CSC 461: Machine Learning

Fall 2024

Decision Trees

Prof. Marco Alvarez, Computer Science
University of Rhode Island

Learning a decision tree

Setup

- Data instances
  - $x \in \mathbb{R}^d$ is typically a **feature vector** of discrete values
    - however, continuous values can also be handled
  - $y \in \{1,2,\ldots,k\}$ for classification and $y \in \mathbb{R}$ for regression
- Hypothesis
  - each solution (hypothesis) is a **decision tree**

$$h : \mathcal{X} \mapsto \mathcal{Y}, h \in \mathcal{H}$$

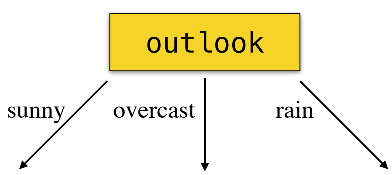
Learning approach

- **top-down** tree construction
  - select best feature to split on
  - for each feature value => split the data into subsets and create child nodes
  - recursively apply these steps to child nodes
- Greedy algorithm
  - makes locally optimal choice at each step
  - efficient, but may lead to suboptimal solutions
  - cannot guarantee optimality (smallest consistent tree)

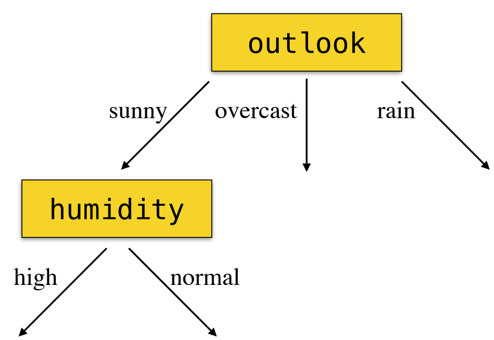
DT induction

- Start at the root node with entire dataset
- Create a decision node based on the best feature
  - best feature chosen by information gain, Gini impurity, variance reduction (for regression) or other
  - split the dataset according to feature values and create child nodes for each data split
- Repeat last step (**recursively**) for each child node
  - use subset of data that reached the node
  - stop the recursion when meeting a criterion (e.g., all data in node belongs to one class, maximum tree depth reached, or minimum samples per leaf reached)

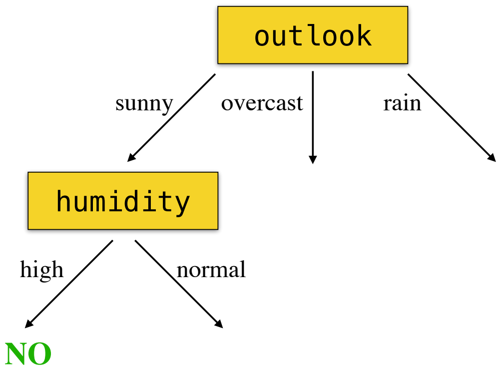
Outlook	Temperature	Humidity	Wind	Play
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no



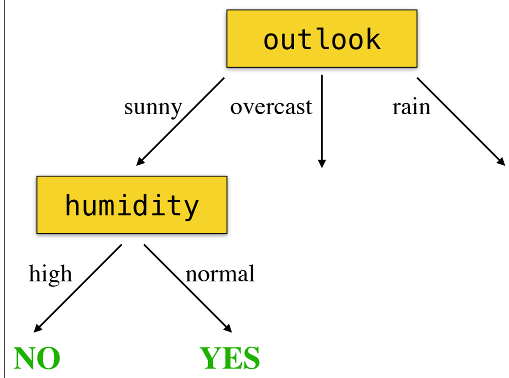
Outlook	Temperature	Humidity	Wind	Play
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no



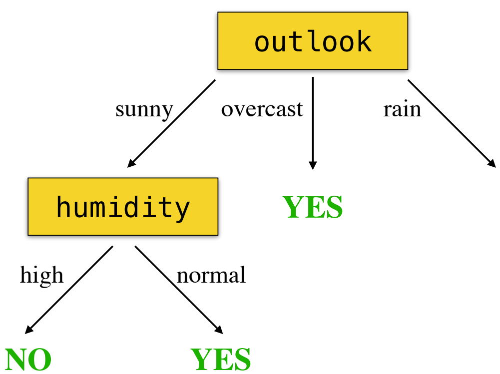
Temperature	Humidity	Wind	Play
hot	high	weak	no
hot	high	strong	no
mild	high	weak	no
cool	normal	weak	yes
mild	normal	strong	yes



