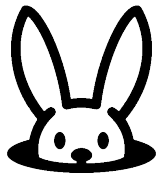
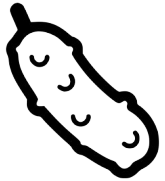
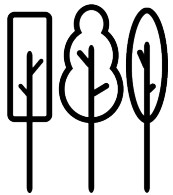
shrinking gigabyte-sized scikit-learn models for deployment

pavel zwerschke and yasin tatar

a story about

a rabbit hole 

a pickle 

and slim trees 

trees are awesome …

- Large-scale machine learning
- Economic applications
- (also: open source)

trees are awesome …

quantco

- Large-scale machine learning
- Economic applications
- (also: open source)

- Random Forests
- Trees
- Gradient Boosted Trees

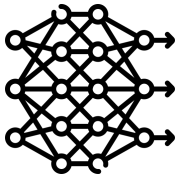
… but get large very fast.

Multiple models combined with

- Hundreds of features
- Thousand trees
- Millions of Leaf-Nodes

A journey through making one such model smaller…

what are our options?

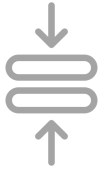


Use other models

Loss of performance, change in interpretability, …

Transpile to other formats

Large engineering effort with compatibility drawbacks



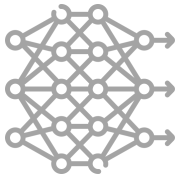
Compress with zstd/gzip

Didn’t bring the required performance

… go down the rabbit hole?

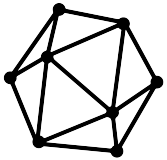
Let’s have a look at the internals!

what are our options?



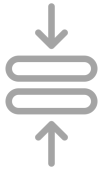
Use other models

Loss of performance, change in interpretability, …



Transpile to other formats

Large engineering effort with compatibility drawbacks



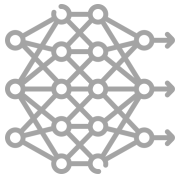
Compress with zstd/gzip

Didn’t bring the required performance

… go down the rabbit hole?

Let’s have a look at the internals!

what are our options?

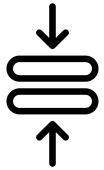


Use other models

Loss of performance, change in interpretability, …

Transpile to other formats

Large engineering effort with compatibility drawbacks



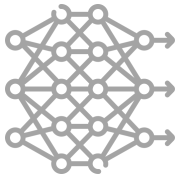
Compress with zstd/gzip

Didn’t bring the required performance

… go down the rabbit hole?

Let’s have a look at the internals!

what are our options?

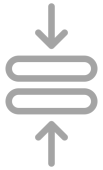


Use other models

Loss of performance, change in interpretability, …

Transpile to other formats

Large engineering effort with compatibility drawbacks



Compress with zstd/gzip

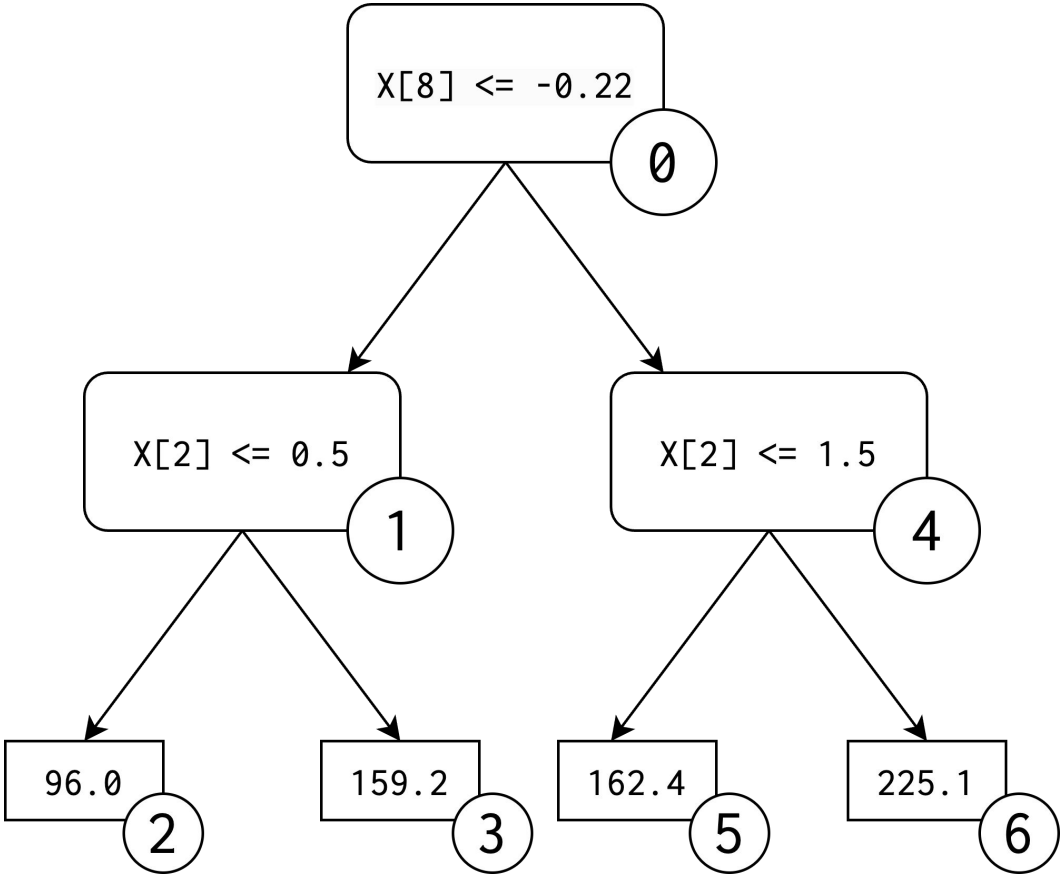
Didn’t bring the required performance

… go down the rabbit hole?

