Gradient Boosted Trees

A high-performing “out-of the box” model for structured data

Anders Poirel

January 14, 2020

DSS@UCSC

Additive Models

- Suppose we have models $f_1, \ldots, f_N$.
- additive model: model of the form $\hat{f}(x) = \sum_{i=1}^{n} \hat{f}_i(x)$
- Additive models are a type of ensemble models, models that combine several simpler models for increased performance.

Motivation of Boosting (1)

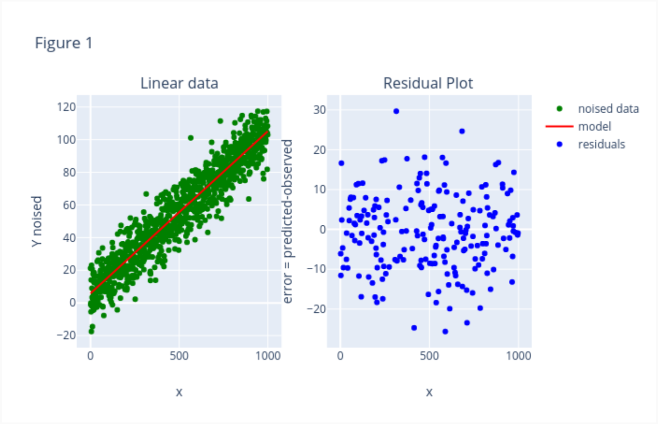
- weak model: model that performs slightly better than random chance
- Idea of boosting: combine many weak models to build a higher-performing one

Motivation of Boosting (2)

- To improve performance of a simple model, build a sequence of simple models that each fit to the error (residuals) of the previous ones
- Make this even more efficient by using pseudo-residuals that give the direction in which the model can be most improved the most, by taking the gradient of the loss with respect to the previous model

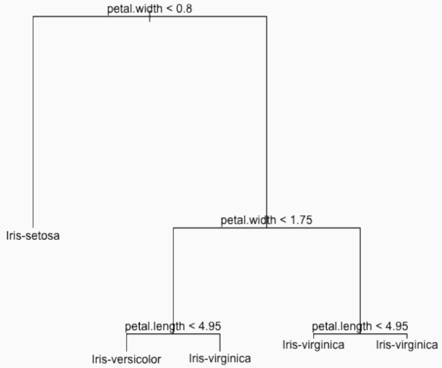
Motivation of Boosting (3)

What are residuals?



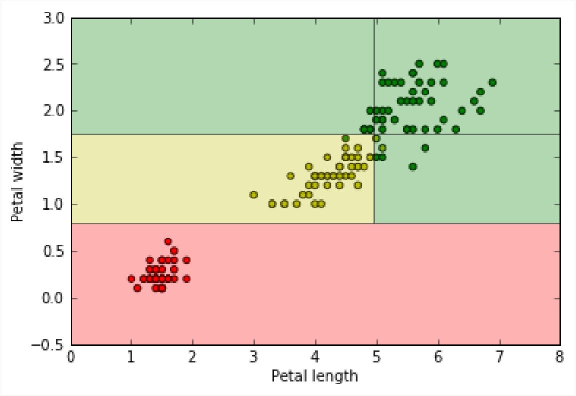
Decision Trees (1)

Decision trees make binary splits, partitioning the feature space into p-dimensional boxes, where the predicted labels are the mean value (or majority category for classification) of instances in that box.



Decision Trees (2)

Decision trees make binary splits, partitioning the feature space into p-dimensional boxes, where the predicted labels are the mean value (or majority category for classification) of instances in that box.



Why Boost Decision Trees ?

