Relative clustering validation

Release 0.0.1

Feb 09, 2022

User guide / Tutorial

sphinx-quickstart on Mon May 11 18:17:16 2020.

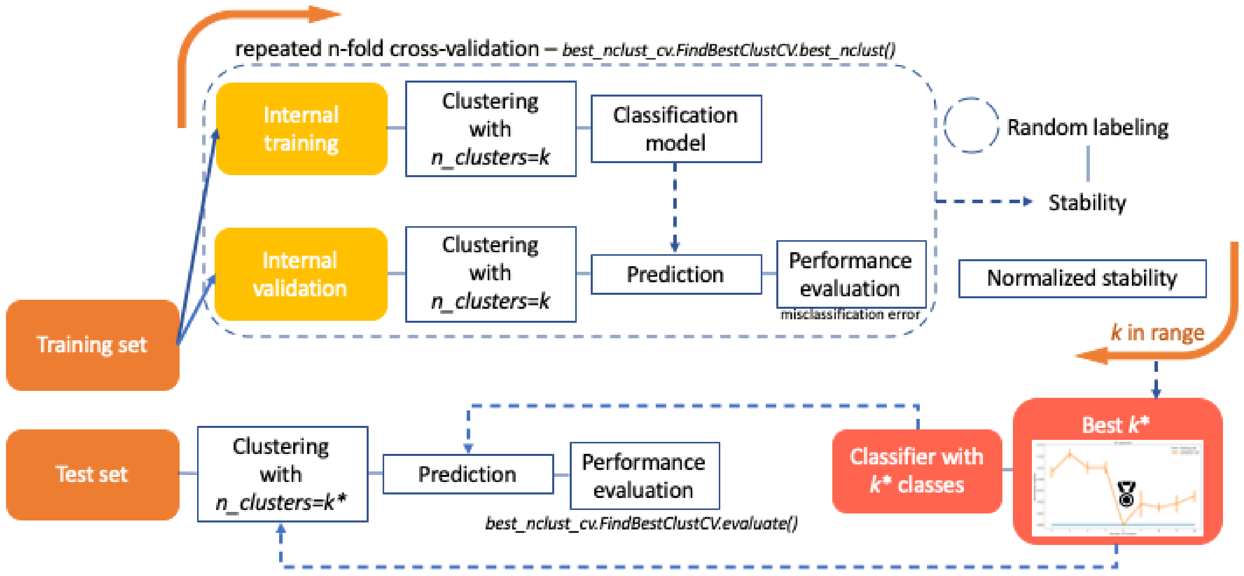
`reval` allows to determine the best clustering solution without a priori knowledge. It leverages a stability-based relative clustering validation method (Lange et al., 2004) that transforms a clustering algorithm into a supervised classification problem and selects the number of clusters that leads to the minimum expected misclassification error, i.e., stability.

This library allows to:

1. Select any classification algorithm from `sklearn` library;
2. **Select a clustering algorithm with `n_clusters` parameter or HDBSCAN density-based algorithm,** i.e., choose among `sklearn.cluster.KMeans`, `sklearn.cluster.AgglomerativeClustering`, `sklearn.cluster.SpectralClustering`, `hdbscan.HDBSCAN`;
3. Perform (repeated) *k*-fold cross-validation to determine the best number of clusters;
4. Test the final model on an held-out dataset.

Theoretical background can be found in (Lange et al., 2004), whereas code can be found on github.

The analysis steps performed by `reval` package are displayed below.


repeated n-fold cross-validation – best_nclust_cv.FindBestClustCV.best_nclust()
Internal training
Clustering with n_clusters=k
Classification model
Random labeling
Stability
Internal validation
Clustering with n_clusters=k
Prediction
Performance evaluation
misclassification error
Normalized stability
Training set
k in range
Test set
Clustering with n_clusters=k*
Prediction
Performance evaluation
best_nclust_cv.FindBestClustCV.evaluate()
Classifier with k* classes
Best k*

Lange, T., Roth, V., Braun, M. L., & Buhmann, J. M. (2004). Stability-based validation of clustering solutions. *Neural computation*, 16(6), 1299-1323.

CHAPTER 1

Installing

From github, navigate to the folder you want `reval` library in, open terminal and run:

```
git clone https://github.com/IIT-LAND/reval_clustering
pip install -r requirements.txt
```

PyPI alternative:

```
pip install reval
```

CHAPTER 2

How to use reval

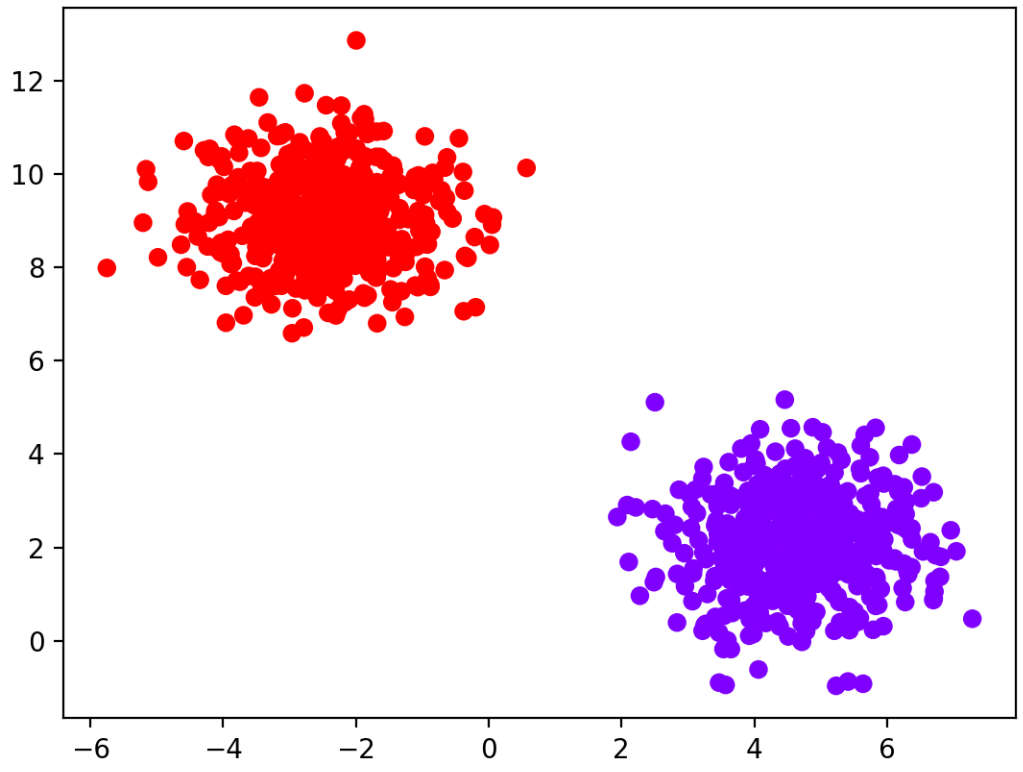
In the following, we are going to simulate N = 1,000 sample dataset with two groups and two features (for visualization purposes), then we will show how to apply the `reval` package and investigate the result types. We will use hierarchical clustering and KNN classification algorithms.

First (after starting `ipython` or a jupyter notebook), let us import a bunch of useful libraries and our class `reval.best_nclust_cv.FindBestClustCV`:

```
from reval.best_nclust_cv import FindBestClustCV
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import AgglomerativeClustering
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Then we simulate the toy dataset and visualize it:

```
data = make_blobs(1000, 2, centers=2, random_state=42)
plt.scatter(data[0][:, 0], data[0][:, 1],
            c=data[1], cmap='rainbow_r')
plt.show()
```



Then, we split the dataset into training and test sets:

```
X_tr, X_ts, y_tr, y_ts = train_test_split(data[0], data[1],
                                          test_size=0.30,
                                          random_state=42,
                                          stratify=data[1])
```

We apply the stability-based relative clustering validation approach with 10x2 repeated cross-validation, 10 iterations of random labeling, and number of clusters ranging from 2 to 10.

```
classifier = KNeighborsClassifier()
clustering = AgglomerativeClustering()
findbestclust = FindBestClustCV(nfold=2,
                                nclust_range=list(range(2, 11)),
                                s=classifier,
                                c=clustering,
                                nrand=100)
metrics, nbest = findbestclust.best_nclust(X_tr, iter_cv=10, strat_vect=y_tr)
out = findbestclust.evaluate(X_tr, X_ts, nbest)
```

To obtain the training stability and the normalized validation stability for the selected number of clusters we need to call:

```
nbest
# 2
metrics['train'][nbest]
```

(continues on next page)

(continued from previous page)

```
# (0.0, (0.0, 0.0)) (stab, (stab, error))
metrics['val'][nbest]
# (0.0, (0.0, 0.0)) (stab, (stab, error))
```

`out` returns train/test accuracies and test set clustering labels.

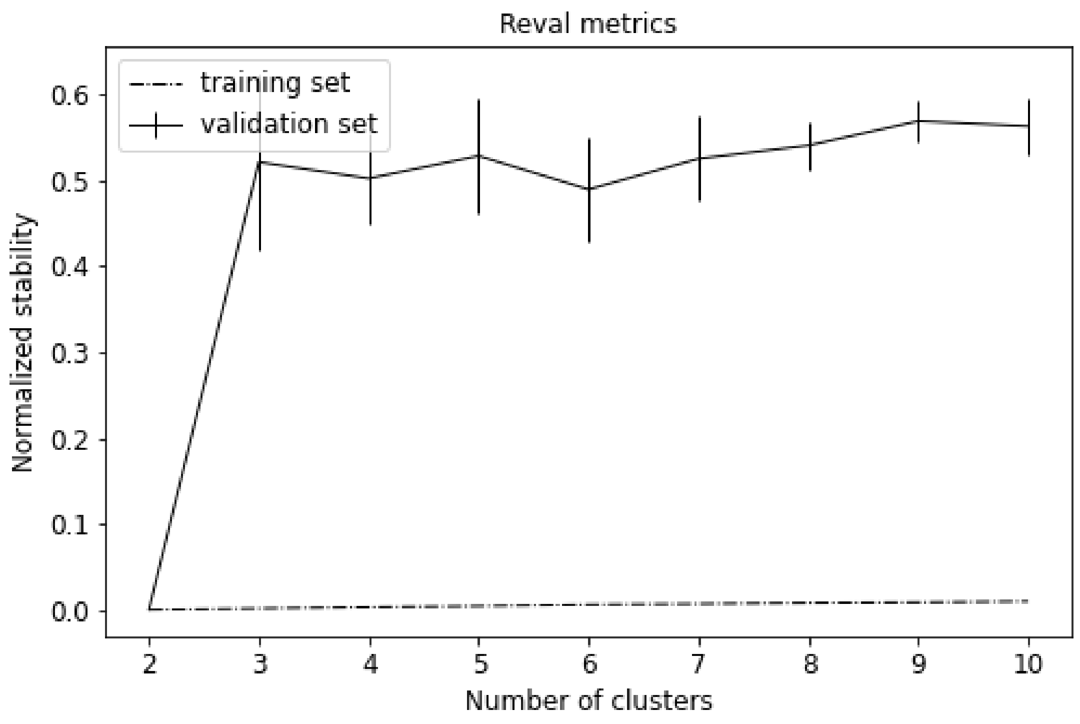
```
out.train_cllab
# array([0, 1, 0, 1, 0, 0, 1...
out.test_cllab
# array([0, 0, 0, 0, 1...
out.train_acc
# 1.0
out.test_acc
# 1.0
```

Attribute `cv_results_` of `FindBestClustCV` returns a dataframe with training and validation misclassification errors.

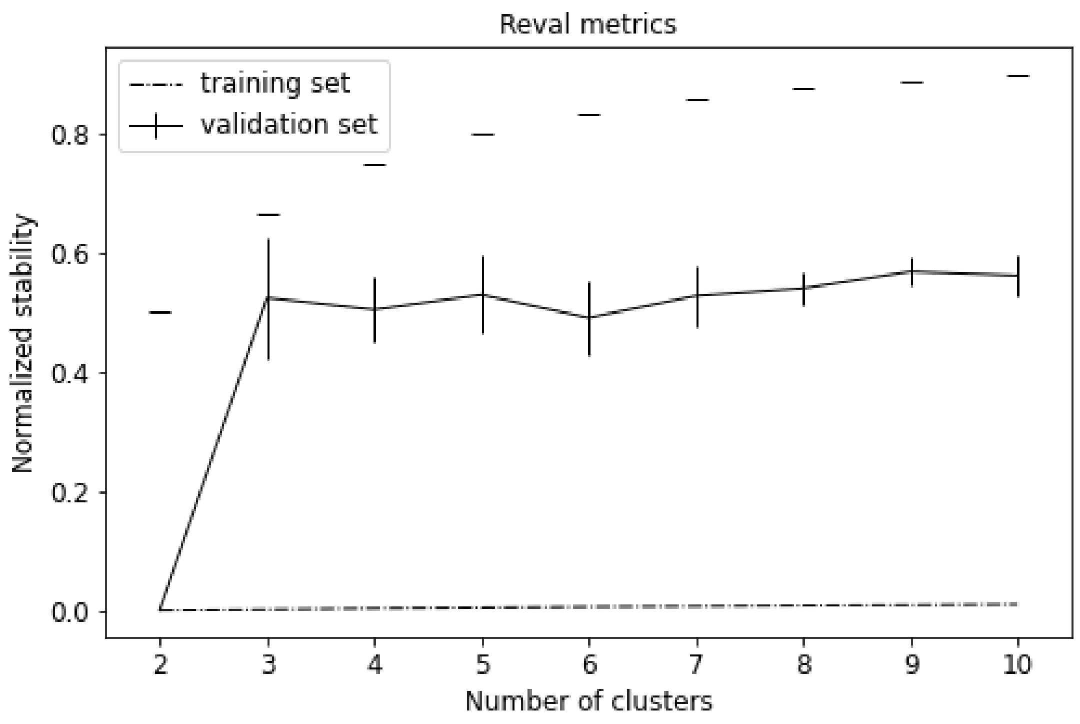
```
findbestclust.cv_results_
```

To visualize performance metrics during cross-validation, i.e., training stability and validation normalized stability with confidence intervals:

```
from reval.visualization import plot_metrics
plot_metrics(metrics, title="Reval metrics")
```



The plot can be customized and also show the normalized stability of a random classifier for each number of clusters to evaluate the model performance.



2.1 Classifier/clustering selection

Let us now suppose that we are not sure which combination of clustering and classifier to use for the blobs dataset. We might want to try both hierarchical clustering and k-means and KNN and logistic regression. We import the libraries we have not imported before including the `SCParamSelection` from the `param_selection.py` module.

```
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from reval.param_selection import SCParamSelection
```

We initialize the parameter selection class with a dictionary that includes the classification and clustering algorithms we want to run and we initialize a 10x2 repeated cross validation with 10 runs of random labeling. We set the number of parallel processes to 7 to speed up computations.

