Lecture 24

LightGBM

COURSE MATH 637: Mathematical Techniques in Data Science

WEBSITE https://jacobcalvert.com/teaching/math637fall2026/

INSTRUCTOR Jacob Calvert (calvert@udel.edu)

OFFICE HOURS MW 2:45pm–3:45pm, Ewing Hall 509

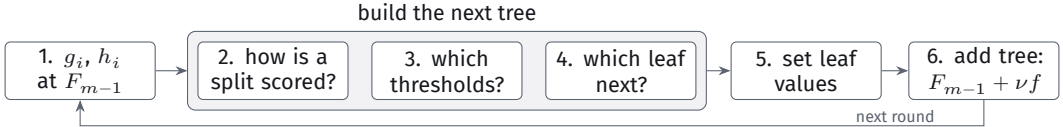
Announcements & Reminders

- HW5 is a 1-page project proposal, due by class on Nov. 11th
- Canvas has details about what aspects to address
- If we have time left today, we can discuss further

Plan for today

Recap	3 min
Searching thresholds via histograms	12 min
Handling missing values	3 min
LightGBM	16 min
Quick comparison	8 min
Project discussion	the rest

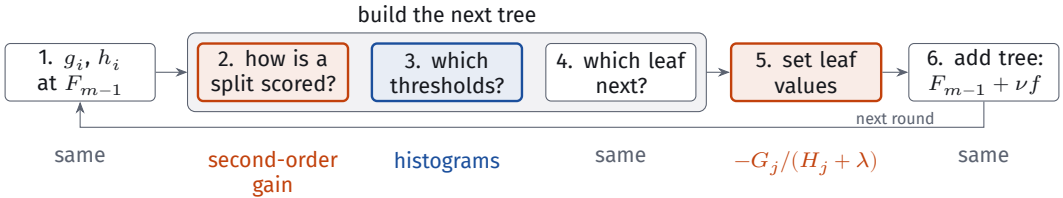
Recall: One round of gradient boosting



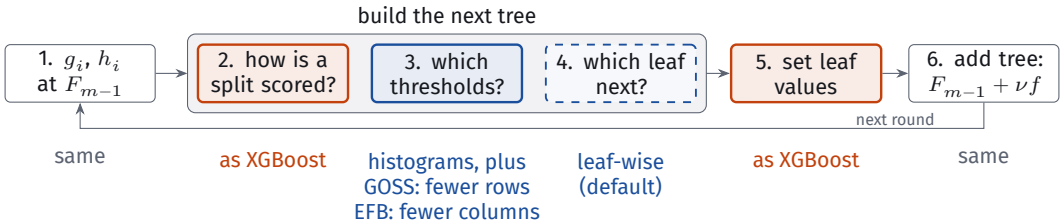
The answers to 2–5 differentiate regular gradient boosting from XGBoost and LightGBM. For regular gradient boosting the answers are:

2. the decrease in the RSS of the targets $-g_i$
3. every distinct value of each feature
4. every leaf at one depth, down to a fixed depth
5. $w_j = -G_j/H_j$

Recall: What XGBoost changes



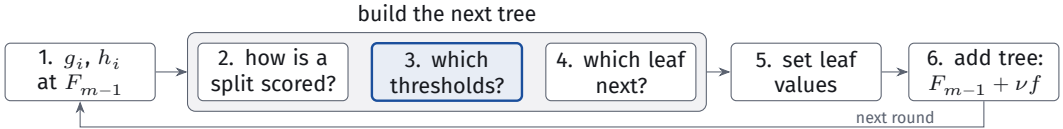
Ahead: What LightGBM changes



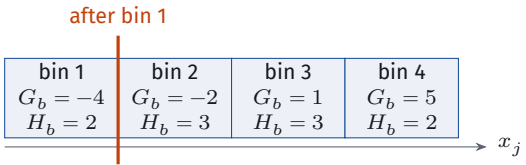
Plan for today

• Recap	3 min
• Searching thresholds via histograms	12 min
• Handling missing values	3 min
• LightGBM	16 min
• Quick comparison	8 min
• Project discussion	the rest

