

# Supervised Learning: Non-parametric methods

Gilles Louppe (@glouppe)

October 17, 2016

# Outline

- 1 Decision trees
- 2 Forests of randomized trees
- 3 Boosting
- 4 Interpreting tree-based models
- 5 Gaussian Processes
- 6 Neural networks

# Running example

From **physicochemical properties** (alcohol, acidity, sulphates, ...),

learn a **model**

to predict **wine taste preferences**.



# Supervised learning

- Data comes as a finite learning set  $\mathcal{L} = (\mathbf{X}, \mathbf{y})$  where
  - **Input samples** are given as an array of shape (n\_samples, n\_features)

E.g., feature values for wine physicochemical properties:

# fixed acidity, volatile acidity, ...

```
X = [[ 7.4    0.    ...  0.56  9.4    0. ]  
      [ 7.8    0.    ...  0.68  9.8    0. ]  
      ...  
      [ 7.8    0.04  ...  0.65  9.8    0. ]]
```

- **Output values** are given as an array of shape (n\_samples,)

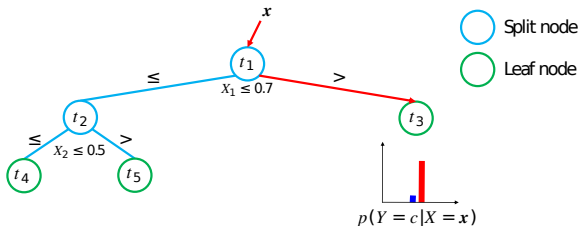
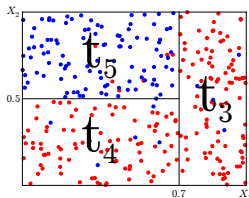
E.g., wine taste preferences (from 0 to 10):

```
y = [5 5 5 ... 6 7 6]
```

- The goal is to build an estimator  $f : \mathcal{X} \mapsto \mathcal{Y}$  minimizing

$$Err(f) = \mathbb{E}_{\mathbf{X}, \mathbf{Y}} \{L(\mathbf{Y}, f(\mathbf{X}))\}.$$

# Decision trees (Breiman et al., 1984)



**function** BUILDDECISIONTREE( $\mathcal{L}$ )

    Create node  $t$

**if** the stopping criterion is met for  $t$  **then**

        Assign a model to  $\hat{y}_t$

**else**

        Find the split on  $\mathcal{L}$  that maximizes impurity decrease

$$s^* = \arg \max_s i(t) - p_L i(t_L^s) - p_R i(t_R^s)$$

        Partition  $\mathcal{L}$  into  $\mathcal{L}_{t_L} \cup \mathcal{L}_{t_R}$  according to  $s^*$

$t_L = \text{BUILDDECISIONTREE}(\mathcal{L}_{t_L})$

$t_R = \text{BUILDDECISIONTREE}(\mathcal{L}_{t_R})$

**end if**

**return**  $t$

**end function**

(Demo)

# Composability of decision trees

Decision trees can be used to solve several machine learning tasks by swapping the impurity and leaf model functions:

## 0-1 loss (classification)

$$\hat{y}_t = \arg \max_{c \in \mathcal{Y}} p(c|t), \quad i(t) = \text{entropy}(t) \text{ or } i(t) = \text{gini}(t)$$

## Mean squared error (regression)

$$\hat{y}_t = \text{mean}(y|t), \quad i(t) = \frac{1}{N_t} \sum_{\mathbf{x}, y \in \mathcal{L}_t} (y - \hat{y}_t)^2$$

## Least absolute deviance (regression)

$$\hat{y}_t = \text{median}(y|t), \quad i(t) = \frac{1}{N_t} \sum_{\mathbf{x}, y \in \mathcal{L}_t} |y - \hat{y}_t|$$

## Density estimation

$$\hat{y}_t = \mathcal{N}(\mu_t, \Sigma_t), \quad i(t) = \text{differential entropy}(t)$$

# Sample weights

Sample weights can be accounted for by adapting the impurity and leaf model functions.

## Weighted mean squared error

$$\hat{y}_t = \frac{1}{\sum_w w} \sum_{\mathbf{x}, y, w \in \mathcal{L}_t} w y$$
$$i(t) = \frac{1}{\sum_w w} \sum_{\mathbf{x}, y, w \in \mathcal{L}_t} w (y - \hat{y}_t)^2$$

Weights are assumed to be **non-negative** since these quantities may otherwise be undefined. (E.g., what if  $\sum_w w < 0$ ?)



