Deep Forest (DF21)

Yi-Xuan Xu

Mar 30, 2023

FOR USERS

DF21 is an implementation of Deep Forest 2021.2.1. It is designed to have the following advantages:

- **Powerful**: Better accuracy than existing tree-based ensemble methods.
- **Easy to Use**: Less efforts on tunning parameters.
- **Efficient**: Fast training speed and high efficiency.
- **Scalable**: Capable of handling large-scale data.

DF21 offers an effective & powerful option to the tree-based machine learning algorithms such as Random Forest or GBDT. This package is actively being developed, and any help would be welcomed. Please check the homepage on Gitee or Github for details.

CHAPTER

ONE

GUIDEPOST

- For a quick start, please refer to How to Get Started.
- For a guidance on tunning parameters for DF21, please refer to Parameters Tunning.
- For a comparison between DF21 and other tree-based ensemble methods, please refer to Experiments.

CHAPTER

TWO

INSTALLATION

DF21 can be installed using pip via PyPI which is the package installer for Python. You can use pip to install packages from the Python Package Index and other indexes. Refer this for the documentation of pip. Use this command to download DF21 :

```
$ pip install deep-forest
```

CHAPTER

THREE

QUICKSTART

3.1 Classification

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from deepforest import CascadeForestClassifier

X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
model = CascadeForestClassifier(random_state=1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
print("\nTesting Accuracy: {:.3f} %".format(acc))
>>> Testing Accuracy: 98.667 %
```

3.2 Regression

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

from deepforest import CascadeForestRegressor

X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
model = CascadeForestRegressor(random_state=1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("\nTesting MSE: {:.3f}".format(mse))
>>> Testing MSE: 8.068
```

CHAPTER

FOUR

RESOURCES

- Deep Forest: [Paper]
- Keynote at AISTATS 2019: [Slides]
- Source Code: [GitHub] | [Gitee]

CHAPTER

FIVE

REFERENCE

```
@article{zhou2019deep,
    title={Deep forest},
    author={Zhi-Hua Zhou and Ji Feng},
    journal={National Science Review},
    volume={6},
    number={1},
    pages={74--86},
    year={2019}}

@inproceedings{zhou2017deep,
    Author = {Zhi-Hua Zhou and Ji Feng},
    Booktitle = {IJCAI},
    Pages = {3553-3559},
    Title = {{Deep Forest:} Towards an alternative to deep neural networks},
    Year = {2017}}
```

5.1 How to Get Started

This is a quick start guide for you to try out deep forest. The full script is available at *Example*.

5.1.1 Installation

The package is available via PyPI. As a kind reminder, do not forget the hyphen (`-`) between `deep` and `forest`.

```
$ pip install deep-forest
```

5.1.2 Load Data

Deep forest assumes data to be in the form of 2D Numpy array of shape (`n_samples`, `n_features`). It will conduct internal check and transformation on the input data. For example, the code snippet below loads a toy dataset on digits classification:

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

X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

5.1.3 Define the Model

Deep forest provides unified APIs on binary classification and multi-class classification. For the demo dataset on classification, the corresponding model is `CascadeForestClassifier`:

```
from deepforest import CascadeForestClassifier

model = CascadeForestClassifier()
```

A key advantage of deep forest is its **adaptive model complexity depending on the dataset**. The default setting on hyper-parameters enables it to perform reasonably well across all datasets. Please refer to API Reference for the meaning of different input arguments.

5.1.4 Train and Evaluate

Deep forest provides Scikit-Learn like APIs on training and evaluating. Given the training data `X_train` and labels `y_train`, the training stage is triggered with the following code snippet:

```
model.fit(X_train, y_train)
```

Once the model was trained, you can call `predict()` to produce prediction results on the testing data `X_test`.

```
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100  # classification accuracy
```

5.1.5 Save and Load

Deep forest also provides easy-to-use APIs on model serialization. Here, `MODE_DIR` is the directory to save the model.

```
model.save(MODEL_DIR)
```

Given the saving results, you can call `load()` to use deep forest for prediction:

```
new_model = CascadeForestClassifier()
new_model.load(MODEL_DIR)
```

Notice that `new_model` is not the same as `model`, because only key information used for model inference was saved.

5.1.6 Example

Below is the full script on using deep forest for classification on a demo dataset.

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from deepforest import CascadeForestClassifier


# Load data
```

(continues on next page)

(continued from previous page)

```
X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)

model = CascadeForestClassifier()

# Train and evaluate
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
print("\nTesting Accuracy: {:.3f} %".format(acc))

# Save the model
model.save("model")
```

5.2 Installation Guide

5.2.1 Stable Version

The stable version is available via PyPI using:

```
$ pip install deep-forest
```

The package is portable and with very few package dependencies. It is recommended to use the package environment from Anaconda since it already installs all required packages.

Notice that only the 64-bit Linux, Windows, and Mac-OS platform are officially supported. To use deep forest on other platforms, you will need to build the entire package from source.

5.2.2 Building from Source

Building from source is required to work on a contribution (bug fix, new feature, code or documentation improvement).

- **Use Git to check out the latest source from the repository on Gitee or Github:**