A faster way to search thresholds: histograms



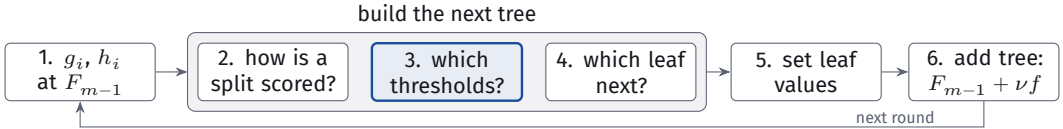
- bin each feature once; in each leaf, add each (g_i, h_i) into its bin
- move the cut one bin right: add that bin to the left; right = total − left



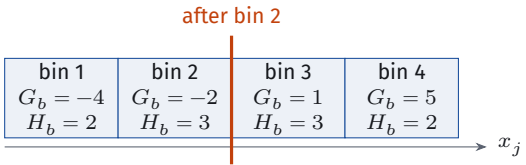
cut	G_L	H_L	G_R	H_R	gain
after 1	-4	2	4	8	5

$\lambda = 0$; the parent has $G = 0$

A faster way to search thresholds: histograms



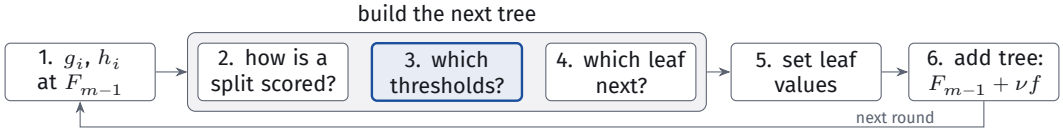
- bin each feature once; in each leaf, add each (g_i, h_i) into its bin
- move the cut one bin right: add that bin to the left; right $=$ total $-$ left



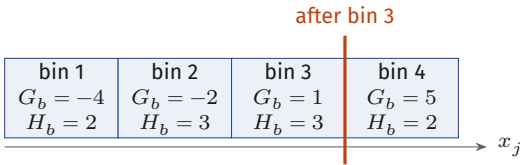
cut	G_L	H_L	G_R	H_R	gain
after 1	-4	2	4	8	5
after 2	-6	5	6	5	7.2

$\lambda = 0$; the parent has $G = 0$

A faster way to search thresholds: histograms



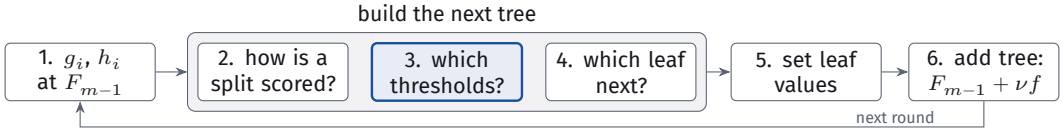
- bin each feature once; in each leaf, add each (g_i, h_i) into its bin
- move the cut one bin right: add that bin to the left; right $=$ total $-$ left



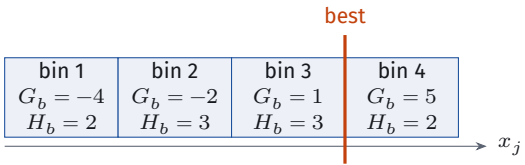
cut	G_L	H_L	G_R	H_R	gain
after 1	-4	2	4	8	5
after 2	-6	5	6	5	7.2
after 3	-5	8	5	2	7.8

$\lambda = 0$; the parent has $G = 0$

A faster way to search thresholds: histograms



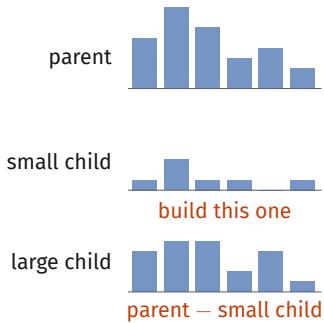
- bin each feature once; in each leaf, add each (g_i, h_i) into its bin
- move the cut one bin right: add that bin to the left; right $=$ total $-$ left



cut	G_L	H_L	G_R	H_R	gain
after 1	-4	2	4	8	5
after 2	-6	5	6	5	7.2
after 3	-5	8	5	2	7.8