```
sc_params = {'s': [LogisticRegression(max_iter=1000), KNeighborsClassifier()],
             'c': [AgglomerativeClustering(), KMeans()]}
scparsel = SCParamSelection(sc_params, cv=2, nrand=10, n_jobs=7,
                            iter_cv=10, clust_range=list(range(2, 11)),
                            strat=y_tr)
scparsel.fit(X_tr, nclass=2)
```

In this case we knew the true number of clusters a priori, so we passed it to the `fit()` method in order to prioritize the parameter combinations that select the true number of clusters, along with the combinations with global minimum stability. As a result, four different combinations are run and all of them selected two as the best number of clusters with minimum stability.

2.2 Parameter selection

Let us now settle with hierarchical clustering and KNN and suppose we want to try different number of neighbors for KNN, i.e., 5 and 15, and different methods for hierarchical clustering, i.e., Ward and single-linkage. We can then use the `ParamSelection` as follows:

```
from reval.param_selection import ParamSelection
params = {'s': {'n_neighbors': [5, 15]},
          'c': {'linkage': ['ward', 'single']}}
parsel = ParamSelection(params, cv=2, s=KNeighborsClassifier(),␣
↪c=AgglomerativeClustering(),
                        nrand=10,
                        n_jobs=7,
                        iter_cv=10,
                        strat=y_tr, clust_range=list(range(2, 11)))
parsel.fit(X_tr, nclass=2)
```

Also in this case we run four different hyperparameter combinations which all report 2 as the best number of clusters with minimum stability.

CHAPTER 3

Performance on benchmark datasets

We present here three examples to test `reval` performance. If github folder is cloned, code for the following experiments can be found in *reval_clustering/working_examples* folder.

1. N = 1,000 Gaussian blob samples with 10 features divided into 5 clusters (code in `blobs.py`);
2. N = 1,000 Gaussian blob samples with 10 features divided into 5 clusters with noise parameter *cluster_std* set at 3 (code in `blobs.py`);

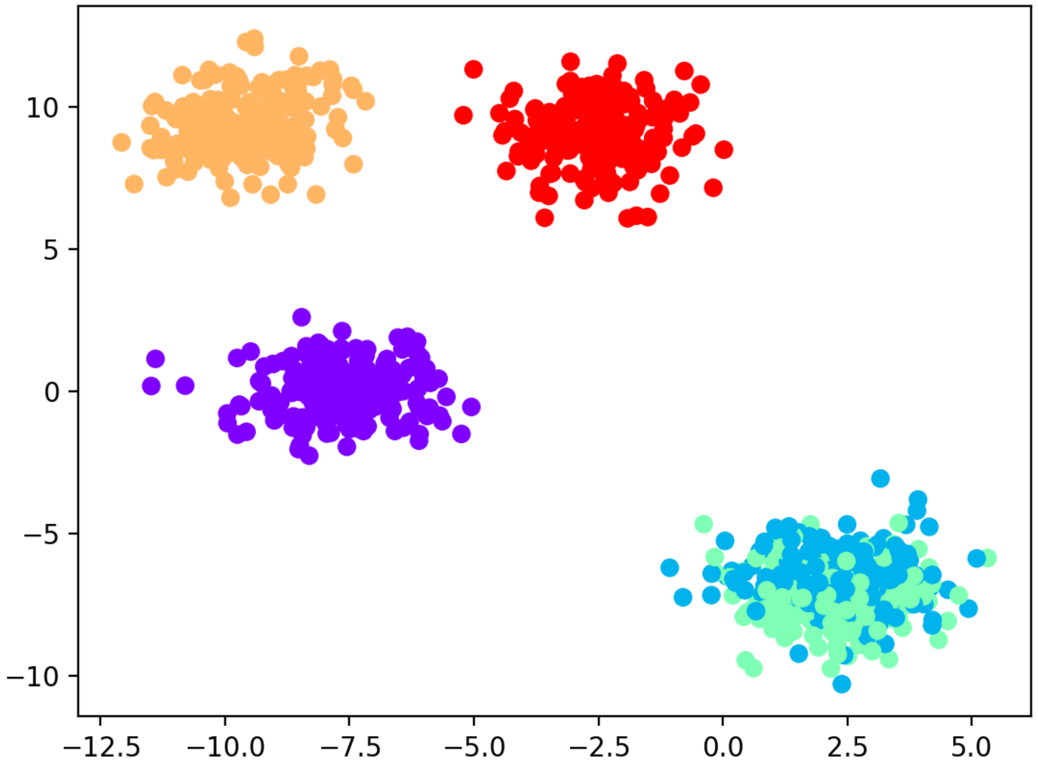
3. N = 14,000 samples from the MNIST handwritten digits dataset loaded from openml public repository (code in `handwrittend_digits.py`).

3.1 Gaussian blobs

```
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from reval.best_nclust_cv import FindBestClustCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import zero_one_loss, adjusted_mutual_info_score
from reval.visualization import plot_metrics
import matplotlib.pyplot as plt
from reval.utils import kuhn_munkres_algorithm
```

Generate sample dataset and visualize blobs (only the first two features).

```
data = make_blobs(1000, 10, centers=5, random_state=42)
plt.scatter(data[0][:, 0],
            data[0][:, 1],
            c=data[1], cmap='rainbow_r')
plt.show()
```



We select hierarchical clustering with k-nearest neighbors classifier for number of cluster selection.

```
classifier = KNeighborsClassifier()
clustering = AgglomerativeClustering()
```

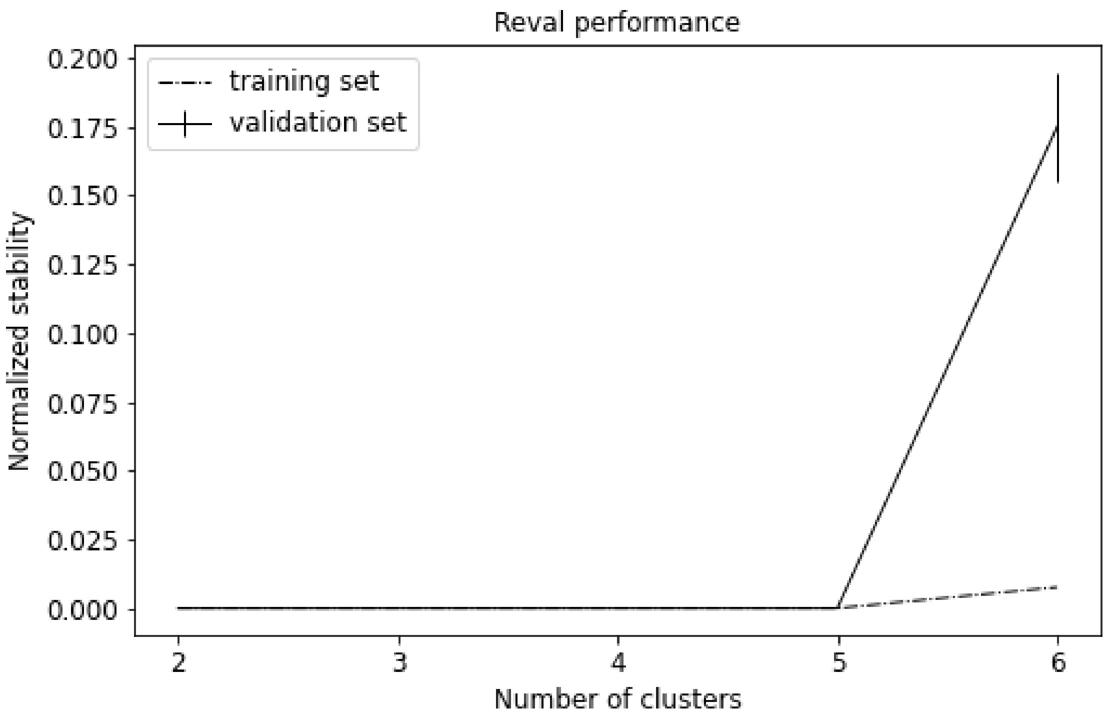
Then we split the dataset into a training and test sets at 30%. We stratify for class labels.

```
X_tr, X_ts, y_tr, y_ts = train_test_split(data[0],
                                          data[1],
                                          test_size=0.30,
                                          random_state=42,
                                          stratify=data[1])
```

Apply `reval` with 10 repetitions of 2-fold cross-validation, 10 random labeling iterations, and number of clusters varying from 2 to 6. We then plot model performance using the function `plot_metrics` from the `reval.visualization` module.

```
findbestclust = FindBestClustCV(nfold=2,
                                nclust_range=list(range(2, 7)),
                                s=classifier,
                                c=clustering,
                                nrand=10)
metrics, nbest = findbestclust.best_nclust(X_tr, iter_cv=10, strat_vect=y_tr)
out = findbestclust.evaluate(X_tr, X_ts, nbest)
plot_metrics(metrics, title="Reval performance")
```

We obtain that the best number of clusters returned by the model is 5 (see performance plot).



Normalized stability in validation is 0.0 (i.e., perfect prediction) and test set accuracy is equal to 1.0.

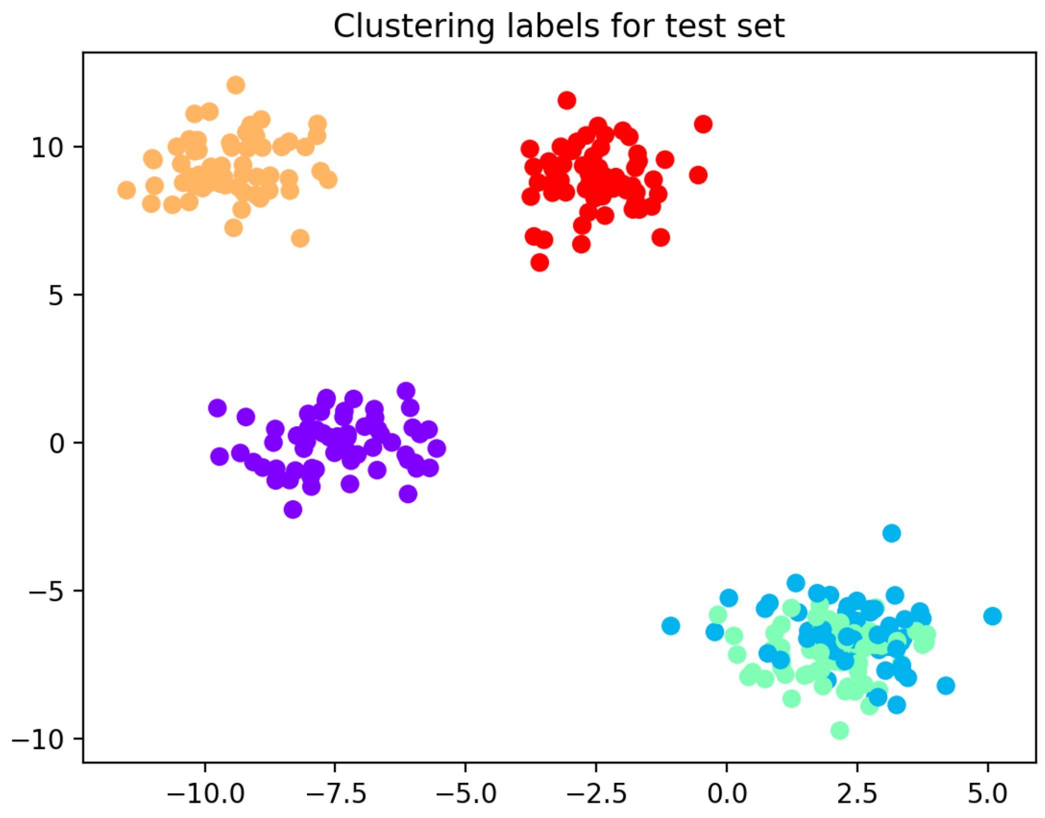
We are now interested in comparing the clustering labels from the test set with the true labels. Hence, we first apply Kuhn-Munkres algorithm to permute the labels returned by the model. This because they may not be ordered as the true labels and lead to an unreliable classification error.

```
perm_lab = kuhn_munkres_algorithm(y_ts, out.test_cllab)
```

Then we compute the classification accuracy and the adjusted mutual information score (AMI) to compare two partitions (this score is independent of label permutations and is equal to 1.0 when two partitions are identical:

```
print(f"Test set external ACC: "
      f"{1 - zero_one_loss(y_ts, perm_lab)}")
print(f'AMI = {adjusted_mutual_info_score(y_ts, out.test_cllab)}')
```

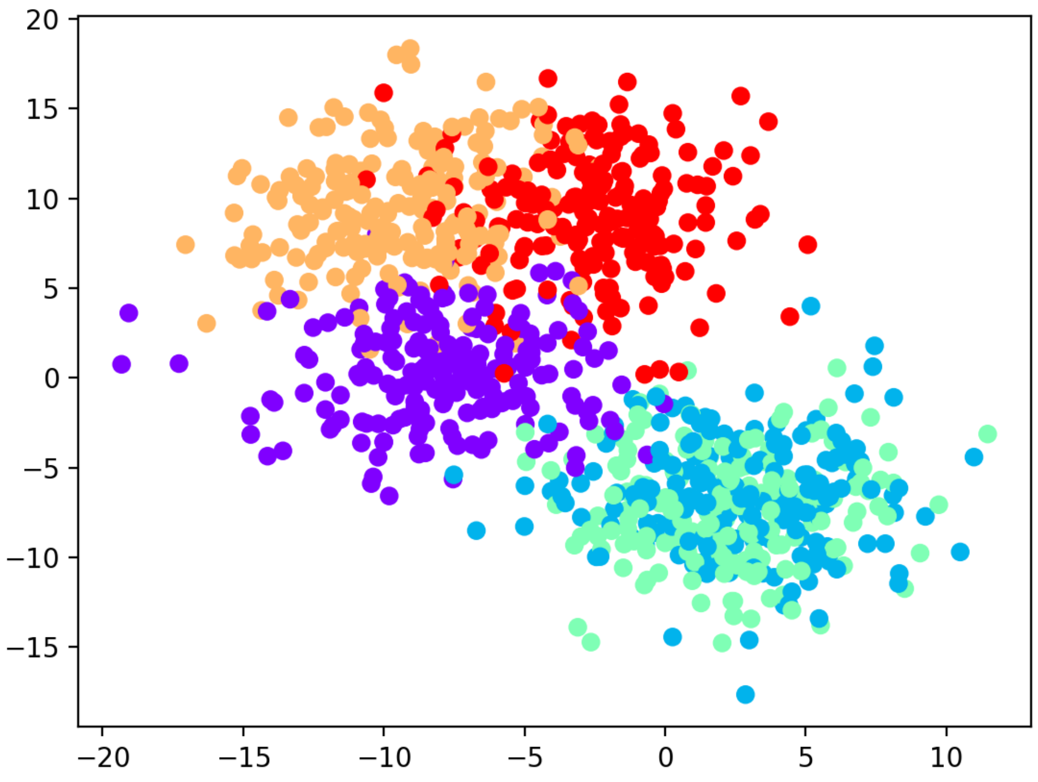
We obtain 100% accuracy and AMI equal to 1.0, see the following scatterplot for visualization of predicted labels.



3.2 Gaussian blobs with noise

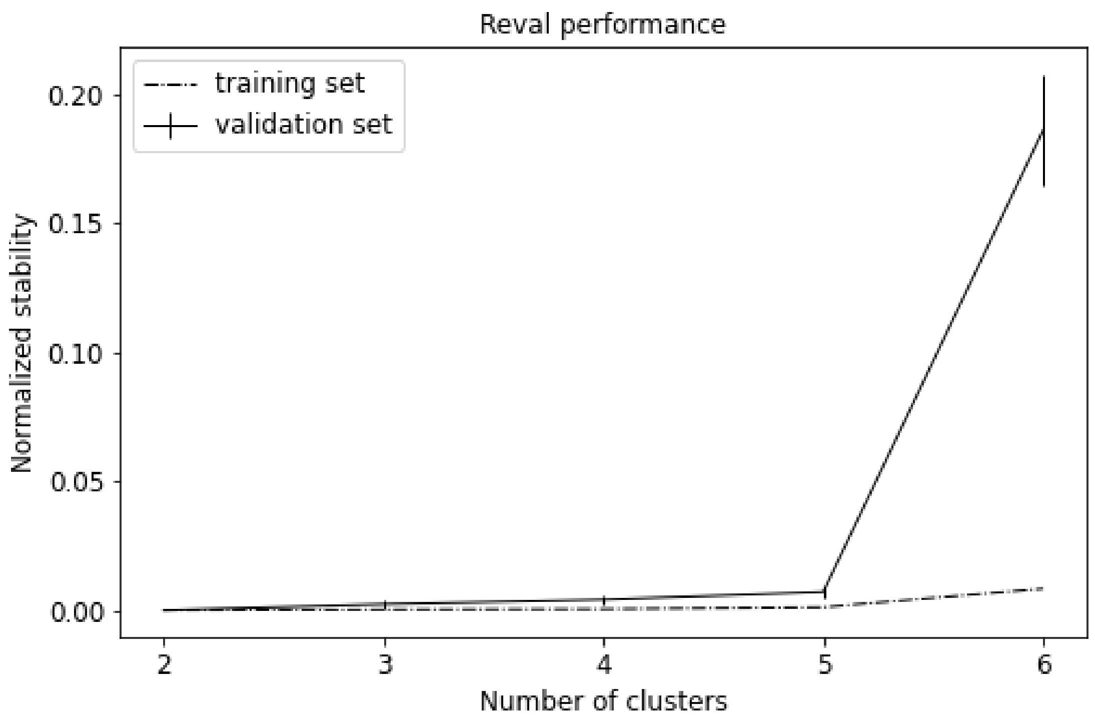
Let us now consider a synthetic dataset of 1,000 samples and 10 features with added noise. We set the number of clusters to 5, as previously. In the following, we will observe how the number of clusters returned by `reval` method is highly influenced by noise. We will show the importance of data pre-processing steps (e.g., PCA, UMAP for clustering) when applying this method.

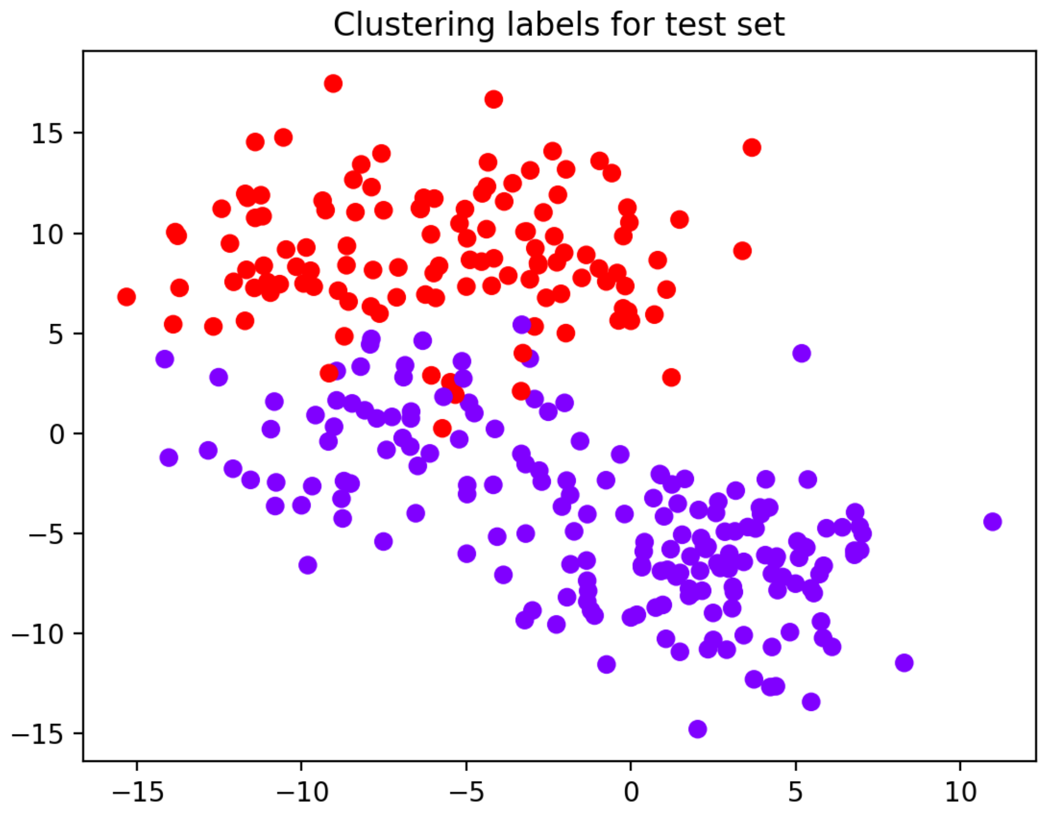
```
data_noisy = make_blobs(1000, 10, centers=5, random_state=42, cluster_std=3)
plt.scatter(data_noisy[0][:, 0],
            data_noisy[0][:, 1],
            c=data_noisy[1],
            cmap='rainbow_r')
plt.show()
```



```
Xnoise_tr, Xnoise_ts, ynoise_tr, ynoise_ts = train_test_split(data_noisy[0],
                                                               data_noisy[1],
                                                               test_size=0.30,
                                                               random_state=42,
                                                               stratify=data_noisy[1])

metrics_noise, nbest_noise = findbestclust.best_nclust(Xnoise_tr, iter_cv=10, strat_
↪vect=ynoise_tr)
out_noise = findbestclust.evaluate(Xnoise_tr, Xnoise_ts, nbest_noise)

plot_metrics(metrics_noise, title="Reval performance")

perm_lab_noise = kuhn_munkres_algorithm(ynoise_ts, out_noise.test_cllab)
plt.scatter(Xnoise_ts[:, 0], Xnoise_ts[:, 1],
            c=perm_lab_noise, cmap='rainbow_r')
plt.title("Clustering labels for test set")
```

We observe that the best number of clusters selected is equal to 2, which does not reflect the true label distributions of the synthetic dataset, although the misclassification performance during training is equal to 0 (see performance plot and scatterplot with predicted labels for the test set).


Reval performance
training set
validation set
Normalized stability
0.00
0.05
0.10
0.15
0.20
Number of clusters
2
3
4
5
6


Clustering labels for test set
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10

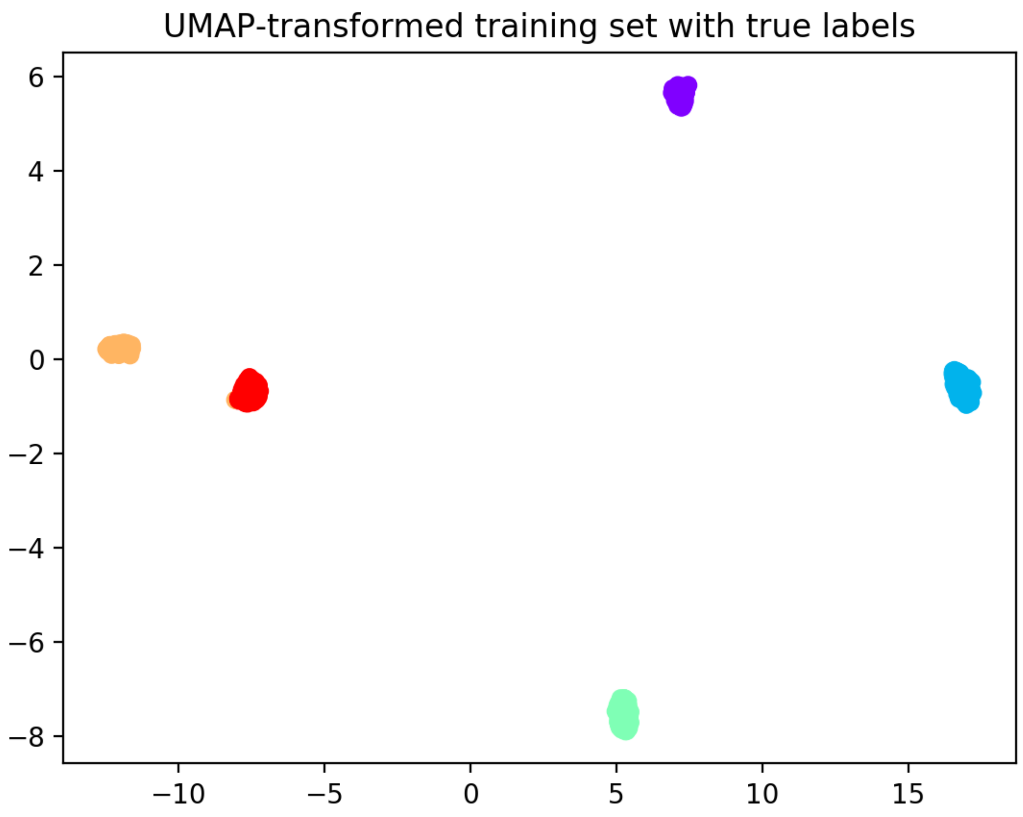
AMI score (0.59) and accuracy value (0.4) suggest that the model generalizes poorly on test set.

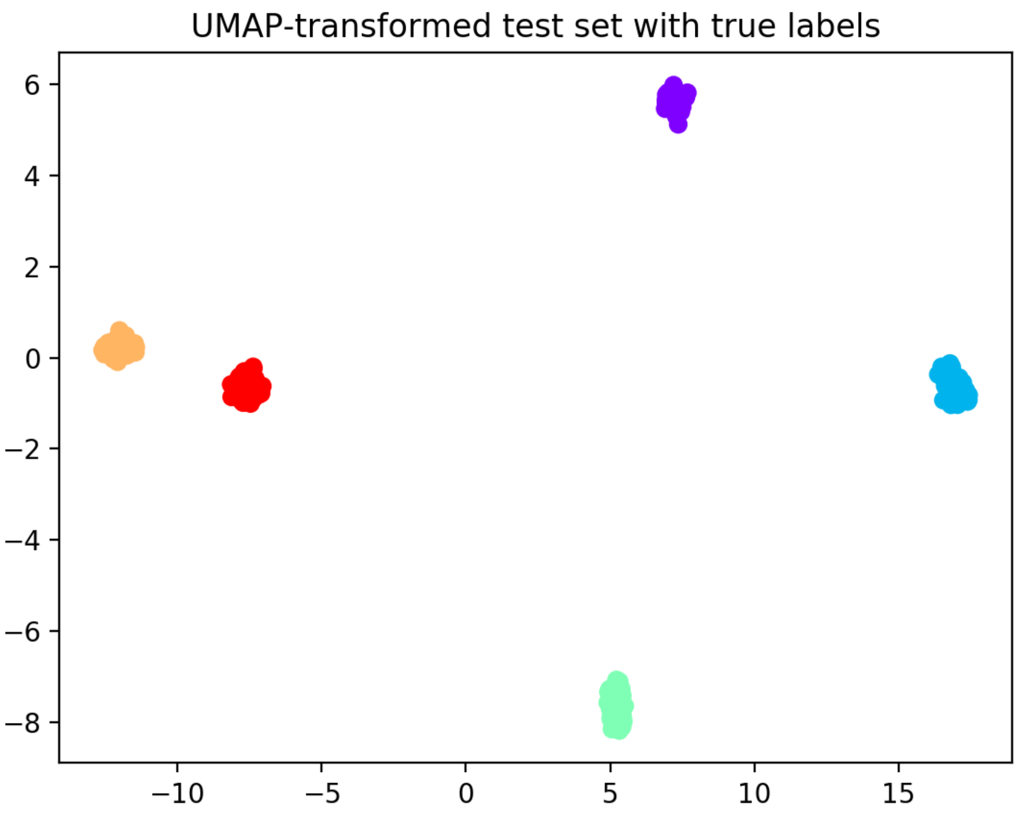
Uniform Manifold Approximation and Projection for Dimensionality Reduction (UMAP; McInnes et al., 2018) is a topology-based dimensionality reduction tool that can be used to pre-process data for clustering (see here). Applied to our noisy dataset with suggested parameters, we obtain that clusters are correctly identified visually as dense and separated blobs, that `reval` now easily detects.

McInnes, L, Healy, J, *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*, ArXiv e-prints 1802.03426, 2018.

```
from umap import UMAP

transform = UMAP(n_components=10, n_neighbors=30, min_dist=0.0)

Xtr_umap = transform.fit_transform(Xnoise_tr)
Xts_umap = transform.transform(Xnoise_ts)

plt.scatter(Xtr_umap[:, 0], Xtr_umap[:, 1],
            c=ynoise_tr, cmap='rainbow_r')
plt.title("UMAP-transformed training set with true labels")
plt.show()

plt.scatter(Xts_umap[:, 0], Xts_umap[:, 1],
            c=ynoise_ts, cmap='rainbow_r')
plt.title("UMAP-transformed test set with true labels")
plt.show()
```

Hereafter, we display UMAP pre-processed training and test sets. We fit the UMAP dimensionality reduction technique on the training set and then applied it to the test set to avoid inflation of performance scores on the test set.


UMAP-transformed training set with true labels
6
4
2
0
−2
−4
−6
−8
−10
−5
0
5
10
15

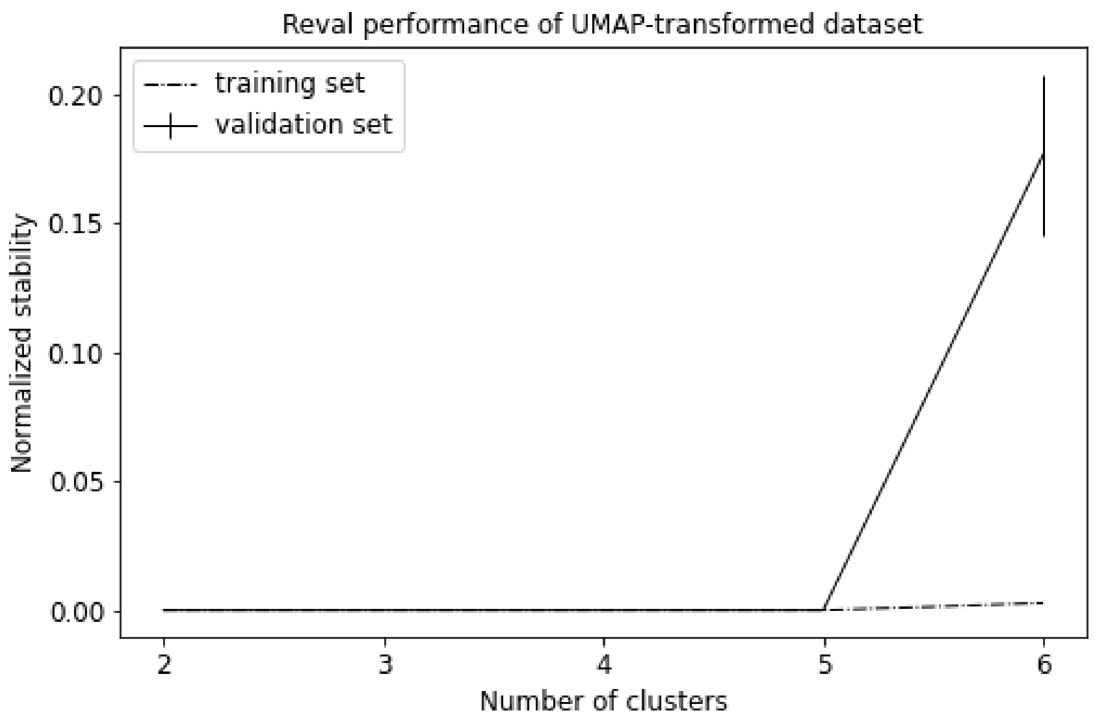


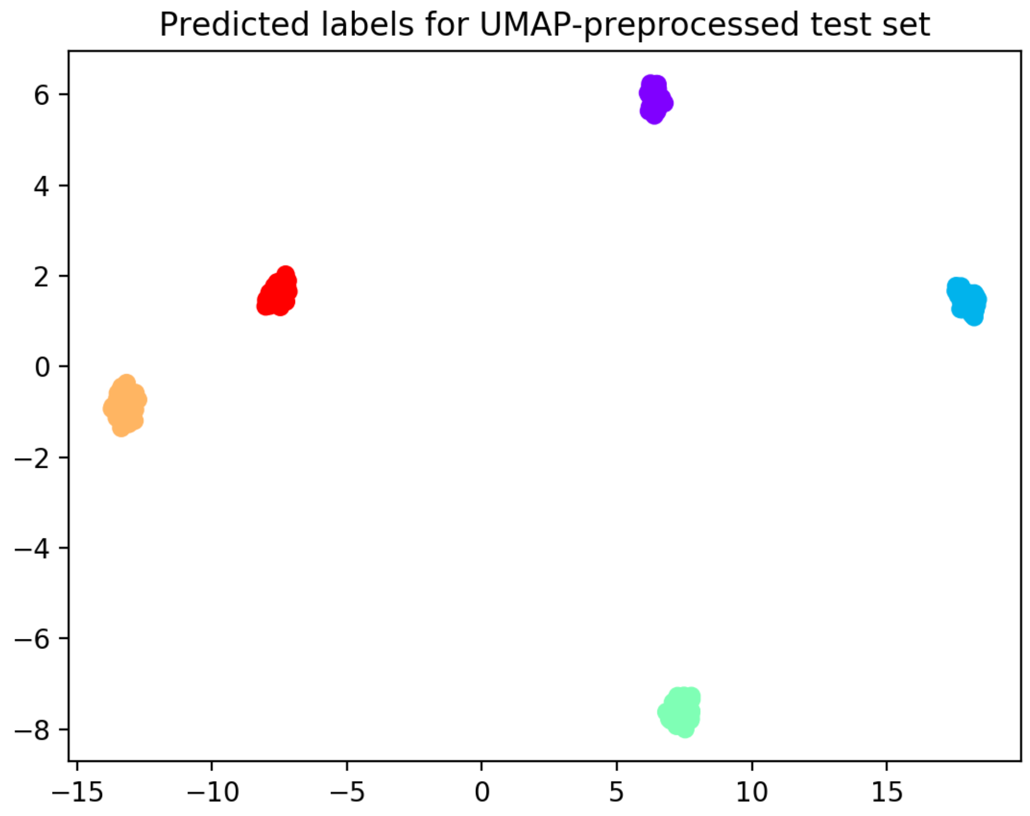
Now we apply `reval` method to the transformed dataset.