## sklearn.tree

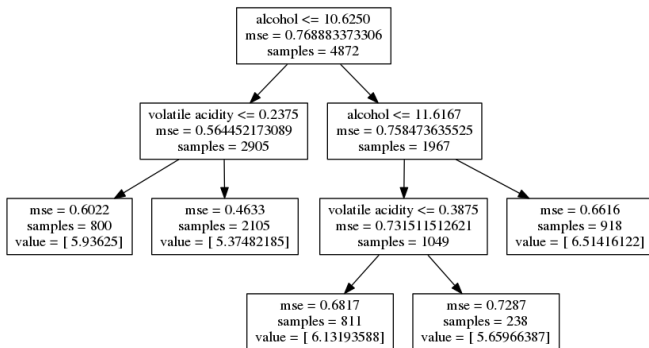
```
# Fit a decision tree
from sklearn.tree import DecisionTreeRegressor
estimator = DecisionTreeRegressor(criterion="mse", # Set i(t) function
                                  max_leaf_nodes=5)
estimator.fit(X_train, y_train)

# Predict target values
y_pred = estimator.predict(X_test)

# MSE on test data
from sklearn.metrics import mean_squared_error
score = mean_squared_error(y_test, y_pred)
>>> 0.572049826453
```

# Visualize and interpret

```
# Display tree
from sklearn.tree import export_graphviz
export_graphviz(estimator, out_file="tree.dot",
                feature_names=feature_names)
```



# Strengths and weaknesses of decision trees

- Non-parametric model, proved to be consistent.
- Support heterogeneous data (continuous, ordered or categorical variables).
- Flexibility in loss functions (but choice is limited).
- Fast to train, fast to predict.
  - In the average case, complexity of training is  $\Theta(pN \log^2 N)$ .
- Easily interpretable.
- Low bias, but usually high variance
  - Solution: Combine the predictions of several randomized trees into a single model.

# Outline

- 1 Decision trees
- 2 Forests of randomized trees**
- 3 Boosting
- 4 Interpreting tree-based models
- 5 Gaussian Processes
- 6 Neural networks

## Bias-variance decomposition in regression

**Theorem.** For the *squared error loss*, the bias-variance decomposition of the expected generalization error at  $X = \mathbf{x}$  is

$$\mathbb{E}_{\mathcal{L}}\{Err(\varphi_{\mathcal{L}}(\mathbf{x}))\} = \text{noise}(\mathbf{x}) + \text{bias}^2(\mathbf{x}) + \text{var}(\mathbf{x})$$

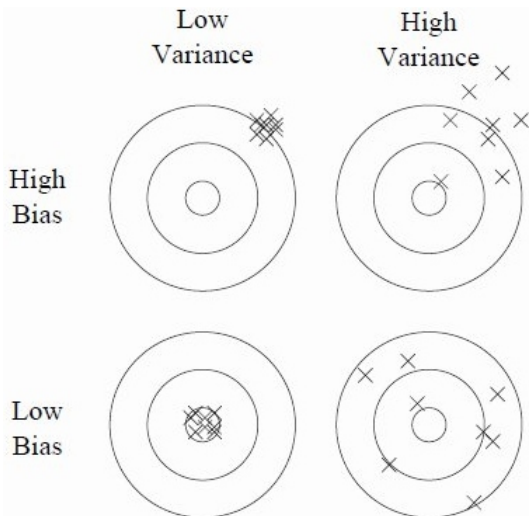
where

$$\text{noise}(\mathbf{x}) = Err(\varphi_B(\mathbf{x})),$$

$$\text{bias}^2(\mathbf{x}) = (\varphi_B(\mathbf{x}) - \mathbb{E}_{\mathcal{L}}\{\varphi_{\mathcal{L}}(\mathbf{x})\})^2,$$

$$\text{var}(\mathbf{x}) = \mathbb{E}_{\mathcal{L}}\{(\mathbb{E}_{\mathcal{L}}\{\varphi_{\mathcal{L}}(\mathbf{x})\} - \varphi_{\mathcal{L}}(\mathbf{x}))^2\}.$$

