Supervised Learning: Non-parametric methods

Gilles Louppe (@glouppe)

October 17, 2016

Outline

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 $\mathcal{L} = (\mathtt{X}, \mathtt{y})$ 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,)`

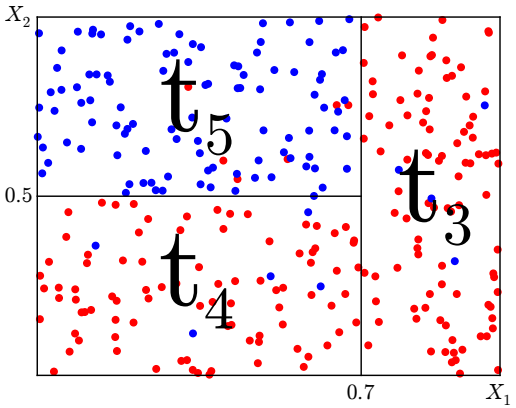
    E.g., wine taste preferences (from 0 to 10):

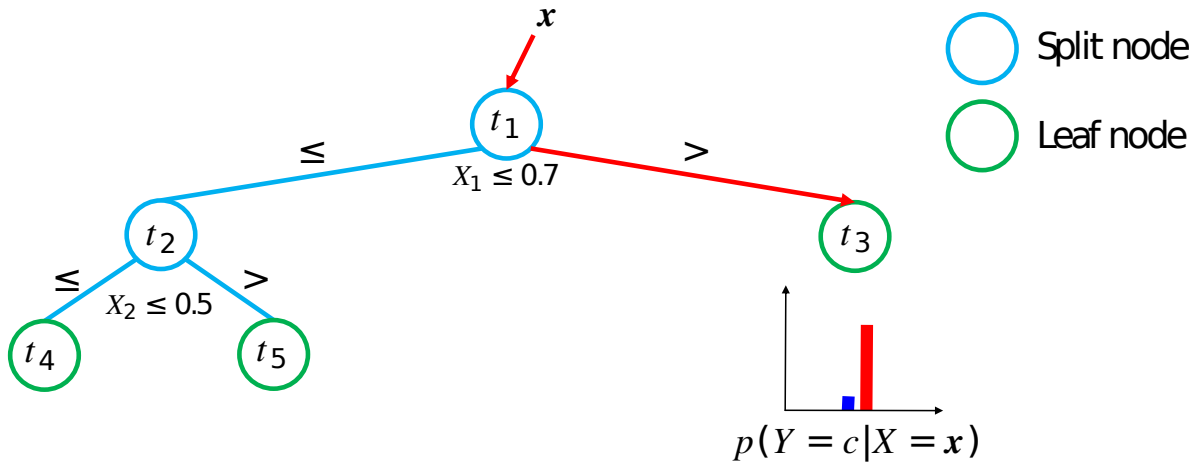
    ```
    y = [5 5 5 ... 6 7 6]
    ```

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

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

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 $\widehat{y}_t$
  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^*
    $t_L =$ BUILDDECISIONTREE($\mathcal{L}_{t_L}$)
    $t_R =$ 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)

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

Mean squared error (regression)

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

Least absolute deviance (regression)

