Outline

• **Decision Trees**
  — Key problems of Decision Trees learning
  — Impurity measure
  — Heuristic algorithms for building Decision Trees
  — Evaluation and tuning
  — Weak spots of Decision Trees

• **Ensemble learning and Random Forests**
  — Ensemble classifiers – a motivation exercise
  — Combining classifiers into ensembles – general scheme
  — Generating random samples by bootstrapping
  — Bagging – a simple ensemble classifier
  — Random Forests: Bootstrapping extension of Decision Trees
Learning a decision tree – key problems

Each node of a decision tree is associated with a subset of training data

Building a decision tree means to make a hierarchical sequence of splits. Each practical algorithm must be able to efficiently decide the following key questions:

1. How to choose a suitable splitting condition?
2. When to stop the splitting process?
Learning a decision tree – key problems

Each node of a decision tree is associated with a subset of training data

Building a decision tree means to make a hierarchical sequence of splits. Each practical algorithm must be able to efficiently decide the following key questions:

(1) How to choose a suitable splitting condition?

(2) When to stop the splitting process?

A practical answer to problem (1) is to employ entropy or another similar measure. Each node is defined by an associated subset of examples with a specific distribution of target values. After a split, the entropy in child nodes should decrease in comparison with entropy in the parent node.
Learning a decision tree – key problems

Each node of a decision tree is associated with a subset of training data

Building a decision tree means to make a hierarchical sequence of splits. Each practical algorithm must be able to efficiently decide the following key questions:

(1) How to choose a suitable splitting condition?

(2) When to stop the splitting process?

A practical answer to problem (1) is to employ entropy or another similar measure. Each node is defined by an associated subset of examples with a specific distribution of target values. After a split, the entropy in child nodes should decrease in comparison with entropy in the parent node.

The splitting process should be duly stopped just to not produce model that overfits the training data. To avoid overfitting, practical implementations usually use pruning after building a relatively deep tree.
Historical excursion

- ID3 ~ Iterative Dichotomiser
- AID ~ Automatic Interaction Detection
- CART ~ Classification and Regression Trees

Probably most well-known is the “C 5.0” algorithm (Quinlan), which has become the industry standard.
Packages in R: `rpart`
Building a classification tree from training data

We work with decisions on the value of only a single feature

• For each categorical feature $A_j$ having values $Values(A_j) = \{b_1, b_2, \ldots, b_L\}$
  
  $$is\ x_j = b_i? \text{ as } i = 1, \ldots, L$$

• For each categorical feature $A_j$
  
  $$is\ x_j \in \text{a subset} \in 2^{Values(A_j)}?$$

• For each numerical feature $A_j$
  
  $$is\ x_j \leq k?, \ k \in (-\infty, +\infty)$$
Which decision is the best?

- Focus on the distribution of target class values in the associated subset of training examples.
- Then select the decision that splits training data into subsets as pure as possible.
Which decision is the best?

We say a data set is pure (or homogenous) if it contains only a single class. If a data set contains several classes, then the data set is impure (or heterogenous).
Which decision is the best?

We say a data set is **pure** (or **homogenous**) if it contains only a single class. If a data set contains several classes, then the data set is **impure** (or **heterogenous**).

Example:

<table>
<thead>
<tr>
<th>⊕: 5, ⊖: 5</th>
<th>⊕: 9, ⊖: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>heterogenous</td>
<td>almost homogenous</td>
</tr>
<tr>
<td>high degree of impurity</td>
<td>low degree of impurity</td>
</tr>
</tbody>
</table>
Building a classification tree from training data

Which decision is the best?

1. Define a candidate set $S$ of splits at each node using possible decisions. $s \in S$ splits $t$ into two subsets $t_1$ and $t_2$.

2. Define the node proportions $p(y_j|t), j = 1, \ldots, k$, to be the proportion of instances $\langle x, y_j \rangle$ in $t$.