```
metrics, nbest = findbestclust.best_nclust(Xtr_umap, iter_cv=10, strat_vect=ynoise_tr)
out = findbestclust.evaluate(Xtr_umap, Xts_umap, nbest)

plot_metrics(metrics, title='Reval performance of UMAP-transformed dataset')

perm_noise = kuhn_munkres_algorithm(ynoise_ts, out.test_cllab)

print(f"Best number of clusters: {nbest}")
print(f"Test set external ACC: "
      f"{1 - zero_one_loss(ynoise_ts, perm_noise)}")
print(f'AMI = {adjusted_mutual_info_score(ynoise_ts, out.test_cllab)}')
print(f"Validation set normalized stability (misclassification): {metrics['val
↪'][nbest]}")
print(f"Result accuracy (on test set): "
      f"{out.test_acc}")

plt.scatter(Xts_umap[:, 0], Xts_umap[:, 1],
            c=perm_noise, cmap='rainbow_r')
plt.title("Predicted labels for UMAP-preprocessed test set")
```

We obtain that 5 clusters are identified (see performance plot) with ACC = 1.0; Normalized stability: 0.0 (0.0, 0.0).

Comparing clustering solution (see scatterplot below) with true labels we obtain AMI = 1.0; ACC: 1.0.


Reval performance of UMAP-transformed dataset
training set
validation set
Normalized stability
0.00
0.05
0.10
0.15
0.20
2
3
4
5
6
Number of clusters

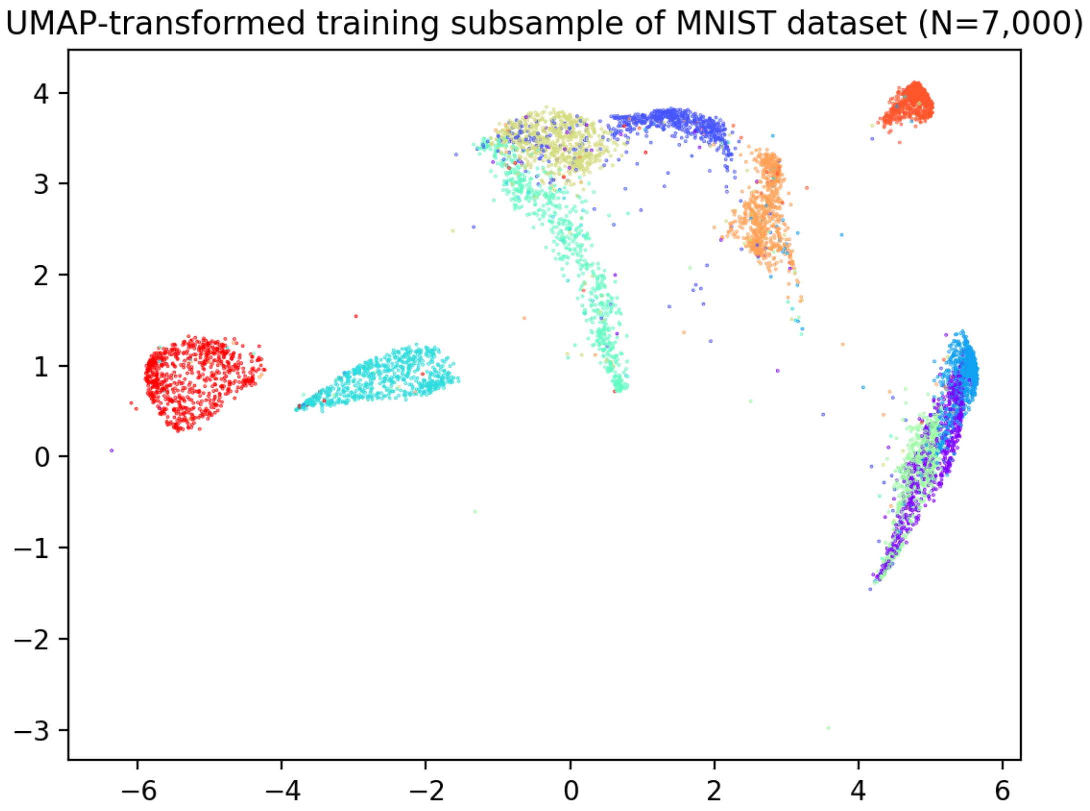

Predicted labels for UMAP-preprocessed test set
6
4
2
0
−2
−4
−6
−8
−15
−10
−5
0
5
10
15

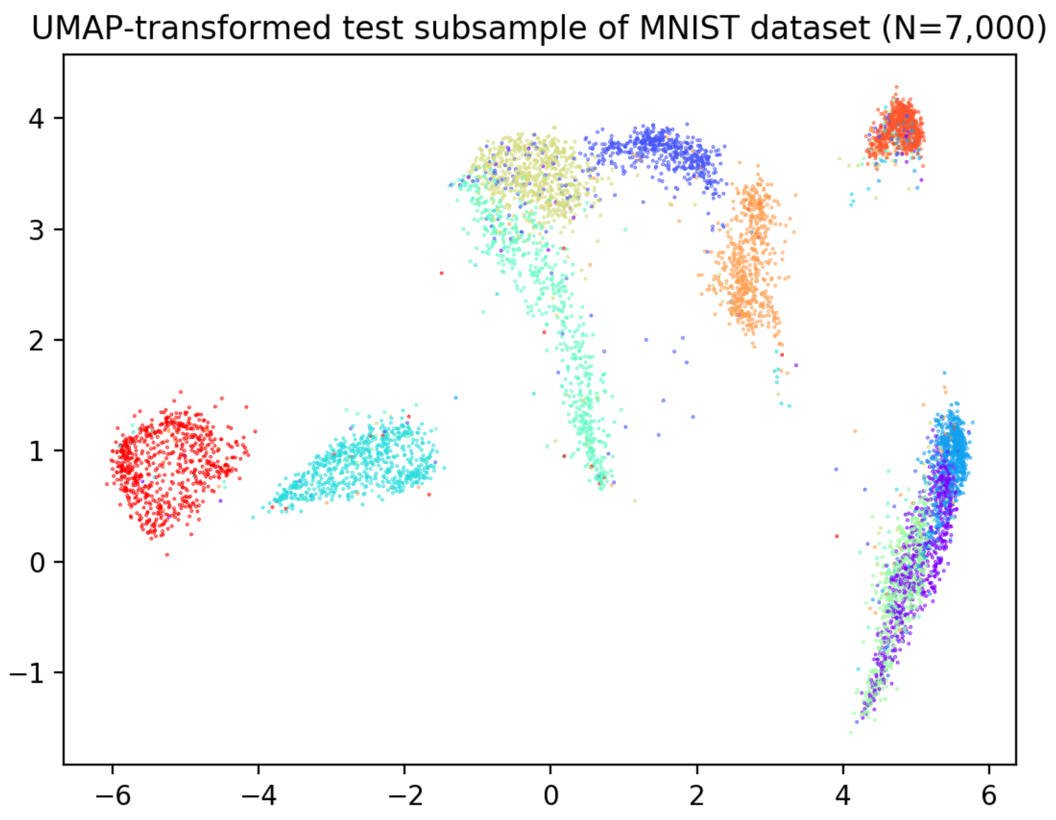
3.3 MNIST dataset

Remark: This example enables multiprocessing to speed up computations. ‘‘n_jobs‘‘ parameter in :class:‘FindBestClustCV‘ set to 7.

From `sklearn.datasets` we can import `fetch_openml` to load MNIST dataset. This dataset includes 70,000 28X28 images of 10 hand-written digits from 0 to 9. To speed up computations we select 14,000 samples that are divided into training and test sets at 50%. Then, we pre-processed these images with UMAP to reduce the number of features (from 784 to 10), see scatterplots below.

```python
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from reval.best_nclust_cv import FindBestClustCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import zero_one_loss, adjusted_mutual_info_score
import matplotlib.pyplot as plt
from umap import UMAP
from reval.visualization import plot_metrics
from reval.utils import kuhn_munkres_algorithm

# MNIST dataset with 10 classes
mnist, label = fetch_openml('mnist_784', version=1, return_X_y=True)
transform = UMAP(n_neighbors=30, min_dist=0.0, n_components=10, random_state=42)

# Stratified subsets of 7000 elements for both training and test set
mnist_tr, mnist_ts, label_tr, label_ts = train_test_split(mnist, label,
                                                          train_size=0.1,
                                                          test_size=0.1,
                                                          random_state=42,
                                                          stratify=label)

# Dimensionality reduction with UMAP as pre-processing step
mnist_tr = transform.fit_transform(mnist_tr)
mnist_ts = transform.transform(mnist_ts)

plt.scatter(mnist_tr[:, 0],
            mnist_tr[:, 1],
            c=label_tr.astype(int),
            s=0.1,
            cmap='rainbow_r')
plt.title('UMAP-transformed training subsample of MNIST dataset (N=7,000)')
plt.show()

plt.scatter(mnist_ts[:, 0], mnist_ts[:, 1],
            c=label_ts.astype(int), s=0.1, cmap='rainbow_r')
plt.title('UMAP-transformed test subsample of MNIST dataset (N=7,000)')
plt.show()
```


UMAP-transformed training subsample of MNIST dataset (N=7,000)
4
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6



We now apply `reval` with 10 repetitions of 2-fold cross-validation, number of clusters ranging from 2 to 11 and random labeling iterated 10 times. We again select hierarchical clustering with k-nearest neighbors classifier for number of cluster selection.

```
classifier = KNeighborsClassifier()
clustering = AgglomerativeClustering()

findbestclust = FindBestClustCV(nfold=2, nclust_range=list(range(2, 12)),
                                s=classifier, c=clustering, nrand=10, n_jobs=7)

metrics, nbest = findbestclust.best_nclust(mnist_tr, iter_cv=10, strat_vect=label_tr)
out = findbestclust.evaluate(mnist_tr, mnist_ts, nbest)

plot_metrics(metrics, title="Relative clustering validation performance on MNIST␣
↪dataset")

perm_lab = kuhn_munkres_algorithm(label_ts.astype(int), out.test_cllab)

plt.scatter(mnist_ts[:, 0], mnist_ts[:, 1],
            c=perm_lab, s=0.1, cmap='rainbow_r')
plt.title("Predicted labels for MNIST test set")
plt.show()