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

Density estimation

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

Sample weights

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

Weighted mean squared error

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

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

sklearn.tree

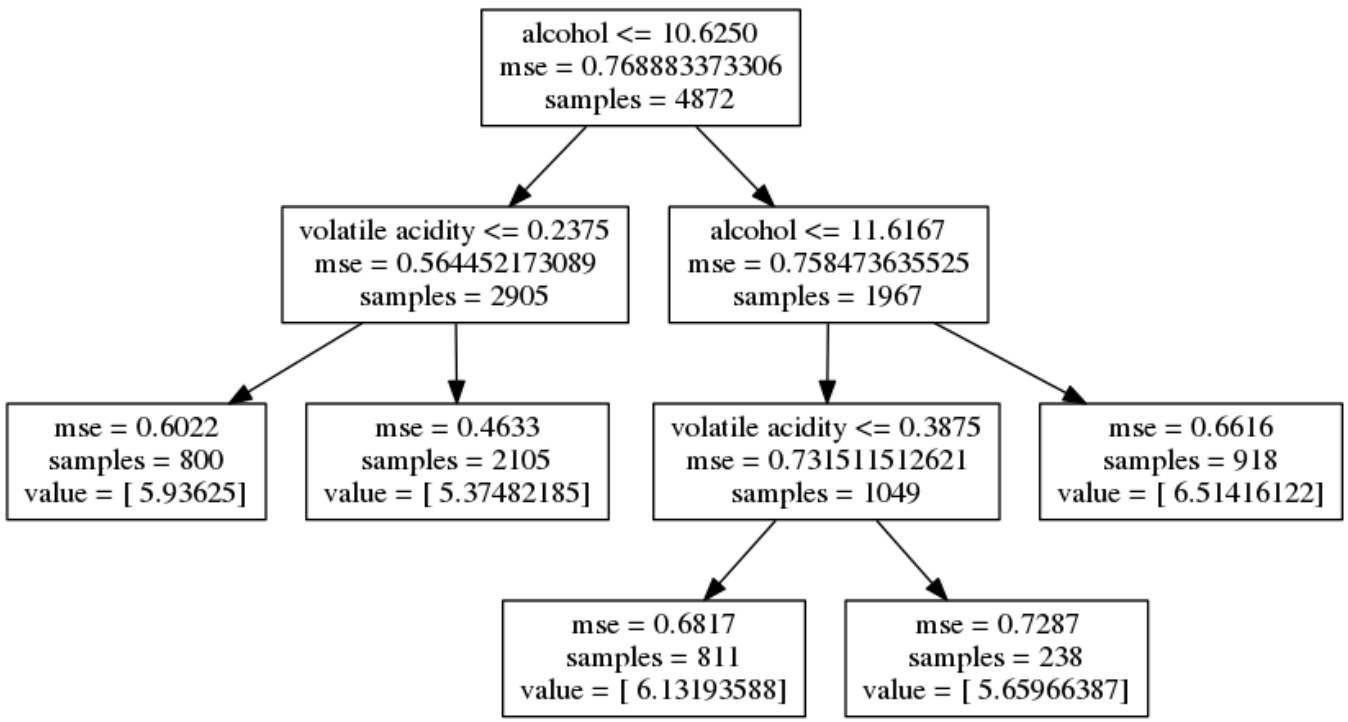
```
# Fit a decision tree
from sklearn.tree import DecisionTreeRegressor
estimator = DecisionTreeRegressor(criterion="mse",    # Set i(t) function
                                  max_leaf_nodes=5)
estimator.fit(X_train, y_train)

# 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)
```

