# modAL: A modular active learning framework for Python

**Tivadar Danka** danka.tivadar@brc.mta.hu
*Biological Research Centre*
*Hungarian Academy of Sciences*
*Szeged, H6720, Hungary*

**Peter Horvath** horvath.peter@brc.mta.hu
*Biological Research Centre*
*Hungarian Academy of Sciences*
*Szeged, H6720, Hungary*



## Abstract

modAL is a modular active learning framework for Python, aimed to make active learning research and practice simpler. Its distinguishing features are (i) clear and modular object oriented design (ii) full compatibility with scikit-learn models and workflows. These features make fast prototyping and easy extensibility possible, aiding the development of real-life active learning pipelines and novel algorithms as well. modAL is fully open source, hosted on GitHub.[1] To assure code quality, extensive unit tests are provided and continuous integration is applied. In addition, a detailed documentation with several tutorials are also available for ease of use. The framework is available in PyPI and distributed under the MIT license.

**Keywords:** Active Learning, scikit-learn, Machine Learning, Python

## 1. Introduction

Upon learning patterns from data in real-life applications, labelling examples often consume significant time and money, which makes it infeasible to obtain large training sets. For example, sentiment analysis of texts requires extensive manual annotation, which costs expert time, see Koncz and Paralič (2013); Zhou et al. (2013). Another example is the optimization of black box functions, for which the evaluation is costly or derivatives are not available, see Shahriari et al. (2016). In these cases, active learning can be used to query labels for the most informative instances. modAL is an active learning framework for Python, designed with modularity, flexibility and extensibility in mind. Built on top of scikit-learn (Pedregosa et al. (2011); Buitinck et al. (2013)), it allows the rapid prototyping of active learning workflows with a large degree of freedom. It was designed to be easily extensible, allowing researchers to implement and test novel active learning strategies with minimal effort.

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

| Uncertainty sampling | Committee-based methods | Bayesian optimization | Other |
|---|---|---|---|
| Least confident | Kullback-Leibler divergence | Probability of improvement | Density-weighted methods |
| Max margin | Consensus entropy | Expected improvement | |
| Max entropy | Vote entropy | Upper confidence bounds | |

Table 1: Algorithms implemented in modAL.

## 2. Design principles and features

Our objective with modAL was to create an active learning library which takes advantage of the advanced features of Python and the extensive ecosystem of scikit-learn, making the implementation of complex workflows simple and intuitive. Specifically, modAL was designed with the following goals in mind.

1. **Modularity: separating and recombining parts of a workflow.** In general, an active learning workflow consists of a learning algorithm and a query strategy. In modAL, this is represented by the `ActiveLearner` class, for which these components are passed upon object creation. Learning algorithms can be used with query strategies in any combination, making rapid prototyping possible.

2. **Extensibility: simple customization of parts.** In a modAL active learning workflow, a query strategy is simply a function, given to the object representing the active learning algorithm upon initialization. Implementing custom query strategies can be done without understanding class structures or modAL internals. Thus it requires minimal effort, allowing researchers to easily test novel strategies and compare them with existing ones. In addition, function factories are provided to simplify assembling a query strategy from parts.

3. **Flexibility: compatibility with the scikit-learn ecosystem.** scikit-learn is one of the most popular machine learning tools in Python, used by researchers and practicioners as well. modAL is built on top of it, allowing the use any of its classifier and regressor algorithms in active learning pipelines. Objects in modAL also follow the scikit-learn API, making it possible to insert them into already existing workflows. For instance, scikit-learn functions implementing k-fold cross-validation work on modAL objects as well. For details on the scikit-learn API, see Buitinck et al. (2013).

modAL supports pool-based sampling and stream-based sampling scenarios as well. Along with classification tasks, regression is also supported in both single and multiple hypothesis settings. In addition, Bayesian optimization algorithms are available. The implemented algorithms are summarized in Table 1.