print(f"Best number of clusters: {nbest}")
print(f"Test set external ACC: "
      f"{1 - zero_one_loss(label_ts.astype(int), perm_lab)}")
```

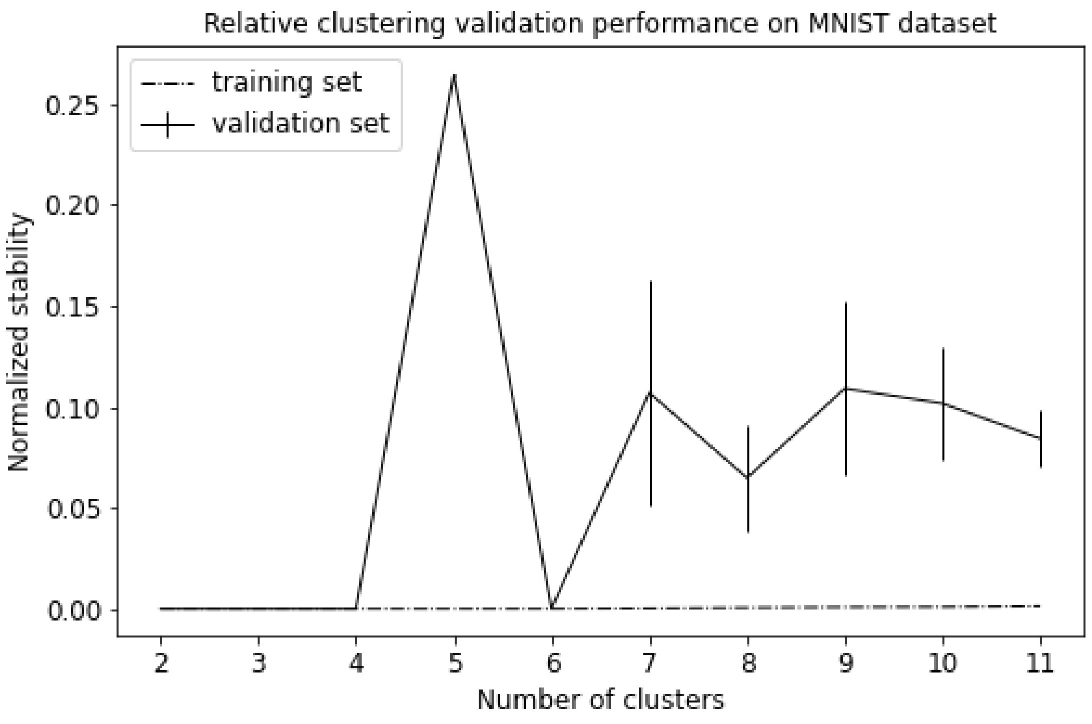
(continues on next page)

(continued from previous page)

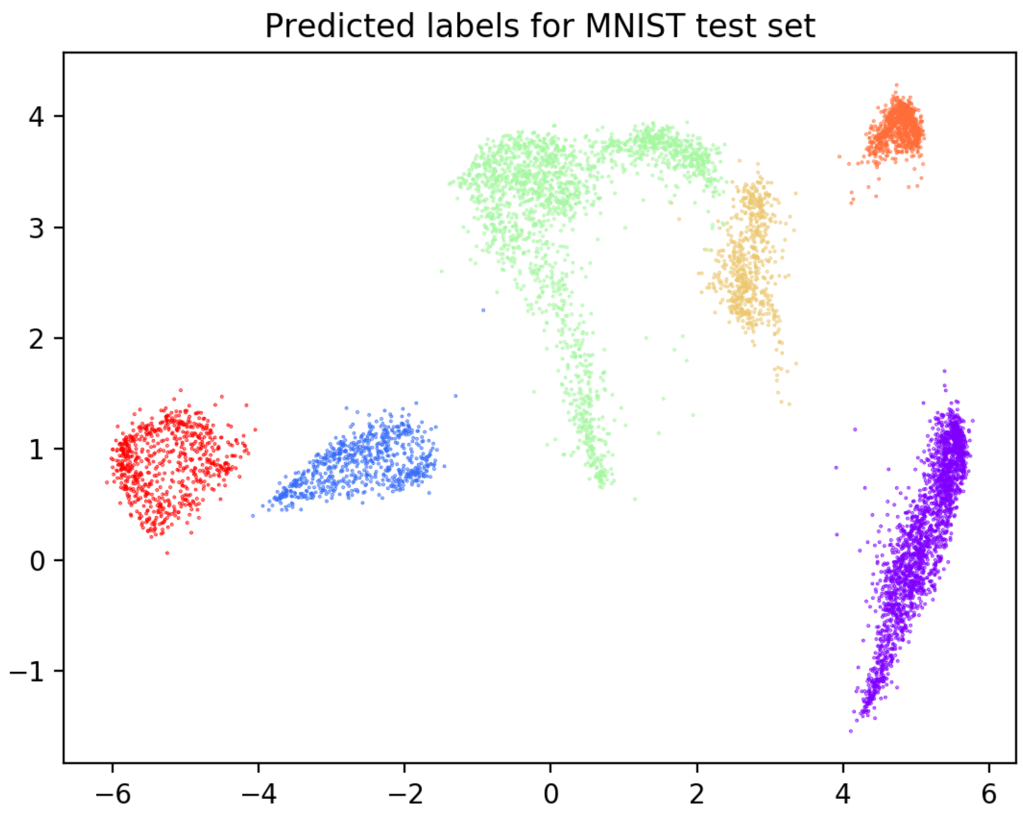
```
print(f'AMI = {adjusted_mutual_info_score(label_ts.astype(int), perm_lab)}')
print(f"Validation set normalized stability (misclassification): {metrics['val
↪'][nbest]}")
print(f"Result accuracy (on test set): "
      f"{out.test_acc}")
```

We obtain that the algorithm returns 6 as the best number of clusters (see performance plot). Comparing true and predicted labels we obtain a good AMI score, but a low accuracy score: AMI = 0.70; ACC = 0.58.

Whereas performance metrics during validation (normalized stability: mean 95% CI) and on test set (ACC) are low and high, respectively. Normalized stability: 0.002 (0.0, 0.003); ACC = 0.72.



We observe that the classes correctly identified are those that, after UMAP reduction, show good cohesion and separation, which is why the model performance is good. On the contrary, clusters that are closer together receive the same labels (see scatterplot below) and are misclassified. This lowers the external ACC score although returning a high AMI score, which is based on cluster overlaps.



In these situations attention should be put in:

1. Choosing the right clustering algorithm;
2. Pre-processing steps;
3. Whether `reval` is the right method to use with the data at hand (e.g., very noisy dataset with unknown labels).

3.4 More examples

Check out more examples including (1) repeated cross validation with HDBSCAN algorithm for the complete MNIST handwritten digits dataset, and (2) `reval` for classifier/clustering selection here. Code can be found in the cloned folder in *reval_clustering/working_examples*.

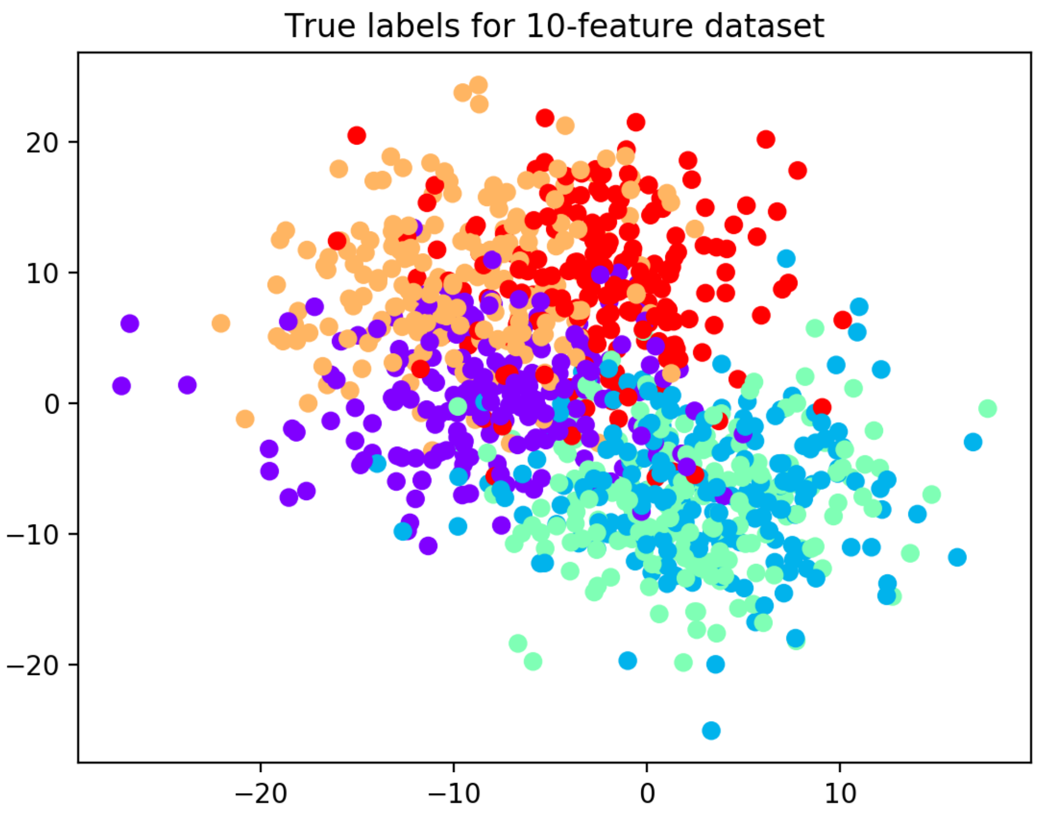
CHAPTER 4

Examples of how things can go wrong

We discuss now typical situations that might happen when processing real-world datasets and how these can modify `reval` performance. Code can be found in *reval_clustering/working_examples/*, `data_dimensionality.py` file.

4.1 Number of features: when enough is enough?

With `make_blobs` function from `sklearn.datasets`, we generate a noisy dataset (i.e., we set `cluster_std=5`) with 5 classes, 1,000 samples, and 10 features (see scatterplot below). We partition it into training and test sets (30%) and we apply the relative validation algorithm with one iteration of 10-fold cross-validation, number of clusters ranging from 2 to 6, k-nearest neighbors and hierarchical clustering as classification and clustering algorithms, respectively, and 100 iterations of random labeling.



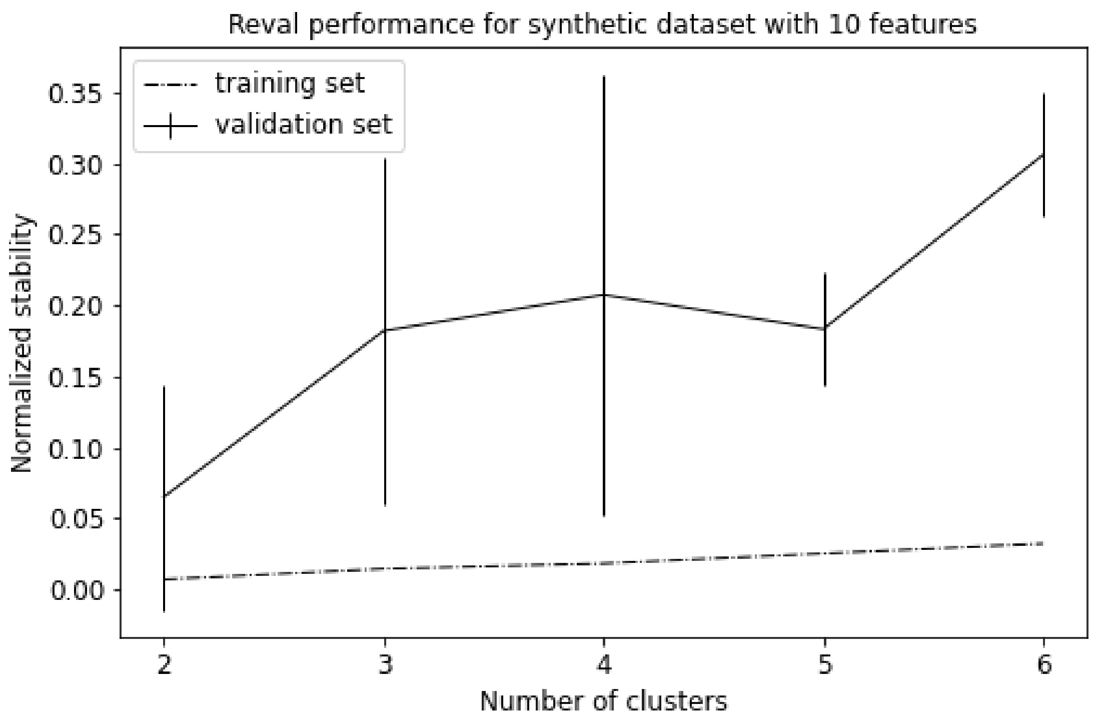
```python
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from reval.best_nclust_cv import FindBestClustCV
from reval.visualization import plot_metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_mutual_info_score, zero_one_loss
from reval.utils import kuhn_munkres_algorithm
import matplotlib.pyplot as plt
import numpy as np

data1 = make_blobs(1000, 10, centers=5, cluster_std=5, random_state=42)

plt.scatter(data1[0][:, 0], data1[0][:, 1],
            c=data1[1], cmap='rainbow_r')
plt.title('True labels for 10-feature dataset')
plt.show()

X_tr, X_ts, y_tr, y_ts = train_test_split(data1[0],
                                          data1[1],
                                          test_size=0.30,
                                          random_state=42,
                                          stratify=data1[1])
# Apply relative clustering validation with KNN and Hierarchical clustering
classifier = KNeighborsClassifier()
```

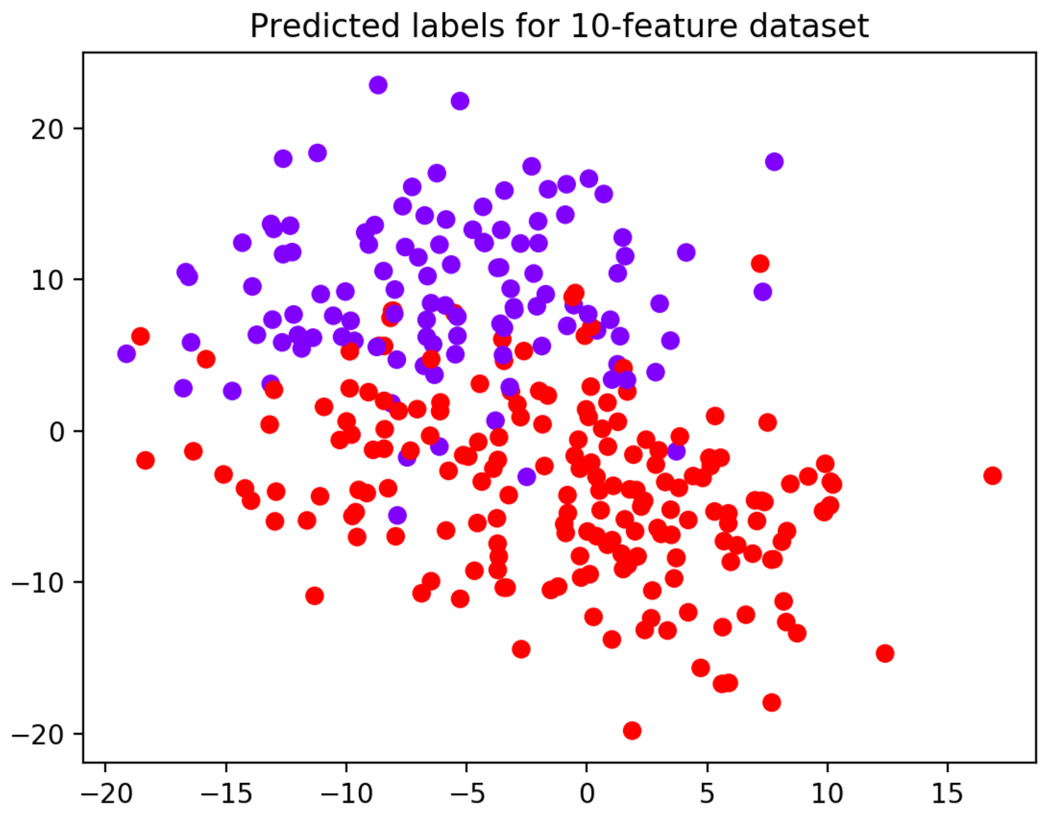
(continues on next page)

(continued from previous page)

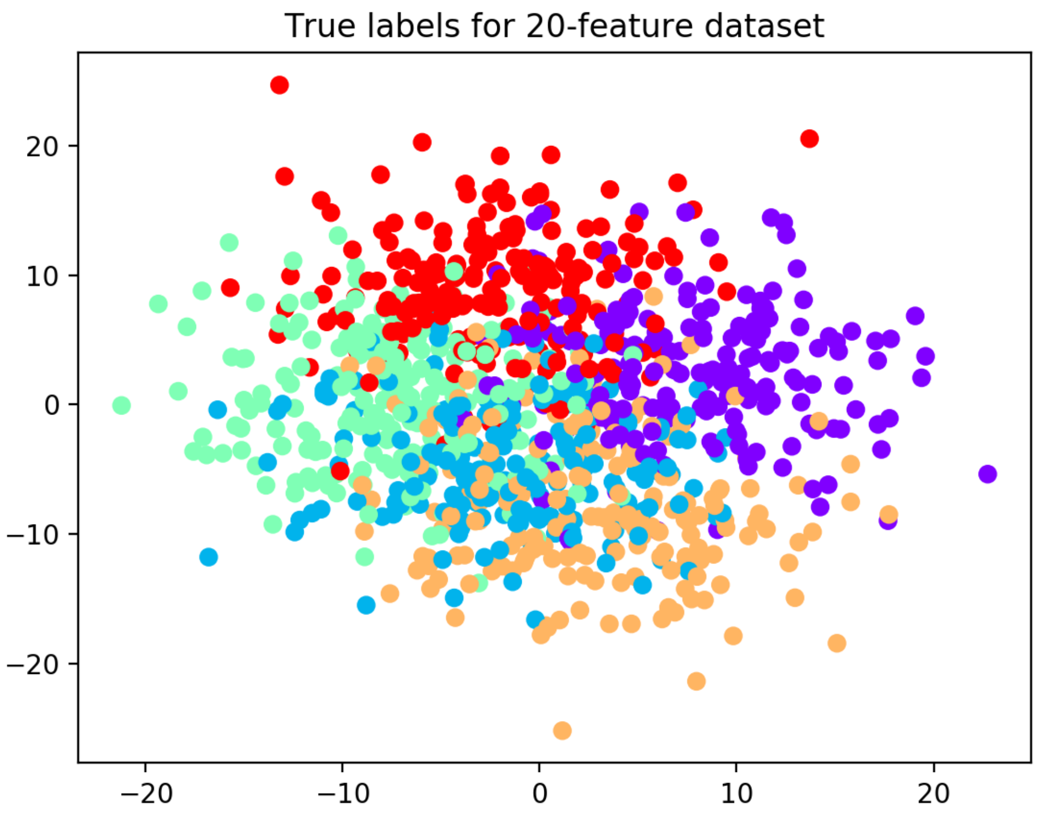
```
clustering = AgglomerativeClustering()