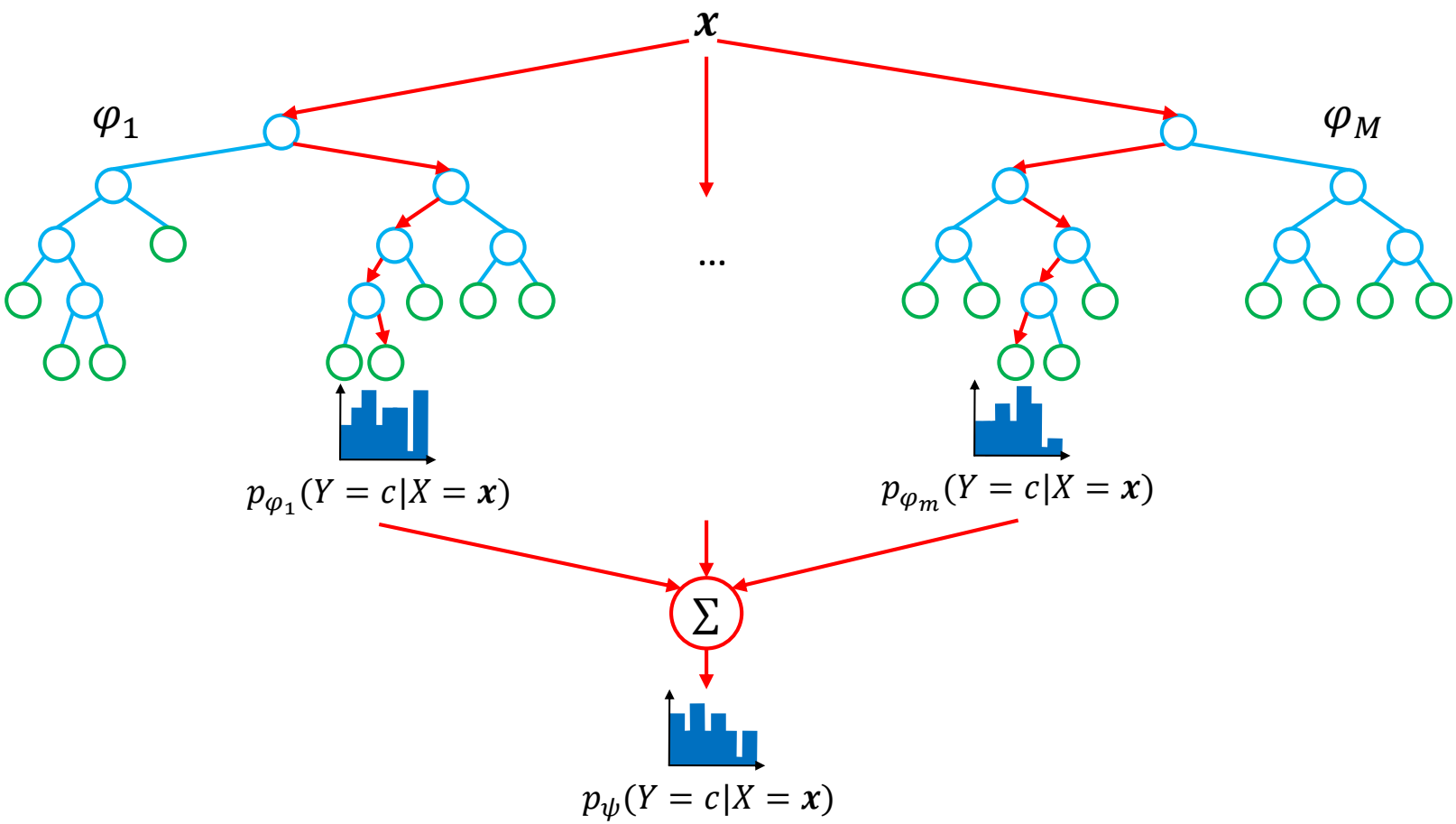


Strengths and weaknesses of decision trees

- 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

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



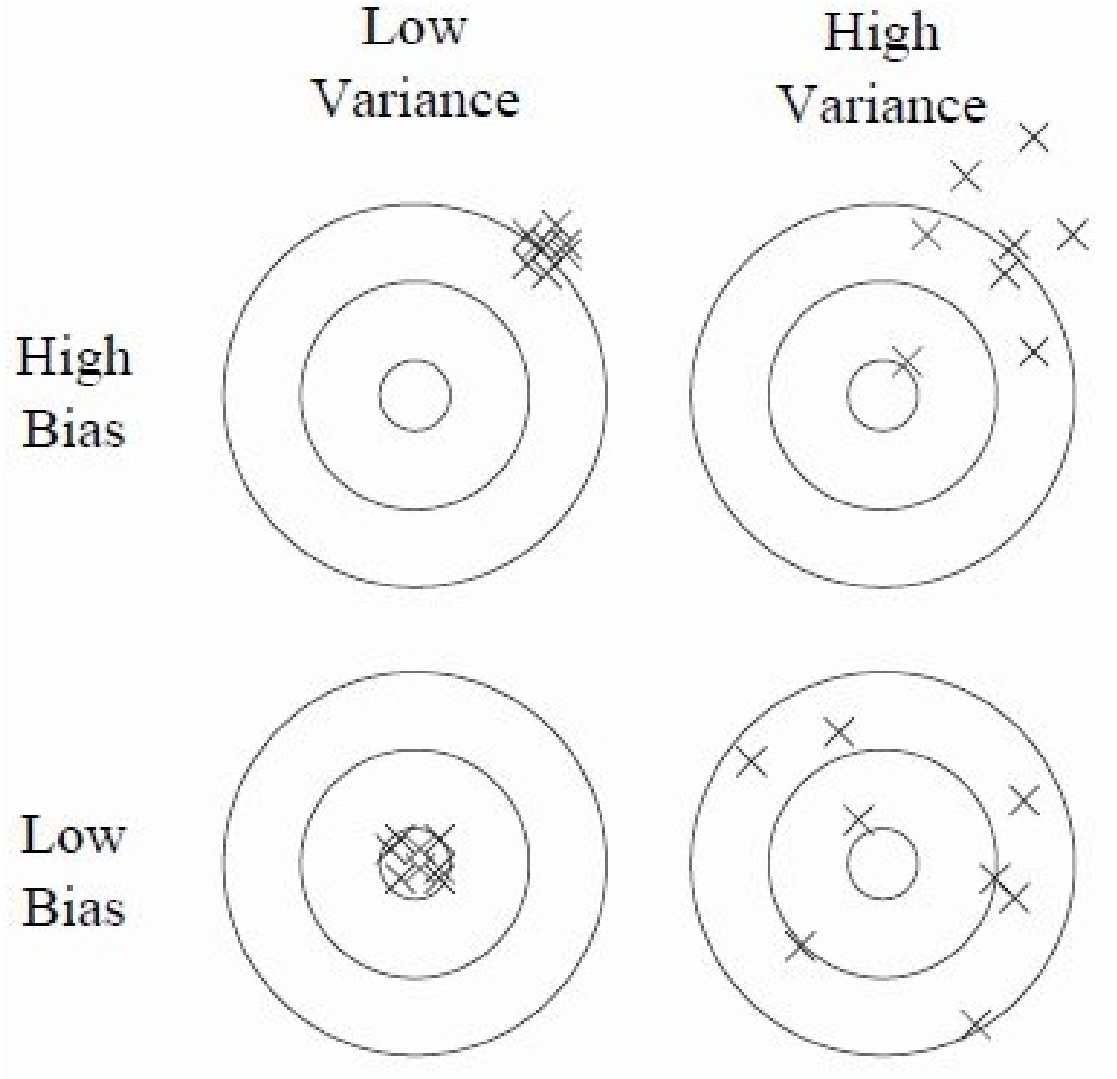
Randomization

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

} Random Forests

} Extra-Trees

Bias and variance



Bias-variance decomposition

Theorem. For the squared error loss, the bias-variance decomposition of the expected generalization error $\mathbb{E}_{\mathcal{L}}\{Err(\psi_{\mathcal{L},\theta_1,\ldots,\theta_M}(\mathbf{x}))\}$ at $X = \mathbf{x}$ of an ensemble of M randomized models $\varphi_{\mathcal{L},\theta_m}$ is

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

where

$$\begin{aligned}
\text{noise}(\mathbf{x}) &= Err(\varphi_B(\mathbf{x})), \\
\text{bias}^2(\mathbf{x}) &= (\varphi_B(\mathbf{x}) - \mathbb{E}_{\mathcal{L},\theta}\{\varphi_{\mathcal{L},\theta}(\mathbf{x})\})^2, \\
\text{var}(\mathbf{x}) &= \rho(\mathbf{x})\sigma^2_{\mathcal{L},\theta}(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M}\sigma^2_{\mathcal{L},\theta}(\mathbf{x}).
\end{aligned}$$

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

Diagnosing the error of random forests (Louppe, 2014)