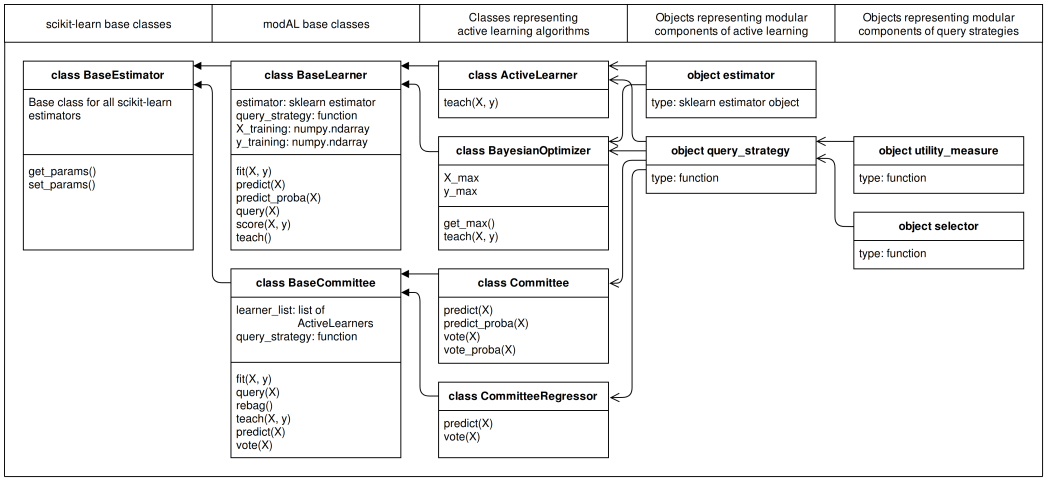


Figure 1: The structure of modAL.

## 3. Classes and interfaces

For modularity and easy extensibility, active learning workflows are abstracted and represented by the `ActiveLearner`, `BayesianOptimizer`, `Committee` and `CommitteeRegressor` classes. All classes inherit from the `sklearn.base.BaseEstimator` class. `ActiveLearner` serves as an abstract model for general active learning algorithms, while `Committee` and `CommitteeRegressor` implements committee-based strategies. Bayesian optimization algorithms are represented by `BayesianOptimizer`. All classes require a learner and a query strategy upon initialization. In the case of `ActiveLearner` and `BayesianOptimizer`, the learner is an arbitrary object implementing the scikit-learn API, while the for `Committee` and `CommitteeRegressor`, a list of `ActiveLearner` instances must be provided. Again, the query function can be factored into two functions: one calculating the utility for each instance and one selecting the instances to be queried based upon the utility score. The structure is summarized in Figure 1. The use of ActiveLearner is demonstrated below.

```
from modAL.models import ActiveLearner
from sklearn.ensemble import RandomForestClassifier

# initializing the learner
learner = ActiveLearner(RandomForestClassifier())

# training
learner.fit(X_training, y_training)

# query for labels
query_idx, query_inst = learner.query(X_pool)
# ...obtaining new labels from the Oracle...
# supply label for queried instance
learner.teach(X_pool[query_idx], y_new)
```

## 4. Comparison with other libraries

To assess the features of modAL, a comparison between libraries is provided. We compare modAL to acton[2], alp[3] and libact[4] (Yang et al. (2017)) in Tables 2, 3. The comparison is made with respect to supported algorithms, design and support.

| | pool | stream | regression | committee | metalearning | Bayesian optimization |
|---|---|---|---|---|---|---|
| modAL | ✓ | ✓ | ✓ | ✓ | X | ✓ |
| acton | ✓ | X | ✓ | ✓ | X | X |
| alp | ✓ | X | X | ✓ | X | X |
| libact | ✓ | X | X | ✓ | ✓ | X |

Table 2: Comparison of libraries with respect to supported algorithms

| | sklearn model usability | follows sklearn API | actively maintained | Python version | documentation, tutorials |
|---|---|---|---|---|---|
| modAL | ✓ | ✓ | ✓ | 3 | ✓ |
| acton | with adapters | ✓ | ✓ | 3 | ✓ |
| alp | ✓ | X | X | 2, 3 | X |
| libact | with adapters | X | ✓ | 2, 3 | ✓ |

Table 3: Comparison of libraries with respect to design and support

## 5. Availability

The framework is fully open-source, hosted on GitHub.[5] Besides the core features, detailed documentation and a wealth of examples and tutorials are available at the project website[6], making active learning accessible for a wide range of users. To assure code quality, extensive unit tests are provided and continuous integration is applied using Travis-CI. modAL is also available from PyPI.


## Acknowledgments

T.D. and P.H. acknowledges support from the European Regional Development Funds (GINOP-2.3.2-15-2016-00001, GINOP-2.3.2-15-2016-00037).


2. https://github.com/chengsoonong/acton
3. https://github.com/davefernig/alp
4. https://github.com/ntucllab/libact
5. https://github.com/cosmic-cortex/modAL
6. https://cosmic-cortex.github.io/modAL

## References

L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

P. Koncz and J. Paralič. Active learning enhanced document annotation for sentiment analysis. In A. Cuzzocrea, C. Kittl, D. E. Simos, E. Weippl, and L. Xu, editors, *Availability, Reliability, and Security in Information Systems and HCI*, pages 345–353, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

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

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, Jan 2016. ISSN 0018-9219. doi: 10.1109/JPROC.2015.2494218.

Y.-Y. Yang, S.-C. Lee, Y.-A. Chung, T.-E. Wu, S.-A. Chen, and H.-T. Lin. libact: Pool-based active learning in Python. Technical report, 2017. arXiv: `https://arxiv.org/abs/1710.00379`.

S. Zhou, Q. Chen, and X. Wang. Active deep learning method for semi-supervised sentiment classification. *Neurocomputing*, 120:536–546, 2013.