findbestclust = FindBestClustCV(nfold=10,
                                nclust_range=list(range(2, 7)),
                                s=classifier,
                                c=clustering,
                                nrand=100)
metrics, nbest = findbestclust.best_nclust(data=X_tr, strat_vect=y_tr)
out = findbestclust.evaluate(X_tr, X_ts, nbest)

plot_metrics(metrics, title="Reval performance for synthetic dataset with 10 features
↪")
```

The algorithm selects 2 as the best clustering solution (see performance plot and scatterplot with predicted labels).





We now increase the number of features from 10 to 20 and rerun the relative validation algorithm with the same parameters as before (see scatterplot with true labels below).



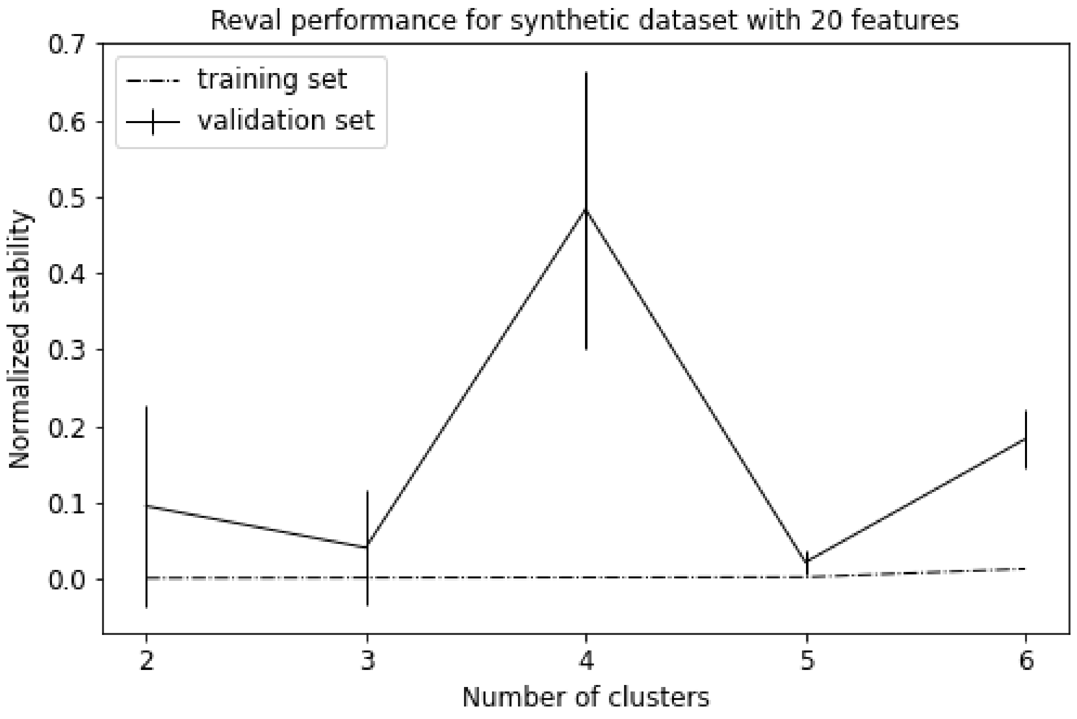
```
data2 = make_blobs(1000, 20, centers=5, cluster_std=5, random_state=42)

plt.scatter(data2[0][:, 0], data2[0][:, 1],
            c=data2[1], cmap='rainbow_r')
plt.title('True labels for 20-feature dataset')
plt.show()

X_tr, X_ts, y_tr, y_ts = train_test_split(data2[0],
                                          data2[1],
                                          test_size=0.30, random_state=42,
                                          stratify=data2[1])

findbestclust = FindBestClustCV(nfold=10, nclust_range=list(range(2, 7)),
                                s=classifier, c=clustering, nrand=100)
metrics, nbest = findbestclust.best_nclust(data=X_tr, strat_vect=y_tr)
out = findbestclust.evaluate(X_tr, X_ts, nbest)

plot_metrics(metrics, title="Reval performance for synthetic dataset with 20 features
↪")

plt.scatter(X_ts[:, 0], X_ts[:, 1],
            c=out.test_cllab, cmap='rainbow_r')
plt.title("Predicted labels for 20-feature dataset")
plt.show()
```

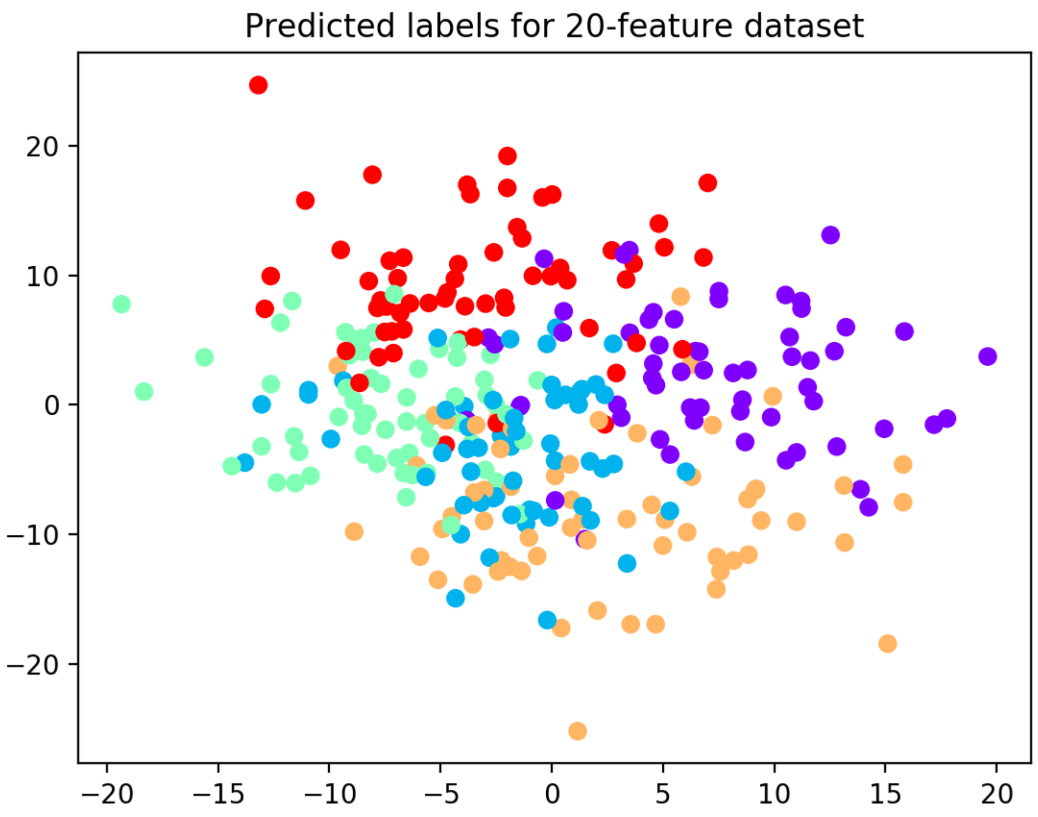
(continues on next page)

(continued from previous page)

```
print(f'AMI test set = {adjusted_mutual_info_score(y_ts, out.test_cllab)}')
relabeling = kuhn_munkres_algorithm(y_ts, out.test_cllab)
print(f'ACC test set = {1 - zero_one_loss(y_ts, relabeling)}')
```

Because we increased the space volume, data become more sparse, but still preserving their group structure. For this reason, now the algorithm is able to detect all 5 clusters. (See performance plot and scatterplot).





We use the adjusted mutual information score (AMI) to account for the amount of information shared between true labels and clustering labels returned by the algorithm. AMI returns 1 when two partitions are identical. Accuracy (ACC) is also used to compare the solutions after the clustering labels have been permuted to match true labels. On the test set, we obtain: AMI = 0.98; ACC = 0.99.

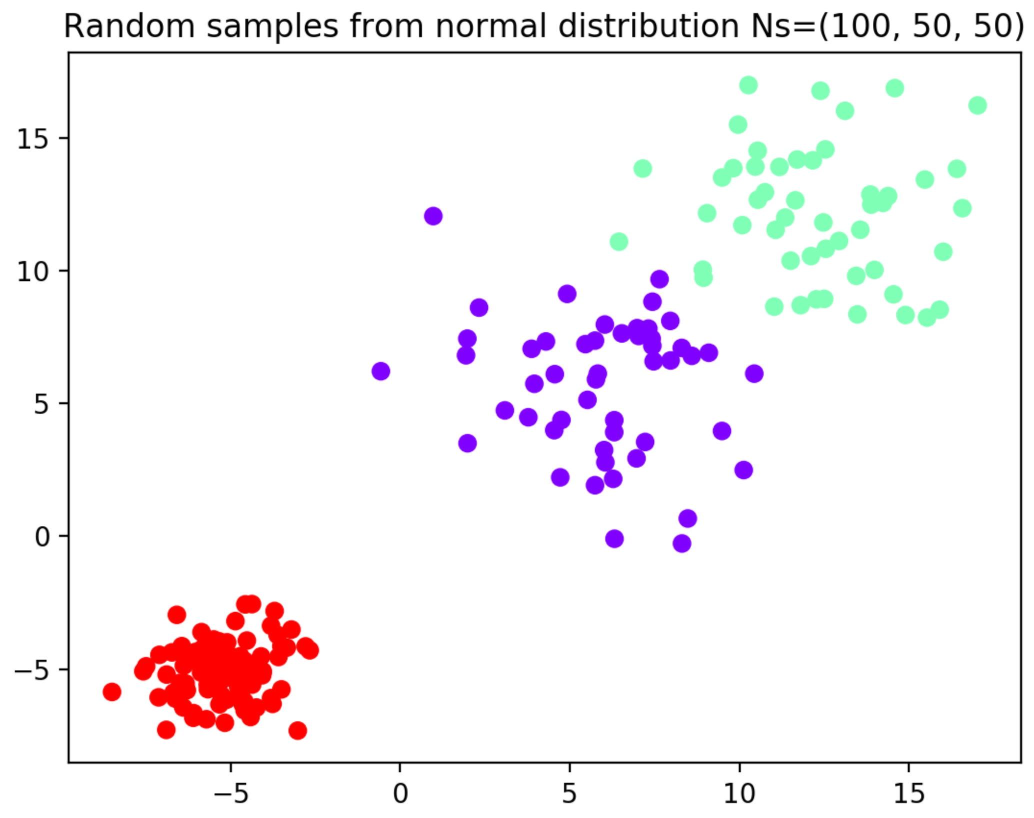
Remark: in situations where we are able to increase the number of features for a dataset, it is important to remember the curse of dimensionality, i.e., the increase of the space dimension determines available data to become sparse and the number of samples required to detect an effect to grow exponentially. For this reason, increasing the number of features might not help detect dataset subgroups, because the data would become sparse, hence losing their structure.

4.2 Number of samples: too few, not good

In small datasets, that we suppose partitioned into groups, the number of samples is important to an algorithm result. Too few samples, in fact, are usually not representative of data distributions and may hinder clustering results. In the following, we randomly sample three groups from normal distributions and we show how `reval` is able to identify the right number of subgroups only if the number of samples is enough for subgroups with greater standard deviation to reliably represent the different distributions.

The first dataset generated comprises (see scatterplot):

- Group 1 (red): N = 100 random samples from normal distribution with m = -5; sd = 1
- Group 2 (purple): N = 50 random samples from normal distribution with m = 12; sd = 2.5
- Group 3 (green): N = 50 random samples from normal distribution with m = 6; sd = 2.5



We instantiate `FindBestClustCV()` class with one repetition of 10-fold cross validation, k-nearest neighbors classifier and hierarchical clustering, number of clusters ranging from 2 to 6, and 100 random labeling iterations.

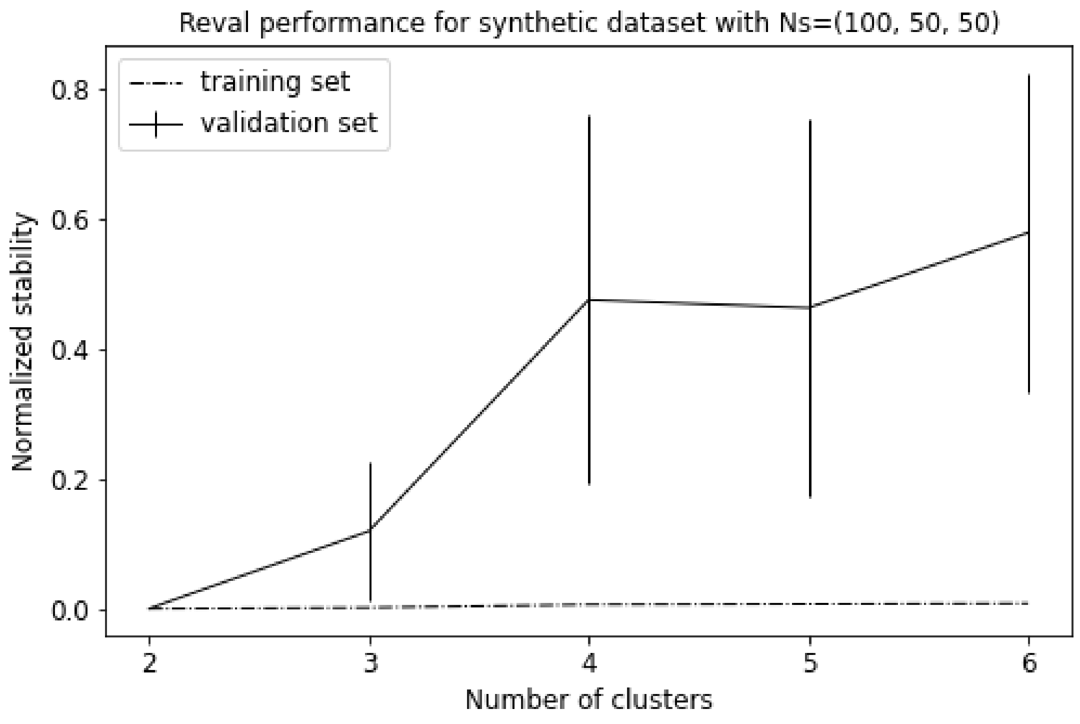
```
# Set seed for reproducible examples
np.random.seed(42)

# We generate three random samples from normal distributions
data1 = np.random.normal(-5, size=(100, 2))
data2 = np.random.normal(12, 2.5, size=(50, 2))
data3 = np.random.normal(6, 2.5, size=(50, 2))
data = np.append(data1, data2, axis=0)
data = np.append(data, data3, axis=0)

label = [0] * 100 + [1] * 50 + [2] * 50

plt.scatter(data[:, 0], data[:, 1],
            c=label, cmap='rainbow_r')
plt.title('Random samples from normal distribution Ns=(100, 50, 50)')
plt.show()

classifier = KNeighborsClassifier()
clustering = AgglomerativeClustering()

