

Improving performance using scipy.sparse

Note that this only works in the special case of binary classification or regression as scipy.sparse only supports 2d matrices.

This could be generalized using pydata.sparse but currently that uses too much memory.

The next two slides show the CPU and Memory profiling analysis of the script to the right.

The profilers seem to indicate that using scipy.sparse cuts runtime by more than half and uses 7% less memory.

```
# Test bench mark different approaches to the pvalue calculation to figure out
# the best way to calculate the pvalue.
# code is based on examples/treepile/treepile_tutorial_1_2_pvalue.py

import sys
from treepile.datasets import make_trunk_classification
from treepile.ensemble import HonestForestClassifier
from treepile.stats import PermutationHonestForestClassifier, build_coleman_forest

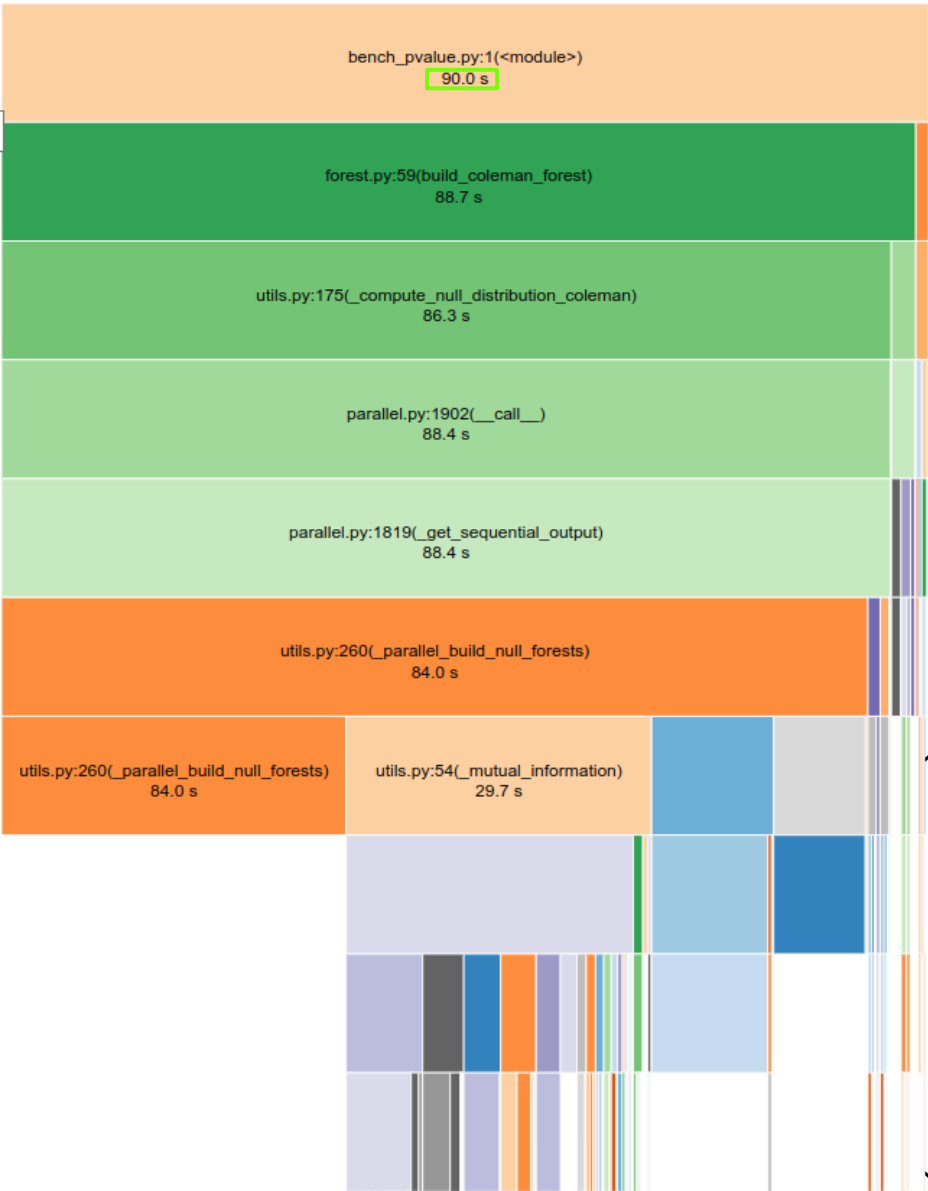
N_PERMUTATIONS = 10000

X, y = make_trunk_classification(
    n_samples=1000,
    n_dim=1,
    mu_0=0,
    mu_1=1,
    n_informative=1,
    seed=1,
)

est = HonestForestClassifier(
    n_estimators=100,
    max_samples=1.6,
    max_features=0.3,
    bootstrap=True,
    stratify=True,
    random_state=1,
)

est_null = PermutationHonestForestClassifier(
    n_estimators=100,
    max_samples=1.6,
    max_features=0.3,
    bootstrap=True,
    stratify=True,
    random_state=1,
)

observed_diff, _, _, pvalue, mix_diff = build_coleman_forest(
    est,
    est_null,
    X,
    y,
    metric="mi",
    n_repeats=N_PERMUTATIONS,
    return_posteriors=False,
    seed=1,
    use_sparse="sparse" in sys.argv,
)
```

