Buddi-CRAVeD Documentation

Release 0.1

R Mukesh

Jul 02, 2018

Contents:

CHAPTER 1

Introduction

1.1 What is Buddi-CRAVeD ?

Buddi-CRAVeD (`ClusteR Analysis and Validation for largE Datasets`) is a easy-to-use Cluster Analysis, Visualisation and Validation suite. It draws upon the power of standard Python3 APIs for data analysis, machine learning and data visualisation.

The suite extends support for an extensive range of :

- Clustering Algorithms
- Internal and External Cluster Validation Indices
- Methods for visualisation of 2D and 3D Data

The suite along with our **modified versions** of the companion libraries (scikit-learn, scipy) allows us to push the limits on **"maximum size of datasets for cluster analysis on common desktops and laptops"** posed due low main memory availabilty. This feat is achieved by moving the memory-intensive data structures that cloge the main memory onto HDF5 files that can be put on hard-disks.

1.2 Who is it intended for ?

Buddi-CRAVeD is a *"quick-starter"* or *"one-stop"* for machine-learning enthusiasts seeking to explore the areas of **Exploratory Data Analysis (EDA)**, **Cluster Analysis** and **Cluster Validation** with ease.

It also qualifies as a great companion tool to assist **quality research** on *Cluster Validation* and allied domains.

For those "craving" to **do more** (i.e., process larger datasets) with the **limited main memory** of their desktops and laptops, we help you push the boundaries.

CHAPTER 2

Setting up the Environment[1]

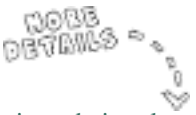
The suite interfaces extensively with some external standard Python3 APIs to accomplish its tasks. It is essential that we get these `dependencies`[2] up and running, before we can get started.

2.1 Installing the Dependencies

Matplotlib (a plotting library for Python) powers the data visulation capabilities of the suite. It is built upon Tkinter (**Tk inter**face for Python). To install tkiner:

```
$ sudo apt-get install python3-tk
```

It is **strongly recommended** to use **`pip`** and **`virtualenv`** for installing python packages specific to projects.



For instructions on **installing and setting up pip and virtualenv**, refer Installing packages using pip and virtualenv.

Note: Ensure that you **activate your virtualenv** (if you have created one) before installing python packages.

The suite is built upon Numpy, the fundamental package for **numerical and scientific computing** in Python. To install **numpy**:

```
(env) $ python3 -m pip install numpy
```

[1] The instructions for setting up the environment are specific to **`Ubuntu`** based operating systems. However, it can replicated for other **Linux** Distros and **Windows** Systems.

[2] The list of dependencies were generated on a python **`virtualenv`** created exclusively for the project and using **`pip`**

```
(env) $ python3 -m pip freeze
```

2.2 Installing Buddi-CRAVeD Suite

2.2.1 Installing from PyPI

The **Buddi-CRAVeD** (alias craved) package is available on PyPI **(the Python Package Index)**. **`Pip`** allows for a **one-step installation** of the package and its dependencies using:

```
(env) $ python3 -m pip install craved
```

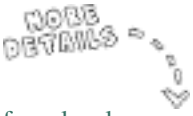
2.2.2 Installing from Source

Alternatively, the **Buddi-CRAVeD** suite (source code) can be also be download from https://github.com/elixir-code/craved.git .

```
$ git clone https://github.com/elixir-code/craved.git
```

`Pip` allows for a **one-step installation** of the package and its dependencies from **local source** using:

```
(env) $ python3 -m pip install <path to source directory>
```



For instructions on installing packages from VCS (Version Control System) or installing packages from local source, refer Python Packaging User Guide.

2.3 Preparing the Data Warehouse

The **Buddi-CRAVeD warehouse** directory functions as an aggregated store for intermediate data structures, sampled datasets and accumulated results.

To configure and setup, an **empty directory** (of your choice) as the suite's data warehouse, execute the "**`craved-warehouse`**" script in the **terminal** from the chosen warehouse directory.

```
$ craved-warehouse
```

Note: The **'craved-warehouse'** script, that configures and sets up the package's warehouse can be invoked on **empty directories only**.

Warning: The **craved** doesn't allow **multiple warehouses** to be configured simaltaneously. Successful reinvokations of the **'craved-warehouse'** script on other directories will force the previous configuration to become invalid.

