Machine Learning 2018 – Ensemble Methods

Kien C Nguyen

January 2, 2018

Contents

Decision Trees – Introduction

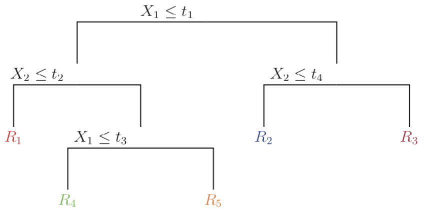
- Also called classification and regression trees or CART models
- The input space is recursively partitioned, and a local model is defined in each resulting region.
- We can represent this structure with a tree, with one leaf per region.
- A mean output is associated with each of these regions
- We then have a piecewise constant surface.
- The model can be written as

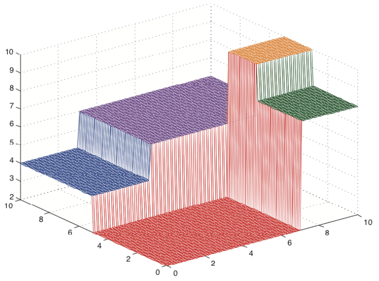
$$f(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \sum_{m=1}^{M} w_m \mathbb{I}(\mathbf{x} \in \mathbb{R}_m) = \sum_{m=1}^{M} w_m \phi(\mathbf{x}; \mathbf{v}_m)$$

R_m is the m'th region, w_m is the mean output in this region, $\mathbf{v}_m$ encodes the choice of variable to split on and the threshold value, on the path from the root to the m'th leaf.

Decision Trees

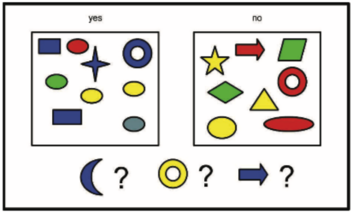
Figure: Decision Trees. (Source: K. Murphy [1])



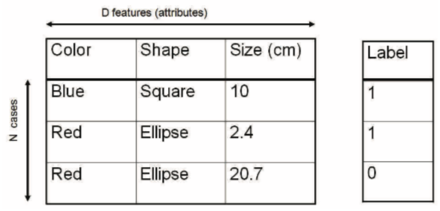


Decision Trees

Figure: Some labeled training examples of colored shapes, along with 3 unlabeled test cases. (Source: K. Murphy [1])



(a)



D features (attributes)
N cases

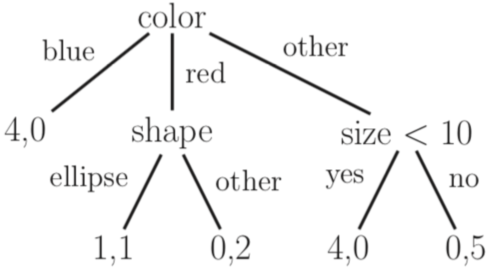
Color	Shape	Size (cm)
Blue	Square	10
Red	Ellipse	2.4
Red	Ellipse	20.7

Label
1
1
0

(b)

Decision Trees

Figure: A simple decision tree for the data in Figure 1.1. A leaf labeled as (n_1, n_0) means that there are n_1 positive examples that match this path, and n_0 negative examples. In this tree, most of the leaves are "pure", meaning they only have examples of one class or the other; the only exception is leaf representing red ellipses, which has a label distribution of (1, 1). We could distinguish positive from negative red ellipses by adding a further test based on size. However, it is not always desirable to construct trees that perfectly model the training data, due to overfitting. (Source: K. Murphy [1])



Decision Tree Regression – Regression cost

- The cost can be written as

$$C(D) = \sum_{i \in D} (y_i - \bar{y})^2$$

  where $\bar{y} = \frac{1}{|D|} \sum_{i \in D} y_i$ is the mean output in set D
- Alternatively, we can also use a linear regression model for each leaf, using the features that have been used on the path from the root to the leaf and calculate the residual errors.