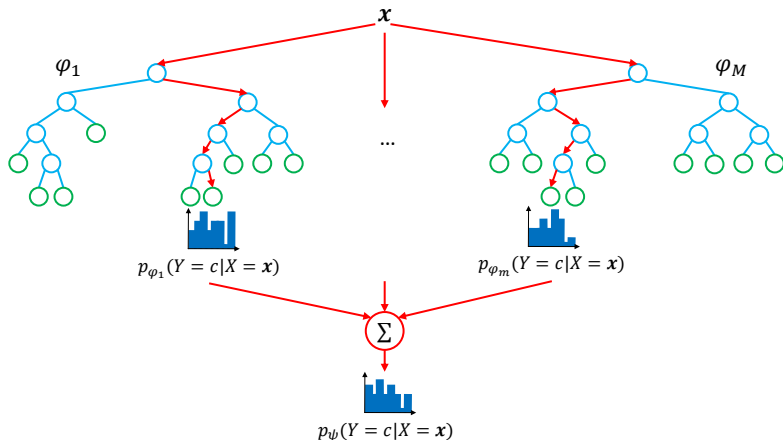
# Bias-variance decomposition



# Diagnosing the generalization error of a decision tree

- (Residual error: Lowest achievable error, independent of  $\varphi_{\mathcal{L}}$ .)
- Bias: Decision trees usually have **low bias**.
- Variance: They often suffer from **high variance**.
- Solution: *Combine the predictions of several randomized trees into a single model.*

# Random Forests (Breiman, 2001; Geurts et al., 2006)



## Randomization

- Bootstrap samples
- Random selection of  $K \leq p$  split variables
- Random selection of the threshold

} Random Forests

} Extra-Trees



# Bias-variance decomposition

**Theorem.** For the squared error loss, the bias-variance decomposition of the expected generalization error

$\mathbb{E}_{\mathcal{L}}\{Err(\psi_{\mathcal{L},\theta_1,\dots,\theta_M}(\mathbf{x}))\}$  at  $X = \mathbf{x}$  of an ensemble of  $M$  randomized models  $\varphi_{\mathcal{L},\theta_m}$  is

$$\mathbb{E}_{\mathcal{L}}\{Err(\psi_{\mathcal{L},\theta_1,\dots,\theta_M}(\mathbf{x}))\} = \text{noise}(\mathbf{x}) + \text{bias}^2(\mathbf{x}) + \text{var}(\mathbf{x}),$$

where

$$\text{noise}(\mathbf{x}) = Err(\varphi_B(\mathbf{x})),$$

$$\text{bias}^2(\mathbf{x}) = (\varphi_B(\mathbf{x}) - \mathbb{E}_{\mathcal{L},\theta}\{\varphi_{\mathcal{L},\theta}(\mathbf{x})\})^2,$$

$$\text{var}(\mathbf{x}) = \rho(\mathbf{x})\sigma_{\mathcal{L},\theta}^2(\mathbf{x}) + \frac{1 - \rho(\mathbf{x})}{M}\sigma_{\mathcal{L},\theta}^2(\mathbf{x}).$$

and where  $\rho(\mathbf{x})$  is the Pearson correlation coefficient between the predictions of two randomized trees built on the same learning set.

# Interpretation of $\rho(\mathbf{x})$ (Louppe, 2014)

**Theorem.** 
$$\rho(\mathbf{x}) = \frac{\mathbb{V}_{\mathcal{L}}\{\mathbb{E}_{\theta|\mathcal{L}}\{\varphi_{\mathcal{L},\theta}(\mathbf{x})\}\}}{\mathbb{V}_{\mathcal{L}}\{\mathbb{E}_{\theta|\mathcal{L}}\{\varphi_{\mathcal{L},\theta}(\mathbf{x})\}\} + \mathbb{E}_{\mathcal{L}}\{\mathbb{V}_{\theta|\mathcal{L}}\{\varphi_{\mathcal{L},\theta}(\mathbf{x})\}\}}$$

In other words, it is the ratio between

- the variance due to the learning set and
- the total variance, accounting for random effects due to both the learning set and the random perturbations.

$\rho(\mathbf{x}) \rightarrow 1$  when variance is mostly due to the learning set;

$\rho(\mathbf{x}) \rightarrow 0$  when variance is mostly due to the random perturbations;

$\rho(\mathbf{x}) \geq 0$ .