Let’s have a look at the internals!


tree internals

how do trees work again? 🌲



what's inside our models?

sklearn Pipeline

1. ColumnTransformer
2. ColumnTransformer
3. RandomForestRegressor
4. RandomForestRegressor

```
pipeline.__getstate__()
```

what's inside our models?

sklearn Pipeline

1. ColumnTransformer
2. ColumnTransformer
3. RandomForestRegressor
4. RandomForestRegressor

```
pipeline.__getstate__()
```

what's inside our models?

sklearn Pipeline

1. ColumnTransformer
2. ColumnTransformer
3. RandomForestRegressor
4. RandomForestRegressor

`pipeline.__getstate__()`

RandomForestRegressor

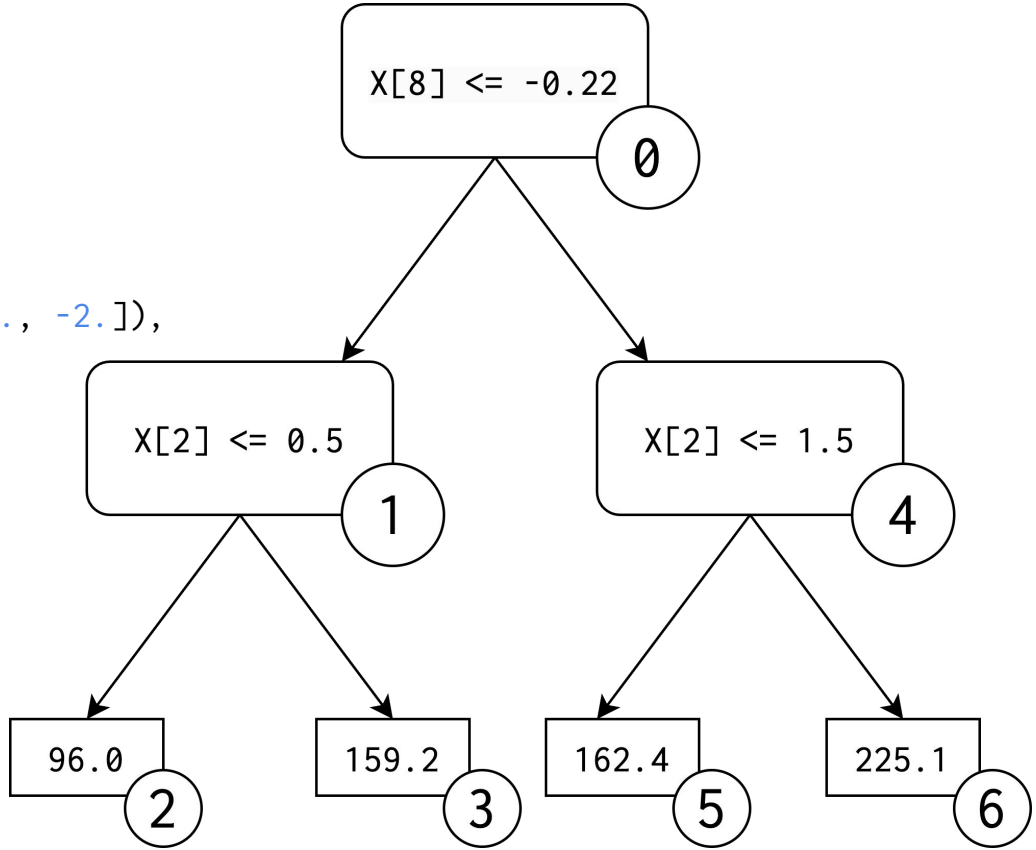
- DecisionTreeRegressor
- DecisionTreeRegressor
- DecisionTreeRegressor
- DecisionTreeRegressor
- ...

`regressor.__getstate__()`

what's inside a tree? 🌲

```
{
  "left_child":
    np.array([1, 2, -1, -1, 5, -1, -1]),
  "right_child":
    np.array([4, 3, -1, -1, 6, -1, -1]),
  "feature":
    np.array([8, 2, -2, -2, 2, -2, -2]),
  "threshold":
    np.array([-0.22, -.5, -2., -2., 1.5, -2., -2.]),
  "impurity":
    np.array([5929.88, 3240.8, ...]),
  "n_node_samples":
    np.array([442, 218, ...]),
  "weighted_n_node_samples":
    np.array([442., 218., ...]),
  "value":
    np.array([152.1, 109.9, 96., ...]),
}
```

