

# Tree models with Scikit-Learn

## Great learners with little assumptions

Material: <https://github.com/glouppe/talk-pydata2015>

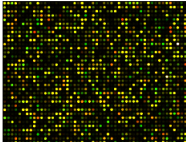
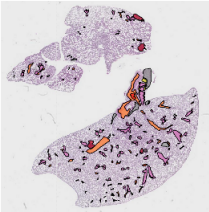
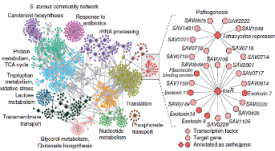
Gilles Louppe (@glouppe)  
*CERN*

PyData, April 3, 2015

# Outline

- ① Motivation
- ② Growing decision trees
- ③ Random forests
- ④ Boosting
- ⑤ Reading tree leaves
- ⑥ Summary

# Motivation



Google

Google Search

Free Training Library

# Running example

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

learn a **model**

to predict **wine taste preferences**.



# Outline

- 1 Motivation
- 2 Growing decision trees**
- 3 Random Forests
- 4 Boosting
- 5 Reading tree leaves
- 6 Summary

# Supervised learning

- Data comes as a finite learning set  $\mathcal{L} = (\mathcal{X}, \mathcal{Y})$  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,)

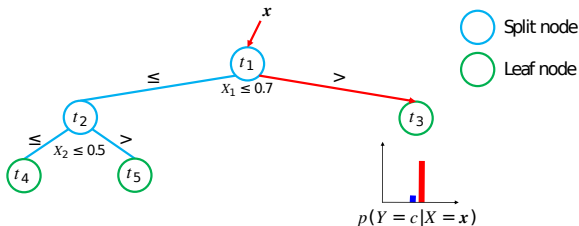
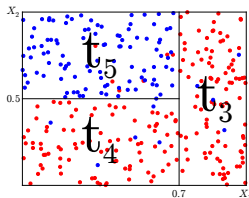
E.g., wine taste preferences (from 0 to 10):

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

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

$$Err(\varphi_{\mathcal{L}}) = \mathbb{E}_{\mathcal{X}, \mathcal{Y}} \{L(\mathcal{Y}, \varphi_{\mathcal{L}}.\text{predict}(\mathcal{X}))\}.$$

# 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**

# 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

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



# sklearn.tree

```
# Fit a decision tree
from sklearn.tree import DecisionTreeRegressor
estimator = DecisionTreeRegressor(criterion="mse",      # Set  $i(t)$  function
                                  max_leaf_nodes=5)    # Tune model complexity
                                                    # with max_leaf_nodes,
                                                    # max_depth or
                                                    # min_samples_split

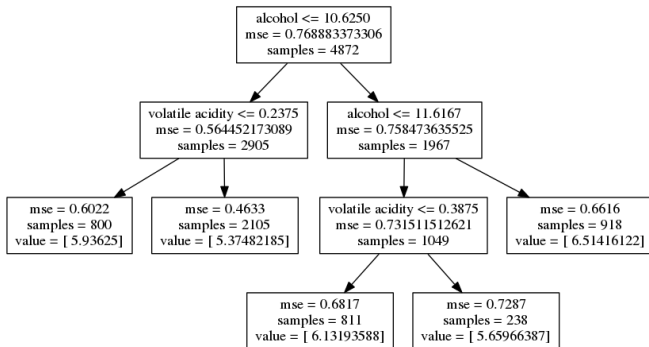
estimator.fit(X_train, y_train)

# 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)
```