# Diagnosing the error of random forests (Louppe, 2014)

- Bias: **Identical** to the bias of a single randomized tree.
- Variance:  $\text{var}(\mathbf{x}) = \rho(\mathbf{x})\sigma_{\mathcal{L},\theta}^2(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M}\sigma_{\mathcal{L},\theta}^2(\mathbf{x})$

As  $M \rightarrow \infty$ ,  $\text{var}(\mathbf{x}) \rightarrow \rho(\mathbf{x})\sigma_{\mathcal{L},\theta}^2(\mathbf{x})$

- The stronger the randomization,  $\rho(\mathbf{x}) \rightarrow 0$ ,  $\text{var}(\mathbf{x}) \rightarrow 0$ .
- The weaker the randomization,  $\rho(\mathbf{x}) \rightarrow 1$ ,  $\text{var}(\mathbf{x}) \rightarrow \sigma_{\mathcal{L},\theta}^2(\mathbf{x})$

**Bias-variance trade-off.** Randomization increases bias but makes it possible to reduce the variance of the corresponding ensemble model. The crux of the problem is to **find the right trade-off**.

# Tuning randomization in sklearn.ensemble

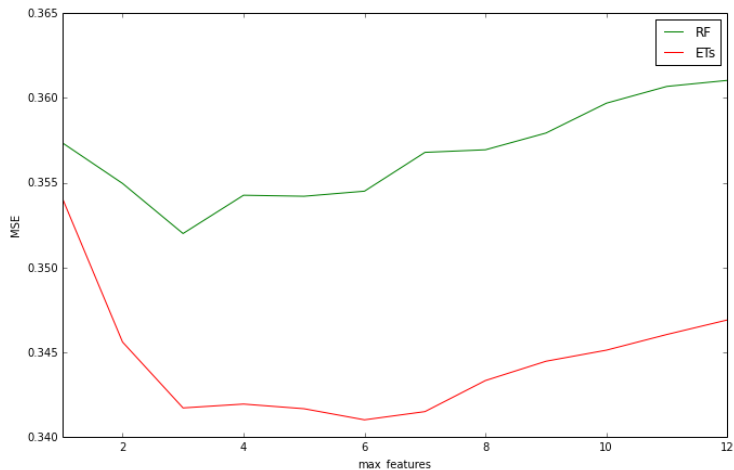
```
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.cross_validation import ShuffleSplit
from sklearn.learning_curve import validation_curve

# Validation of max_features, controlling randomness in forests
param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

_, test_scores = validation_curve(
    RandomForestRegressor(n_estimators=100, n_jobs=-1), X, y,
    cv=ShuffleSplit(n=len(X), n_iter=10, test_size=0.25),
    param_name="max_features", param_range=param_range,
    scoring="mean_squared_error")
test_scores_mean = np.mean(-test_scores, axis=1)
plt.plot(param_range, test_scores_mean, label="RF", color="g")

_, test_scores = validation_curve(
    ExtraTreesRegressor(n_estimators=100, n_jobs=-1), X, y,
    cv=ShuffleSplit(n=len(X), n_iter=10, test_size=0.25),
    param_name="max_features", param_range=param_range,
    scoring="mean_squared_error")
test_scores_mean = np.mean(-test_scores, axis=1)
plt.plot(param_range, test_scores_mean, label="ETs", color="r")
```

## Tuning randomization in sklearn.ensemble



Best-tradeoff: ExtraTrees, for max\_features=6.

# Strengths and weaknesses of forests

- One of the best off-the-shelf learning algorithm, requiring almost no tuning.
- Fine control of bias and variance through averaging and randomization, resulting in better performance.
- Moderately fast to train and to predict.
  - $\Theta(MK\tilde{N}\log^2\tilde{N})$  for RFs (where  $\tilde{N} = 0.632N$ )
  - $\Theta(MKN\log N)$  for ETs
- Embarrassingly parallel (use `n_jobs`).
- Less interpretable than decision trees.

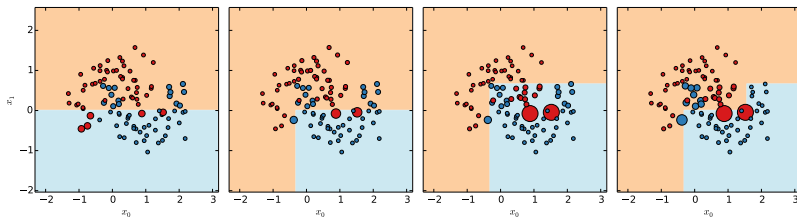
# Outline

- 1 Decision trees
- 2 Forests of randomized trees
- 3 Boosting**
- 4 Interpreting tree-based models
- 5 Gaussian Processes
- 6 Neural networks

# AdaBoost (Freund and Schapire, 1995)

Can we build a strong learner from multiple weak learners?

