Background: Outlier detection (OD) is a key data mining task for identifying abnormal objects from general samples with numerous high-stake applications including fraud detection and intrusion detection. Due to the lack of ground truth labels, practitioners often have to build a large number of unsupervised models that are heterogeneous (i.e., different algorithms and hyperparameters) for further combination and analysis with ensemble learning, rather than relying on a single model. However, this yields severe scalability issues on high-dimensional, large datasets.

SUOD (Scalable Unsupervised Outlier Detection) is an acceleration framework for large-scale unsupervised heterogeneous outlier detector training and prediction. It focuses on three complementary aspects to accelerate (dimensionality reduction for high-dimensional data, model approximation for complex models, and execution efficiency improvement for taskload imbalance within distributed systems), while controlling detection performance degradation.

Since its inception in Sep 2019, SUOD has been successfully used in various academic researches and industry applications with more than 700,000 downloads, including PyOD [BZNL19] and IQVIA medical claim analysis.

SUOD is featured for:

- **Unified APIs, detailed documentation, and examples** for the easy use.
- **Optimized performance with JIT and parallelization** when possible, using numba and joblib.
- **Fully compatible with the models in PyOD.**
- **Customizable modules and flexible design**: each module may be turned on/off or totally replaced by custom functions.

Roadmap:

- Provide more choices of distributed schedulers (adapted for SUOD), e.g., batch sampling, Sparrow (SOSP’13), Pigeon (SoCC’19) etc.
- Enable the flexibility of selecting data projection methods.

API Demo:
from suod.models.base import SUOD

# initialize a set of base outlier detectors to train and predict on
base_estimators = [
    LOF(n_neighbors=5, contamination=contamination),
    LOF(n_neighbors=15, contamination=contamination),
    LOF(n_neighbors=25, contamination=contamination),
    HBOS(contamination=contamination),
    PCA(contamination=contamination),
    OCSVM(contamination=contamination),
    KNN(n_neighbors=5, contamination=contamination),
    KNN(n_neighbors=15, contamination=contamination),
    KNN(n_neighbors=25, contamination=contamination)]

# initialize a SUOD model with all features turned on
model = SUOD(base_estimators=base_estimators, n_jobs=6,  # number of workers
              rp_flag_global=True,  # global flag for random projection
              bps_flag=True,  # global flag for balanced parallel scheduling
              approx_flag_global=False,  # global flag for model approximation
              contamination=contamination)

model.fit(X_train)  # fit all models with X
model.approximate(X_train)  # conduct model approximation if it is enabled
predicted_labels = model.predict(X_test)  # predict labels
predicted_scores = model.decision_function(X_test)  # predict scores
predicted_probs = model.predict_proba(X_test)  # predict outlying probability


If you use SUOD in a scientific publication, we would appreciate citations to the following paper:
@inproceedings{zhao2021suod,
  title={SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection},
  author={Zhao, Yue and Hu, Xiyang and Cheng, Cheng and Wang, Cong and Wan, Changlin and Wang, Wen and Yang, Jianing and Bai, Haoping and Li, Zheng and Xiao, Cao and Wang, Y. and others},
  journal={Proceedings of Machine Learning and Systems},
  year={2021}
}

CHAPTER ONE

INSTALLATION

It is recommended to use `pip` for installation. Please make sure the latest version is installed, as SUOD is updated frequently:

```
pip install suod     # normal install
pip install --upgrade suod  # or update if needed
pip install --pre suod  # or include pre-release version for new features
```

Alternatively, you could clone and run setup.py file:

```
git clone https://github.com/yzhao062/suod.git
cd suod
pip install .
```

**Required Dependencies:**

- Python 3.5, 3.6, or 3.7
- joblib
- numpy>=1.13
- pandas (optional for building the cost forecast model)
- pyod
- scipy>=0.19.1
- scikit_learn>=0.19.1

