Multi-label classification using Scikit-multilearn

Nicolás García-Pedrajas

Computational Intelligence and Bioinformatics Research Group

March 22, 2020

Table of contents

Introduction

Basic concepts of scikit-multilearn

Multi-label methods

Metrics

Useful functions from scikit-learn

Scikit-multilearn

- Multi-Label Classification in Python
- Scikit-multilearn is a BSD-licensed library for multi-label classification that is built on top of the well-known scikit-learn ecosystem.
- Webpage
- Install with pip:
  - ✓ `pip install scikit-multilearn`

Datasets available

- There are some available datasets than can be loaded

Datasets

```
X_train, y_train, feature_names, label_names = load_dataset('emotions', 'train')
X_test, y_test, _, _= load_dataset('emotions', 'test')
```

- The `feature_names` variable contains list of pairs (feature name, type) that were provided in the original data set

Datasets

Feature names

Feature names

```
>>> feature_names[:10]
[('Mean_Acc1298_Mean_Mem40_Centroid', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_Rolloff', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_Flux', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_MFCC_0', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_MFCC_1', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_MFCC_2', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_MFCC_3', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_MFCC_4', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_MFCC_5', 'NUMERIC'),
  ('Mean_Acc1298_Mean_Mem40_MFCC_6', 'NUMERIC')]
```

Datasets

Label names

Label names

```
>> label_names
('amazed-suprised', ['0', '1']), ('happy-pleased', ['0', '1']), ('relaxing-calm', ['0', '1']), ('quiet-still', ['0', '1']), ('sad-lonely', ['0', '1']), ('angry-aggresive', ['0', '1'])]
```

Data representation

- scikit-multilearn expects on input:
  - ✓ X to be a matrix of shape (n_samples, n_features)
  - ✓ y to be a matrix of shape (n_samples, n_labels)

Data
Available datasets

Datasets

```
>>> from skmultilearn.dataset import available_data_sets
>>> set([x[0] for x in available_data_sets().keys()])
{'mediamill', 'Corel5k', 'rcv1subset1', 'rcv1subset5', 'emotions',
 ↪ 'genbase', 'yeast', 'delicious', 'medical', 'rcv1subset4', 'enron',
 ↪ 'bibtex', 'tmc2007_500', 'rcv1subset3', 'rcv1subset2', 'scene',
 ↪ 'birds'}
```

Datasets

- To download a data set use the `:meth:load_dataset` function.

Datasets

```
>>> from skmultilearn.dataset import load_dataset
>>> X, y, feature_names, label_names = load_dataset('scene', 'train')
scene - exists, not redownloading
>>> X, y, feature_names[:3], label_names[:3]
(<1211x294 sparse matrix of type '<class 'numpy.float64'>'
with 351805 stored elements in LInked List format>,
<1211x6 sparse matrix of type '<class 'numpy.int64'>'
with 1286 stored elements in LInked List format>,
[('Att1', 'NUMERIC'), ('Att2', 'NUMERIC'), ('Att3', 'NUMERIC')],
[('Beach', ['0', '1']), ('Sunset', ['0', '1']), ('FallFoliage', ['0',
↪  '1'])])
```

ARFF files

- The most common way for storing multi-label data is the ARFF file format created by the WEKA library. You can find many benchmark data sets in ARFF format on the MULAN data repository.

- Loading both dense and sparse ARFF files is simple in scikit-multilearn, just use :func:load_from_arff, like this:

```
>> from skmultilearn.dataset import load_from_arff
```

- Loading multi-label ARFF files requires additional information as the number or placement of labels, is not indicated in the format directly.

```
>> path_to_arff_file = '_static/example.arff'
>> label_count = 7
>> label_location="end"
```

scikit-multilearn methods

- The package offers 11 methods
- There are problem transformation and algorithm adaptation methods

BRkNNaClassifier, BRkNNbClassifier

- Parameter estimation needed: Yes, 1 parameter
- Complexity: $O(n_{labels} \times n_{samples} \times n_{features} \times k)$
- BRkNN classifiers train a k Nearest Neighbor per label and use infer label assignment in one of the two variants.
- Strong sides
  - ✓ takes some label relations into account while estimating single-label classifers
  - ✓ works when distance between samples is a good predictor for label assignment. Often used in biosciences.
- Weak sides
  - ✓ trains a classifier per label
  - ✓ less suitable for large label space

