yall Documentation

Release 0.0.1

Jake Vasilakes

Nov 04, 2019

API:

CHAPTER 1

yall.ActiveLearningModel

class yall.activelearning.**ActiveLearningModel**(*classifier*, *query_strategy*, *eval_metric='auc'*, *U_proportion=0.9*, *init_L='random'*, *random_state=None*)

Bases: object

Parameters

- **classifier** (*sklearn.base.BaseEstimator*) – Classifier to build the model.
- **query_strategy** (*QueryStrategy*) – QueryStrategy instance to use.
- **eval_metric** (*str*) – One of "auc", "accuracy".
- **U_proportion** (*float*) – proportion of training data to be assigned the unlabeled set.
- **init_L** (*str*) – How to initialize L: "random" or "LDS".
- **random_state** (*int*) – Sets the random_state parameter of train_test_split.

partial_train(*new_x*, *new_y*)

Given a subset of training examples, calls partial_fit.

Parameters

- **new_x** (*numpy.ndarray*) – Feature array.
- **new_y** (*numpy.ndarray*) – Label array.

prepare_data(*train_X*, *test_X*, *train_y*, *test_y*)

Splits data into unlabeled, labeled, and test sets according to self.U_proportion.

Parameters

- **train_X** (*np.array*) – Training data features.
- **test_X** (*np.array*) – Test data features.
- **train_y** (*np.array*) – Training data labels.
- **test_y** (*np.array*) – Test data labels.

run (*train_X*, *test_X*, *train_y*, *test_y*, *ndraws=None*, *verbose=0*)
Run the active learning model. Saves AUC scores for each sampling iteration.

Parameters

- **train_X** (*np.array*) – Training data features.
- **test_X** (*np.array*) – Test data features.
- **train_y** (*np.array*) – Training data labels.
- **test_y** (*np.array*) – Test data labels.
- **ndraws** (*int*) – Number of times to query the unlabeled set. If None, query entire unlabeled set.
- **verbose** (*int*) – If > 0, print information.

Returns AUC scores for each sampling iteration.

Return type numpy.ndarray(shape=(ndraws,))

score()
Computes the performance of the current classifier according to self.eval_metric.

Returns performance score

Return type float

train()
Trains the classifier on L.

update_labels()
Gets the chosen index from the query strategy, adds the corresponding data point to L and removes it from U. Logs which instance is picked from U.

Returns chosen x and y, for use with partial_train()

Return type tuple(numpy.ndarray, numpy.ndarray)

CHAPTER 2

yall.containers module

class yall.containers.**Choice**(*X*, *y*, *score*)
Bases: tuple

Create new instance of Choice(X, y, score)

X
Alias for field number 0

score
Alias for field number 2

y
Alias for field number 1

class yall.containers.**Data**(*X=None*, *y=None*)
Bases: object

Data container object to hold features and labels.

CHAPTER 3

yall.initializations module

class yall.initializations.**CentralityMeasure**(*X*, *k*)
Bases: object

$score(x) = \frac{1}{k-1} \sum_{x_j \in NN(x_i)} \omega(x_i, x_j)$

$NN(x)$: The k nearest neighbors of x.

ω: A weight method.

centrality()

find_centers(*n=50*)

weight(*i*, *j*)
Computes the weight between nodes i and j according to the graph matrix.

class yall.initializations.**ClosenessCentrality**(*X*, *k=30*)
Bases: *yall.initializations.CentralityMeasure*

$\omega(x_i, x_j) = \frac{1}{dist(x_i, x_j)}$

weight(*i*, *j*)
Computes the weight between nodes i and j according to the graph matrix.

class yall.initializations.**DegreeCentrality**(*X*, *k=30*)
Bases: *yall.initializations.CentralityMeasure*

$\omega(x_i, x_j) = \delta_{ij}$

weight(*i*, *j*)
Computes the weight between nodes i and j according to the graph matrix.

class yall.initializations.**EigenvectorCentrality**(*X*, *k=30*, *n='auto'*)
Bases: *yall.initializations.CentralityMeasure*

We solve for the eigenvalues λ of the adjecency matrix A

$Ax = \lambda x$

The nodes with the highest eigenvalues λ are the most central.

centrality()

class yall.initializations.**FacilityLocation**(*X*, *k=30*, *solver='GUROBI'*)
Bases: *yall.initializations.SetCover*

This is a simplified version of the uncapacitated facility location problem in which there is no cost to open a facility. Customers are data points and facilities are centers. The cost to ship from a facility to a customer is computed as the distance between them in a k nearest neighbor graph.

