



## More on Supervised Learning

Amir H. Payberah  
payberah@kth.se  
2020-11-10



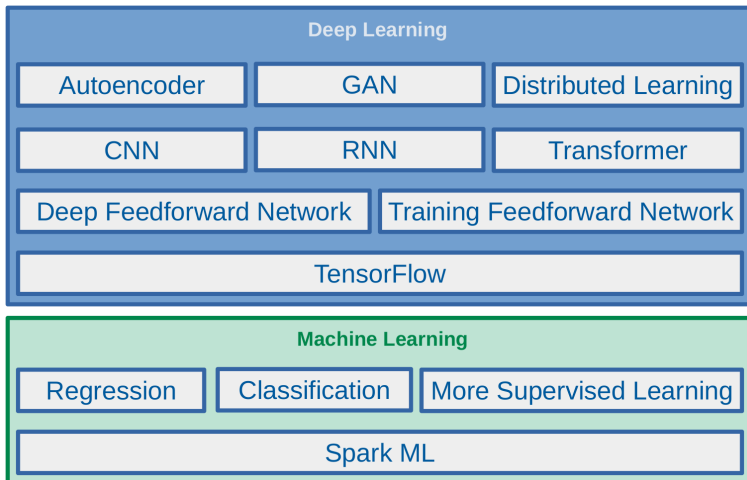


## The Course Web Page

`https://id2223kth.github.io`  
`https://tinyurl.com/y6kcpmzy`

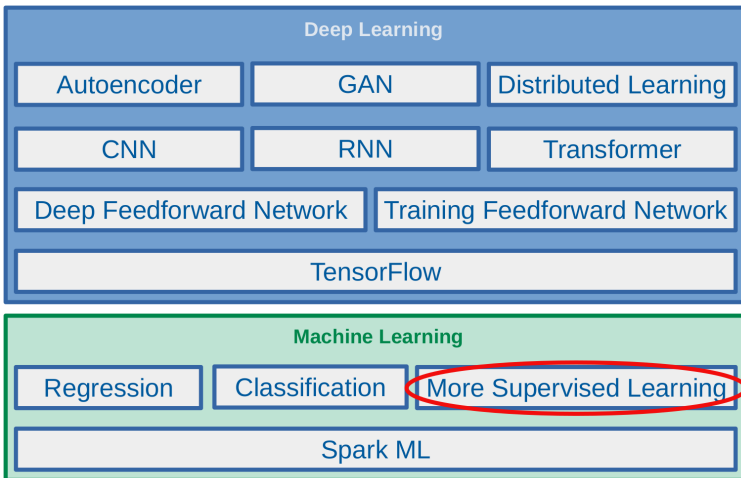


# Where Are We?





# Where Are We?





# Let's Start with an Example



## Buying Computer Example (1/3)

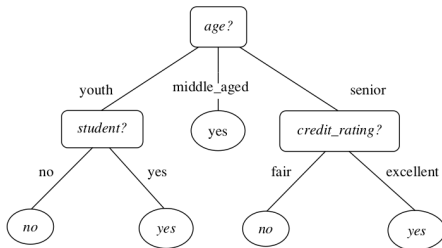
- ▶ Given the dataset of  $m$  people.

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	midleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
⋮	⋮	⋮	⋮	⋮	⋮

- ▶ Predict if a new person buys a computer?
- ▶ Given an instance  $\mathbf{x}^{(i)}$ , e.g.,  $x_1^{(i)} = \text{senior}$ ,  $x_2^{(i)} = \text{medium}$ ,  $x_3^{(i)} = \text{no}$ , and  $x_4^{(i)} = \text{fair}$ , then  $y^{(i)} = ?$

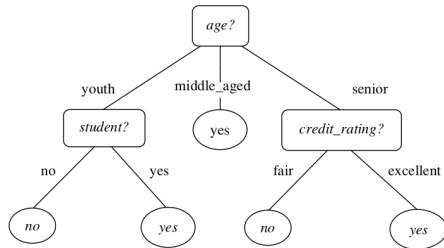
# Buying Computer Example (2/3)

id	age	income	student	credit rating	buys computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middleage	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
⋮	⋮	⋮	⋮	⋮	⋮



## Buying Computer Example (3/3)

- ▶ Given an input instance  $\mathbf{x}^{(i)}$ , for which the class label  $y^{(i)}$  is **unknown**.
- ▶ The **attribute values** of the input (e.g., **age** or **income**) are **tested**.
- ▶ A **path** is traced from the **root** to a **leaf** node, which holds the **class prediction** for that input.
- ▶ E.g., input  $\mathbf{x}^{(i)}$  with  $x_1^{(i)} = \text{senior}$ ,  $x_2^{(i)} = \text{medium}$ ,  $x_3^{(i)} = \text{no}$ , and  $x_4^{(i)} = \text{fair}$ .

