CS229
Decision Trees
May 14, 2021
Decision Trees: nonlinear classifier
Decision Trees: canonical situation

- No linear separation line
- Want to divide input space into “regions”
- Can do this by dividing input space into disjoint regions $R_i$

\[ x = \bigcup_{i=0}^{n} R_i \]

s.t. $R_i \cap R_j = \emptyset$ for $i \neq j$
Recursively splitting regions

- Parent region $R_p$
- “Children” regions $R_1$ and $R_2$
- Split on feature $X_j$

\[
R_1 = \{ X \mid X_j < t, X \in R_p \}
\]
\[
R_2 = \{ X \mid X_j \geq t, X \in R_p \}
\]
(b)
How ‘good’ is a split?

- Need to define a loss function $L$ on a region
- Loss of the parent region $L(R_p)$ must be higher than that of child regions $R_1$ and $R_2$
- When deciding which attribute to split on, pick the one which maximizes the ‘gain’ in the loss
  - Greedy splitting

\[
L(R_p) = \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}
\]
Why greedy splitting?

- Checking every possible way of splitting every single feature in every possible order is computationally intractable!
- Greedy splitting is much easier: just compute the loss for each feature you want to consider splitting on
Entropy loss

- Looks like the cross-entropy loss that you have seen before
- $\hat{p}_c$ is the prevalence of class c in region R
- $L_{cross}(R) = 0$ if all the data in region R belongs to a single class

$$L_{cross}(R) = - \sum_c \hat{p}_c \log_2 \hat{p}_c$$
Entropy loss

- Note that the entropy loss is convex
- Can be shown that, under reasonable conditions, weighted average of children’s loss is always less than parent’s loss

\[ L(R_p) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|} \]
Common alternative: Gini impurity

- Closely related to entropy loss
- Default splitting loss for many ML libraries like scikit-learn

\[
I_G(\hat{p}) = \sum_{i=1}^{c} \hat{p}_i \left( \sum_{k \neq c} \hat{p}_k \right) = \sum_{i=1}^{c} \hat{p}_i (1 - \hat{p}_i)
\]
What about regression?

- Same growth process, but final prediction is now the mean of all datapoints in region:
  \[ \hat{y} = \frac{\sum_{i \in R} y_i}{|R|} \]

- Use least-squares loss to split:
  \[ L_{squared}(R) = \frac{\sum_{i \in R} (y_i - \hat{y})^2}{|R|} \]
Regularization

- Decision trees are highly prone to overfitting! High variance, low bias
- **Minimum leaf size**
  - Do not split R if its cardinality falls below a fixed threshold
- **Maximum depth**
  - Do not split R if more than a fixed threshold of splits were already taken to reach R
- **Maximum number of nodes**
  - Stop if a tree has more than a fixed threshold of leaf nodes
Runtime Complexity

- n examples, f features and a tree of depth d
- Test time complexity: $O(d)$
  - If balanced tree, $O(d)=O(\log n)$
- Train time complexity: $O(nfd)$
  - Relatively fast since data matrix size is $O(nf)$
Decision trees lack “additive” structure
Random Forests

- Decision trees are prone to overfitting, so use a randomized ensemble of decision trees
  - Typically works a lot better than a single tree
- Each tree can use feature and sample bagging
  - Randomly select a subset of the data to grow tree
  - Randomly select a set of features
  - Decreases the correlation between different trees in the forest
Live Demo!
A few words about boosting...

- Iteratively add simple “weak” classifiers to improve classification performance
- After adding weak classifier, evaluate performance and reweight training samples
- Weak classifier can be decision tree of depth 1 (decision stump)
- Theoretically, can achieve zero training loss!
- Python libraries: LightGBM, XGBoost
- More in the boosting pdf notes!
Thank you