The Journal of Open Source Software

# 'ControlBurn: Explainable Feature Selection with Sparse Forests in Python'

**Brian Liu**[1], **Miaolan Xie**[2], **and Madeleine Udell**[3]

**1** Operations Research Center, Massachusetts Institute of Technology **2** Operations Research and Information Engineering, Cornell University **3** Management Science and Engineering, Stanford University

**DOI:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:** Open Journals

**Reviewers:**

- @openjournals

**Submitted:** 01 January 1970
**Published:** unpublished



## Summary

`ControlBurn` is a Python package for nonlinear feature selection and interpretable machine learning. The algorithms in this package first build large tree ensembles that prioritize basis functions with few features and then select a feature-sparse subset of these basis functions using a weighted lasso optimization criterion. The package is scalable and flexible: for example, it can compute the regularization path (prediction error for any number of selected features) for a dataset with tens of thousands of samples and hundreds of features in seconds. Moreover, `ControlBurn` explains why features are selected and includes visualizations to analyze the impact of these features on the predictions of the model.

## Statement of Need

Feature selection is commonly used in machine learning to improve model interpretability, parsimony, and generalization. For linear models, methods such as the lasso (Tibshirani, 1996), group lasso (Friedman et al., 2010), and elastic net (Zou & Hastie, 2005) are frequently used to obtain sparse models. These techniques are valued for their ease of use and computational efficiency. Feature selection for nonlinear models, however, is a more challenging task. Wrapper-based feature selection algorithms are computationally expensive and require repeatedly retraining the model (Darst et al., 2018). Feature selection methods based on importance metrics derived from nonlinear models, such as the mean decrease in impurity (MDI) importance scores for random forests, can be biased and fail when features are correlated (Liu et al., 2021; Zhou & Hooker, 2021).

The flexibility of nonlinear models, such as random forests and boosted decision trees, enhance their ability to capture complex relationships in the data and as such, these models are extremely popular in many data science applications. However, black-box nonlinear models lack interpretability and are typically feature dense. `ControlBurn` improves the interpretability of nonlinear models by implementing an efficient feature selection algorithm that works well, even when there are many correlated features. More importantly, `ControlBurn` explains why it selects a set of features, which allows practitioners to gain real-world insight into the underlying relationships in the data. `ControlBurn` is a valuable tool for data scientists, researchers, and academics looking to increase the interpretability of nonlinear models.

## Nonlinear feature selection with sparse tree ensembles

`ControlBurn` builds a large tree ensemble that captures nonlinear relationships and feature interactions and choses a subset of trees that jointly use a small subset of features. The performance of the feature selection algorithm is sensitive to the quality and diversity of the tree ensemble; `ControlBurn` works best when each tree in the ensemble only uses a few features.

We discuss how `ControlBurn` builds good tree ensembles and selects feature sparse subsets of trees below.

### Building good ensembles

`ControlBurn` contains many specialized algorithms to build good tree ensembles, ensembles where each tree uses a different subset of features. These algorithms include:

– Incremental Depth Bagging

– Incremental Depth Bag-Boosting

– Incremental Depth Double Bag-Boosting.

These algorithms start by building simple trees to isolate the effects of single features. They increase tree complexity only if doing so improves the validation accuracy of the ensemble. We discuss how to build good ensembles in greater detail in (Liu et al., 2021).

### Selecting sparse subsets of tree

Suppose that we have constructed $T$ decision trees, each tree $t \in 1 ... T$ is associated with a vector of predictions $a^{(t)} \in \mathbb{R}^N$. Our goal is to choose a non-negative sparse weight vector $w \in \mathbb{R}^T$ such that the weighted sum of the prediction vectors matches the response $y$ as closely as possible. We also add a regularization penalty to encourage feature sparsity in the selected ensemble. Let $u \in \mathbb{R}^T$ denote where each entry $t$ denots the number of features used by tree $t$. The optimization formulation to select feature sparse subsets of tree is given by:

$$\min_{w \geq 0} L(y, \sum_{t=1}^{T} a^{(t)} w^{(t)}) + \alpha u^\top w,$$

where $L$ is the loss function, for example least-squares loss foo regression or log-loss for classification, and $\alpha$ is the regularization penalty that controls sparsity. We say that a feature is pruned from the ensemble if none of the selected trees use that feature. This optimization problem can be reformulated into a weighted non-negative garrote problem and efficiently solved using off-the-shelf solvers. For more details on selecting sparse subsets of trees refer to (Liu et al., 2021).

