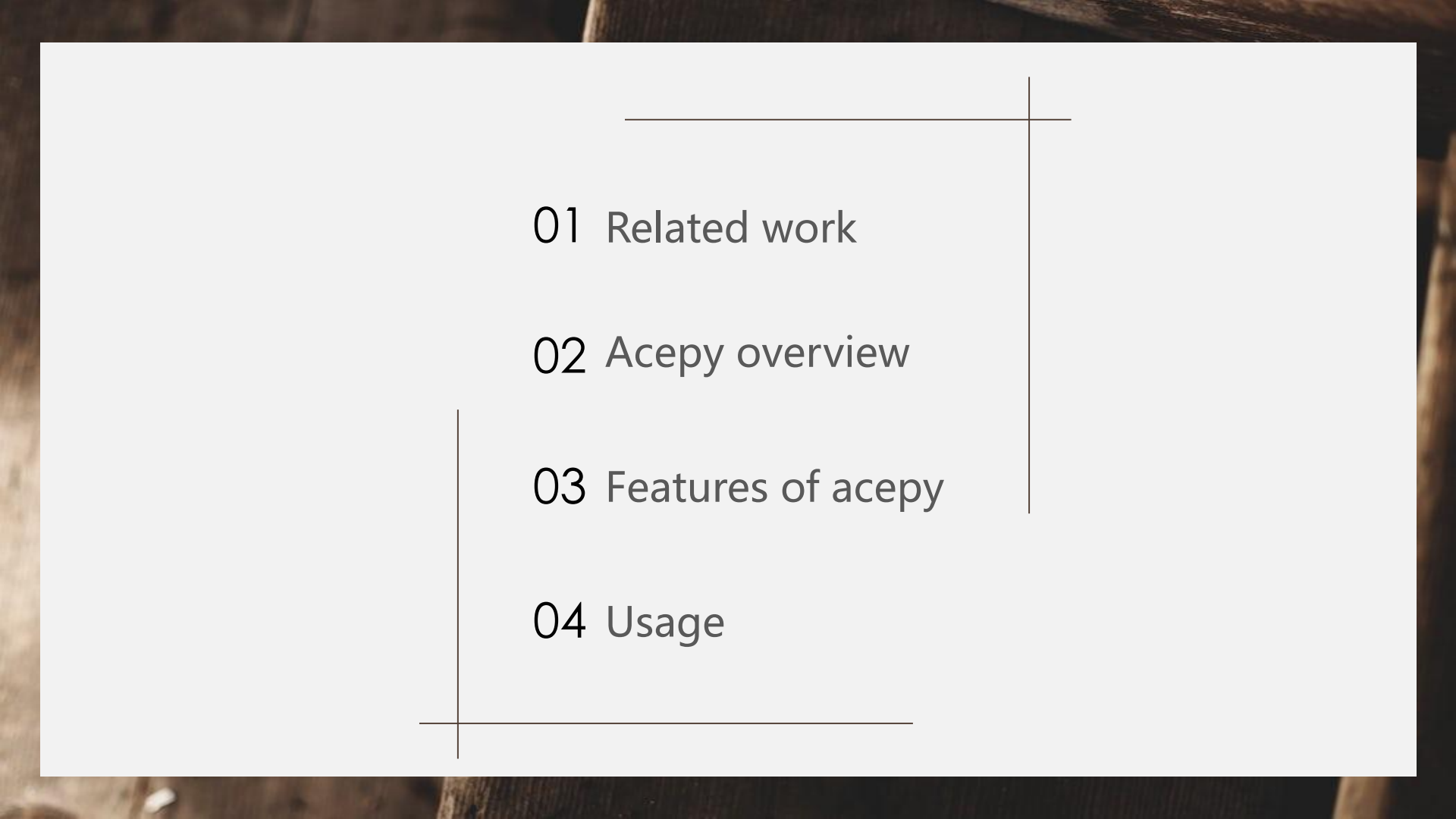
ACEPY

ACtive lEarning toolbox for PYthon

Developer : Ying-Peng Tang, Guo-Xiang Li

Supervisor: Sheng-Jun Huang

https://github.com/tangypnuaa/acepy



01
Related work

Related work

most of existing toolbox for active learning provide a framework or a high level encapsulation which limits the scope of application and lead to some constraints when using.

