arXiv.org > cs > arXiv:2110.04374

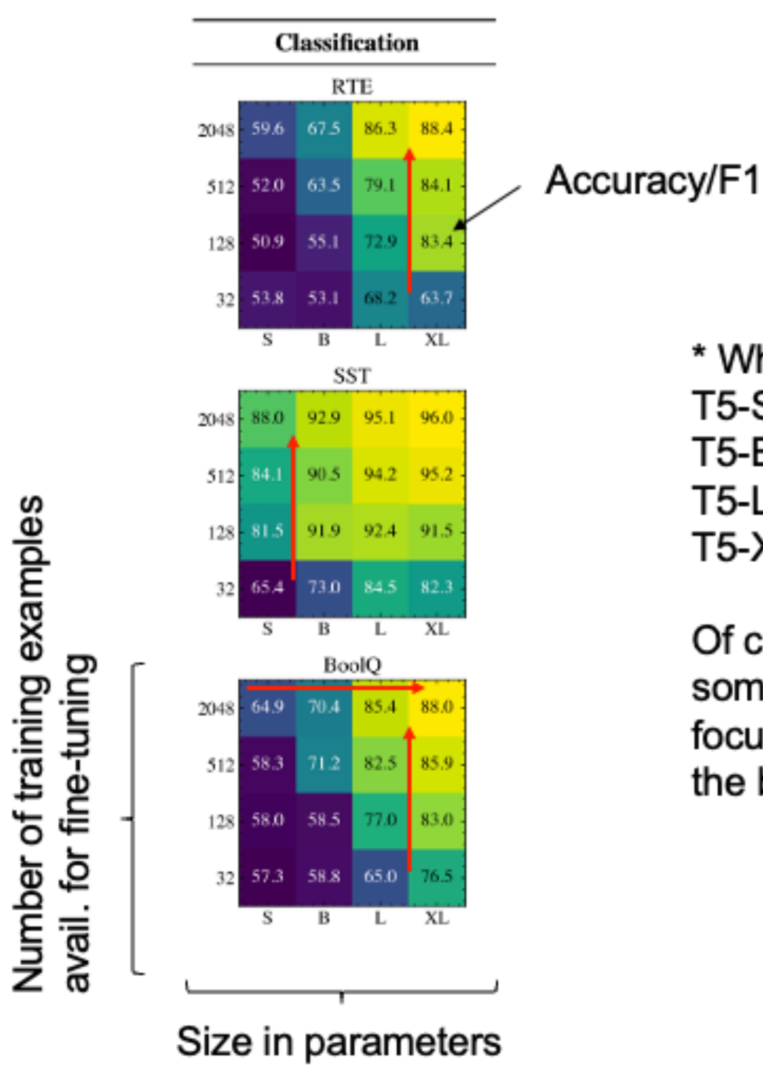
Computer Science > Computation and Language

[Submitted on 8 Oct 2021]

A Few More Examples May Be Worth Billions of Parameters

Yuval Kirstain, Patrick Lewis, Sebastian Riedel, Omer Levy

https://arxiv.org/abs/2110.04374



* Where:
T5-Small 60 million parameters
T5-Base 220 million parameters
T5-Large 770 million parameters
T5-XL 3300 million parameters

Of course, you probably have the strongest gradients somewhere along the diagonal, but still interesting to focus on training ex. vs model size for the best bang for the buck.

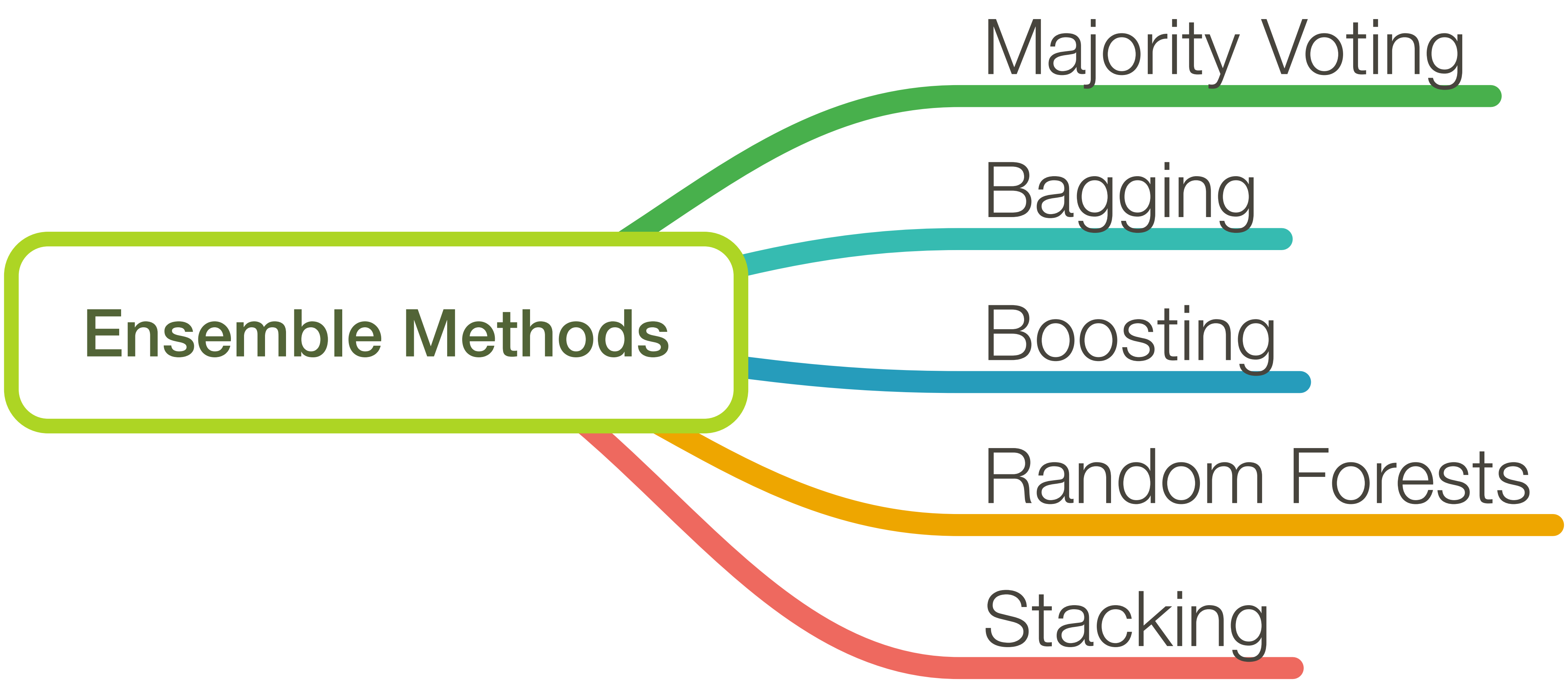
Lecture 07

Ensemble Methods Part 2/3

STAT 451: Machine Learning, Fall 2021

Sebastian Raschka

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking

7.1 Ensemble Methods -- Intro and Overview

7.2 Majority Voting

7.3 Bagging

7.4 Boosting

7.5 Gradient Boosting

7.6 Random Forests

7.7 Stacking

Gradient Boosting

Gradient Boosting

Gradient boosting is somewhat similar to AdaBoost:

- trees are fit sequentially to improve error of previous trees
- boost weak learners to a strong learner

The way how the trees are fit sequentially differs in AdaBoost and Gradient Boosting, though ...

