QuadratiK

Release 1.0.0

Giovanni Saraceno, Marianthi Markatou,
Raktim Mukhopadhyay, Mojgan Golzy

Mar 08, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

The QuadratiK package is implemented in both **R** and **Python**, providing a comprehensive set of goodness-of-fit tests and a clustering technique using kernel-based quadratic distances. This framework aims to bridge the gap between the statistical and machine learning literatures. It includes:

- **Goodness-of-Fit Tests** : The software implements one, two, and k-sample tests for goodness of fit, offering an efficient and mathematically sound way to assess the fit of probability distributions. Expanded capabilities include supporting tests for uniformity on the d-dimensional Sphere based on Poisson kernel densities.
- **Clustering Algorithm for Spherical Data**: the package incorporates a unique clustering algorithm specifically tailored for spherical data. This algorithm leverages a mixture of Poisson-kernel-based densities on the sphere, enabling effective clustering of spherical data or data that has been spherically transformed. This facilitates the uncovering of underlying patterns and relationships in the data.
- **Additional Features**: Alongside these functionalities, the software includes additional graphical functions, aiding users in validating cluster results as well as visualizing and representing clustering results. This enhances the interpretability and usability of the analysis.

1.1 Funding Information

The work has been supported by Kaleida Health Foundation, Food and Drug Administration, and Department of Biostatistics, University at Buffalo.

1.2 Authors

Giovanni Saraceno <gsaracen@buffalo.edu>, Marianthi Markatou <markatou@buffalo.edu>, Raktim Mukhopadhyay <raktimmu@buffalo.edu>, Mojgan Golzy <golzym@health.missouri.edu>

Mantainer: Raktim Mukhopadhyay <raktimmu@buffalo.edu>

1.3 References

Saraceno G., Markatou M., Mukhopadhyay R., Golzy M. (2023). Goodness of- fit and clustering of spherical data: The QuadratiK package in R and Python. Technical Report, Department of Biostatistics, University at Buffalo.

Ding Y., Markatou M., Saraceno G. (2023). "Poisson Kernel-Based Tests for Uniformity on the d-Dimensional Sphere." Statistica Sinica. DOI: 10.5705/ss.202022.0347.

Golzy M. & Markatou M. (2020) Poisson Kernel-Based Clustering on the Sphere: Convergence Properties, Identifiability, and a Method of Sampling, Journal of Computational and Graphical Statistics, 29:4, 758-770, DOI: 10.1080/10618600.2020.1740713.

Markatou M, Saraceno G, Chen Y (2023). "Two- and k-Sample Tests Based on Quadratic Distances." Manuscript, (Department of Biostatistics, University at Buffalo).

1.3.1 Getting Started

1.3.1.1 Installation

Which Python?

You'll need **Python 3.9 (except 3.9.7) or greater**.

Install using `pip`

```
pip install QuadratiK
```

Dependencies

QuadratiK requires the following (arranged alphabetically):

- matplotlib (>=3.8.2)
- nest-asyncio (>=1.5)
- numpy (>= 1.26.2)
- pandas (>= 2.1.3)
- plotly (>=5.15.0)
- scikit-learn (>= 1.3)
- scipy (>= 1.11)
- streamlit (>=1.30.0)
- tabulate (>= 0.8)

Testing

QuadratiK uses the Python `pytest` package. To install `pytest`, please go here. To run the tests using `pytest`, please follow these instructions. Navigate to the tests folder to run the tests.

1.3.2 API Reference

1.3.2.1 Kernel Test

KernelTest([h, method, num_iter, b, ...])	Class for performing the kernel-based quadratic distance goodness-of-fit tests using the Gaussian kernel with tuning parameter h.
select_h(x[, y, alternative, method, b, ...])	This function computes the kernel bandwidth of the Gaussian kernel for the one sample, two-sample and k-sample kernel-based quadratic distance (KBQD) tests.

KernelTest

class QuadratiK.kernel_test.**KernelTest**(*h=None*, *method='subsampling'*, *num_iter=150*, *b=0.9*, *quantile=0.95*, *mu_hat=None*, *sigma_hat=None*, *centering_type='nonparam'*, *alternative=None*, *k_threshold=10*, *random_state=None*, *n_jobs=8*)

Class for performing the kernel-based quadratic distance goodness-of-fit tests using the Gaussian kernel with tuning parameter h. Depending on the input *y* the function performs the test of multivariate normality, the non-parametric two-sample tests or the k-sample tests.

Parameters

h
[float, optional] Bandwidth for the kernel function.

method
[str, optional] The method used for critical value estimation ("subsampling", "bootstrap", or "permutation").

num_iter
[int, optional] The number of iterations to use for critical value estimation. Defaults to 150.

b
[float, optional] The size of the subsamples used in the subsampling algorithm. Defaults to 0.9.

quantile
[float, optional] The quantile to use for critical value estimation. Defaults to 0.95.

mu_hat
[numpy.ndarray, optional] Mean vector for the reference distribution. Defaults to None.

sigma_hat
[numpy.ndarray, optional] Covariance matrix of the reference distribution. Defaults to None.

alternative
[str, optional] String indicating the type of alternative to be used for calculating "h" by the tuning parameter selection algorithm when h is not provided. Defaults to 'None'

k_threshold
[int, optional] Maximum number of groups allowed. Defaults to 10. Change in case of more than 10 groups.

random_state
[int, None, optional.] Seed for random number generation. Defaults to None

n_jobs
[int, optional.] n_jobs specifies the maximum number of concurrently running workers. If 1 is given, no joblib parallelism is used at all, which is useful for debugging. For more information on joblib n_jobs refer to - https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html. Defaults to 8.

Attributes

test_type_
[str] The type of test performed on the data

execution_time
[float] Time taken for the test method to execute

h0_rejected_
[boolean] Whether the null hypothesis is rejected (True) or not (False)

test_statistic_
[float] Test statistic of the perfomed test type

cv_
[float] Critical value

cv_method_
[str] Critical value method used for performing the test

References

Markatou M., Saraceno G., Chen Y (2023). "Two- and k-Sample Tests Based on Quadratic Distances. "Manuscript, (Department of Biostatistics, University at Buffalo)

Lindsay BG, Markatou M. & Ray S. (2014) Kernels, Degrees of Freedom, and Power Properties of Quadratic Distance Goodness-of-Fit Tests, Journal of the American Statistical Association, 109:505, 395-410, DOI: 10.1080/01621459.2013.836972

Examples

```
>>> # Example for normality test
>>> import numpy as np
>>> from QuadratiK.kernel_test import KernelTest
>>> np.random.seed(42)
>>> data = np.random.randn(100,5)
>>> normality_test = KernelTest(h=0.4, centering_type="param",random_state=42).
↪test(data)
>>> print("Test : {}".format(normality_test.test_type_))
>>> print("Execution time: {:.3f}".format(normality_test.execution_time))
>>> print("H0 is Rejected : {}".format(normality_test.h0_rejected_))
>>> print("Test Statistic : {}".format(normality_test.test_statistic_))
>>> print("Critical Value (CV) : {}".format(normality_test.cv_))
>>> print("CV Method : {}".format(normality_test.cv_method_))
```

(continues on next page)

(continued from previous page)

```
>>> print("Selected tuning parameter : {}".format(normality_test.h))
... Test : Kernel-based quadratic distance Normality test
... Execution time: 0.096
... H0 is Rejected : False
... Test Statistic : -8.588873037044384e-05
... Critical Value (CV) : 0.0004464111809800183
... CV Method : Empirical
... Selected tuning parameter : 0.4
```

```
>>> # Example for two sample test
>>> import numpy as np
>>> from QuadratiK.kernel_test import KernelTest
>>> np.random.seed(42)
>>> X = np.random.randn(100,5)
>>> np.random.seed(42)
>>> Y = np.random.randn(100,5)
>>> two_sample_test = KernelTest(h=0.4, centering_type="param").test(X,Y)
>>> print("Test : {}".format(two_sample_test.test_type_))
>>> print("Execution time: {:.3f}".format(two_sample_test.execution_time))
>>> print("H0 is Rejected : {}".format(two_sample_test.h0_rejected_))
>>> print("Test Statistic : {}".format(two_sample_test.test_statistic_))
>>> print("Critical Value (CV) : {}".format(two_sample_test.cv_))
>>> print("CV Method : {}".format(two_sample_test.cv_method_))
>>> print("Selected tuning parameter : {}".format(two_sample_test.h))
... Test : Kernel-based quadratic distance two-sample test
... Execution time: 0.092
... H0 is Rejected : False
... Test Statistic : -0.019707895277270022
... Critical Value (CV) : 0.003842482597612725
... CV Method : subsampling
... Selected tuning parameter : 0.4
```

Methods

KernelTest.stats()	Function to generate descriptive statistics per variable (and per group if available).
KernelTest.summary([print_fmt])	Summary function generates a table for the kernel test results and the summary statistics.
KernelTest.test(x[, y])	Function to perform the kernel-based quadratic distance tests using the Gaussian kernel with bandwidth parameter h.