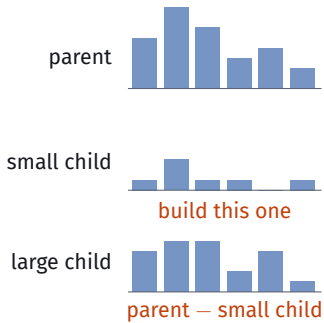
$\lambda = 0$; the parent has $G = 0$

Why histograms are fast



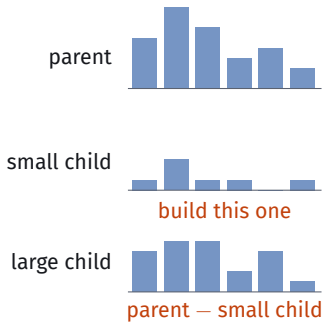
- instead of checking every distinct feature value, check the edges of its bins

Why histograms are fast



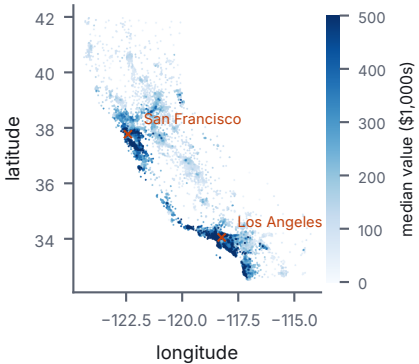
- instead of checking every distinct feature value, check the edges of its bins
- after a split, you only need to build the smaller “child’s” histograms for each feature, because you can get the larger child’s histograms by subtraction

Why histograms are fast



- instead of checking every distinct feature value, check the edges of its bins
- after a split, you only need to build the smaller “child’s” histograms for each feature, because you can get the larger child’s histograms by subtraction
- the bins for each feature are chosen once, before boosting starts, and reused in every leaf of every tree

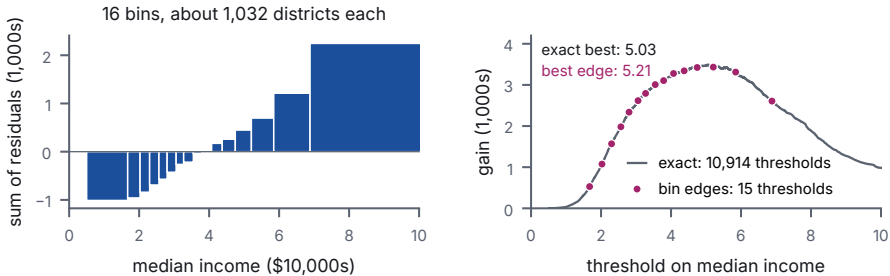
Example: California housing data



- Consists of $20{,}640$ districts from 1990 census. Task is to predict each district's median house value, capped at $\$500{,}001$.
- Features are: location of district, median house age, total rooms, total bedrooms, population, # of households, median income, and ocean proximity (5 categories)
- Use squared loss for training objective. Report error as RMSE in dollars.

Example: Histograms for the CA data

The first split on median income with 16 bins of equal counts.
Left: $-G_b$. Right: gain at every bin edge.

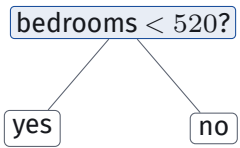


Best bin edge gives 98.4% of best gain over *all* thresholds. With 256 bins, 99.9%.
For 500 trees of depth 6, an exact search takes 3.5 s; with histograms, only 0.6 s.