Friedman, J. H. (1999). "Greedy Function Approximation: A Gradient Boosting Machine".

Gradient Boosting -- Conceptual Overview

- **Step 1:** Construct a base tree (just the root node).
- **Step 2:** Build next tree based on errors of the previous tree.
- **Step 3:** Combine tree from step 1 with trees from step 2. Go back to step 2.

Gradient Boosting -- Conceptual Overview
--> A Regression-based Example

In million US Dollars

x1# Rooms	x2=City	x3=Age	y=Price
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

- **<u>Step 1:</u>** Construct a base tree (just the root node)

$$\hat{y}_1 = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} = 0.5875$$

Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **<u>Step 2:</u>** Build next tree based on errors of the previous tree

First, compute (pseudo) residuals: $r_1 = y_1 - \hat{y}_1$

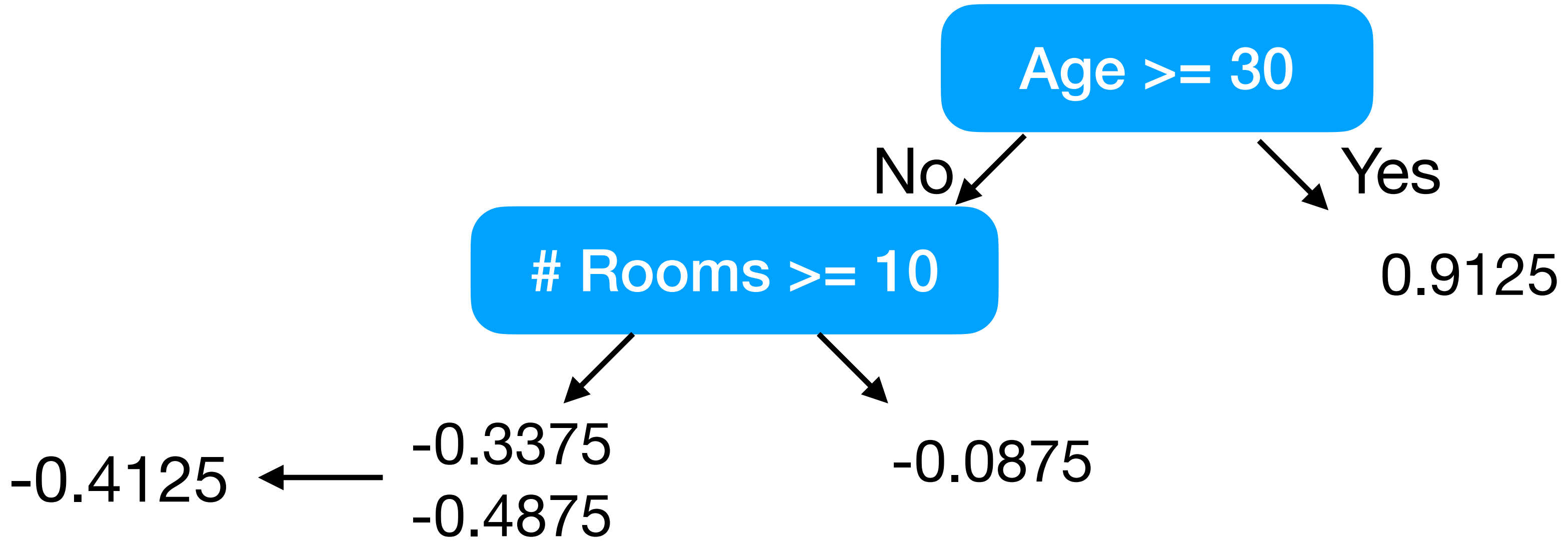
x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **<u>Step 2:</u>** Build next tree based on errors of the previous tree

That is, create a tree based on $x_1, \ldots, x_m$ **to fit the residuals**

x1#	x2=City	x3=Age	y=Price	r1=Residual
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

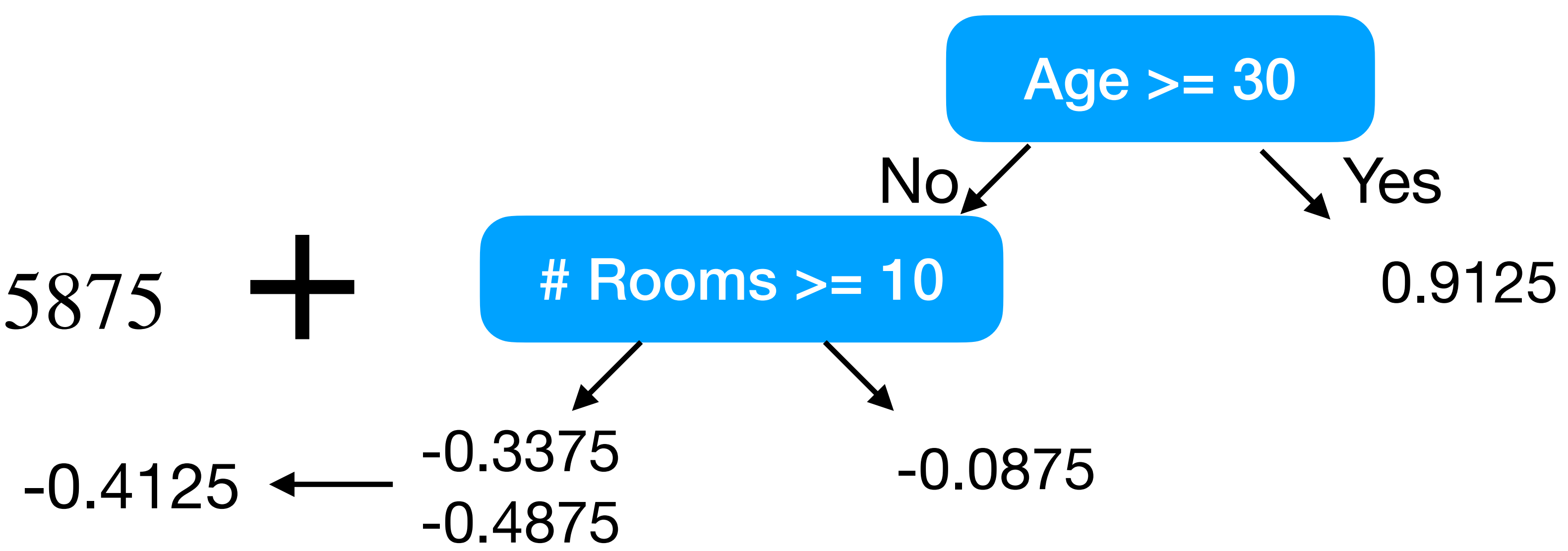


Gradient Boosting -- Conceptual Overview --> A Regression-based Example

- **<u>Step 3:</u>** Combine tree from step 1 with trees from step 2

x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunake	10	0.1	0.1 - 0.5875 = -0.4875

