CS 236756 - Technion - Intro to Machine Learning

Tal Daniel

Tutorial 06 - Decision Trees

Agenda

In [1]:
```
# imports for the tutorial
import numpy as np
import pandas as pd
import re  # regex
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
%matplotlib notebook
```

Motivation

* Decision Trees are ML algorithms that can perform **both** classification and regression tasks, and even multioutput tasks.
* They are very powerful algorithms, capable of fitting complex datasets.
* They are a fundamental component of **Random Forests**, which are amongst the most powerful ML algorithms today.
* Decision Trees require **very little data** preparation - they do not require **feature scaling or centering**.

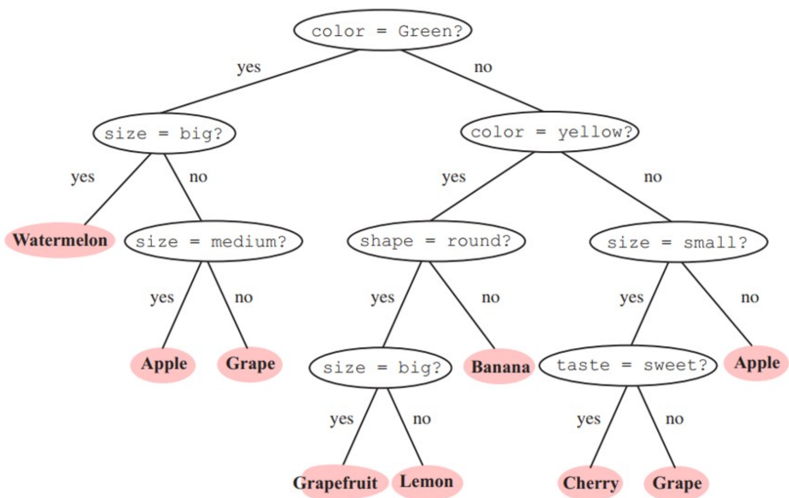
Examples

[**Akinator** (http://en.akinator.com/personnages/jeu)](http://en.akinator.com/personnages/jeu) - uses some implementation of *Decision Trees*.

20 Questions - one player is chosen to be the answerer. That person chooses a subject (object) but does not reveal this to the others. All other players are questioners. They each take turns asking a question which can be answered with a simple "Yes" or "No."

- *Features* - different properties of a character we ask about
- *Label* - the identity of the character
- *Training Set* - the characters you have encountered in your life
- *Test/Inference* - play "20 Questions"
- A good strategy:
  - Eliminate as many options as possible
  - Simple questions
  - Consider more likely characters (Leibniz isn't who most people think of, but in the Technion, this might not be the case...)
- Note: this game is a special case of classification where *each sample is its own class*

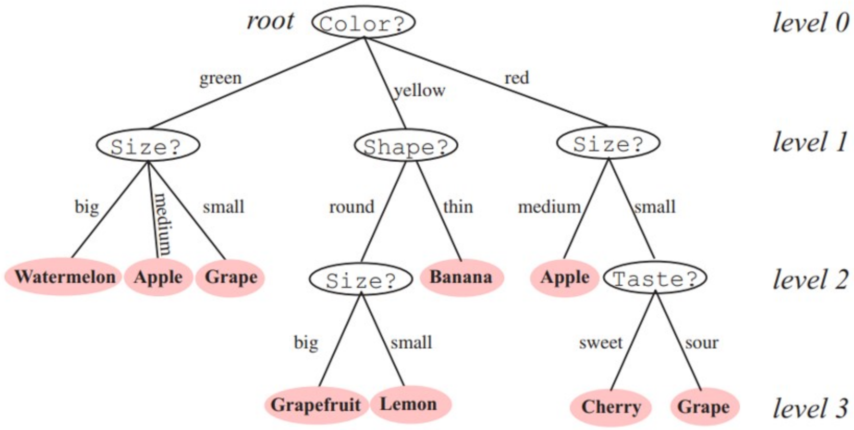
Decision Tree Visualization



Considerations for Building a Tree

Binary or Multi-Valued

- Each split can be into 2 nodes or more than 2 nodes. Binary trees are easier to work with.
- Every *Polythetic* tree can be represented by an equivalent *Monothetic* tree.
- For example, the following tree can be transformed to the tree in the previous section:



- Most algorithms use *binary* trees, mainly, we will focus on the **CART** algorithm which is implemented in the Scikit-Learn library.
  - The CART algorithm produces only binary trees: nonleaf nodes **always** have 2 children (i.e. questions only have yes/no answers).