KernelTest.**stats**()

Function to generate descriptive statistics per variable (and per group if available).

Returns

summary_stats_df
[pandas.DataFrame] Dataframe of descriptive statistics

`KernelTest.`**`summary`**`(`*print_fmt='simple_grid'*`)`

Summary function generates a table for the kernel test results and the summary statistics.

Parameters

print_fmt
[str, optional.] Used for printing the output in the desired format. Defaults to "simple_grid". Supports all available options in tabulate, see here: https://pypi.org/project/tabulate/

Returns

summary
[str] A string formatted in the desired output format with the kernel test results and summary statistics.

`KernelTest.`**`test`**`(`*x*, *y=None*`)`

Function to perform the kernel-based quadratic distance tests using the Gaussian kernel with bandwidth parameter h. Depending on the shape of the *y*, the function performs the tests of multivariate normality, the non-parametric two-sample tests or the k-sample tests.

Parameters

x
[numpy.ndarray or pandas.DataFrame.] A numeric array of data values.

y
[numpy.ndarray or pandas.DataFrame, optional] A numeric array data values (for two-sample test) and a 1D array of class labels (for k-sample test). Defaults to None.

Returns

self
[object] Fitted estimator

select_h

`QuadratiK.kernel_test.`**`select_h`**`(`*x*, *y=None*, *alternative='location'*, *method='subsampling'*, *b=0.8*, *num_iter=150*, *delta_dim=1*, *delta=None*, *h_values=None*, *n_rep=50*, *n_jobs=8*, *quantile=0.95*, *k_threshold=10*, *power_plot=False*, *random_state=None*`)`

This function computes the kernel bandwidth of the Gaussian kernel for the one sample, two-sample and k-sample kernel-based quadratic distance (KBQD) tests.

The function performs the selection of the optimal value for the tuning parameter h of the normal kernel function, for the two-sample and k-sample KBQD tests. It performs a small simulation study, generating samples according to the family of alternative specified, for the chosen values of h_values and delta.

Parameters

x
[numpy.ndarray or pandas.DataFrame] Data set of observations from X

y
[numpy.ndarray or pandas.DataFrame, optional] Data set of observations from Y for two sample test or set of labels in case of k-sample test

alternative
[str, optional] Family of alternative chosen for selecting h, must be one of "location", "scale" and "skewness". Defaults to "location"

method
[str, optional.] The method used for critical value estimation, must be one of "subsampling", "bootstrap", or "permutation". Defaults to "subsampling".

b
[float, optional.] The size of the subsamples used in the subsampling algorithm. Defaults to 0.8.

num_iter
[int, optional.] The number of iterations to use for critical value estimation. Defaults to 150.

delta_dim
[int, numpy.ndarray, optional.] Array of coefficient of alternative with respect to each dimension. Defaults to 1.

delta
[numpy.ndarray, optional.] Array of parameter values indicating chosen alternatives. Defaults to None.

h_values
[numpy.ndarray, optional.] Values of the tuning parameter used for the selection. Defaults to None.

n_rep
[int, optional. Defaults to 50.] Number of bootstrap replications

n_jobs
[int, optional.] n_jobs specifies the maximum number of concurrently running workers. If 1 is given, no joblib parallelism is used at all, which is useful for debugging. For more information on joblib n_jobs refer to - https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html. Defaults to 8.

quantile
[float, optional.] Quantile to use for critical value estimation. Defaults to 0.95.

k_threshold
[int.] Maximum number of groups allowed. Defaults to 10.

power_plot
[boolean, optional.] If True, plot is displayed the plot of power for values in h_values and delta. Defaults to False.

random_state
[int, None, optional.] Seed for random number generation. Defaults to None

Returns

h
[float] The selected value of tuning parameter h

h vs Power table
[pandas.DataFrame] A table containing the h, delta and corresponding powers

References

Markatou M., Saraceno G., Chen Y. (2023). "Two- and k-Sample Tests Based on Quadratic Distances. "Manuscript, (Department of Biostatistics, University at Buffalo)

Examples

```
>>> import numpy as np
>>> from QuadratiK.kernel_test import select_h
>>> np.random.seed(42)
>>> X = np.random.randn(200, 2)
>>> np.random.seed(42)
>>> y = np.random.randint(0, 2, 200)
>>> h_selected, all_values, power_plot = select_h(
...     X, y, alternative='location', power_plot=True, random_state=42)
>>> print("Selected h is: ", h_selected)
... Selected h is:  1.2
```

1.3.2.2 Poisson Kernel Test

PoissonKernelTest(rho[, num_iter, quantile, ...])	Class for Poisson kernel-based quadratic distance test of Uniformity on the Sphere

PoissonKernelTest

class QuadratiK.poisson_kernel_test.**PoissonKernelTest**(*rho*, *num_iter=300*, *quantile=0.95*, *random_state=None*, *n_jobs=8*)

Class for Poisson kernel-based quadratic distance test of Uniformity on the Sphere

Parameters

rho
[float] The value of concentration parameter used for the Poisson kernel function.

num_iter
[int, optional] Number of iterations for critical value estimation of U-statistic.

quantile
[float, optional] The quantile to use for critical value estimation

random_state
[int, None, optional.] Seed for random number generation. Defaults to None

n_jobs
[int, optional.] n_jobs specifies the maximum number of concurrently running workers. If 1 is given, no joblib parallelism is used at all, which is useful for debugging. For more information on joblib n_jobs refer to - https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html. Defaults to 8.

Attributes

test_type_
[str] The type of test performed on the data

execution_time
[float] Time taken for the test method to execute

u_statistic_h0_
[boolean] A logical value indicating whether or not the null hypothesis is rejected according to Un

u_statistic_un_
[float] The value of the U-statistic.

u_statistic_cv_
[float] The empirical critical value for Un

v_statistic_h0_
[boolean] A logical value indicating whether or not the null hypothesis is rejected according to Vn.

v_statistic_vn_
[float] The value of the V-statistic.

v_statistic_cv_
[float] The critical value for Vn computed following the asymptotic distribution.

References

Ding Y., Markatou M., Saraceno G. (2023). "Poisson Kernel-Based Tests for Uniformity on the d-Dimensional Sphere." Statistica Sinica. doi: doi:10.5705/ss.202022.0347

Examples

```
>>> from QuadratiK.tools import sample_hypersphere
>>> from QuadratiK.poisson_kernel_test import PoissonKernelTest
>>> np.random.seed(42)
>>> X = sample_hypersphere(100,3, random_state=42)
>>> unif_test = PoissonKernelTest(rho = 0.7, random_state=42).test(X)
>>> print("Execution time: {:.3f} seconds".format(unif_test.execution_time))
>>> print("U Statistic Results")
>>> print("H0 is rejected : {}".format(unif_test.u_statistic_h0_))
>>> print("Un Statistic : {}".format(unif_test.u_statistic_un_))
>>> print("Critical Value : {}".format(unif_test.u_statistic_cv_))
```

(continues on next page)

(continued from previous page)

```
>>> print("V Statistic Results")
>>> print("H0 is rejected : {}".format(unif_test.v_statistic_h0_))
>>> print("Vn Statistic : {}".format(unif_test.v_statistic_vn_))
>>> print("Critical Value : {}".format(unif_test.v_statistic_cv_))
... Execution time: 0.181 seconds
... U Statistic Results
... H0 is rejected : False
... Un Statistic : 1.6156682048968174
... Critical Value : 0.06155875299050079
... V Statistic Results
... H0 is rejected : False
... Vn Statistic : 22.83255917641962
... Critical Value : 23.229486935225513
```

Methods

PoissonKernelTest.stats()	Function to generate descriptive statistics.
PoissonKernelTest.summary([print_fmt])	Summary function generates a table for the poisson kernel test results and the summary statistics.
PoissonKernelTest.test(x)	Performs the Poisson kernel-based quadratic distance Goodness-of-fit tests for Uniformity for spherical data using the Poisson kernel with concentration parameter rho

---

PoissonKernelTest.**stats**()

Function to generate descriptive statistics.

Returns

summary_stats_df
[pandas.DataFrame] Dataframe of descriptive statistics

PoissonKernelTest.**summary**(*print_fmt='simple_grid'*)

Summary function generates a table for the poisson kernel test results and the summary statistics.