```
$ git clone https://gitee.com/lamda-nju/deep-forest.git
$ git clone https://github.com/LAMDA-NJU/Deep-Forest.git
$ cd Deep-Forest
```

- **Install a C compiler for your platform.**

Note: The compiler is used to compile the Cython files in the package. Please refer to Installing Cython for details on choosing the compiler.

- **Optional (but recommended): create and activate a dedicated virtual environment or conda environment.**
- **Build the project with pip in the editable mode:**

```
$ pip install --verbose -e .
```

Note: The key advantage of the editable mode is that there is no need to re-install the entire package if you have modified a Python file. However, you still have to run the `pip install --verbose -e .` command again if the source code of a Cython file is updated (ending with .pyx or .pxd).

- **Optional: run the tests on the module:**

```
$ cd tests
% pytest
```

5.2.3 Acknowledgement

The installation instructions were adapted from Scikit-Learn's advanced installation instructions.

5.3 API Reference

Below is the class and function reference for `deepforest`. Notice that the package is still under active development, and some features may not be stable yet.

5.3.1 CascadeForestClassifier

fit(X, y[, sample_weight])	Build a deep forest using the training data.
predict_proba(X)	Predict class probabilities for X.
predict(X)	Predict class for X.
clean()	Clean the buffer created by the model.
get_estimator(layer_idx, est_idx, estimator_type)	Get estimator from a cascade layer in the deep forest.
get_layer_feature_importances(layer_idx)	Return the feature importances of `layer_idx`-th cascade layer.
load(dirname)	Load the model from the directory `dirname`.
save([dirname])	Save the model to the directory `dirname`.
set_estimator(estimators[, n_splits])	Specify the custom base estimators for cascade layers.
set_predictor(predictor)	Specify the custom predictor concatenated to deep forest.

class deepforest.**CascadeForestClassifier**(*n_bins=255*, *bin_subsample=200000*, *bin_type='percentile'*, *max_layers=20*, *criterion='gini'*, *n_estimators=2*, *n_trees=100*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *use_predictor=False*, *predictor='forest'*, *predictor_kwargs={}*, *backend='custom'*, *n_tolerant_rounds=2*, *delta=1e-05*, *partial_mode=False*, *n_jobs=None*, *random_state=None*, *verbose=1*)

Bases: `deepforest.cascade.BaseCascadeForest`, `sklearn.base.ClassifierMixin`

Implementation of the deep forest for classification.

Parameters

- **n_bins** (`int`, default=255) – The number of bins used for non-missing values. In addition

to the `n_bins` bins, one more bin is reserved for missing values. Its value must be no smaller than 2 and no greater than 255.

- **bin_subsample** (`int`, default=200,000) – The number of samples used to construct feature discrete bins. If the size of training set is smaller than `bin_subsample`, then all training samples will be used.
- **bin_type** (`{"percentile", "interval"}`, default= `"percentile"`) – The type of binner used to bin feature values into integer-valued bins.
  - If `"percentile"`, each bin will have approximately the same number of distinct feature values.
  - If `"interval"`, each bin will have approximately the same size.
- **max_layers** (`int`, default=20) – The maximum number of cascade layers in the deep forest. Notice that the actual number of layers can be smaller than `max_layers` because of the internal early stopping stage.
- **criterion** (`{"gini", "entropy"}`, default= `"gini"`) – The function to measure the quality of a split. Supported criteria are `gini` for the Gini impurity and `entropy` for the information gain. Note: this parameter is tree-specific.
- **n_estimators** (`int`, default=2) – The number of estimator in each cascade layer. It will be multiplied by 2 internally because each estimator contains a `RandomForestClassifier` and a `ExtraTreesClassifier`, respectively.
- **n_trees** (`int`, default=100) – The number of trees in each estimator.
- **max_depth** (`int`, default=None) – The maximum depth of each tree. `None` indicates no constraint.
- **min_samples_split** (`int`, default=2) – The minimum number of samples required to split an internal node.
- **min_samples_leaf** (`int`, default=1) – The minimum number of samples required to be at a leaf node.
- **use_predictor** (`bool`, default=False) – Whether to build the predictor concatenated to the deep forest. Using the predictor may improve the performance of deep forest.
- **predictor** (`{"forest", "xgboost", "lightgbm"}`, default= `"forest"`) – The type of the predictor concatenated to the deep forest. If `use_predictor` is False, this parameter will have no effect.
- **predictor_kwargs** (`dict`, default={}) – The configuration of the predictor concatenated to the deep forest. Specifying this will extend/overwrite the original parameters inherit from deep forest. If `use_predictor` is False, this parameter will have no effect.
- **backend** (`{"custom", "sklearn"}`, default= `"custom"`) – The backend of the forest estimator. Supported backends are `custom` for higher time and memory efficiency and `sklearn` for additional functionality.
- **n_tolerant_rounds** (`int`, default=2) – Specify when to conduct early stopping. The training process terminates when the validation performance on the training set does not improve compared against the best validation performance achieved so far for `n_tolerant_rounds` rounds.
- **delta** (`float`, default=1e-5) – Specify the threshold on early stopping. The counting on `n_tolerant_rounds` is triggered if the performance of a fitted cascade layer does not improve by `delta` compared against the best validation performance achieved so far.

- **partial_mode** (bool, default=False) – Whether to train the deep forest in partial mode. For large datasets, it is recommended to use the partial mode.
  - If True, the partial mode is activated and all fitted estimators will be dumped in a local buffer;
  - If False, all fitted estimators are directly stored in the memory.
- **n_jobs** (int or None, default=None) – The number of jobs to run in parallel for both *fit()* and *predict()*. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors.
- **random_state** (int or None, default=None) –
  - If int, random_state is the seed used by the random number generator;
  - If None, the random number generator is the RandomState instance used by np.random.
- **verbose** (int, default=1) – Controls the verbosity when fitting and predicting.