- Bias: Identical to the bias of a single randomized tree.
- Variance: $\text{var}(\mathbf{x}) = \rho(\mathbf{x})\sigma^2_{\mathcal{L},\theta}(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M}\sigma^2_{\mathcal{L},\theta}(\mathbf{x})$
  As $M \to \infty$, $\text{var}(\mathbf{x}) \to \rho(\mathbf{x})\sigma^2_{\mathcal{L},\theta}(\mathbf{x})$
  - The stronger the randomization, $\rho(\mathbf{x}) \to 0$, $\text{var}(\mathbf{x}) \to 0$.
  - The weaker the randomization, $\rho(\mathbf{x}) \to 1$, $\text{var}(\mathbf{x}) \to \sigma^2_{\mathcal{L},\theta}(\mathbf{x})$

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

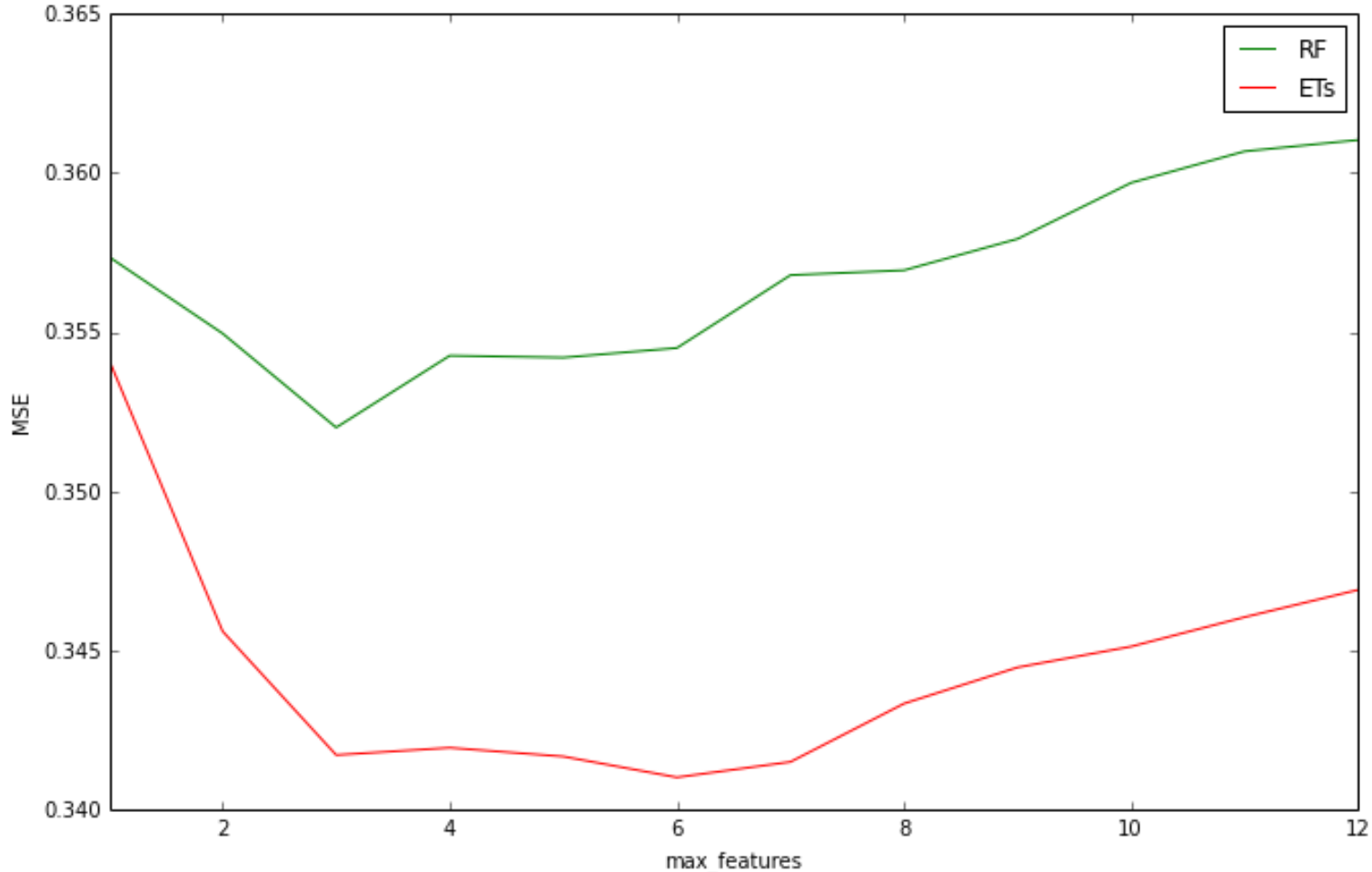
```
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.cross_validation import ShuffleSplit
from sklearn.learning_curve import validation_curve

# 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`.

Variable importances

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

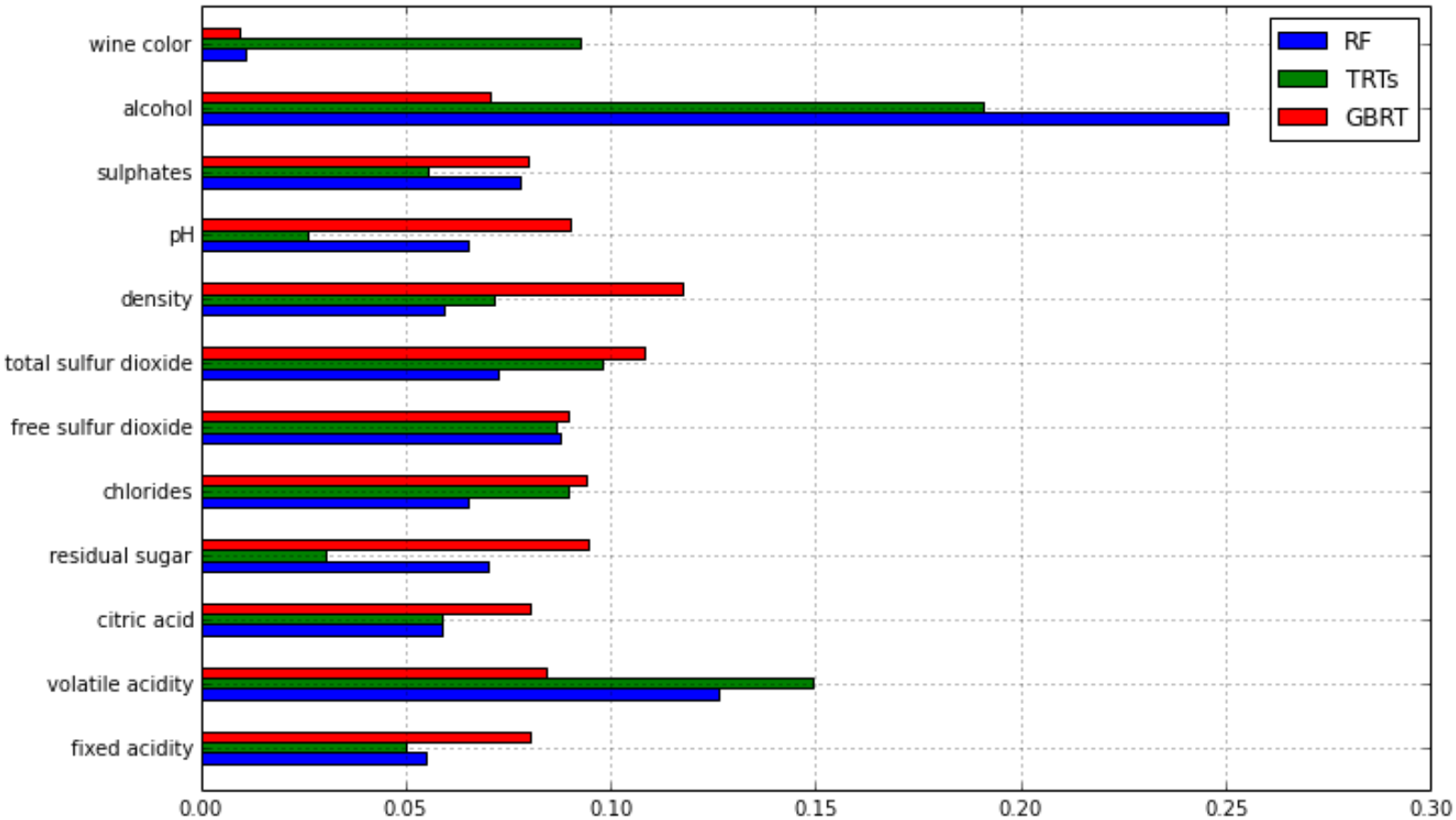
Variable selection/ranking/exploration

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



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

Strengths and weaknesses of forests

- One of the best off-the-self 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.
  - $\Theta(MK\widetilde{N}\log^2\widetilde{N})$ for RFs (where $\widetilde{N} = 0.632N$)