Parameters

print_fmt
[str, optional.] Used for printing the output in the desired format. Supports all available options in tabulate, see here: https://pypi.org/project/tabulate/. Defaults to "simple_grid".

Returns

summary
[str] A string formatted in the desired output format with the kernel test results and summary statistics.

`PoissonKernelTest.test(x)`

Performs the Poisson kernel-based quadratic distance Goodness-of-fit tests for Uniformity for spherical data using the Poisson kernel with concentration parameter rho

Parameters

x
[numpy.ndarray, pandas.DataFrame] a numeric d-dim matrix of data points on the Sphere $S^{(d-1)}$.

Returns

self
[object] Fitted estimator

1.3.2.3 Spherical Clustering

PKBC(num_clust[, max_iter, stopping_rule, ...])	Poisson kernel-based clustering on the sphere.
PKBD()	Class for estimating density and generating samples of Poisson-kernel based distribution (PKBD).

PKBC

class `QuadratiK.spherical_clustering.PKBC`(*num_clust*, *max_iter=300*, *stopping_rule='loglik'*, *init_method='sampledata'*, *num_init=10*, *tol=1e-07*, *random_state=None*, *n_jobs=4*)

Poisson kernel-based clustering on the sphere. The class performs the Poisson kernel-based clustering algorithm on the sphere based on the Poisson kernel-based densities. It estimates the parameter of a mixture of Poisson kernel-based densities. The obtained estimates are used for assigning final memberships, identifying the data points.

Parameters

num_clust
[int] Number of clusters.

max_iter
[int] Maximum number of iterations before a run is terminated.

stopping_rule
[str, optional] String describing the stopping rule to be used within each run. Currently must be either 'max', 'membership', or 'loglik'.

init_method
[str, optional] String describing the initialization method to be used. Currently must be 'sampleData'.

num_init
[int, optional] Number of initializations.

tol
[float.] Constant defining threshold by which log likelihood must change to continue iterations, if applicable. Defaults to 1e-7.

random_state
[int, None, optional.] Seed for random number generation. Defaults to None

n_jobs
[int] Used only for computing the WCSS efficiently. n_jobs specifies the maximum number of concurrently running workers. If 1 is given, no joblib parallelism is used at all, which is useful for debugging. For more information on joblib n_jobs refer to - https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html. Defaults to 4.

Attributes

alpha_
[numpy.ndarray of shape (n_clusters,)] Estimated mixing proportions

labels_
[numpy.ndarray of shape (n_samples,)] Final cluster membership assigned by the algorithm to each observation

log_lik_vec
[numpy.ndarray of shape (num_init,)] Array of log-likelihood values for each initialization

loklik_
[float] Maximum value of the log-likelihood function

mu_
[numpy.ndarray of shape (n_clusters, n_features)] Estimated centroids

num_iter_per_run
[numpy.ndarray of shape (num_init,)] Number of E-M iterations per run

post_probs_
[numpy.ndarray of shape (n_samples, n_features)] Posterior probabilities of each observation for the indicated clusters

rho_
[numpy.ndarray of shape (n_clusters,)] Estimated concentration parameters rho

euclidean_wcss_
[float] Values of within-cluster sum of squares computed with Euclidean distance.

cosine_wcss_
[float] Values of within-cluster sum of squares computed with cosine similarity.

References

Golzy M. & Markatou M. (2020) Poisson Kernel-Based Clustering on the Sphere: Convergence Properties, Identifiability, and a Method of Sampling, Journal of Computational and Graphical Statistics, 29:4, 758-770, DOI: 10.1080/10618600.2020.1740713.

Examples

```
>>> from QuadratiK.datasets import load_wireless_data
>>> from QuadratiK.spherical_clustering import PKBC
>>> from sklearn.preprocessing import LabelEncoder
>>> X, y = load_wireless_data(return_X_y=True)
>>> le = LabelEncoder()
>>> le.fit(y)
>>> y = le.transform(y)
>>> cluster_fit = PKBC(num_clust=4, random_state=42).fit(X)
>>> ari, macro_precision, macro_recall, avg_silhouette_Score = cluster_fit.
↪validation(y)
>>> print("Estimated mixing proportions :", cluster_fit.alpha_)
>>> print("Estimated concentration parameters: ", cluster_fit.rho_)
>>> print("Adjusted Rand Index:", ari)
>>> print("Macro Precision:", macro_precision)
>>> print("Macro Recall:", macro_recall)
>>> print("Average Silhouette Score:", avg_silhouette_Score)
... Estimated mixing proportions : [0.23590339 0.24977919 0.25777522 0.25654219]
... Estimated concentration parameters:  [0.97773265 0.98348976 0.98226901 0.
↪98572597]
... Adjusted Rand Index: 0.9403086353805835
... Macro Precision: 0.9771870612442508
... Macro Recall: 0.9769999999999999
... Average Silhouette Score: 0.3803089203572107
```

Methods

PKBC.fit(dat)	Performs Poisson Kernel-based Clustering.
PKBC.stats()	Function to generate descriptive statistics per variable (and per group if available).
PKBC.validation([y_true])	Computes validation metrics such as ARI, Macro Precision and Macro Recall when true labels are provided.

PKBC.**fit**(*dat*)

Performs Poisson Kernel-based Clustering.

Parameters

dat
[numpy.ndarray, pandas.DataFrame] A numeric array of data values.

Returns

self
[object] Fitted estimator

`PKBC.stats()`

Function to generate descriptive statistics per variable (and per group if available).

Returns

summary_stats_df
[pandas.DataFrame] Dataframe of descriptive statistics

`PKBC.validation(y_true=None)`

Computes validation metrics such as ARI, Macro Precision and Macro Recall when true labels are provided.

Parameters

y_true
[numpy.ndarray.] Array of true memberships to clusters, Defaults to None.

Returns

validation metrics
[tuple] The tuple consists of the following:

- **Adjusted Rand Index**
  [float (returned only when y_true is provided)] Adjusted Rand Index computed between the true and predicted cluster memberships.
- **Macro Precision**
  [float (returned only when y_true is provided)] Macro Precision computed between the true and predicted cluster memberships.
- **Macro Recall**
  [float (returned only when y_true is provided)] Macro Recall computed between the true and predicted cluster memberships.
- **Average Silhouette Score**
  [float] Mean Silhouette Coefficient of all samples.

References

Rousseeuw, P.J. (1987) Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics, 20, 53–65.

Notes

We have taken a naive approach to map the predicted cluster labels to the true class labels (if provided). This might not work in cases where *num_clust* is large. Please use *sklearn.metrics* for computing metrics in such cases, and provide the correctly matched labels.

See also

sklearn.metrics : Scikit-learn metrics functionality support a wide range of metrics.

PKBD

`class QuadratiK.spherical_clustering.PKBD`

Class for estimating density and generating samples of Poisson-kernel based distribution (PKBD).

Methods

`PKBD.dpkb`(x, mu, rho[, logdens])	Function for estimating the density function of the PKB distribution.
`PKBD.rpkb`(n, mu, rho[, method, random_state])	Function for generating a random sample from PKBD.

`PKBD.dpkb`(*x*, *mu*, *rho*, *logdens=False*)

Function for estimating the density function of the PKB distribution.

Parameters

x
[numpy.ndarray, pandas.DataFrame] A matrix with a number of columns >= 2.

mu
[float] Location parameter with the same length as the rows of x. Normalized to length one.

rho
[float] Concentration parameter. $\rho \in (0, 1]$.

logdens
[bool, optional] If True, densities d are given as $\log(d)$. Defaults to False.

Returns

density
[numpy.ndarray] An array with the evaluated density values.

PKBD.**rpkb**(*n*, *mu*, *rho*, *method='rejvmf'*, *random_state=None*)

Function for generating a random sample from PKBD. The number of observation generated is determined by *n*.

Parameters

n
[int] Sample size.

mu
[float] Location parameter with the same length as the quantiles.

rho
[float] Concentration parameter. $\rho \in (0, 1]$.

method
[str, optional] String that indicates the method used for sampling observations. The available methods are :

- **'rejvmf': acceptance-rejection algorithm using von Mises-Fisher envelops.**
  (Algorithm in Table 2 of Golzy and Markatou 2020);
- **'rejacg': using angular central Gaussian envelops.**
  (Algorithm in Table 1 of Sablica et al. 2023);

Defaults to 'rejvmf'.

random_state
[int, None, optional.] Seed for random number generation. Defaults to None

Returns

samples
[numpy.ndarray] Generated observations from a poisson kernel-based density. This function returns a list with the matrix of generated observations, the number of tries and the number of acceptance.