I : Set of candidate center locations.

J : Set of data points.

N.B. In this case $I = J$ as each data point is a potential center.

M : Maximum number of centers.

$$y_{ij} = \begin{cases} 1 & \text{if center } i \text{ covers data point } j \\ 0 & \text{otherwise} \end{cases}$$

$D_{ij} =$ distance between center i and data point j

$\epsilon =$ number of permissable outliers

minimize $\sum_{i \in I} \sum_{j \in J} D_{ij} y_{ij}$

subject to

$\sum_i max_j \; y_{ij} \leq M$

$\sum_{ij} y_{ij} = \; |J| - \; \epsilon$

$y_{ij} \in \{0, 1\} \;\; \forall i \in I, j \in J$

find_centers(*n=50*, *epsilon='auto'*)

class yall.initializations.**GreedySetCover**(*X*, *k=30*)
Bases: *yall.initializations.SetCover*

Given a set of partial covers S of X, greedily search for a subset of them, indexed by I such that $\bigcup_{i \in I} S_i \; = X$

find_centers(*n=50*)

class yall.initializations.**LDSCentrality**(*X*, *k=30*)
Bases: *yall.initializations.CentralityMeasure*

$\omega(x_i, x_j) = \mid NN(x_i) \cap NN(x_j) \mid$

weight(*i*, *j*)
Computes the weight between nodes i and j according to the graph matrix.

class yall.initializations.**SetCover**(*X*, *k=30*)
Bases: object

find_centers(*n=50*)

CHAPTER 4

yall.querystrategies

class yall.querystrategies.**UncertaintySampler**(*model_change=False*)

Bases: yall.querystrategies.QueryStrategy

choose(*scores*)

Parameters **scores** (*numpy.ndarray*) – Output of self.score()

Returns Index of chosen example.

Return type int

model_change_wrapper(*score_func*)

Model change wrapper around the scoring function. See doc for __score() above for usage insructions.

$score_{mc}(X) = score(X;t) - w_o score(X;t-1)$

$score(X,t)$: The score at time t

$w_o = \frac{1}{|L|}$

Parameters **score_func** (*function*) – Scoring function to wrap.

Returns Wrapped scoring function.

Return type function

class yall.querystrategies.**CombinedSampler**(*qs1=None*, *qs2=None*, *beta=1*, *choice_metric=<function argmax>*)

Bases: yall.querystrategies.QueryStrategy

Allows one sampler's scores to be weighted by anothers according to the equation:

$score(x) = score_{qs1}(x) \times score_{qs2}(x)^{\beta}$

Assumes $x^* = argmax(score)$

Parameters

- **qs1** (*QueryStrategy*) – Main query strategy.
- **qs2** (*QueryStrategy*) – Query strategy to use as weight.

- **beta** (*float*) – Scale factor for score_qs2.
- **choice_metric** (*function*) – Function that takes a 1d np.array and returns a chosen index.

choose(*scores*)
Returns the example with the "best" score according to self.choice_metric.

score(**args*)
Computes the combined scores from qs1 and qs2. :returns: scores :rtype: numpy.ndarray

class yall.querystrategies.**DistDivSampler**(*qs1=None*, *qs2=None*, *lam=0.5*, *choice_metric=<function argmax>*)
Bases: yall.querystrategies.QueryStrategy

Combined sampling method as in "Active learning for clinical text classification: is it better than random sampling?"