Choosing the Property (=Feature) to Test

- For a given point of splitting, what is the next feature the descision should build upon?
- *Simple* questions: each branch should consider **few features** (1 if it is binary)
- The first features to be chosen should be the ones that *eliminate* the most options
  - Which is to say, **increase the certainty of the decision**
  - Or, lead us to **nodes that are homogenous (pure)**

Stopping Criteria

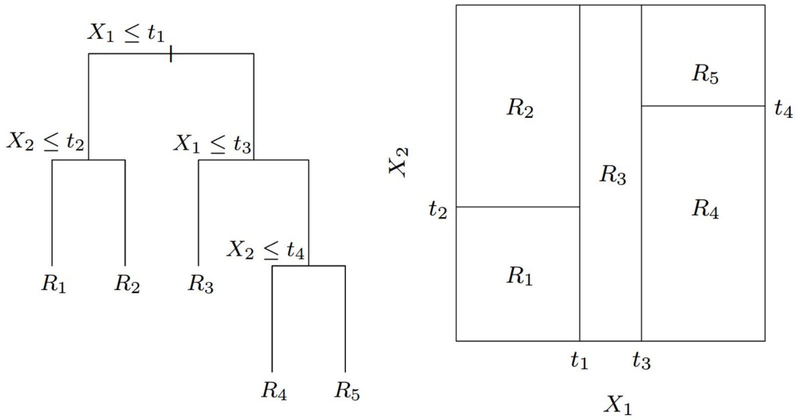
- When should a node be declared a *leaf*?

Pruning

- If the tree becomes too large (and thus, overfitting to the train dataset), how, and to what length, it can be made smaller and simpler?

Decision Boundaries (Also for Regression Tasks)

- In regression problems, the data is continuous, and boundaries have to be set such that the input falls within a certain range.
- These decision boundaries can easily drawn on a graph as in the following example:



Decision Trees for Regression Tasks

- In this tutorial we deal with classification tasks, where the input can be continuous features, but what if we want to complete a regression task, that is, output a value and not a class?
- Decision Trees are also capable of performing these tasks. We will not get into it, but the idea is the same.
- Instead of predicting a class, the tree predicts a value which is determined by the mean of the values in that branch.
- The CART algorithm is almost the same except for instead of minimizing the impurity, the MSE is minimized.
- You can read more about it on Scikit-Learn's documentation, under `DecisionTreeRegressor` .

Estimating Class Probabilities (Class Conditional Probabality)

- A Decision Tree can also estimate the probability that an instance belongs to a particular class k
  - First, it traverses the tree to find the leaf node for this instance
  - Then, it returns the ratio of *training* instances of class k in this node
  - In Scikit-Learn (after training) - `tree_clf.predict_proba([list of samples])`
- The class-conditional probabilities $P(c|D)$ are estimated by fitting a categorical MLE:

$$p_i = P(c|D) = \frac{1}{|D|}\sum_{i \in D} 1(y_i = c)$$

  - c - class
  - D - Data
  - Reminder - **Categorical Distribution** is a generalization of Bernoulli to multiple classes

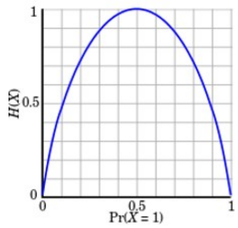
Impurity & Impurity Metrics

- **Simplicity** is the leading principle in the tree building process
- Therefore, we seek an attribute A at each node S that makes the immediate descendents as "pure" as possible.
- At each node S, the vector $P = (p_1, p_2, \dots, p_k)$, represents the probability (ratio) of samples belonging to class i
  - **Impurity** of node S - $\phi(P) \geq 0$
  - $\phi(P) = 0$ if all samples have the same label
    - That is, a node is "pure" if all training instances it applies to belong to the same class
  - $\phi(P)$ is *maximized* for **uniform distribution**
  - $\phi(P)$ is **symmetric** w.r.t. P
  - $\phi(P)$ is *smooth*, that is, differentiable everywhere

Impurity Metrics

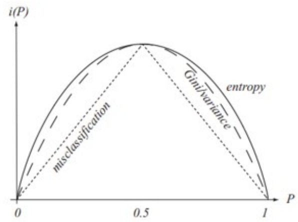
Entropy

- The concept of entropy originates in thermodynamics as a measure of *molecular disorder*. Entropy approaches 0 when molecules are still and well ordered.
- Later, it was introduced in the *information theory* domain, where the average information content of a message is measured (a reduction in entropy is called **information gain**). When all messages are identical, the entropy is zero.
- In Machine Learning, it is used as an impurity measure - a set's entropy is 0 when it contains instances of only one class.
- Given a system whose exact description is *unknown*, its entropy is defined as **the amount of information needed to exactly specify the state of the system**
  - If the system has k states, we would need $\log k$ bits to represent them