2.4 Extending Support for Large Datasets

The *Buddi-CRAVeD* suite's enhanced support for **cluster analysis of "larger" datasets** is enabled through our modified versions of the companion libraries – scikit-learn and scipy.

These libraries in part derive their numerical computation capabilities from ATLAS **(Automatically Tuned Linear Algebra Software)**. To install **ATLAS**:

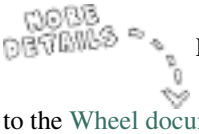
```
$ sudo apt-get install libatlas-base-dev
```

The python **wheel** formats (built for linux systems) of the modified companion libraries can be downloaded from sourceforge (project : **craved-support**) - scikit_learn-0.18.1-cp35-cp35m-linux_x86_64.whl and scipy-0.19.1-cp35-cp35m-linux_x86_64.whl.

`Pip` allows for **easy overwrite and installation** of the **remote** wheels.

```
(env) $ python3 -m pip uninstall scikit-learn
(env) $ python3 -m pip install --use-wheel --no-index --find-links=https://
↪sourceforge.net/projects/craved-support/files/scikit_learn-0.18.1-cp35-cp35m-linux_
↪x86_64.whl scikit-learn
```

```
(env) $ python3 -m pip uninstall scipy
(env) $ python3 -m pip install --use-wheel --no-index --find-links=https://
↪sourceforge.net/projects/craved-support/files/scipy-0.19.1-cp35-cp35m-linux_x86_64.
↪whl scipy
```



For instructions on the usage of pip and wheel utilities for installing **remote and local wheels**, refer to the Wheel documentation.

CHAPTER 3

Quick Starters Guide

Setup the Buddi-CRAVeD environment by following the instructions on the *Setting up the Environment* page.

Use the `generate_samplings.py` and `generate_indices.py` template scripts to perform **random sampling and cluster validation** of the datasets. Uncomment, comment or modify the parameter arguments in the scripts to suit your needs and run them.

The **random samplings** (bagging) and **cluster validation** (internal and external indices) results are dumped into the Buddi-CRAVeD warehouse.

3.1 Random Sampling of Dataset

The `generate_samplings.py` script performs the following tasks:

- Reads the dataset from CSV, LIBSVM or ARFF data files
- Preprocesses (i.e., Dummy codes and Standardizes) the data
- Performs Repeated Random Sampling (with replacement across samplings) of the datasets into bags

The sampled data bags are dumped into the `BAGS` folder in the Buddi-CRAVeD warehouse.

Listing 1: generate_samplings.py

```
""" Script to perform repeated random sampling (with replacement across samplings) of␣
↪datasets into bags
Supported Data Formats: CSV, LIBSVM, ARFF """

from craved import eda
main = eda.eda()

# Path and file name of the data file
filename = '<replace with path and filename of CSV, LIBSVM or ARFF data file>'
```

(continues on next page)

(continued from previous page)

```
# Uncomment the following code segment for CSV data file -- start
"""

''' column (aka. feature) seperator, i.e, sequence of characters (or a regex) that␣
↪seperate two features of a data sample '''

seperator = ',' # default
# seperator = '\s+' # combination of spaces and tabs
# seperator = None # autodetect delimiter (warning:  may not work with certain␣
↪combinations of parameters)
# seperator = '<replace with a character or regular expression>' # examples: '\r\t'


''' number of initial lines <int> or list of line numbers <list> of the data file to␣
↪skip before start of data samples
Note: include the commented and blank lines in the count of intial lines to skip. Don
↪'t skip over header line (with column names) if any '''

skiplines = None # default: no lines to skip (aka. skiplines = 0)
# skiplines = <replace with number of initial lines to skip>
# skiplines = [<replace with list of line numbers to skip>]


''' relative zero-index based line number of header line containing column names to␣
↪use
Note:
        * The indexing starts (from line number=0) for lines immediately following␣
↪the skipped lines.
        * Blank lines are ignored in the indexing.
        * All lines following the skipped lines until the header line are ignored.
'''

headerline = None # default: No header lines (containing column names) to use.
# headerline = 0 # the first non-blank line following skipped lines contains column␣
↪names to use.
# headerline = <replace with relative zero-index of header line>


''' List of columns to use from the data file
Note:
        * Use list of column names (as inferred from header line) or zero-based␣
↪column indices (only if no header line)
        * Include the 'target' value column in the list of columns to use
'''

usecolumns = None # default: Use all columns from data file
# usecolumns = [<replace with list of columns to use>]


''' relative zero-based index (among selected columns) or column name (as inferred␣
↪from header line) of 'target' values column '''

targetcol = -1 # default: last column among list of selected columns
# targetcol = 0 # first column among list of selected columns
```