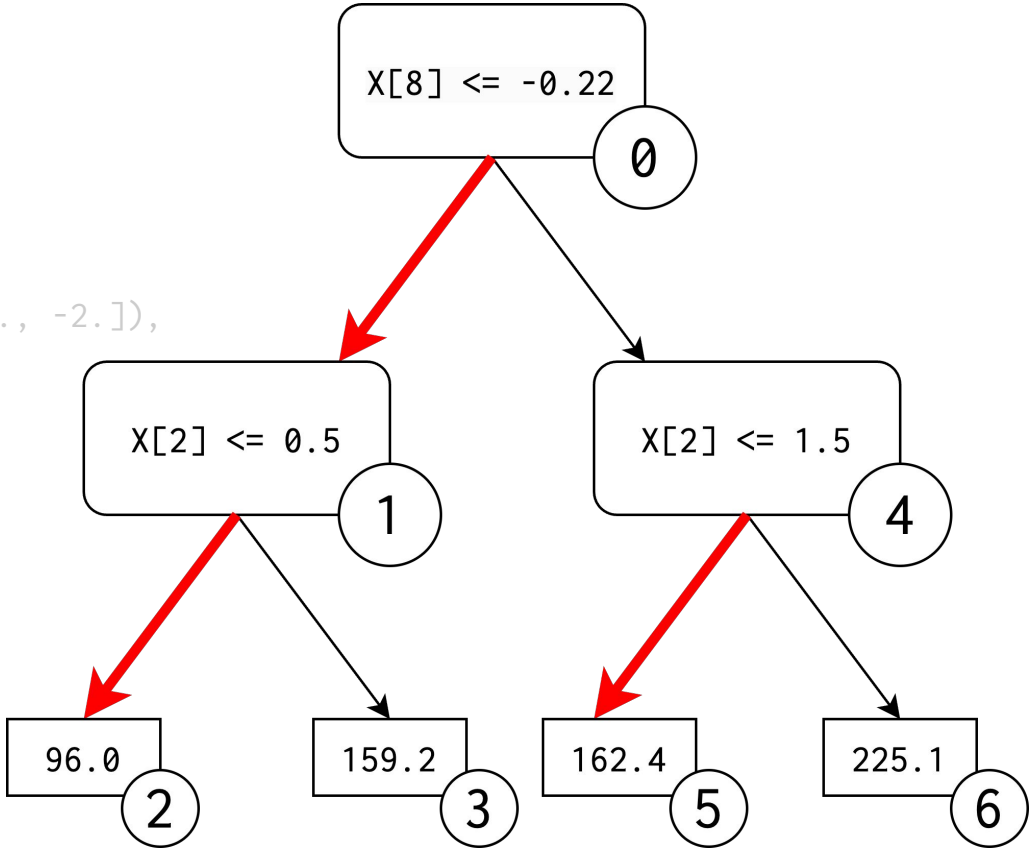
`tree.__getstate__()`



what's inside a tree? 🌲

```
{
  "left_child":
    np.array([1, 2, -1, -1, 5, -1, -1]),
  "right_child":
    np.array([4, 3, -1, -1, 6, -1, -1]),
  "feature":
    np.array([8, 2, -2, -2, 2, -2, -2]),
  "threshold":
    np.array([-0.22, -.5, -2., -2., 1.5, -2., -2.]),
  "impurity":
    np.array([5929.88, 3240.8, ...]),
  "n_node_samples":
    np.array([442, 218, ...]),
  "weighted_n_node_samples":
    np.array([442., 218., ...]),
  "value":
    np.array([152.1, 109.9, 96., ...]),
}
```

tree.__getstate__()



where do the `1.21 GB` come from?

```
left_child:
  8 Byte
right_child:
  8 Byte
feature:
  8 Byte
threshold:
  8 Byte
impurity:
  8 Byte
n_node_samples:
  8 Byte
weighted_n_node_samples:
  8 Byte
value:
  8 Byte
```

~12,500 nodes per tree

x ~1,500 trees

x 64 Bit = 8 Byte per value

= 150 MB per attribute

→ 1,200 MB in total

```
left_child:
  150 MB
right_child:
  150 MB
feature:
  150 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

left_child / right_child

```
left_child: np.array([1, 2, -1, -1, 5, -1, -1], dtype=np.int64)
```

- A lot of -1 values (leaves)

```
left_child:
  150 MB
right_child:
  150 MB
feature:
  150 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

1200 MB

left_child / right_child

```
left_child: np.array([1, 2, -1, -1, 5, -1, -1], dtype=np.int64)
```

- A lot of -1 values (leaves)
- Only save non-leaves and leaf indices

```
left_child:
  150 MB
right_child:
  150 MB
feature:
  150 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

1200 MB

left_child / right_child

```
left_child: np.array([1, 2, -1, -1, 5, -1, -1], dtype=np.int64)
```

- A lot of -1 values (leaves)
- Only save non-leaves and leaf indices
- Most trees have < **65535** nodes → **uint16** can be used

```
left_child: np.array([1, 2, 5], dtype=np.uint16)
is_leaf: np.array([False, False, True, True, False, True, True])
```

```
left_child:
  150 MB
right_child:
  150 MB
feature:
  150 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

1200 MB

left_child / right_child

```
left_child: np.array([1, 2, -1, -1, 5, -1, -1], dtype=np.int64)
```

- A lot of -1 values (leaves)
- Only save non-leaves and leaf indices
- Most trees have < **65535** nodes → **uint16** can be used

```
left_child: np.array([1, 2, 5], dtype=np.uint16)
is_leaf: np.array([False, False, True, True, False, True, True])
```

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

1200 MB → 940 MB

feature

```
feature: np.array([8, 2, -2, -2, 2, -2, -2], dtype=np.int64)
```

- A lot of -2 values (leaves)
- Only save non-leaves
- Most models have < **65535** features → **uint16** can be used

```
feature: np.array([8, 2, 2], dtype=np.uint16)
```

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

940 MB → 809 MB

threshold

```
threshold: np.array(
    [-0.22, 0.5, -2., -2., 1.5, -2., -2.],
    dtype=np.float64
)
```

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

809 MB

threshold

```
threshold: np.array(
    [-0.22, 0.5, -2., -2., 1.5, -2., -2.],
    dtype=np.float64
)
```

Remove leaf thresholds `-2.`

```
threshold: np.array([-0.22, 0.5, 1.5], dtype=np.float64)
```

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

809 MB

threshold

```
threshold: np.array(
    [-0.22, 0.5, -2., -2., 1.5, -2., -2.],
    dtype=np.float64
)
```

Remove leaf thresholds `-2.`

```
threshold: np.array([-0.22, 0.5, 1.5], dtype=np.float64)
```

float64 → **float32**

```
threshold: np.array([-0.22, 0.5, 1.5], dtype=np.float32)
```