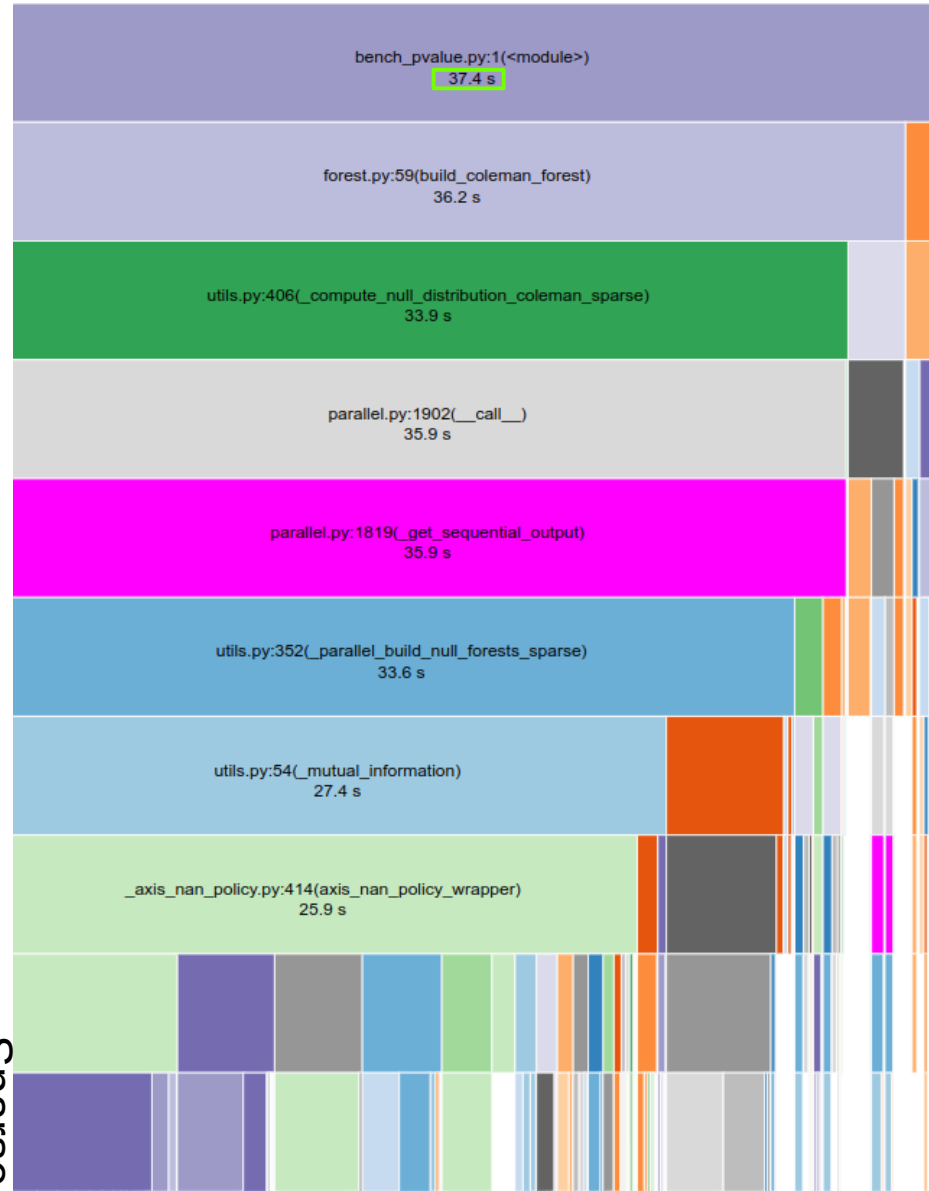


CPU Profiling

The sparse implementation takes ~58% less time according to cProfile

Dense (includes bottleneck)

Sparse



Dense

Total for benchmarking script: ~129MB

Total for build_coleman_forest: ~24.2MB

Sparse

Total for benchmarking script: ~120.1MB

Total for build_coleman_forest: ~15.2MB

```
<root>
runpy.run_module(args.script, run_name="__main__", alter_sys=True)
observed_diff, _, pvalue, mix_diff = build_coleman_forest(
est, orig_forest_proba = build... metric_star, metric_star_pi = _compute_null_distribution_coleman( perm_est, perm_forest_proba ...
est.fit(X, y.ravel(),... all_pr... out = Parallel(n_jobs=n_jobs)( for i, seed in zip(ra... all_y_pred = np... est.fit(X, y.ravel(), **e... all_pr...
return fit_method(... a = e... return output if self.return_generator else list(... return fit_method(est... a = e...
super().fit(X, y, sa... res = func(*args, **kwargs) self = super().fit(X, y, ...
return fit_method... se... y_pred_se... firs... second_for... return fit_method(est...
self_construct_tr... H_... H_... super().fit(X, y, samp...
trees = Parallel( re... x ... return fit_method(est...
return output if s... wit... ar... self_construct_trees(
res = func(*args, ... sel... ret... trees = Parallel(
tree_fit( return output if self.r...
    self.estimat... ret... res = func(*args, **k...
    self = self_... (ar... tree_fit(