**Note on Python 2:** The maintenance of Python 2.7 has stopped since January 1, 2020 (see official announcement). To be consistent with the Python change and SUOD’s dependent libraries, e.g., scikit-learn, **SUOD only supports Python 3.5+** and we encourage you to use Python 3.5 or newer for the latest functions and bug fixes. More information can be found at [Moving to require Python 3](#).
All three modules can be executed separately and the demo codes are in /examples/module_examples/:

- M1_RP: demo_random_projection.py
- M2_PSA: demo_pseudo_sup_approximation.py
- M3_BPS: demo_balanced_scheduling.py

For instance, you could navigate to /M1_RP/demo_random_projection.py. Demo codes all start with “demo_* .py”.

The examples for the full framework can be found under /examples folder; run “demo_base.py” for a simplified example. Run “demo_full.py” for a full example.

API Demo:

```python
from suod.models.base import SUOD

# initialize a set of base outlier detectors to train and predict on
base_estimators = [
    LOF(n_neighbors=5, contamination=contamination),
    LOF(n_neighbors=15, contamination=contamination),
    LOF(n_neighbors=25, contamination=contamination),
    HBOS(contamination=contamination),
    PCA(contamination=contamination),
    OCSVM(contamination=contamination),
    KNN(n_neighbors=5, contamination=contamination),
    KNN(n_neighbors=15, contamination=contamination),
    KNN(n_neighbors=25, contamination=contamination)]

# initialize a SUOD model with all features turned on
model = SUOD(base_estimators=base_estimators, n_jobs=6,  # number of workers
              rp_flag_global=True,  # global flag for random projection
              bps_flag=True,       # global flag for balanced parallel scheduling
              approx_flag_global=False,  # global flag for model approximation
              contamination=contamination)

model.fit(X_train)  # fit all models with X
model.approximate(X_train)  # conduct model approximation if it is enabled
predicted_labels = model.predict(X_test)  # predict labels
predicted_scores = model.decision_function(X_test)  # predict scores
predicted_probs = model.predict_proba(X_test)  # predict outlying probability
```
SUOD takes a similar approach of sklearn regarding model persistence. See model persistence for clarification.

In short, we recommend to use joblib or pickle for saving and loading SUOD models. See “examples/demo_model_save_load.py” for an example. In short, it is simple as below:

```python
from joblib import dump, load

# save the fitted model
dump(model, 'model.joblib')

# load the model
model = load('model.joblib')
```
The following APIs are the key ones for using SUOD:

- `suod.models.base.SUOD.fit()`: Fit estimator. \( y \) is optional for unsupervised methods.
- `suod.models.base.SUOD.approximate()`: Use supervised models to approximate unsupervised base detectors. Fit should be invoked first.
- `suod.models.base.SUOD.predict()`: Predict on a particular sample once the estimator is fitted.
- `suod.models.base.SUOD.predict_proba()`: Predict the probability of a sample is an anomaly once the estimator is fitted.
- `suod.models.base.SUOD.decision_function()`: Predict raw anomaly scores of \( X \) using the fitted detectors.
- `suod.models.base.SUOD.get_params()`: Get the parameters of the model.
- `suod.models.base.SUOD.set_params()`: Set the parameters of the model.
- Each base estimator can be accessed by calling `clf[i]` where \( i \) is the estimator index.

## 4.1 All Models

### 4.1.1 suod.models package

**Subpackages**

- `suod.models.saved_models` package

**Module contents**

**Submodules**

- `suod.models.base` module

Base class and functions of SUOD (Scalable Unsupervised Outlier Detection)
class suod.models.base.SUOD(base_estimators, contamination=0.1, n_jobs=None, rp_clf_list=None, rp_ng_clf_list=None, rp_flag_global=True, target_dim_frac=0.5, jl_method='basic', bps_flag=True, approx_clf_list=None, approx_ng_clf_list=None, approx_flag_global=True, approx_clf=None, cost_forecast_loc_fit=None, cost_forecast_loc_pred=None, verbose=False)

Bases: object

SUOD (Scalable Unsupervised Outlier Detection) is an acceleration framework for large scale unsupervised outlier detector training and prediction. The corresponding paper is under review in KDD 2020.