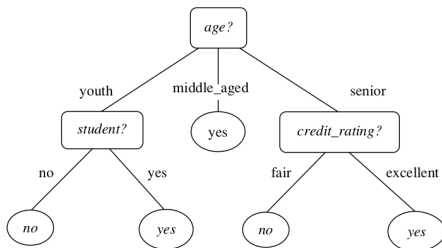




# Decision Tree

# Decision Tree

- ▶ A **decision tree** is a **flowchart-like tree structure**.
  - The **topmost node**: represents the **root**
  - Each **internal node**: denotes a **test on an attribute**
  - Each **branch**: represents an **outcome of the test**
  - Each **leaf**: holds a **class label**





## Training Algorithm (1/2)

- ▶ Decision trees are constructed in a top-down recursive divide-and-conquer manner.
- ▶ The algorithm is called with the following parameters.
  - Data partition  $D$ : initially the complete set of training data and labels  $D = (\mathbf{X}, \mathbf{y})$ .
  - Feature list: list of features  $\{\mathbf{x}_1^{(i)}, \dots, \mathbf{x}_n^{(i)}\}$  of each data instance  $\mathbf{x}^{(i)}$ .
  - Feature selection method: determines the splitting criterion.



## Training Algorithm (2/2)

- ▶ 1. The tree starts as a **single node**,  $N$ , representing the **training data instances**  $D$ .
- ▶ 2. If all instances  $x$  in  $D$  are all of the **same class**, then node  $N$  becomes a **leaf**.
- ▶ 3. The algorithm calls **feature selection method** to determine the **splitting criterion**.
  - Indicates (i) the **splitting feature**  $x_k$ , and (ii) a **split-point** or a **splitting subset**.
  - The instances in  $D$  are partitioned accordingly.
- ▶ 4. The algorithm repeats the same process **recursively** to form a decision tree.

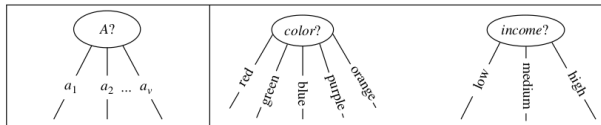


## Training Algorithm - Termination Conditions

- ▶ The training algorithm **stops** only when any one of the following **conditions** is **true**.
- ▶ 1. **All the instances** in partition  $D$  at a node  $N$  belong to **the same class**.
  - It is **labeled with that class**.
- ▶ 2. **No remaining features** on which the instances may be **further partitioned**.
- ▶ 3. There are **no instances** for a **given branch**, that is, a partition  $D_j$  is **empty**.
- ▶ In **conditions 2 and 3**:
  - Convert node  $N$  into a **leaf**.
  - Label it either with the **most common class** in  $D$ .
  - Or, the **class distribution** of the node tuples may be stored.

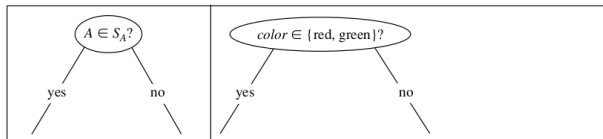
# Training Algorithm - Partitioning Instances (1/3)

- ▶ Assume  $A$  is the **splitting feature**
- ▶ **Three** possibilities to **partition instances** in  $D$  based on the feature  $A$ .
- ▶ 1.  $A$  is **discrete-valued**
  - Assume  $A$  has  $v$  distinct values  $\{a_1, a_2, \dots, a_v\}$
  - A **branch** is created for **each known value  $a_j$**  of  $A$  and **labeled with that value**.
  - Partition  $D_j$  is the **subset of tuples** in  $D$  having value  $a_j$  of  $A$ .



## Training Algorithm - Partitioning Instances (2/3)

- ▶ 2.  $A$  is **discrete-valued**
  - A **binary tree** must be produced.
  - The **test** at node  $N$  is of the form  $A \in S_A?$ , where  $S_A$  is the **splitting subset** for  $A$ .
  - The **left branch** out of  $N$  corresponds to the **instances in  $D$  that satisfy the test**.
  - The **right branch** out of  $N$  corresponds to the **instances in  $D$  that do not satisfy the test**.