Decision Tree Regression – Classification cost

- For a test $X_j < t$, the class-conditional probabilities are given as

$$\hat{\pi}_c = \frac{1}{|D|} \sum_{i \in D} \mathbb{I}\{y_i = c\}$$

- Misclassification rate: The most probable class label is given by $\hat{y}_c = argmax_c \ \hat{\pi}_c$

$$\frac{1}{|D|} \sum_{i \in D} \mathbb{I}\{y_i \neq \bar{y}\} = 1 - \hat{\pi}_c$$

Decision Tree Regression – Classification cost

- Entropy

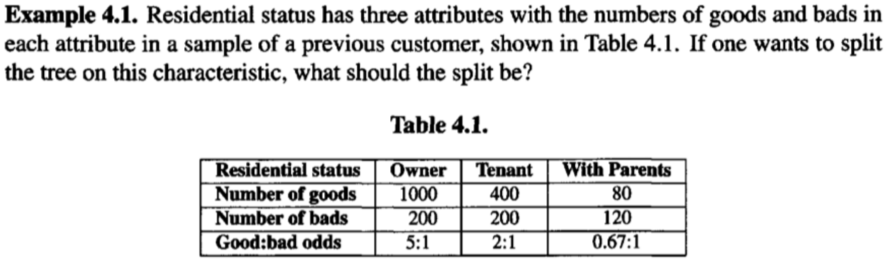
$$\mathbb{H}(\hat{\pi}) = -\sum_{c=1}^{C} \hat{\pi}_c log \hat{\pi}_c$$

- Gini index:

$$\sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c) = 1 - \sum_{c=1}^{C} \hat{\pi}_c^2$$

Decision Trees - Classification examples

Figure: Credit scoring using residential status. (Source: TEC[5])



Example 4.1. Residential status has three attributes with the numbers of goods and bads in each attribute in a sample of a previous customer, shown in Table 4.1. If one wants to split the tree on this characteristic, what should the split be?

Table 4.1.

Residential status	Owner	Tenant	With Parents
Number of goods	1000	400	80
Number of bads	200	200	120
Good:bad odds	5:1	2:1	0.67:1

Decision Trees - Using entropy for classification cost

$l =$ parent, $\quad r =$ owner + tenant:

$$i(v) = -\left(\frac{520}{2000}\right)\ln\left(\frac{520}{2000}\right) - \left(\frac{1480}{2000}\right)\ln\left(\frac{1480}{2000}\right) = 0.573,$$

$$p(l) = \frac{200}{2000} = 0.1, \quad i(l) = -\left(\frac{80}{200}\right)\ln\left(\frac{80}{200}\right) - \left(\frac{120}{200}\right)\ln\left(\frac{120}{200}\right) = 0.673,$$

$$p(r) = \frac{1800}{2000} = 0.9, \quad i(r) = -\left(\frac{400}{1800}\right)\ln\left(\frac{400}{1800}\right) - \left(\frac{1400}{1800}\right)\ln\left(\frac{1400}{1800}\right) = 0.530,$$

$$I = 0.573 - 0.1(0.673) - 0.9(0.530) = 0.0287;$$

Decision Trees - Using entropy for classification cost

l = parent + tenant, $\quad r$ = owner:

$$i(v) = -\left(\frac{520}{2000}\right)\ln\left(\frac{520}{2000}\right) - \left(\frac{1480}{2000}\right)\ln\left(\frac{1480}{2000}\right) = 0.573,$$

$$p(l) = \frac{800}{2000} = 0.4, \quad i(l) = -\left(\frac{320}{800}\right)\ln\left(\frac{320}{800}\right) - \left(\frac{480}{800}\right)\ln\left(\frac{480}{800}\right) = 0.673,$$

$$p(r) = \frac{1200}{2000} = 0.6, \quad i(r) = -\left(\frac{200}{1200}\right)\ln\left(\frac{200}{1200}\right) - \left(\frac{1000}{1200}\right)\ln\left(\frac{1000}{1200}\right) = 0.451,$$