(continues on next page)

(continued from previous page)

```
# targetcol = '<replace with name of target column as inferred from header line (if␣
↪any)>'


''' Should target values be treated as nominal values and be encoded (with zero-
↪indexed integers) '''

encode_target = True # default: Encode target values
# encode_target = False # Target values represent a non-nominal (or continous)␣
↪feature and should not be encoded


''' List of column names (as inferred from header line) or absolute column indices␣
↪(index of column as in data file) if no header line
of nominal categorical columns to encode.

Note:
        * nominal_cols='infer' infers all string dtype columns with relatively large␣
↪number of unique entries as 'string' or 'date' features and drops them from the␣
↪dataset.
'''

nominal_cols = 'infer' # default: infer all string dtype columns with reasonably few␣
↪unique entries as nominal columns. See Note (1).
# nominal_cols = 'all' # All columns are nominal categorical
# nominal_cols = None # No nominal categorical columns
# nominal_cols = [<list of nominal categorical columns to encode>]


''' List of strings to be inferred as NA (Not Available) or NaN (Not a Number) in␣
↪addition to the default NA string values.
Default NA values :  '', '#N/A', '#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN', '-
↪nan', '1.#IND', '1.#QNAN', 'N/A', 'NA', 'NULL', 'NaN', 'n/a', 'nan', 'null'
'''

na_values = None # default: no additional strings (asides default strings) to infer␣
↪as NA/NaN
# na_values = [<list of additional strings to infer as NA/NaN>]


''' Dictionary of other keyword arguments accepted by :func:`pandas.read_csv`␣
↪(Keyword Arguments: comment, lineterminator, ...) '''
kargs = {}


main.read_data_csv(filename, sep=seperator, skiprows=skiplines, header_row=headerline,
↪ usecols=usecolumns, target_col=targetcol, encode_target=encode_target, categorical_
↪cols=nominal_cols, na_values=na_values, nrows=None, **kargs)
"""
# Uncomment the above code segment for CSV data file -- end
```

(continues on next page)

(continued from previous page)

```
# Uncomment the following code segment for LIBSVM data file -- start
"""
Note: The column indices are one-based (i.e., [1 ... n_features]) irrespective of␣
↪actual indices in the data file

main.read_data_libsvm(filename)
"""
# Uncomment the above code segment for LIBSVM data file -- end









# Uncomment the following code segment for ARFF data file -- start
"""

''' Attribute name of the target column '''

target_attr = 'class' # default: 'class' attribute contains the target values
# target_attr = '<replace with attribute name of target column>'


''' Should target values be encoded (with zero-indexed integers) '''

encode_target = 'infer' # default: encode if target attribute is nominal type and␣
↪doesn't otherwise (i.e., if numeric type)
# encode_target = True
# encode_target = False


''' List of 'names' of numeric attributes to be inferred as nominal and to be encoded.
Note: All nominal attributes are implicitly encoded and should not be included in the␣
↪list. '''

numeric_nominal_attrs = None # default: No numeric attributes are to be inferred as␣
↪nominal
# numeric_nominal_attrs = [<replace with list of numeric type attributes to be␣
↪inferred as nominal and encoded>]


# Note: Data samples with any NA/NaN (i.e., '?') features are implicitly dropped

main.read_data_arff(filename, target_attr=target_attr, encode_target=encode_target,␣
↪num_categorical_attrs=numeric_nominal_attrs)
"""
# Uncomment the above code segment for ARFF data file -- end
```

(continues on next page)

