Lecture 11: Decision Trees

Introduction to Learning and Analysis of Big Data
**Decision Trees**

- **Decision trees** are a natural model of decision-making.
- A decision tree is a predictor \( h : \mathcal{X} \rightarrow \mathcal{Y} \) that is defined by traveling from a root of a tree to a leaf.

![Decision Tree Diagram]

- Sweet?
  - No
    - Color?
      - Red: **Tomato**
      - Green: **Cucumber**
  - Yes
    - Hard?
      - No
        - Color?
          - Green: **Grape**
      - Yes
        - Color?
          - Red: **Strawberry**
          - Yes: **Apple**
Decision trees

- Tree inner nodes match attributes (or tests) on $\mathcal{X}$.
- Common test type: $\mathbb{I}[x(i) \leq \theta]$ for some threshold $\theta$.
- Each inner node has children for each possible result of the test.
- Tree leaves match a predicted label.
- Same label can repeat in several leaves!
  - Illustration: Should a person be admitted to hospital?
  - Use attributes such as blood pressure, fever, heart-rate.
A hypothesis class for decision trees

- Assume for simplicity $\mathcal{X} = \{0, 1\}^d$, $\mathcal{Y} = \{0, 1\}$.
- Goal: Learn a decision tree with inner nodes of the form “$x(i) = 1$?”.
- Can we learn using the hypothesis class of all possible decision trees?
- Problem: This allows all functions $f : \mathcal{X} \to \mathcal{Y}$. We will get overfitting.
- Solution: Regularize by restricting the size of the tree.
- $\mathcal{H}_n :=$ decision tree with at most $n$ internal nodes and leaves.
- What is the size of $\mathcal{H}_n$?
  - For each node, select feature $i$ to test, or mark it as a leaf with 0 or 1 (some options are not legal, this is an upper bound)
  - Each node has $\leq d + 2$ options.
  - $|\mathcal{H}_n| \leq (d + 2)^n$.
  - Sample complexity is $O(\log(|\mathcal{H}_n|)) \leq O(n \log d)$.
- Bias-complexity trade-off:
  - Larger $n$: Smaller approximation error, larger estimation error
  - Smaller $n$: Larger approximation error, smaller estimation error.
Learning a decision tree

- Would like to do ERM on $\mathcal{H}_n$ for the best possible $n$.
- ERM on decision trees in NP-hard 😞.
- Also, we don’t know what $n$ should be 😞.
- Solution: Use a heuristic to find a good tree for $S$.
- If:
  - Found a tree $h$ with low $err(h, S)$, and
  - Sample size is large enough for the size of the tree,
- Then: $err(h, D)$ will be low (since it will be similar to $err(h, S)$).
A greedy heuristic: ID3

- ID3: Learn a decision tree greedily, starting from the root.
- Gain(S, i): a function that estimates the improvement of the tree error on S if a split using attribute i is used.
- In each iteration decide which attribute to use based on Gain(S, i).
- Stop when tree cannot be improved by adding splits.
- The algorithm is recursive.

**ID3 algorithm**

**input**  Training sample S, feature subset A ⊆ {1, ..., d}.

**output** A tree for S using only attributes in A.

1. if all of S is labeled y ∈ {0, 1}, return a leaf labeled y.
2. if A = ∅, return a leaf labeled with the majority label on S.
3. Let $j = \arg\max_{i \in A} \text{Gain}(S, i)$.
4. For $a \in \{0, 1\}$, let $S_a = \{(x, y) \in S \mid x(j) = a\}$.
5. Return a tree with root j, left child ID3(S₀, A \ {j}), and right child ID3(S₁, A \ {j}).
A greedy heuristic: ID3