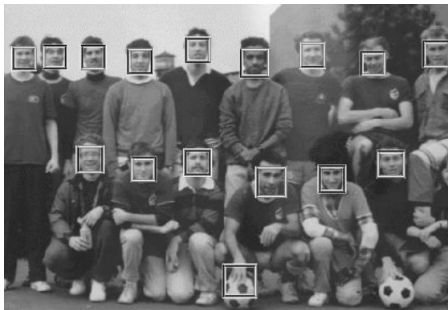
- Ensemble: each member is an expert on the errors of its predecessor;
- Iteratively re-weights training examples based on errors.





## Huge success

- Viola-Jones Face Detector (2001)



- Freund & Schapire won the Gödel prize (2003)

# Gradient Boosted Machines (Friedman, 2001)

Statistical view of boosting:

- Ultimately, we are interested in finding

$$f^* = \arg \min_f \mathbb{E}_{X,Y}[L(Y, f(X))]$$

for an arbitrary loss  $L$ ;

- Under the boosting assumption that  $f^*$  can be expressed as an additive model  $\hat{f}$  of the form

$$\hat{f} = c + \sum_{m=1}^M \rho_m h_m.$$

## Greedy approximation

- The ensemble is built in a forward stagewise fashion:

$$f_m = f_{m-1} + \rho_m h_m$$

- In accordance with the empirical risk minimization principle, the weak learner  $\rho_m h_m$  is chosen to minimize the loss  $L$  on the training set, given the current model  $f_{m-1}$ :

$$f_m = f_{m-1} + \arg \min_{\rho, h} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \rho h(x_i))$$

$$f_0 = \arg \min_c \sum_{i=1}^N L(y_i, c)$$

- Solving for  $\rho, h$  at each step for an arbitrary loss  $L$  and/or arbitrary family  $\mathcal{H}$  is a difficult optimization problem.

# Steepest descent in function space

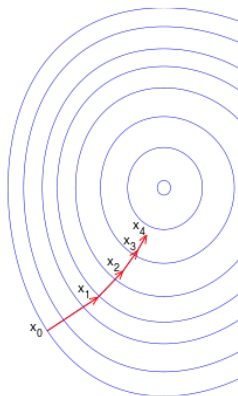
Solution: approximate using steepest descent in function space, by choosing a function  $h$  that is the most parallel to the negative gradient  $\{g_m(x_i)\}_{i=1}^N$  on the training data:

$$g_m(x) = \mathbb{E}_{Y|X=x} \left[ \frac{\partial L(Y, f(x))}{\partial f(x)} \right]_{f(x)=f_{m-1}(x)}$$

$$h_m = \arg \min_h \sum_{i=1}^N (-g_m(x_i) - h(x_i))^2$$

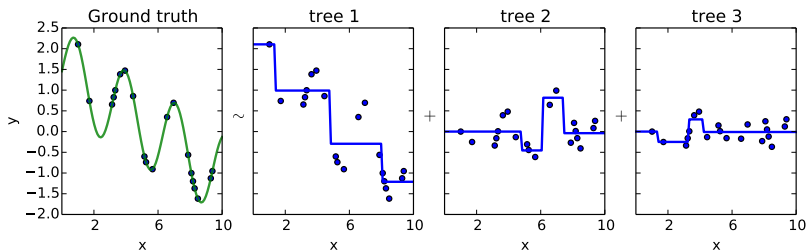
$$\rho_m = \arg \min_{\gamma} L(y_i, f_{m-1}(x_i) + \rho h_m(x_i))$$

where the second equation is solved using a standard regression algorithm (e.g. decision trees) and the third using line search.