  - $\Theta(MKN\log N)$ for ETs
- Embarrassingly parallel (use `n_jobs`).
- Less interpretable than decision trees.

Outline

Gradient Boosted Regression Trees (Friedman, 2001)

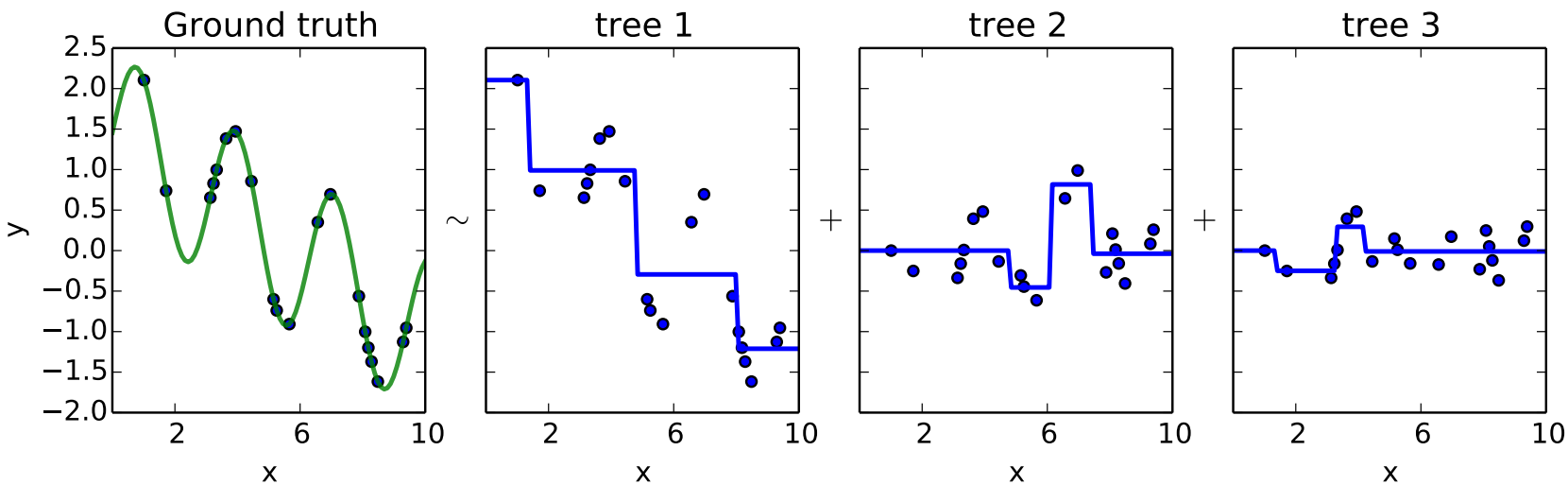
- GBRT fits an additive model of the form

$$\varphi(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

- The ensemble is built in a forward stagewise manner. That is

$$\varphi_m(x) = \varphi_{m-1}(x) + \gamma_m h_m(x)$$

where $h_m : \mathcal{X} \mapsto \mathbb{R}$ is a regression tree *approximating* the gradient step $\Delta_\varphi L(Y, \varphi_{m-1}(X))$.



Careful tuning required

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.cross_validation import ShuffleSplit
from sklearn.grid_search import GridSearchCV

# Careful tuning is required to obtained good results
param_grid = {"loss": ["mse", "lad", "huber"],
              "learning_rate": [0.1, 0.01, 0.001],
              "max_depth": [3, 5, 7],
              "min_samples_leaf": [1, 3, 5],
              "subsample": [1.0, 0.9, 0.8]}

est = GradientBoostingRegressor(n_estimators=1000)
grid = GridSearchCV(est, param_grid,
                    cv=ShuffleSplit(n=len(X), n_iter=10, test_size=0.25),
                    scoring="mean_squared_error",
                    n_jobs=-1).fit(X, y)

gbrt = grid.best_estimator_
```