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

809 MB

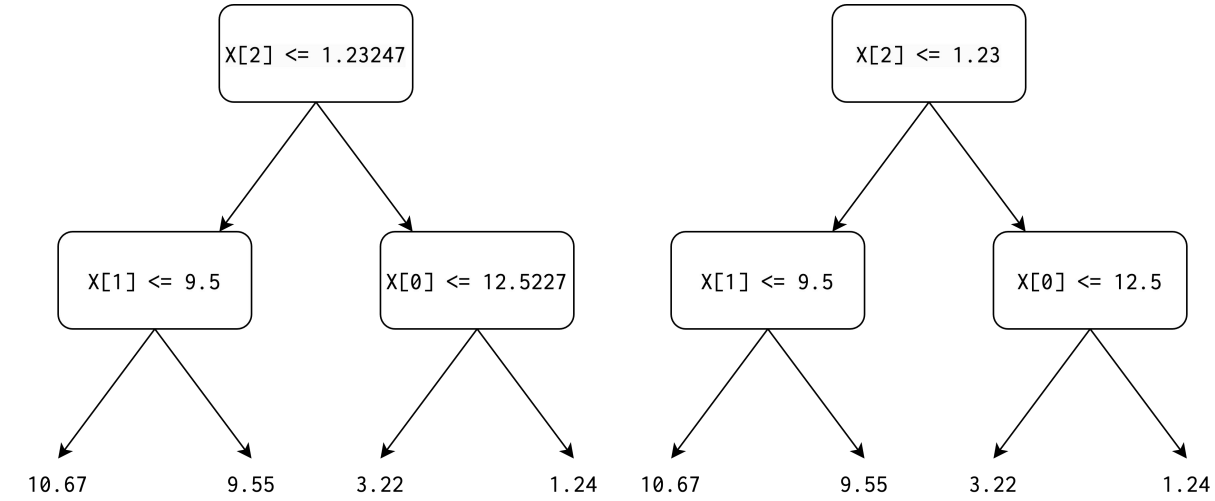
… what about the predictions, though?

threshold

⚠️ `float64` → `float32`

In our tests, we found that in rare cases, the new model predicts something completely different

`X = [3.22, 11, 1.231]`




```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

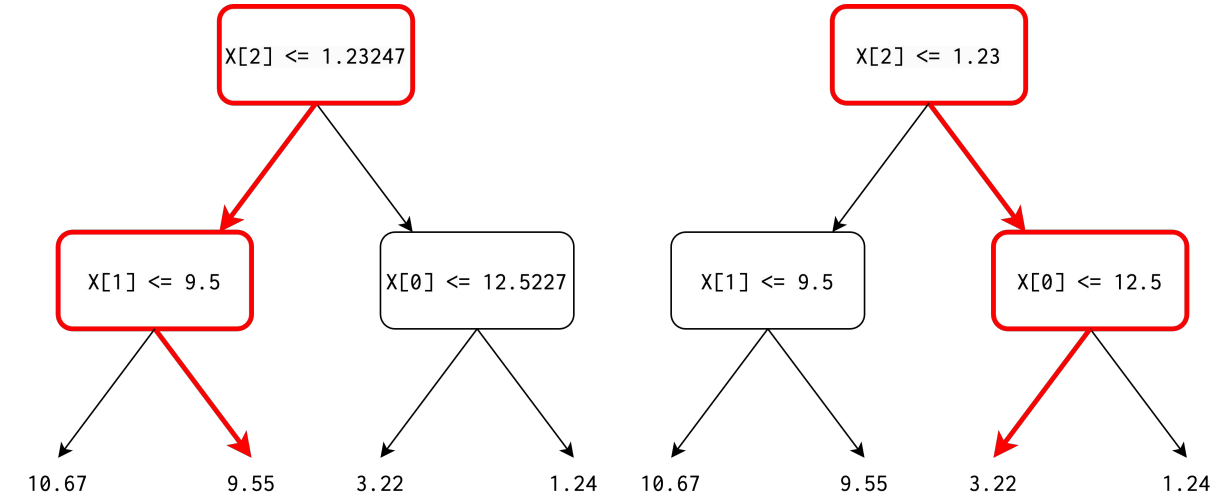
809 MB

threshold

⚠️ `float64` → `float32`

In our tests, we found that in rare cases, the new model predicts something completely different

`X = [3.22, 11, 1.231]`




```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

809 MB

half-int compression

```
threshold: np.array([-0.22, 0.5, 1.5], dtype=np.float64)
```

A lot of `x.5` splits (categoricals)

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

809 MB

half-int compression

`threshold: np.array([-0.22, 0.5, 1.5], dtype=np.float64)`

A lot of `x.5` splits (categoricals)

Store the `x.5` values as an `uint8` by rounding down

```
threshold_is_compressible:
  np.array([False, True, True])
threshold_compressible:
  np.array([0, 1], dtype=np.uint8)
threshold_incompressible:
  np.array([-0.22], dtype=np.float64)
```

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

809 MB

half-int compression

```
threshold: np.array([-0.22, 0.5, 1.5], dtype=np.float64)
```

A lot of `x.5` splits (categoricals)

Store the `x.5` values as an `uint8` by rounding down

```
threshold_is_compressible:
  np.array([False, True, True])
threshold_compressible:
  np.array([0, 1], dtype=np.uint8)
threshold_incompressible:
  np.array([-0.22], dtype=np.float64)
```

*~65% of our splits are categorical

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB → 33.5 MB*
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

809 MB → 692 MB

some things are not worth holding on to…

these values are
not needed for inference at all!
🎉

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB → 33.5 MB
impurity:
  150 MB
n_node_samples:
  150 MB
weighted_n_node_samples:
  150 MB
value:
  150 MB
```

692 MB

some things are not worth holding on to…

these values are
not needed for inference at all!
🎉

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB → 33.5 MB
impurity:
  150 MB → 0 MB
n_node_samples:
  150 MB → 0 MB
weighted_n_node_samples:
  150 MB → 0 MB
value:
  150 MB
```

692 MB → 242 MB

value

```
value: np.array(
  [152.1, 109.9, 96., 159., 184.6, 162., 225.],
  dtype=np.float64
)
```

- Only leaves are of relevance for inference

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB → 33.5 MB
impurity:
  150 MB → 0 MB
n_node_samples:
  150 MB → 0 MB
weighted_n_node_samples:
  150 MB → 0 MB
value:
  150 MB → 37.5 MB
```

242 MB → 130 MB

value

```
value: np.array(
  [152.1, 109.9, 96., 159., 184.6, 162., 225.],
  dtype=np.float64
)
```

- Only leaves are of relevance for inference
- We can save `value` → `float32`
- ⚠️ loss of precision

