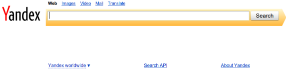
Gradient Boosted Regression Trees

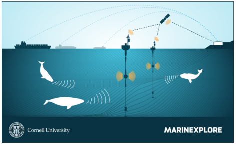
Peter Prettenhofer (@pprett)
DataRobot

Gilles Louppe (@glouppe)
Université de Liège, Belgium

Motivation



GEFCom2012
Wind Forecasting



CARVANA
SELL NO EVIL.

Motivation

NETFLIX
Movies For You
You really liked it...
Heritage Health Prize
Yandex
Web Images Video Mail Translate
Yandex worldwide
Search API
LEARNING TO RANK
March - May 2010
Cornell University
MARINEXPLORE
Expedia
CARVANA
SELL NO EVIL.

Outline

About us

Peter

- @pprett
- Python & ML $\sim$ 6 years
- sklearn dev since 2010

Gilles

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

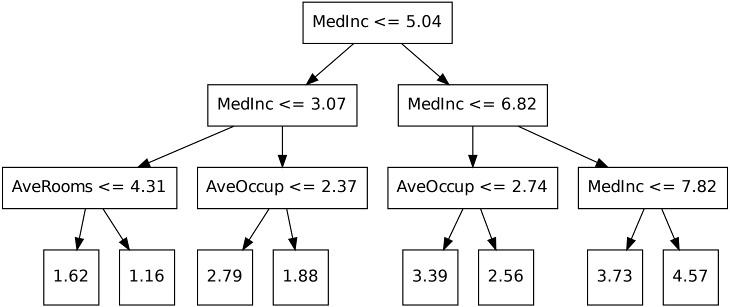
Outline

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

Machine Learning 101

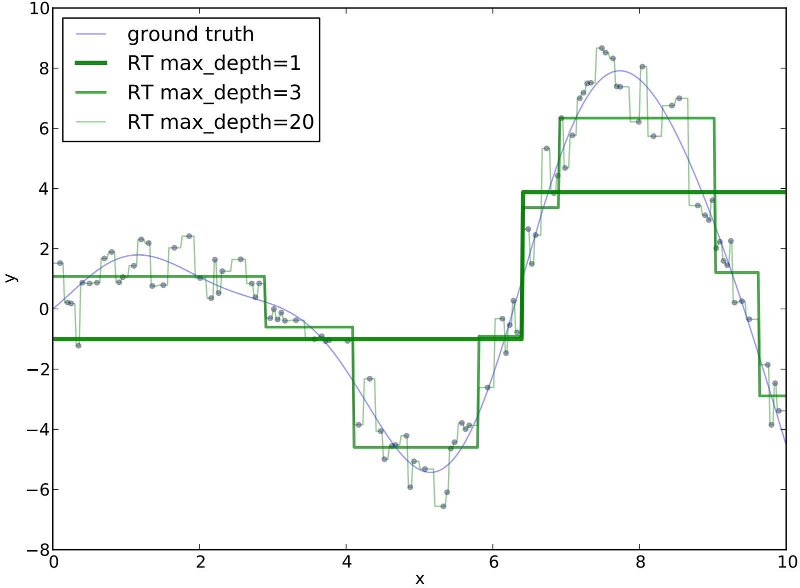
- Data comes as...
  - A set of examples $\{(\mathbf{x}_i, y_i) | 0 \leq i < \text{n\_samples}\}$, with
  - Feature vector $\mathbf{x} \in \mathbb{R}^{\text{n\_features}}$, and
  - Response $y \in \mathbb{R}$ (regression) or $y \in \{-1, 1\}$ (classification)
- Goal is to...
  - Find a function $\hat{y} = f(\mathbf{x})$
  - Such that error $L(y, \hat{y})$ on new (unseen) $\mathbf{x}$ is minimal

Classification and Regression Trees [Breiman et al, 1984]

