project-template Documentation

Release 0.1.0b

Christian Schulze

Apr 25, 2018

Contents

This project implements a number of multi-label classification (MLC) problem transformation methods, multi-label ensembles as well as adapted algorithms with scikit-learn compatible estimators.

Note that skml is in an early stage, and if you observe any unexpected behavior or have questions, please create an issue.

Please refer to the *User Guide* for an overview or background information. The *Multi-label Classification Examples* section holds simple examples to common problems.

CHAPTER 1

API Documentation

1.1 Problem Transformation Methods

Problem transformation methods reduce the problem of multi-label classification into a number of easier problems, for example binary or multi-class classification.

1.2 Ensemble Methods

Ensemble methods provide multi-label classification compatible ensemble methods, where a number of estimators (or classifiers) are used to gather a number of predictions, and then obtain votes by majority vote or averaging. This is expected to achieve better results, as the diversity of classifiers (optimally) works as an error correction to the other classifiers.

1.3 Multi-label Data Sets

This sub-module provides loading of data sets and down sampling of the label space.

`skml.datasets.`**`load_dataset`**(*name*)

Loads a multi-label classification dataset.

Parameters

name [string] Name of the dataset. Currently only 'yeast' is available.

`skml.datasets.`**`sample_down_label_space`**(*y*, *k*, *method='most-frequent'*)

Samples down label space, such that the returned label space retains order of the original labels, but removes labels which do not meet certain criteria (see *method*).

Parameters

y [(sparse) array-like, shape = [n_samples,], [n_samples, n_classes]] Multi-label targets

k [number] Number of returned labels, has to be smaller than the number of distinct labels in *y*

method [string, default = 'most-frequent'] Method to sample the label space down. Currently supported is only by top k most frequent labels.

CHAPTER 2

Multi-label Classification Examples

Introductory examples.

2.1 Ensemble Classifier Chain Example

An example of `skml.ensemble.EnsembleClassifierChain`

```
from sklearn.metrics import hamming_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import numpy as np


from skml.ensemble import EnsembleClassifierChain
from skml.datasets import load_dataset

X, y = load_dataset('yeast')
X_train, X_test, y_train, y_test = train_test_split(X, y)

ensemble = EnsembleClassifierChain(RandomForestClassifier())
ensemble.fit(X, y)
y_pred = ensemble.predict(X)

print("hamming loss: ")
print(hamming_loss(y, y_pred))

print("accuracy:")
print(accuracy_score(y, y_pred))
```

(continues on next page)

(continued from previous page)

```
print("f1 score:")
print("micro")
print(f1_score(y, y_pred, average='micro'))
print("macro")
print(f1_score(y, y_pred, average='macro'))

print("precision:")
print("micro")
print(precision_score(y, y_pred, average='micro'))
print("macro")
print(precision_score(y, y_pred, average='macro'))

print("recall:")
print("micro")
print(recall_score(y, y_pred, average='micro'))
print("macro")
print(recall_score(y, y_pred, average='macro'))
```

Total running time of the script: (0 minutes 0.000 seconds)

2.2 Ensemble Label Powerset Example

An example of `skml.problem_transformation.LabelPowerset`

```
from sklearn.metrics import hamming_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

from skml.problem_transformation import LabelPowerset
from skml.datasets import load_dataset

X, y = load_dataset('yeast')
X_train, X_test, y_train, y_test = train_test_split(X, y)

clf = LabelPowerset(LogisticRegression())
clf.fit(X_test, y_test)
y_pred = clf.predict(X_test)

print("hamming loss: ")
print(hamming_loss(y_test, y_pred))

print("accuracy:")
print(accuracy_score(y_test, y_pred))

print("f1 score:")
print("micro")
print(f1_score(y_test, y_pred, average='micro'))
print("macro")
print(f1_score(y_test, y_pred, average='macro'))
```

(continues on next page)

(continued from previous page)

```
print("precision:")
print("micro")
print(precision_score(y_test, y_pred, average='micro'))
print("macro")
print(precision_score(y_test, y_pred, average='macro'))

print("recall:")
print("micro")
print(recall_score(y_test, y_pred, average='micro'))
print("macro")
print(recall_score(y_test, y_pred, average='macro'))
```

Total running time of the script: (0 minutes 0.000 seconds)

2.3 Classifier Chain Example

An example of `skml.problem_transformation.ClassifierChain`

```
from sklearn.metrics import hamming_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np


from skml.problem_transformation import ClassifierChain
from skml.datasets import load_dataset

X, y = load_dataset('yeast')
X_train, X_test, y_train, y_test = train_test_split(X, y)

cc = ClassifierChain(LogisticRegression())
cc.fit(X_train, y_train)
y_pred = cc.predict(X_test)


print("hamming loss: ")
print(hamming_loss(y_test, y_pred))

print("accuracy:")
print(accuracy_score(y_test, y_pred))

print("f1 score:")
print("micro")
print(f1_score(y_test, y_pred, average='micro'))
print("macro")
print(f1_score(y_test, y_pred, average='macro'))

print("precision:")
print("micro")
```

(continues on next page)

(continued from previous page)

```
print(precision_score(y_test, y_pred, average='micro'))
print("macro")
print(precision_score(y_test, y_pred, average='macro'))

print("recall:")
print("micro")
print(recall_score(y_test, y_pred, average='micro'))
print("macro")
print(recall_score(y_test, y_pred, average='macro'))
```

Total running time of the script: (0 minutes 0.000 seconds)