- Denote $p_{i,k}$ as the ratio of class k instances among the training instances in the i^{th} node, the entropy is:
$$H_i = \sum_{k=1}^{n} p_{i,k} \log \frac{1}{p_{i,k}} = -\sum_{k=1}^{n} p_{i,k} \log p_{i,k}$$
- [A friendly introduction to Information Theory (http://colah.github.io/posts/2015-09-Visual-Information/)](http://colah.github.io/posts/2015-09-Visual-Information/)
- 

Gini Index

- Gini is another common measure for impurity which can be intepreted as a *variance impurity*
- Denote $p_{i,k}$ as the ratio of class k instances among the training instances in the i^{th} node, the GiniIndex is:
$$G_i(P) = \sum_{k=1}^{n} p_{i,k}(1 - p_{i,k}) = 1 - \sum_{k=1}^{n} p_{i,k}^2$$
  - Reminder: given $X \sim Bern(p) \rightarrow Var(X) = p \cdot (1-p)$
- This is the expected error rate at node N if the category label is selected randomly from class distribution present at N



Gini vs. Entropy

- Scikit-Learn's default is **Gini**, but by changing the call to `criterion="entropy"` , the Entropy will be used to build the tree
- Most of the time, **both** lead to similar trees.
- **Gini** is (slightly) faster to compute
- However, sometimes they are different as Gini tends to isloate the most frequent class in its own branch of the tree, while Entropy tends to produce slightly more *balanced* trees

Choosing the Features/Attributes for the Splits

- Choose the one that most decreases impurity (and thus, increases purity)
- Given node S with probabilities P and discrete attribute A, the **impurity drop** is defined as:
$$\Delta\phi(S, A) = \phi(S) - \sum_{v \in Values(A)} p_v \phi(S_v)$$
  - For the **Binary** case:
$$\Delta\phi(S, A) = \phi(S) - p_L \phi(S_L) - (1 - p_L)\phi(S_R)$$
- For the *Entropy* measure, this measure is called **Information Gain (IG)** (or Mutual Information - MI):
$$IG(P) = H(P) - \sum_{v \in Values(A)} p_a H(p_a)$$
- For *Gini*, the measure is called **Gini Gain**

- We wish to **maximize** the information gain on the impurity drop, which is equivalent to **minimizing** the right expression.
- Denote a single feature as k and a threshold value as t_k (for example, height ≤ 30 cm) ,**the CART algorithm cost function** (which we wish to minimize):
$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} = p_L G_{left} + p_R G_{right}$$
where $G_{left/right}$ is the measure of impurity of the left/right subset and $m_{left/right}$ is the number of instances in the left/right subset.
- The tree building algorithm (CART) is *greedy*, it greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity several levels down. A greedy algorithm usually produces reasonably good solutions, but they are not guaranteed to be optimal.

Relation to Mutual Information (Information Gain)

- **Mutual Information (MI)** calculates the statistical dependence between two variables and is the name given to information gain when applied to variable selection.
- MI is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable.
- The mutual information between two random variables X and Y can be stated formally as follows:
$$I(X;Y) = H(X) - H(X \mid Y)$$
  - $I(X;Y)$ is the mutual information for X and Y, $H(X)$ is the entropy for X and $H(X \mid Y)$ is the conditional entropy for X given Y.
  - The result has the units of *bits*.
- MI is always **larger than or equal to zero**, where the larger the value, the greater the relationship between the two variables. If the calculated result is zero, then the variables are **independent**.

- Mutual Information and Information Gain are the same thing, although the context or usage of the measure often gives rise to the different names. For example:
  - Effect of Transforms to a Dataset (Decision Trees): Information Gain.
  - Dependence Between Variables (Feature Selection): Mutual Information.
- Notice the similarity in the way that the mutual information is calculated and the way that information gain is calculated, they are equivalent:
$$I(X;Y) = H(X) - H(X \mid Y)$$
$$IG(S,a) = H(S) - H(S \mid a)$$

Exercise 1 - Decsion Tree by Hand

As of September 2012, 800 extrasolar planets have been identified in our galaxy. Supersecret surveying spaceships sent to all these planets have established whether they are habitable for humans or not, but sending a spaceship to each planet is expensive. In this problem, you will come up with decision trees to predict if a planet is habitable based only on features observable using telescopes.

In the following table you are given the data from all 800 planets surveyed so far. The features observed by telescope are Size (“Big” or “Small”), and Orbit (“Near” or “Far”). Each row indicates the values of the features and habitability, and how many times that set of values was observed. So, for example, there were 20 “Big” planets “Near” their star that were habitable.

