Decision Trees

Decision Trees lean a hierarchy of if/else questions leading to a decision. Each question is usually represented as a terminal node/leaf that contains an answer.

- Tests - The sequence of if/else questions that get us to the true answer the quickest

Usually decision tree models don't handle binary yes or no data, but continuous data, with the tests usually being "is feature *i* larger than value *a*". To build a tree the algorithm recursively searched over all possible tests and finds the one that is most informative about the target value, e.g. for classification tasks the trend of features of each class. Predictions are made by running data through the generated decision tree to discern which theoretical partition of a feature space the point would lie in, and then prediction the majority target (.i.e target with the most similar features) in that region.

Usually building a tree this way, recursively until all leaves point to a single target leads to models that are very complex and highly overfit to training data. The mere presence of pure leaves means that a tree is 100% accurate on the training set and thus usually abysmal generalization performance.

- Pure leaves

Preventing Overfitting

- Pre-Pruning - Stopping the creation of the tree early
- Post-Pruning/Pruning - Building the tree then removing/collapsing nodes that control little information

Feature Importance A summary method that rates how important each feature is for the decision a tree makes. It is assigning a number between 0 and 1 for each feature, where zero means not used at all and 1 means perfectly predicts the target. The feature importances always sum to 1.

Regression

Decision trees for regression are mostly similar to classification with `DescisionTreeRegressor` being the implementation in `scikit-learn`. However tree based regression models are not able to extrapolate or make predictions outside of the range of the training data, i.e tree regressors are not able to predict responses outside of what it saw in training data.

Strengths, Weaknesses, and Parameters

Parameters

- `max_depth`, `max_leaf_nodes` and `min_samples_leaf` - Pre-pruning. Used to limit the number of branches the tree creates.

Strengths

- The resulting model can be easily visualized and understood by non experts.
- Scales very well as algorithms are completely invariant to scaling - Each feature is processed separately.
- Works well with a mix of binary and continuous features.

Weaknesses

- Even with the use of pre-pruning they tend to overfit and thus provide poor generalization performance. As a result ensembles of decision trees are used in place of a single one

Ensembles of Decision Trees

Ensembles

> Methods that combine multiple machine learning models to create more powerful models.

Random Forests

> A collection of decision trees where each tree is slightly different from the others.

The idea behind this is that each tree might to a relatively good job of predicting but will likely overfit on a part of the data. If we have many trees that work well and overfit in different ways we can reduce the amount of overfitting by averaging their results.

To build a random forest, each decision tree build should do an acceptable job of predicting the target and be different from the other trees. We get build trees to have some randomness by:

- Selection the data points used to build a tree
- Selection the features in each split test

  ###### Building

- Number of trees to build - `n_estimators`
- Bootstrap sample

  - Taking a data point randomly from `n_samples`, `n_samples` times.

- The number of trees is built using the bootstrap sample

Strengths, Weaknesses and Parameters Parameters

- `n_estimators` - Number of trees to build. Larger = better
- `n_jobs`- Used to set the number of cores for CPU core parallelization.
- `max_features` - Determines how random each generated tree is. `max_features = sqrt(n_features)` for classification and `max_features = n_features` for regression.

Strengths

- Work well without heavy tuning of parameters
- Doesn't require data scaling
- Works well on very large data sets
- Training can easily be parallelized over many CPU cores

Weaknesses

- Basically Impossible to interpret a large number of trees in detail
- Don't tend to perform on very high dimensional, sparse data such as text data.
- Require more memory and are slower to train and predict than linear models.

Gradient Boosted Regression Trees (GBRT)

> Works by building trees in a serial manner where each tree tries to correct the mistakes of the previous one.

By default there is no randomization in GBRTs, instead strong pre-pruning is used. GBRTs usually use shallow trees of depth one to five making the model smaller in terms of memory and makes predictions faster. The idea of GB is to combine many simple models (weak learners).

An important parameter of GB is the `learning_rate` which controls how strongly each tree tries to correct the mistakes of previous trees. A higher learning rate means each tree makes stronger corrections allowing for more complex models.

Strengths, Weaknesses, and Parameters Parameters

- `n_estimators`
- `learning_rate` - Controls how strongly each tree tries to correct the mistakes of previous trees

- `max_depth` - Usually set very low, not deeper than five splits

Strengths

- Works well without scaling

Weaknesses

- Require careful tuning of parameters
- Can take a long time to train