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875 \quad +$$



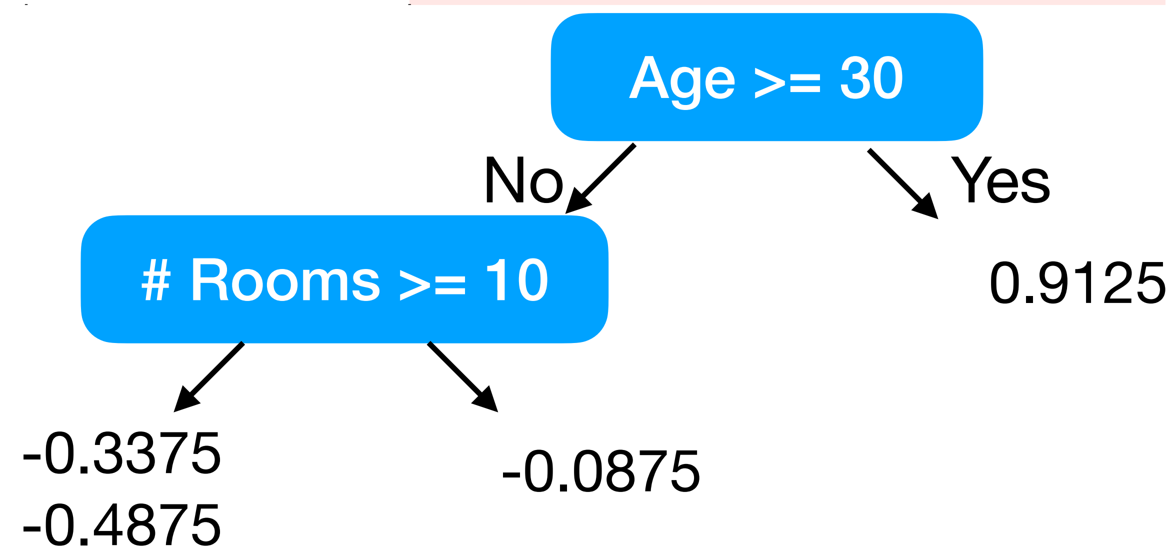
Gradient Boosting -- Conceptual Overview
--> A Regression-based Example

- **<u>Step 3:</u>** Combine tree from step 1 with trees from step 2

x1#	x2=City	x3=Age	y=Price	r1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunakee	10	0.1	0.1 - 0.5875 = -0.4875

E.g., predict Lansing →

$$\hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 0.5875 \quad +$$



-0.4125

E.g., predict Lansing $0.5875 + \alpha \times (-0.4125)$

where α learning rate between 0 and 1 (if $\alpha = 1$, low bias but high variance)

Gradient Boosting -- Algorithm Overview

Step 0: Input data $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{n}$

Differentiable Loss function $L\big(y^{(i)}, h(\mathbf{x}^{(i)})\big)$

Step 1: Initialize model $h_0(\mathbf{x}) = \operatorname*{argmin}_{\hat{y}} \sum_{i=1}^{n} L\big(y^{(i)}, \hat{y}\big)$

Step 2:

...

Gradient Boosting -- Algorithm Overview

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\big(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y}\big)$$

D. Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} \, \mathbb{I}\big(\mathbf{x} \in R_{j,t}\big)$

Step 3: Return $h_t(\mathbf{x})$

Gradient Boosting -- Algorithm Overview Discussion

Step 0: Input data $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{n}$

Differentiable Loss function $L\left(y^{(i)}, h(\mathbf{x}^{(i)})\right)$

E.g., Sum-squared error in regression

$$SSE' = \frac{1}{2}\left(y^{(i)} - h(\mathbf{x}^{(i)})\right)^2$$

$$\frac{\partial}{\partial h(\mathbf{x}^{(i)})} \frac{1}{2}\left(y^{(i)} - h(\mathbf{x}^{(i)})\right)^2$$ [chain rule]

$$= 2 \times \frac{1}{2}\left(y^{(i)} - h(\mathbf{x}^{(i)})\right) \times (0 - 1) = -\left(y^{(i)} - h(\mathbf{x}^{(i)})\right)$$

[neg. residual]

Gradient Boosting -- Algorithm Overview Discussion

Step 1: Initialize model $h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}\right)$

pred. target

turns out to be the average (in regression)

$$\frac{1}{n}\sum_{i=1}^{n} y^{(i)}$$

Gradient Boosting -- Algorithm Overview Discussion

Loop to make T trees (e.g., $T=100$)

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = - \left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})} \right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

pseudo residual of the t-th tree and i-th example

Derivative of the loss function

Gradient Boosting -- Algorithm Overview Discussion

Loop to make T trees (e.g., $T=100$)