Size (Big / Small)	Orbit (Near / Far)	Habitable (Yes / No)	Count
Big	Near	Yes	20
Big	Far	Yes	170
Small	Near	Yes	139
Small	Far	Yes	45
Big	Near	No	130
Big	Far	No	30
Small	Near	No	11
Small	Far	No	255
190+, 160- / 184+, 266-	159+, 141- / 215+, 285-	374 / 426	800

Derive and draw the decision tree on this data (use the maximum information gain (entropy) criterion for splits, don’t do any pruning).

Solution 1

Let's first calculate the entropy of starting state:

$$H(Habitable) = -\frac{20+170+139+45}{800}\log_2\frac{20+170+139+45}{800} - \frac{130+30+11+255}{800}\log_2\frac{130+30+11+255}{800}$$

$$= -\frac{374}{800}\log_2\frac{374}{800} - \frac{426}{800}\log_2\frac{426}{800} = 0.4841 + 0.5183 \approx 1$$

Let's calculate the **Information Gain** from each split:

$$H(Habitable|Size) = \frac{190+160}{800}H(Size = Big) + \frac{184+266}{800}H(Size = Small)$$

$$= \frac{350}{800}\cdot(-\frac{190}{350}\log_2(\frac{190}{350}) - \frac{160}{350}\log_2(\frac{160}{350})) + \frac{450}{800}\cdot(-\frac{184}{450}\log_2(\frac{184}{450}) - \frac{266}{450}\log_2(\frac{266}{450}))$$

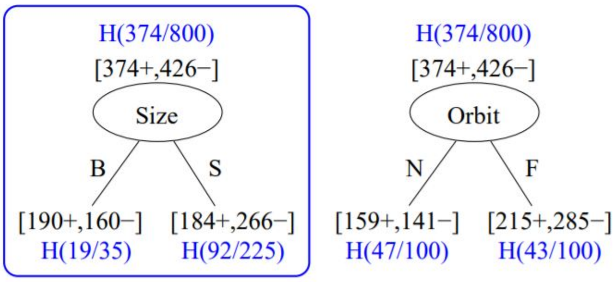
$$= 0.984 \to IG(Habitable|Size) = 1 - 0.984 = 0.016$$

$$H(Habitable|Orbit) = \frac{159+141}{800}H(Orbit = Near) + \frac{215+285}{800}H(Orbit = Far)$$

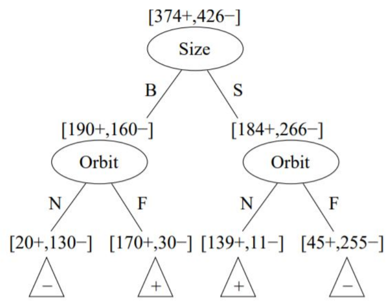
$$= \frac{300}{800}\cdot(-\frac{159}{300}\log_2(\frac{159}{300}) - \frac{141}{300}\log_2(\frac{141}{300})) + \frac{500}{800}\cdot(-\frac{215}{500}\log_2(\frac{215}{500}) - \frac{285}{500}\log_2(\frac{282}{500}))$$

$$= 0.9901 \to IG(Habitable|Orbit) = 1 - 0.9901 = 0.009$$

So the Information Gain from the size is larger, thus, the first split will be based on the Size feature.



The final tree looks like this:



[More Decision Trees Practice Excercises (https://profs.info.uaic.ro/~ciortuz/ML.ex-book/SLIDES/ML.ex-book.SLIDES.DT.pdf)](https://profs.info.uaic.ro/~ciortuz/ML.ex-book/SLIDES/ML.ex-book.SLIDES.DT.pdf)

Computational Complexity

This section's material goes beyond the scope of this course, but it is important to understand why we can't really build an **optimal** tree and what is the computational cost of training and making predicitons using the built tree.

- Denote m as the number of instances (the training set size) and n as the number of features.
- Finding the **optimal tree** is known to be an *NP-Complete* problem, as it requires $O(exp(m))$ time, making the problem intractable even for small training sets.
  - *P* is the set of problems that can be solved in polynomial time. *NP* is the set of problems whose solutions can be verfied in polynomial time. An *NP-Hard* problem is a problem which any *NP* problem can be reduced in polynimial time. An *NP-Complete* is both *NP* and *NP-Hard*.

- **Making predicitons** requires traversing the Decision Tree from root to a leaf. Decision Trees are generally approximately balanced, so traversing a Decision Tree requires going through roughly $O(\log_2(m))$ nodes. Since each node only requires checking the value of one feature, the overall prediciton complexity is just $O(\log_2(m))$, independent of the number of features (so prediciton is quite **fast**, even with large training sets).
- **Training or building the tree** requires comparing all features on all samples at each node. This results in a training complexity of $O(n \times m \log(m))$. For small training sets (less than a few thousands instances), Scikit-Learn can speed up training by presorting the data (set `presort=True`), but this slows down training considerably for larger training sets.