Plan for today

• Recap	3 min
• Searching thresholds via histograms	12 min
• Handling missing values	3 min
• LightGBM	16 min
• Quick comparison	8 min
• Project discussion	the rest

For missing values, try each side of split

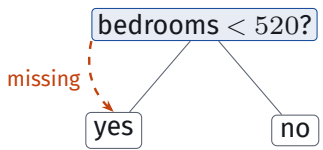


missing: left or right?

Example: 207 of the $20{,}640$ districts have no value for the total number of bedrooms.

- at each split, try sending all points with no value of the feature left, then right, and keep the larger gain

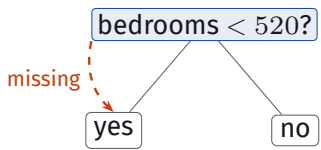
For missing values, try each side of split



Example: 207 of the $20{,}640$ districts have no value for the total number of bedrooms.

- at each split, try sending all points with no value of the feature left, then right, and keep the larger gain
- the direction is learned from the training data, split by split

For missing values, try each side of split



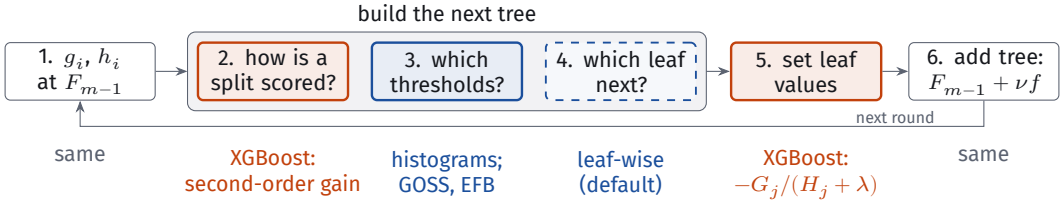
Example: 207 of the $20{,}640$ districts have no value for the total number of bedrooms.

- at each split, try sending all points with no value of the feature left, then right, and keep the larger gain
- the direction is learned from the training data, split by split
- XGBoost and LightGBM both do this, while logistic regression requires imputation

Plan for today

• Recap	3 min
• Searching thresholds via histograms	12 min
• Handling missing values	3 min
• LightGBM	16 min
• Quick comparison	8 min
• Project discussion	the rest

LightGBM



Definition (LightGBM (Ke et al., 2017))

Another fast implementation of gradient-boosted trees, with XGBoost's gain and leaf values, but faster threshold search and leaf-wise growth.

GOSS = gradient-based one-side sampling
EFB = exclusive feature bundling

Key idea: build and search histograms faster

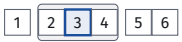


box 3: which thresholds?



- a leaf's histograms: each point adds (g_i, h_i) to a bin of every column, about $n \times p$ additions; scanning the bins after is cheap

Key idea: build and search histograms faster



box 3: which thresholds?



- a leaf's histograms: each point adds (g_i, h_i) to a bin of every column, about $n \times p$ additions; scanning the bins after is cheap
- GOSS shrinks the height n, EFB shrinks the width p

Fewer rows: GOSS

Gradient-based one-side sampling: grow each tree from fewer points.



- keep the top 20% of $|g_i|$, i.e., the examples you predict the worst

Fewer rows: GOSS

Gradient-based one-side sampling: grow each tree from fewer points.



- keep the top 20% of $|g_i|$, i.e., the examples you predict the worst
- randomly sample 10% of other examples, scale their g_i and h_i by 8

Fewer rows: GOSS

Gradient-based one-side sampling: grow each tree from fewer points.



- keep the top 20% of $|g_i|$, i.e., the examples you predict the worst
- randomly sample 10% of other examples, scale their g_i and h_i by 8
- do this once per tree, before making feature histograms

Fewer rows: GOSS

Gradient-based one-side sampling: grow each tree from fewer points.



- keep the top 20% of $|g_i|$, i.e., the examples you predict the worst
- randomly sample 10% of other examples, scale their g_i and h_i by 8
- do this once per tree, before making feature histograms
- use `data_sample_strategy="goss"` in LightGBM

Fewer columns: EFB

Exclusive feature bundling: features that are rarely nonzero in the same row share one histogram. Example: CA ocean proximity.