Parameters

- **base_estimators** *(list, length must be greater than 1)* – A list of base estimators. Certain methods must be present, e.g., fit and predict.

- **contamination** *(float in (0., 0.5), optional (default=0.1))* – The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function.

- **n_jobs** *(optional (default=1))* – The number of jobs to run in parallel for both fit and predict. If -1, then the number of jobs is set to the the number of jobs that can actually run in parallel.

- **rp_clf_list** *(list, optional (default=None))* – The list of outlier detection models to use random projection. The detector name should be consistent with PyOD.

- **rp_ng_clf_list** *(list, optional (default=None))* – The list of outlier detection models NOT to use random projection. The detector name should be consistent with PyOD.

- **rp_flag_global** *(bool, optional (default=True))* – If set to False, random projection is turned off for all base models.

- **target_dim_frac** *(float in (0., 1), optional (default=0.5))* – The target compression ratio.

- **jl_method** *(string, optional (default = 'basic'))* – The JL projection method:
  - "basic": each component of the transformation matrix is taken at random in N(0,1).
  - "discrete", each component of the transformation matrix is taken at random in {-1,1}.
  - "circulant": the first row of the transformation matrix is taken at random in N(0,1), and each row is obtained from the previous one by a one-left shift.
  - "toeplitz": the first row and column of the transformation matrix is taken at random in N(0,1), and each diagonal has a constant value taken from these first vector.

- **bps_flag** *(bool, optional (default=True))* – If set to False, balanced parallel scheduling is turned off.

- **approx_clf_list** *(list, optional (default=None))* – The list of outlier detection models to use pseudo-supervised approximation. The detector name should be consistent with PyOD.

- **approx_ng_clf_list** *(list, optional (default=None))* – The list of outlier detection models NOT to use pseudo-supervised approximation. The detector name should be consistent with PyOD.
• **approx_flag_global** *(bool, optional (default=True)) – If set to False, pseudo-supervised approximation is turned off.*

• **approx_clf** *(object, optional (default: sklearn RandomForestRegressor)) – The supervised model used to approximate unsupervised models.*

• **cost_forecast_loc_fit** *(str, optional) – The location of the pretrained cost prediction forecast for training.*

• **cost_forecast_loc_pred** *(str, optional) – The location of the pretrained cost prediction forecast for prediction.*

• **verbose** *(int, optional (default=0)) – Controls the verbosity of the building process.*

**approximate** *(X)*

Use the supervised regressor (random forest by default) to approximate unsupervised fitted outlier detectors.

**Parameters**

- **X** *(numpy array of shape (n_samples, n_features)) – The input samples. The same feature space of the unsupervised outlier detector will be used.*

**Returns**

- **self** – The estimator after with approximation.

**Return type** *object*

**decision_function** *(X)*

Predict raw anomaly scores of X using the fitted detectors.

The anomaly score of an input sample is computed based on the fitted detector. For consistency, outliers are assigned with higher anomaly scores.

**Parameters**

- **X** *(numpy array of shape (n_samples, n_features)) – The input samples. Sparse matrices are accepted only if they are supported by the base estimator.*

**Returns**

- **anomaly_scores** – The anomaly score of the input samples.

**Return type** *numpy array of shape (n_samples,)*

**fit** *(X)*

Fit all base estimators.

**Parameters**

- **X** *(numpy array of shape (n_samples, n_features)) – The input samples.*

**Returns**

- **self** – Fitted estimator.

**Return type** *object*

**get_params** *(deep=True)*

Get parameters for this estimator.


**Parameters**

- **deep** *(boolean, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.*

**Returns**

- **params** – Parameter names mapped to their values.