(continued from previous page)

```
""" Perform dummy coding of nominal columns (or features) """

''' List of column names (as inferred from 'header line in CSV' or 'metadata section␣
↪in ARFF') or
column indices (if 'no header line' in CSV) of nominal columns to dummy code '''


nominal_columns = 'infer' # Default: Use list of nominal columns supplied to (or␣
↪inferred by) :func:`read_data_csv` or :func:`read_data_arff`
# nominal_columns = 'all' # All columns are nominal columns and are to be dummy coded
# nominal_columns = None # No nominal columns to dummy code
# nominal_columns = [<list of nominal columns to dummy code>]

main.dummy_coding(nominal_columns=nominal_columns)




main.standardize_data()




""" Perform repeated random sampling (with replacement across samplings) of dataset␣
↪into bags """

# Name of folder that contains sampled bags and metadata files
dataset_name = '<Name of the dataset>'

# No. of samples in each bag (aka. sampling)
sampling_size = 3000
# sampling_size = <replace with an integer denoting number of samples per bag>


# Number of samplings (aka. bagging) to perform
n_iterations = 10
# n_iterations = <replace with an integer denoting number of sampling to perform>

main.random_sampling(dataset_name, sampling_size, n_iterations=n_iterations)
```

3.2 Cluster Validation on Sampled Data

The `generate_indices.py` script performs the following tasks:

- Loads the sampled data bags one after another
- Processeses (i.e, standardizes) the bag data samples
- Estimates Optimal Number of Clusters in data for K-Means and Hierarchial Cluster Analysis (using **GAP STATISTICS**)
- Performs K-Means and Hierarchial Clustering of the data
- Evaluates cluster quality by generating internal and external cluster validation indices.

The cluster validation results are appended to the `results.csv` file in the Buddi-CRAVeD warehouse.

Listing 2: generate_indices.py

```
""" Script to generate indices from randomly sampled bags of data """

from craved import eda
from craved import internal_indices
from craved import external_indices

import pickle

""" Parameters to be specified by the user
Instruction: uncomment and modify the necessary argument lines for the parameters and␣
↪comment other (default) argument lines.
"""

# path to the location where the folder with dataset's (randomly sampled) bag folder␣
↪resides
location = eda.__warehouse__ + 'BAGS/' # default
# location = '<replace with path to the location>'


# name of folder which contains the dataset's (randomly sampled) bag and metadata␣
↪files.
bag_name = '<replace with name of folder which contains datasets bag and metadata␣
↪files>'


'''Parameters for K-Means Cluster Analysis'''

# Method used to determine the number of clusters in data for K-Means Clustering.
# Allowed arguments: {'gap', 'n_classes', <int>, <list of integers of shape (n_bags,)>
↪}
kmeans_n_clusters = 'gap' # Use gap statistics to estimate the optimal number of␣
↪clusters from data (default)
# n_clusters = 'n_classes' # Use the number of classes in data as the number of␣
↪clusters
# n_clusters = <int>
#n_clusters = <list of {int, 'gap', 'n_classes'} of shape (n_bags,)> denoting the␣
↪number of clusters in each bag

kmeans_kargs = {} # Other Keyword arguments (other than 'n_clusters') to␣
↪:func:`sklearn.cluster.KMeans`


'''Parameters for Ward's Agglomerative Cluster Analysis'''

# Method used to determine the number of clusters in data for Hierarchial Clustering.
# Allowed arguments: {'gap', 'n_classes', <int>, <list of integers of shape (n_bags,)>
↪}
hierarchial_n_clusters = 'gap' # Use gap statistics to estimate the optimal number of␣
↪clusters from data (default)
# n_clusters = 'n_classes' # Use the number of classes in data as the number of␣
↪clusters
# n_clusters = <int>
#n_clusters = <list of {int, 'gap', 'n_classes'} of shape (n_bags,)> denoting the␣
↪number of clusters in each bag
```

(continues on next page)