```
value: np.array(
  [96., 159., 162., 225.],
  dtype=np.float32
)
```

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB → 33.5 MB
impurity:
  150 MB → 0 MB
n_node_samples:
  150 MB → 0 MB
weighted_n_node_samples:
  150 MB → 0 MB
value:
  150 MB → 37.5 MB
```

242 MB → 130 MB

final verdict

1200 MB

130 MB

9x smaller 🎉

```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB → 33.5 MB
impurity:
  150 MB → 0 MB
n_node_samples:
  150 MB → 0 MB
weighted_n_node_samples:
  150 MB → 0 MB
value:
  150 MB → 37.5 MB
```

130 MB

final verdict

1200 MB

130 MB

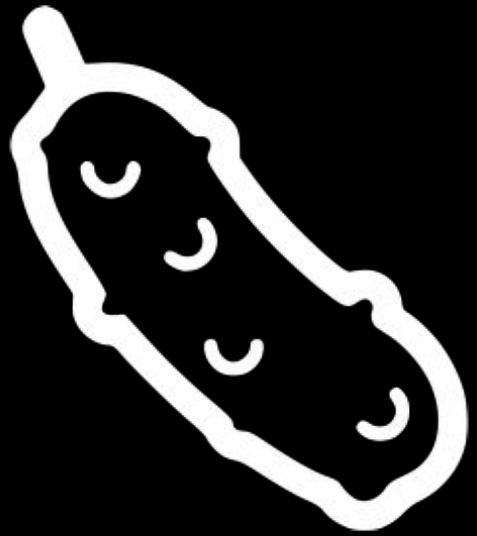
9x smaller 🎉

How to implement this, though?

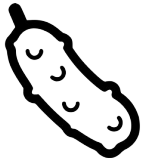
```
left_child:
  150 MB → 21.1 MB
right_child:
  150 MB → 18.8 MB
feature:
  150 MB → 18.8 MB
threshold:
  150 MB → 33.5 MB
impurity:
  150 MB → 0 MB
n_node_samples:
  150 MB → 0 MB
weighted_n_node_samples:
  150 MB → 0 MB
value:
  150 MB → 37.5 MB
```

130 MB

pickle internals



custom pickling functions

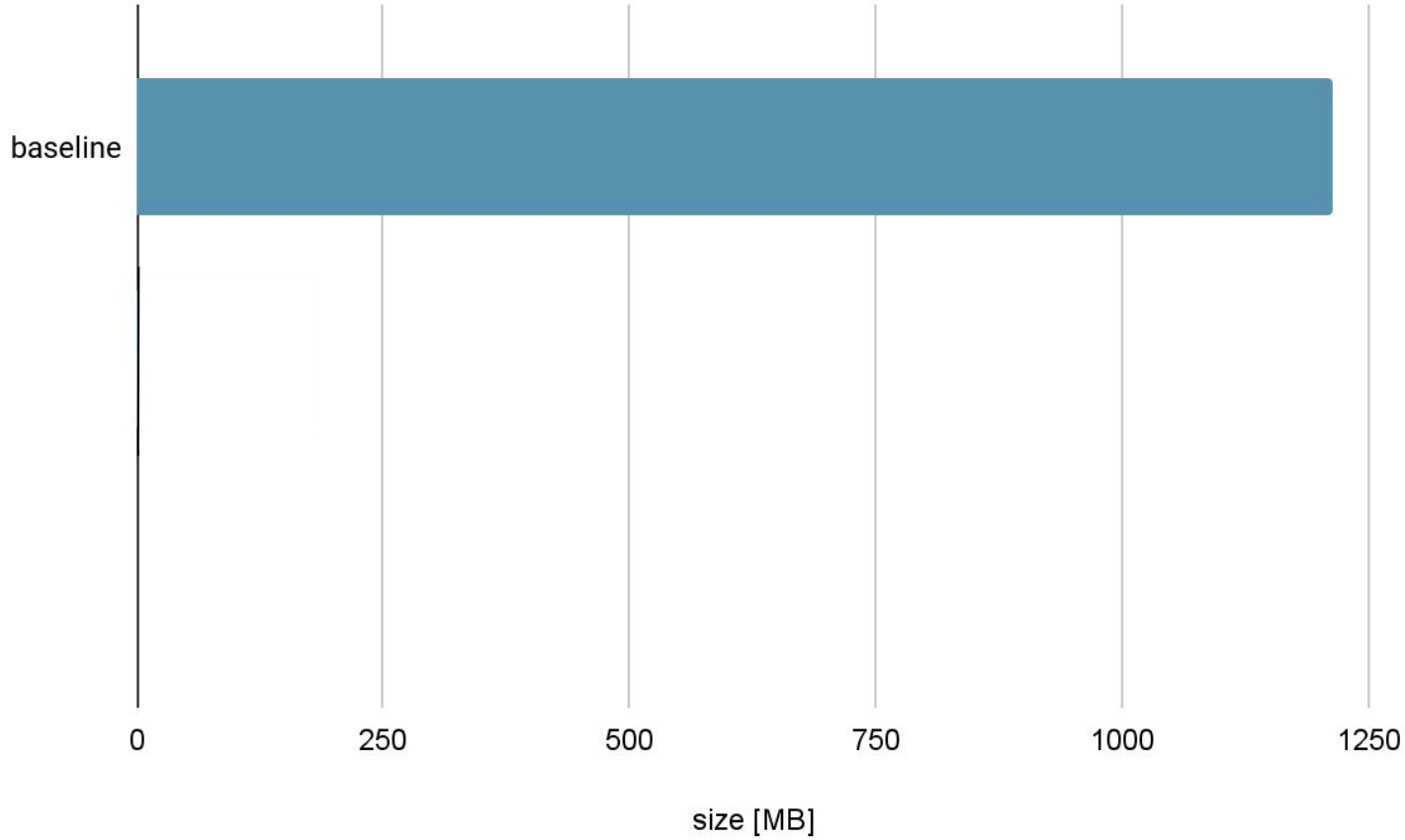


Create custom pickling behavior for sklearn Trees by customising the `dispatch_table` of a `Pickler`

