Adaboost

Frank Burkholder

Objectives

After this lecture you will be able to:

- Review: compare and contrast boosted trees with random forests
- Describe the Adaboost algorithm
- List some characteristics of a state-of-the-art boosting algorithm: XGBoost
- Explain when to use a partial dependency plot and be able to construct and interpret one.

Review (Breakout)

With a partner do Part 1 off the Adaboost Breakout

Review

TABLE 10.1. *Some characteristics of different learning methods. Key: ▲= good, ◆=fair, and ▼=poor.*

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Hastie et. al., Elements of Statistical Learning 2nd ed., 2008

7. Conclusions

The field has made substantial progress in the last decade. Learning methods such as boosting, random forests, bagging, and SVMs achieve excellent performance that would have been difficult to obtain just 15 years ago. Of the earlier learning methods, feedforward neural nets have the best performance and are competitive with some of the newer methods, particularly if models will not be calibrated after training.

Caruana et. al., "An Empirical Comparison of Supervised Learning Algorithms", Proceedings of 23rd Intnl. Conf. on Machine Learning, 2006.

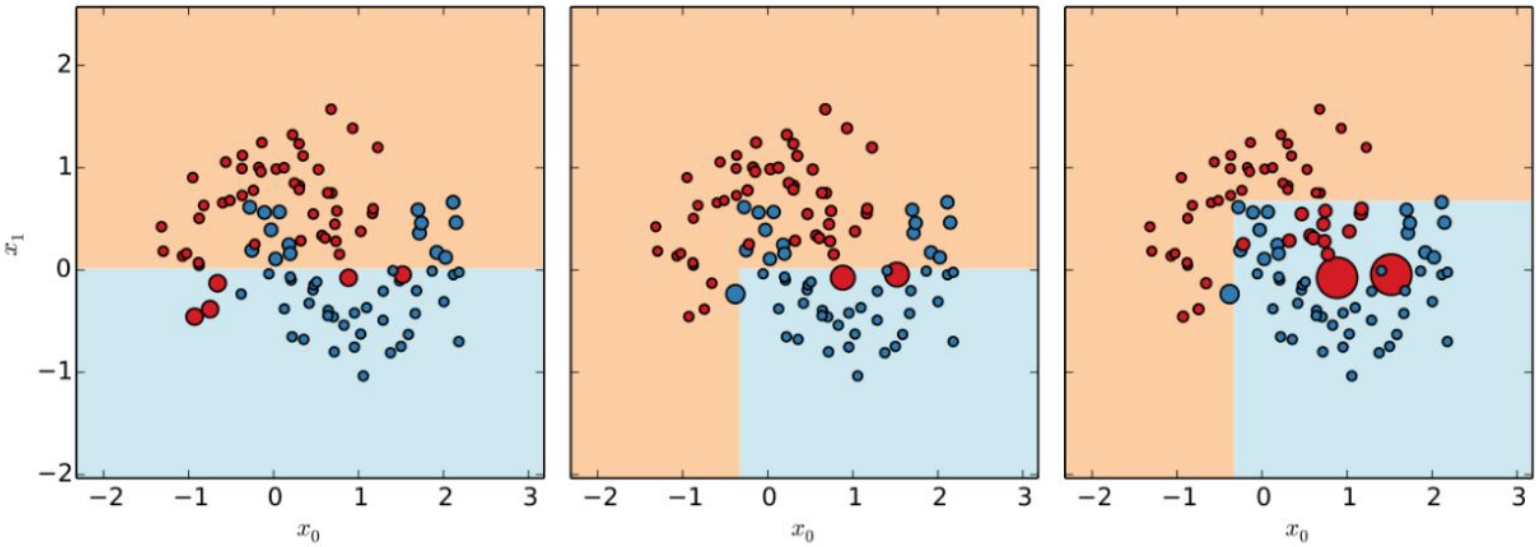
"Adaboost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier."