(continued from previous page)

```
hierarchial_kargs = {} # Other Keyword arguments (other than 'n_clusters') to␣
↪:func:`sklearn.cluster.AgglomerativeClustering`


""" Code to perform cluster analysis and Validation
Note: Not to be modified by the user
"""

internal_indices_functions = {
                                    'WGSS':'wgss_index',
                                    'BGSS':'bgss_index',
                                    'Ball-Hall':'ball_hall_index',
                                    'Banfeld-Raftery':'banfeld_raftery_index',
                                    'Det-Ratio':'det_ratio_index',
                                    'Ksq-DetW':'ksq_detw_index',
                                    'Log-Det-Ratio':'log_det_ratio_index',
                                    'Log-SS-Ratio':'log_ss_ratio_index',
                                    'Scott-Symons':'scott_symons_index',
                                    'Trace-WiB':'trace_wib_index',
                                    'Silhouette':'silhouette_score',
                                    'Calinski-Harabasz':'calinski_harabaz_score',
                                    'C':'c_index',
                                    'Dunn':'dunn_index',
                                    'Davies-Bouldin':'davies_bouldin_index',
                                    'Ray-Turi':'ray_turi_index',
                                    'Hartigan':'hartigan_index',
                                    'PBM':'pbm_index',
                                    'Score':'score_function'
                                }

# Note: Strictly don't modify (or reorder) the 'choosen_internal_indices' list␣
↪(without corresponding changes to 'results.csv' in warehouse)
choosen_internal_indices = [
                                    'WGSS', 'BGSS',

                                    'Ball-Hall',
                                    'Banfeld-Raftery',
                                    'Calinski-Harabasz',

                                    'Det-Ratio',
                                    'Ksq-DetW',
                                    'Log-Det-Ratio',
                                    'Log-SS-Ratio',

                                    #'Scott-Symons',
                                    'Silhouette',

                                    'Trace-WiB',
                                    'C',
                                    'Dunn',
                                    'Davies-Bouldin',
                                    'Ray-Turi',
                                    'PBM',
                                    'Score'
                                ]
```

(continues on next page)

(continued from previous page)

```
# TODO*: Check if Sklearn's Precision, Recall, ... which are for classification work␣
↪for clustering as well (change cluster indices and observe change in precision)
external_indices_functions = {
                                'Entropy':'entropy',
                                'Purity':'purity',

                                'Precision':'precision_coefficient',
                                'Recall':'recall_coefficient',
                                'F':'f_measure',
                                'Weighted-F':'weighted_f_measure',

                                'Folkes-Mallows':'folkes_mallows_index',
                                'Rand':'rand_index',
                                'Adjusted-Rand':'adjusted_rand_index',

                                'Adjusted-Mutual-Info':'adjusted_mutual_info',
                                'Normalised-Mutual-Info':'normalized_mutual_info',

                                'Homegeneity':'homogeneity_score',
                                'Completeness':'completness_score',
                                'V-Measure':'v_measure_score',

                                'Jaccard':'jaccard_coeff',
                                'Hubert Γ':'hubert_T_index',
                                'Kulczynski':'kulczynski_index',

                                'McNemar':'mcnemar_index',
                                'Phi':'phi_index',
                                'Russel-Rao':'russel_rao_index',

                                'Rogers-Tanimoto':'rogers_tanimoto_index',
                                'Sokal-Sneath1':'sokal_sneath_index1',
                                'Sokal-Sneath2':'sokal_sneath_index2'
                            }

# Note: Strictly don't modify (or reorder) the 'choosen_external_indices' list␣
↪(without corresponding changes to 'results.csv' in warehouse)
choosen_external_indices = [
                                'Entropy',
                                'Purity',

                                'Precision',
                                'Recall',
                                'F',
                                'Weighted-F',

                                'Folkes-Mallows',
                                'Rand',
                                'Adjusted-Rand',

                                'Adjusted-Mutual-Info',
                                'Normalised-Mutual-Info',

                                'Homegeneity',
                                'Completeness',
                                'V-Measure',
```

(continues on next page)