3. Define an impurity measure $i(t)$, i.e. splitting criterion, as a non-negative function $\Phi$ of the $p(y_1|t), p(y_2|t), \ldots, p(y_k|t)$,

\[
i(t) = \Phi(p(y_1|t), p(y_2|t), \ldots, p(y_k|t)), \tag{1}\]

such that

- $\Phi(\frac{1}{k}, \frac{1}{k}, \ldots, \frac{1}{k}) = \text{max}$, i.e. the node impurity is largest when all examples are equally mixed together in it.

- $\Phi(1, 0, \ldots, 0) = 0, \Phi(0, 1, \ldots, 0) = 0, \ldots, \Phi(0, 0, \ldots, 1) = 0$, i.e. the node impurity is smallest when the node contains instances of only one class.
Building a classification tree from training data

Which decision is the best?

4. Define the **goodness of split** $s$ to be the decrease in impurity
   \[ \Delta i(s, t) = i(t) - (p_1 \times i(t_1) + p_2 \times i(t_2)), \]
   where $p_i$ is the proportion of instances in $t$ that go to $t_i$.

5. Find split $s^*$ with the largest decrease in impurity:
   \[ \Delta i(s^*, t) = \max_{s \in S} \Delta i(s, t). \]

6. Use splitting criterion $i(t)$ to compute $\Delta i(s, t)$ and get $s^*$.

---

NPFL054, 2023 Hladká & Holub Lecture 5, page 9/39
Which decision is the best?

Splitting criteria – examples that are really used

- Misclassification Error – $i(t)_{ME}$
- Information Gain – $i(t)_{IG}$
- Gini Index – $i(t)_{GI}$
Building a classification tree from training data
Which decision is the best?

Splitting criteria — Misclassification Error \( i(t)_{ME} \)

\[
i(t)_{ME} = 1 - \max_{j=1,...,k} p(y_j|t)
\] (2)
Building a classification tree from training data
Which decision is the best?

Splitting criteria — Misclassification Error \( i(t)_{ME} \)

\[
i(t)_{ME} = 1 - \max_{j=1,...,k} p(y_j | t)
\]

(2)

Example:

<table>
<thead>
<tr>
<th></th>
<th>( \oplus: 0, \odot: 6 )</th>
<th>( \oplus: 1, \odot: 5 )</th>
<th>( \oplus: 2, \odot: 4 )</th>
<th>( \oplus: 3, \odot: 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i(t)_{ME} )</td>
<td>( 1 - \frac{6}{6} = 0 )</td>
<td>( 1 - \frac{5}{6} = 0.17 )</td>
<td>( 1 - \frac{4}{6} = 0.33 )</td>
<td>( 1 - \frac{3}{6} = 0.5 )</td>
</tr>
</tbody>
</table>
Building a classification tree from training data
Which decision is the best?

Splitting criteria — Information Gain $i(t)_{IG}$

$$i(t)_{IG} = - \sum_{j=1}^{k} p(y_j|t) \log p(y_j|t). \quad (3)$$

Recall the notion of entropy $H(t)$, $i(t)_{IG} = H(t)$.

$$\text{Gain}(s,t) = \Delta i(s,t)_{IG} \quad (4)$$
Building a classification tree from training data
Which decision is the best?

Splitting criteria — Gini Index $i(t)_{GI}$

$$i(t)_{GI} = 1 - \sum_{j=1}^{k} p^2(y_j|t) = \sum_{j=1}^{k} p(y_j|t)(1 - p(y_j|t)).$$

(5)
Building a classification tree from training data
Which decision is the best?

Splitting criteria — Gini Index $i(t)_{GI}$

$$i(t)_{GI} = 1 - \sum_{j=1}^{k} p^2(y_j|t) = \sum_{j=1}^{k} p(y_j|t)(1 - p(y_j|t)). \quad (5)$$

**Interpretation**

Use the rule that assigns an instance selected at random from the node to class $i$ with probability $p(i|t)$. The estimated probability that the item is actually in class $j$ is $p(j|t)$. The estimated probability of misclassification is the Gini index. In other words, Gini can be interpreted as expected error rate.
Building a classification tree from training data