References

Golzy M. & Markatou M. (2020) Poisson Kernel-Based Clustering on the Sphere: Convergence Properties, Identifiability, and a Method of Sampling, Journal of Computational and Graphical Statistics, 29:4, 758-770, DOI: 10.1080/10618600.2020.1740713.

Sablica L., Hornik K., Leydold J. "Efficient sampling from the PKBD distribution," Electronic Journal of Statistics, 17(2), 2180-2209, (2023)

Examples

```
>>> from QuadratiK.spherical_clustering import PKBD
>>> pkbd_data = PKBD().rpkb(10,[0.5,0],0.5, "rejvmf", random_state= 42)
>>> dens_val  = PKBD().dpkb(pkbd_data, [0.5,0.5],0.5)
>>> print(dens_val)
... [0.46827108 0.05479605 0.21163936 0.06195099 0.39567698 0.40473724
...     0.26561508 0.36791766 0.09324676 0.46847274]
```

1.3.2.4 User Interface

UI()	The UI class is a user interface class that runs a Streamlit dashboard using asyncio.

UI

class QuadratiK.ui.**UI**

The UI class is a user interface class that runs a Streamlit dashboard using asyncio.

Examples

```
>>> from QuadratiK.ui import UI
>>> UI().run()
```

Methods

UI.main()	The *main* function runs a Streamlit dashboard by executing a command-line command.
UI.run()	The function runs the main function asynchronously using the asyncio library in Python.

---

async UI.**main**()

The *main* function runs a Streamlit dashboard by executing a command-line command.

UI.**run**()

The function runs the main function asynchronously using the asyncio library in Python.

1.3.2.5 Datasets

load_wireless_data([desc, return_X_y, ...])	The wireless data frame has 2000 rows and 8 columns.

load_wireless_data

QuadratiK.datasets.**load_wireless_data**(*desc=False*, *return_X_y=False*, *as_dataframe=True*, *scaled=False*)

The wireless data frame has 2000 rows and 8 columns. The first 7 variables report the measurements of the Wi-Fi signal strength received from 7 Wi-Fi routers in an office location in Pittsburgh (USA). The last column indicates the class labels.

The function load_wireless_data loads a wireless localization dataset.

Read more in the *User Guide*.

Parameters

desc
[boolean, optional] If set to *True*, the function will return the description along with the data. If set to *False*, the description will not be included. Defaults to False.

return_X_y
[boolean, optional] Determines whether the function should return the data as separate arrays (*X* and *y*). Defaults to False.

as_dataframe
[boolean, optional] Determines whether the function should return the data as a pandas DataFrame (Trues) or as a numpy array (False). Defaults to True.

scaled
[boolean, optional] Determines whether or not the data should be scaled. If set to True, the data will be divided by its Euclidean norm along each row. Defaults to False.

Returns

(data, target)
[tuple, if return_X_y is True] A tuple of two ndarray. The first containing a 2D array of shape (n_samples, n_features) with each row representing one sample and each column representing the features. The second ndarray of shape (n_samples,) containing the target samples.

data
[pandas.DataFrame, if as_dataframe is True] Dataframe of the data with shape (n_samples, n_features + class)

(desc, data, target)
[tuple, if desc is True and return_X_y is True] A tuple of description and two numpy.ndarray. The first containing a 2D array of shape (n_samples, n_features) with each row representing one sample and each column representing the features. The second ndarray of shape (n_samples,) containing the target samples.

(desc, data)
[tuple, if desc is True and as_dataframe is True] A tuple of description and pandas.DataFrame. Dataframe of the data with shape (n_samples, n_features + class)

References

Rohra, J.G., Perumal, B., Narayanan, S.J., Thakur, P., Bhatt, R.B. (2017). User Localization in an Indoor Environment Using Fuzzy Hybrid of Particle Swarm Optimization & Gravitational Search Algorithm with Neural Networks. In: Deep, K., et al. Proceedings of Sixth International Conference on Soft Computing for Problem Solving. Advances in Intelligent Systems and Computing, vol 546. Springer, Singapore. https://doi.org/10.1007/978-981-10-3322-3_27

Source

Bhatt,Rajen. (2017). Wireless Indoor Localization. UCI Machine Learning Repository. https://doi.org/10.24432/C51880.

Examples

```
>>> from QuadratiK.datasets import load_wireless_data
>>> X, y = load_wireless_data(return_X_y=True)
```

1.3.2.6 Tools

sample_hypersphere([npoints, ndim, random_state])	Generate random samples from the hypersphere
stats(x[, y])	The stats function calculates statistics for one or multiple groups of data.
qq_plot(x[, y, dist])	The function qq_plot is used to create a quantile-quantile plot, either for a single sample or for two samples.
sphere3d(x[, y])	The function sphere3d creates a 3D scatter plot with a sphere as the surface and data points plotted on it.
plot_clusters_2d(x[, y])	This function plots a 2D scatter plot of data points, with an optional argument to color the points based on a cluster label, and also plots a unit circle.

sample_hypersphere

QuadratiK.tools.**sample_hypersphere**(*npoints=100*, *ndim=3*, *random_state=None*)

Generate random samples from the hypersphere

Parameters

npoints
[int, optional.] The number of points to generate. Default is 100.

ndim
[int, optional.] The dimensionality of the hypersphere. Default is 3.

random_state
[int, None, optional.] Seed for random number generation. Defaults to None

Returns

data on sphere
[numpy.ndarray] An array containing random vectors sampled uniformly from the surface of the hypersphere.

Examples

```
>>> from QuadratiK.tools import sample_hypersphere
>>> sample_hypersphere(100,3,random_state = 42)
... array([[ 0.60000205, -0.1670153 ,  0.78237039],
...        [ 0.97717133, -0.15023209, -0.15022156], ........
```

stats

`QuadratiK.tools.stats(x, y=None)`

The stats function calculates statistics for one or multiple groups of data.

Parameters

x
[numpy.ndarray, pandas.DataFrame] Data for which statistics is to be calculated.

y
[numpy.ndarray, pandas.DataFrame, optional] The parameter *y* is an optional input that can be either another set of observations, or the associated labels for observations (data points).

Returns

summary statistics
[pandas.DataFrame] Summary statistics of the input data.

Examples

```
>>> import numpy as np
>>> from QuadratiK.tools import stats
>>> np.random.seed(42)
>>> X = np.random.randn(100,4)
>>> stats(X)
...           Feature 0  Feature 1  Feature 2  Feature 3
    Mean     -0.009811   0.033746   0.022496   0.043764
    Std Dev   0.868065   0.952234   1.044014   0.982240
    Median   -0.000248  -0.024646   0.068665   0.075219
    IQR       1.244319   1.111478   1.318245   1.506492
    Min      -2.025143  -1.959670  -3.241267  -1.987569
    Max       2.314659   3.852731   2.189803   2.720169
```

qq_plot

`QuadratiK.tools.`**`qq_plot`**`(`*x*, *y=None*, *dist='norm'*`)`

The function qq_plot is used to create a quantile-quantile plot, either for a single sample or for two samples.

Parameters

x

[numpy.ndarray] The *x* parameter represents the data for which you want to create a QQ plot. It can be a single variable or an array-like object containing multiple variables

y

[numpy.ndarray, optional] The parameter *y* is an optional argument that represents the second sample for a two-sample QQ plot. If provided, the function will generate a QQ plot comparing the two samples

dist

[str, optional] Supports all the scipy.stats.distributions. The *dist* parameter specifies the distribution to compare the data against in the QQ plot. By default, it is set to "norm" which represents the normal distribution. However, you can specify a different distribution if you want to compare the data against a different distribution. Defaults to "norm".

Returns

Returns QQ plots.

Examples

```
>>> import numpy as np
>>> from QuadratiK.tools import qq_plot
>>> np.random.seed(42)
>>> X = np.random.randn(100,4)
>>> qq_plot(X)
```

sphere3d

`QuadratiK.tools.sphere3d(x, y=None)`

The function sphere3d creates a 3D scatter plot with a sphere as the surface and data points plotted on it.

Parameters

x
[numpy.ndarray, pandas.DataFrame] The parameter *x* represents the input data for the scatter plot. It should be a 2D array-like object with shape (n_samples, 3), where each row represents the coordinates of a point in 3D space

y
[numpy.ndarray, list, optional] The parameter *y* is an optional input that determines the color and shape of each data point in the plot. If *y* is not provided, the scatter plot will have the default marker symbol and color.

Returns

Returns a 3D plot of a sphere with data points plotted on it.

Examples

```
>>> from QuadratiK.tools import sphere3d
>>> np.random.seed(42)
>>> X = np.random.randn(100,3)
>>> sphere3d(X)
```

plot_clusters_2d

`QuadratiK.tools.plot_clusters_2d(x, y=None)`

This function plots a 2D scatter plot of data points, with an optional argument to color the points based on a cluster label, and also plots a unit circle.