$$I = 0.573 - 0.4(0.673) - 0.6(0.451) = 0.0332.$$

Decision Trees - Using gini index for classification cost

l = parent, r = owner + tenant:

$$i(v) = \left(\frac{1480}{2000}\right)\left(\frac{520}{2000}\right) = 0.1924,$$

$$p(l) = \frac{200}{2000} = 0.1, \quad i(l) = \left(\frac{80}{200}\right)\left(\frac{120}{200}\right) = 0.24,$$

$$p(r) = \frac{1800}{2000} = 0.9, \quad i(r) = \left(\frac{400}{1800}\right)\left(\frac{1400}{1800}\right) = 0.1728,$$

$$I = 0.1924 - 0.1(0.24) - 0.9(0.1728) = 0.01288;$$

Decision Trees - Using gini index for classification cost

$l =$ parent $+$ tenant, $\quad r =$ owner:

$$i(v) = \left(\frac{520}{2000}\right)\left(\frac{1480}{2000}\right) = 0.1924,$$

$$p(l) = \frac{800}{2000} = 0.4, \quad i(l) = \left(\frac{320}{800}\right)\left(\frac{480}{800}\right) = 0.24,$$

$$p(r) = \frac{1200}{2000} = 0.6, \quad i(r) = \left(\frac{200}{1200}\right)\left(\frac{1000}{1200}\right) = 0.1389,$$

$$I = 0.1924 - 0.4(0.24) - 0.6(0.1389) = 0.01306.$$

Decision Tree Classification

```
# Splitting the dataset
from sklearn.model_selection\
 import train_test_split
X_train, X_test, y_train, y_test\
 = train_test_split(X, y, test_size = 0.25,\
  random_state = 0)

# Scaling features
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Decision Tree Classification

```python
# Fitting Decision Tree Classifier to the test set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(
        criterion = 'entropy',
         random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the test set
y_pred = classifier.predict(X_test)
```

Decision Tree Classification – Some parameters

- **criterion** : string, optional (default="gini") The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
- **max_depth** : int or None, optional (default=None) The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split** : int, float, optional (default=2) The minimum number of samples required to split an internal node:
  If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and $ceil(min\_samples\_split * n\_samples)$ are the minimum number of samples for each split.

Decision Tree Classification – Some parameters

- **min_samples_leaf** : int, float, optional (default=1) The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
  If int, then consider min_samples_leaf as the minimum number. If float, then min_samples_leaf is a fraction and $ceil(min_samples_leaf * n_samples)$are the minimum number of samples for each node. Changed in version 0.18: Added float values for fractions.
- **min_weight_fraction_leaf** : float, optional (default=0.) The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features** : int, float, string or None, optional (default=None) The number of features to consider when looking for the best split:

Contents

Bagging

- We can reduce the variance (without increasing the bias) of an estimate by averaging together multiple estimates.
- While the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated (Wikipedia).
- For example, we can train N different trees on different subsets of the data, chosen randomly with replacement, and then compute the ensemble

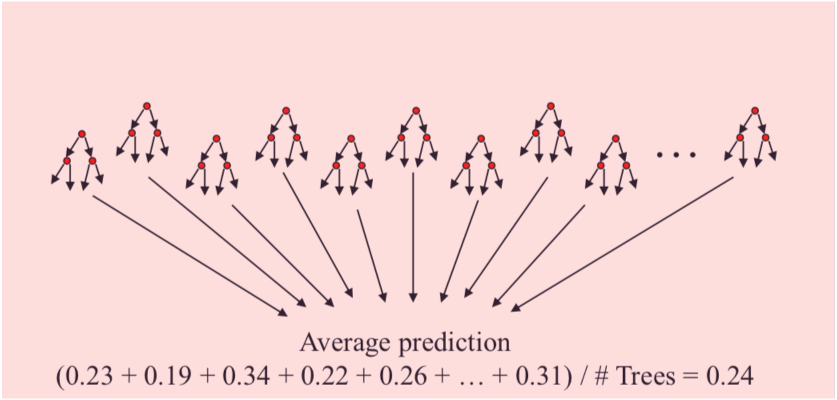
$$f(\mathbf{x}) = \sum_{n=1}^{N} \frac{1}{N} f_n(\mathbf{x})$$

  where f_m is the m'th tree.
- This technique is known as **bagging**, which stands for "bootstrap aggregating".

Bagging

- Draw 100 bootstrap samples of data
- Train a tree on each sample (100 trees)
- Average prediction of trees on out-of-bag samples

Figure: An example of bagging. (Source: UIUC CS446 Lecture notes)



. . .
Average prediction
(0.23 + 0.19 + 0.34 + 0.22 + 0.26 + … + 0.31) / # Trees = 0.24

Random Forest

- Unfortunately, simply re-running the same learning algorithm on different subsets of the data can result in highly correlated predictors, which limits the amount of variance reduction that is possible.
- The technique known as random forests tries to decorrelate the base learners by learning trees based on a randomly chosen subset of features, as well as a randomly chosen subset of data cases.
- This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the trees, causing them to become correlated (Wikipedia).
- Such models often have very good predictive accuracy, and have been widely used in many applications

Random Forest

- Draw N bootstrap samples of data
- Draw sample of available attributes at each split
- Train trees on each sample/attribute set (N trees)
- Average prediction of trees on out-of-bag samples

Random Forest Classification