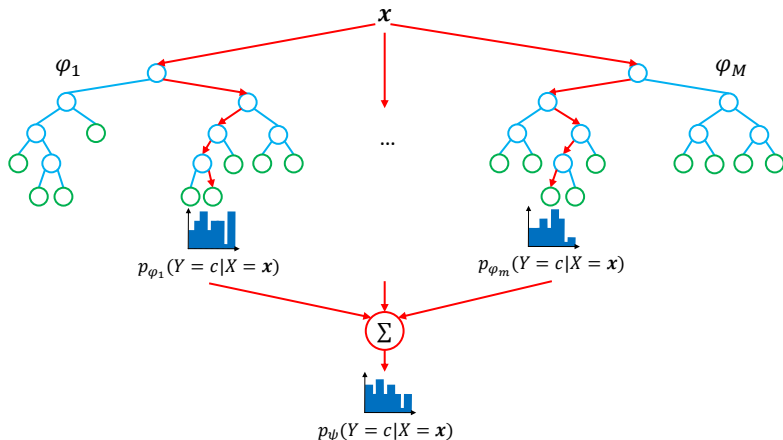
# Strengths and weaknesses of decision trees

- 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 Motivation
- 2 Growing decision trees
- 3 Random Forests**
- 4 Boosting
- 5 Reading tree leaves
- 6 Summary

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



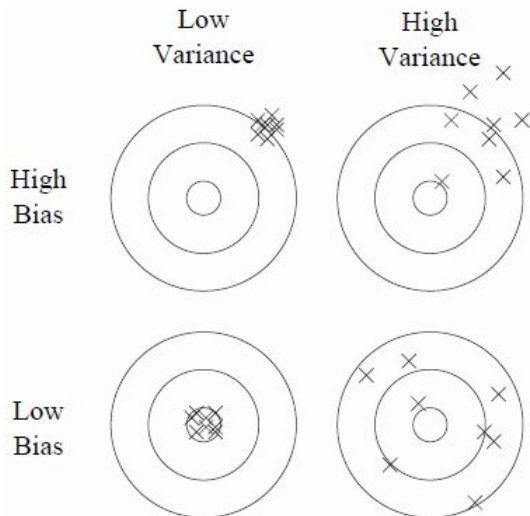
## Randomization

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

} Random Forests

} Extra-Trees

# Bias and variance



## Bias-variance decomposition

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

$\mathbb{E}_{\mathcal{L}}\{Err(\psi_{\mathcal{L},\theta_1,\dots,\theta_M}(\mathbf{x}))\}$  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.

# 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

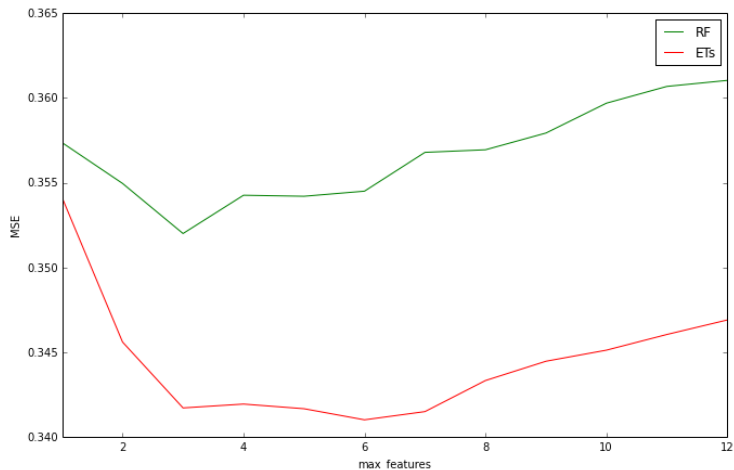
```
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.cross_validation import ShuffleSplit
from sklearn.learning_curve import validation_curve

# Validation of max_features, controlling randomness in forests
param_name = "max_features"
param_range = range(1, X.shape[1]+1)

for Forest, color, label in [(RandomForestRegressor, "g", "RF"),
                             (ExtraTreesRegressor, "r", "ETs")]:
    _, test_scores = validation_curve(
        Forest(n_estimators=100, n_jobs=-1), X, y,
        cv=ShuffleSplit(n=len(X), n_iter=10, test_size=0.25),
        param_name=param_name, 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=label, color=color)

plt.xlabel(param_name)
plt.xlim(1, max(param_range))
plt.ylabel("MSE")
plt.legend(loc="best")
plt.show()
```