Parameters

x
: [numpy.ndarray, pandas.DataFrame] The parameter *x* is a 2-dimensional array or matrix containing the coordinates of the data points to be plotted. Each row of *x* represents the coordinates of a single data point in the 2-dimensional space

y
: [numpy.ndarray, pandas.DataFrame, optional] The parameter *y* is an optional array that represents the labels or cluster assignments for each data point in *x*. If *y* is provided, the data points will be colored according to their labels or cluster assignments.

Returns

A matplotlib figure object.

Examples

```
>>> import numpy as np
>>> from QuadratiK.tools import plot_clusters_2d
>>> np.random.seed(42)
>>> X = np.random.randn(100,2)
>>> X = X/np.linalg.norm(X,axis = 1, keepdims=True)
>>> plot_clusters_2d(X)
```

1.3.3 User Guide

1.3.3.1 Dataset

Wireless Indoor Localization Dataset

The *wireless* data frame has 2000 rows and 8 columns. The first 7 variables report the measurements of the Wi-Fi signal strength received from 7 Wi-Fi routers in an office location in Pittsburgh (USA). The last column indicates the class labels.

Format

A data frame containing the following columns:

- *V1*: signal strength from router 1.
- *V2*: signal strength from router 2.
- *V3*: signal strength from router 3.
- *V4*: signal strength from router 4.
- *V5*: signal strength from router 5.
- *V6*: signal strength from router 6.
- *V7*: signal strength from router 7.

- *V8*: group memberships, from 1 to 4.

Details

The Wi-Fi signal strength is measured in dBm, decibel milliwatts, which is expressed as a negative value ranging from -100 to 0. The labels correspond to 'conference room', 'kitchen', 'indoor sports room', and 'other'. In total, we have 4 groups with 500 observations each.

Source

Bhatt,Rajen. (2017). Wireless Indoor Localization. UCI Machine Learning Repository. https://doi.org/10.24432/C51880.

References

Rohra, J.G., Perumal, B., Narayanan, S.J., Thakur, P., Bhatt, R.B. (2017). User Localization in an Indoor Environment Using Fuzzy Hybrid of Particle Swarm Optimization & Gravitational Search Algorithm with Neural Networks. In: Deep, K., et al. Proceedings of Sixth International Conference on Soft Computing for Problem Solving. Advances in Intelligent Systems and Computing, vol 546. Springer, Singapore. https://doi.org/10.1007/978-981-10-3322-3_27

1.3.3.2 Usage Examples

QuadratiK Usage Examples

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(42)
import pandas as pd
```

Normality Test

This section contains example for the Parametric and Non-parametric Normality Test based on kernel-based quadratic distances

Parametric

[2]:
```
from QuadratiK.kernel_test import KernelTest

data = np.random.randn(100,2)

normality_test = KernelTest(h=0.4, centering_type="param",random_state=42).test(data)
print("Test : {}".format(normality_test.test_type_))
print("Execution time: {:.3f}".format(normality_test.execution_time))
print("H0 is Rejected : {}".format(normality_test.h0_rejected_))
print("Test Statistic : {}".format(normality_test.test_statistic_))
```

(continues on next page)

(continued from previous page)

```
print("Critical Value (CV) : {}".format(normality_test.cv_))
print("CV Method : {}".format(normality_test.cv_method_))
print("Selected tuning parameter : {}".format(normality_test.h))
```

```
Test : Kernel-based quadratic distance Normality test
Execution time: 1.776
H0 is Rejected : False
Test Statistic : -0.004422397826208057
Critical Value (CV) : 0.00495159345113745
CV Method : Empirical
Selected tuning parameter : 0.4
```

[3]:
```
print(normality_test.summary())
```

```
Time taken for execution: 1.776 seconds
Test Results
--------------  ----------------------------------------------
Test Type       Kernel-based quadratic distance Normality test
Test Statistic  -0.004422397826208057
Critical Value  0.00495159345113745
Reject H0       False
--------------  ----------------------------------------------
Summary Statistics
           Feature 0    Feature 1
-------  -----------  -----------
Mean         -0.1156        0.034
Std Dev       0.8563        0.9989
Median       -0.0353        0.1323
IQR           1.0704        1.3333
Min          -2.6197       -1.9876
Max           1.8862        2.7202
```

Non-parametric

[4]:
```
normality_test = KernelTest(h=0.4, centering_type="nonparam").test(data)
print("Test : {}".format(normality_test.test_type_))
print("Execution time: {:.3f}".format(normality_test.execution_time))
print("H0 is Rejected : {}".format(normality_test.h0_rejected_))
print("Test Statistic : {}".format(normality_test.test_statistic_))
print("Critical Value (CV) : {}".format(normality_test.cv_))
print("CV Method : {}".format(normality_test.cv_method_))
print("Selected tuning parameter : {}".format(normality_test.h))
```

```
Test : Kernel-based quadratic distance Normality test
Execution time: 0.118
H0 is Rejected : False
Test Statistic : 0.0015387891795935942
Critical Value (CV) : 0.002029405786586928
CV Method : Empirical
Selected tuning parameter : 0.4
```

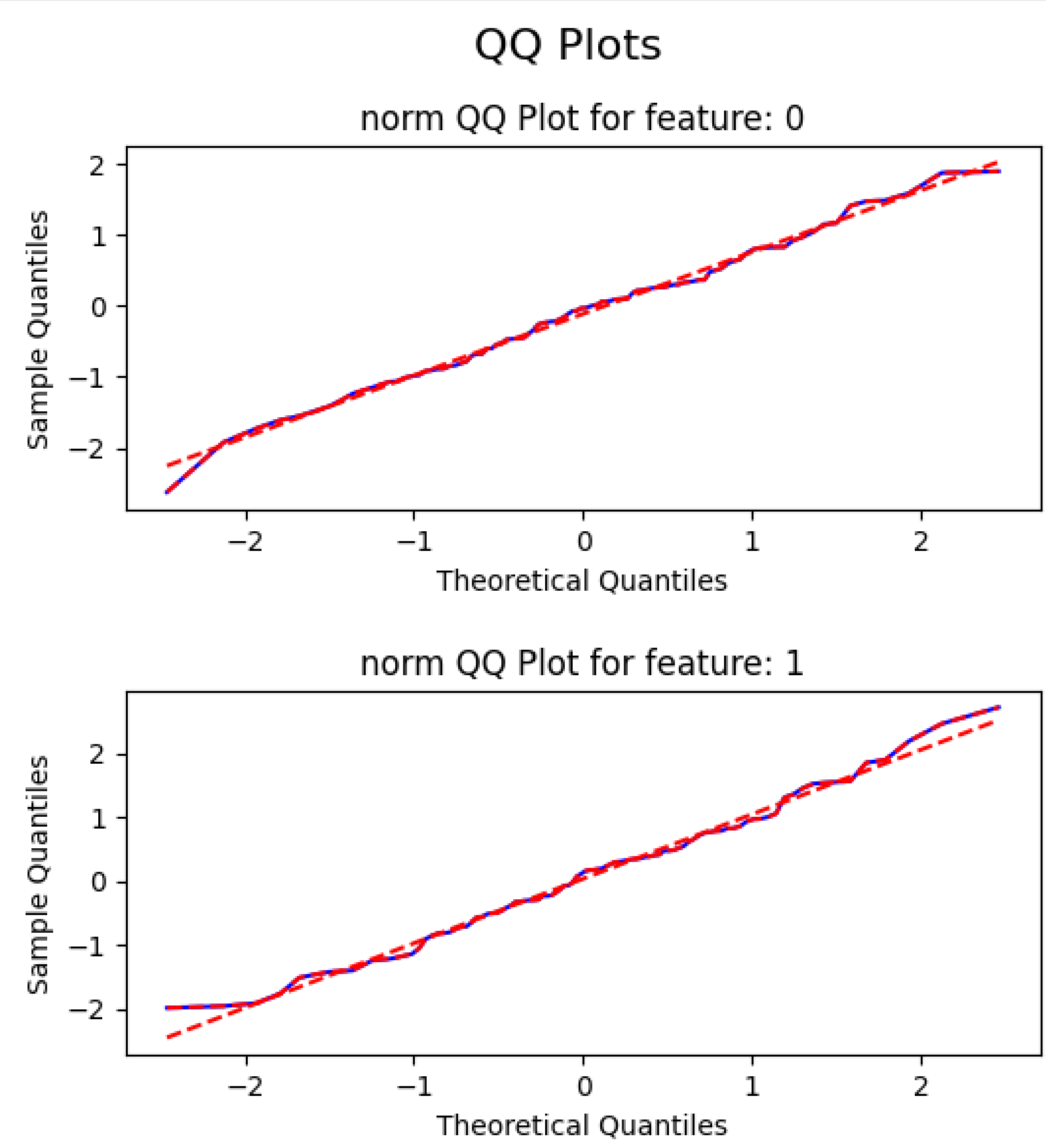
[5]:
```
print(normality_test.summary())
```