X_tr, X_ts, y_tr, y_ts = train_test_split(data, label,
                                          test_size=0.30,
                                          random_state=42,
```

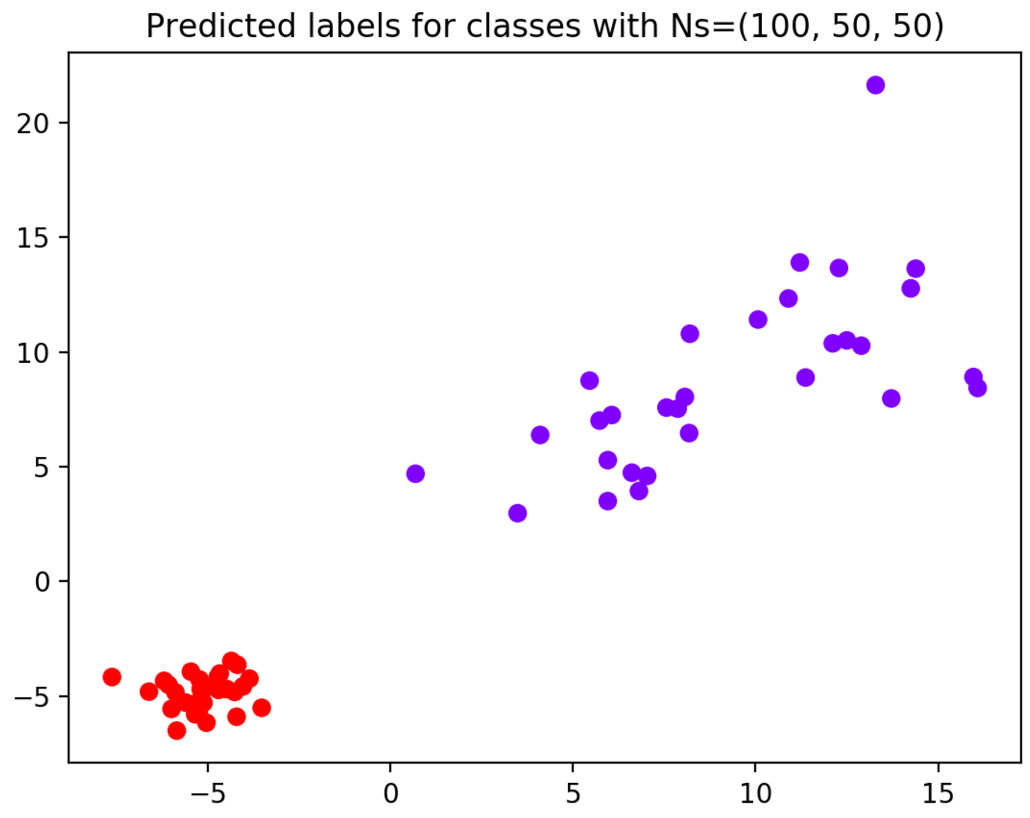
(continues on next page)

(continued from previous page)

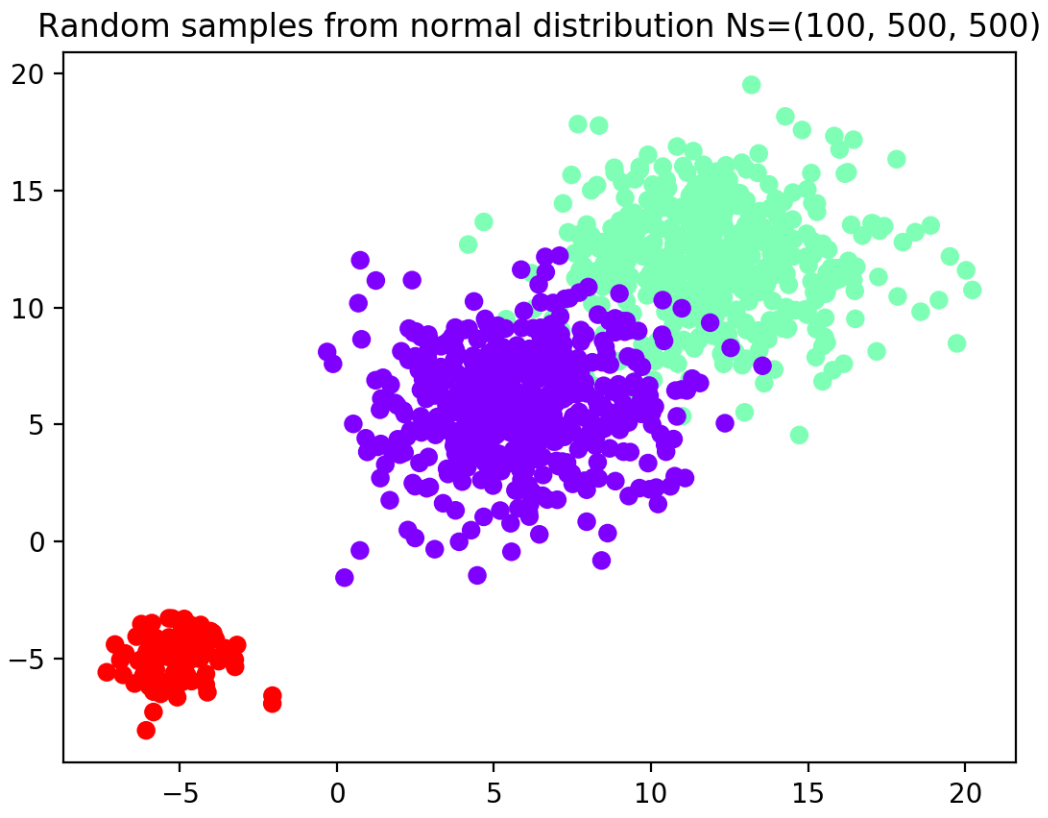
```
                                                                stratify=label)

# Apply relative clustering validation with KNN and Hierarchical clustering
findbestclust = FindBestClustCV(nfold=10, nclust_range=list(range(2, 7)),
                                s=classifier, c=clustering, nrand=100)
metrics, nbest = findbestclust.best_nclust(data=X_tr, strat_vect=y_tr)
out = findbestclust.evaluate(X_tr, X_ts, nbest)
plot_metrics(metrics, title="Reval performance for synthetic dataset with Ns=(100, 50,
↪ 50)")

plt.scatter(X_ts[:, 0], X_ts[:, 1],
            c=kuhn_munkres_algorithm(np.array(y_ts),
                                     out.test_cllab),
            cmap='rainbow_r')
plt.title(f'Predicted labels for classes with Ns=(100, 50, 50)')
plt.show()
```

Result reports 2 as the best clustering solution (see performance plot and scatterplot). Groups 2 and 3, i.e., with least number of subjects and higher standard deviation, are considered as a unique group by the algorithm.





To fix this, we try to increase the number of samples for groups 2 and 3 from 50 to 500 (see scatterplot) and we rerun the algorithm with the same parameters.



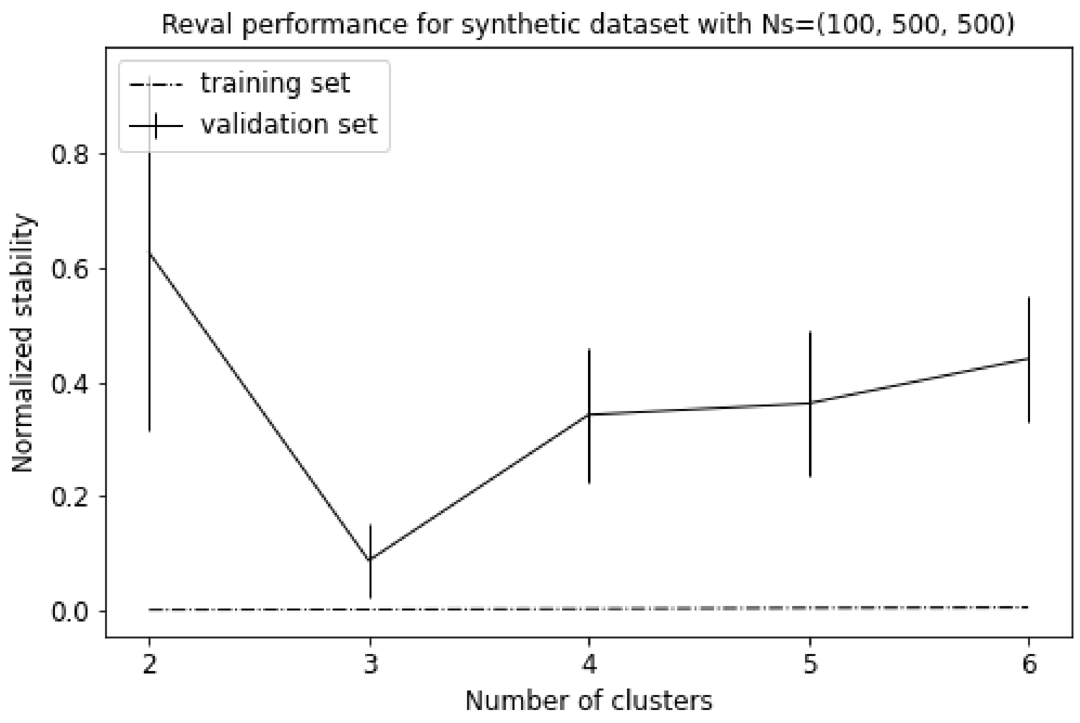
```
# We now increase the number of samples in groups 2 and 3 to 500
data1 = np.random.normal(-5, size=(100, 2))
data2 = np.random.normal(12, 2.5, size=(500, 2))
data3 = np.random.normal(6, 2.5, size=(500, 2))
data = np.append(data1, data2, axis=0)
data = np.append(data, data3, axis=0)

label = [0] * 100 + [1] * 500 + [2] * 500

plt.scatter(data[:, 0], data[:, 1],
            c=label, cmap='rainbow_r')
plt.title('Random samples from normal distribution Ns=(100, 500, 500)')
plt.show()

classifier = KNeighborsClassifier()
clustering = AgglomerativeClustering()

X_tr, X_ts, y_tr, y_ts = train_test_split(data, label,
                                          test_size=0.30,
                                          random_state=42,
                                          stratify=label)

# Apply relative clustering validation with KNN and Hierarchical clustering
findbestclust = FindBestClustCV(nfold=10, nclust_range=list(range(2, 7)),
                                s=classifier, c=clustering, nrand=100)
```

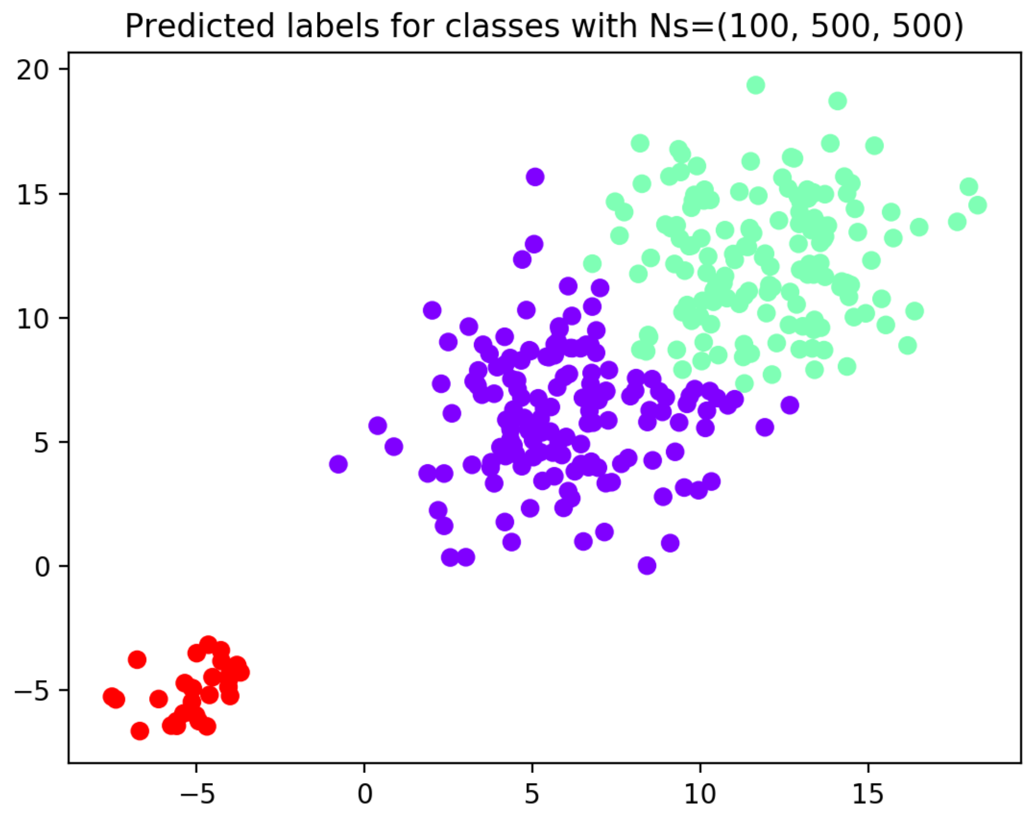
(continues on next page)

(continued from previous page)

```
metrics, nbest = findbestclust.best_nclust(X_tr, strat_vect=y_tr)
out = findbestclust.evaluate(X_tr, X_ts, nbest)
plot_metrics(metrics, title="Reval performance for synthetic dataset with Ns=(100,␣
↪500, 500)")

plt.scatter(X_ts[:, 0], X_ts[:, 1],
            c=y_ts,
            cmap='rainbow_r')
plt.title(f'Test set true labels for classes with Ns=(100, 500, 500)')
plt.show()

plt.scatter(X_ts[:, 0], X_ts[:, 1],
            c=kuhn_munkres_algorithm(np.array(y_ts),
                                     out.test_cllab),
            cmap='rainbow_r')
plt.title(f'Predicted labels for classes with Ns=(100, 500, 500)')
plt.show()

