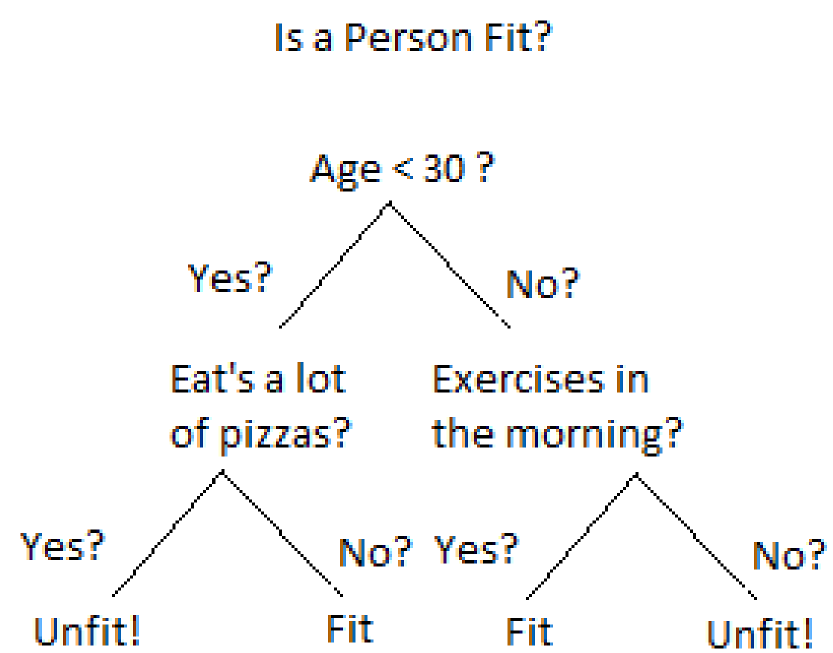
Decision Trees

Decision Trees

Decision Trees are a type of Machine Learning model

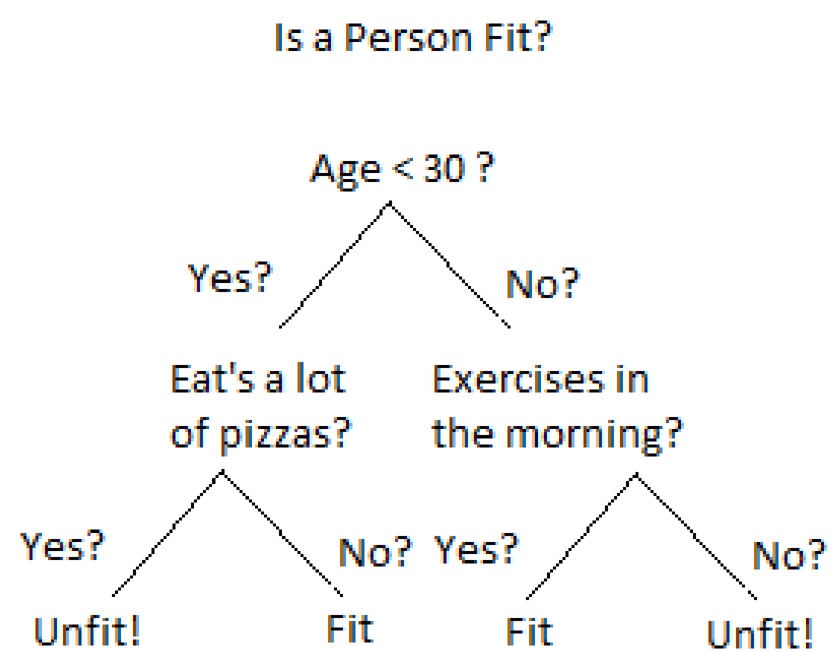
- They were originally introduced for classification tasks
- ...And they provide a prediction via recursive splitting



Decision Trees

Decision Trees are a type of Machine Learning model

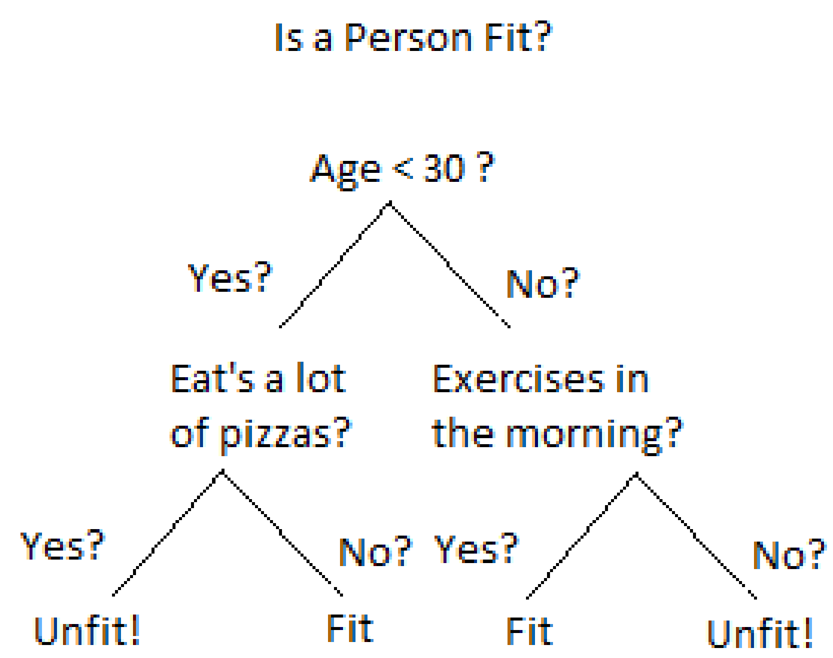
- Decision Trees consist of nodes, connected by parent-child relations
- There is a single root with no parent. Nodes with no child are called leaves