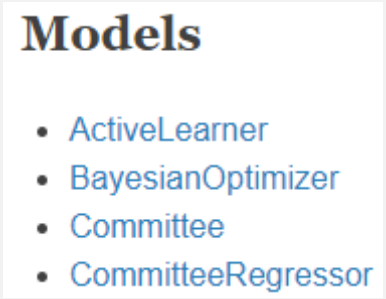
Libact (a framework of unified interfaces for active learning inspired by scikit-learn):

- Model must accord scikit-learn api.
- Must construct and pass a Dataset object defined in libact.
- Only support label querying.
- Own algorithm must inherent from the interface in libact.

https://github.com/google/active-learning

ModAL (create active learning workflows with nearly complete freedom):

- Model must accord scikit-learn api.
- Only implement classical query strategies.
- Only support label querying.
- Very limited tools



https://github.com/cosmic-cortex/modAL

Summary

Model limitation

The base model must accord scikit-learn api.

Scenario limitation

Support label querying only.

User limitation

The user-defined modules must match a signature (e.g., inherit from the base class).

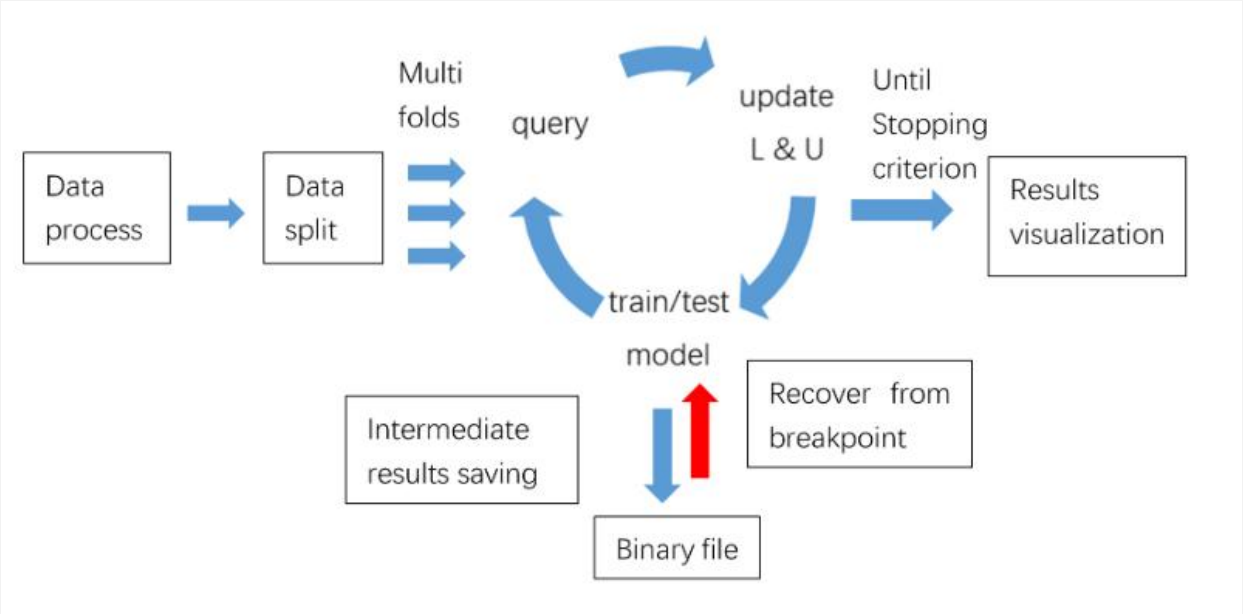
Support limitation

Most existed toolbox/framework only provide [Labeler, QueryStrategy] supporting.

02
Acepy overview

Acepy

In this work, we develop a toolbox called acepy to make users implement their own experiment in different settings without restrictions. It provides miscellaneous independent tool classes corresponding to active learning framework. Each of these tools can be used solely and replaced by users' own implementation freely.



Why independent tools?

1. Different users have different styles in programming.
2. Can implement different active learning scenario easily.
3. The logic of the program is absolutely clear and thus easy to debug.
4. Any modules can be substituted by users' own implementations freely for special usage.

Features of ACEPY

Model independent
There is no limitation of the model. You may use SVM in sklearn or deep model in tensorflow as you need.

Module independent
There is no framework limitation in our toolbox, using any tools are independent and optional.

Implement your own algorithm without inheriting anything
There are few limitations of the user-defined functions, such as the parameters or names.

Variant Settings supported
Noisy oracles, Multi-label, Cost effective, Feature querying, etc.

Powerful tools
intermediate results saving & loading; multi-threading; experiement result analysing, etc.