# Training Algorithm - Partitioning Instances (3/3)

► 3. **A** is **continuous-valued**

- A test at node **N** has **two possible outcomes**: corresponds to  $A \leq s$  or  $A > s$ , with **s** as the **split point**.
- The **instances are partitioned** such that  $D_1$  holds the instances in **D** for which  $A \leq s$ , while  $D_2$  holds **the rest**.
- **Two branches** are **labeled** according to the **previous outcomes**.







## Training Algorithm - Feature Selection Measures (1/2)

- ▶ Feature selection measure: how to split instances at a node  $N$ .
- ▶ Pure partition: if all instances in a partition belong to the same class.
- ▶ The best splitting criterion is the one that most closely results in a pure scenario.



## Training Algorithm - Feature Selection Measures (2/2)

- ▶ It provides a ranking for each feature describing the given training instances.
- ▶ The feature having the best score for the measure is chosen as the splitting feature for the given instances.
- ▶ Two popular feature selection measures are:
  - Information gain (ID3)
  - Gini index (CART)

# Information Gain (Entropy)



## ID3 (1/7)

- ▶ ID3 (Iterative Dichotomiser 3) uses information gain as its feature selection measure.
- ▶ The feature with the highest information gain is chosen as the splitting feature for node  $N$ .
- ▶ The information gain is based on the decrease in entropy after a dataset is split on a feature.



## ID3 (2/7)

- ▶ What's entropy?
- ▶ The average information needed to identify the class label of an instance in  $D$ .

$$\text{entropy}(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

- ▶  $p_i$  is the probability that an instance in  $D$  belongs to class  $i$ , with  $m$  distinct classes.
- ▶  $D$ 's entropy is zero when it contains instances of only one class (pure partition).

<i>RID</i>	<i>age</i>	<i>income</i>	<i>student</i>	<i>credit_rating</i>	<i>Class: buys_computer</i>
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle.aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle.aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle.aged	medium	no	excellent	yes
13	middle.aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\text{entropy}(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

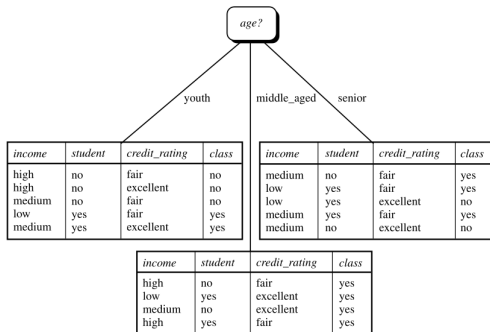
label = buys\_computer  $\Rightarrow m = 2$

$$\text{entropy}(D) = - \frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.94$$

- ▶ Suppose we want to **partition** instances in  $D$  on some feature  $A$  with  $v$  distinct values,  $\{a_1, a_2, \dots, a_v\}$ .
- ▶  $A$  can split  $D$  into  $v$  partitions  $\{D_1, D_2, \dots, D_v\}$ .
- ▶ The **expected information** required to **classify an instance** from  $D$  based on the partitioning by  $A$  is:

$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

- ▶  $\frac{|D_j|}{|D|}$  is the **weight** of the  $j$ th partition.
- ▶ The **smaller** the **expected information** required, the **greater** the **purity** of the partitions.



$$\text{entropy}(A, D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{entropy}(D_j)$$

$$\text{entropy}(\text{age}, D) = \frac{5}{14} \text{entropy}(D_{\text{youth}}) + \frac{4}{14} \text{entropy}(D_{\text{middle\_aged}}) + \frac{5}{14} \text{entropy}(D_{\text{senior}})$$

$$\text{entropy}(\text{age}, D) = \frac{5}{14} \left( -\frac{2}{5} \log_2\left(\frac{2}{5}\right) - \frac{3}{5} \log_2\left(\frac{3}{5}\right) \right) + \frac{4}{14} \left( -\frac{4}{4} \log_2\left(\frac{4}{4}\right) \right) + \frac{5}{14} \left( -\frac{3}{5} \log_2\left(\frac{3}{5}\right) - \frac{2}{5} \log_2\left(\frac{2}{5}\right) \right) = 0.694$$





## ID3 (6/7)

- ▶ The information gain  $\text{Gain}(A, D)$  is defined as:

$$\text{Gain}(A, D) = \text{entropy}(D) - \text{entropy}(A, D)$$

- ▶ It shows how much would be gained by branching on  $A$ .
- ▶ The feature  $A$  with the highest  $\text{Gain}(A, D)$  is chosen as the splitting feature at node  $N$ .