(continued from previous page)

```
                                    'Jaccard',
                                    'Hubert Γ',
                                    'Kulczynski',

                                    'McNemar',
                                    'Phi',
                                    'Russel-Rao',

                                    'Rogers-Tanimoto',
                                    'Sokal-Sneath1',
                                    'Sokal-Sneath2'
                                ]

# Extract the list of all (randomly sampled) dataset bag files in sorted order
import re, os
bag_files = sorted([file.path for file in os.scandir(location + bag_name) if file.is_
↪file() and re.match(r"[_]?bag[0-9]+.p$", file.name)])
bag_files_names = sorted([file.name for file in os.scandir(location + bag_name) if␣
↪file.is_file() and re.match(r"[_]?bag[0-9]+.p$", file.name)])

# Open and read the bags and dataset metadata info.
metadata_file = location + bag_name + "/metadata.p"
metadata = pickle.load( open(metadata_file, 'rb') )

# Open 'results.csv' file in warehouse and append all results to it.
results_file = open(eda.__warehouse__+"results.csv", 'a')

n_bags = len(bag_files)

# Preprocess the 'kmeans_n_clusters' argument to standard accepted forms
if isinstance(kmeans_n_clusters, str) and kmeans_n_clusters in ['gap', 'n_classes']:
    kmeans_n_clusters = [kmeans_n_clusters] * n_bags

elif isinstance(kmeans_n_clusters, int):
    kmeans_n_clusters = [kmeans_n_clusters] * n_bags

elif isinstance(kmeans_n_clusters, list):
    if len(kmeans_n_clusters)!=n_bags:
        print("error: The length of 'kmeans_n_clusters' list supplied doesn't match␣
↪the count of the number of bags")
        sys.exit(1)

    for bag_n_cluster in kmeans_n_clusters:
        if isinstance(bag_n_cluster, int):
            pass

        elif isinstance(bag_n_cluster, 'str') and bag_n_cluster in ['gap','n_classes
↪']:
            pass

        else:
            print("error: invalid entry %s in list supplied as argument to parameter
↪'kmeans_n_clusters'"%bag_n_cluster.__repr__())
            sys.exit()

else:
    print("error: invalid argument to parameter 'kmeans_n_clusters'. Accepted␣
↪arguments: {'gap', 'n_classes', <int>, <list of {int,'gap','n_classes'} of shape (nb
↪bags,)> }")
```

(continues on next page)

(continued from previous page)

```
# Preprocess the 'hierarchial_n_clusters' argument to standard accepted forms
if isinstance(hierarchial_n_clusters, str) and hierarchial_n_clusters in ['gap', 'n_
↪classes']:
    hierarchial_n_clusters = [hierarchial_n_clusters] * n_bags

elif isinstance(hierarchial_n_clusters, int):
    hierarchial_n_clusters = [hierarchial_n_clusters] * n_bags

elif isinstance(hierarchial_n_clusters, list):
    if len(hierarchial_n_clusters)!=n_bags:
        print("error: The length of 'hierarchial_n_clusters' list supplied doesn't␣
↪match the count of the number of bags")
        sys.exit(1)

    for bag_n_cluster in hierarchial_n_clusters:
        if isinstance(bag_n_cluster, int):
            pass

        elif isinstance(bag_n_cluster, 'str') and bag_n_cluster in ['gap','n_classes
↪']:
            pass

        else:
            print("error: invalid entry %s in list supplied as argument to parameter
↪'hierarchial_n_clusters'"%bag_n_cluster.__repr__())
            sys.exit()

else:
    print("error: invalid argument to parameter 'hierarchial_n_clusters'. Accepted␣
↪arguments: {'gap', 'n_classes', <int>, <list of {int,'gap','n_classes'} of shape (n_
↪bags,)> }")


for bag_index, bag_file in enumerate(bag_files):

    print("Processing '{file_name}' ({bag_index}/{n_bags})".format(file_name=bag_
↪files_names[bag_index], bag_index=bag_index+1, n_bags=n_bags))

    dataset = pickle.load( open(bag_file, 'rb') )
    data, target = dataset['data'], dataset['target']

    # Load the data into an eda object :obj:`main`
    main = eda.eda()
    main.load_data(data, target)

    # perform kmeans clustering on the data
    main.perform_kmeans(n_clusters=kmeans_n_clusters[bag_index], **kmeans_kargs)

    # Compute the internal indices for K-Means Clustering Results
    kmeans_internal_validation = internal_indices.internal_indices(main.data, main.
↪kmeans_results['labels'])

    kmeans_internal_indices = []

    for index in choosen_internal_indices:
```

(continues on next page)