Modules in acepy

* Using acepy.data_manipulate to preprocess and split your data sets for experiments.

* Using acepy.query_strategy to invoke traditional and state-of-the-art methods.

* Using acepy.index.IndexCollection to manage your labeled indexes and unlabeled indexes.

* Using acepy.metric to calculate your model performances.

* Using acepy.experiment.state and acepy.experiment.state_io to save the intermediate results after each query and recover the program from the breakpoints.

* Using acepy.experiment.stopping_criteria to get some example stopping criteria.

* Using acepy.experiment.experiment_analyser to gathering, process and visualize your experiment results.

* Using acepy.oracle to implement clean, noisy, cost-sensitive oracles.

* Using acepy.utils.multi_thread to parallel your k-fold experiment.

03
Features of acepy

Features of ACEPY

Model independent
There is no limitation of the model. You may use SVM in sklearn or deep model in tensorflow as you need.

Module independent
There is no framework limitation in our toolbox, using any tools are independent and optional.

Implement your own algorithm without inheriting anything
There are few limitations of the user-defined functions, such as the parameters or names.

Variant Settings supported
Noisy oracles, Multi-label, Cost effective, Feature querying, etc.

Powerful tools
intermediate results saving & loading; multi-threading; experiement result analysing, etc.

Model independent

There is no limitation of the model. You may use SVM in sklearn or deep model in tensorflow as you need.

1. Provide the prediction matrix of unlabeled data:

```
predict_result = my_model.get_prediction_of_data(X[Uind.index])
select_ind = QBCStrategy.select_by_prediction_mat(unlabel_index=Uind,
                                                   predict=predict_result,
                                                   batch_size=1)
```

2. Use the default model to select instance:

```
select_ind = uncertainStrategy.select(Lind, Uind, batch_size=1, model=None)
```

3. If you are using a sklearn model:

```
model.fit(X[Lind.index], y[Lind.index])
select_ind = uncertainStrategy.select(Lind, Uind, batch_size=1, model=model)
```

Module independent

There is no framework limitation in the toolbox, using any tools are independent and optional.

1. Each module only implements necessary function.

2. Each function accepts multiple legal inputs.

```
radom_result = [[0.6, 0.7, 0.8, 0.9], [0.7, 0.7, 0.75, 0.85]]  # 2 folds, 4 queries for each fold.
uncertainty_result = [saver1, saver2] # 2 StateIO object for 2 folds experiments
from acepy.experiment import ExperimentAnalyser
analyser = ExperimentAnalyser(x_axis='num_of_queries')
analyser.add_method('random', radom_result)
analyser.add_method('uncertainty', uncertainty_result)
```

Powerful tools

1. Intermediate results manager

- Save intermediate results to files
- Recover workspace (label set and unlabel set) at any iterations
- Recover program from the breakpoint in case the program exits unexpectedly
- Print the active learning progress: current_iteration, current_mean_performance, current_cost, etc.

2. Multi-threading

- Parallel your k-fold experiments and print the status of each thread timely.

3. Plotting

- Extract the raw data of a list of acepy.experiment.StateIO object for plotting.
- Plot the experiment results of different strategies. (Include validity checking, interpolating, alignment, etc.)

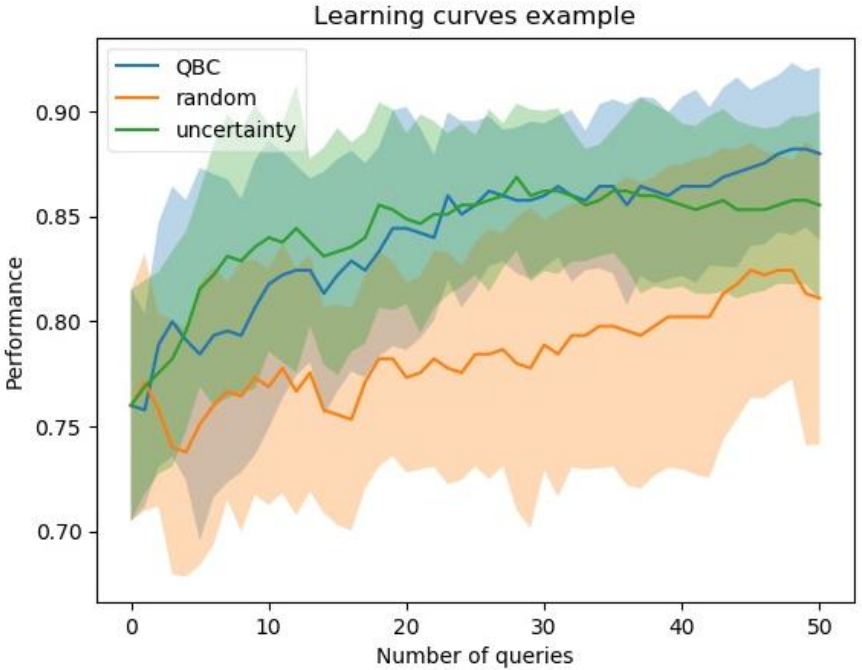
2. Multi-threading

```
acethread.start_all_threads()
```