**Return type** *mapping of string to any*

**predict** *(X)*

Predict the class labels for the provided data.
Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns **outlier_labels** – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. 0 stands for inliers and 1 for outliers.

Return type *numpy array of shape (n_samples, n_estimators)*

```
predict_proba(X)
```
Predict the probability of a sample being outlier. Two approaches are possible:

1. simply use Min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.
2. use unifying scores, see [BKKSZ11].

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns **outlier_probability** – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. Return the outlier probability, ranging in [0,1].

Return type *numpy array of shape (n_samples,)*

```
set_params(**params)
```
Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it’s possible to update each component of a nested object.


Returns **self**

Return type *object*

suod.models.cost_predictor module

Cost predictor function for forecasting base model training and prediction cost.

```
suod.models.cost_predictor.build_cost_predictor(file_name, output_file, save_to_local=True)
```
Build cost predictor from the scratch. In general, this does not need to be used.

Parameters

- **file_name** (*string*) – The training table of algorithm performance.
- **output_file** –
- **save_to_local** –

```
suod.models.cost_predictor.indices_to_one_hot(data, nb_classes)
```
Convert an iterable of indices to one-hot encoded labels.

Parameters

- **data** (*list*) – The raw data.
- **nb_classes** (*int*) – The number of targeted classes.
suod.models.jl_projection module

Johnson–Lindenstrauss process. Part of the code is adapted from https://github.com/PTAug/jlt-python

suod.models.jl_projection.jl_fit_transform(X, objective_dim, method='basic')

Fit and transform the input data by Johnson–Lindenstrauss process. See [BJL84] for details.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **objective_dim** (*int*) – The expected output dimension.
- **method** (*string, optional (default = 'basic')*) – The JL projection method:
  - "basic": each component of the transformation matrix is taken at random in N(0,1).
  - "discrete": each component of the transformation matrix is taken at random in {-1,1}.
  - "circulant": the first row of the transformation matrix is taken at random in N(0,1), and each row is obtained from the previous one by a one-left shift.
  - "toeplitz": the first row and column of the transformation matrix is taken at random in N(0,1), and each diagonal has a constant value taken from these first vector.

Returns

- **X_transformed** (*numpy array of shape (n_samples, objective_dim)*) – The dataset after the JL projection.
- **jl_transformer** (*object*) – Transformer instance.

suod.models.jl_projection.jl_transform(X, jl_transformer)

Use the fitted transformer to conduct JL projection.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **jl_transformer** (*object*) – Fitted transformer instance.

Returns **X_transformed** – Transformed matrix.

Return type *numpy array of shape (n_samples, reduced_dimensions)*

suod.models.parallel_processes module

suod.models.parallel_processes.balanced_scheduling(time_cost_pred, n_estimators, n_jobs)

Conduct balanced scheduling based on the sum of rank, for both train and prediction. The algorithm will enforce the equal sum of ranks among workers.

Parameters

- **time_cost_pred** (*list*) – The list of time cost by the cost predictor. The length is equal to the number of base detectors.
- **n_estimators** (*int*) – The number of base estimators.
- **n_jobs** (*optional (default=1)*) – The number of jobs to run in parallel for both fit and predict. If -1, then the number of jobs is set to the number of cores.

Returns
• **n_estimators_list** *(list)* – The number of estimators for each worker

• **starts** *(list)* – The actual index of base detectors to be scheduled. For instance, starts[k, k+1] base detectors will be assigned to worker k.

• **n_jobs** – The actual usable number of jobs to run in parallel.

```python
suod.models.parallel_processes.cost_forecast_meta(clf, X, base_estimator_names)
```

Forecast model cost by pretrained cost estimator.

**Parameters**

• **clf** *(object, sklearn regressor)* – Random forest regressor trained to forecast model cost

• **X** *(numpy array of shape (n_samples, n_features))† – The input samples.