## `ControlBurn` Example

We use `ControlBurn` to select features from the California Housing Prices regression dataset (Dua & Graff, 2017) and highlight the features that `ControlBurn` uses for explainable feature selection. The code chunk below uses `ControlBurn` to select 4 features out of the 8 in the dataset.

```
from ControlBurn.ControlBurnModel import ControlBurnRegressor
cb = ControlBurnRegressor(build_forest_method = 'doublebagboost', alpha = 0.02)
cb.fit(xTrain,yTrain)
prediction = cb.predict(xTest)
features = cb.features_selected_

features
>>> ['MedInc', 'HouseAge', 'Latitude', 'Longitude']
```

In this case, the test performance of the sparse model is the same as the test performance of the full model; `ControlBurn` was able to quickly remove 4 redundant features.

### Explainable Feature Selection

We use the `interpreter` module in `ControlBurn` to explain why these features were selected. We can first plot the feature importance scores of the features.

```
from ControlBurn.ControlBurnInterpret import InterpretRegressor
interpreter = InterpretRegressor(cb,xTrain,yTrain)
importance_score = interpreter.plot_feature_importances()
```

This allows us to understand the relative impact of each feature.

We can also visualize the features used by each tree in the selected sparse ensemble.

```
features_per_tree = interpreter.list_subforest(verbose = True)
>>> ['MedInc'], ['MedInc'], ['MedInc'], ['MedInc'], ['MedInc'], ['MedInc'],
    ['MedInc'], ['MedInc'], ['MedInc'], ['MedInc'], ['MedInc'],
    ['Latitude' 'Longitude'], ['Latitude' 'Longitude'], ['MedInc' 'HouseAge'],
    ['Latitude' 'Longitude'], ['Latitude' 'Longitude']
```

This provides details on if a feature was selected due to its individual contribution to the prediction, like 'MedInc' or if two features are important due to an interaction, for example 'Latitude' and 'Longitude'. The latter case is important since nonlinear black-box models typically use many feature interactions.

For single features and pairwise interaction we can use `ControlBurn` to further examine how the features impact the prediction of the model by visualizing the shape functions of the features.

The `interpreter` module accomplishes this using the code,

```
plot_single = interpreter.plot_single_feature_shape('MedInc')
```

which outputs this figure.

From this shape function, we observe that as expected, the predicted housing price value increase with the median income of the surrounding area.

To visualize the shape function of interactions, use the code:

```
plot_pairwise = interpreter.plot_pairwise_interactions('Latitude','Longitude')
```

which outputs this plot.

We can overlay this plot over a map of California to see that housing prices increase around the San Francisco Bay Area and Los Angeles, as expected.

To summarize, `ControlBurn` can rapidly select a subset of high performing features to include in nonlinear models. With the `interpreter` module, a practitioner can analyze why such models were selected and visualize how the features contribute to the prediction of the model. This allows a practitioner to discover real-world insights from the underlying patterns in the data.

## Implementation and Additional Capabilities

`ControlBurn` can be installed via the Python Package Index and is available for Python 3.7 and above. The following dependencies are required.

– Numpy (Harris et al., 2020)

– Pandas (team, 2022)

– Scikit-learn (Pedregosa et al., 2011)

In addition, `ControlBurn` can support many additional functionalities for nonlinear feature selection, including CV-tuning, feature costs, feature groupings, and custom ensembles. Liu et al. (2022) provides a comprehensive tutorial on these additional capabilities.

## References


Darst, B. F., Malecki, K. C., & Engelman, C. D. (2018). Using recursive feature elimination in random forest to account for correlated variables in high dimensional data. *BMC Genetics*, *19*(1), 1–6.

Dua, D., & Graff, C. (2017). *UCI machine learning repository*. University of California, Irvine, School of Information; Computer Sciences. http://archive.ics.uci.edu/ml

Friedman, J., Hastie, T., & Tibshirani, R. (2010). A note on the group lasso and a sparse group lasso. *arXiv Preprint arXiv:1001.0736*.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

Liu, B., Xie, M., & Udell, M. (2021). ControlBurn: Feature selection by sparse forests. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1045–1054.

Liu, B., Xie, M., Yang, H., & Udell, M. (2022). ControlBurn: Nonlinear feature selection with sparse tree ensembles. *arXiv Preprint arXiv:2207.03935*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

team, The pandas development. (2022). *Pandas-dev/pandas: Pandas 1.4.1* (Version v1.4.1). Zenodo. https://doi.org/10.5281/zenodo.6053272

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.

Zhou, Z., & Hooker, G. (2021). Unbiased measurement of feature importance in tree-based methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *15*(2), 1–21.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), 301–320.