```
Time taken for execution: 0.118 seconds
Test Results
--------------  ----------------------------------------------
Test Type       Kernel-based quadratic distance Normality test
Test Statistic  0.0015387891795935942
Critical Value  0.002029405786586928
Reject H0       False
--------------  ----------------------------------------------
Summary Statistics
           Feature 0    Feature 1
-------  -----------  -----------
Mean         -0.1156        0.034
Std Dev       0.8563       0.9989
Median       -0.0353       0.1323
IQR           1.0704       1.3333
Min          -2.6197      -1.9876
Max           1.8862       2.7202
```

QQ Plot

[6]:
```
from QuadratiK.tools import qq_plot
qq_plot(data)
```

[6]:



Two Sample Test

This sections shows example for the two-sample test using normal kernel-based quadratic distance

[7]:
```
from QuadratiK.kernel_test import KernelTest

X = np.random.randn(100,2)
Y = np.random.randn(100,2)

two_sample_test = KernelTest(h=0.4, random_state=42).test(X,Y)
print("Test : {}".format(two_sample_test.test_type_))
print("Execution time: {:.3f}".format(two_sample_test.execution_time))
print("H0 is Rejected : {}".format(two_sample_test.h0_rejected_))
print("Test Statistic : {}".format(two_sample_test.test_statistic_))
```
(continues on next page)

(continued from previous page)

```
print("Critical Value (CV) : {}".format(two_sample_test.cv_))
print("CV Method : {}".format(two_sample_test.cv_method_))
print("Selected tuning parameter : {}".format(two_sample_test.h))
```

```
Test : Kernel-based quadratic distance two-sample test
Execution time: 0.048
H0 is Rejected : False
Test Statistic : 0.0029175386660962063
Critical Value (CV) : 0.00931651659981921
CV Method : subsampling
Selected tuning parameter : 0.4
```

[8]:

```
print(two_sample_test.summary())
```

```
Time taken for execution: 0.048 seconds
Test Results
--------------  -----------------------------------------------
Test Type       Kernel-based quadratic distance two-sample test
Test Statistic  0.0029175386660962063
Critical Value  0.00931651659981921
Reject H0       False
--------------  -----------------------------------------------
Summary Statistics
                            Group 1    Group 2    Overall
------------------------  ---------  ---------  ---------
('Feature 0', 'Mean')        0.1282    -0.045      0.0416
('Feature 0', 'Std Dev')     1.0396     1.025      1.0334
('Feature 0', 'Median')      0.1056     0.0118     0.0737
('Feature 0', 'IQR')         1.4912     1.2409     1.345
('Feature 0', 'Min')        -3.2413    -2.4716    -3.2413
('Feature 0', 'Max')         2.3147     3.0789     3.0789
('Feature 1', 'Mean')        0.0435    -0.1263    -0.0414
('Feature 1', 'Std Dev')     0.9348     0.9656     0.9517
('Feature 1', 'Median')      0.0114    -0.1967    -0.1224
('Feature 1', 'IQR')         1.2379     1.4056     1.3208
('Feature 1', 'Min')        -1.9521    -2.3019    -2.3019
('Feature 1', 'Max')         3.8527     2.2707     3.8527
```

K-Sample Test

Shows examples for the kernel-based quadratic distance k-sample tests with the Normal kernel and bandwidth parameter h.

[9]:

```
from QuadratiK.kernel_test import KernelTest
X = np.random.randn(500,2)
y = np.random.randint(0,5,500)

k_sample_test = KernelTest(h = 1.5, method = "permutation").test(X,y)

print("Test : {}".format(k_sample_test.test_type_))
print("Execution time: {:.3f} seconds".format(k_sample_test.execution_time))
```

(continues on next page)

(continued from previous page)

```
print("H0 is Rejected : {}".format(k_sample_test.h0_rejected_))
print("Test Statistic : {}".format(k_sample_test.test_statistic_))
print("Critical Value (CV) : {}".format(k_sample_test.cv_))
print("CV Method : {}".format(k_sample_test.cv_method_))
print("Selected tuning parameter : {}".format(k_sample_test.h))
```

```
Test : Kernel-based quadratic distance K-sample test
Execution time: 0.343 seconds
H0 is Rejected : False
Test Statistic : [0.00154197 0.00038549]
Critical Value (CV) : [0.00367584 0.00091896]
CV Method : permutation
Selected tuning parameter : 1.5
```

[10]:
```
print(k_sample_test.summary())
```

```
Time taken for execution: 0.343 seconds
Test Results
--------------  --------------------------------------------
Test Type       Kernel-based quadratic distance K-sample test
Test Statistic  [0.00154197 0.00038549]
Critical Value  [0.00367584 0.00091896]
Reject H0       False
--------------  --------------------------------------------
Summary Statistics
                           Group 0    Group 1    Group 2    Group 3    Group 4    ␣
↪Overall
-------------------------  ---------  ---------  ---------  ---------  ---------  --------
↪-
('Feature 0', 'Mean')      -0.0309    -0.0533     0.1141     0.3318    -0.0612     0.
↪0721
('Feature 0', 'Std Dev')    0.8923     0.8846     1.0046     0.9959     0.8949     0.
↪9497
('Feature 0', 'Median')    -0.035      0.0282     0.0386     0.3026     0.0104     0.
↪0463
('Feature 0', 'IQR')        1.2255     1.1546     1.336      1.3052     1.1691     1.
↪2548
('Feature 0', 'Min')       -2.4994    -2.3629    -2.6969    -2.4242    -2.5539    -2.
↪6969
('Feature 0', 'Max')        1.849      1.8846     2.5601     2.5734     1.882      2.
↪5734
('Feature 1', 'Mean')       0.2104     0.1435     0.1877    -0.0868    -0.0808     0.
↪0715
('Feature 1', 'Std Dev')    0.9641     1.0551     1.0554     1.0276     1.1685     1.
↪0591
('Feature 1', 'Median')     0.279      0.0861     0.1893    -0.1258    -0.1441     0.
↪1132
('Feature 1', 'IQR')        1.3873     1.2558     1.323      1.3384     1.5526     1.
↪4146
('Feature 1', 'Min')       -1.9664    -2.591     -2.8723    -2.8485    -2.9214    -2.
↪9214
('Feature 1', 'Max')        2.2909     2.6017     2.5797     2.6324     2.5582     2.
↪6324
```

Poisson Kernel Test

Shows example for perforing the the kernel-based quadratic distance Goodness-of-fit tests for Uniformity for spherical data using the Poisson kernel with concentration parameter rho.

[11]:
```
from QuadratiK.tools import sample_hypersphere
from QuadratiK.poisson_kernel_test import PoissonKernelTest

X = sample_hypersphere(100,3, random_state=42)

unif_test = PoissonKernelTest(rho = 0.7, random_state=42).test(X)

print("Execution time: {:.3f} seconds".format(unif_test.execution_time))

print("U Statistic Results")
print("H0 is rejected : {}".format(unif_test.u_statistic_h0_))
print("Un Statistic : {}".format(unif_test.u_statistic_un_))
print("Critical Value : {}".format(unif_test.u_statistic_cv_))

print("V Statistic Results")
print("H0 is rejected : {}".format(unif_test.v_statistic_h0_))
print("Vn Statistic : {}".format(unif_test.v_statistic_vn_))
print("Critical Value : {}".format(unif_test.v_statistic_cv_))
```

```
Execution time: 0.068 seconds
U Statistic Results
H0 is rejected : False
Un Statistic : 1.6156682048968174
Critical Value : 0.06155875299050079
V Statistic Results
H0 is rejected : False
Vn Statistic : 22.83255917641962
Critical Value : 23.229486935225513
```

[12]:
```
print(unif_test.summary())
```

