlapping subsets: D_{train} and T_{valid}*
  - *For each $\lambda \in S$: train the system given D_{train} and λ. Measure the quality on T_{valid} to get P_λ.*
  - *Select the best λ* which corresponds to the best P_λ.*
- It is often beneficial to learn again from D given λ* to get a better function.
- Hold-out can be replaced with other techniques e.g., sampling, cross-validation.

4. Model assessment and selection

- Given an observed dataset D, we need to **select** a good value for hyperparamerter λ and **evaluate** the overall performance of a method A:
  - *Select a finite set S which contains all potential values of λ.*
  - *Select a performance measure P.*
  - *Split D into 3 non-overlapping subsets: D_{train}, T_{valid} and T_{test}*
  - *For each $\lambda \in S$: train the system given D_{train} and λ. Measure the quality on T_{valid} to get P_λ.*
  - *Select the best λ* which corresponds to the best P_λ.*
  - Train the system again from $D_{train} \cup T_{valid}$ given λ*.
  - Test performance of the system on T_{test}.
- Hold-out can be replaced with other techniques.

5. Performance measures

- Accuracy:
  - Percentage of correct predictions on testing data.
- Efficiency:
  - The cost in time and storage when learning/prediction.
- Robustness:
  - The ability to reduce possible affects by noises/errors/missings.
- Scalability:
  - The relation between the performance and training size.
- Complexity:
  - The comlexity of the learned function.
- …

Accuracy

- Classification:

$$Accuracy = \frac{number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

- Regression: (MAE – mean absolute error)

$$MAE = \frac{1}{|D_{test}|} \sum_{x \in D_{test}} |o(x) - y(x)|$$

  - o(x) is the prediction for an instance x.
  - y(x) is the true value.

Precision and Recall (1)

- These two measures are often used for classification
- **Precision** for class c_i:
  - Percentage of correct instances, among all that are assigned to c_i.

$$Precision(c_i) = \frac{TP_i}{TP_i + FP_i}$$

- **Recall** for class c_i:
  - Percentage of instances in c_i that are correctly assigned to c_i.

$$Recall(c_i) = \frac{TP_i}{TP_i + FN_i}$$

- TP_i: the number of instances that are assigned correctly to class c_i.
- FP_i: the number of instances that are assigned incorrectly to class c_i.
- FN_i: the number of instances inside c_i that are assigned incorrectly to another class.
- TN_i: the number of instances outside c_i that are not assigned to class c_i.

Precision and Recall (2)

- To give an overall summary, we can take an average from individual classes.
- Micro-averaging:

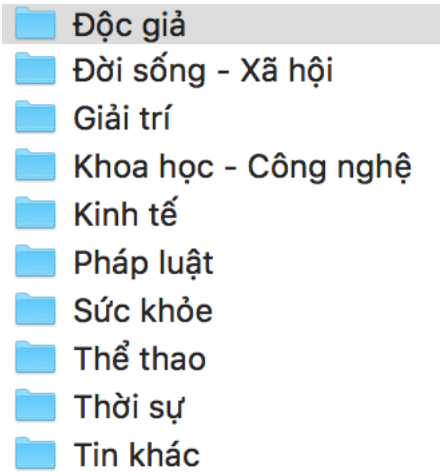
$$Precision = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FP_i)} \qquad Recall = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FN_i)}$$

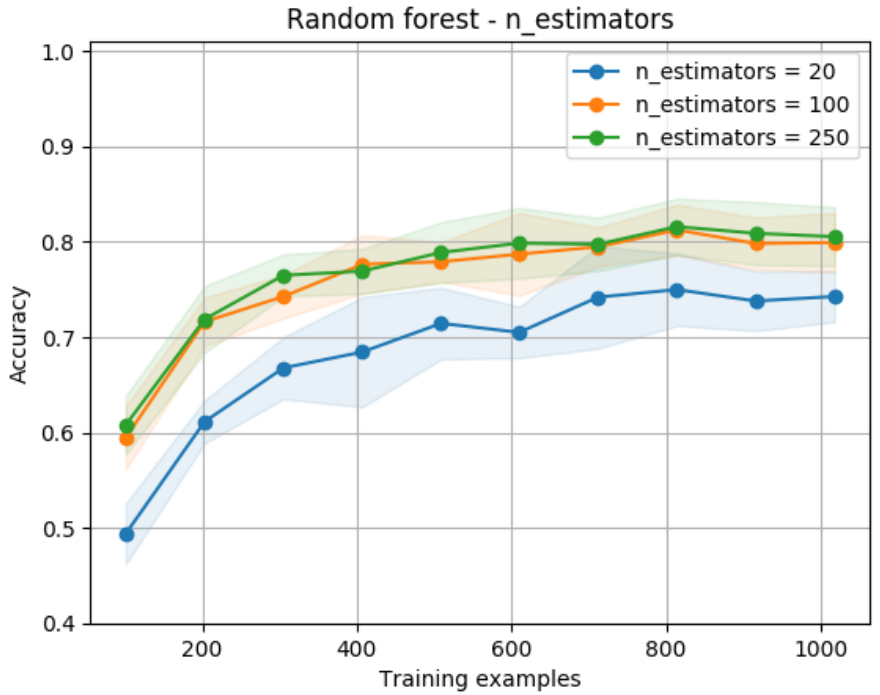
- Macro-averaging:

$$Precision = \frac{\sum_{i=1}^{|C|} Precision(c_i)}{|C|} \qquad Recall = \frac{\sum_{i=1}^{|C|} Recall(c_i)}{|C|}$$

Example: select parameters

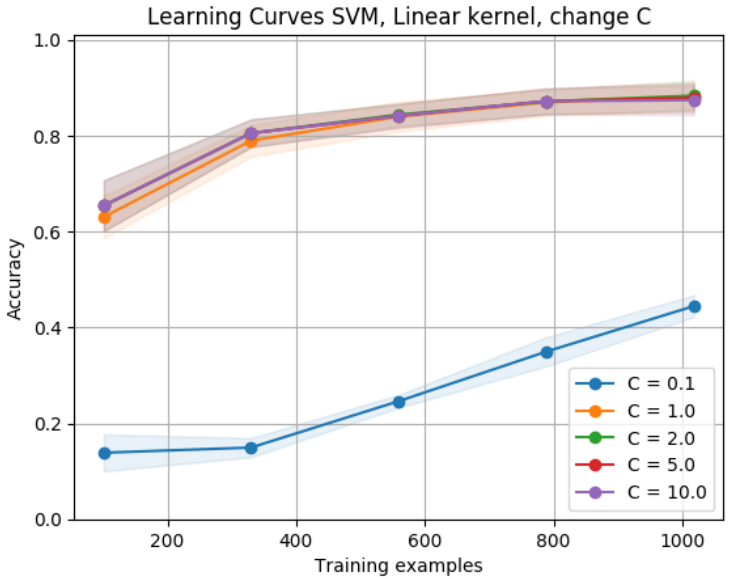
- Random forest for news classification
  - Parameter: n_estimates (number of trees)
- Dataset: *1135 news, 10 classes, vocabulary of 25199 terms*
- 10-fold cross-validation is used