- ID3: Learn a decision tree greedily, starting from the root.
- \( \text{Gain}(S, i) \): a function that estimates the improvement of the tree.
- Options for \( \text{Gain} \) function:
  - **Improvement in sample error after split.**
    - Define \( C(a) := \min\{a, 1-a\} \). Denote: \( P_S \) is the “probability” on \( S \).
    - Sample error without the split:
      \[
      \text{err}_{\text{before}} := C(P_S[Y = 1]).
      \]
    - Sample error with the split:
      \[
      \text{err}_{\text{after}} := P_S[X(i) = 1]C(P_S[Y = 1 | X(i) = 1]) + P_S[X(i) = 0]C(P_S[Y = 1 | X(i) = 0]).
      \]
    - Define \( \text{Gain}(S, i) := \text{err}_{\text{before}}(S) - \text{err}_{\text{after}}(S, i) \).
  - **Information Gain**: Same as above, except
    \( C(a) := -a \log(a) - (1-a) \log(1-a) \). (entropy). Used by popular algorithms such as ID3 and C4.5.
  - **Gini Index**: Same as above, except \( C(a) := 2a(1-a) \). Used by popular CART algorithm.
A greedy heuristic: ID3

- Which gain function is the best?
- Problem is NP-hard — no gain function guarantees optimal (smallest) tree.
- Can prove some guarantees for Information gain and Gini index.
- Information gain and Gini index are used more in practice.
Avoiding Overfitting

- ID3 tries to find a tree with a small sample error
- This tree can be very large $\implies$ danger of overfitting.
- Solution: restrict size of tree.
  - Can decide on tree size in advance, or
  - Can learn the whole tree, then prune
- Pruning: Remove sub-trees if it doesn’t hurt the sample error too much.
- Idea: If tree is smaller but sample error remains the same, error on distribution should be improved (less overfitting).
Pruning

- Assume some function $f(T)$, which estimates the true error of the decision tree $T$ on the distribution.
  - $f$ can be based on PAC analysis and the size of the tree
  - $f$ can be estimated using a validation sample

- Start with a tree $T$.

- For each node $j$ in the tree (starting from the bottom), check the following options:
  - Do nothing
  - Replace node $j$ with leaf with label 0
  - Replace node $j$ with leaf with label 1
  - Replace node $j$ with left sub-tree
  - Replace node $j$ with right sub-tree

- In each node, choose the option that gives the smallest value of $f$.

- Return the new tree $T'$.
Real-valued features

- When examples have real valued features, there are many possible tests for each feature.
- Considering only threshold tests: For feature $i$ and threshold $\theta$, can use the test: $\mathbb{I}[x(i) \leq \theta]$
- For sample $S$ of size $m$, for each feature $i$ there are at most $m + 1$ thresholds that induce different splits. Denote them $\theta_{1,i}, \ldots, \theta_{m+1,i}$.
- Define a new binary feature for each $i \leq d, j \leq m + 1$: For example $x$, value of feature is $\mathbb{I}[x(i) \leq \theta_{j,i}]$.
- Run ID3 on the $(m + 1)d$ binary features.
- There are more efficient implementations, because the binary features have a special structure.
Random Forests

- Another approach that reduces overfitting: Random Forests.
- A random forest is a classifier consisting of several decision trees.
- The prediction of the random forest is the majority vote of predictions of each tree in the forest.
- Assume a decision tree learning algorithm that accepts a vector of parameters $\theta$.
- A popular choice: $\theta$ defines which features are allowed to be used in each level of the tree.
- Each tree in the forest is learned as follows:
  - Select some random vector $\theta$ based on some distribution.
  - Learn a decision tree on $S$ using the algorithm with input $\theta$.
- This process generates many different trees.
- Idea: the collection of trees is more robust than any one tree.
- Random forests are very successful on many practical problems.
- Their success depends on many fine details that we did not cover.
Summary

- Decision trees are a popular model used for learning.
- The hypothesis class of all decision trees is too large.
- Learning the optimal decision trees is NP-hard.
- In practice
  - Use a greedy algorithm for growing the decision tree,
  - Prune the decision tree after learning it.
  - Or, use a random forest: a collection of different trees.