```
Time taken for execution: 0.068 seconds
Test Results
--------------------------  -------------------
Test Type                   Poisson Kernel-based quadratic
 distance test of Uniformity on the Sphere
U Statistic Un              1.6156682048968174
U Statistic Critical Value  0.06155875299050079
U Statistic Reject H0       False
V Statistic Vn              22.83255917641962
V Statistic Critical Value  23.229486935225513
V Statistic Reject H0       False
--------------------------  -------------------
Summary Statistics
           Feature 0    Feature 1    Feature 2
-------  -----------  -----------  -----------
Mean          0.0451      -0.1206       0.0309
Std Dev       0.509        0.5988       0.6122
Median        0.132       -0.1596       0.0879
```

(continues on next page)

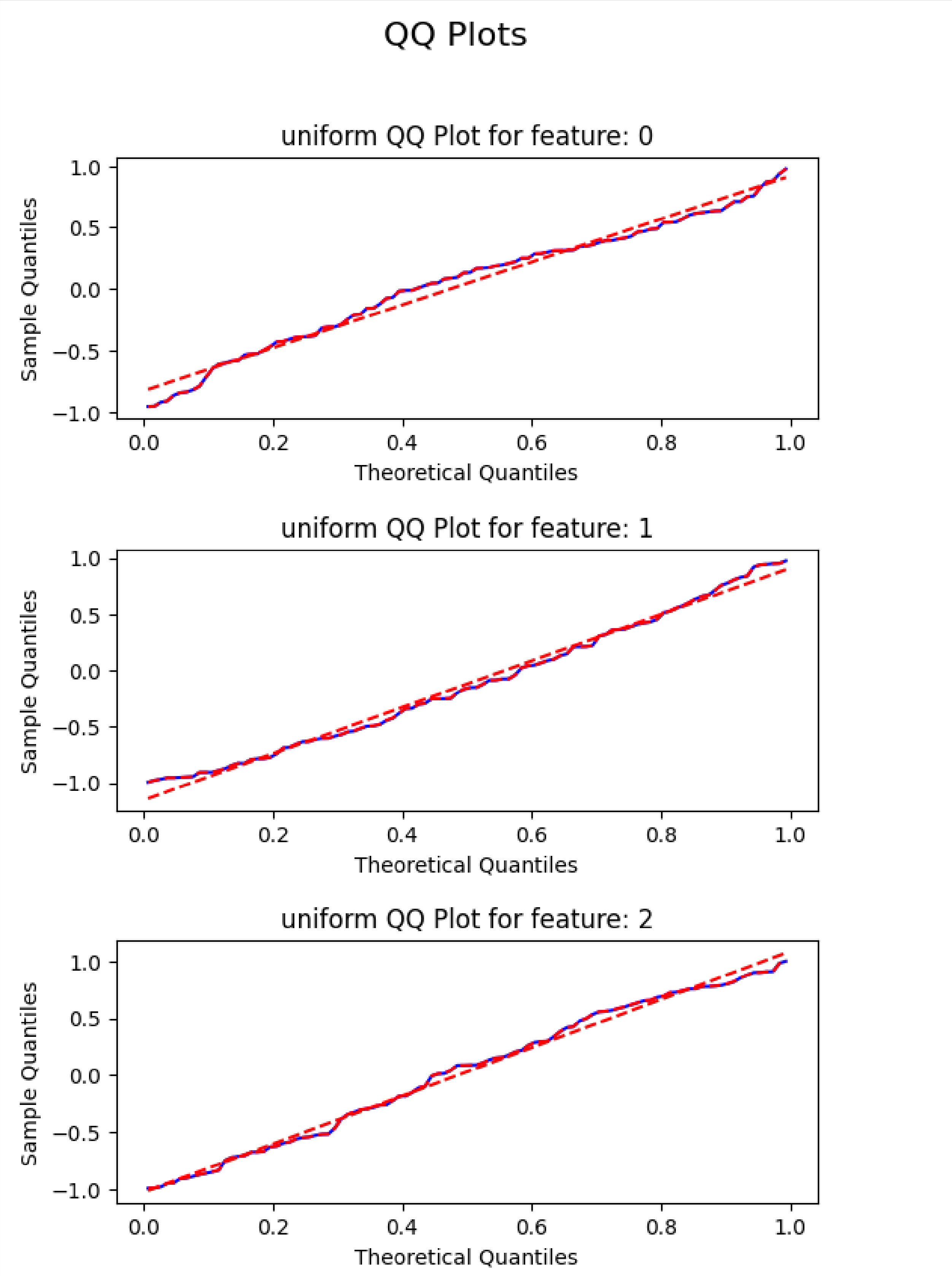
(continued from previous page)

```
IQR           0.8051       1.0063       1.1473
Min          -0.9548      -0.9929      -0.9904
Max           0.9772       0.9738       0.9996
```

QQ Plot

[13]:
```
from QuadratiK.tools import qq_plot

qq_plot(X,dist = "uniform")
```

[13]:


QQ Plots
uniform QQ Plot for feature: 0
uniform QQ Plot for feature: 1
uniform QQ Plot for feature: 2
Sample Quantiles
Theoretical Quantiles
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

Poisson Kernel based Clustering

Shows example for performing the Poisson kernel-based clustering algorithm on the Sphere based on the Poisson kernel-based densities.

[14]:
```python
from QuadratiK.datasets import load_wireless_data
from QuadratiK.spherical_clustering import PKBC
from sklearn.preprocessing import LabelEncoder

X, y = load_wireless_data(return_X_y=True)

le = LabelEncoder()
le.fit(y)
y = le.transform(y)

cluster_fit = PKBC(num_clust=4, random_state=42).fit(X)
ari, macro_precision, macro_recall, avg_silhouette_Score = cluster_fit.validation(y)

print("Estimated mixing proportions :", cluster_fit.alpha_)
print("Estimated concentration parameters: ", cluster_fit.rho_)

print("Adjusted Rand Index:", ari)
print("Macro Precision:", macro_precision)
print("Macro Recall:", macro_recall)
print("Average Silhouette Score:", avg_silhouette_Score)
```