  - If <= 0, silent mode, which means no logging information will be displayed;
  - If 1, logging information on the cascade layer level will be displayed;
  - If > 1, full logging information will be displayed.

fit (*X*, *y*, *sample_weight=None*)
Build a deep forest using the training data.

Note: Deep forest supports two kinds of modes for training:

- **Full memory mode**, in which the training / testing data and all fitted estimators are directly stored in the memory.
- **Partial mode**, in which after fitting each estimator using the training data, it will be dumped in the buffer. During the evaluating stage, the dumped estimators are reloaded into the memory sequentially to evaluate the testing data.

By setting the partial_mode to True, the partial mode is activated, and a local buffer will be created at the current directory. The partial mode is able to reduce the running memory cost when training the deep forest.

Parameters

- **X** – The training data. Internally, it will be converted to np.uint8.
- **y** (numpy.ndarray of shape (n_samples,)) – The class labels of input samples.
- **sample_weight** (numpy.ndarray of shape (n_samples,), default=None) – Sample weights. If None, then samples are equally weighted.

predict_proba (*X*)
Predict class probabilities for X.

Parameters **X** – The input samples. Internally, its dtype will be converted to np.uint8.

Returns **proba** – The class probabilities of the input samples.

Return type numpy.ndarray of shape (n_samples, n_classes)

predict (*X*)
Predict class for X.

Parameters **X** – The input samples. Internally, its dtype will be converted to `np.uint8`.

Returns **y** – The predicted classes.

Return type `numpy.ndarray` of shape (n_samples,)

clean()
Clean the buffer created by the model.

get_estimator(*layer_idx*, *est_idx*, *estimator_type*)
Get estimator from a cascade layer in the deep forest.

Parameters

- **layer_idx** (`int`) – The index of the cascade layer, should be in the range `[0, self.n_layers_-1]`.
- **est_idx** (`int`) – The index of the estimator, should be in the range `[0, self.n_estimators]`.
- **estimator_type** (`{"rf", "erf", "custom"}`) – Specify the forest type.
  - If `rf`, return the random forest.
  - If `erf`, return the extremely random forest.
  - If `custom`, return the customized estimator, only applicable when using customized estimators in deep forest via *set_estimator()*.

Returns **estimator**

Return type Estimator with the given index.

get_layer_feature_importances(*layer_idx*)
Return the feature importances of `layer_idx`-th cascade layer.

Parameters **layer_idx** (`int`) – The index of the cascade layer, should be in the range `[0, self.n_layers_-1]`.

Returns **feature_importances_** – The impurity-based feature importances of the cascade layer. Notice that the number of input features are different between the first cascade layer and remaining cascade layers.

Return type `numpy.ndarray` of shape (n_features,)

Note:

- This method is only applicable when deep forest is built using the `sklearn` backend
- The functionality of this method is not available when using customized estimators in deep forest.

---

load(*dirname*)
Load the model from the directory `dirname`.

Parameters **dirname** (`str`) – The name of the input directory.

Note: The dumped model after calling `load_model()` is not exactly the same as the model before saving, because many objects irrelevant to model inference will not be saved.

---

save(*dirname='model'*)
Save the model to the directory `dirname`.

Parameters **dirname** (str, default="model") – The name of the output directory.

> **Warning:** Other methods on model serialization such as pickle or joblib are not recommended, especially when partial_mode is set to True.

set_estimator(*estimators*, *n_splits=5*)
Specify the custom base estimators for cascade layers.

Parameters

- **estimators** (list) – A list of your base estimators, will be used in all cascade layers.
- **n_splits** (int, default=5) – The number of folds, must be at least 2.

set_predictor(*predictor*)
Specify the custom predictor concatenated to deep forest.

Parameters **predictor** (object) – The instantiated object of your predictor.

5.3.2 CascadeForestRegressor

fit(X, y[, sample_weight])	Build a deep forest using the training data.
predict(X)	Predict regression target for X.
clean()	Clean the buffer created by the model.
get_estimator(layer_idx, est_idx, estimator_type)	Get estimator from a cascade layer in the deep forest.
get_layer_feature_importances(layer_idx)	Return the feature importances of layer_idx-th cascade layer.
load(dirname)	Load the model from the directory dirname.
save([dirname])	Save the model to the directory dirname.
set_estimator(estimators[, n_splits])	Specify the custom base estimators for cascade layers.
set_predictor(predictor)	Specify the custom predictor concatenated to deep forest.

class deepforest.**CascadeForestRegressor**(*n_bins=255*, *bin_subsample=200000*, *bin_type='percentile'*, *max_layers=20*, *criterion='mse'*, *n_estimators=2*, *n_trees=100*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *use_predictor=False*, *predictor='forest'*, *predictor_kwargs={}*, *backend='custom'*, *n_tolerant_rounds=2*, *delta=1e-05*, *partial_mode=False*, *n_jobs=None*, *random_state=None*, *verbose=1*)
Bases: deepforest.cascade.BaseCascadeForest, sklearn.base.RegressorMixin

Implementation of the deep forest for regression.

Parameters

- **n_bins** (int, default=255) – The number of bins used for non-missing values. In addition to the n_bins bins, one more bin is reserved for missing values. Its value must be no smaller than 2 and no greater than 255.
- **bin_subsample** (int, default=200,000) – The number of samples used to construct feature discrete bins. If the size of training set is smaller than bin_subsample, then all training samples will be used.

- **bin_type** ({"percentile", "interval"}, default= "percentile") – The type of binner used to bin feature values into integer-valued bins.
  - If "percentile", each bin will have approximately the same number of distinct feature values.
  - If "interval", each bin will have approximately the same size.
- **max_layers** (int, default=20) – The maximum number of cascade layers in the deep forest. Notice that the actual number of layers can be smaller than max_layers because of the internal early stopping stage.