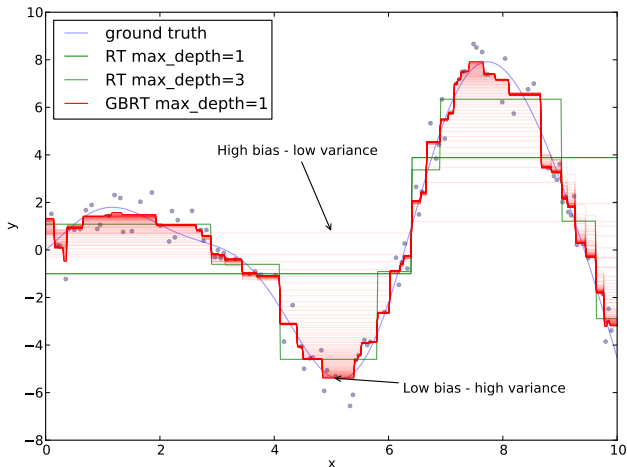
## In short

- GBM fits an additive model of the form  $\hat{f} = c + \sum_{m=1}^M \rho_m h_m$ .
- Where  $h_m$  are regression trees approximating the negative gradient step  $-\left[\frac{\partial L(Y, f(x))}{\partial f(x)}\right]_{f(x)=f_{m-1}(x)}$ .
- For  $L(Y, f(X)) = (Y - f(X))^2$ , regression trees  $h_m$  approximate the residuals  $y_i - f_{m-1}(x_i)$ .

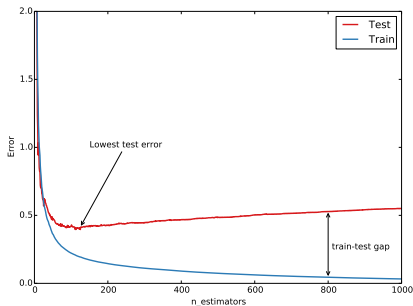


## Example

```
from sklearn.ensemble import GradientBoostingRegressor  
est = GradientBoostingRegressor(n_estimators=100).fit(X, y)
```



# Model complexity and overfitting

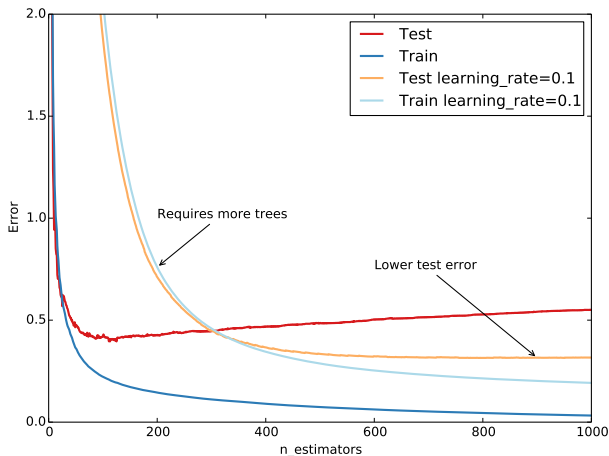


GBM provides a number of knobs to control overfitting:

- Shrinkage
- Tree structure
- Stochastic Gradient Boosting

## Regularization: shrinkage

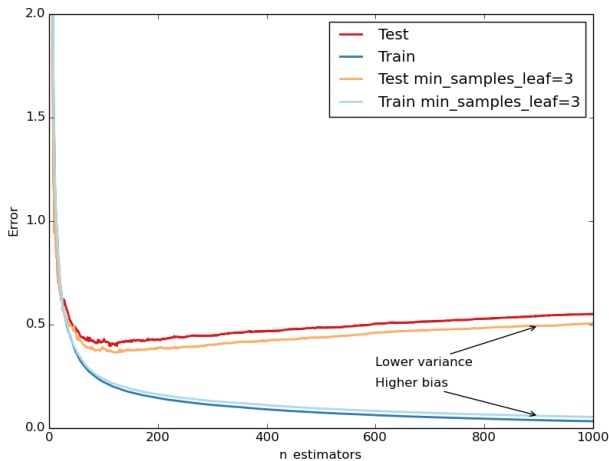
- Slow learning by shrinking tree predictions with  $0 < \text{learning\_rate} \leq 1$
- Lower learning\_rate requires higher n\_estimators





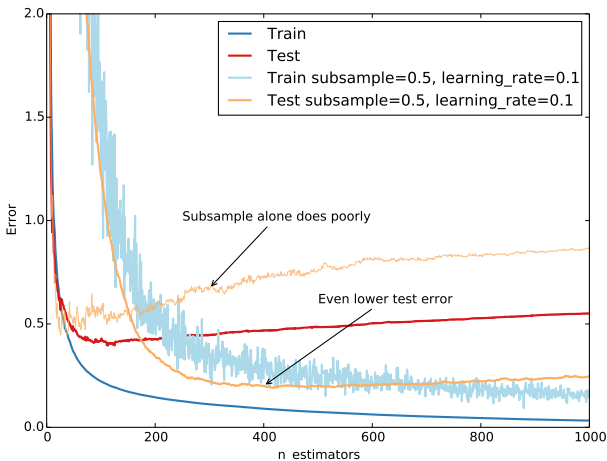
## Regularization: tree structure

- 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: 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`.