Highly-Branchig Attributes

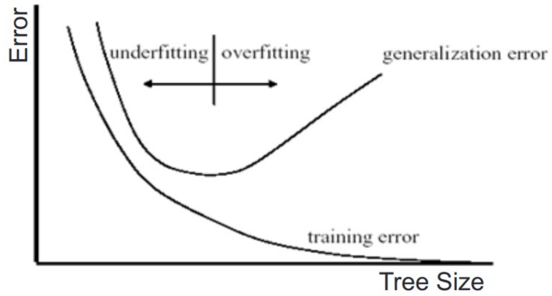
- **Problem**: attributes with a large number of values
  - Extreme case: each example has its own value. E.g. example ID, Date/Day attribute in weather data
- Information gain is **biased** towards choosing attributes with a large number of values
- This may cause:
  - **Overfitting** - selection of an attribute that is non-optimal for prediction
  - **Fragmentation** - data is fragmented into (too) many small sets

Possible Solution - Normalization of the Inforamtion Gain of Features

- Define **Split Information** of feature A as the intrinsic information of a split:
  - Entropy of distribution of instances into branches, or, how much information do we need to tell which branch an instance belongs to.
  - Formally:
$$SplitInformation(S, A) = -\sum_{v \in Values(A)} p_v log(p_v)$$
  - Observation: attributes with higher intrinsic information are less useful
  - Another way of defining it is $SplitInformation(S, A) = \log n(A)$, where $n(A)$ is the number of different values feature A can split over the group of samples S

- **Gain Ratio**
  - Modification of the information gain (IG) that reduces its bias towards multi-valued atributes.
  - Takes number of and size of branches into account when chossing an attribute
    - Corrects the IG by taking the intrinsic information of a split into account
  - Formally:
$$GR(S, A) = \frac{IG(S, A)}{SplitInformation(S, A)}$$

🚧 Pruning & Regularization Hyperparameters

- **Underfitting & Overfitting** - if we allow Decision Trees to grow until *absolute purity*, we will reach overfitting
  - 

  - Such model is often called a *nonparametric model*, not because it does not have many parameters (it often has a lot!), but because the number of parameters is not determined prior to learning (the tree keeps on growing).
- Decision Trees make very few assumptions about the training data (unlike linear models, which assume that the data is linear).

- To avoid **overfitting** the training data, we need to restrict the Decision Tree's freedom during training - this is called **regularization**.
- The methods:
  - **Stop Splitting** - stop growing a tree once a stopping criteria is reached
    - May suffer from the *horizon effect* - such a method will be biased towards trees in which greatest impurity reduction is near the root
  - **Prunning** - grow a full tree, then merge nodes until a stopping criteria is reached or deleting (=prunning) unneccessary nodes if the purity improvement it provides is not *statistically signifacnt*
    - Use Hypothesis Testing - going back to Tutorial 02, standard statistical tests, such as chi-square, $\chi^2 test$, are used to estimate the probability (*p-value*) that the improvement is purely the result of chance (the *null hypothesis*, H_0). If the *p-value* is higher than a given threshold (usually 5%), then the node is considered unneccessary and its children are deleted). The prunning continues until all unnccessary nodes have been pruned.

- **Common Stopping Criteria** (in practice):
  - *Minimum number of samples* in leaf nodes - **increasing** will regularize the model
    - In Scikit-Learn, set the input parameter `min_samples_leaf`
  - *Minimum samples split* - the minimum number of samples a node must have before it can split, **increasing** will regularize the model
    - In Scikit-Learn, set the input parameter `min_samples_split`
  - *Limit the depth of the tree* - **reducing** the max depth will regularize the model (less chance of overfitting)
    - In Scikit-Learn, set the input parameter `max_depth` (the default is `None` which means unlimited)
  - *Maximum number of leaves* - maximum number of leaf nodes, **reducing** will regularize the model
    - In Scikit-Learn, set the input parameter `max_leaf_nodes`
  - Using the validation accuracy, stop if no significant improvement
  - Also consider `min_weight_fraction_leaf` and `max_features` (more in the Scikit-Learn docs)

🚢 Example - Titanic Dataset

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

We will build a decision tree to decide what sorts of people were likely to survive.