• **base_estimator_names** *(list of str)* – The list of outlier detection model names in the string format

**Returns** time_cost_pred

**Return type** numpy array of outlier detection model cost in seconds.

```python
suod.models.parallel_processes.indices_to_one_hot(data, nb_classes)
```

Convert an iterable of indices to one-hot encoded labels.

**Module contents**

**References**

**4.1.2 suod.utils package**

**Submodules**

```python
suod.utils.utility module
```

```python
suod.utils.utility.build_codes(base_estimators, clf_list, ng_clf_list, flag_global)
```

Core function for building codes for deciding whether enable random projection and supervised approximation.

**Parameters**

• **base_estimators** *(list, length must be greater than 1)* – A list of base estimators. Certain methods must be present, e.g., fit and predict.

• **clf_list** *(list)* – The list of outlier detection models to use a certain function. The detector name should be consistent with PyOD.

• **ng_clf_list** *(list)* – The list of outlier detection models to NOT use a certain function. The detector name should be consistent with PyOD.

• **flag_global** *(bool)* – The global flag to override the code build.

```python
suod.utils.utility.get_estimators(contamination=0.1)
```

Internal method to create a list of 600 base outlier detectors.

**Parameters** contamination *(float in (0., 0.5), optional (default=0.1))† – The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function.

**Returns** base_detectors – A list of initialized random base outlier detectors.
Return type  list

suod.utils.utility.get_estimators_small(\texttt{contamination=0.1})

Internal method to create a list of 600 base outlier detectors.

Parameters  \texttt{contamination} (float in (0., 0.5), optional (default=0.1)) –

The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function.

Returns  \texttt{base_detectors} – A list of initialized random base outlier detectors.

Return type  list

suod.utils.utility.raw_score_to_proba(\texttt{decision_scores, test_scores, method='linear'})

Utility function to convert raw scores to probability. The transformation can be either linear or using unify introduced in \cite{BKKSZ11}.

Parameters

\begin{itemize}
  \item \texttt{decision_scores} (numpy array of shape (n_samples,)) – The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.
  \item \texttt{test_scores} (numpy array of shape (n_samples,)) – The outlier scores of the test data to be converted. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.
  \item \texttt{method} (str, optional (default='linear')) – The transformation method, either ‘linear’ or ‘unify’
\end{itemize}

Returns  \texttt{outlier_probability} – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. Return the outlier probability, ranging in [0,1].

Return type  numpy array of shape (n_samples,)

Module contents
CHAPTER
FIVE

ABOUT US

5.1 Core Development Team

Yue Zhao (initialized the project in Jan 2020): Homepage
Xiyang Hu (initialized the project in Jan 2020): Homepage
Cheng Cheng (initialized the project in Jan 2020): Google Scholar

References
CHAPTER SIX

INDICES AND TABLES

- genindex
- modindex
- search
BIBLIOGRAPHY


PYTHON MODULE INDEX

S
suod.models, 16
suod.models.base, 11
suod.models.cost_predictor, 14
suod.models.jl_projection, 15
suod.models.parallel_processes, 15
suod.models.saved_models, 11
suod.utils, 17
suod.utils.utility, 16
INDEX

A
approximate() (suod.models.base.SUOD method), 13

B
balanced_scheduling() (in module suod.models.parallel_processes), 15
build_codes() (in module suod.utils.utility), 16
build_cost_predictor() (in module suod.models.cost_predictor), 14

C
cost_forecast_meta() (in module suod.models.parallel_processes), 16

d
decision_function() (suod.models.base.SUOD method), 13

F
fit() (suod.models.base.SUOD method), 13

g
get_estimators() (in module suod.utils.utility), 16
get_estimators_small() (in module suod.utils.utility), 17
get_params() (suod.models.base.SUOD method), 13

I
indices_to_one_hot() (in module suod.models.cost_predictor), 14
indices_to_one_hot() (in module suod.models.parallel_processes), 16

J
jl_fit_transform() (in module suod.models.jl_projection), 15
jl_transform() (in module suod.models.jl_projection), 15

M
module