MLTSVM

- Parameter estimation needed: Yes, 2 parameters
- Complexity: $O((n_{samples} \times n_{features} + n_{labels}) \times k)$
- Twin multi-Label Support Vector Machines
- Strong sides
  - ✓ estimates one multi-label SVM subclassifier without any one-vs-all or one-vs-rest comparisons, $O(1)$ classifiers instead of $O(l^2)$
  - ✓ works when distance between samples is a good predictor for label assignment
- Weak sides
  - ✓ requires parameter estimation

MLkNN

- Parameter estimation needed: Yes, 2 parameters
- Complexity: $O((n_{samples} \times n_{features} + n_{labels}) \times k)$
- MLkNN builds uses k-NearestNeighbors find nearest examples to a test class and uses Bayesian inference to select assigned labels.
- Strong sides
  - ✓ estimates one multi-class subclassifier
  - ✓ works when distance between samples is a good predictor for label assignment
  - ✓ often used in biosciences.
- Weak sides
  - ✓ requires parameter estimation

MLARAM

- Parameter estimation needed: Yes, 2 parameters
- Complexity: $O(n_{samples})$
- An ART classifier which uses clustering of learned prototypes into large clusters improve performance.
- Strong sides
  - ✓ linear in number of samples, scales well
- Weak sides
  - ✓ requires parameter estimation
  - ✓ ART techniques have had generalization limits in the past

BinaryRelevance

- Parameter estimation needed: Only for base classifier
- Complexity: $O(n_{labels} \times base\_single\_class\_classifier\_complexity)$
- Transforms a multi-label classification problem with L labels into L single-label separate binary classification problems.
- Strong sides
  - ✓ estimates single-label classifiers
  - ✓ can generalize beyond available label combinations
- Weak sides
  - ✓ not suitable for large number of labels
  - ✓ ignores label relations

ClassifierChain

- Parameter estimation needed: Yes, 1 + parameters for base classifier
- Complexity: $O(n_{labels} \times base\_single\_class\_classifier\_complexity)$
- Transforms multi-label problem to a multi-class problem where each label combination is a separate class.
- Strong sides
  - ✓ estimates single-label classifiers
  - ✓ can generalize beyond available label combinations
  - ✓ takes label relations into account
- Weak sides
  - ✓ not suitable for large number of labels
  - ✓ quality strongly depends on the label ordering in chain

LabelPowerset

- Parameter estimation needed: Only for base classifier
- Complexity: $O(base\_multi\_class\_classifier\_complexity(n_{classes} = n_{label\_combinations}))$
- Transforms multi-label problem to a multi-class problem where each label combination is a separate class and uses a multi-class classifier to solve the problem.
- Strong sides
  - ✓ estimates label dependencies, with only one classifier
  - ✓ often best solution for subset accuracy if training data contains all relevant label combinations
- Weak sides
  - ✓ requires all label combinations predictable by the classifier to be present in the training data
  - ✓ very prone to underfitting with large label spaces

RakelD

- Parameter estimation needed: Yes, 1 + base classifier's parameters
- Complexity:
  $O(n_{partitions} \times base\_multi\_class\_classifier\_complexity(n_{classes} = n_{label\_combinations\_per\_partition}))$
- Randomly partitions label space and trains a Label Powerset classifier per partition with a base multi-class classifier.
- Strong sides
  - ✓ may use less classifiers than Binary Relevance and still generalize label relations while not underfitting like LabelPowerset
- Weak sides
  - ✓ using random approach is not very probable to draw an optimal label space division

RakelO

- Parameter estimation needed: Yes, 2 + base classifier's parameters
- Complexity:
  $O(n_{partitions} \times base\_multi\_class\_classifier\_complexity(n_{classes} = n_{label\_combinations\_per\_cluster}))$
- Randomly draw label subspaces (possibly overlapping) and trains a Label Powerset classifier per partition with a base multi-class classifier, labels are assigned based on voting
- Strong sides
  - ✓ may provide better results with overlapping models
- Weak sides
  - ✓ takes large number of classifiers to generate improvement, not scalable
  - ✓ random subspaces may not be optimal

LabelSpacePartitioningClassifier

- Parameter estimation needed: Only base classifier
- Complexity: $O(n_{partitions} \times classifier\_complexity(n_{classes} = n_{label\_combinations\_per\_partition}))$
- Uses clustering methods to divide the label space into subspaces and trains a base classifier per partition with a base multi-class classifier.
- Strong sides
  - ✓ accommodates to different types of problems
  - ✓ infers when to divide into subproblems or not and decide when to use less classifiers than Binary Relevance
  - ✓ scalable to data sets with large numbers of labels
  - ✓ generalizes label relations well while not underfitting like LabelPowerset
  - ✓ does not require parameter estimation
- Weak sides
  - ✓ requires label relationships present in training data to be representative of the problem
  - ✓ partitioning may prevent certain label combinations from being correctly classified, depends on base classifier