# Performance scores
# Test set ACC
print(f'Test set external '
      f'ACC = {1 - zero_one_loss(y_ts, kuhn_munkres_algorithm(np.array(y_ts), out.
↪test_cllab))}')
print(f"Validation stability metrics: {metrics['val'][nbest]}")
print(f"Test set model ACC = {out.test_acc}")
print(f"AMI = {adjusted_mutual_info_score(y_ts, out.test_cllab)}")
```

This time the algorithm correctly identifies all three groups (see performance plot and scaterplot with predicted labels).





To evaluate the algorithm performance we compute AMI and ACC between the true and `reval` partitions and report the validation and testing metrics, i.e., normalized stability with 95% confidence interval and testing accuracy, respectively. AMI = 0.79; ACC (external) = 0.95; Normalized stability: 0.08 (0.02; 0.14); ACC = 0.99

Increasing the sampling size, the algorithm was able to correctly identify the three distributions.

CHAPTER 5

Code description

`reval` module has one superclass `FindBestClustCV` and a subclass `RelativeValidation`. `SCParamSelection` and `ParamSelection` have been added to later releases to perform hyperparameter selections.

5.1 Classes

class `reval.relative_validation.`**`RelativeValidation`**(*s*, *c*, *nrand=10*)

This class allows to perform the relative clustering validation procedure. A supervised algorithm is required to test cluster stability. Labels output from a clustering algorithm are used as true labels.

Parameters

- **s** (*class*) – initialized class for the supervised method.
- **c** (*class*) – initialized class for clustering algorithm.
- **nrand** (*int*) – number of iterations to normalize cluster stability.

`_rescale_score_`(*xtr*, *xts*, *randlabtr*, *labts*)

Private method that computes the misclassification error when predicting test labels with classification model fitted on training set with random labels.

Parameters

- **xtr** (*ndarray, (n_samples, n_features)*) – training dataset.
- **xts** (*ndarray, (n_samples, n_features)*) – test dataset.
- **randlabtr** (*ndarray, (n_samples,)*) – random labels.
- **labts** (*ndarray, (n_samples,)*) – test set labels.

Returns misclassification error.

Return type float

rndlabels_traineval(*train_data*, *test_data*, *train_labels*, *test_labels*)

Method that performs random labeling on the training set (N times according to `reval.relative_validation.RelativeValidation.nrand` instance attribute) and evaluates the fitted models on test set.

Parameters

- **train_data** (*ndarray, (n_samples, n_features)*) – training dataset.
- **test_data** (*ndarray, (n_samples, n_features)*) – test dataset.
- **train_labels** (*ndarray, (n_samples,)*) – training set clustering labels.
- **test_labels** (*ndarray, (n_samples,)*) – test set clustering labels.

Returns averaged misclassification error on the test set.

Return type float

test(*test_data*, *fit_model*)

Method that compares test set clustering labels (i.e., A(X'), computed by `reval.relative_validation.RelativeValidation.clust_method`) against the (permuted) labels obtained through the classification algorithm fitted to the training set (i.e., f(X'), computed by `reval.relative_validation.RelativeValidation.class_method`). It returns the misclassification error, together with both clustering and classification labels.

Parameters

- **test_data** (*ndarray, (n_samples, n_features)*) – test dataset.
- **fit_model** (*class*) – fitted supervised model.

Returns misclassification error, clustering and classification labels.

Return type float, dictionary of ndarrays (n_samples,)

train(*train_data*, *tr_lab=None*)

Method that performs training. It compares the clustering labels on training set (i.e., A(X) computed by `reval.relative_validation.RelativeValidation.clust_method`) against the labels obtained from the classification algorithm (i.e., f(X), computed by `reval.relative_validation.RelativeValidation.class_method`). It returns the misclassification error, the supervised model fitted to the data, and both clustering and classification labels.

Parameters

- **train_data** (*ndarray, (n_samples, n_features)*) – training dataset.
- **tr_lab** (*list*) – cluster labels found during CV for clustering methods with no *n_clusters* parameter. If not None the clustering method is not performed on the whole test set. Default None.

Returns misclassification error, fitted supervised model object, clustering and classification labels.

Return type float, object, ndarray (n_samples,)

class reval.best_nclust_cv.**FindBestClustCV**(*s*, *c*, *nrand*, *nfold=2*, *n_jobs=1*, *nclust_range=None*)

Child class of *reval.relative_validation.RelativeValidation*. It performs (repeated) k-fold cross validation on the training set to select the best number of clusters, i.e., the number that minimizes the normalized stability (i.e., average misclassification error/asymptotic misclassification rate).

Parameters

- **nfold** (*int*) – number of CV folds.

- **nclust_range** (*list of int*) – list with clusters to look for, default None.
- **s** (*class*) – classification object inherited from *reval.relative_validation.RelativeValidation*.
- **c** (*class*) – clustering object inherited from *reval.relative_validation.RelativeValidation*.
- **nrand** (*int*) – number of random labeling iterations to compute asymptotic misclassification rate, inherited from *reval.relative_validation.RelativeValidation* class.
- **n_jobs** (*int*) – number of processes to be run in parallel, default 1.

Attribute cv_results_ dataframe with cross validation results. Columns are ncl = number of clusters; ms_tr = misclassification training; ms_val = misclassification validation.

static _fit (*data_obj*, *idxs*, *ncl=None*)

Function that calls training, test, and random labeling.

Parameters

- **data_obj** (*tuple*) – dataset and *reval.RelativeValidation* class.
- **idxs** (*tuple*) – lists of training and validation indices.
- **ncl** (*int*) – number of clusters, default None

Returns number of clusters and misclassification errors for training and validation.

Return type tuple (int, float, float)

best_nclust (*data*, *iter_cv=1*, *strat_vect=None*)

This method takes as input the training dataset and the stratification vector (if available) and performs a (repeated) CV procedure to select the best number of clusters that minimizes normalized stability.

Parameters

- **data** (*ndarray, (n_samples, n_features)*) – training dataset.
- **iter_cv** (*integer*) – number of iteration for repeated CV, default 1.
- **strat_vect** (*ndarray, (n_samples,)*) – vector for stratification, defaults to None.

Returns CV metrics for training and validation sets, best number of clusters, misclassification errors at each CV iteration.

Return type dictionary, int, (list) if n_clusters parameter is not available

evaluate (*data_tr*, *data_ts*, *nclust=None*, *tr_lab=None*)

Method that applies the selected clustering algorithm with the best number of clusters to the test set. It returns clustering labels.

Parameters

- **data_tr** (*ndarray, (n_samples, n_features)*) – training dataset.
- **data_ts** (*ndarray, (n_samples, n_features)*) – test dataset.
- **nclust** (*int*) – best number of clusters, default None.
- **tr_lab** (*array-like*) – clustering labels for the training set. If not None the clustering algorithm is not performed and the classifier is fitted. Available for clustering methods without *n_clusters* parameter. Default None.

Returns labels and accuracy for both training and test sets.

Return type namedtuple, (train_cllab: array, train_acc:float, test_cllab:array, test_acc:float)

class `reval.param_selection.`**`SCParamSelection`**(*sc_params*, *cv*, *nrand*, *n_jobs*, *iter_cv=1*, *clust_range=None*, *strat=None*)

Class that implements grid search cross-validation in parallel to select the best combination of classifier/clustering methods.

Parameters

- **`sc_params`** (*`dict`*) – dictionary of the form {'s': list, 'c': list} including the lists of classifiers and clustering methods to fit to the data.
- **`cv`** (*`int`*) – cross-validation folds.
- **`nrand`** (*`int`*) – number of random label iterations.
- **`n_jobs`** (*`int`*) – number of jobs to run in parallel, default (number of cpus - 1).
- **`iter_cv`** (*`int`*) – number of repeated cv, default 1.
- **`clust_range`** (*`list`*) – list with number of clusters to investigate, default None.
- **`strat`** (*`numpy array`*) – stratification vector for cross-validation splits, default None.

Attribute **cv_results_** cross-validation results that can be directly transformed to a dataframe. Key names: 's', 'c', 'best_nclust', 'mean_train_score', 'sd_train_score', 'mean_val_score', 'sd_val_score', 'validation_meanerror'. Dictionary of lists.

Attribute **best_param_** best solution(s) selected (minimum validation error). List.

Attribute **best_index_** index/indices of the best solution(s). Values correspond to the rows of the *cv_results_* table. List.

`_run_gridsearchcv`(*data*, *sc*)

Private function with different initializations of *`reval.best_nclust_cv.FindBestClustCV`*.

Parameters

- **`data`** (*`numpy array`*) – input dataset.
- **`sc`** (*`dict`*) – classifier/clustering of the form {'s':, 'c':}.

Returns performance list.

Return type list

`fit`(*data_tr*, *nclass=None*)

Class method that performs grid search cross-validation on training data. If the number of true classes is known, the method returns both the best result with the correct number of clusters (and minimum stability), if available, and the overall best result (overall minimum stability). The output reports None if the clustering algorithm does not find any cluster (e.g., HDBSCAN label all points as -1).

Parameters

- **`data_tr`** (*`numpy array`*) – training dataset.
- **`nclass`** (*`int`*) – number of true classes, default None.

class `reval.param_selection.`**`ParamSelection`**(*params*, *cv*, *s*, *c*, *nrand*, *n_jobs*, *iter_cv=1*, *strat=None*, *clust_range=None*)

Class that implements grid search cross-validation in parallel to select the best combinations of parameters for fixed classifier/clustering algorithms.

Parameters

- **params** (*dict*) – dictionary of dictionaries of the form {'s': {classifier parameter grid}, 'c': {clustering parameter grid}}. If one of the two dictionary of parameters is not available, initialize key but leave dictionary empty.
- **cv** (*int*) – cross-validation folds.
- **clust_range** (*list*) – list with number of clusters to investigate.
- **n_jobs** (*int*) – number of jobs to run in parallel, default (number of cpus - 1).
- **iter_cv** (*int*) – number of repeated cv loops, default 1.
- **strat** (*numpy array*) – stratification vector for cross-validation splits, default None.

Attribute **cv_results_** cross-validation results that can be directly transformed to a dataframe. Key names: classifier parameters, clustering parameters, 'best_nclust', 'mean_train_score', 'sd_train_score', 'mean_val_score', 'sd_val_score', 'validation_meanerror'. Dictionary of lists.

Attribute **best_param_** best solution(s) selected (minimum validation error). List.

Attribute **best_index_** index/indices of the best solution(s). Values correspond to the rows of the *cv_results_* table. List.

_allowed_par (*par_dict*)
Private method that controls the allowed parameter combinations for hierarchical clustering.

Parameters **par_dict** (*dict*) – clustering parameter grid.

Returns whether the parameter combination can be allowed.

Return type bool

_run_gridsearchcv (*data*, *param_s*, *param_c*)
Private method that initializes classifier/clustering with different parameter combinations and *reval.best_nclust_cv.FindBestClustCV*.

Parameters

- **data** (*numpy array*) – training dataset.
- **param_s** – dictionary of classifier parameters.
- **param_c** (*dict*) – dictionary of clustering parameters.

Type dict

Returns performance list.

Return type list

fit (*data_tr*, *nclass=None*)
Class method that performs grid search cross-validation on training data. It deals with the error due to wrong parameter combinations (e.g., ward linkage with no euclidean affinity). If the true number of classes is know, the method selects both the best parameter combination that selects the true number of clusters (minimum stability) and the best parameter combination that minimizes overall stability.

Parameters

- **data_tr** (*numpy array*) – training dataset.
- **nclass** (*int*) – number of true classes, default None.

5.2 Functions

Useful functions that can be used on their own are also available. In particular, `reval.utils.kuhn_munkres_algorithm` is an implementation of the Kuhn-Munkres algorithm (Kuhn, 1955; Munkres, 1957), that performs consistent permutation of predicted labels in order to minimize the misclassification error with respect to true labels. `reval.utils.compute_metrics` takes as input clustering and classification labels and returns classification metrics, such as F1 score, accuracy and Matthews correlation coefficient for generalization.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2), 83-97.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1), 32-38.

`reval.utils.`**`kuhn_munkres_algorithm`**(*true_lab*, *pred_lab*)

Function that implements the Kuhn-Munkres algorithm method. It selects the best label permutation of the predicted labels that minimizes the misclassification error when compared to the true labels. In order to allow for the investigation of replicability of findings between training and test sets, in the context of reval we permute clustering labels to match classification labels, in order to retain the label organization based on training dataset. This because otherwise we would loose the correspondence between training and test sets labels.

Parameters

- **`true_lab`** (*`ndarray, (n_samples,)`*) – classification algorithm labels (for reval).
- **`pred_lab`** (*`ndarray, (n_samples,)`*) – clustering algorithm labels (for reval).

Returns permuted labels that minimize the misclassification error.

Return type ndarray, (n_samples,)

`reval.utils.`**`compute_metrics`**(*class_labels*, *clust_labels*, *perm=False*)

Function that computes useful classification metrics. If needed the clustering labels are permuted with :reval.utils.kuhn_munkres_algorithm: The function returns a dictionary with ACC, MCC, F1, precision, recall as keys for accuracy, Matthews correlation coefficient, F1 score, precision, and recall, respectively.

Parameters

- **`class_labels`** (*`array-like`*) – labels returned by the classifier.
- **`clust_labels`** (*`array-like`*) – labels returned by the clustering.
- **`perm`** (*`bool`*) – flag to enable permutation of clustering labels, default False.

Returns dictionary of scores.

Return type dict

The `reval.best_nclust_cv._confint` computes 95% confidence interval using `scipy.stats.t.ppf()` function.

`reval.best_nclust_cv.`**`_confint`**(*vect*)

Private function to compute confidence interval.

Parameters **`vect`** (*`array-like`*) – performance scores.

Returns mean and error.

Return type tuple

The module `reval.internal_baselines` includes functions `select_best` and `evaluate_best` that allow comparisons between `reval` method and internal validation measures.

`reval.internal_baselines.`**`select_best`**(*data*, *c*, *int_measure*, *select='max'*, *nclust_range=None*)

Select the best number of clusters that minimizes/maximizes the internal measure selected.

Parameters

- **`data`** (*`array-like`*) – dataset.
- **`c`** (*`obj`*) – clustering algorithm class.
- **`int_measure`** (*`obj`*) – internal measure function.
- **`select`** (*`str`*) – it can be 'min', if the internal measure is to be minimized or 'max' if the internal measure should be macimized.
- **`nclust_range`** (*`list`*) – Range of clusters to consider, default None.

Returns internal score and best number of clusters.

Return type float, int

`reval.internal_baselines.`**`evaluate_best`**(*data*, *c*, *int_measure*, *ncl=None*)

Function that, given a number of clusters, returns the corresponding internal measure for a dataset.

Parameters

- **`data`** (*`array-like`*) – dataset.
- **`c`** (*`obj`*) – clustering algorithm class.
- **`int_measure`** (*`obj`*) – internal measure function.
- **`ncl`** (*`int`*) – number of clusters.

Returns internal score.

Return type float

5.3 Visualization

`reval.visualization` enables plotting the cross-validation performance.

`reval.visualization.`**`plot_metrics`**(*cv_score*, *figsize=(8, 5)*, *linewidth=1*, *color=('black', 'black')*, *legend_loc=2*, *fontsize=12*, *title="*, *prob_lines=False*, *save_fig=None*)

Function that plots the average performance (i.e., normalized stability) over cross-validation for training and validation sets. The horizontal lines represent the random performance error for the correspondent number of clusters.

Parameters

- **`cv_score`** (*`dictionary`*) – collection of cv scores as output by *reval.best_nclust_cv.FindBestCLustCV.best_nclust*.
- **`figsize`** (*`tuple`*) – (width, height), default (8, 5).
- **`linewidth`** (*`int`*) – width of the lines to draw.
- **`color`** (*`tuple`*) – line colors for train and validation sets, default ('black', 'black').
- **`legend_loc`** (*`int`*) – legend location, default 2.
- **`fontsize`** (*`int`*) – size of fonts, default 12.
- **`title`** (*`str`*) – figure title, default "".

- **prob_lines** (*bool*) – plot the normalized stability of random labeling as thresholds, default False.
- **save_fig** (*str*) – file name for saving figure in png format, default None.

CHAPTER 6

Cite as

```
Landi I, Mandelli V, Lombardo MV.
reval: A Python package to determine best clustering solutions with stability-based␣
↪relative clustering validation.
Patterns (N Y). 2021 Apr 2;2(4):100228.
doi: 10.1016/j.patter.2021.100228. PMID: 33982023; PMCID: PMC8085609.
```

BibTeX alternative

```
@article{landi2021100228,
         title = {reval: A Python package to determine best clustering solutions with␣
↪stability-based relative clustering validation},
         journal = {Patterns},
         volume = {2},
         number = {4},
         pages = {100228},
         year = {2021},
         issn = {2666-3899},
         doi = {https://doi.org/10.1016/j.patter.2021.100228},
         url = {https://www.sciencedirect.com/science/article/pii/S2666389921000428},
         author = {Isotta Landi and Veronica Mandelli and Michael V. Lombardo},
         keywords = {stability-based relative validation, clustering, unsupervised␣
↪learning, clustering replicability}
         }
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index