$x^* = argmin_x(\lambda score_{qs1}(x) + (1 - \lambda) score_{qs2}(x))$

Parameters

- **qs1** (*QueryStrategy*) – Uncertainty sampling query strategy.
- **qs2** (*QueryStrategy*) – Representative sampling query strategy.
- **lambda** (*float*) – Query strategy weight [0,1] or "dynamic".
- **choice_metric** (*function*) – Function that takes a 1d np.array and returns a chosen index.

choose(*scores*)
Returns the example with the "best" score according to self.choice_metric.

score(**args*)
Computes the combined scores from qs1 and qs2. :returns: scores :rtype: numpy.ndarray

class yall.querystrategies.**Random**
Bases: yall.querystrategies.QueryStrategy

Random query strategy. Equivalent to passive learning.

choose(*scores*)
Picks an index at random. :param numpy.ndarray scores: Output of self.score() :returns: Index of chosen example. :rtype: int

score(**args*)
In the random case, just output the indices.

class yall.querystrategies.**SimpleMargin**
Bases: yall.querystrategies.QueryStrategy

Finds the example x that is closest to the separating hyperplane.

$x^* = argmin_x |f(x)|$

choose(*scores*)
Returns the example with the shortest distance to the hyperplane. In the multiclass case, his will return the row index of the example with the smallest absolute distance to any hyperplane. Could be modified to choose the smallest average distance to all hyperplanes. :param numpy.ndarray scores: Output of self.score() :returns: Index of chosen example. :rtype: int

score(**args*)
Computes distances to the hyperplane for each member of the unlabeled set.

class yall.querystrategies.**Margin**
Bases: yall.querystrategies.QueryStrategy

Margin Sampler. Chooses the member from the unlabeled set with the smallest difference between the posterior probabilities of the two most probable class labels.

$x^* = argmin_x P(\hat{y_1}|x) - P(\hat{y_2}|x)$

where $\hat{y_1}$ is the most probable label and $\hat{y_2}$ is the second most probable label.

choose(*scores*)
Returns the example with the smallest difference between the two most probable class labels. :param numpy.ndarray scores: Output of self.score() :returns: Index of chosen example. :rtype: int

score(**args*)
Computes the difference between posterior probability estimates for the top two most probable labels. :returns: Posterior probability differences. :rtype: numpy.ndarray

class yall.querystrategies.**Entropy**(*model_change=False*)
Bases: *yall.querystrategies.UncertaintySampler*

Entropy Sampler. Chooses the member from the unlabeled set with the greatest entropy across possible labels.

$x^* = argmax_x - \sum_i P(y_i|x) \times log_2(P(y_i|x))$

class yall.querystrategies.**LeastConfidence**(*model_change=False*)
Bases: *yall.querystrategies.UncertaintySampler*

Least confidence (uncertainty sampling). Chooses the member from the unlabeled set with the greatest uncertainty, i.e. the greatest posterior probability of all labels except the most likely one.

$x^* = argmax_x 1 - P(\hat{y}|x)$

where $\hat{y} = argmax_y P(y|x)$

class yall.querystrategies.**LeastConfidenceBias**(*model_change=False*)
Bases: *yall.querystrategies.UncertaintySampler*

Least confidence with bias. This is the same as least confidence, but moves the decision boundary according to the current class distribution.

$$x^* = \begin{cases} \frac{P(\hat{y}|x)}{P_{max}}, \text{ if } P(\hat{y}|x) < P_{max} \\ \frac{1-P(\hat{y}|x)}{P_{max}}, \text{ otherwise} \end{cases}$$

where

$P_{max} = mean(0.5, 1 - pp)$ and *pp* is the percentage of positive examples in the labeled set.

class yall.querystrategies.**LeastConfidenceDynamicBias**(*model_change=False*)
Bases: *yall.querystrategies.UncertaintySampler*

Least confidence with dynamic bias. This is the same as least confidence with bias, but the bias also adjusts for the relative sizes of the labeled and unlabeled data sets.

$$x^* = \begin{cases} \frac{P(\hat{y}|x)}{P_{max}}, \text{ if } P(\hat{y}|x) < P_{max} \\ \frac{1-P(\hat{y}|x)}{P_{max}}, \text{ otherwise} \end{cases}$$

where

$P_{max} = (1 - pp)w_b + 0.5w_y$

pp is the percentage of positive examples in the labeled set.

$w_u = \frac{|L|}{U_0}$ and U_0 is the initial unlabeled set.