- each point adds into one bin of one histogram, not into five

Fewer columns: EFB

Exclusive feature bundling: features that are rarely nonzero in the same row share one histogram. Example: CA ocean proximity.



- each point adds into one bin of one histogram, not into five
- splits stay on the original columns, e.g., INLAND $= 1$ or not

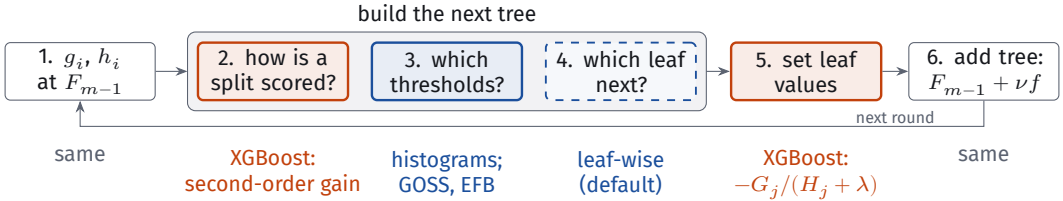
Fewer columns: EFB

Exclusive feature bundling: features that are rarely nonzero in the same row share one histogram. Example: CA ocean proximity.



- each point adds into one bin of one histogram, not into five
- splits stay on the original columns, e.g., INLAND $= 1$ or not
- done once, when data are loaded; worth it for 1000s of sparse columns

Recall: LightGBM

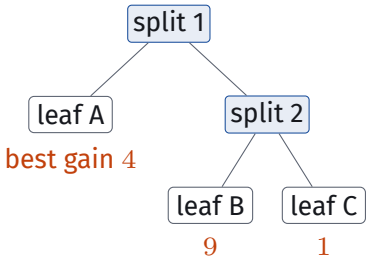


Definition (LightGBM (Ke et al., 2017))

Another fast implementation of gradient-boosted trees, with XGBoost's gain and leaf values, but faster threshold search and leaf-wise growth.

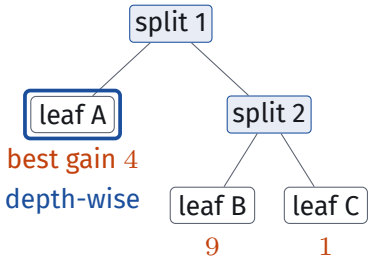
GOSS = gradient-based one-side sampling
EFB = exclusive feature bundling

Which leaf to split next?



Each leaf shows the gain of its best split.
One more split: which leaf?

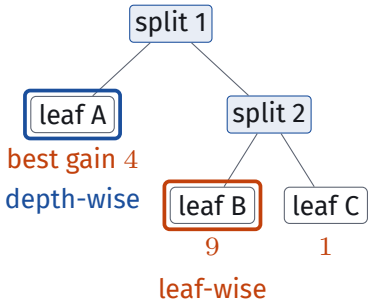
Which leaf to split next?



Each leaf shows the gain of its best split.
One more split: which leaf?

- depth-wise (XGBoost default) splits leaf A next

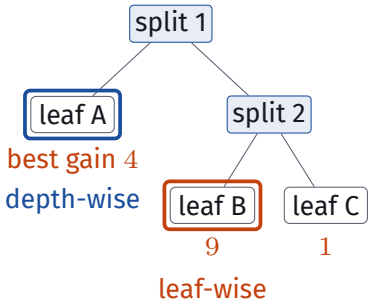
Which leaf to split next?



Each leaf shows the gain of its best split.
One more split: which leaf?

- depth-wise (XGBoost default) splits leaf A next
- leaf-wise (LightGBM) splits leaf B next beause it has the largest gain

Which leaf to split next?



Each leaf shows the gain of its best split. One more split: which leaf?