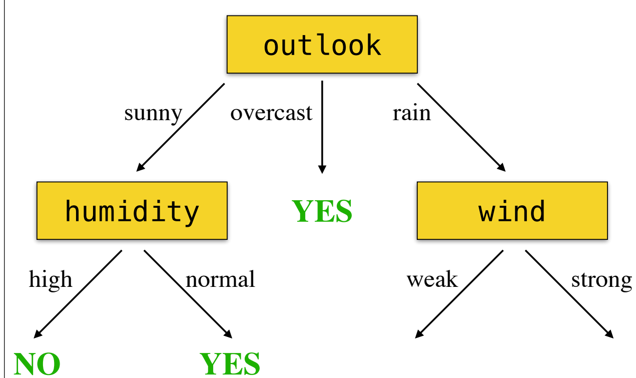
Temperature	Wind	Play
hot	weak	no
hot	strong	no
mild	weak	no



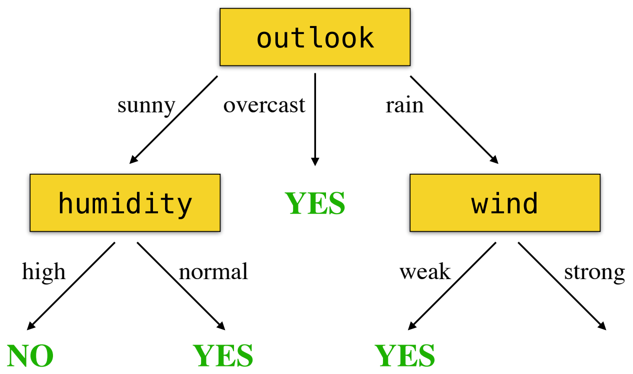
Temperature	Wind	Play
cool	weak	yes
mild	strong	yes



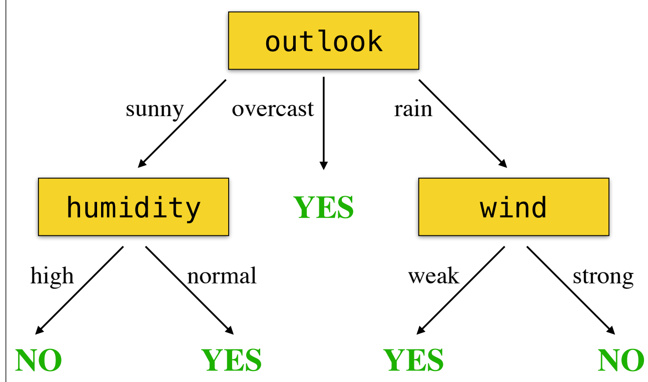
Temperature	Humidity	Wind	Play
hot	high	weak	yes
cool	normal	strong	yes
mild	high	strong	yes
hot	normal	weak	yes



Temperature	Humidity	Wind	Play
mild	high	weak	yes
cool	normal	weak	yes
cool	normal	strong	no
mild	normal	weak	yes
mild	high	strong	no



Temperature	Humidity	Play
mild	high	yes
cool	normal	yes
mild	normal	yes



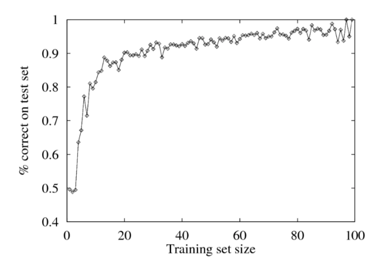
Temperature	Humidity	Play
cool	normal	no
mild	high	no