To read more about the dataset and the fields - [Click Here (https://www.kaggle.com/c/titanic/data)](https://www.kaggle.com/c/titanic/data)

In [2]:
```
# let's load the titanic dataset and speratre it into train and test set
dataset = pd.read_csv('./datasets/titanic_dataset.csv')
# print the number of rows in the data set
number_of_rows = len(dataset)
num_train = int(0.8 * number_of_rows)
# reminder, the data looks like this
dataset.sample(10)
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
201	202	0	3	Sage, Mr. Frederick	male	NaN	8	2	CA. 2343	69.5500	NaN	S
333	334	0	3	Vander Planke, Mr. Leo Edmondus	male	16.00	2	0	345764	18.0000	NaN	S
639	640	0	3	Thorneycroft, Mr. Percival	male	NaN	1	0	376564	16.1000	NaN	S
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42	0	1	2625	8.5167	NaN	C
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.00	1	1	11751	52.5542	D35	S
571	572	1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.00	2	0	11769	51.4792	C101	S
99	100	0	2	Kantor, Mr. Sinai	male	34.00	1	0	244367	26.0000	NaN	S
783	784	0	3	Johnston, Mr. Andrew G	male	NaN	1	2	W./C. 6607	23.4500	NaN	S
73	74	0	3	Chronopoulos, Mr. Apostolos	male	26.00	1	0	2680	14.4542	NaN	C
653	654	1	3	O'Leary, Miss. Hanora "Norah"	female	NaN	0	0	330919	7.8292	NaN	Q

In [3]:
```python
# preprocessing
original_ds = dataset.copy() # Using 'copy()' allows to clone the dataset, creating a different object wit
h the same values
dataset['Has_Cabin'] = dataset["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
# Create new feature FamilySize as a combination of SibSp and Parch
dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1
# Create new feature IsAlone from FamilySize
dataset['IsAlone'] = 0
dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1
# Remove all NULLS in the Embarked column
dataset['Embarked'] = dataset['Embarked'].fillna('S')
# Remove all NULLS in the Fare column
dataset['Fare'] = dataset['Fare'].fillna(dataset['Fare'].median())

# Remove all NULLS in the Age column
age_avg = dataset['Age'].mean()
age_std = dataset['Age'].std()
age_null_count = dataset['Age'].isnull().sum()
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std, size=age_null_count)
# Next line has been improved to avoid warning
dataset.loc[np.isnan(dataset['Age']), 'Age'] = age_null_random_list
dataset['Age'] = dataset['Age'].astype(int)

# Define function to extract titles from passenger names
def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    # If the title exists, extract and return it.
    if title_search:
        return title_search.group(1)
    return ""

dataset['Title'] = dataset['Name'].apply(get_title)

# Group all non-common titles into one single grouping "Rare"
dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr', 'Major',
                                            'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

# mapping
# Mapping Sex
dataset['Sex'] = dataset['Sex'].map( {'female': 0, 'male': 1} ).astype(int)
# Mapping titles
title_mapping = {"Mr": 1, "Master": 2, "Mrs": 3, "Miss": 4, "Rare": 5}
dataset['Title'] = dataset['Title'].map(title_mapping)
dataset['Title'] = dataset['Title'].fillna(0)
# Mapping Embarked
dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)
# Mapping Fare
dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']  = 2
dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
dataset['Fare'] = dataset['Fare'].astype(int)

# Mapping Age
dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
dataset.loc[ dataset['Age'] > 64, 'Age']

# Feature selection: remove variables no longer containing relevant information
drop_elements = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']
dataset = dataset.drop(drop_elements, axis=1)
```

In [4]:
```
# show a sample of the preprocessed dataset
dataset.sample(10)
```

Out[4]:

	Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Has_Cabin	FamilySize	IsAlone	Title
359	1	3	0	1	0	0	2	0	1	1	4
505	0	1	1	1	0	3	1	1	2	0	1
566	0	3	1	1	0	0	0	0	1	1	1
878	0	3	1	1	0	0	0	0	1	1	1
843	0	3	1	2	0	0	1	0	1	1	1
884	0	3	1	1	0	0	0	0	1	1	1
685	0	2	1	1	2	3	1	0	4	0	1
851	0	3	1	74	0	0	0	0	1	1	1
64	0	1	1	2	0	2	1	0	1	1	1
373	0	1	1	1	0	3	1	0	1	1	1

In [5]:
```
# Take the relevant training data and labels
x = dataset.drop('Survived', axis=1).values
y = dataset['Survived'].values  # 1 for Survived, 0 for Dead
# shuffle
rand_gen = np.random.RandomState(0)
shuffled_indices = rand_gen.permutation(np.arange(len(x)))

x_train = x[shuffled_indices[:num_train]]
y_train = y[shuffled_indices[:num_train]]
x_test = x[shuffled_indices[num_train:]]
y_test = y[shuffled_indices[num_train:]]

print("total training samples: {}, total test samples: {}".format(num_train, number_of_rows - num_train))
```