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

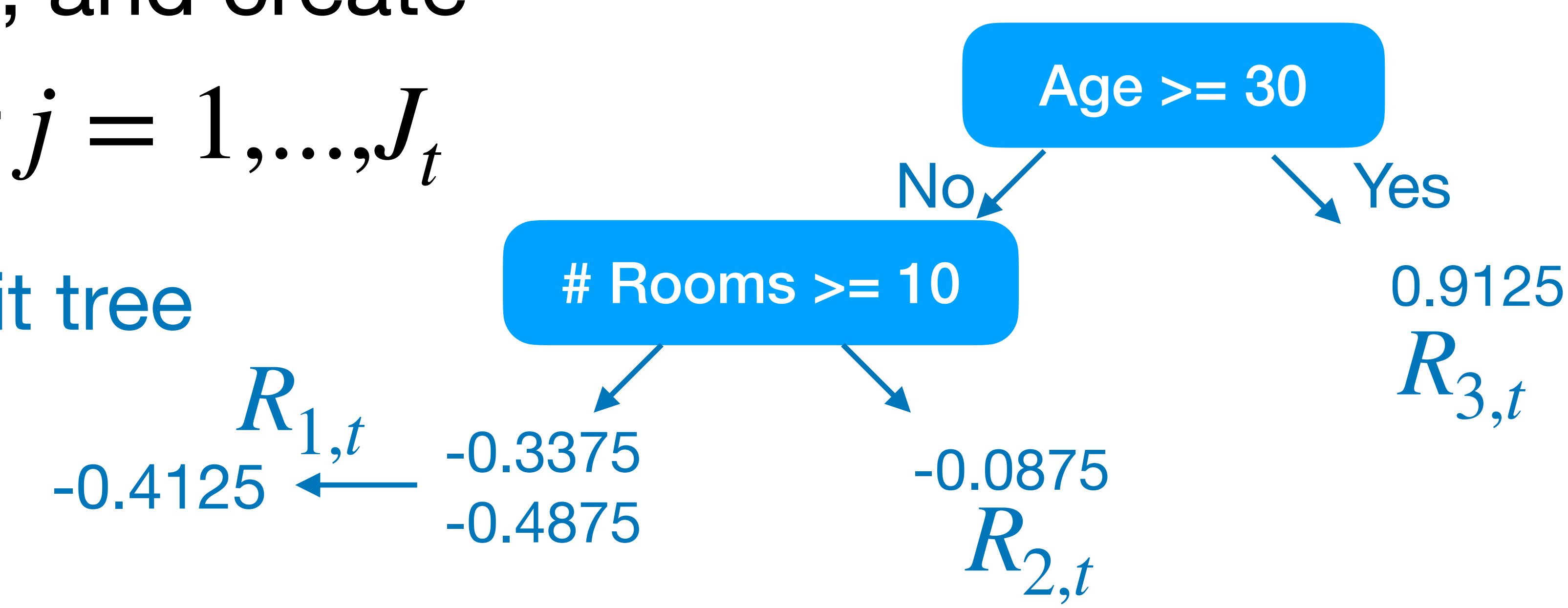
for $i = 1$ to n

pseudo residual of the t-th tree and i-th example

Derivative of the loss function

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

Use features in dataset to fit tree



Gradient Boosting -- Algorithm Overview Discussion

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = - \left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})} \right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

Compute the prediction for each leaf node

Only consider examples at that leaf node

Like step 1 but add previous prediction

Gradient Boosting -- Algorithm Overview Discussion

Step 2: for $t = 1$ to T

A. Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$

for $i = 1$ to n

B. Fit tree to $r_{i,t}$ values, and create terminal nodes $R_{j,t}$ for $j = 1,\ldots,J_t$

C. for $j = 1,\ldots,J_t$, compute

$$\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

D. Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} \, \mathbb{I}\left(\mathbf{x} \in R_{j,t}\right)$

learning rate between 0 and 1 (usually 0.1)

Summation just in case examples end up in multiple nodes

Gradient Boosting -- Algorithm Overview Discussion

For prediction, combine all T trees, e.g.,

$$h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}\right)$$

$$+\alpha\ \hat{y}_{j,t=1} \quad = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{(t=1)-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

...

$$+\alpha\ \hat{y}_{j,T} \quad = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L\left(y^{(i)}, h_{T-1}(\mathbf{x}^{(i)}) + \hat{y}\right)$$

Gradient Boosting -- Algorithm Overview Discussion

For prediction, combine all T trees, e.g.,

$$h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}\right)$$

$$+\alpha\ \hat{y}_{j,t=1}$$

...

$$+\alpha\ \hat{y}_{j,T}$$

The idea is that we decrease the pseudo residuals by a small amount at each step

XGBoost

learning system for tree boosting. The system is available as an open source package[2]. The impact of the system has been widely recognized in a number of machine learning and data mining challenges. Take the challenges hosted by the machine learning competition site Kaggle for example. Among the 29 challenge winning solutions [3] published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. For comparison, the second most popular method, deep neural nets, was used in 11 solutions. The success of the system was also witnessed in KDDCup 2015, where XGBoost was used by every winning team in the top-10. Moreover, the winning teams reported that ensemble

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.

XGBoost

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

Chen, T., & Guestrin, C. (2016, August). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

XGBoost

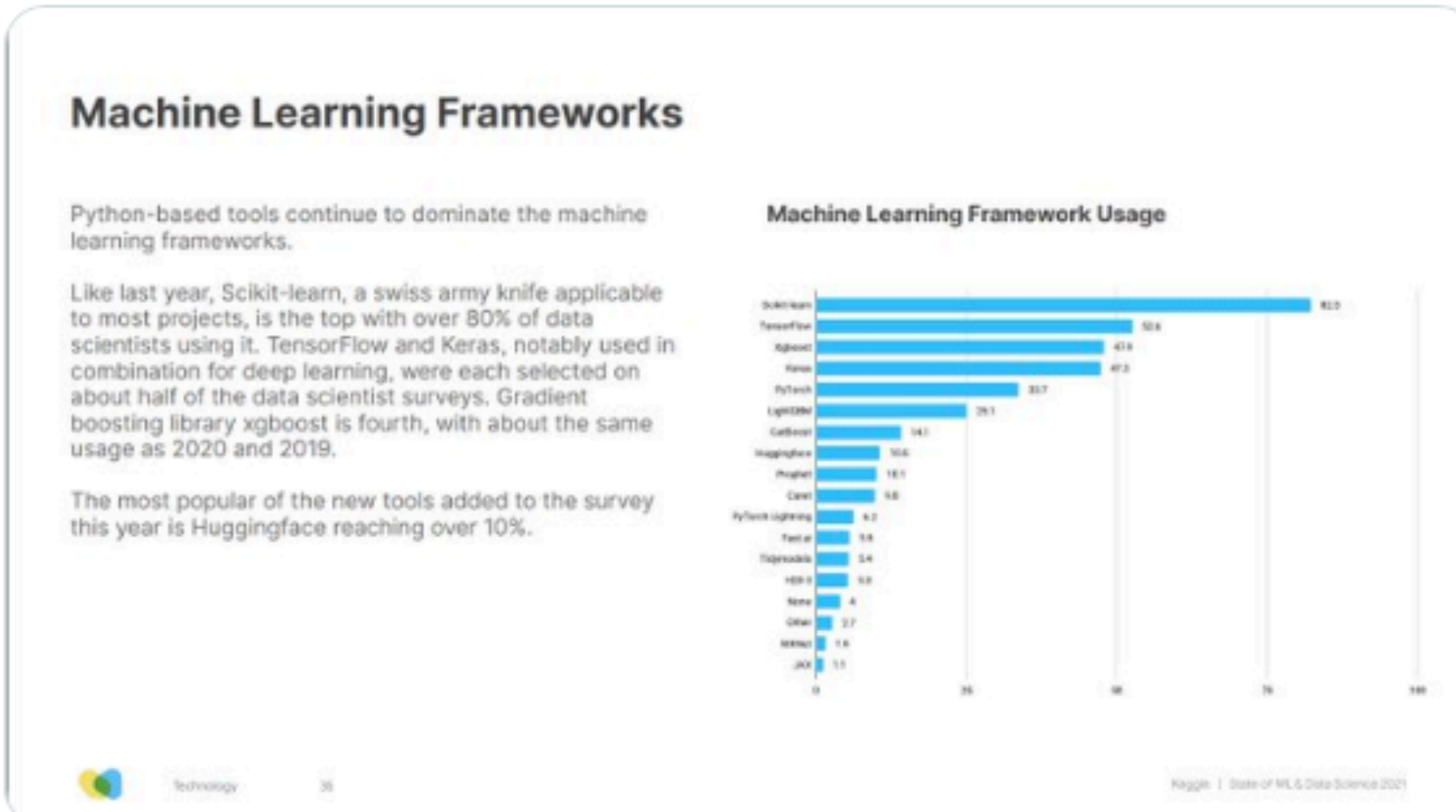
Summary and Main Points:

- scalable implementation of gradient boosting
- Improvements include: regularized loss, sparsity-aware algorithm, weighted quantile sketch for approximate tree learning, caching of access patterns, data compression, sharding
- Decision trees based on CART
- Regularization term for penalizing model (tree) complexity
- Uses second order approximation for optimizing the objective
- Options for column-based and row-based subsampling
- Single-machine version of XGBoost supports the exact greedy algorithm

Chen, T., & Guestrin, C. (2016, August). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM.