## Tuning randomization in sklearn.ensemble



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

# Strengths and weaknesses of forests

- 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
  - $\Theta(MKN\log N)$  for ETs
- Embarrassingly parallel (use `n_jobs`)
- Less interpretable than decision trees

# Outline

- 1 Motivation
- 2 Growing decision trees
- 3 Random Forests
- 4 Boosting**
- 5 Reading tree leaves
- 6 Summary

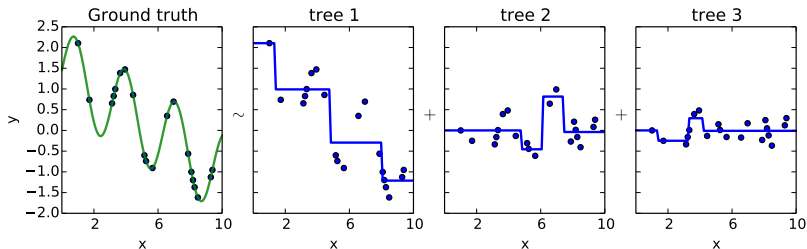
# Gradient Boosted Regression Trees (Friedman, 2001)

- 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, where each weak learner is a successive gradient step

$$\varphi_m(x) = \varphi_{m-1}(x) + \gamma_m \sum_{i=1}^N \Delta_{\varphi} L(y_i, \varphi_{m-1}(x_i))$$



## 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 = {"learning_rate": [0.1, 0.01, 0.001],
              "subsample": [1.0, 0.9, 0.8],
              "max_depth": [3, 5, 7],
              "min_samples_leaf": [1, 3, 5]}

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>

# Strengths and weaknesses of GBRT

- 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 Motivation
- 2 Growing decision trees
- 3 Random Forests
- 4 Boosting
- 5 Reading tree leaves**
- 6 Summary



# Variable importances

```
importances = pd.DataFrame()

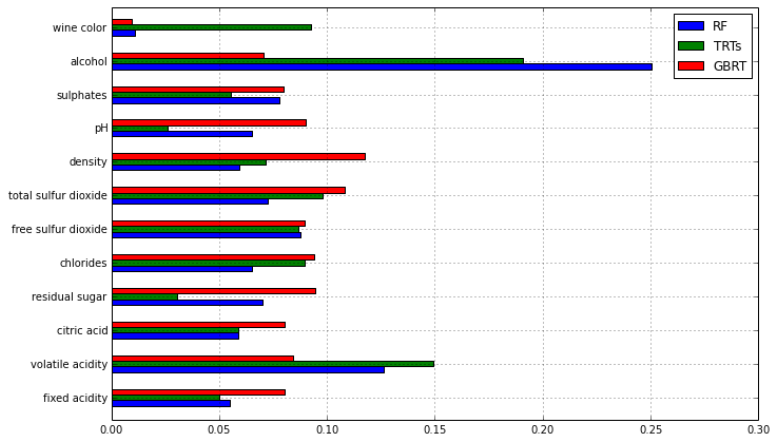
# Variable importances with Random Forest, default parameters
est = RandomForestRegressor(n_estimators=10000, n_jobs=-1).fit(X, y)
importances["RF"] = pd.Series(est.feature_importances_,
                              index=feature_names)

# Variable importances with Totally Randomized Trees
est = ExtraTreesRegressor(max_features=1, max_depth=3,
                          n_estimators=10000, n_jobs=-1).fit(X, y)
importances["TRTs"] = pd.Series(est.feature_importances_,
                                 index=feature_names)

# Variable importances with GBRT
importances["GBRT"] = pd.Series(gbrt.feature_importances_,
                                 index=feature_names)

importances.plot(kind="barh")
```