- **criterion** ({"mse", "mae"}, default= "mse") – The function to measure the quality of a split. Supported criteria are mse for the mean squared error, which is equal to variance reduction as feature selection criterion, and mae for the mean absolute error.
- **n_estimators** (int, default=2) – The number of estimator in each cascade layer. It will be multiplied by 2 internally because each estimator contains a RandomForestRegressor and a ExtraTreesRegressor, respectively.
- **n_trees** (int, default=100) – The number of trees in each estimator.
- **max_depth** (int, default=None) – The maximum depth of each tree. None indicates no constraint.
- **min_samples_split** (int, default=2) – The minimum number of samples required to split an internal node.
- **min_samples_leaf** (int, default=1) – The minimum number of samples required to be at a leaf node.
- **use_predictor** (bool, default=False) – Whether to build the predictor concatenated to the deep forest. Using the predictor may improve the performance of deep forest.
- **predictor** ({"forest", "xgboost", "lightgbm"}, default= "forest") – The type of the predictor concatenated to the deep forest. If use_predictor is False, this parameter will have no effect.
- **predictor_kwargs** (dict, default={}) – The configuration of the predictor concatenated to the deep forest. Specifying this will extend/overwrite the original parameters inherit from deep forest. If use_predictor is False, this parameter will have no effect.
- **backend** ({"custom", "sklearn"}, default= "custom") – The backend of the forest estimator. Supported backends are custom for higher time and memory efficiency and sklearn for additional functionality.
- **n_tolerant_rounds** (int, default=2) – Specify when to conduct early stopping. The training process terminates when the validation performance on the training set does not improve compared against the best validation performance achieved so far for n_tolerant_rounds rounds.
- **delta** (float, default=1e-5) – Specify the threshold on early stopping. The counting on n_tolerant_rounds is triggered if the performance of a fitted cascade layer does not improve by delta compared against the best validation performance achieved so far.
- **partial_mode** (bool, default=False) – Whether to train the deep forest in partial mode. For large datasets, it is recommended to use the partial mode.
  - If True, the partial mode is activated and all fitted estimators will be dumped in a local buffer;
  - If False, all fitted estimators are directly stored in the memory.

- **n_jobs** (int or None, default=None) – The number of jobs to run in parallel for both *fit()* and *predict()*. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors.
- **random_state** (int or None, default=None) –
  - If int, random_state is the seed used by the random number generator;
  - If None, the random number generator is the RandomState instance used by np.random.
- **verbose** (int, default=1) – Controls the verbosity when fitting and predicting.
  - If <= 0, silent mode, which means no logging information will be displayed;
  - If 1, logging information on the cascade layer level will be displayed;
  - If > 1, full logging information will be displayed.

fit(*X*, *y*, *sample_weight=None*)
Build a deep forest using the training data.

Note: Deep forest supports two kinds of modes for training:

- **Full memory mode**, in which the training / testing data and all fitted estimators are directly stored in the memory.
- **Partial mode**, in which after fitting each estimator using the training data, it will be dumped in the buffer. During the evaluating stage, the dumped estimators are reloaded into the memory sequentially to evaluate the testing data.

By setting the partial_mode to True, the partial mode is activated, and a local buffer will be created at the current directory. The partial mode is able to reduce the running memory cost when training the deep forest.

Parameters

- **X** – The training data. Internally, it will be converted to np.uint8.
- **y** (numpy.ndarray of shape (n_samples,) or (n_samples, n_outputs)) – The target values of input samples.
- **sample_weight** (numpy.ndarray of shape (n_samples,), default=None) – Sample weights. If None, then samples are equally weighted.

predict(*X*)
Predict regression target for X.

Parameters **X** – The input samples. Internally, its dtype will be converted to np.uint8.

Returns **y** – The predicted values.

Return type numpy.ndarray of shape (n_samples,) or (n_samples, n_outputs)

clean()
Clean the buffer created by the model.

get_estimator(*layer_idx*, *est_idx*, *estimator_type*)
Get estimator from a cascade layer in the deep forest.

Parameters

- **layer_idx** (int) – The index of the cascade layer, should be in the range [0, self.n_layers_-1].
- **est_idx** (int) – The index of the estimator, should be in the range [0, self.n_estimators].
- **estimator_type** ({"rf", "erf", "custom"}) – Specify the forest type.
  - If rf, return the random forest.
  - If erf, return the extremely random forest.
  - If custom, return the customized estimator, only applicable when using customized estimators in deep forest via *set_estimator()*.

Returns estimator

Return type Estimator with the given index.

get_layer_feature_importances(*layer_idx*)

Return the feature importances of layer_idx-th cascade layer.

Parameters **layer_idx** (int) – The index of the cascade layer, should be in the range [0, self.n_layers_-1].

Returns **feature_importances_** – The impurity-based feature importances of the cascade layer. Notice that the number of input features are different between the first cascade layer and remaining cascade layers.

Return type numpy.ndarray of shape (n_features,)

Note:

- This method is only applicable when deep forest is built using the sklearn backend
- The functionality of this method is not available when using customized estimators in deep forest.

---

load(*dirname*)

Load the model from the directory dirname.

Parameters **dirname** (str) – The name of the input directory.

Note: The dumped model after calling load_model() is not exactly the same as the model before saving, because many objects irrelevant to model inference will not be saved.

---

save(*dirname='model'*)

Save the model to the directory dirname.

Parameters **dirname** (str, default="model") – The name of the output directory.

Warning: Other methods on model serialization such as pickle or joblib are not recommended, especially when partial_mode is set to True.

set_estimator(*estimators*, *n_splits=5*)