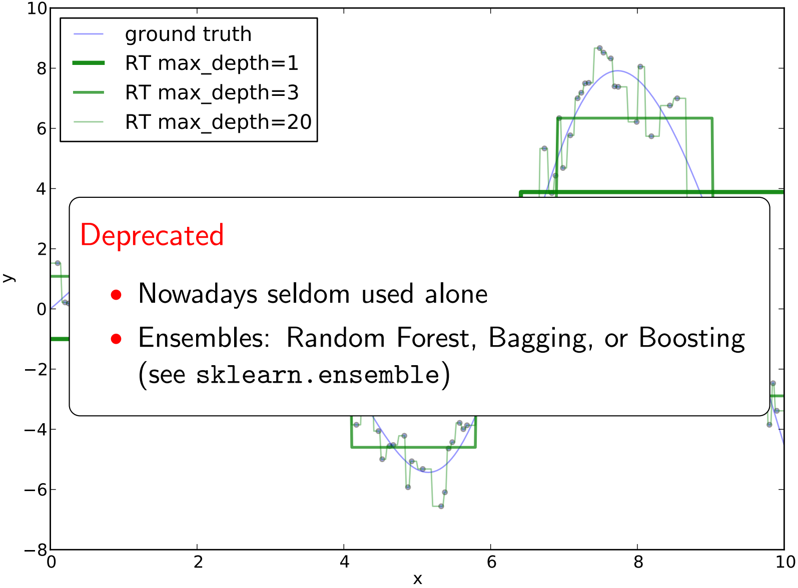


`sklearn.tree.DecisionTreeClassifier|Regressor`

Function approximation with Regression Trees



Function approximation with Regression Trees



Deprecated

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

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

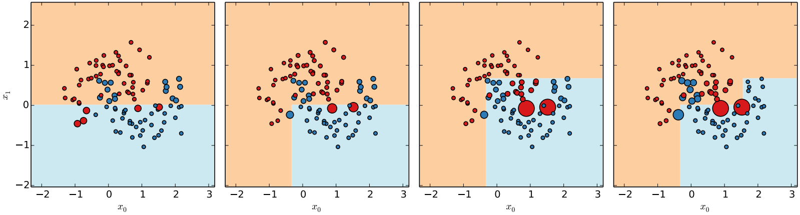
Disadvantages

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

Boosting

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

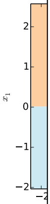
- Ensemble: each member is an expert on the errors of its predecessor
- Iteratively re-weights training examples based on errors



`sklearn.ensemble.AdaBoostClassifier|Regressor`

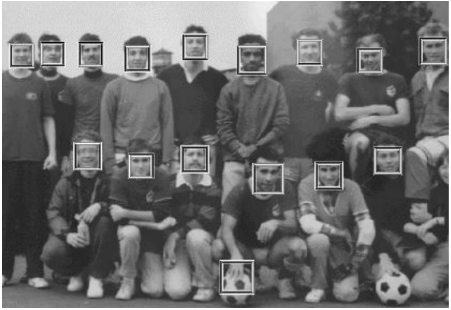
Boosting

Adal

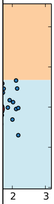


Huge success

- Viola-Jones Face Detector (2001)



- Freund & Schapire won the Gödel prize 2003



`sklearn.ensemble.AdaBoostClassifier|Regressor`

Gradient Boosting [J. Friedman, 1999]

Statistical view on boosting

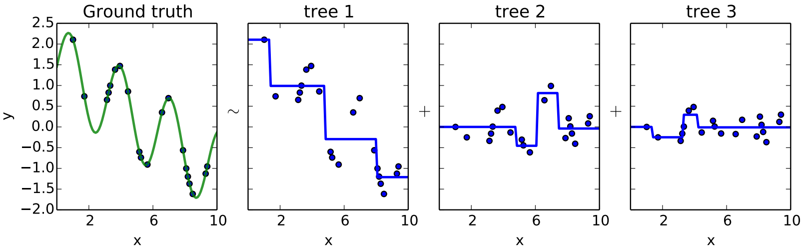
- $\Rightarrow$ Generalization of boosting to arbitrary loss functions

Gradient Boosting [J. Friedman, 1999]

