Gradient Boosted Regression Trees

Peter Prettenhofer (@pprett)  
DataRobot

Gilles Louppe (@glouppe)  
Université de Liège, Belgium
Motivation
Motivation
Outline

1. Basics
2. Gradient Boosting
3. Gradient Boosting in scikit-learn
4. Case Study: California housing
About us

Peter

- @ppret
- Python & ML ~ 6 years
- sklearn dev since 2010

Gilles

- @glouppe
- PhD student (Liège, Belgium)
- sklearn dev since 2011
  *Chief tree hugger*
Outline

1. Basics

2. Gradient Boosting

3. Gradient Boosting in scikit-learn

4. Case Study: California housing
• Data comes as...
  • A set of examples \( \{(x_i, y_i)\mid 0 \leq i < n_{\text{samples}}\} \), with
  • Feature vector \( x \in \mathbb{R}^{n_{\text{features}}} \), and
  • Response \( y \in \mathbb{R} \) (regression) or \( y \in \{-1, 1\} \) (classification)

• Goal is to...
  • Find a function \( \hat{y} = f(x) \)
  • Such that error \( L(y, \hat{y}) \) on new (unseen) \( x \) is minimal
Classification and Regression Trees [Breiman et al, 1984]

```
MedInc <= 5.04
    MedInc <= 3.07
        AveRooms <= 4.31
            1.62
        AveOccup <= 2.37
            1.16
            2.79
            1.88
    MedInc <= 6.82
        AveOccup <= 2.74
            3.39
            2.56
        MedInc <= 7.82
            3.73
            4.57
```

`sklearn.tree.DecisionTreeClassifier | Regressor`
Function approximation with Regression Trees

- Deprecated
- Nowadays seldom used alone
- Ensembles: Random Forest, Bagging, or Boosting (see sklearn.ensemble)
Function approximation with Regression Trees

- Nowadays seldom used alone
- Ensembles: Random Forest, Bagging, or Boosting (see sklearn.ensemble)

Deprecated
Outline

1 Basics

2 Gradient Boosting

3 Gradient Boosting in scikit-learn

4 Case Study: California housing
Gradient Boosted Regression Trees

Advantages

- Heterogeneous data (features measured on different scale)
- Supports different loss functions (e.g. huber)
- Automatically detects (non-linear) feature interactions

Disadvantages

- Requires careful tuning
- Slow to train (but fast to predict)
- Cannot extrapolate
Boosting

AdaBoost [Y. Freund & R. Schapire, 1995]

- Ensemble: each member is an expert on the errors of its predecessor
- Iteratively re-weights training examples based on errors

```
sklearn.ensemble.AdaBoostClassifier|Regressor
```
Boosting

• AdaBoost [Y. Freund & R. Schapire, 1995]
  • Ensemble: each member is an expert on the errors of its predecessor
  • Iteratively re-weights training examples based on errors

Huge success

• Viola-Jones Face Detector (2001)

• Freund & Schapire won the Gödel prize 2003
Gradient Boosting [J. Friedman, 1999]

Statistical view on boosting

• ⇒ Generalization of boosting to arbitrary loss functions
Gradient Boosting [J. Friedman, 1999]

Statistical view on boosting

- Generalization of boosting to arbitrary loss functions

Residual fitting

```
- sklearn.ensemble.GradientBoostingClassifier
- sklearn.ensemble.GradientBoostingRegressor
```
Functional Gradient Descent

Least Squares Regression

- Squared loss: $L(y_i, f(x_i)) = (y_i - f(x_i))^2$
- The residual $\sim$ the (negative) gradient $\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}$
Functional Gradient Descent

Least Squares Regression

- Squared loss: $L(y_i, f(x_i)) = (y_i - f(x_i))^2$
- The residual $\sim$ the (negative) gradient $\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}$

Steepest Descent

- Regression trees approximate the (negative) gradient
- Each tree is a successive gradient descent step
Outline

1 Basics

2 Gradient Boosting

3 Gradient Boosting in scikit-learn

4 Case Study: California housing
GBRT in scikit-learn

How to use it

```python
>>> from sklearn.ensemble import GradientBoostingClassifier
>>> from sklearn.datasets import make_hastie_10_2
>>> X, y = make_hastie_10_2(n_samples=10000)
>>> est = GradientBoostingClassifier(n_estimators=200, max_depth=3)
>>> est.fit(X, y)
...

>>> # get predictions
>>> pred = est.predict(X)
>>> est.predict_proba(X)[0]  # class probabilities
array([ 0.67, 0.33])
```

Implementation

- Written in pure Python/Numpy (easy to extend).
- Builds on top of sklearn.tree.DecisionTreeRegressor (Cython).
- Custom node splitter that uses pre-sorting (better for shallow trees).
from sklearn.ensemble import GradientBoostingRegressor
est = GradientBoostingRegressor(n_estimators=2000, max_depth=1).fit(X, y)
for pred in est.staged_predict(X):
    plt.plot(X[:, 0], pred, color='r', alpha=0.1)
Model complexity & Overfitting

test_score = np.empty(len(est.estimators_))
for i, pred in enumerate(est.staged_predict(X_test)):
    test_score[i] = est.loss_(y_test, pred)
plt.plot(np.arange(n_estimators) + 1, test_score, label='Test')
plt.plot(np.arange(n_estimators) + 1, est.train_score_, label='Train')