You can get the progress information in the screen:

round	number_of_queries	time_elapse	performance (mean ± std)
0	13	00:00:07	0.966 ± 0.02
1	13	00:00:06	0.954 ± 0.05
2	15	00:00:06	0.956 ± 0.02
3	13	00:00:06	0.954 ± 0.04
4	13	00:00:06	0.959 ± 0.02
5	13	00:00:06	0.947 ± 0.04
6	14	00:00:06	0.951 ± 0.01
7	14	00:00:06	0.890 ± 0.07
8	13	00:00:06	0.915 ± 0.03
9	13	00:00:06	0.947 ± 0.01

3. Plotting



Variant Settings supported

Noisy oracles, Multi-label, Cost effective, Feature querying, etc.

1. Module variant

acepy.oracle.Oracle

acepy.oracle.OracleQueryFeatures

acepy.oracle.OracleQueryMultiLabel

acepy.oracle.Oracles

acepy.index.FeatureIndexCollection

acepy.index.IndexCollection

acepy.index.MultiLabelIndexCollection

2. Function variant

```
def __split_data_matrix(data_matrix=None, matrix_shape=None, test_ratio=0.3, initial_label_rate=0.05,
                        split_count=10, all_class=True, partially_labeled=False, saving_path='.'):
```

3. Query strategy supporting

QueryInstance	
Informative:	Uncertainty(least_confident, margin, entropy, distance_to_boundary), QBC(vote_entropy, KL_divergence), EER
Representative:	Random, GraphDensity (CVPR'12)
Mix:	QUIRE (TPAMI'14), BMDR (KDD'13), LAL (NIPS'17), SPAL (AAAI'19)

QueryNoisyOracle	
SingleOracle:	CEAL (IJCAI'17), ALC (ICML'11), Random, BestQuality, LowestCost
MultipleOracle:	IEthresh (KDD'09), Repeated
Crowdsource:	Majority Vote

QueryType	
Strategy	AURO (IJCAI'15)

3. Query strategy supporting

Multi-Label	
Instance-label pair	QUIRE (TPAMI'14), cosMAL (ICIP'15), Random
All labels	MMC (KDD'09), Adaptive (IJCAI'13), Random

QueryCostEffective	
Multi-Class	ACTIVE-CSL (IJCAI'05)
Multi-Label	HALC (IJCAI'18), Uncertainty, Random

LargeScale Active learning	
Strategy	Subsampling

Feature Querying	
Strategy	AFASMC (KDD'18), QBC, Stability (ICDM'13), Random

04
Usage

```python
import copy
from sklearn.datasets import load_iris
from acepy import ToolBox

X, y = load_iris(return_X_y=True)
acebox = ToolBox(X=X, y=y, query_type='AllLabels', saving_path='.')

# Split data
acebox.split_AL(test_ratio=0.3, initial_label_rate=0.1, split_count=10)

# Use the default Logistic Regression classifier
model = acebox.get_default_model()

# The cost budget is 50 times querying
stopping_criterion = acebox.get_stopping_criterion('num_of_queries', 50)

# Use pre-defined strategy
QBCStrategy = acebox.get_query_strategy(strategy_name='QueryInstanceQBC')
QBC_result = []

for round in range(10):
    # Get the data split of one fold experiment
    train_idx, test_idx, label_ind, unlab_ind = acebox.get_split(round)
    # Get intermediate results saver for one fold experiment
    saver = acebox.get_stateio(round)

    while not stopping_criterion.is_stop():
        # Select a subset of Uind according to the query strategy
        # Passing model=None to use the default model for evaluating the committees' disagreement
        select_ind = QBCStrategy.select(label_ind, unlab_ind, model=None, batch_size=1)
        label_ind.update(select_ind)
        unlab_ind.difference_update(select_ind)
```