Show me the code

```
def create_tree(tree, data_X, data_y, att_list, depth, max_depth):
    node_id = depth + random.randint(0,10e10)
    # stop condition, then create a node with the majority label
    if data_y.nunique() == 1 or len(att_list) == 0 or depth == max_depth:
        tree.add_node(node_id, label=data_y.mode()[0])
    else:
        # select attribute with largest IG
        best = best_att(data_X, data_y, att_list)
        tree.add_node(node_id, label=best)
        new_atts = [x for x in att_list if x != best]
        # create branches
        for val in data_X[best].unique():
            idx = data_X[best] == val
            if idx.shape[0] > 0:
                id = create_tree(tree, data_X.loc[idx], data_y.loc[idx],
                                 new_atts, depth+1, max_depth)
                tree.add_edge(node_id, id, label=val)
    return node_id
```

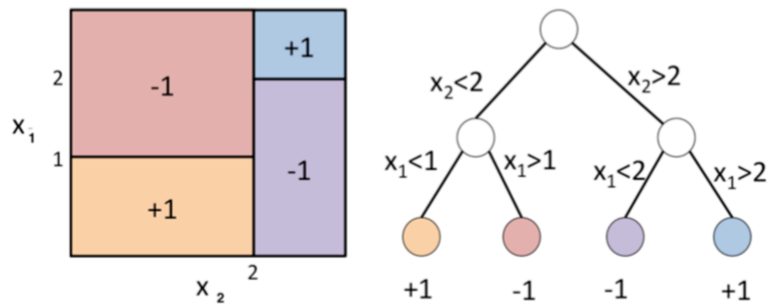
Resulting decision tree

- Preference for smaller trees
  - reduced overfitting, improved generalization and interpretability
- Consistency with training examples
  - tree aims for consistency, however it may lead to overfitting especially on noisy data
- Agreement with true function
  - may not always agree due to limited training data, noise in the data, overfitting
  - larger training datasets generally lead to better approximation



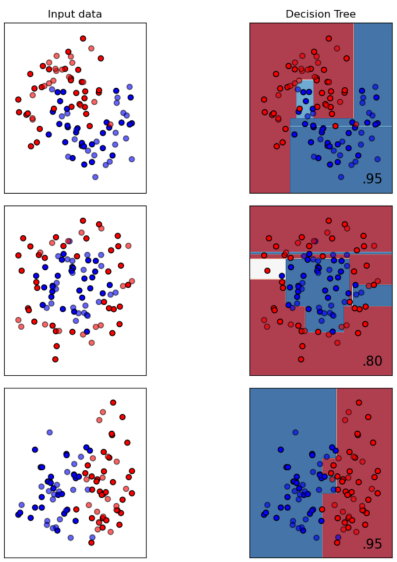
Final remarks

Nonlinear decision boundary



from: CS260 Machine Learning Algorithms, Cho-Jui Hsieh, UCLA, 2019

Nonlinear decision boundary



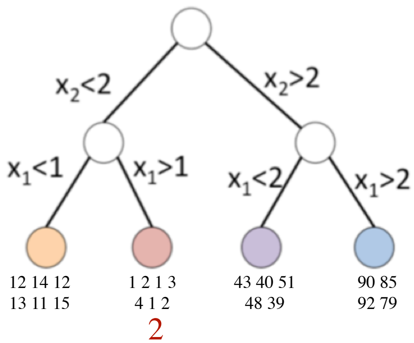
Continuous features

- Transform continuous into discrete features
  - use thresholds defined by domain experts or automatically calculate from training data
- For example:
  - sort values in training set
  - find split points where class changes

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

Continuous outputs

- Regression trees
  - assign continuous values to leaves
  - e.g., the **mean** of all *y* values that fall into the leaf



Preventing overfitting

- Remove irrelevant features
- Increase dataset size
- Stop growing branches during training (early stopping)
  - use hard thresholds or statistical measures
- Post-training pruning
  - remove branches that don't significantly contribute to performance

Conclusion

- Advantages
  - nonlinear decision boundaries
  - interpretability
  - fast inference
- Limitations:
  - training can be computationally expensive
  - prone to overfitting without proper regularization
  - instability: small data changes can result in very different trees
  - some complex functions require exponentially large trees (e.g., majority, parity functions)