Statistical view on boosting

- $\Rightarrow$ Generalization of boosting to arbitrary loss functions

Residual fitting



sklearn.ensemble.GradientBoostingClassifier|Regressor

Functional Gradient Descent

Least Squares Regression

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

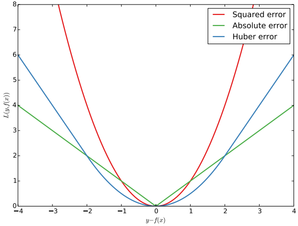
Functional Gradient Descent

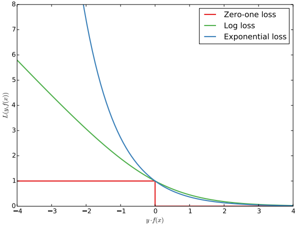
Least Squares Regression

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

Steepest Descent

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





Outline

GBRT in scikit-learn

How to use it

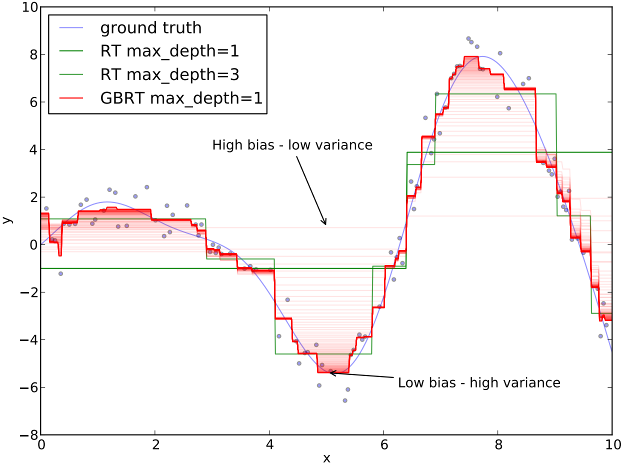
```
>>> from sklearn.ensemble import GradientBoostingClassifier
>>> from sklearn.datasets import make_hastie_10_2
>>> X, y = make_hastie_10_2(n_samples=10000)
>>> est = GradientBoostingClassifier(n_estimators=200, max_depth=3)
>>> est.fit(X, y)
...
>>> # get predictions
>>> pred = est.predict(X)
>>>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

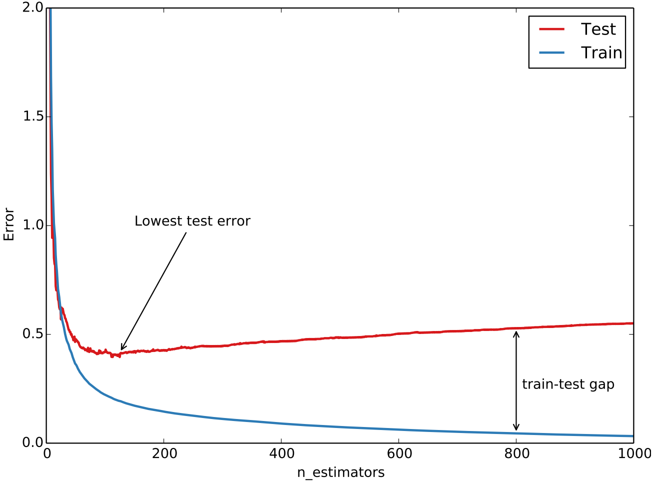
Example

```
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)
```



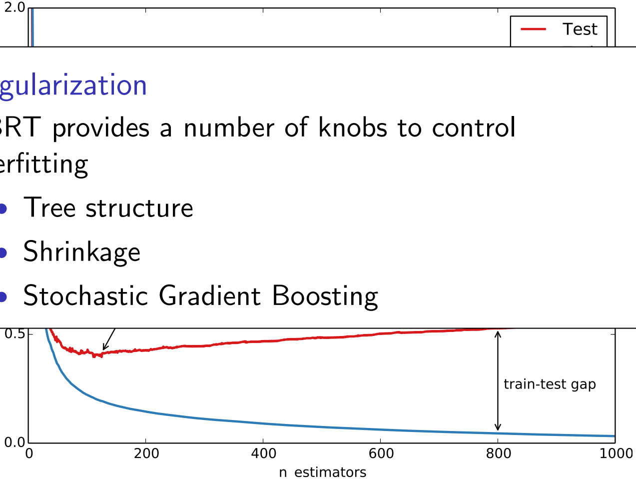
Model complexity & Overfitting

```
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')
```



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')
```