## ID3 (7/7)

- ▶ Now, we can compute the information gain  $\text{Gain}(A)$  for the feature  $A = \text{age}$ .

$$\text{Gain}(\text{age}, D) = \text{entropy}(D) - \text{entropy}(\text{age}, D) = 0.940 - 0.694 = 0.246$$

- ▶ Similarly we have:
  - $\text{Gain}(\text{income}, D) = 0.029$
  - $\text{Gain}(\text{student}, D) = 0.151$
  - $\text{Gain}(\text{credit\_rating}, D) = 0.048$
- ▶ The **age** has the highest information gain among the attributes, it is selected as the **splitting feature**.

# Gini Impurity



## CART (1/8)

- ▶ CART (Classification And Regression Tree) considers a binary split for each feature.
- ▶ It uses the Gini index to measure the misclassification (impurity of D).

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

- ▶  $p_i$  is the probability that an instance in D belongs to class  $i$ , with  $m$  distinct classes.
- ▶ It will be zero if all partitions are pure. Why?
- ▶ We need to determine the splitting criterion: splitting feature + splitting subset.



## CART (2/8)

- ▶ Assume  $A$  is a **discrete-valued feature** with  $v$  distinct values,  $\{a_1, a_2, \dots, a_v\}$ , occurring in  $D$ .
- ▶  $S_A$  will be all **possible subsets** of  $A$ .
  - E.g.,  $A = \text{income} = \{\text{low}, \text{medium}, \text{high}\}$
  - $S_A = \{\{\text{low}, \text{medium}, \text{high}\}, \{\text{low}, \text{medium}\}, \{\text{medium}, \text{high}\}, \{\text{low}, \text{high}\}, \{\text{low}\}, \{\text{medium}\}, \{\text{high}\}, \{\}\}$
  - The **test** is of the form  $D_1 \in s_A?$ , where  $s_A$  is a subset of  $S_A$ , e.g.,  $s_A = \{\text{low}, \text{high}\}$ .

# CART (3/8)

<i>RID</i>	<i>age</i>	<i>income</i>	<i>student</i>	<i>credit_rating</i>	<i>Class: buys_computer</i>
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$\text{Gini}(D) = 1 - \sum_{i=1}^m p_i^2$$

label = buys\_computer  $\Rightarrow m = 2$

$$\text{Gini}(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$



## CART (4/8)

- ▶ If a binary split on  $A$  partitions  $D$  into  $D_1$  and  $D_2$ , the **Gini index** of  $D$  given that partitioning is:

$$\text{Gini}(A, D) = \frac{|D_1|}{D} \text{Gini}(D_1) + \frac{|D_2|}{D} \text{Gini}(D_2)$$

- ▶ The subset that gives the **minimum Gini index** is selected as its **splitting subset**.

- ▶ For a feature  $A = \text{income}$ , we consider each of the possible **splitting subsets**.
  - $S_A = \{\{\text{low, medium, high}\}, \{\text{low, medium}\}, \{\text{medium, high}\}, \{\text{low, high}\}, \{\text{low}\}, \{\text{medium}\}, \{\text{high}\}, \{\}\}$
- ▶ Assume, we choose the **splitting subset**  $s_A = \{\text{low, medium}\}$ .
- ▶ Consider partition  $D_1$  satisfies the condition  $D_1 \in s_A$ , and  $D_2$  does not.

$$\begin{aligned} \text{Gini}_{\text{income} \in \{\text{low, medium}\}}(A, D) &= \frac{10}{14} \text{Gini}(D_1) + \frac{4}{14} \text{Gini}(D_2) \\ &= \frac{10}{14} \text{Gini}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14} \left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) = 0.443 \end{aligned}$$



- ▶ Similarly, we calculate the **Gini index** values for splits on the **remaining subsets**.

$$\text{Gini}_{\text{income} \in \{\text{low}, \text{medium}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{high}\}}(A, D) = 0.443$$

$$\text{Gini}_{\text{income} \in \{\text{low}, \text{high}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{medium}\}}(A, D) = 0.458$$

$$\text{Gini}_{\text{income} \in \{\text{medium}, \text{high}\}}(A, D) = \text{Gini}_{\text{income} \in \{\text{low}\}}(A, D) = 0.450$$

- ▶ The best binary split for attribute  $A = \text{income}$  is on  $s_A = \{\text{low}, \text{medium}\}$  because it **minimizes the Gini index**.