Decision Trees

Decision Trees are a type of Machine Learning model

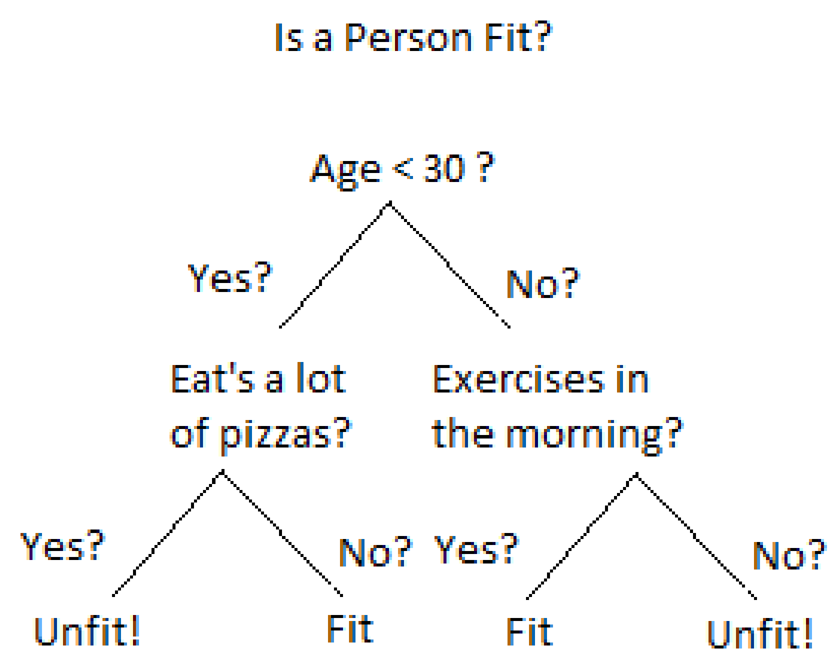
- The decision process always starts from the root
- ...And leaf nodes are labeled with a prediction



Decision Trees

Decision Trees are a type of Machine Learning model

- Non-leaves node correspond to a fork in the decision process
- ...When making predictions, a child is picked based on the value of one attribute



Decision Trees

Decision Trees are a type of symbolic ML models

...Actually, they are among the best examples of a symbolic technique

- They are interpretable
- They reason using discrete concepts
- They are easy to analyze

They are very versatile

- They can handle both categorical and numerical input
- They can handle inputs with missing values
- They can approximate non-linear relation

They serve as the basis for some of the most effective ML methods

...Such as Random Forests, Gradient Boosted Trees, and Extra Randomized Trees

Learning a Decision Trees

Decision trees are constructed via a recursive algorithm

- learn(x, y, n) :
  - if a stopping condition is met: :
    - return a leaf labeled with the majority class
  - if the termination condition is not satisfied :
    - pick an optimal attribute j and threshold θ
    - $n_{left} = \text{learn}(x_{x_j \le \theta}, y_{x_j \le \theta})$
    - $n_{right} = \text{learn}(x_{x_j > \theta}, y_{x_j > \theta})$
    - connect n_{left} and n_{right} to the n

The process starts by calling **learn** with the original training set and $n = \text{root}$

Learning Decision Trees

How do we evaluate an attribute and threshold?

Typically, we look at the uniformity of the resulting split

- We say that a j, θ is better
- ...If it leads to more uniform training sets in the children nodes

In detail:

- We consider the two vectors $y_{x_j \leq \theta}$ and $y_{x_j > \theta}$
- For each of them we compute a impurity index $H(y_{x_j \leq \theta})$ and $H(y_{x_j < \theta})$
- Then we average over the set size:

$$\frac{|y_{x_j \leq \theta}|}{|y|} H(y_{x_j \leq \theta}) + \frac{|y_{x_j > \theta}|}{|y|} H(y_{x_j > \theta})$$

In practice, there are a few important adjustments (we will not cover them)

Learning Decision Trees

Common impurity criteria include

The Gini index:

$$H(y) = \sum_{k \in K} p_k (1 - p_k)$$

The information entropy

$$H(y) = -\sum_{k \in K} p_k \log(p_k)$$

The misclassification index:

$$H(y) = 1 - \max(p_k)$$

In all notations, $\boldsymbol{p_k}$ is the frequency of class $\boldsymbol{k}$ in the output vector $\boldsymbol{y}$

Learning Decision Trees

How do we get the attribute and threshold to be evaluated?