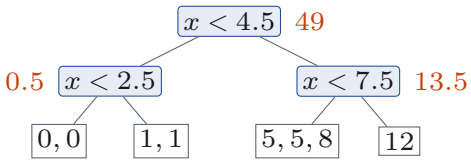
- depth-wise (XGBoost default) splits leaf A next
- leaf-wise (LightGBM) splits leaf B next beause it has the largest gain
- leaf-wise growth can produce lopsided trees

Example: depth-wise vs. leaf-wise

$x = 1, \dots, 8$, $y = (0, 0, 1, 1, 5, 5, 8, 12)$, split on squared loss until four leaves. Splits show the gains, leaves show the y values.

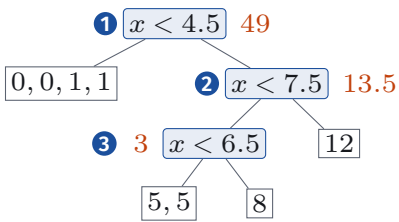
depth-wise, to depth 2

gradient boosting, XGBoost; RSS 6



leaf-wise, to 4 leaves

LightGBM, `lossguide`; RSS 1

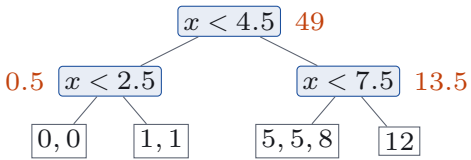


Example: depth-wise vs. leaf-wise

$x = 1, \ldots, 8$, $y = (0, 0, 1, 1, 5, 5, 8, 12)$, split on squared loss until four leaves. Splits show the gains, leaves show the y values.

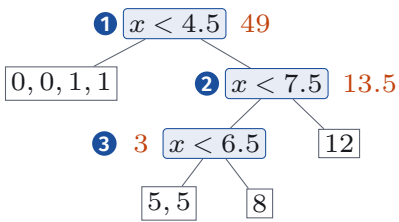
depth-wise, to depth 2

gradient boosting, XGBoost; RSS 6



leaf-wise, to 4 leaves

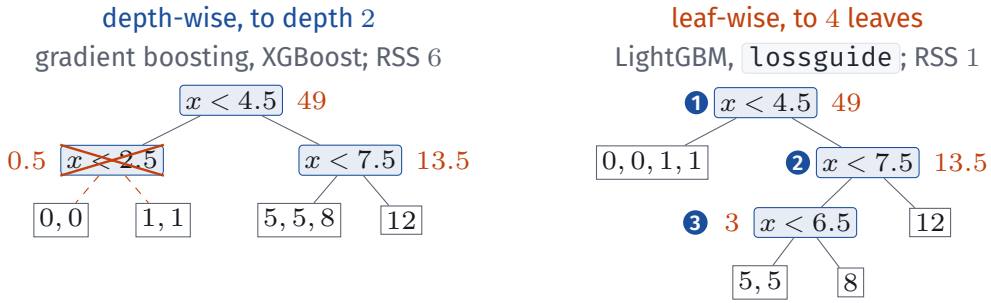
LightGBM, `lossguide`; RSS 1



- leaf-wise: after split 1, the best gains are 0.5 and 13.5, so split 2 is on the right; then 0.5 against 3, so split 3 is on the right too

Example: depth-wise vs. leaf-wise

$x = 1, \dots, 8$, $y = (0, 0, 1, 1, 5, 5, 8, 12)$, split on squared loss until four leaves. Splits show the gains, leaves show the y values.



- leaf-wise: after split 1, the best gains are 0.5 and 13.5, so split 2 is on the right; then 0.5 against 3, so split 3 is on the right too
- XGBoost with $\gamma = 1$ prunes leaf with small gain

Why might leaf-wise growth overfit?

Question

Leaf-wise growth always makes the split that lowers the training loss most. Why might that hurt on new data?

Why might leaf-wise growth overfit?

Question

Leaf-wise growth always makes the split that lowers the training loss most. Why might that hurt on new data?

- it might grow a deep branch on a few points while the rest of the tree stays shallow

Why might leaf-wise growth overfit?