XGBoost
@XGBoostProject

@kaggle State of machine learning annual survey. XGboost, as usual, is among the top framework choices. Glad that we can help to make ml and data science better together with collection of other awesome frameworks kaggle.com/kaggle-survey-...



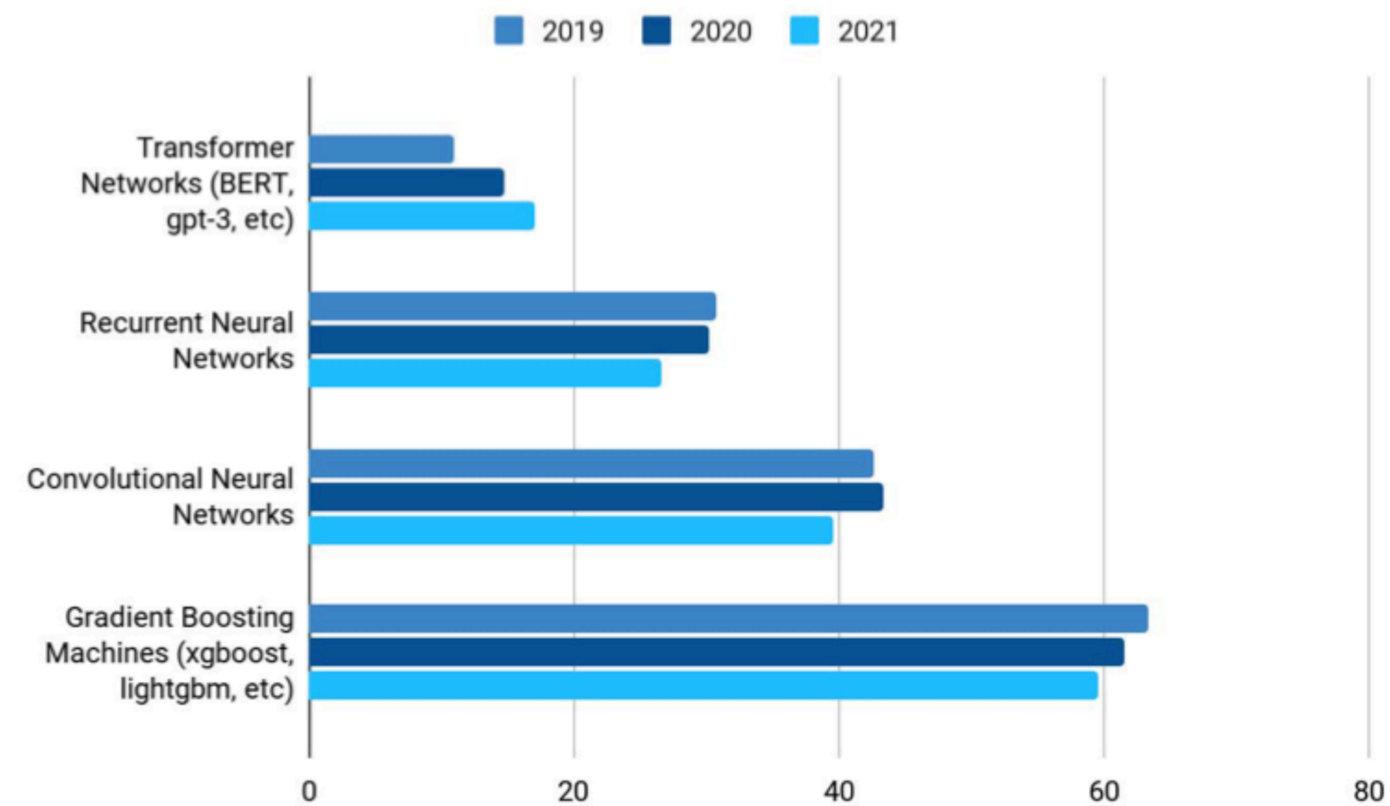
5:12 PM · Oct 15, 2021 · Twitter Web App

32 Retweets **9** Quote Tweets **194** Likes

Methods & Algorithms (cont.)

We also saw strong year-over-year growth in the use of large language models such as transformer networks (BERT, GPT-3, etc).

Popular ML Algorithms



Technology 34

Kaggle | State of ML & Data Science 2021

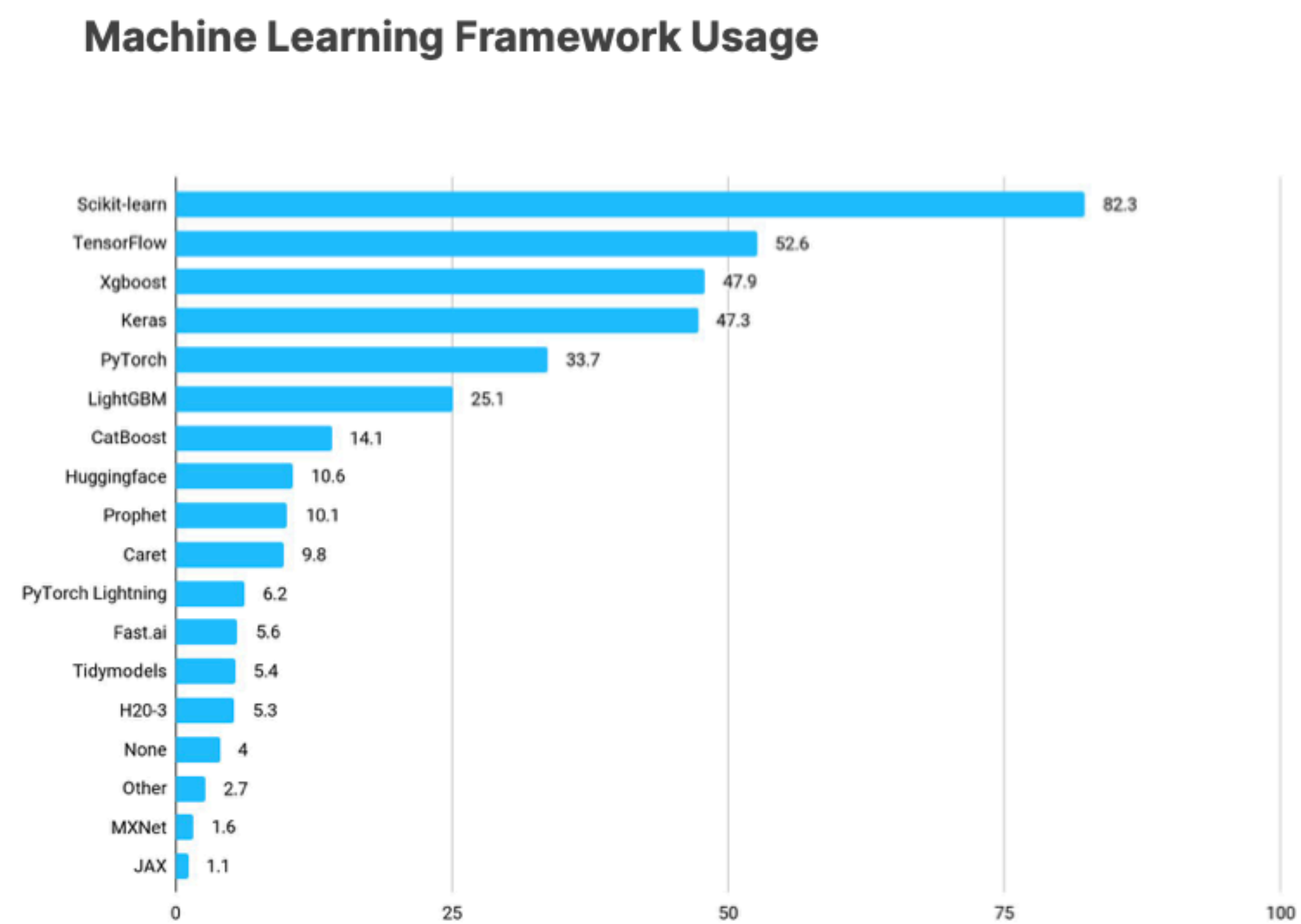
https://www.kaggle.com/kaggle-survey-2021