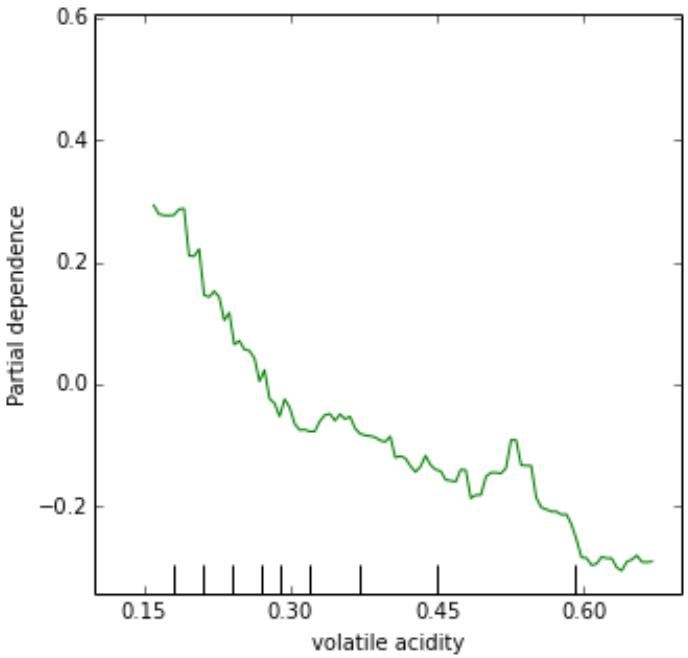
See our PyData 2014 tutorial for further guidance