Question

Leaf-wise growth always makes the split that lowers the training loss most. Why might that hurt on new data?

- it might grow a deep branch on a few points while the rest of the tree stays shallow
- limit it via `num_leaves`, `max_depth`, and the fewest points in a leaf, `min_data_in_leaf` (default 20)

Why might leaf-wise growth overfit?

Question

Leaf-wise growth always makes the split that lowers the training loss most. Why might that hurt on new data?

- it might grow a deep branch on a few points while the rest of the tree stays shallow
- limit it via `num_leaves`, `max_depth`, and the fewest points in a leaf, `min_data_in_leaf` (default 20)

Why might leaf-wise growth overfit?

Question

Leaf-wise growth always makes the split that lowers the training loss most. Why might that hurt on new data?

- it might grow a deep branch on a few points while the rest of the tree stays shallow
- limit it via `num_leaves`, `max_depth`, and the fewest points in a leaf, `min_data_in_leaf` (default 20)

California, `num_leaves`	4	8	16	31	63	127
validation RMSE ($1,000s)	46.7	46.2	46.0	46.1	46.7	47.1

Plan for today

• Recap	3 min
• Searching thresholds via histograms	12 min
• Handling missing values	3 min
• LightGBM	16 min
• Quick comparison	8 min
• Project discussion	the rest

XGBoost vs. LightGBM

	XGBoost	LightGBM
objective	gain; $-G/(H+\lambda)$	the same
thresholds	histograms, or exact	histograms
growth	depth-wise; leaf-wise option	leaf-wise
fewer points	random, or gradient-based	random, or GOSS
sparse features	sparsity-aware search	EFB
missing values	a learned direction	the same

Key idea: Two implementations of one method, with same gains and leaf values. LightGBM uses GOSS and EFB for histograms, leaf-wise growth.

Python: XGBoost and LightGBM

```
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

xg = XGBRegressor(n_estimators=1414, max_depth=4,
                  learning_rate=0.1)        # depth-wise
lg = LGBMRegressor(n_estimators=871, num_leaves=16,
                   learning_rate=0.1)       # leaf-wise
xg.fit(X_train, y_train)
lg.fit(X_train, y_train)
```

Example: test on California housing data

Test on $4{,}128$ districts, after validation and refitting.

	test RMSE
linear regression (ocean proximity one-hot)	$\$68{,}700$
one tree (leaf size ≥ 20)	$\$56{,}000$
random forest (300 trees)	$\$47{,}700$
XGBoost, depth-wise, depth 4, $M = 1{,}414$	$\$44{,}200$
LightGBM, leaf-wise, 16 leaves, $M = 871$	$\$43{,}800$

Summary

- XGBoost: one penalized second-order objective scores the splits, by the gain $S_L + S_R - S - \gamma$, and sets the leaf values $-G/(H + \lambda)$
- scoring a split, choosing which thresholds to try, and choosing which leaf to split next are separate choices; histograms make the search fast for both XGBoost and LightGBM
- LightGBM has the same objective but uses GOSS (fewer rows) and EFB (fewer columns) to make histograms faster on very large data; grown leaf-wise by default

Plan for today

• Recap	3 min
• Searching thresholds via histograms	12 min
• Handling missing values	3 min
• LightGBM	16 min
• Quick comparison	8 min
• Project discussion	the rest

Final projects

- HW5 is a 1-page final project proposal, due by class next Wed. (Nov. 11th)
- For full credit, describe project idea in enough detail that I can reasonably evaluate its suitability. In particular, address:
  1. Will you work alone, or with a small team? Name team members, explain division of work. More will be expected of larger groups.
  2. What question(s) will you try to answer?
  3. What dataset(s) will you use? Provide specifications, e.g., # examples, features and their types, data missingness, etc.
  4. What simple baseline technique(s) will you use, e.g., for data exploration and prediction/inference?
  5. What ideas from the course/literature will you use beyond these baselines? (Note: we'll cover neural net–based methods soon.)
  6. What is the ideal outcome? The main piftall?