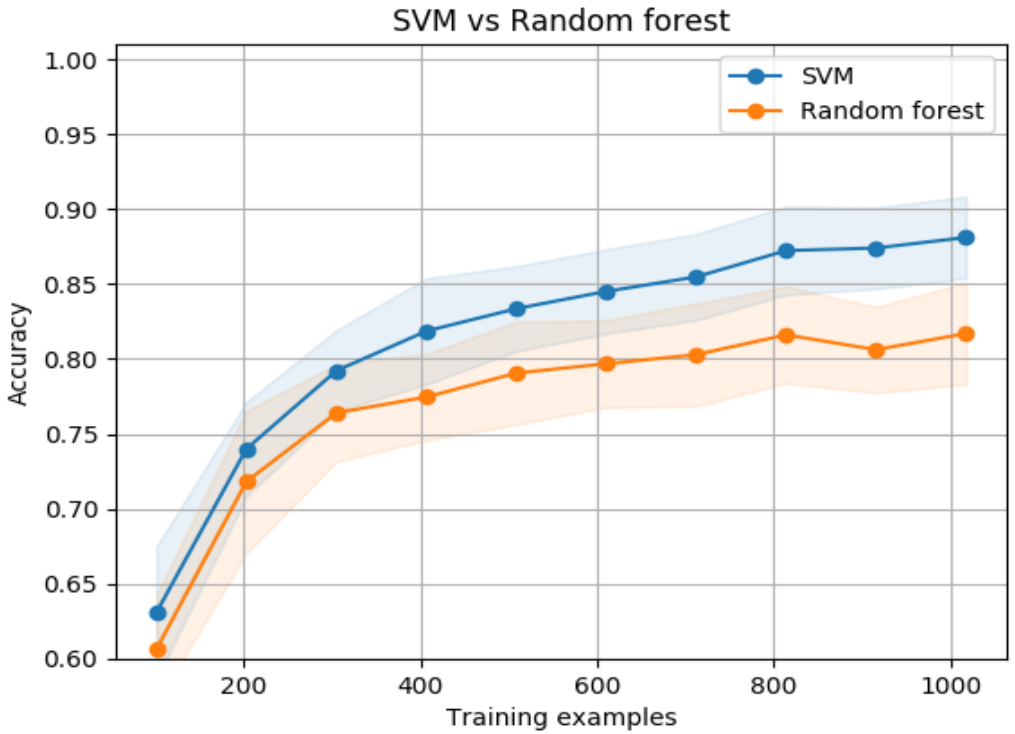




Example: compare 2 methods

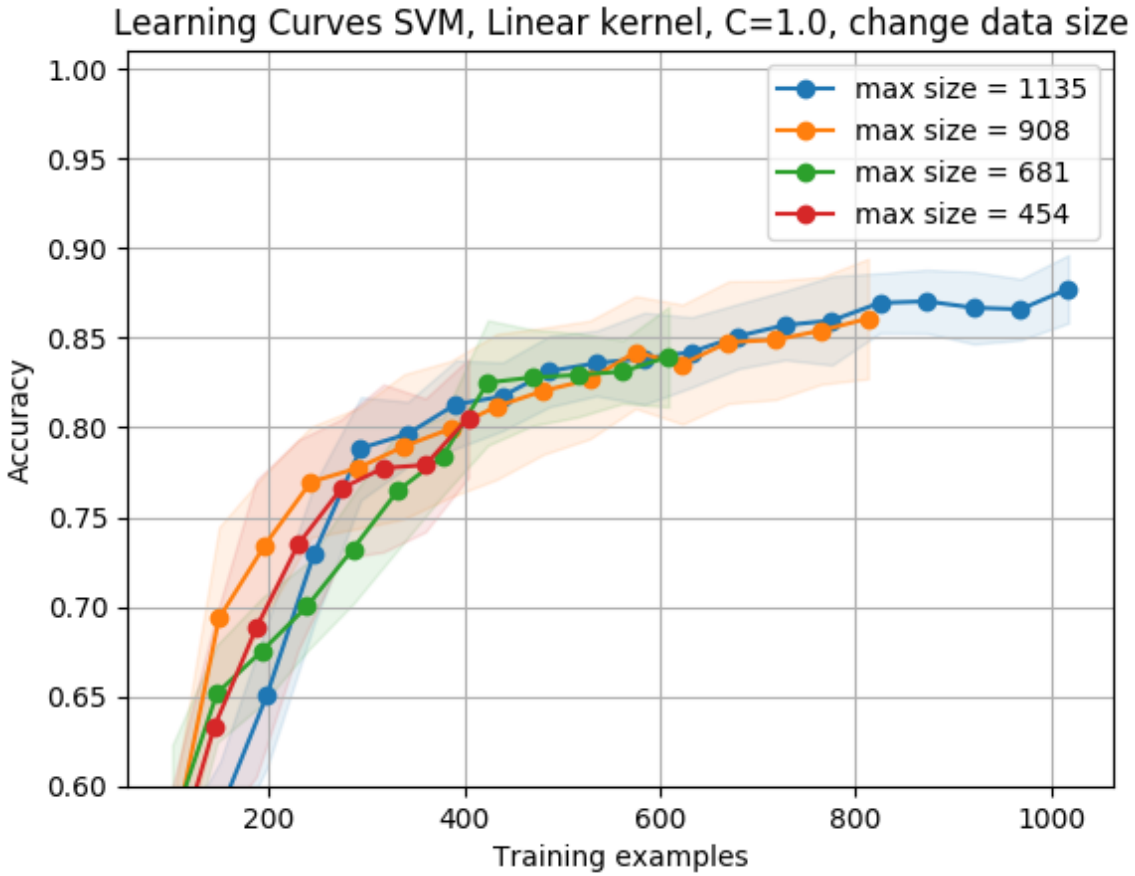
- Methods: **Random forest** vs **Support vector machines** (SVM)
- Parameter selection: 10-fold cross-validation
  - Random forest: n_estimate = 250
  - SVM: regularization constant C = 1





Example: effect of data size

- SVM
  - Parameter: size of training data
- Dataset: *1135 news, 10 classes, vocabulary of 25199 terms*
- 10-fold cross-validation is used



References

- L. Breiman. *Random forests*. Machine learning, 45(1), 5-32, 2001.
- Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, Dinani Amorim. *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?* Journal of Machine Learning Research, 15(Oct):3133−3181, 2014.
- T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- M. Nunez. *The use of background knowledge in decision tree induction*. Machine Learning, 6(3): 231-250, 1991.
- Quinlan, J. R. *Induction of Decision Trees*. Mach. Learn. 1, 1 (Mar. 1986), 81-106, 1986
- Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.
- Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, *34*(1), 1-47.