MajorityVotingClassifier

- Parameter estimation needed: Only base classifier
- Complexity: $O(n_{clusters} \times classifier\_complexity(n_{classes} = n_{label\_combinations\_per\_cluster}))$
- Uses clustering methods to divide the label space into subspaces (possibly overlapping) and trains a base classifier per partition with a base multi-class classifier, labels are assigned based on voting.
- Strong sides
  - ✓ accommodates to different types of problems
  - ✓ infers when to divide into subproblems or not and decide when to use less classifiers than Binary Relevance
  - ✓ scalable to data sets with large numbers of labels
  - ✓ generalizes label relations well while not underfitting like LabelPowerset
  - ✓ does not require parameter estimation
- Weak sides
  - ✓ requires label relationships present in training data to be representative of the problem

LabelSpacePartitioningClassifier

- Parameter estimation needed: Only for embeder
- Complexity: depends on the selection of embeder, regressor and classifier
- Embeds the label space, trains a regressor (or many) for unseen samples to predict their embeddings, and a classifier to correct the regression error
- Strong sides
  - ✓ improves discriminability and joint label probability distributions
  - ✓ good results with low-complexity linear embeddings and weak regressors/classifiers
- Weak sides
  - ✓ requires some parameter estimation while rule-of-thumb ideas exist in papers

- Package `sklearn.metrics` implements several measures
- Some of them are useful for multi-label learning
  - ✓ Hamming loss
  - ✓ Accuracy score
  - ✓ Coverage
  - ✓ A few others...
- The remaining ones must be programmed

Binary relevance method

- The method is available for all classifiers with `MultiOutputClassifier()` and `OneVsRestClassifier()`
- This strategy consists of fitting one classifier per target.
- This allows multiple target variable classifications.
- The purpose of this class is to extend estimators to be able to estimate a series of target functions $(f_1, f_2, f_3, \ldots f_n)$ that are trained on a single X predictor matrix to predict a series of responses $(y_1, y_2, y_3, \ldots, y_n)$
- This method implements both multi-label and multi-output classification.

Binary relevance method

MultiOutputClassifier()

BR method

```python
#!  /usr/bin/python

from sklearn.datasets import make_classification
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import shuffle
import numpy as np

X, y1 = make_classification(n_samples=10, n_features=100, n_informative=30, n_classes=2, random_state=1)
y2 = shuffle(y1, random_state=1)
y3 = shuffle(y1, random_state=2)

Y = np.vstack((y1, y2, y3)).T

n_samples, n_features = X.shape # 10,100
n_outputs = Y.shape[1] # 3
n_classes = 2

forest = RandomForestClassifier(random_state=1)

multi_target_forest = MultiOutputClassifier(forest, n_jobs=-1)
yhat = multi_target_forest.fit(X, Y).predict(X)

print(yhat)
```

Binary relevance method

OneVsRestClassifier()

BR method

```
#!  /usr/bin/python

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.utils import shuffle
import numpy as np

X, y1 = make_classification(n_samples=10, n_features=100, n_informative=30, n_classes=2,
↪  random_state=1)
y2 = shuffle(y1, random_state=1)
y3 = shuffle(y1, random_state=2)

Y = np.vstack((y1, y2, y3)).T

n_samples, n_features = X.shape # 10,100
n_outputs = Y.shape[1] # 3
n_classes = 2

multi_label = OneVsRestClassifier(LogisticRegression(solver='sag'), n_jobs=-1)
yhat = multi_label.fit(X, Y).predict(X)

print(yhat)
```

Miscellaneous tools

Generation of a random problems

- Generate a random multi-label classification problem
- `sklearn.datasets.make_multilabel_classification()`
- sklearn.datasets.make_multilabel_classification(n_samples=100, n_features=20, n_classes=5, n_labels=2, length=50, allow_unlabeled=True, sparse=False, return_indicator='dense', return_distributions=False, random_state=None)

Miscellaneous tools

Confusion matrix

- Compute a confusion matrix for each class or sample
- `sklearn.metrics.multilabel_confusion_matrix()`
- sklearn.metrics.multilabel_confusion_matrix(y_true, y_pred, sample_weight=None, labels=None, samplewise=False)