- *Wikipedia Adaboost entry, 2016*

Boosting - general idea

- In Boosting, an ensemble of trees is grown *sequentially*, based on information learned in the previous tree.
- Specifically, each new tree (after the first) is fit on some **modified version of the training set** based on results from the previous tree.
- In **Adaboost**, the datapoints associated with the largest residuals are *weighted the most* in the new training set. We'll talk about Adaboost this afternoon.
- In **Gradient Boosting**, the new training set **is the residuals**. (Makes "new" training sets)
- Can think of each boosted tree as a weak "rule of thumb," where the final prediction is a combination of these increasingly specific rules-of-thumb to yield good prediction accuracy in the end.
- We want our learners to be weak learners: high bias, low variance. By sequential addition of models the bias is decreased to get a low bias, low variance model.

(Discrete) Adaboost - Conceptual diagram

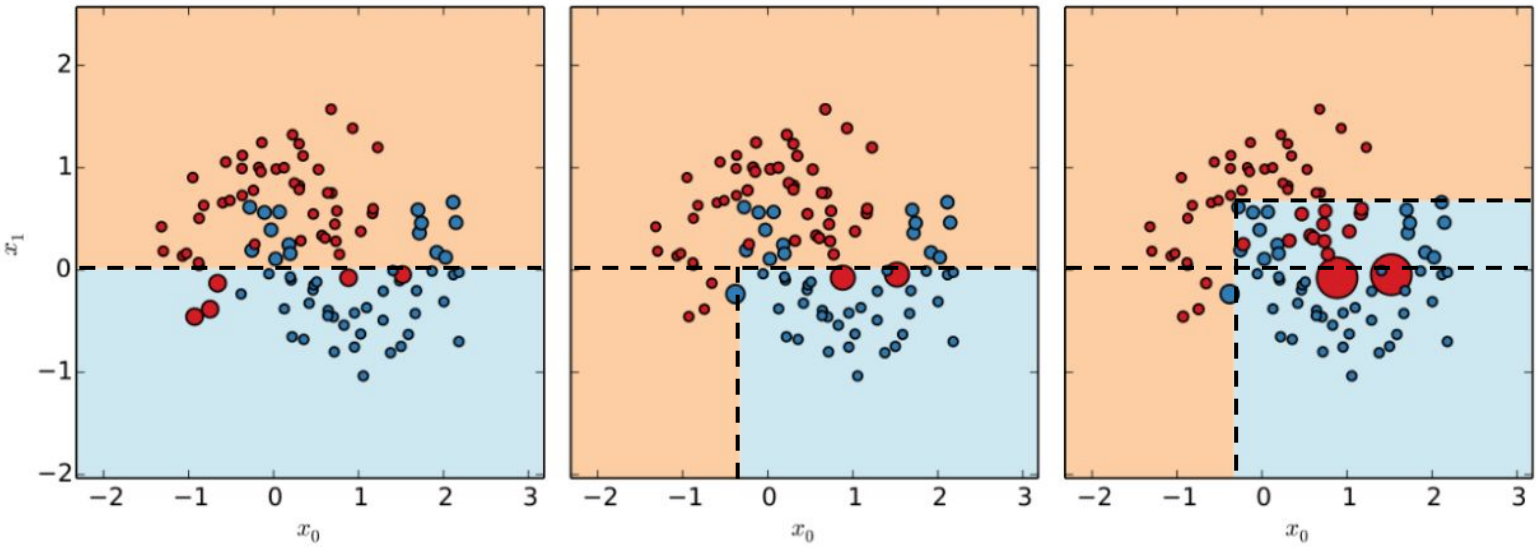


Tree #1: Original classification and assignment of weights for next tree.

Tree #2: When added to Tree #1 better classifies some of the data. Assigns weights for Tree #3.

Tree #3: And so on...

(Discrete) Adaboost - Conceptual diagram



Split #1 Split #2 Split #3?

Thought experiments:

1) Is the split on the far right #3? Why or why not?
2) After the split on the far right, what's the next split?

The Discrete Adaboost Algorithm

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Hastie, ESL, 2008

The Discrete Adaboost Algorithm

Algorithm 10.1 *AdaBoost.M1.*

Weight all the i = 1, N datapoints equally

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.