(continued from previous page)

```
        index_function = getattr(kmeans_internal_validation, internal_indices_
↪functions[index])
        print("Computing Internal Indices (KMeans Clustering): %s Index"%index)
        kmeans_internal_indices.append(index_function())

    # Compute the external indices for K-Means Clustering Results
    kmeans_external_validation = external_indices.external_indices(main.target, main.
↪kmeans_results['labels'])

    kmeans_external_indices = []

    for index in choosen_external_indices:
        index_function = getattr(kmeans_external_validation, external_indices_
↪functions[index])
        print("Computing External Indices (KMeans Clustering): %s Index"%index)
        kmeans_external_indices.append(index_function())


    # Print the KMeans Clustering results to a 'results.csv' in warehouse
    results_file.write("\"{dataset}\",{timestamp_id},{n_samples},{n_features},{n_
↪classes},\"{class_distrn}\",{nominal_features},".format(dataset=bag_name, timestamp_
↪id=metadata['timestamp'], n_samples=metadata['n_samples'], n_features=metadata['n_
↪features'], n_classes=len(metadata['classes']), class_distrn=metadata['classes'].__
↪repr__(), nominal_features="No" if all(value is None for value in metadata['column_
↪categories'].values()) else "Yes"))
    results_file.write("\"{file_name}\",{sample_size},\"{algorithm}\",\"{parameters}\
↪",{n_clusters},\"{cluster_distrn}\",".format(file_name=bag_files_names[bag_index],␣
↪sample_size=main.n_samples, algorithm='KMeans', parameters=main.kmeans_results[
↪'parameters'].__repr__(), n_clusters=main.kmeans_results['n_clusters'], cluster_
↪distrn=main.kmeans_results['clusters'].__repr__()))

    for index in kmeans_internal_indices:
        results_file.write("{},".format(index))

    for index in kmeans_external_indices:
        results_file.write("{},".format(index))

    results_file.write('\n')

    # perform Ward's Agglomerative clustering on the data
    main.perform_hierarchial(n_clusters=hierarchial_n_clusters[bag_index],␣
↪**hierarchial_kargs)

    # Compute the internal indices for K-Means Clustering Results
    hierarchial_internal_validation = internal_indices.internal_indices(main.data,␣
↪main.hierarchial_results['labels'])

    hierarchial_internal_indices = []

    for index in choosen_internal_indices:
        index_function = getattr(hierarchial_internal_validation, internal_indices_
↪functions[index])
        print("Computing Internal Indices (Hierarchial Clustering): %s Index"%index)
        hierarchial_internal_indices.append(index_function())

    # Compute the external indices for Hierarchial Clustering Results
    hierarchial_external_validation = external_indices.external_indices(main.target,␣
↪main.hierarchial_results['labels'])
```

(continues on next page)

(continued from previous page)

```
    hierarchial_external_indices = []

    for index in choosen_external_indices:
        index_function = getattr(hierarchial_external_validation, external_indices_
↪functions[index])
        print("Computing External Indices (Hierarchial Clustering): %s Index"%index)
        hierarchial_external_indices.append(index_function())

    # Print the Hierarchial Clustering results to a 'results.csv' in warehouse
    results_file.write("\"{dataset}\",{timestamp_id},{n_samples},{n_features},{n_
↪classes},\"{class_distrn}\",{nominal_features},".format(dataset=bag_name, timestamp_
↪id=metadata['timestamp'], n_samples=metadata['n_samples'], n_features=metadata['n_
↪features'], n_classes=len(metadata['classes']), class_distrn=metadata['classes'].__
↪repr__(), nominal_features="No" if all(value is None for value in metadata['column_
↪categories'].values()) else "Yes"))
    results_file.write("\"{file_name}\",{sample_size},\"{algorithm}\",\"{parameters}\
↪",{n_clusters},\"{cluster_distrn}\",".format(file_name=bag_files_names[bag_index],␣
↪sample_size=main.n_samples, algorithm='Hierarchial', parameters=main.hierarchial_
↪results['parameters'].__repr__(), n_clusters=main.hierarchial_results['n_clusters'],
↪ cluster_distrn=main.hierarchial_results['clusters'].__repr__()))

    for index in hierarchial_internal_indices:
        results_file.write("{},".format(index))

    for index in hierarchial_external_indices:
        results_file.write("{},".format(index))

    results_file.write('\n')
    del main

    print('\n\n')

results_file.close()
```

CHAPTER 4

API Documentation

CHAPTER 5

Examples and Experiments

CHAPTER 6

Contributor's Guide

CHAPTER 7

Indices and tables

- genindex
- modindex
- search