Machine Learning Frameworks

Python-based tools continue to dominate the machine learning frameworks.

Like last year, Scikit-learn, a swiss army knife applicable to most projects, is the top with over 80% of data scientists using it. TensorFlow and Keras, notably used in combination for deep learning, were each selected on about half of the data scientist surveys. Gradient boosting library xgboost is fourth, with about the same usage as 2020 and 2019.

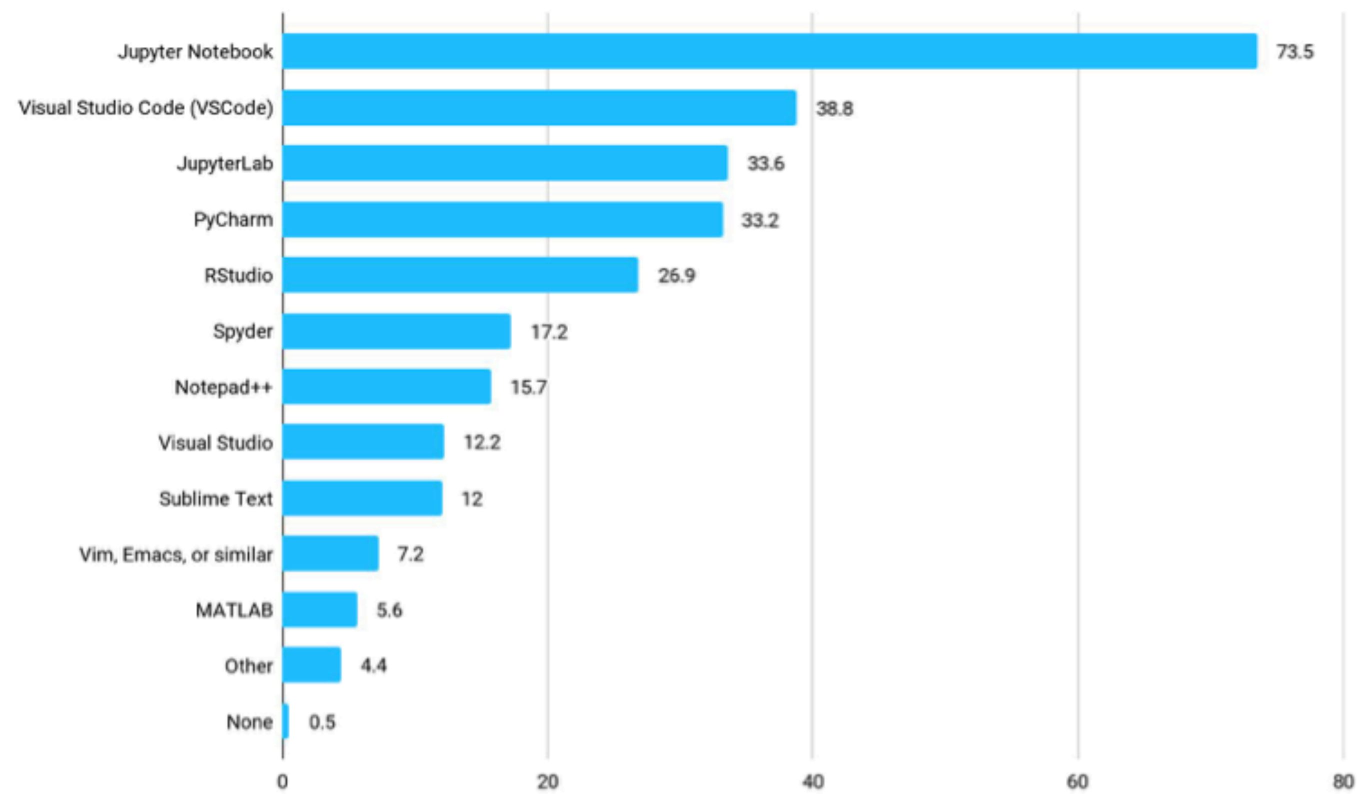
The most popular of the new tools added to the survey this year is Huggingface reaching over 10%.



Interactive Development Environments

Jupyter-based IDEs continue to be the go-to tool for data scientists, with around three-quarters of Kaggle data scientists using it. However, Visual Studio Code is in the second spot with 38%.

IDE Popularity



Technology 31

Kaggle | State of ML & Data Science 2021

LightGBM

Speed

We compared speed using only the training task without any test or metric output. We didn't count the time for IO. For the ranking tasks, since XGBoost and LightGBM implement different ranking objective functions, we used `regression` objective for speed benchmark, for the fair comparison.

The following table is the comparison of time cost:

Data	xgboost	xgboost_hist	LightGBM
Higgs	3794.34 s	165.575 s	**130.094 s**
Yahoo LTR	674.322 s	131.462 s	**76.229 s**
MS LTR	1251.27 s	98.386 s	**70.417 s**
Expo	1607.35 s	137.65 s	**62.607 s**
Allstate	2867.22 s	315.256 s	**148.231 s**

LightGBM ran faster than xgboost on all experiment data sets.

Accuracy

We computed all accuracy metrics only on the test data set.