Specify the custom base estimators for cascade layers.

Parameters

- **estimators** (list) – A list of your base estimators, will be used in all cascade layers.

- **n_splits** (int, default=5) – The number of folds, must be at least 2.

set_predictor(*predictor*)
Specify the custom predictor concatenated to deep forest.

Parameters **predictor** (object) – The instantiated object of your predictor.

5.4 Parameters Tunning

This page contains parameters tuning guides for deep forest.

5.4.1 Better Accuracy

Increase model complexity

An intuitive way of improving the performance of deep forest is to increase its model complexity, below are some important parameters controlling the model complexity:

- `n_estimators`: Specify the number of estimators in each cascade layer.
- `n_trees`: Specify the number of trees in each estimator.
- `max_layers`: Specify the maximum number of cascade layers.

Using a large value of parameters above, the performance of deep forest may improve on complex datasets that require a larger model to perform well.

Add the predictor

In addition to increasing the model complexity, you can borrow the power of random forest or gradient boosting decision tree (GBDT), which can be helpful depending on the dataset:

- `use_predictor`: Decide whether to use the predictor concatenated to the deep forest.
- `predictor`: Specify the type of the predictor, should be one of `"forest"`, `"xgboost"`, `"lightgbm"`.

Please make sure that XGBoost or LightGBM are installed if you are going to use them as the predictor.

Tip: A useful rule on deciding whether to add the predictor is to compare the performance of deep forest with a standalone predictor produced from the training data. If the predictor consistently outperforms deep forest, then the performance of deep forest is expected to improve via adding the predictor. In this case, the augmented features produced from deep forest also facilitate training the predictor.

5.4.2 Faster Speed

Parallelization

Using parallelization is highly recommended because deep forest is naturally suited to it.

- `n_jobs`: Specify the number of workers used. Setting its value to an integer larger than 1 enables the parallelization. Setting its value to `-1` means all processors are used.

Fewer Splits

- `n_bins`: Specify the number of feature discrete bins. A smaller value means fewer splitting cut-offs will be considered, should be an integer in the range [2, 255].
- `bin_type`: Specify the binning type. Setting its value to `"interval"` enables less splitting cut-offs to be considered on dense intervals where the feature values accumulate.

Decrease model complexity

Setting parameters below to a smaller value decreases the model complexity of deep forest, and may lead to a faster speed on training and evaluating.

- `max_depth`: Specify the maximum depth of tree. `None` indicates no constraint.
- `min_samples_leaf`: Specify the minimum number of samples required to be at a leaf node. The smallest value is `1`.
- `n_estimators`: Specify the number of estimators in each cascade layer.
- `n_trees`: Specify the number of trees in each estimator.
- `n_tolerant_rounds`: Specify the number of tolerant rounds when handling early stopping. The smallest value is `1`.

Warning: Since deep forest automatically determines the model complexity according to the validation performance on the training data, setting parameters above to a smaller value may lead to a deep forest model with more cascade layers.

5.4.3 Lower Memory Usage

Partial Mode

- `partial_mode`: Decide whether to train and evaluate the model in partial mode. If set to `True`, the model will actively dump fitted estimators in a local buffer. As a result, **the memory usage of deep forest no longer increases linearly with the number of fitted cascade layers**.

In addition, decreasing the model complexity also pulls down the memory usage.

5.5 Experiments

5.5.1 Baseline

For all experiments, we used 5 popular tree-based ensemble methods as baselines. Details on the baselines are listed in the following table:

Name	Introduction
Random Forest	An efficient implementation of Random Forest in Scikit-Learn
HGBDT	Histogram-based GBDT in Scikit-Learn
XGBoost EXACT	The vanilla version of XGBoost
XGBoost HIST	The histogram optimized version of XGBoost
LightGBM	Light Gradient Boosting Machine

5.5.2 Environment

For all experiments, we used a single linux server. Details on the specifications are listed in the table below. All processors were used for training and evaluating.

OS	CPU	Memory
Ubuntu 18.04 LTS	Xeon E-2288G	128GB

5.5.3 Setting

We kept the number of decision trees the same across all baselines, while remaining hyper-parameters were set to their default values. Running scripts on reproducing all experiment results are available, please refer to this Repo.

5.5.4 Classification

Dataset

We have collected a number of datasets for both binary and multi-class classification, as listed in the table below. They were selected based on the following criteria:

- Publicly available and easy to use;
- Cover different application areas;
- Reflect high diversity in terms of the number of samples, features, and classes.

As a result, some baselines may fail on datasets with too many samples or features. Such cases are indicated by `N/A` in all tables below.

Name	# Training	# Testing	# Features	# Classes
ijcnn1	49,990	91,701	22	2
pendigits	7,494	3,498	16	10
letter	15,000	5,000	16	26
connect-4	67,557	20,267	126	3
sector	6,412	3,207	55,197	105
covtype	406,708	174,304	54	7
susy	4,500,000	500,000	18	2
higgs	10,500,000	500,000	28	2
usps	7,291	2,007	256	10
mnist	60,000	10,000	784	10
fashion mnist	60,000	10,000	784	10

Classification Accuracy

The table below shows the testing accuracy of each method, with the best result on each dataset **bolded**. Each experiment was conducted over 5 independently trials, and the average result was reported.

Name	RF	HGBDT	XGB EXACT	XGB HIST	LightGBM	Deep Forest
ijcnn1	98.07	98.43	98.20	98.23	**98.61**	98.16
pendigits	96.54	96.34	96.60	96.60	96.17	**97.50**
letter	95.39	91.56	90.80	90.82	88.94	**95.92**
connect-4	70.18	70.88	71.57	71.57	70.31	**72.05**
sector	85.62	N/A	66.01	65.61	63.24	**86.74**
covtype	73.73	64.22	66.15	66.70	65.00	**74.27**
susy	80.19	80.31	80.32	**80.35**	80.33	80.18
higgs	N/A	74.95	75.85	76.00	74.97	**76.46**
usps	93.79	94.32	93.77	93.37	93.97	**94.67**
mnist	97.20	98.35	98.07	98.14	**98.42**	98.11
fashion mnist	87.87	87.02	90.74	90.80	**90.81**	89.66