## CART (7/8)

- ▶ But, **which feature?**
- ▶ The **reduction in impurity** that would be incurred by a binary split on feature **A** is:

$$\Delta\text{Gini}(A) = \text{Gini}(D) - \text{Gini}(A,D)$$

- ▶ The feature that **maximizes the reduction in impurity** (has the **minimum Gini index**) is selected as the **splitting feature**.



## CART (8/8)

- ▶ Now, we can compute the **information gain**  $\text{Gain}(A)$  for different features.
  - $\Delta\text{Gini}(\text{income}) = 0.459 - 0.443 = 0.016$
  - $\Delta\text{Gini}(\text{age}) = 0.459 - 0.357 = 0.102$
  - $\Delta\text{Gini}(\text{student}) = 0.459 - 0.367 = 0.092$
  - $\Delta\text{Gini}(\text{credit\_rating}) = 0.459 - 0.429 = 0.03$
- ▶ The feature  $A = \text{age}$  and splitting subset  $s_A = \{\text{youth}, \text{senior}\}$  gives the **minimum Gini index** overall.



# Decision Tree in Spark (1/4)

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `DecisionTreeRegressor`

```
val dt_regressor = new DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("features")
val model = dt_regressor.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```

- ▶ Classifier: `DecisionTreeClassifier`

```
val dt_classifier = new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("features")
val model = dt_classifier.fit(trainingData)
val predictions = model.transform(testData)
predictions.select("prediction", "rawPrediction", "probability", "label", "features").show(5)
```



## Decision Tree in Spark (2/4)

- ▶ Input and output columns
- ▶ `labelCol` and `featuresCol` identify **label** and **features** column's names.
- ▶ `predictionCol` indicates the **predicted label**.
- ▶ `rawPredictionCol` is a **vector** of length of number of classes, with the **counts of training instance** labels at the tree node which makes the prediction.
- ▶ `probabilityCol` is a **vector** of length of number of classes equal to `rawPrediction` **normalized to a multinomial distribution**.



## Decision Tree in Spark (3/4)

- ▶ Tunable parameters
- ▶ `maxBins`: number of bins used when discretizing continuous features.
- ▶ `impurity`: impurity measure used to choose between candidate splits, e.g., `entropy` and `gini`.

```
val maxBins = ...  
val dt_classifier = new DecisionTreeClassifier().setMaxBins(maxBins).setImpurity("gini")
```



## Decision Tree in Spark (4/4)

- ▶ **Stopping criteria** that determines when the tree stops building.
- ▶ **maxDepth**: **maximum depth** of a tree.
- ▶ **minInstancesPerNode**: for a node to be split further, each of its **children** must receive **at least this number of training instances**.
- ▶ **minInfoGain**: for a node to be split further, the split must **improve** at least this much (in terms of **information gain**).

```
val maxDepth = ...
val minInstancesPerNode = ...
val minInfoGain = ...
val dt_classifier = new DecisionTreeClassifier()
    .setMaxDepth(maxDepth)
    .setMinInstancesPerNode(minInstancesPerNode)
    .setMinInfoGain(minInfoGain)
```



# Ensemble Methods





## Wisdom of the Crowd

- ▶ Ask a **complex question** to **thousands of random people**, then aggregate their answers.
- ▶ In many cases, this **aggregated answer** is **better** than an **expert's answer**.
- ▶ This is called the **wisdom of the crowd**.
- ▶ Similarly, the aggregated estimations of a **group of estimators** (e.g., **classifiers or regressors**), often gets **better estimations** than with the best individual estimator.
- ▶ A **group of estimators** is an **ensemble**, and this technique is called **Ensemble Learning**.

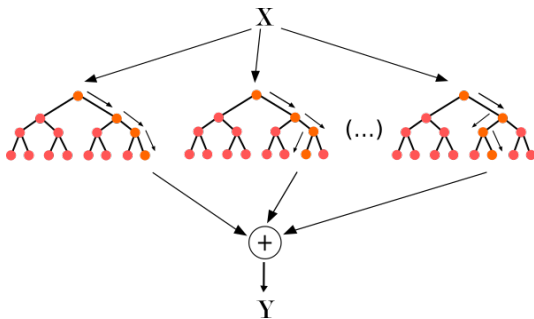


# Ensemble Learning

- ▶ Two main categories of **ensemble learning** algorithms.
- ▶ **Bagging**
  - Use the **same training algorithm** for **every estimator**, but to train them on **different random subsets** of the training set.
  - E.g., **random forest**
- ▶ **Boosting**
  - Train estimators **sequentially**, each trying to **correct its predecessor**.
  - E.g., **adaboost** and **gradient boosting**

# Random Forest

- ▶ **Random forest** builds **multiple decision trees** that are most of the time trained with the **bagging** method.
- ▶ It, then, merges the trees together to get a more **accurate and stable prediction**.





## Random Forest in Spark (1/2)

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `RandomForestRegressor`