        builder.bull... self.estimator_...
        self = self_buil...
        builder.build(sel...
```

```
<root>
runpy.run_module(args.script, run_name="__main__", alter_sys=True)
observed_diff, _, pvalue, mix_diff = build_coleman_forest(
est, orig_forest_proba = build_oob... perm_est, perm_forest_proba = build_oo... _compute_null_distribution_coleman_sparse(
est.fit(X, y.ravel(), **est_kwargs) est.fit(X, y.ravel(), **est_kwargs) out = Parallel(n_job... for _, seed in zip(range(n_repe... all...
return fit_method(estimator, *a... return fit_method(estimator, *args, *... return output if self...
super().fit(X, y, sample_weigh... self = super().fit(X, y, sample_weigh... res = func(*args, *...
return fit_method(estimator, *... return fit_method(estimator, *args, *... first_h... second...
self_construct_trees( super().fit(X, y, sample_weight=sa... H_YX ... H_YX ...
trees = Parallel( return fit_method(estimator, *args, *... return ... x = _br...
return output if self.return_ge... self_construct_trees( return ... arrays ...
res = func(*args, **kwargs) trees = Parallel( return ...
tree_fit( return output if self.return_generato... return ...
    self.estimator_fit( res = func(*args, **kwargs) return ...
    self = self_build_tree( tree_fit( (array...
        builder.build(self.tree... self.estimator_fit(
        self = self_build_tree(
        builder.build(self.tree_X, y,...
```