Runtime

Runtime in seconds reported in the table below covers both the training stage and evaluating stage.

Name	RF	HGBDT	XGB EXACT	XGB HIST	LightGBM	Deep Forest
ijcnn1	9.60	6.84	11.24	1.90	1.99	8.37
pendigits	1.26	5.12	0.39	0.26	0.46	2.21
letter	0.76	1.30	0.34	0.17	0.19	2.84
connect-4	5.17	7.54	13.26	3.19	1.12	10.73
sector	292.15	N/A	632.27	593.35	18.83	521.68
covtype	84.00	2.56	58.43	11.62	3.96	164.18
susy	1429.85	59.09	1051.54	44.85	34.40	1866.48
higgs	N/A	523.74	7532.70	267.64	209.65	7307.44
usps	9.28	8.73	9.43	5.78	9.81	6.08
mnist	590.81	229.91	1156.64	762.40	233.94	599.55
fashion mnist	735.47	32.86	1403.44	2061.80	428.37	661.05

Some observations are listed as follow:

- Histogram-based GBDT (e.g., `HGBDT`, `XGB HIST`, `LightGBM`) are typically faster mainly because decision tree in GBDT tends to have a much smaller tree depth;
- With the number of input dimensions increasing (e.g., on mnist and fashion-mnist), random forest and deep forest can be faster.

5.5.5 Regression

Dataset

We have also collected four datasets on univariate regression for a comparison on the regression problem.

Name	# Training	# Testing	# Features
wine	1,071	528	11
abalone	2,799	1,378	8
cpusmall	5,489	2,703	12
boston	379	127	13
diabetes	303	139	10

Testing Mean Squared Error

The table below shows the testing mean squared error of each method, with the best result on each dataset **bolded**. Each experiment was conducted over 5 independently trials, and the average result was reported.

Name	RF	HGBDT	XGB EXACT	XGB HIST	LightGBM	Deep Forest
wine	0.35	0.40	0.41	0.41	0.39	**0.34**
abalone	4.79	5.40	5.73	5.75	5.60	**4.66**
cpusmall	8.31	9.01	9.86	11.82	8.99	**7.15**
boston	**16.61**	20.68	20.61	19.65	20.27	19.87
diabetes	3796.62	4333.66	4337.15	4303.96	4435.95	**3431.01**

Runtime

Runtime in seconds reported in the table below covers both the training stage and evaluating stage.

Name	RF	HGBDT	XGB EXACT	XGB HIST	LightGBM	Deep Forest
wine	0.76	2.88	0.30	0.30	0.30	1.26
abalone	0.53	1.57	0.47	0.50	0.17	1.29
cpusmall	1.87	3.59	1.71	1.25	0.36	2.06
boston	0.70	1.75	0.19	0.22	0.20	1.45
diabetes	0.37	0.66	0.14	0.18	0.06	1.09

5.6 Report from Users

The page collects user reports on using deep forest. Thanks all of them for their nice work!

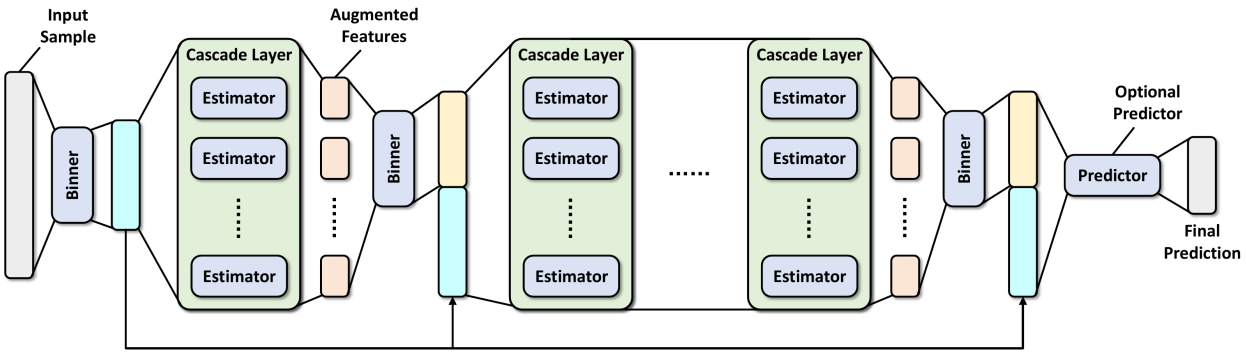
5.6.1 Competition

- 1st winning solution of the competition Insurance-Pricing-Game@AIcrowd: [Solution] | [Presentation]

5.6.2 Application

5.7 Model Architecture

This page introduces the model architecture, training stage, and evaluating stage of DF21. You may find this page helpful on understanding the meanings of different parameters listed in API Reference.



5.7.1 Component

This section presents the meanings of key components in DF21, along with associated parameters.

- `Binner`: The class used to reduce the number of splitting candidates for building decision trees.
  - `n_bins`, `bin_subsample`, `bin_type`
- `Estimator`: Base estimators used in cascades layer of DF21. Default estimators are RandomForestClassifier and ExtraTreesClassifier.
  - `n_trees`, `max_depth`, `min_samples_split`, `min_samples_leaf`, `criterion`, `backend`
- `Layer`: The cascade layer of DF21, which consists of multiple estimators.
  - `max_layers`, `n_estimators`
- `Predictor`: The optional predictor concatenated to the DF21 model.
  - `use_predictor`, `predictor`, `predictor_kwargs`

5.7.2 Training

The training stage of DF21 starts with discretizing feature-wise values of training samples into `n_bins` unique values, which is a commonly-used technique on accelerating building decision trees. After then, the first cascade layer in DF21 with `n_estimators` estimators is produced using the binned data (Notice that by default `n_estimators` would be multiplied by 2 internally). Furthermore, each estimator consists of `n_trees` decision trees that adopt the splitting criterion `criterion`, satisfying the constraints enforced by `max_depth` and `min_samples_leaf`.

After data binning and building the first cascade layer, DF21 enters the main training loop:

1. Bin the out-of-bag predictions of the previous cascade layer (denoted by augmented features in the figure above) using a newly-fitted `binner`;
2. Concatenate the augmented features to the binned training samples, serving as the new training data for the cascade layer to be built;
3. Build a new `layer` using the concatenated training data, following the same training protocols as that used to build the first cascade layer;
4. Get the out-of-bag predictions of the `layer` and estimate its generalization performance via out-of-bag estimation;
5. If the estimated performance is better than all previously-built layers, DF21 continues to build a new layer. Otherwise, the early-stopping procedure is triggered, and DF21 will terminate the training stage before reaching `max_layers` if the performance does not improve for `n_tolerant_rounds` rounds.

As an optional step, DF21 builds another predictor if `use_predictor` is set to `True`. This predictor takes the input the concatenated training data from the last cascade layer, and outputs the predicted class probabilities for classification problems, and predicted values for regression problems. One can use predictors like random forest or GBDT through setting `predictor`. Besides, you can better configure it through setting `predictor_kwargs`.

5.7.3 Evaluating

The evaluating stage follows the sequential structure of DF21. First, the testing samples are binned using the first `binner` and passed into the first `layer`. After then, DF21 sets the augmented features as the output of the current cascade layer, and bins it using the subsequent `binner`. After concatenating augmented features to the binned testing samples, DF21 moves to the next layer, util reaching the last cascade layer or the predictor.

5.8 Use Customized Estimators

The version v0.1.4 of `deepforest` has added the support on:

- using customized base estimators in cascade layers of deep forest
- using the customized predictor concatenated to the deep forest

The page gives a detailed introduction on how to use this new feature.

5.8.1 Instantiate the deep forest model

To begin with, you need to instantiate a deep forest model. Notice that some parameters specified here will be overridden by downstream steps. For example, if the parameter `use_predictor` is set to `False` here, whereas `set_predictor()` is called latter, then the internal attribute `use_predictor` will be altered to `True`.