```
val rf_regressor = new RandomForestRegressor().setLabelCol("label")  
                                                    .setFeaturesCol("features").setNumTrees(10)  
val model = rf_regressor.fit(trainingData)  
val predictions = model.transform(testData)  
predictions.select("prediction", "label", "features").show(5)
```

- ▶ Classifier: `RandomForestClassifier`

```
val rf_classifier = new RandomForestClassifier().setLabelCol("label")  
                                                    .setFeaturesCol("features").setNumTrees(10)  
val model = rf_classifier.fit(trainingData)  
val predictions = model.transform(testData)  
predictions.select("prediction", "label", "features").show(5)
```

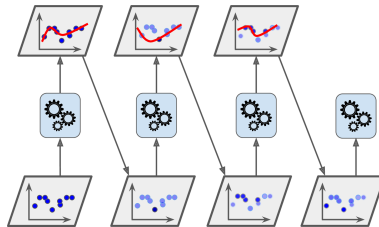


## Random Forest in Spark (2/2)

- ▶ `numTrees`: **number of trees** in the forest.
- ▶ `subsamplingRate`: specifies the **size of the dataset** used for training each tree in the forest, as a **fraction of the size of the original dataset**.
  - Default is 1.0 and decreasing it can speed up training.
- ▶ `featureSubsetStrategy`: **number of features** to use as candidates for splitting at each tree node, as a **fraction of the total number of features**.
  - Possible values: `auto`, `all`, `onethird`, `sqrt`, `log2`, `n`

# AdaBoost

- ▶ **AdaBoost**: train a **new estimator** by paying more attention to the training instances that the **predecessor underfitted**.
- ▶ Each **estimator** is trained on a **random subset** of the **total training set**.
- ▶ AdaBoost assigns a **weight** to each **training instance**, which determines the **probability** that each instance should **appear in the training set**.





## Gradient Boosting (1/3)

- ▶ Just like AdaBoost, **Gradient Boosting** works by **sequentially** adding **estimators to an ensemble**, each one **correcting its predecessor**.
- ▶ However, instead of tweaking the instance weights at every iteration, this method **tries to fit the new estimator** to the **residual errors** made by the previous estimator.



## Gradient Boosting (2/3)

- ▶ Let's go through a regression example using **Gradient Boosted Regression Trees**.
- ▶ Fit the **first estimator** on the **training set**.

```
tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

- ▶ Now train the **second estimator** on the **residual errors** made by the **first estimator**.

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```





## Gradient Boosting (3/3)

- ▶ Then we train the **third estimator** on the **residual errors** made by the **second estimator**.

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

- ▶ Now we have an **ensemble containing three trees**.
- ▶ It can **make predictions** on a new instance simply by adding up the predictions of all the trees.

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```



# Gradient Boosting in Spark

- ▶ Two classes in `spark.ml`.
- ▶ Regression: `GBTRegressor`

```
val gbt = new GBTRegressor().setLabelCol("label").setFeaturesCol("features")
    .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```

- ▶ Classifier: `GBTClassifier`

```
val gbt = new GBTClassifier().setLabelCol("label").setFeaturesCol("features")
    .setMaxIter(10).setFeatureSubsetStrategy("auto")

val model = gbt.fit(trainingData)
val predictions = model.transform(testData)
```

# Summary



# Summary

- ▶ Decision tree
  - Top-down training algorithm
  - Termination condition
  - Feature selection: entropy, gini
  
- ▶ Ensemble models
  - Bagging: random forest
  - Boosting: AdaBoost, Gradient Boosting



## Reference

- ▶ Aurélien Géron, Hands-On Machine Learning (Ch. 5, 6, 7)
- ▶ Matei Zaharia et al., Spark - The Definitive Guide (Ch. 27)

Questions?