Which decision is the best?
Splitting criteria – a comparison example

<table>
<thead>
<tr>
<th>Split</th>
<th>Gini</th>
<th>Entropy</th>
<th>ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊕: 0</td>
<td>0</td>
<td>0.65</td>
<td>0.17</td>
</tr>
<tr>
<td>⊖: 6</td>
<td>0</td>
<td>0.278</td>
<td>0.444</td>
</tr>
<tr>
<td>⊕: 1</td>
<td>0.278</td>
<td>0.92</td>
<td>0.333</td>
</tr>
<tr>
<td>⊖: 5</td>
<td>0.278</td>
<td>0.92</td>
<td>0.333</td>
</tr>
<tr>
<td>⊕: 2</td>
<td>0.444</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>⊖: 4</td>
<td>0.444</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>⊕: 3</td>
<td>0.5</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>⊖: 3</td>
<td>0.5</td>
<td>1.0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

For two classes \((k = 2)\), if \(p\) is the proportion of the class "1", the measures are:

- Misclassification error: \(1 - \max(p, 1 - p)\)
- Entropy: \(-p \times \log p - (1 - p) \times \log(1 - p)\)
- Gini: \(2p \times (1 - p)\)
Building a classification tree from training data

Which decision is the best?

Splitting criteria

![Graph showing Misclassification, Entropy, and Gini values](image)

- **Misclassification error**
- **Gini index**
- **Entropy**
Again, we work with decisions on the value of only a single feature

Which decision is the best?

Splitting criterion – usually used

- Squared Error – \( i(t)_{SE} \)

\[
i(t)_{SE} = \frac{1}{|t|} \sum_{x_i \in t} (y_i - y^t)^2,
\]

where \( y^t = \frac{1}{|t|} \sum_{x_i \in t} y_i \).
The recursive binary splitting is stopped when a stopping criterion is fulfilled. Then a leaf node is created with an output value.

**Stopping criteria**, e.g.

- the leaf node is associated with less than five training instances
- the maximum tree depth has been reached
- the best splitting criteria is not greater than a certain threshold
Overfitting can be avoided by

- applying a stopping criterion that prevents some sets of training instances from being subdivided,
- removing some of the structure of the decision tree after it has been produced.

Preferred strategy
Grow a large tree $T_0$, stop the splitting process when only some minimum node size (say 5) is reached. Then prune $T_0$ using some pruning criteria.
Decision trees learning parameters

2 phases of decision tree learning:

- growing
- pruning

Learning parameters are used to control these two phases:
Decision trees learning parameters

2 phases of decision tree learning:
  • growing
  • pruning

Learning parameters are used to control these two phases:
  • when to stop growing
  • how much to prune the tree
2 phases of decision tree learning:

- growing
- pruning

Learning parameters are used to control these two phases:

- when to stop growing
- how much to prune the tree

... to avoid overfitting and improve performance
There are two widely used packages in R

- `rpart`
- `tree`

The algorithms used are very similar.

**References**

- An Introduction to Recursive Partitioning Using the RPART Routines by Terry M. Therneau, Elizabeth J. Atkinson, and Mayo Foundation (available online)
- *An Introduction to Statistical Learning with Application in R* Chapters 8.1, 8.3.1, and 8.3.2 by Gareth James, Daniela Witten, Trevor Hastie and Rob Tibshirani (available online)
- R packages documentation — `rpart`, `tree` (available online)
Example heuristic — implementation in R

Learning parameters in `rpart()`

```
rpart.control
```

**minsplt**
- the minimum number of observations that must exist in a node in order for a split to be attempted

**cp**
- complexity parameter, influences the depth of the tree

... and others, see `?rpart.control`

T: try to set different `cp` and `minsplt` values in the M1 model learning and observe the resulting tree
Meaning of the \text{cp} parameter

- Any split that does not decrease the \textit{relative training error} by a factor of \text{cp} is not attempted

\implies That means, the learning algorithm measures for each split how it improves the tree relative error and if the improvement is too small, the split will not be performed.