Data	Metric	xgboost	xgboost_hist	LightGBM
Higgs	AUC	0.839593	0.845314	**0.845724**
Yahoo LTR	$NDCG_1$	0.719748	0.720049	**0.732981**
	$NDCG_3$	0.717813	0.722573	**0.735689**
	$NDCG_5$	0.737849	0.740899	**0.75352**
	$NDCG_{10}$	0.78089	0.782957	**0.793498**
MS LTR	$NDCG_1$	0.483956	0.485115	**0.517767**
	$NDCG_3$	0.467951	0.47313	**0.501063**
	$NDCG_5$	0.472476	0.476375	**0.504648**
	$NDCG_{10}$	0.492429	0.496553	**0.524252**
Expo	AUC	0.756713	0.776224	**0.776935**
Allstate	AUC	0.607201	**0.609465**	0.609072

https://lightgbm.readthedocs.io/en/latest/Experiments.html

2.2 Histogram-based Split Finding

Both XGBoost and LightGBM support **histogram-based** algorithm for split finding. As mentioned in XGBoost paper, the exact-greedy (brute-force) split find algorithm is time consuming: for current feature to search, need to sort feature values and iterate through. For faster training, histogram-based algorithm is used, which bucket continuous feature into discrete bins. This speeds up training and reduces memory usage.

LightGBM is using histogram-based algorithm. Related parameters are:

- `max_bin` : max number of bins that feature values will be bucketed in.
- `min_data_in_bin` : minimal number of data inside one bin.
- `bin_construct_sample_cnt` : number of data that sampled to construct histogram bins.

XGBoost has options to choose histogram-based algorithm, it is specified by `tree_method` with options:

- `auto` : (default) use heuristic to choose the fastest method.
- `exact` : exact greedy algorithm.
- `approx` : approximate greedy algorithm using quantile sketch and gradient histogram.
- `hist` : fast histogram optimized approximate greedy algorithm, with this option enabled, `max_bin` (default 256) could be tuned

https://bangdasun.github.io/2019/03/21/38-practical-comparison-xgboost-lightgbm/

More GBM Implementations

LightGBM, Light Gradient Boosting Machine

From https://github.com/Microsoft/LightGBM:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Support of parallel and GPU learning
- Capable of handling large-scale data

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems* (pp. 3146-3154).

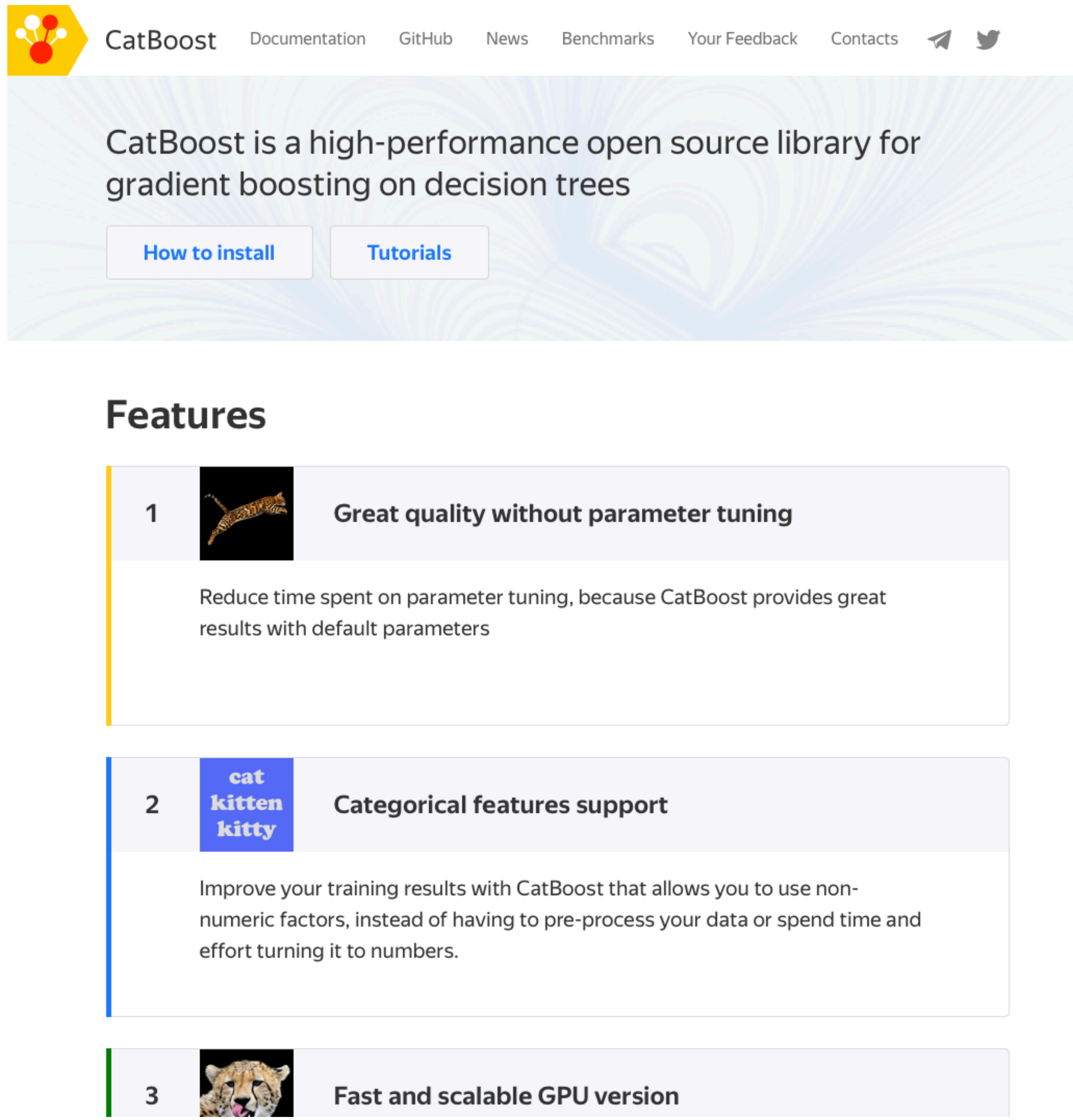
https://scikit-learn.org/stable/whats_new.html#version-0-21-0

sklearn.ensemble ¶

- **Major Feature** Add two new implementations of gradient boosting trees: `ensemble.HistGradientBoostingClassifier` and `ensemble.HistGradientBoostingRegressor`. The implementation of these estimators is inspired by LightGBM and can be orders of magnitude faster than `ensemble.GradientBoostingRegressor` and `ensemble.GradientBoostingClassifier` when the number of samples is larger than tens of thousands of samples. The API of these new estimators is slightly different, and some of the features from `ensemble.GradientBoostingClassifier` and `ensemble.GradientBoostingRegressor` are not yet supported.

CatBoost

https://catboost.ai


CatBoost Documentation GitHub News Benchmarks Your Feedback Contacts

CatBoost is a high-performance open source library for gradient boosting on decision trees

How to install Tutorials

Features

1 Great quality without parameter tuning

Reduce time spent on parameter tuning, because CatBoost provides great results with default parameters

2 cat kitten kitty Categorical features support

Improve your training results with CatBoost that allows you to use non-numeric factors, instead of having to pre-process your data or spend time and effort turning it to numbers.