# Strengths and weaknesses of Boosting

- Often **more accurate** than random forests.
- **Flexible framework**, that can adapt to arbitrary loss functions.
- Fine control of under/overfitting through **regularization** (e.g., learning rate, subsampling, tree structure, penalization term in the loss function, etc).
- **Careful tuning** required.
- **Slow** to train, **fast** to predict.

# Outline

- 1 Decision trees
- 2 Forests of randomized trees
- 3 Boosting
- 4 Interpreting tree-based models**
- 5 Gaussian Processes
- 6 Neural networks

# Variable selection/ranking/exploration

Tree-based models come with built-in methods for variable selection, ranking or exploration.

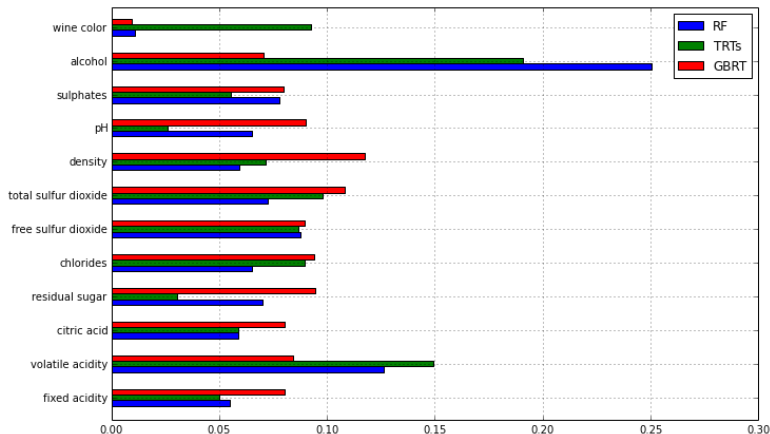
The main goals are:

- To reduce training times;
- To enhance generalisation by reducing overfitting;
- To uncover relations between variables and ease model interpretation.

# Variable importances

```
# Variable importances with Random Forest, default parameters  
forest = RandomForestRegressor(n_estimators=10000, n_jobs=-1).fit(X, y)  
forest.feature_importances_
```

# Variable importances



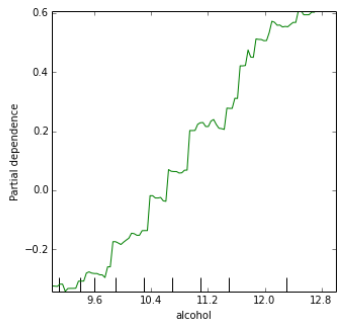
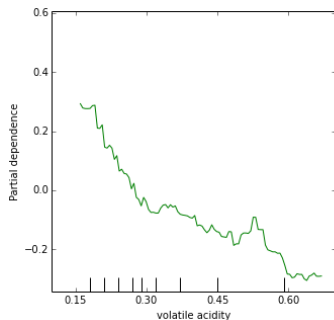
Importances are measured only through the eyes of the model.  
They may not tell the entire nor the same story! (Louppe et al., 2013)



# Partial dependence plots

Relation between the response  $Y$  and a subset of features, marginalized over all other features.

```
from sklearn.ensemble.partial_dependence import plot_partial_dependence
plot_partial_dependence(gbrt, X,
                        features=[1, 10], feature_names=feature_names)
```



# Outline

- 1 Decision trees
- 2 Forests of randomized trees
- 3 Boosting
- 4 Interpreting tree-based models
- 5 Gaussian Processes**
- 6 Neural networks

# Outline

- 1 Decision trees
- 2 Forests of randomized trees
- 3 Boosting
- 4 Interpreting tree-based models
- 5 Gaussian Processes
- 6 Neural networks**

# References

- Breiman, L. (2001). Random Forests. *Machine learning*, 45(1):5–32.
- Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*.
- Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer.
- Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232.
- Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.
- Louppe, G. (2014). Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*.
- Louppe, G., Wehenkel, L., Sutter, A., and Geurts, P. (2013). Understanding variable importances in forests of randomized trees. In *Advances in Neural Information Processing Systems*, pages 431–439.