Understanding Random Forests

Gilles Louppe (@glouppe)

CERN, September 21, 2015
Outline

1. Motivation
2. Growing decision trees
3. Random forests
4. Boosting
5. Variable importances
6. Summary
Motivation
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. Variable importances

6. Summary
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 $(n_{\text{samples}}, n_{\text{features}})$

    E.g., feature values for wine physicochemical properties:
    
    # fixed acidity, volatile acidity, ...
    
    $\mathbf{X} = [
    \begin{bmatrix}
    7.4 & 0. & \ldots & 0.56 & 9.4 & 0. \\
    7.8 & 0. & \ldots & 0.68 & 9.8 & 0. \\
    \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
    7.8 & 0.04 & \ldots & 0.65 & 9.8 & 0. \\
    \end{bmatrix}
    $

  - **Output values** are given as an array of shape $(n_{\text{samples}},)$

    E.g., wine taste preferences (from 0 to 10):
    
    $\mathbf{y} = [5 \ 5 \ 5 \ \ldots \ 6 \ 7 \ 6]$

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

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

Decision trees (Breiman et al., 1984)

\[
\begin{align*}
\text{function } & \text{BuildDecisionTree}(L) \\
& \text{Create node } t \\
& \text{if the stopping criterion is met for } t \text{ then} \\
& \quad \text{Assign a model to } \hat{y}_t \\
& \text{else} \\
& \quad \text{Find the split on } L \text{ that maximizes impurity decrease} \\
& \quad \quad s^* = \arg \max_s i(t) - p_L i(t_L^s) - p_R i(t_R^s) \\
& \quad \text{Partition } L \text{ into } L_{t_L} \cup L_{t_R} \text{ according to } s^* \\
& \quad t_L = \text{BuildDecisionTree}(L_{t_L}) \\
& \quad t_R = \text{BuildDecisionTree}(L_{t_R}) \\
& \text{end if} \\
& \text{return } t \\
& \text{end function}
\end{align*}
\]
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)
\[ \hat{y}_t = \arg \max_{c \in Y} p(c|t), \quad i(t) = \text{entropy}(t) \text{ or } i(t) = \text{gini}(t) \]

Mean squared error (regression)
\[ \hat{y}_t = \text{mean}(y|t), \quad i(t) = \frac{1}{N_t} \sum_{x,y \in \mathcal{L}_t} (y - \hat{y}_t)^2 \]

Least absolute deviance (regression)
\[ \hat{y}_t = \text{median}(y|t), \quad i(t) = \frac{1}{N_t} \sum_{x,y \in \mathcal{L}_t} |y - \hat{y}_t| \]

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**

\[
\hat{y}_t = \frac{1}{\sum_w w} \sum_{x,y,w \in \mathcal{L}_t} wy
\]

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

Weights are assumed to be non-negative since these quantities may otherwise be undefined. (E.g., what if \(\sum_w w < 0\)?)
# Fit a decision tree
from sklearn.tree import DecisionTreeRegressor
estimator = DecisionTreeRegressor(criterion="mse", 
                                   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
# Display tree

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

```
 alcohol <= 10.6250
  mse = 0.768883373306
  samples = 4872

  volatile acidity <= 0.2375
    mse = 0.564452173089
    samples = 2905

    mse = 0.6022
    samples = 800
    value = [5.93625]

    mse = 0.4633
    samples = 2105
    value = [5.37482185]

  alcohol <= 11.6167
    mse = 0.758473635525
    samples = 1967

    volatile acidity <= 0.3875
      mse = 0.73151512621
      samples = 1049

      mse = 0.6616
      samples = 918
      value = [6.51416122]

      mse = 0.6817
      samples = 811
      value = [6.13193588]

      mse = 0.7287
      samples = 238
      value = [5.65966387]
```
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 Variable importances

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

\[
\sum_{\varphi_1} p_{\varphi_1}(Y = c|X = x) \quad ... \quad \sum_{\varphi_M} p_{\varphi_M}(Y = c|X = x)
\]

\[
p_{\varphi_1}(Y = c|X = x)
\]

\[
p_{\varphi_M}(Y = c|X = x)
\]

\[
p_{\psi}(Y = c|X = x)
\]

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

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

where

\[
\text{noise}(x) = \text{Err}(\varphi_B(x)),
\]

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

\[
\text{var}(x) = \rho(x)\sigma^2_{\mathcal{L}, \theta}(x) + \frac{1 - \rho(x)}{M}\sigma^2_{\mathcal{L}, \theta}(x).
\]

and where \( \rho(x) \) 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**: \( \text{var}(\mathbf{x}) = \rho(\mathbf{x})\sigma_{\mathcal{L},\theta}(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M} \sigma_{\mathcal{L},\theta}(\mathbf{x}) \)

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

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

**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`.
Benchmarks and implementation

Scikit-Learn provides a robust implementation combining both algorithmic and code optimizations. It is one of the fastest among all libraries and programming languages.
Benchmarks and implementation

Andreas Mueller
@t3kc1t

Just a quick reminder what sklearn random forests look like on EC2. Want?
aws.amazon.com/grants/
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.
  - $\Theta(MK\tilde{N}\log^2\tilde{N})$ for RFs (where $\tilde{N} = 0.632N$)
  - $\Theta(MKN\log N)$ for ETs

• Embarrassingly parallel (use $n_{\text{jobs}}$).

• Less interpretable than decision trees.
Outline

1 Motivation

2 Growing decision trees

3 Random Forests

4 Boosting

5 Variable importances

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. That is
  \[ \varphi_m(x) = \varphi_{m-1}(x) + \gamma_m h_m(x) \]

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

```python
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
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

1 Motivation

2 Growing decision trees

3 Random Forests

4 Boosting

5 Variable importances

6 Summary
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.
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.

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

1 Motivation

2 Growing decision trees

3 Random Forests

4 Boosting

5 Variable importances

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.
References