```
total training samples: 712, total test samples: 179
```

In [6]:
```
# let's build a tree with no limitations
tree_clf = DecisionTreeClassifier(criterion='gini')
tree_clf.fit(x_train, y_train)
```

Out[6]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [7]:
```
# let's check the accuracy on the test set
y_pred = tree_clf.predict(x_test)
accuracy = np.sum(y_pred == y_test) / len(y_test)
print("accuracy: {:.3f} %".format(accuracy * 100))
```

```
accuracy: 75.419 %
```

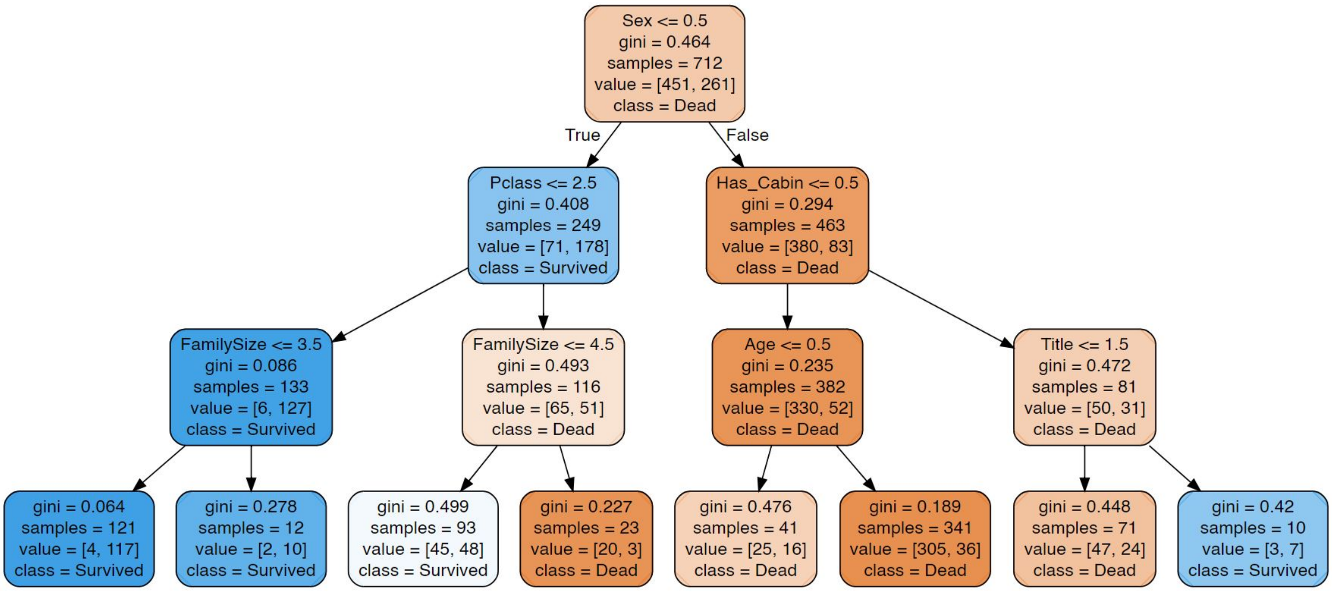
In [8]:
```
# let's add regularization
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=3)
tree_clf.fit(x_train, y_train)
# let's check the accuracy on the test set
y_pred = tree_clf.predict(x_test)
accuracy = np.sum(y_pred == y_test) / len(y_test)
print("accuracy: {:.3f} %".format(accuracy * 100))
```