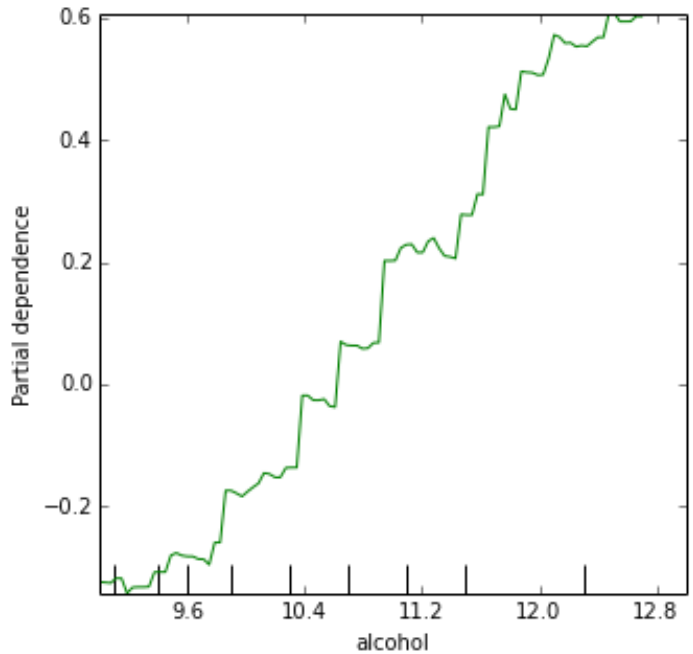
https://github.com/pprett/pydata-gbrt-tutorial

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)
```





Strengths and weaknesses of GBRT

- 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

Outline

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*.

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., Sutera, A., and Geurts, P. (2013). Understanding variable importances in forests of randomized trees. In *Advances in Neural Information Processing Systems*, pages 431–439.