$w_b = 1 - w_u$

class yall.querystrategies.**DistanceToCenter**(*metric='euclidean'*)
Bases: yall.querystrategies.QueryStrategy

Distance to Center sampling. Measures the distance of each point to the average x (center) in the labeled data set and computes the similarity using the equation below.

$x* = argmin_x \frac{1}{1+dist(x,x_L)}$

where dist(A, B) is the distance between vectors A and B.

x_L is the mean vector in L (i.e. L's center).

Parameters **metric** (*str*) – Distance metric to use. See spd.cdist doc for available metrics.

choose(*scores*)
Returns the example with the lowest similarity to the average x in L. :param numpy.ndarray scores: Output of self.score() :returns: Index of chosen example. :rtype: int

score(**args*)

Returns Distances.

Return type numpy.ndarray

class yall.querystrategies.**MinMax**(*metric='euclidean'*)
Bases: yall.querystrategies.QueryStrategy

Finds the exmaple x in U that has the maximum smallest distance to every point in L. Ensures representative coverage of the dataset.

$x^* = argmax_{x_i}(min_{x_j} dist(x_i, x_j))$

where $x_i \in U$, $x_j \in L$, dist(.) is the given distance metric.

Parameters **metric** (*str*) – Distance metric to use. See the spd.cdist doc for available metrics.

choose(*scores*)
Returns the examples with the greatest minimum distance to every other x in L. :param numpy.ndarray scores: Output of self.score() :returns: Index of chosen example. :rtype: int

score(**args*)

Computes minimum distance between each member of unlabeled_x and each member of labeled_x.

Returns Minimum distances from each unlabeled_x to each labeled_x.

Return type numpy.ndarray

class yall.querystrategies.**Density**(*metric='euclidean'*)
Bases: yall.querystrategies.QueryStrategy

Finds the example x in U that has the greatest average distance to every other point in U.

$x^* = argmin_x \frac{1}{U} \sum_{u=1} \frac{1}{1+dist(x,x_u)}$

Parameters **metric** (*str*) – Distance metric to use. See spd.cdist doc for available metrics.

choose(*scores*)
Returns the example with the lowest similarity to the average x in U. :param numpy.ndarray scores: Output of self.score() :returns: Index of chosen example. :rtype: int

score(**args*)
Computes average distance between each member of U and each other member of U. :returns: Minimum distances from each point in U to each other point. :rtype: numpy.ndarray

CHAPTER 5

yall.utils module

yall.utils.**compute_alc**(*aucs*, *normalize=True*)

Compute the normalized Area under the Learning Curve (ALC) for a set of AUCs.

param aucs: np.array of AUC values. param normalize: Whether to normalize the ALC, default: True.

yall.utils.**plot_learning_curve**(*aucs*, *L_init*, *L_end*, *title='ALC'*, *eval_metric='auc'*, *saveto=None*)

Plots the learning curve for a set of AUCs.

param aucs: np.array of AUC values. param L_init: The initial size of the labeled set. param L_end: The final size of the labeled set. param title: The title of this plot. param saveto: Filename to which to save this plot instead of showing it.

CHAPTER 6

yall.datasets

6.1 yall.datasets.base module

class yall.datasets.base.**Bunch**(*data*, *target*, *filenames*)

Bases: tuple

Create new instance of Bunch(data, target, filenames)

data

Alias for field number 0

filenames

Alias for field number 2

target

Alias for field number 1

yall.datasets.base.**load_dexter**()

yall.datasets.base.**load_spect**()

yall.datasets.base.**load_spectf**()