3 Fast and scalable GPU version

Code 1

```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn import datasets


data = datasets.load_breast_cancer()
X, y = data.data, data.target

X_temp, X_test, y_temp, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=123, stratify=y)

X_train, X_valid, y_train, y_valid = \
    train_test_split(X_temp, y_temp, test_size=0.2, random_state=123, stratify=y_temp)

print('Train/Valid/Test sizes:', y_train.shape[0], y_valid.shape[0], y_test.shape[0])
```

```
Train/Valid/Test sizes: 318 80 171
```

Original gradient boosting

```
from sklearn.ensemble import GradientBoostingClassifier


boost = GradientBoostingClassifier(
    learning_rate=0.1,
    n_estimators=100,
    max_depth=8,
    random_state=1)

boost.fit(X_train, y_train)


print("Training Accuracy: %0.2f" % boost.score(X_train, y_train))
print("Validation Accuracy: %0.2f" % boost.score(X_valid, y_valid))
print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Training Accuracy: 1.00
Validation Accuracy: 0.90
Test Accuracy: 0.92
```

Code 2

HistGradientBoostingClassifier (inspired by LightGBM)

```
#from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier


boost = HistGradientBoostingClassifier(
    learning_rate=0.1,
    #n_estimators=100,
    #max_depth=8,
    random_state=1)

boost.fit(X_train, y_train)

print("Training Accuracy: %0.2f" % boost.score(X_train, y_train))
print("Validation Accuracy: %0.2f" % boost.score(X_valid, y_valid))
print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Training Accuracy: 1.00
Validation Accuracy: 0.96
Test Accuracy: 0.97
```

Code 3

XGBoost

```
# https://xgboost.readthedocs.io/en/latest/build.html
```

```
#!pip install xgboost
```

```
import numpy as np
import xgboost as xgb


boost = xgb.XGBClassifier()

boost.fit(X_train, y_train)

print("Training Accuracy: %0.2f" % boost.score(X_train, y_train))
print("Validation Accuracy: %0.2f" % boost.score(X_valid, y_valid))
print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
[07:41:34] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default eval
om 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old be
Training Accuracy: 1.00
Validation Accuracy: 0.95
Test Accuracy: 0.98
```

Code 4

LightGBM

```
# https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html
# conda install -c conda-forge lightgbm
```

```
import lightgbm as lgb


boost = lgb.LGBMClassifier()

boost.fit(X_train, y_train)


print("Training Accuracy: %0.2f" % boost.score(X_train, y_train))
print("Validation Accuracy: %0.2f" % boost.score(X_valid, y_valid))
print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Training Accuracy: 1.00
Validation Accuracy: 0.96
Test Accuracy: 0.98
```

Code 5

CatBoost

```
# https://catboost.ai
# conda install -c conda-forge catboost
```

```
from catboost import CatBoostClassifier


boost = CatBoostClassifier(verbose=0)

boost.fit(X_train, y_train)


print("Training Accuracy: %0.2f" % boost.score(X_train, y_train))
print("Validation Accuracy: %0.2f" % boost.score(X_valid, y_valid))
print("Test Accuracy: %0.2f" % boost.score(X_test, y_test))
```

```
Training Accuracy: 1.00
Validation Accuracy: 0.97
Test Accuracy: 0.98
```

Show Jupyter Notebook with Categorical Examples

7.1 Ensemble Methods -- Intro and Overview

7.2 Majority Voting

7.3 Bagging

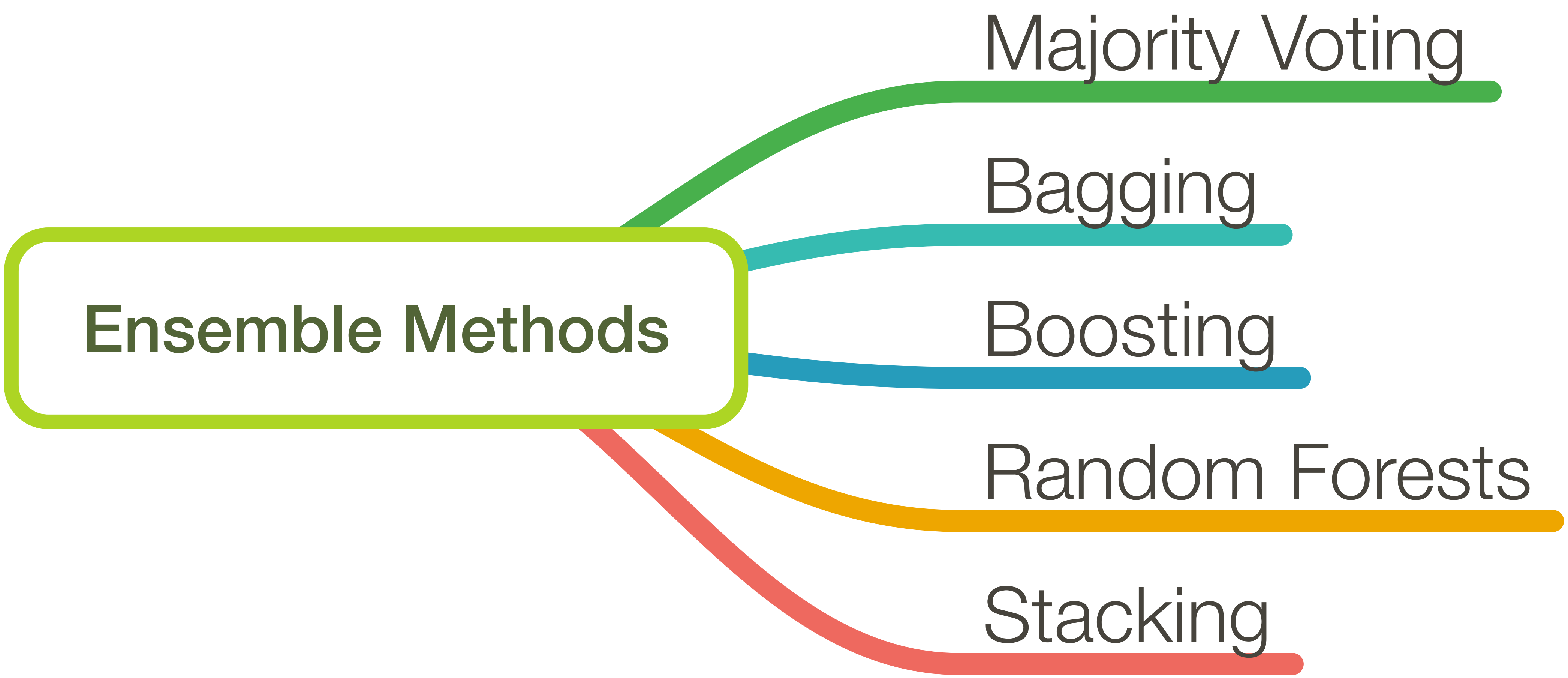
7.4 Boosting

7.5 Gradient Boosting

7.6 Random Forests

7.7 Stacking

Overview


Ensemble Methods
Majority Voting
Bagging
Boosting
Random Forests
Stacking