```
from deepforest import CascadeForestClassifier
model = CascadeForestClassifier()
```

5.8.2 Instantiate your estimators

In order to use customized estimators in the cascade layer of deep forest, the next step is to instantiate the estimators and encapsulate them into a Python list:

```
n_estimators = 4  # the number of base estimators per cascade layer
estimators = [your_estimator(random_state=i) for i in range(n_estimators)]
```

Tip: You need to make sure that instantiated estimators in the list are with different random seeds if seeds are manually specified. Otherwise, they will have the same behavior on the dataset and make cascade layers less effective.

For the customized predictor, you only need to instantiate it, and there is no extra step:

```
predictor = your_predictor()
```

Deep forest will conduct internal checks to make sure that `estimators` and `predictor` are valid for training and evaluating. To pass the internal checks, the class of your customized estimators or predictor should at least implement methods listed below:

- `fit()` for training

- **[Classification]** `predict_proba()` for evaluating
- **[Regression]** `predict()` for evaluating

The name of these methods follow the convention in scikit-learn, and they are already implemented in a lot of packages offering scikit-learn APIs (e.g., XGBoost, LightGBM, CatBoost). Otherwise, you have to implement a wrapper on your customized estimators to make these methods callable.

5.8.3 Call `set_estimator()` and `set_predictor()`

The core step is to call `set_estimator()` and `set_predictor()` to override estimators used by default:

```
# Customized base estimators
model.set_estimator(estimators)

# Customized predictor
model.set_predictor(predictor)
```

`set_estimator()` has another parameter `n_splits`, which determines the number of folds of the internal cross-validation strategy. Its value should be at least `2`, and the default value is `5`. Generally speaking, a larger `n_splits` leads to better generalization performance. If you are confused about the effect of cross-validation here, please refer to the original paper for details on how deep forest adopts the cross-validation strategy to build cascade layers.

5.8.4 Train and Evaluate

Remaining steps follow the original workflow of deep forest.

```
model.train(X_train, y_train)
y_pred = model.predict(X_test)
```

Note: When using customized estimators via `set_estimator()`, deep forest adopts the cross-validation strategy to grow cascade layers. Suppose that `n_splits` is set to `5` when calling `set_estimator()`, each estimator will be repeatedly trained over five times to get full augmented features from a cascade layer. As a result, you may experience a drastic increase in running time and memory.

5.9 Contributors

Thanks goes to these wonderful people (emoji key):

This project follows the all-contributors specification. Contributions of any kind welcome!

5.10 Changelog

Badge	Meaning
[FEATURE]	Add something that cannot be achieved before.
[EFFICIENCY]	Improve the efficiency on the computation or memory.
[ENHANCEMENT]	Miscellaneous minor improvements.
[FIX]	Fix up something that does not work as expected.
[API CHANGE]	You will need to change the code to have the same effect.

5.10.1 Version 0.1.*

- [FEATURE] support the latest version of scikit-learn and drop support on python 3.6 (#115) @xuyxu
- [FEATURE] [API CHANGE] add support on `pandas.DataFrame` for `X` and `y` (#86) @IncubatorShokuhou
- [FIX] fix missing functionality of `_set_n_trees()` @xuyxu
- [FIX] [API CHANGE] add docstrings for parameter `bin_type` (#74) @xuyxu
- [FEATURE] [API CHANGE] recover the parameter `min_samples_split` (#73) @xuyxu
- [FIX] fix the breakdown under the corner case where no internal node exists (#70) @xuyxu
- [FEATURE] support python 3.9 (#69) @xuyxu
- [FIX] fix inconsistency on array shape for `CascadeForestRegressor` in customized mode (#67) @xuyxu
- [FIX] fix missing sample indices for parameter `sample_weight` in `KFoldWrapper` (#48) @xuyxu
- [FEATURE] [API CHANGE] add support on customized estimators (#48) @xuyxu
- [ENHANCEMENT] improve target checks for `CascadeForestRegressor` (#53) @chendingyan
- [FIX] fix the prediction workflow with only one cascade layer (#56) @xuyxu
- [FIX] fix inconsistency on predictor name (#52) @xuyxu
- [FEATURE] add official support for ManyLinux-aarch64 (#47) @xuyxu
- [FIX] fix accepted types of target for `CascadeForestRegressor` (#44) @xuyxu
- [FEATURE] [API CHANGE] add multi-output support for `CascadeForestRegressor` (#40) @Alex-Medium
- [FEATURE] [API CHANGE] add layer-wise feature importances (#39) @xuyxu
- [FEATURE] [API CHANGE] add scikit-learn backend (#36) @xuyxu
- [FEATURE] add official support for Mac-OS (#34) @T-Allen-sudo
- [FEATURE] [API CHANGE] support configurable criterion (#28) @tczhao
- [FEATURE] [API CHANGE] support regression prediction (#25) @tczhao
- [FIX] fix accepted data types on the `binner` (#23) @xuyxu
- [FEATURE] [API CHANGE] implement the `get_estimator()` method for efficient indexing (#22) @xuyxu
- [FEATURE] support class label encoding (#18) @NiMaZi
- [FEATURE] [API CHANGE] support sample weight in `fit()` (#7) @tczhao
- [FEATURE] [API CHANGE] configurable predictor parameter (#9) @tczhao
- [ENHANCEMENT] add base class `BaseEstimator` and `ClassifierMixin` (#8) @pjgao

5.11 Related Software

[1] RandomForestClassifier and RandomForestRegressor: Random Forest in Scikit-Learn.

[2] XGBoost: A Scalable Tree Boosting System.

[3] LightGBM: Light Gradient Boosting Machine.

[4] CatBoost: A Fast, Scalable, High Performance Gradient Boosting on Decision Trees Library.

[5] gcForest: An older version of Deep Forest.

INDEX