```
accuracy: 83.799 %
```

In [9]:
```
# predicting probabilities
# for the first passenger in the test set, the probability of survival:
prob = tree_clf.predict_proba([x_train[0]])
print("passenger 0 chances of survival: {:.3f} %".format(prob[0][0] * 100))
```

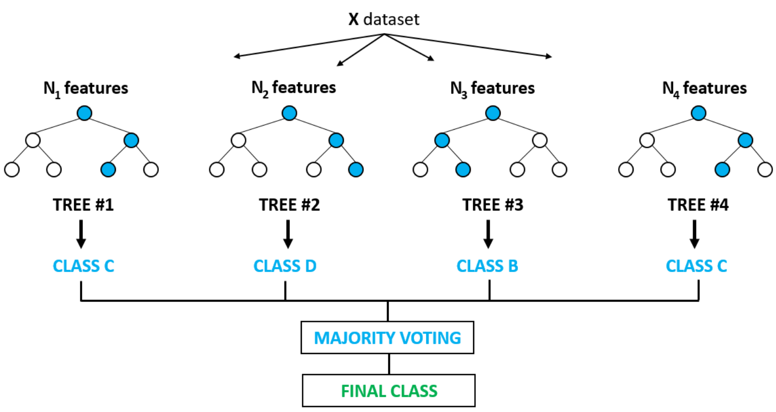
```
passenger 0 chances of survival: 89.318 %
```

In [ ]:
```
# generate an image graph
from sklearn.tree import export_graphviz
export_graphviz(tree_clf, out_file="./titanic_tree.dot", feature_names=dataset.columns.values[1:].tolist
(),
               class_names=['Dead', 'Survived'], rounded=True, filled=True)
# open the file with your favourite text editor and copy-pase it into http://www.webgraphviz.com/
```



Random Forest

- Random forests (or random decision forests) are an **ensemble learning** method for classification, regression and other tasks.
- They operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
  - We will talk on ensemble learning later in the course.
- Random decision forests try tend to decision trees' habit of overfitting to their training set.



- [Image Source (https://medium.com/@ar.ingenious/applying-random-forest-classification-machine-learning-algorithm-from-scratch-with-real-24ff198a1c57)](https://medium.com/@ar.ingenious/applying-random-forest-classification-machine-learning-algorithm-from-scratch-with-real-24ff198a1c57)

Random Forest Algorithm

- The training algorithm for random forests applies **bootstrap aggregating, or bagging** (tutorial 11), to tree learners.
- Given a training set $X = x_1, \dots, x_n$ with targets $Y = y_1, \dots, y_n$, bagging repeatedly (B times) selects a **random sample with replacement** of the training set and fits trees to these samples:
  - For $b = 1, \dots, B$:
    - Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
    - Train a classification or regression tree f_b on X_b, Y_b.
  - After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':
$$\hat{f} = \frac{1}{B}\sum_{i=1}^{B} f_b(x')$$
    - Or by taking the **majority vote** in the case of *classification* trees.

In [29]:
```python
from sklearn.ensemble import RandomForestClassifier
random_state= 42
rnd_clf = RandomForestClassifier(random_state=random_state, n_estimators=50, max_depth=3)
rnd_clf.fit(x_train, y_train)
# let's check the accuracy on the test set
y_pred = rnd_clf.predict(x_test)
accuracy = np.sum(y_pred == y_test) / len(y_test)
print("accuracy: {:.3f} %".format(accuracy * 100))
```

```
accuracy: 84.916 %
```

- This bootstrapping procedure leads to better model performance because it decreases the *variance* of the model, without increasing the *bias*.
- This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated.
- Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic).
  - Bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Decision Trees - [Machine Learning Lecture 29 "Decision Trees / Regression Trees" -Cornell CS4780 (https://www.youtube.com/watch?v=a3ioGSwfVpE)](https://www.youtube.com/watch?v=a3ioGSwfVpE)
- Decision Trees Example with Gini - [StatQuest: Decision Trees (https://www.youtube.com/watch?v=7VeUPuFGJHk)](https://www.youtube.com/watch?v=7VeUPuFGJHk)
- Information Gain, Mutual Information - [Decision Tree 4: Information Gain (https://www.youtube.com/watch?v=nodQ2s0CUbI)](https://www.youtube.com/watch?v=nodQ2s0CUbI)
- Random Forests - [Random Forest - Fun and Easy Machine Learning (https://www.youtube.com/watch?v=D_2LkhMJcfY)](https://www.youtube.com/watch?v=D_2LkhMJcfY)
  - Random Forests / Bagging - [Machine Learning Lecture 31 "Random Forests / Bagging" -Cornell CS4780 (https://www.youtube.com/watch?v=4EOCQJgqAOY)](https://www.youtube.com/watch?v=4EOCQJgqAOY)

Credits

- Based upon *Pattern Classification*, chapter 8
- [Information Gain and Mutual Information for Machine Learning (https://machinelearningmastery.com/information-gain-and-mutual-information/)](https://machinelearningmastery.com/information-gain-and-mutual-information/)
- Icons from [Icon8.com (https://icons8.com/)](https://icons8.com/) - [https://icons8.com (https://icons8.com)](https://icons8.com)
- Datasets from [Kaggle (https://www.kaggle.com/)](https://www.kaggle.com/) - [https://www.kaggle.com/ (https://www.kaggle.com/)](https://www.kaggle.com/)
- Examples and code snippets were taken from ["Hands-On Machine Learning with Scikit-Learn and TensorFlow" (http://shop.oreilly.com/product/0636920052289.do)](http://shop.oreilly.com/product/0636920052289.do)