2. For $m = 1$ to M: *For each tree **m** from the first to the last, **M***

   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.

   (b) Compute

*Error rate for tree **m**, 0 - 1 (No errors = 0)*

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$. *Tree **m** weighting factor*

   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.

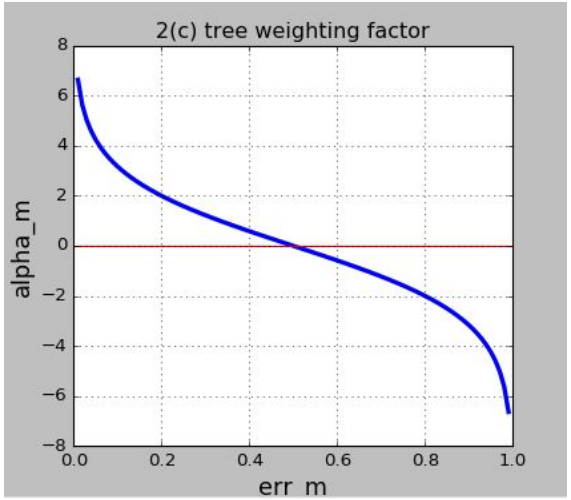
Update training data weights

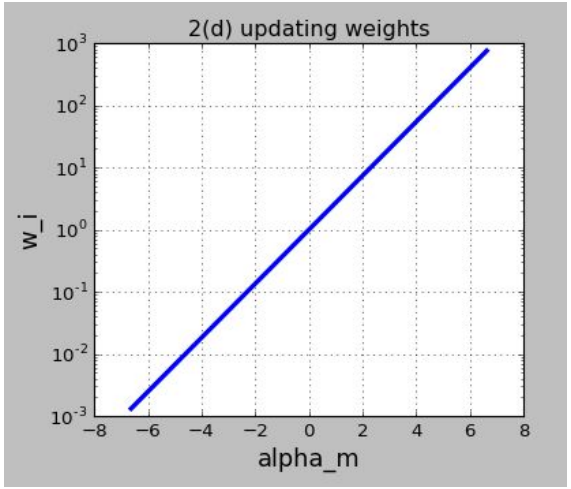
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Final model G(x) is weighted sum of 1..M models

Hastie, ESL, 2008

By the way in this algorithm $G(x_i)$ is either -1 or 1!





Adaboost tuning parameters (hyperparameters)

class sklearn.ensemble. **AdaBoostRegressor** (*base_estimator=None, n_estimators=50, learning_rate=1.0, loss='linear', random_state=None*) ¶

class sklearn.tree. **DecisionTreeRegressor** (*criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False*) [source]

class sklearn.ensemble. **AdaBoostClassifier** (*base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None*) ¶

class sklearn.tree. **DecisionTreeClassifier** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*) [source]

Breakout

With a partner do Part 2 off the Adaboost Breakout

Public Service Advisory: XGBoost

XGBoost (short for extreme gradient boosting) is gradient boosting with some niceties built in that make it much faster and more efficient than standard gradient boosting.

The primary change from standard gradient boosting is that column values are binned into percentiles when performing splitting.

For some j^{th} column, split all observations up into percentiles based off their values for that j^{th} column. Then, only consider some average value (mean, median, etc.) across each percentile for splitting (this narrows the search space quite a bit).

Other changes that XGBoost includes are:

- Handles sparsity in the data (e.g. missing values)
- Handles mixed data types (e.g. categoricals, continuous)
- Allows for out-of-core computation
- Builds in smart memory management

XGBoost - install and use in Python

It can be installed using pip*, and it does have an sklearn interface (e.g. fit, predict methods, etc.)

```
import xgboost as xgb
gbm = xgb.XGBClassifier(max_depth=3, n_estimators=300,
                        learning_rate=0.05)
gbm.fit(train_X, train_y)
predictions = gbm.predict(test_X)
```

* Very difficult installation for Macs, please see the installation guide provided in the afternoon folder. Recommend that you install it on your own time. You will want it for future case studies and your capstone project!