\textbf{Relative error} is the error relative to the misclassification error (without any splitting relative error is 100%)
How to choose the optimal cp value?

```r
> m = rpart(profits ~ category + sales + assets + marketvalue,
             data=F[data.train, 1:8], cp=0.001)
> m$cptable

   CP  nsplit rel error  xerror     xstd
1 0.5432596      0 1.0000000 1.0482897 0.03178559
2 0.027162978     1 0.4567404 0.4607646 0.02673551
3 0.007042254     3 0.4024145 0.4446680 0.02640028
4 0.006036217     6 0.3762575 0.4507042 0.02652763
5 0.005030181     8 0.3641851 0.4567404 0.02665301
6 0.004024145    15 0.3279678 0.4768612 0.02705703
7 0.003018109    19 0.3118712 0.4688129 0.02689795
8 0.002012072    21 0.3058350 0.4869215 0.02725122
9 0.001006036    23 0.3018109 0.5171026 0.02780383
10 0.001000000   25 0.2997988 0.5412475 0.02821490
```

**rel error**
- relative error on training data

**xerror**
- relative error in x-fold **cross-validation**

**xstd**
- standard deviation of xerror on x validation folds
How to choose the optimal $cp$ value?

The graph shows the relationship between the size of the tree and the X-val Relative Error. The $cp$ value is represented on the x-axis, and the size of the tree is on the y-axis. The optimal $cp$ value is chosen where the X-val Relative Error is minimized. In this case, the optimal $cp$ value appears to be around 0.0065.
• data splitting
  — deeper nodes can learn only from small data portions

• sensitivity to training data set (unstable algorithm)
  — learning algorithm is called unstable if small changes in the training set
  cause large differences in generated models
Consider the following task – we have a binary classification problem and a number of predictors, each with error less than 0.5. Will the resulting majority voting ensemble have an error lower than the single classifiers?
Consider the following task – we have a binary classification problem and a number of predictors, each with error less than 0.5. Will the resulting majority voting ensemble have an error lower than the single classifiers?

Depends on the accuracy and the diversity of the base learners!
Consider the following task – we have a binary classification problem and a number of predictors, each with error less than 0.5. Will the resulting majority voting ensemble have an error lower than the single classifiers?

Depends on the accuracy and the diversity of the base learners!

Illustrative example
Particular settings – assume that you have
- 21 classifiers
- each with error \( p = 0.3 \)
- their outputs are statistically independent

Compute the error of the ensemble under these conditions!
Solution of the exercise

How many classifiers will produce error output?
Key idea: The number of them will be binomially distributed! $\sim \text{Bi}(21, 0.3)$

```r
> plot(0:21, dbinom(0:21, 21, 0.3))
> dbinom(11, 21, 0.3)
[1] 0.01764978
> pbinom(10, 21, 0.3)
[1] 0.9736101
```

Conclusion: Accuracy of the ensemble will be more than 97.3%!
General scheme of combining classifiers

- Training data
- $L_1, \ldots, L_k$, $k$ base learners that produce $k$ hypotheses $h_1, \ldots, h_k$
- Combining function $h(x_i)$
- Final ensemble prediction
Resampling can be used as a way to produce diversity among base learners

- Distribute the training data into $K$ portions
- Run the learning process to get $K$ different models
- Collect the output of the $K$ models use a combining function to get a final output value
Bootstrapping principle

- New data sets $Data_1, \ldots, Data_K$ are drawn from $Data$ with replacement, each of the same size as the original $Data$, i.e. $n$.

- In the $i$-th step of the iteration, $Data_i$ is used as a training set, while the examples $\{x \mid x \in Data \land x \notin Data_i\}$ form the test set.
Bootstrapping principle

• New data sets $Data_1, \ldots, Data_K$ are drawn from $Data$ with replacement, each of the same size as the original $Data$, i.e. $n$.

• In the $i$-th step of the iteration, $Data_i$ is used as a training set, while the examples $\{x \mid x \in Data \land x \notin Data_i\}$ form the test set.

• The probability that we pick an instance is $1/n$, and the probability that we do not pick an instance is $1 - 1/n$. The probability that we do not pick it after $n$ draws is $(1 - 1/n)^n \approx e^{-1} \approx 0.368$.