- Fast to construct with low tree depth
- Easily mix categorical and numerical variables
- Automatically perform feature selection so resistant to overfitting via too many predictors
- Transformation-invariant

Gradient Tree Boosting Algorithm

```
1. Initialize $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1, \ldots, M$:
    (a) For $i = 1, \ldots N$ compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$
    (b) Fit regression tree to target $r_{im}$ giving terminal regions
        $R_{jm}, j = 1, \ldots, J_m$
    (c) For $j = 1, \ldots, J_m$ compute
        $\gamma_{jm} = \text{argmin}_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{gm} \mathbb{1}(x \in R_{jm})$
3. Output $\hat{f}(x) = f_M(x)$
```

eXtreme Gradient Boosting (1)

- `xgboost` is a framework implementing gradient boosted trees with arbitray loss functions with frontends in Python, R and others
- `lightgbm` and `catboost` are other implementations of the algorithm worth looking into

eXtreme Gradient Boosting (2)

A minimal example with parameters that perform well in practice:

```
import numpy as np
import xgboost as xgb

X_train = np.load('X_train.npy')
y_train = np.load('y_train.npy')

model = xgb.XGBRegressor(eta=0.1, num_estimators=
                          100, max_tree_depth=
                          2tree_method=hist,
                          n_jobs=-1)
model.fit(X_train, y_train)
```

Tuning XGBoost (1)

Important parameters:

- `learning_rate` (alias `gamma`): step size shrinkage used in updates (range: $(0, 1)$)
- `max_depth`: maximum depth of a tree
- `min_split_loss` (alias `gamma`): minimum loss reduction required to make a partition on a leaf node
- `reg_lambda` or `reg_alpha`: L2 (resp. L1) penalty term on weights
- `subsample`: proportion of training data to use at each training step

Tuning XGBoost (2)

Grid search example (assume same libraries as before)

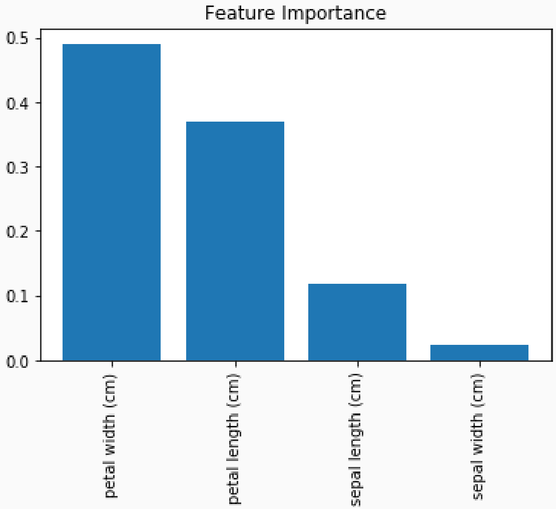
```
from sklearn.model_selection import
                                    GridSearchCV
params = { ’learning_rate’ : [0.001, 0.01, 0.1
                                    , 1],
           ’reg_lambda’: [0, 1, 10],
           ’min_split_loss’: [0, 1, 10],
           ’max_depth’: [n for n in range(1,8)
                                               ],
           ’n_estimators’: [25, 50, 100, 200]
}
model = GridSearchCV(model = XGBRegressor
                    (tree_methond = hist),
                    scoring =
                    ’neg_mean_absolute_error’,
                    cv = 10, njobs = -1)
```

Interpreting XGBoost (1)

- For a single tree T with $J-1$ internal nodes, importance of feature X_l: $\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \mathbb{1}(v(t) = l)$ where $\hat{i}_t^2$ is the absolute loss decrease brought by split on that node
- Generalize to ensembles of M trees with $\mathcal{I}_l^2(T) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_l^2(T_m)$

Interpreting XGBoost (2)

Example of a feature importance plot (we usually scale feature importances to be in (0,1) or (0, 100) ranges)



When You Shouldn’t Use Gradient Boosted Trees

- you are working with unstructured data (text, image, video): variations of neural networks perform far better
- you need interpretable models (law, medicine): GBTs are a “black-box” algorithm
- you don’t have a lot of data: GBTs will easily overfit
- you care about inference more than predictive power: rigorous statistical inference theory is an active research area for GBTs

References

- Xgboost Developer, *XGBoost Tutorials*, https://xgboost.readthedocs.io/en/latest/tutorials/index.html 2019
- T. Chen, *Introduction to Boosted Trees*, https://homes.cs.washington.edu/ tqchen/pdf/BoostedTree.pdf, University of Washington, 2014
- T. Hastie, R. Tibshirani, J. Friedman, Boosting and Additive Methods. *Elements of Statistical Learning*, Springer, 2011