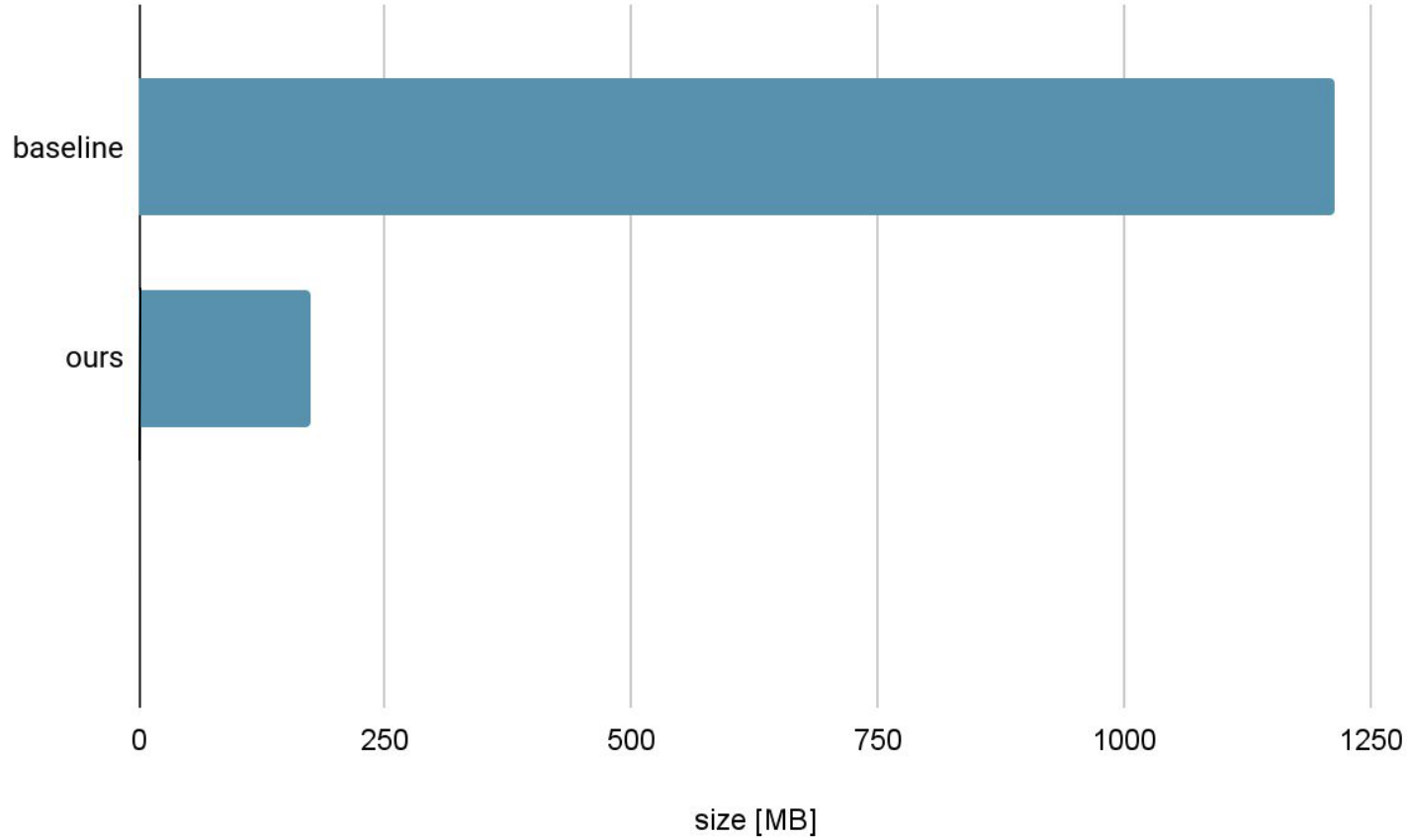
```
model: sklearn.RandomForestRegressor
file: io.BinaryIO

p = pickle.Pickler(file)
p.dispatch_table = copyreg.dispatch_table.copy()
p.dispatch_table[Tree] = compressed_tree_pickle
p.dump(model)
```