2.4 Ensemble Binary Relevance Example

An example of `skml.problem_transformation.BinaryRelevance`

```
from __future__ import print_function

from sklearn.metrics import hamming_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

from skml.problem_transformation import BinaryRelevance
from skml.datasets import load_dataset

X, y = load_dataset('yeast')
X_train, X_test, y_train, y_test = train_test_split(X, y)

clf = BinaryRelevance(LogisticRegression())
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)


print("hamming loss: ")
print(hamming_loss(y_test, y_pred))

print("accuracy:")
print(accuracy_score(y_test, y_pred))

print("f1 score:")
print("micro")
print(f1_score(y_test, y_pred, average='micro'))
print("macro")
print(f1_score(y_test, y_pred, average='macro'))

print("precision:")
print("micro")
print(precision_score(y_test, y_pred, average='micro'))
print("macro")
```

(continues on next page)

(continued from previous page)

```
print(precision_score(y_test, y_pred, average='macro'))

print("recall:")
print("micro")
print(recall_score(y_test, y_pred, average='micro'))
print("macro")
print(recall_score(y_test, y_pred, average='macro'))
```

Total running time of the script: (0 minutes 0.000 seconds)

2.5 Probabilistic Classifier Chain Example

An example of `skml.problem_transformation.ProbabilisticClassifierChain`

```
from sklearn.metrics import hamming_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

from skml.problem_transformation import ProbabilisticClassifierChain
from skml.datasets import load_dataset


X, y = load_dataset('yeast')
# sample down the label space to make the example faster.
# you shouldn't do this on your own data though!
y = y[:, :6]

X_train, X_test, y_train, y_test = train_test_split(X, y)

pcc = ProbabilisticClassifierChain(LogisticRegression())
pcc.fit(X_train, y_train)
y_pred = pcc.predict(X_test)


print("hamming loss: ")
print(hamming_loss(y_test, y_pred))

print("accuracy:")
print(accuracy_score(y_test, y_pred))

print("f1 score:")
print("micro")
print(f1_score(y_test, y_pred, average='micro'))
print("macro")
print(f1_score(y_test, y_pred, average='macro'))

print("precision:")
print("micro")
print(precision_score(y_test, y_pred, average='micro'))
```

(continues on next page)

(continued from previous page)

```
print("macro")
print(precision_score(y_test, y_pred, average='macro'))

print("recall:")
print("micro")
print(recall_score(y_test, y_pred, average='micro'))
print("macro")
print(recall_score(y_test, y_pred, average='macro'))
```

Total running time of the script: (0 minutes 0.000 seconds)

CHAPTER 3

User Guide

3.1 Multi-label Data Sets

The `skml.datasets` component provides popular multi-label classification datasets, as well as methods to reduce the label space size by different means.

3.2 Problem Transformation

The `skml.problem_transformation` module implements so called *meta-estimators* to solve multi-label classification problems by transforming them into a number of easier problems, i.e. binary classification problems.

3.2.1 Binary Relevance

Binary Relevance (`skml.problem_transformation.BinaryRelevance`) transforms a problem of multi-classification with $|\mathcal{L}|$ labels into a problem of $|\mathcal{L}|$ binary classification problems, hence trains $|\mathcal{L}|$ classifiers that decide label-wise, if the example that is currently being observed should have the label or not. (Dubbed PT-4 in the cited paper.)

Note, that binary relevance is not capable of modeling label interdependence.

References:

3.2.2 Label Powerset

Label Powerset (`skml.problem_transformation.LabelPowerset`) transforms a multi-class classification problem into one multi-class problem, where all possible label combinations are used as classes. So each possible combination of labels is turned into one class. If the underlying multi-label

problem operates on the label space $\mathcal{L}$ with $|\mathcal{L}|$ labels, label powerset will train $|2^{\mathcal{L}}|$ classifiers, where each one decides if the label combination should be assigned to an example.

Note, that while label powerset can model label interdependence, the computational feasibility can be reduced for a large number of labels, as the number of trained classifiers grows exponentially. (Dubbed PT-5 in the cited paper.)

References:

3.2.3 Classifier Chains

Classifier chains (`skml.problem_transformation.ClassifierChain`) improve the binary relevance (`skml.problem_transformation.BinaryRelevance`) method to use label interdependence as well. For each label a classifier is trained. Besides the first classifier in the chain, each subsequent classifier is trained on a modified input vector, where the previous predicted labels are incorporated. Thus, a chain of classifiers is predicted, and each classifier in the chain gets also the previous class predictions as an input.

Note, that the performance of a single chain depends heavily on the order of the classifiers in the chain.

References:

3.2.4 Probabilistic Classifier Chains

Probabilistic Classifier Chains (`skml.problem_transformation.ProbabilisticClassifierChain`) –also known as PCC– are an extension to the classic Classifier Chains (`skml.problem_transformation.ClassifierChain`) and can be seen as a discrete greedy approximation of probabilistic classifier chains with probabilities valued zero/one [3].

For each label a classifier is trained as in CC, however probabilistic classifiers are used. In fact [3], when used with non-probabilistic classifiers, CC is recovered from the posterior probability distribution $\mathbf{P}_{\mathbf{y}}(\mathbf{x})$.

Note, that PCC performs best, when a loss function that models label interdendence (such as Subset 0/1 loss) is used, and the labels in the data set are in fact interdependent. For more information on this, see [3].

The training is equivalent to CC, the inference (prediction) however is more complex. For a detailed description of the inference, see `skml.problem_transformation.ProbabilisticClassifierChains` directly, have a look at the source code, or refer to the paper [3].

References:

3.3 Ensemble Methods

The `skml.ensemble` module implements ensembles to be used for multi-label classification.

3.3.1 Ensemble Classifier Chains

Ensemble of classifier chains (ECC) trains an ensemble of bagged classifier chains. Each chain is trained on a randomly sampled subset of the training data (with replacement, also known as bagging).

References:

CHAPTER 4

Indices and tables

- genindex

Index