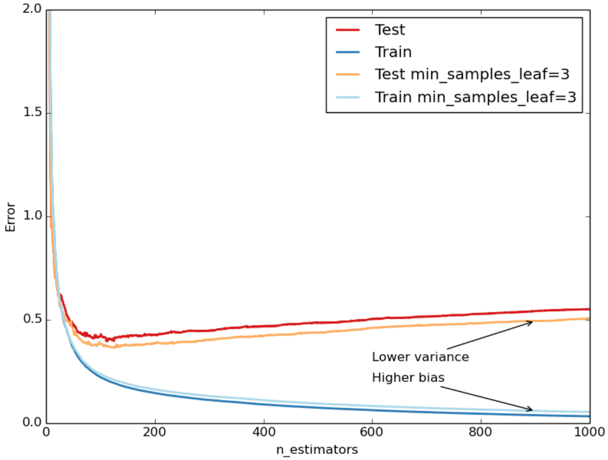
Regularization

GBRT provides a number of knobs to control overfitting

- Tree structure
- Shrinkage
- Stochastic Gradient Boosting

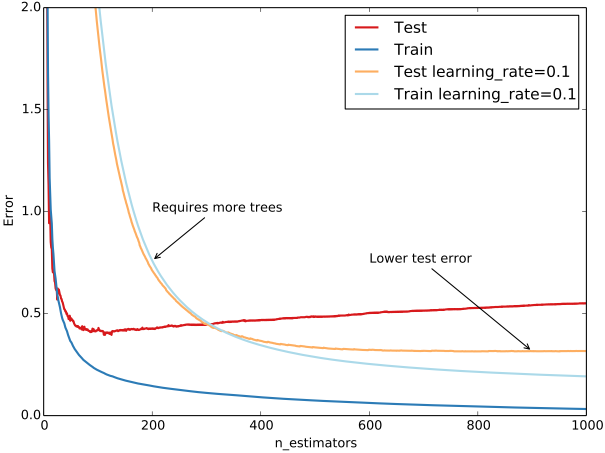
Regularization: Tree structure

- 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.



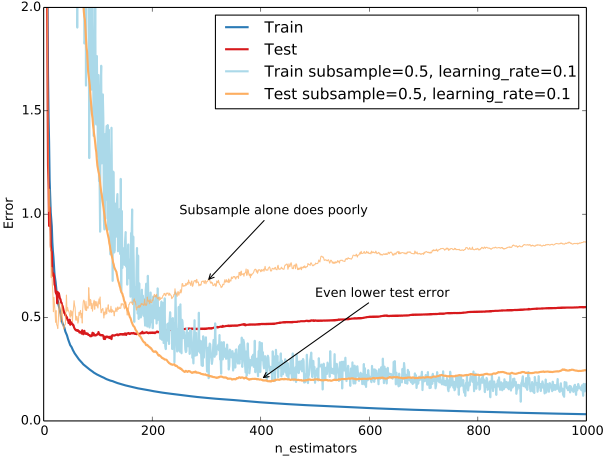
Regularization: Shrinkage

- Slow learning by shrinking tree predictions with $0 < \text{learning\_rate} <= 1$
- 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.

   ```
   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

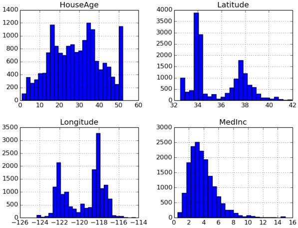
Case Study

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

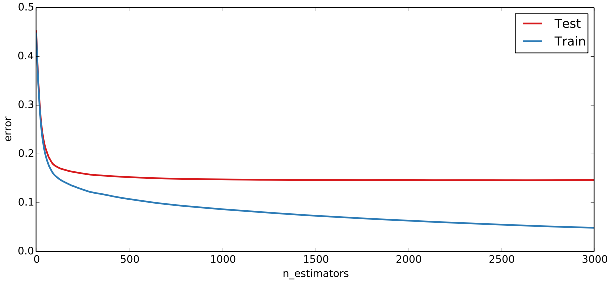
Challenges

- Heterogeneous features
- Non-linear interactions



Predictive accuracy & runtime

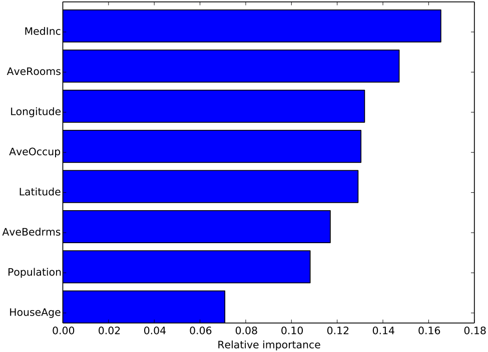
	Train time [s]	Test time [ms]	MAE
Mean	-	-	0.4635
Ridge	0.006	0.11	0.2756
SVR	28.0	2000.00	0.1888
RF	26.3	605.00	0.1620
GBRT	192.0	439.00	**0.1438**



Model interpretation

Which features are important?

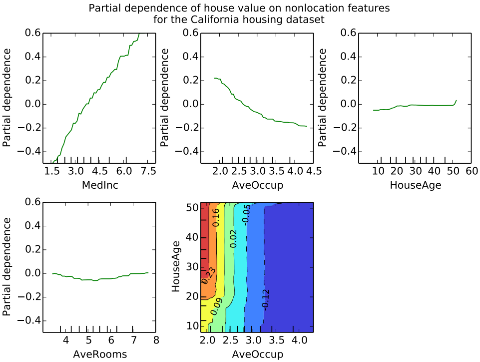
```
>>> est.feature_importances_
array([ 0.01,  0.38, ...])
```



Model interpretation

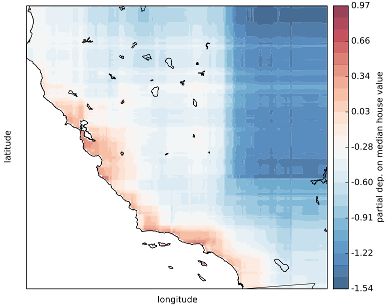
What is the effect of a feature on the response?

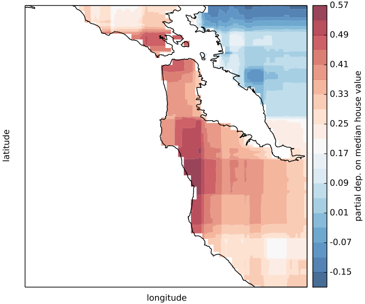
```
from sklearn.ensemble import partial_dependence import as pd