GBRT provides a number of knobs to control overfitting:
- Tree structure
- Shrinkage
- Stochastic Gradient Boosting
Model complexity & Overfitting

test_score = np.empty(len(est.estimators_))
for i, pred in enumerate(est.staged_predict(X_test)):
    test_score[i] = est.loss_(y_test, pred)
plt.plot(np.arange(n_estimators) + 1, test_score, label='Test')
plt.plot(np.arange(n_estimators) + 1, est.train_score_, label='Train')

Regularization

GBRT provides a number of knobs to control overfitting

- Tree structure
- Shrinkage
- Stochastic Gradient Boosting
Regularization: Tree structure

- The `max_depth` of the trees controls the degree of features interactions
- Use `min_samples_leaf` to have a sufficient nr. of samples per leaf.
Regularization: Shrinkage

- Slow learning by shrinking tree predictions with $0 < \text{learning\_rate} \leq 1$
- Lower learning_rate requires higher n_estimators
Regularization: Stochastic Gradient Boosting

- Samples: random subset of the training set (subsample)
- Features: random subset of features (max_features)
- Improved accuracy – reduced runtime
Hyperparameter tuning

1. Set `n_estimators` as high as possible (eg. 3000)

2. Tune hyperparameters via grid search.

   ```python
   from sklearn.grid_search import GridSearchCV
   param_grid = {
       'learning_rate': [0.1, 0.05, 0.02, 0.01],
       'max_depth': [4, 6],
       'min_samples_leaf': [3, 5, 9, 17],
       'max_features': [1.0, 0.3, 0.1]}
   est = GradientBoostingRegressor(n_estimators=3000)
   gs_cv = GridSearchCV(est, param_grid).fit(X, y)
   # best hyperparameter setting
   gs_cv.best_params_
   ```

3. Finally, set `n_estimators` even higher and tune `learning_rate`. 
Outline

1 Basics

2 Gradient Boosting

3 Gradient Boosting in scikit-learn

4 Case Study: California housing
Case Study

California Housing dataset

- Predict $\log(medianHouseValue)$
- Block groups in 1990 census
- 20.640 groups with 8 features (median income, median age, lat, lon, ...)
- Evaluation: Mean absolute error on 80/20 split

Challenges

- Heterogeneous features
- Non-linear interactions
## Predictive accuracy & runtime

<table>
<thead>
<tr>
<th>Method</th>
<th>Train time [s]</th>
<th>Test time [ms]</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>-</td>
<td>-</td>
<td>0.4635</td>
</tr>
<tr>
<td>Ridge</td>
<td>0.006</td>
<td>0.11</td>
<td>0.2756</td>
</tr>
<tr>
<td>SVR</td>
<td>28.0</td>
<td>2000.00</td>
<td>0.1888</td>
</tr>
<tr>
<td>RF</td>
<td>26.3</td>
<td>605.00</td>
<td>0.1620</td>
</tr>
<tr>
<td>GBRT</td>
<td>192.0</td>
<td>439.00</td>
<td><strong>0.1438</strong></td>
</tr>
</tbody>
</table>

![Graph showing error vs n_estimators for both Test and Train data](image-url)
Model interpretation

Which features are important?

```python
>>> est.feature_importances_
array([ 0.01, 0.38, ...])
```
Model interpretation

What is the effect of a feature on the response?

```python
from sklearn.ensemble import partial_dependence import as pd

features = ['MedInc', 'AveOccum', 'HouseAge', 'AveRooms', ('AveOccum', 'HouseAge')]
fig, axs = pd.plot_partial_dependence(est, X_train, features, feature_names=names)
```

Partial dependence of house value on nonlocation features for the California housing dataset.
Model interpretation

Automatically detects spatial effects
Summary

- Flexible non-parametric classification and regression technique
- Applicable to a variety of problems
- Solid, battle-worn implementation in scikit-learn
Thanks! Questions?
Benchmarks

![Bar chart showing error, train time, and test time for different datasets.](chart.png)
Input layout

Use `dtype=np.float32` to avoid memory copies and fortran layout for slight runtime benefit.

```python
X = np.asfortranarray(X, dtype=np.float32)
```
Feature interactions
GBRT automatically detects feature interactions but often explicit interactions help.

Trees required to approximate $X_1 - X_2$: 10 (left), 1000 (right).
Categorical variables

Sklearn requires that categorical variables are encoded as numerics. Tree-based methods work well with ordinal encoding:

df = pd.DataFrame(data={'icao': ['CRJ2', 'A380', 'B737', 'B737']})
# ordinal encoding
df_enc = pd.DataFrame(data={'icao': np.unique(df.icao, return_inverse=True)[1]})
X = np.asfortranarray(df_enc.values, dtype=np.float32)