• It means that for training the system will not use 36.8% of the data, and the error estimate will be pessimistic. So the solution is to repeat the process many times.
Ensemble methods – key ideas

• combining the classification results from different classifiers to produce the final output
• using (un)weighted voting
• different training data – e.g. bootstrapping
• different features
• different values of the relevant parameters
• performance: complementarity → potential improvement

Two fundamental approaches

• **Bagging** works by taking a bootstrap sample from the training set
• **Boosting** works by changing the weights on the training set
Are ensembles effective?

Combining multiple learners

- the more **complementary** the learners are, the more useful their combining is
- the simplest way to combine multiple learners is **voting**
- in **weighted voting** the voters (= base-learners) can have different weights

Unstable learning

- learning algorithm is called unstable if small changes in the training set cause large differences in generated models
- typical unstable algorithm is the decision trees learning
- bagging or boosting techniques are a natural remedy for unstable algorithms
• Bagging is a voting method that uses slightly different training sets (generated by bootstrap) to make different base-learners.

• Generating complementary base-learners is left to chance and to unstability of the learning method.

• Generally, bagging can be combined with any approach to learning.
Simple bagging algorithm

Bootstrap AGGregatING

1. for $i \leftarrow 1$ to $K$ do
2. $Train_i \leftarrow$ bootstrap($Data$)
3. $h_i \leftarrow$ TrainPredictor($Train_i$)

Combining function

- **Classification:** $h_{final}(x) = \text{MajorityVote}(h_1(x), h_2(x), \ldots, h_K(x))$
- **Regression:** $h_{final}(x) = \text{Mean}(h_1(x), h_2(x), \ldots, h_K(x))$
Random Forests — an extension of Decision Trees

- Random Forest is another example of bagging
- an ensemble method based on decision trees and bagging
- builds a number of independent random decision trees and then uses voting
- introduced by L. Breiman (2001), then developed by L. Breiman and A. Cutler
- very good (state-of-the-art) prediction performance
- a nice page with description
  www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- important: Random Forests helps to
  - avoid overfitting (by random sampling the training data set)
  - select important/useful features (by random sampling the feature set)
Building Random Forests

The algorithm for building a tree in the ensemble

1. Having a training set of the size $n$, sample $n$ cases at random – with replacement, and use the sample to build a decision tree.

2. If there are $M$ input features, choose a less number $m \ll M$. When building the tree, at each node a random sample of $m$ features is selected as split candidates from the full set of $M$ available features. Then the best split on these $m$ features is used to split the node. A fresh sample of $m$ features is taken at each split.
   - $m$ is fixed for the whole procedure

3. Each tree is grown to the largest extent possible. There is no pruning.

The more trees in the ensemble, the better.
There is no risk of overfitting!
R packages for Random Forests

• **randomForest**: Breiman and Cutler’s random forests for classification and regression
  – Classification and regression based on a forest of trees using random inputs.

• **RRF**: Regularized Random Forest
  – Feature Selection with Regularized Random Forest. This package is based on the 'randomForest' package by Andy Liaw. The key difference is the RRF function that builds a regularized random forest.
  – [http://cran.r-project.org/web/packages/RRF/index.html](http://cran.r-project.org/web/packages/RRF/index.html)

• **party**: A Laboratory for Recursive Partytioning
  – a computational toolbox for recursive partitioning
  – `cforest()` provides an implementation of Breiman’s random forests
  – extensible functionality for visualizing tree-structured regression models is available
Examination requirements

• You should understand the basic ideas of building and using Decision Trees for both classification and regression task.
  • Decision Trees – splitting criteria: typical heuristics
  • Decision Trees – pruning and overfitting: the complexity parameter
  • Decision Trees – practical use of the `rpart()` package

• You should understand Random Forests, which is an important and effective extension of simple Decision Trees.

• You should be able to practically use `rpart()` and `randomForest()` packages in R.

• Also, later we will discuss Random Forests again, in connection with more general ensemble methods.