```python
# Fitting Random Forest Classifier
# to the training set
from sklearn.ensemble import\
 RandomForestClassifier
classifier = RandomForestClassifier(
    n_estimators = 10,
    criterion = 'entropy',
    random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the test set
y_pred = classifier.predict(X_test)
```

Random Forest Classification – Some parameters

- **n_estimators** : integer, optional (default=10) The number of trees in the forest.
- **criterion** : string, optional (default="gini") The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.
- **max_depth** : integer or None, optional (default=None) The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split** : int, float, optional (default=2) The minimum number of samples required to split an internal node: If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and $ceil(min\_samples\_split * n\_samples)$ are the minimum number of samples for each split.

Contents

Boosting

- An **adaptive basis-function model** (ABM) is a model of the form

$$f(x) = w_0 + \sum_{m=1}^{M} w_m \phi_m(x)$$

  where $\phi_m(x)$ is the m's basis function, which is learned from data
- In **boosting**, ϕ_m are generated by an algorithm called a **weak learner** or a **base learner**.
- This weak learner can be any classification or regression algorithm, but it is common to use a CART model.

Boosting

- Initialization:
  - Weigh all training samples equally
- Iteration Step:
  - Train model on (weighted) train set
  - Compute error of model on train set
  - Increase weights on training cases model gets wrong
- Typically requires 100's to 1000's of iterations
- Return final model:
  - Carefully weighted prediction of each model

XGBoost Classification

```python
# Fitting XGBoost to the training set
from xgboost import XGBClassifier
classifier = XGBClassifier()
classifier.fit(X_train, y_train)

# Predicting the test set
y_pred = classifier.predict(X_test)
```

XGBoost Classification – Some parameters

- **loss** : 'deviance', 'exponential', optional (default='deviance') loss function to be optimized. 'deviance' refers to deviance (= logistic regression) for classification with probabilistic outputs. For loss 'exponential' gradient boosting recovers the AdaBoost algorithm.
- **learning_rate** : float, optional (default=0.1) learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.
- **n_estimators** : int (default=100) The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
- **subsample** : float, optional (default=1.0) The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. subsample interacts with the parameter n_estimators. Choosing subsample ¡ 1.0 leads to a reduction of variance and an increase in bias.

Contents

References

[1] K. P. Murpy – Machine Learning – A Probabilistic Perspective, MIT Press, 2012.
[2] UIUC CS 446 Machine Learning
[3] Udemy's Machine Learning,
`https://www.udemy.com/machinelearning/`
[4] scikit-learn website – `https://scikit-learn.org`
[5] L. C. Thomas, D. B. Edelman, J. N. Crook Credit Scoring and Its Applications, Second Edition, 2017