```
Estimated mixing proportions : [0.23590339 0.24977919 0.25777522 0.25654219]
Estimated concentration parameters:  [0.97773265 0.98348976 0.98226901 0.98572597]
Adjusted Rand Index: 0.9403086353805835
Macro Precision: 0.9771870612442508
Macro Recall: 0.9769999999999999
Average Silhouette Score: 0.3803089203572107
```

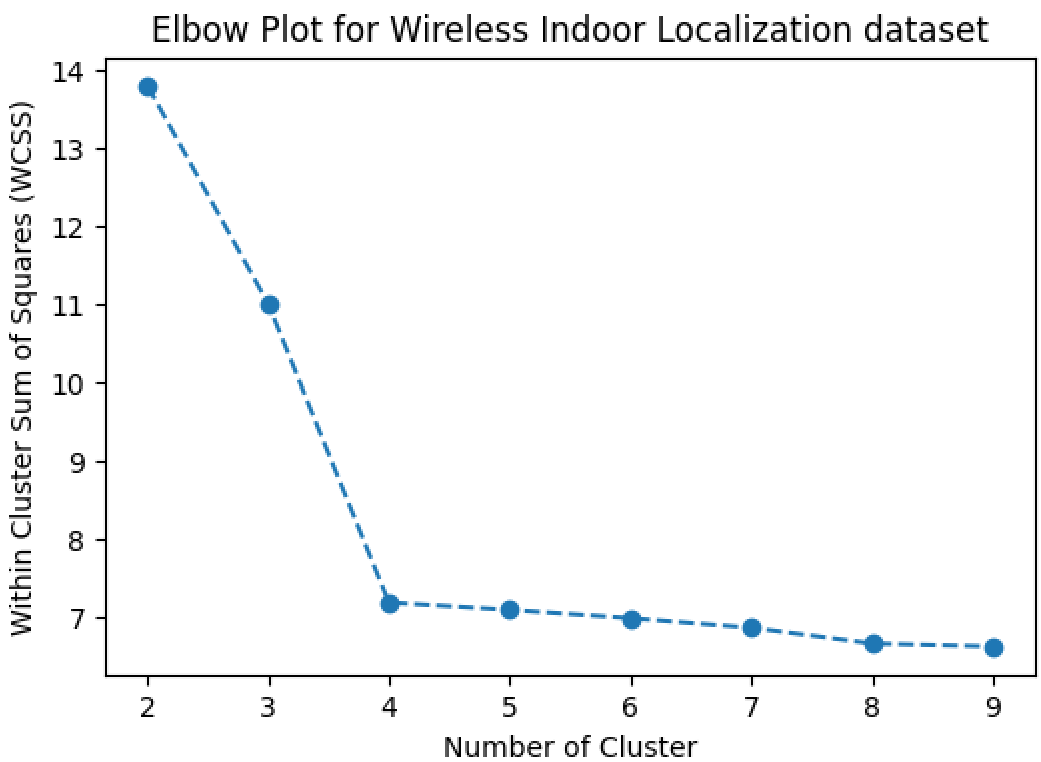
Elbow Plot using Euclidean Distance and Cosine Similarity based WCSS

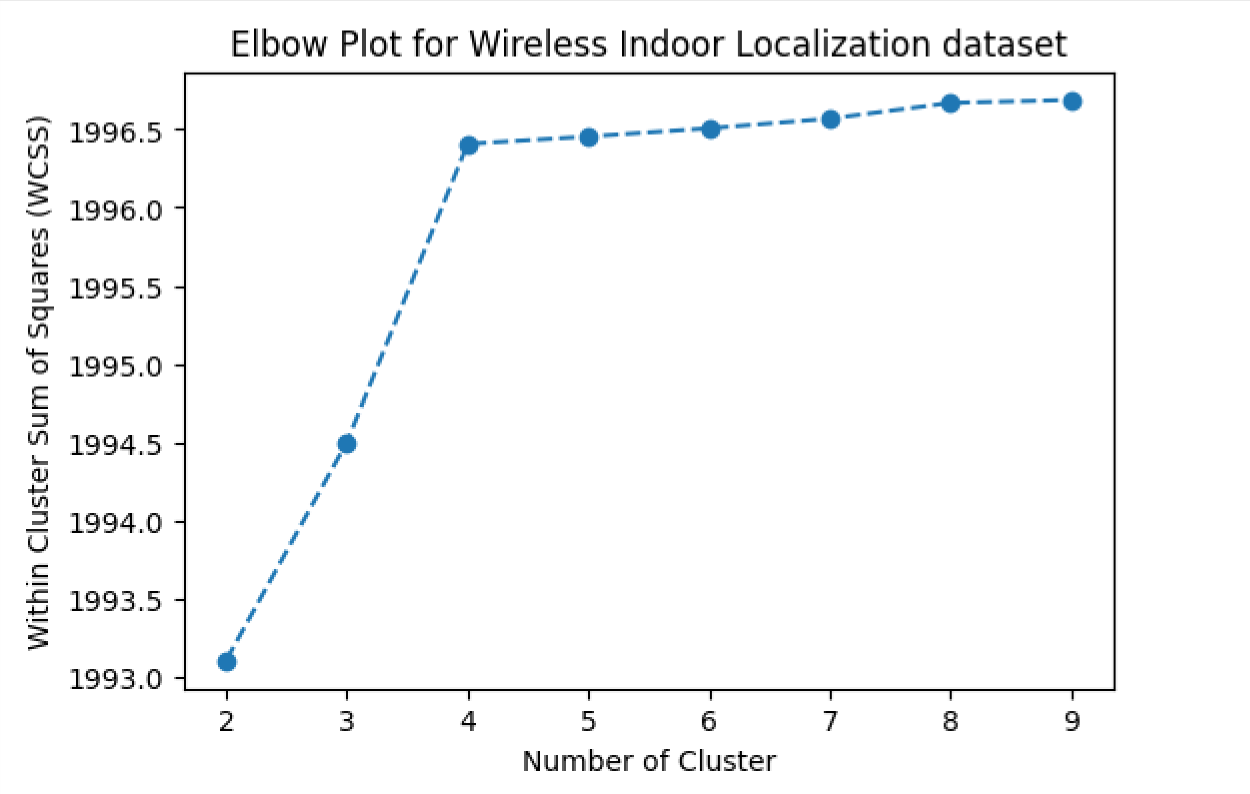
[15]:
```python
wcss_euc = []
wcss_cos = []

for i in range(2, 10):
    clus_fit = PKBC(num_clust=i).fit(X)
    wcss_euc.append(clus_fit.euclidean_wcss_)
    wcss_cos.append(clus_fit.cosine_wcss_)

fig = plt.figure(figsize=(6, 4))
plt.plot(list(range(2, 10)), wcss_euc, "--o")
plt.xlabel("Number of Cluster")
plt.ylabel("Within Cluster Sum of Squares (WCSS)")
plt.title("Elbow Plot for Wireless Indoor Localization dataset")
plt.show()

fig = plt.figure(figsize=(6, 4))
plt.plot(list(range(2,10)),wcss_cos, "--o")
```

(continues on next page)

(continued from previous page)

```
plt.xlabel("Number of Cluster")
plt.ylabel("Within Cluster Sum of Squares (WCSS)")
plt.title("Elbow Plot for Wireless Indoor Localization dataset")
plt.show()
```





Density Estimation and Sample Generation from PKBD

[16]:
```
from QuadratiK.spherical_clustering import PKBD
pkbd_data = PKBD().rpkb(10,[0.5,0],0.5, "rejvmf", random_state= 42)
dens_val  = PKBD().dpkb(pkbd_data, [0.5,0.5],0.5)
print(dens_val)
```

```
[0.46827108 0.05479605 0.21163936 0.06195099 0.39567698 0.40473724
 0.26561508 0.36791766 0.09324676 0.46847274]
```

Tuning Parameter h selection

Computes the kernel bandwidth of the Gaussian kernel for the two-sample and ksample kernel-based quadratic distance (KBQD) tests.

[17]:
```
from QuadratiK.kernel_test import select_h

X = np.random.randn(200, 2)
y = np.random.randint(0, 2, 200)

h_selected, all_values, power_plot = select_h(
    X, y, alternative='location', power_plot=True,  random_state=None)
print("Selected h is: ", h_selected)
```

```
Selected h is:  1.6
```

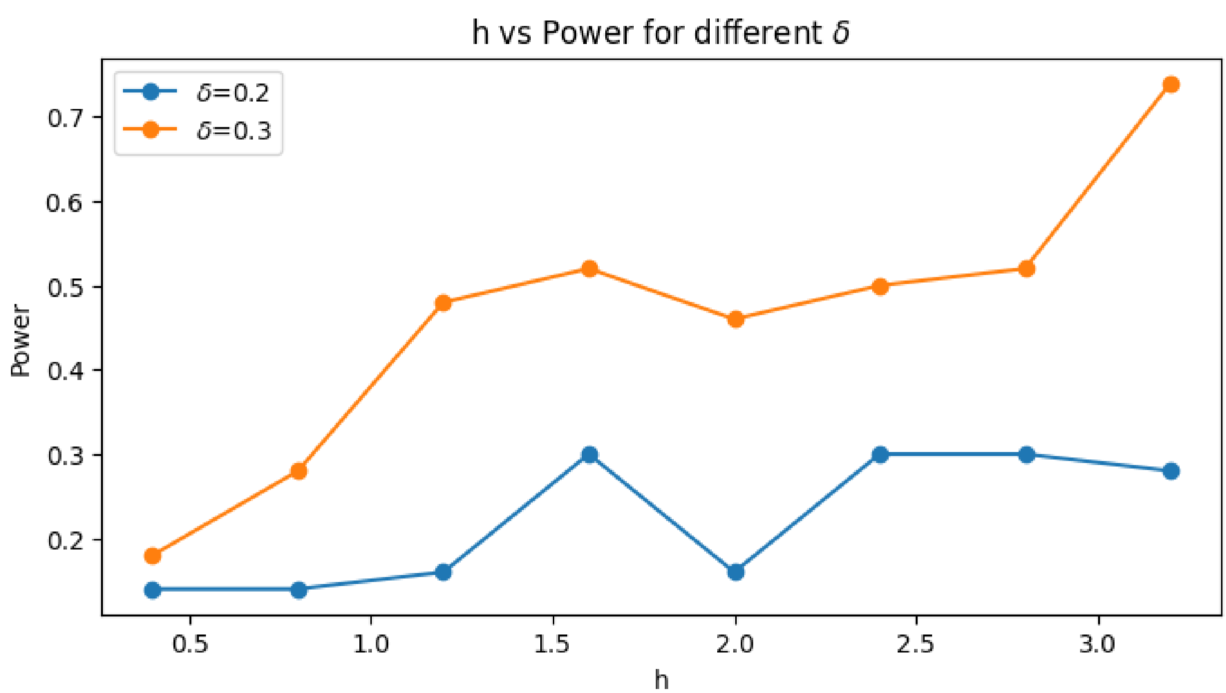
[18]:
```
#shows the detailed power vs h table
all_values
```

[18]:

	h	delta	power
0	0.4	0.2	0.14
1	0.8	0.2	0.14
2	1.2	0.2	0.16
3	1.6	0.2	0.30
4	2.0	0.2	0.16
5	2.4	0.2	0.30
6	2.8	0.2	0.30
7	3.2	0.2	0.28
0	0.4	0.3	0.18
1	0.8	0.3	0.28
2	1.2	0.3	0.48
3	1.6	0.3	0.52
4	2.0	0.3	0.46
5	2.4	0.3	0.50
6	2.8	0.3	0.52
7	3.2	0.3	0.74

[19]:
```
#shows the power plot
power_plot
```

[19]:



1.3.4 Development

PYTHON MODULE INDEX

q