CHAPTER 7

Prerequisites

- Python 3
- NumPy
- SciPy
- scikit-learn

7.1 Installation

Clone or download this repository and run:

```
python setup.py install
```

7.2 A motivating example

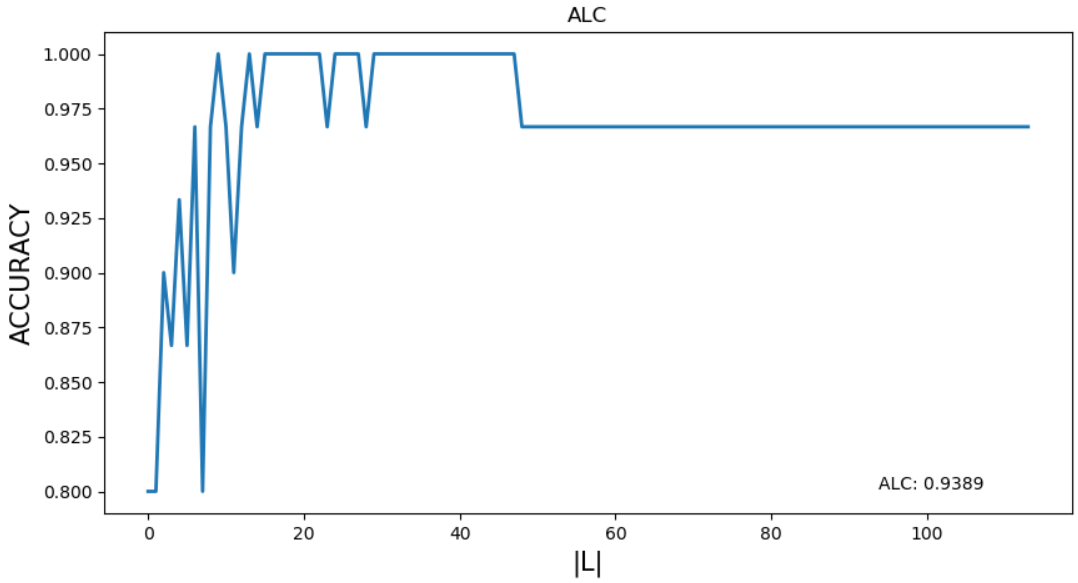
Active learning can often discover a subset of the full data set that generalizes well to the test set. For example, we consider the Iris data set:

```
>>> import numpy as np
>>> from yall import ActiveLearningModel
>>> from yall.querystrategies import Margin
>>> from yall.utils import plot_learning_curve
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.linear_model import LogisticRegression as LR
>>> from sklearn.base import clone
```

```
>>> np.random.seed(0)
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2)
>>> lr = LR(solver="liblinear", multi_class="auto")
>>> lr = lr.fit(train_X, train_y)
>>> print(lr.score(test_X, test_y))
0.967
```

Using the full data set, logistic regression acheives an accuracy of 0.967 on the test data.

```
>>> alm = ActiveLearningModel(clone(lr), Margin(),
...                           eval_metric="accuracy",
...                           U_proportion=0.95, random_state=0)
>>> accuracies, choices = alm.run(train_X, test_X, train_y, test_y)
>>> plot_learning_curve(accuracies, 0, len(accuracies),
...                     eval_metric="accuracy")
```



From the learning curve we see that only the first 25 or so data points are required to acheive perfect 1.0 accuracy on the test data.

```
>>> lr_small = clone(lr)
>>> lr_small = lr_small.fit(alm.L.X[:25, ], alm.L.y[:25])
>>> print(lr_small.score(test_X, test_y))
1.0
```

7.3 Supported query strategies

- Random Sampling (passive learning)
- Uncertainty Sampling
  - Entropy Sampling

  - Least Confidence
  - Least Confidence with Bias
  - Least Confidence with Dynamic Bias
  - Margin Sampling
  - Simple Margin Sampling
- Representative Sampling
  - Density Sampling
  - Distance to Center
  - MinMax Sampling
- Combined Sampling
  - Beta-weighted Combined Sampling
  - Lambda-weighted Combined Sampling

7.4 Running Tests

First install pytest-cov

Then, from the project home directory run

```
py.test --cov=yall tests
```

CHAPTER 8

Authors

- Jake Vasilakes - jvasilakes@gmail.com

CHAPTER 9

License

This project is licensed under the MIT License. See LICENSE for details.

CHAPTER 10

Acknowledgements

This project grew out of a study of active learning methods for biomedical text classification. The paper associated with this study can be found at https://doi.org/10.1093/jamiaopen/ooy021

Python Module Index

y

Index

P

R

S

T

U

W

X

Y