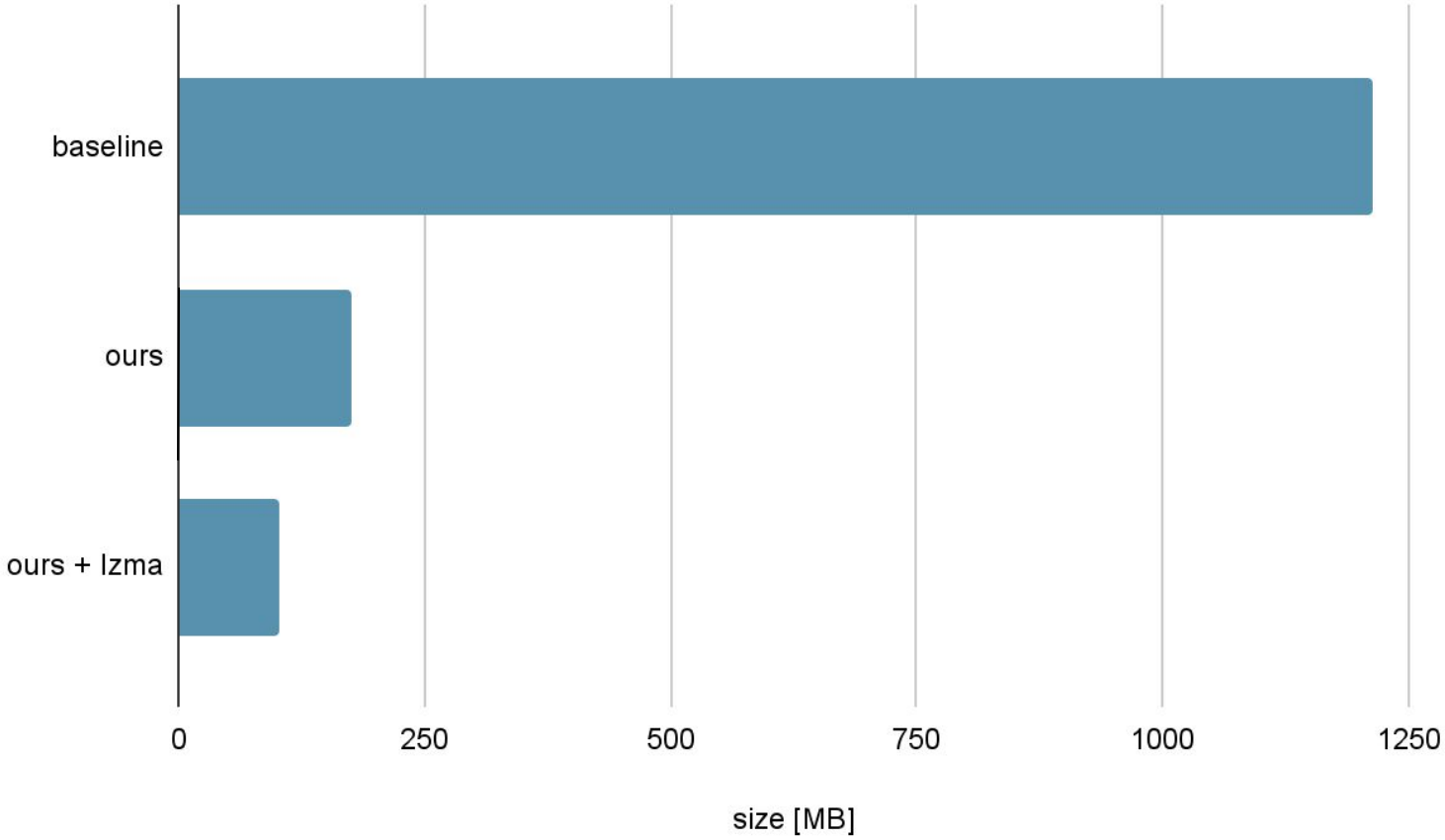
benchmark – size



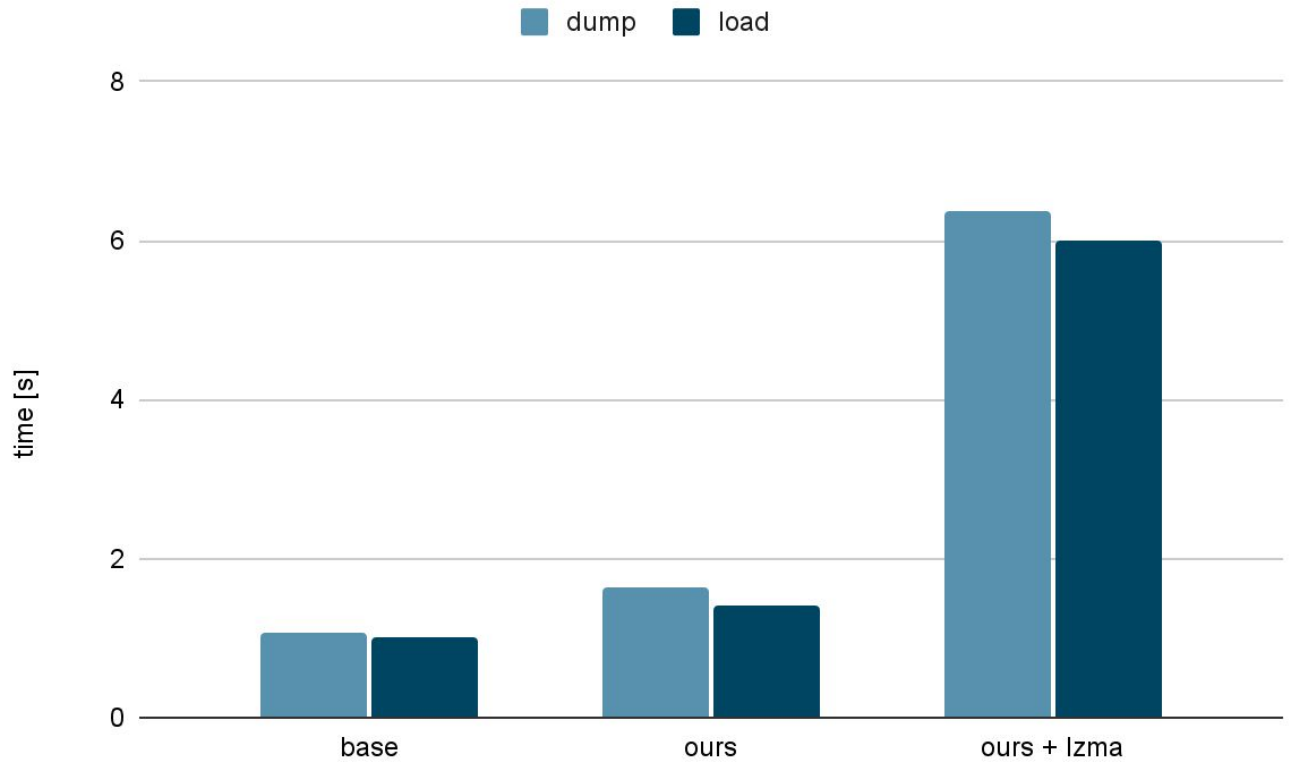
benchmark – size



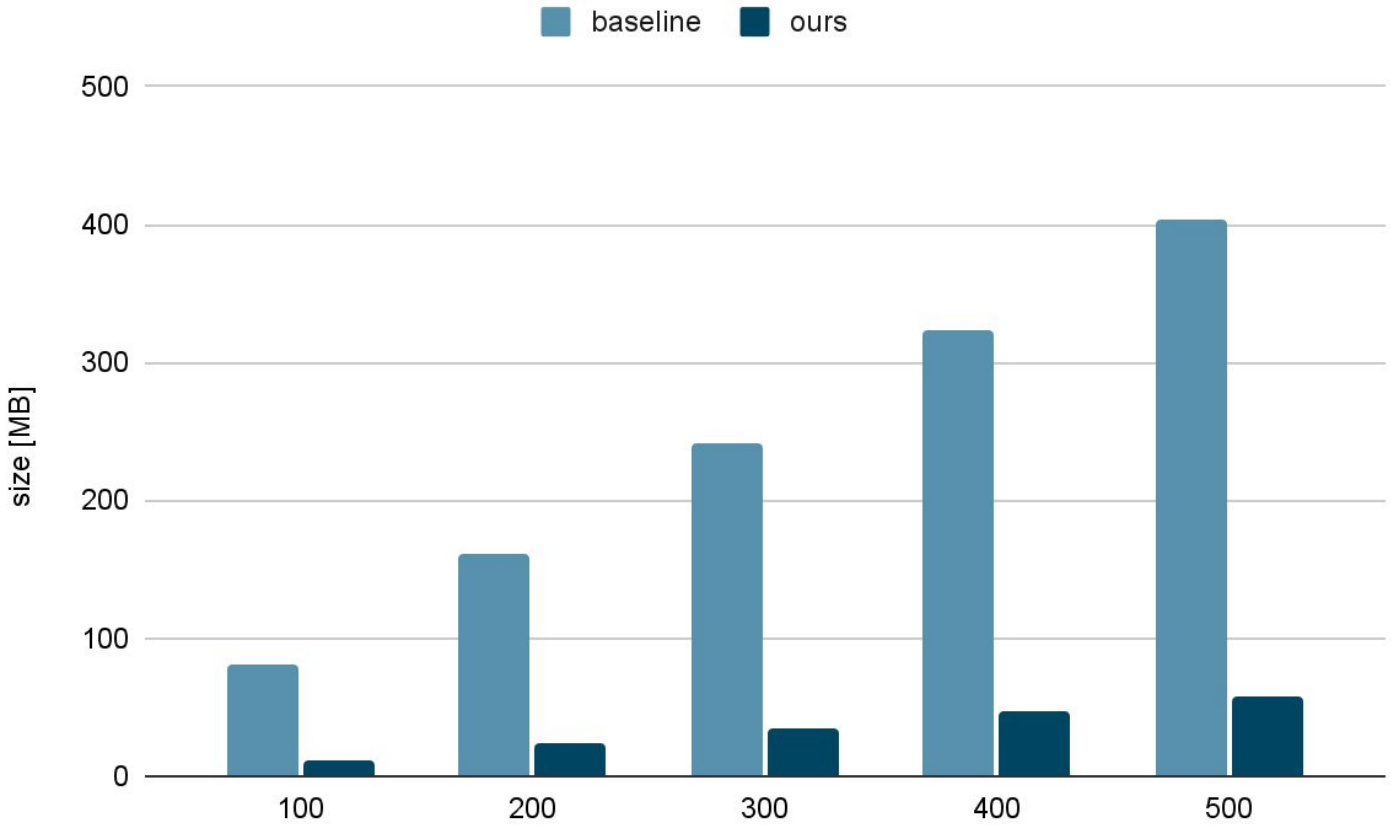
benchmark – size



benchmark – time (lower is better)

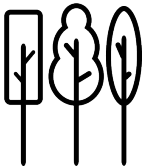


it scales



now at your doorstep 

slim-trees 🌲



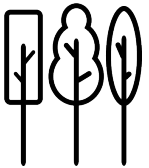
Installation

```
pip install slim-trees
conda install -c conda-forge slim-trees
```

```
from slim_trees import dump_sklearn_compressed

model = RandomForestRegressor(...)
model.fit(...)
path = "model.pkl.lzma"
dump_sklearn_compressed(model, path, compression="lzma")
```

slim-trees 🌲



…or use it as a drop-in replacement for **`pickle.dump(...)`**

```python
from slim_trees import sklearn_tree

with open(path, "wb") as f:
  sklearn_tree.dump(model, f)
```

Instead of **`pickle.dump(model, f)`**

```python
with open(path, "rb") as f:
  model_loaded = pickle.load(f)
```

customize *your* pickle pipelines - a blueprint

1. find out how your object is saved

2. look you if you can do better

3. modify pickling behaviour

4. profit 💸💸💸

example: lightgbm stores models as .txt files

```
Tree=0
num_leaves=8
num_cat=0
split_feature=182 199 261
split_gain=969644 107552 83253.3
threshold=-0.0059538649999999992
1498.0047400000001
...
```

customize *your* pickle pipelines - a blueprint

1. find out how your object is saved