# Variable importances

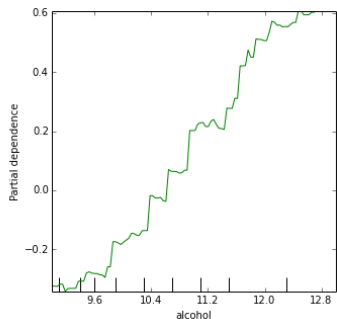
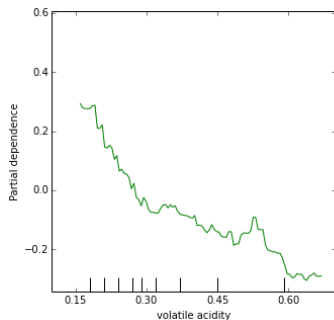


Importances are measured only through the eyes of the model.  
They may not tell the entire nor the same story! (Louppe et al., 2013)

# 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)
```



# Embedding

```
from sklearn.ensemble import RandomTreesEmbedding
from sklearn.decomposition import TruncatedSVD

# Project wines through a forest of totally randomized trees
# and use the leafs the samples end into as a high-dimensional representation
hasher = RandomTreesEmbedding(n_estimators=1000)
X_transformed = hasher.fit_transform(X)

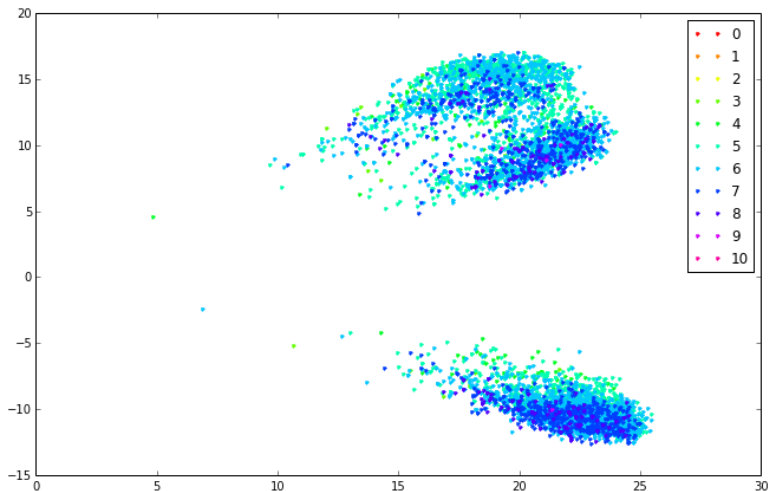
# Plot wines on a plane using the 2 principal components
svd = TruncatedSVD(n_components=2)
coords = svd.fit_transform(X_transformed)

n_values = 10 + 1 # Wine preferences are from 0 to 10
cm = plt.get_cmap("hsv")
colors = (cm(1. * i / n_values) for i in range(n_values))

for k, c in zip(range(n_values), colors):
    plt.plot(coords[y == k, 0], coords[y == k, 1], 'r.', label=k, color=c)

plt.legend()
plt.show()
```

# Embedding



Can you guess what these 2 clusters correspond to?

# Outline

- 1 Motivation
- 2 Growing decision trees
- 3 Random Forests
- 4 Boosting
- 5 Reading tree leaves
- 6 Summary**

# Summary

- Tree-based methods offer a **flexible and efficient non-parametric** framework for classification and regression.
- Applicable to a wide variety of problems, with a **fine control** over the model that is learned.
- Assume a good feature representation – i.e., tree-based methods are often **not that good on very raw input data**, like pixels, speech signals, etc.
- **Insights on the problem** under study (variable importances, dependence plots, embedding, ...)
- **Efficient implementation** in Scikit-Learn.

Join us on  
[https://github.com/  
scikit-learn/scikit-learn](https://github.com/scikit-learn/scikit-learn)





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