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



Model interpretation

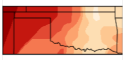
Automatically detects spatial effects





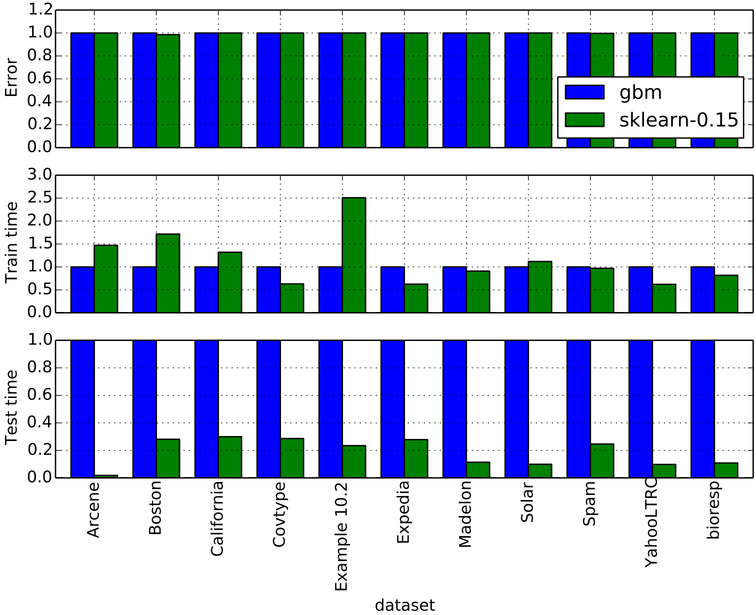
Summary

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



Thanks! Questions?

Benchmarks



Error
Train time
Test time
gbm
sklearn-0.15
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.5
1.5
2.0
2.5
3.0
Arcene
Boston
California
Covtype
Example 10.2
Expedia
Madelon
Solar
Spam
YahooLTRC
bioresp
dataset

Tipps & Tricks 1

Input layout

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

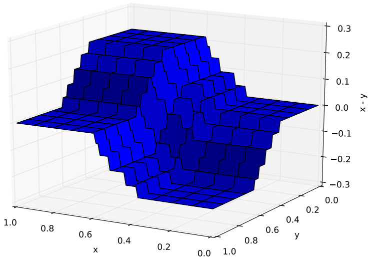
```
X = np.asfortranarray(X, dtype=np.float32)
```

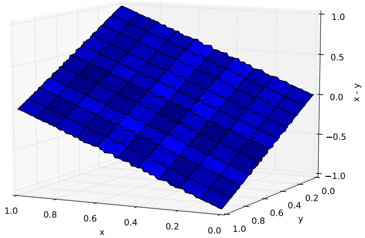
Tipps & Tricks 2

Feature interactions

GBRT automatically detects feature interactions but often explicit interactions help.

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





Tipps & Tricks 3

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)
```