2. look you if you can do better
3. modify pickling behaviour
4. profit 💸💸💸

```
def parse_string(model_string: str) → dict

def apply_compression(values: dict) → dict

def create_model_string(values: dict) → str
```

customize *your* pickle pipelines - a blueprint

1. find out how your object is saved

2. look you if you can do better

3. modify pickling behaviour

4. profit 💸💸💸

```
p = pickle.Pickler(file)
p.dispatch_table = \
     copyreg.dispatch_table.copy()
p.dispatch_table[Booster] = handle_tree
p.dump(model)
```

customize *your* pickle pipelines - a blueprint

1. find out how your object is saved

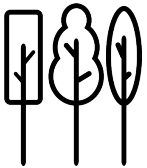
2. look you if you can do better

3. modify pickling behaviour

4. profit 💸💸💸

6x improvement on space!

slim-trees 🌲



Installation

```
pip install slim-trees
conda install -c conda-forge slim-trees
```

```
from slim_trees import dump_lgbm_compressed
```

Also supports LightGBM 🎉

```
model = LGBMRegressor(...)
model.fit(...)
path = "model.pkl.lzma"
dump_lgbm_compressed(model, path, compression="lzma")
```

overview

half-int compression

only store what's necessary

modify pickling behaviour

```
pip install slim-trees
conda install slim-trees
```

https://github.com/quantco/slim-trees

pavel
data engineer
@pavelzw

yasin
data engineer
@YYYasin19

come work on open source software with us!
join@quantco.com