We start with a main observation

- Two thresholds θ' and θ'' actually make a difference
- ...Only if they lead to different splits

So we can actually enumerate all attribute/threshold combinations!

- We loop over all the attributes
- We consider all the values for the attributes in the training input data x
- ...And we evaluate all the resulting splits

At the end of the process we have the best j, θ pair

- It may seem like an expensive calculation
- ...But in fact it can be performed very quickly

Learning Decision Trees

The termination condition has some flexibility

- We stop after a certain depth
- We stop if there are not enough examples
- We stop if there is no way to obtain children with enough examples

By tweaking the conditions we can prevent overfitting

Decision trees can handle missing values in the dataset

- If we need to split on attribute j, which is missing for an example
- ...Then we consider fractions of that example
  - The fractions depend on how attribute j is distributed for the known examples
- One fraction goes in $\hat{x}_{x_j \leq \theta}$, the other in $\hat{x}_{x_j > \theta}$

A Practical Example

Loading and Preparing the Data

Let's test the approach on the `weather.csv` dataset

We start by loading the data and encoding the categorical attributes:

In [2]:
```
data = pd.read_csv('data/weather.csv', sep=',')
data['windy'] = data['windy'].astype('category').cat.codes
data['play'] = data['play'].astype('category').cat.codes
data['outlook'] = data['outlook'].astype('category').cat.codes
data.head()
```

Out[2]:

	outlook	temperature	humidity	windy	play
0	2	85	85	0	0
1	2	80	90	1	0
2	0	83	86	0	1
3	1	70	96	0	1
4	1	68	80	0	1

- There's no need to use a one-hot encoding for outlook
- ...Soince with the splitting mechanism a categorical encoding is enough

Loading and Preparing the Data

Then we separate the training and test set

In [3]:
```
from sklearn.model_selection import train_test_split

input_cols = [c for c in data.columns if c != 'play']
X, y = data[input_cols], data['play']
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.34, random_state=0)

print(f'#examples: {len(X_tr)} (training), {len(X_ts)} (test)')
```

```
#examples: 9 (training), 5 (test)
```

There no need to normalize the input data

- Not even from an interpretation purpose!
- We'll get to that later :-)

Learning a Tree

We will use scikit-learn to learn a DT

First, we build the model:

In [4]:
```
from sklearn.tree import DecisionTreeClassifier
mdl = DecisionTreeClassifier()
```

- Special termination conditions can be specified when building the object

Then we call the fit method:

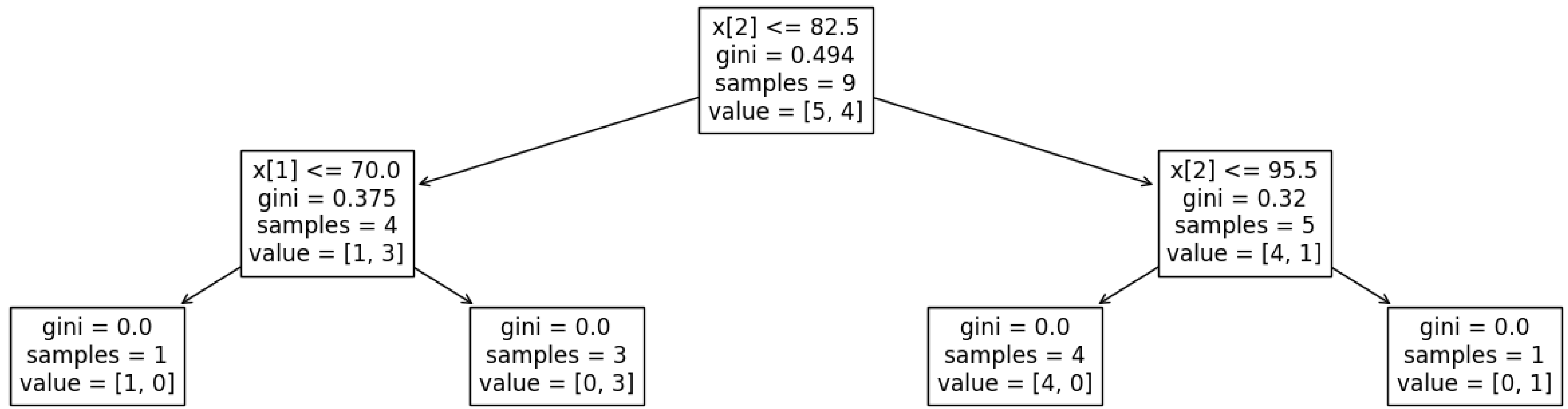
In [5]:
```
mdl.fit(X_tr, y_tr);
```

- The process is the same we used for Linear Regression
- Actually, all scikit-learns model have the same basic API

Plotting the Tree

We can now have a look at the trained tree

In [6]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=figsize)
plot_tree(mdl);
plt.tight_layout(); plt.show()
```



Evaluting the Tree

Our DT can be evaluated as any other classification model

In [7]:
```
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
ConfusionMatrixDisplay.from_estimator(mdl, X_tr, y_tr, display_labels=['not play', 'play'],
```