Distance Measures and Clustering Group T4-2

1.0

Generated by Doxygen 1.9.1

Chapter 1

Description

This is a group project done for the Lecture "Data Science 1" at the Goethe University Frankfurt exploring the effects of different distance measures on distance-based clustering algorithm.

1.1 Frontend

The web frontend is accessible here: `Web Tool`

1.2 Documentation

The doxygen Documentation of the codebase can be acessed here: `Docs` A PDF Documentation is also available in the docs directory (file: `documentation.pdf`)

1.3 Dependencies and Sources

The SessionState.py is directly taken from a `gist` by Thiago Teixeira, user `tvst` on github. We take absolutly no credit for it!

The code for the altair chart used for displaying the DBSCAN heuristic is based heavily upon the multiline tooltip example from the altair example gallery `Link`.

The project depends on following python libraries:

- `matplotlib`
- `numpy`
- `pandas`
- `pyclustering`
- `scikit-learn`
- `scikit-learn-extra`
- `seaborn`
- `streamlit`
- `altair`

1.4 ToDos

- [ ] Web Frontend Anleitung
- [ ] Web Frontend Doxygen Doku?
- [ ] Conclusion
- [ ] Abstract

1.5 Nice-To-Haves

- [ ] Dendrogramm
- [ ] Diskriminanzanalyse (Trennfähigste Variablen)

1.6 Authors

- Niklas Conen
- Jonas Elpelt
- Franziska Hicking
- Julian Rummel

So long, and thanks for all the fish

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 clustering Namespace Reference

Classes

- class Clustering

  Base Class for all subsequent clustering algorithms implements all functions needed for running the different cluster algorithms.

6.2 comparison_plots Namespace Reference

Variables

- list kalgos = ['kmeans', 'kmedians', 'kmedoids']
- dictionary kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids'↩ : kmedoidsClustering}
- list distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list datasets = ["iris", "wine", "diabetes", "housevotes"]
- list index_ext_eval = ["ARI", "NMI", "Completeness Score", "Homogeneity Score"]
- list index_int_eval = ["Silhouette Score"]
- list num_of_classes = [3,3,2,2]
- int seed = 42
- results = Results("./results")
- all_kalgos = np.zeros((3,4, 9,len(index_ext_eval)))
- all_kalgos_int = pd.DataFrame(columns=['k', 'Distance (Silhouette Score)', 'Distance (Clustering)', 'sil_score', 'kalgo'])
- all_dist = np.zeros((4, 9,len(index_ext_eval)))
- all_k = np.zeros((9,len(index_ext_eval)))
- clusters
- stuff
- s
- c
- d
- k
- dictionary cluster = kalgoclass[c](d, s, seed)

- clustered_data = np.zeros(len(cluster.data))
- dictionary labels = cluster.labels.tolist()
- predicted = clustered_data.tolist()
- I1 = Indices(predicted, labels)
- index_scores = np.zeros_like(index_ext_eval, dtype=float)
- index_score = I1.index_internal(index=index_int_eval[0], points=cluster.data.tolist(), metric=di)
- index
- fig = plt.figure(figsize=(15, 10))
- bbox_to_anchor
- loc
- borderaxespad
- ax
- figsize
- data
- x
- y
- hue
- style
- legend
- all_kalgos_df = pd.DataFrame(all_kalgos[:,:,num_of_classes[isx]-2,i], columns=distances, index=kalgos)
- kind
- title

6.2.1 Variable Documentation

6.2.1.1 all_dist

```
comparison_plots.all_dist = np.zeros((4, 9,len(index_ext_eval)))
```

6.2.1.2 all_k

```
comparison_plots.all_k = np.zeros((9,len(index_ext_eval)))
```

6.2.1.3 all_kalgos

```
comparison_plots.all_kalgos = np.zeros((3,4, 9,len(index_ext_eval)))
```

6.2.1.4 all_kalgos_df

```
comparison_plots.all_kalgos_df = pd.DataFrame(all_kalgos[:,:,num_of_classes[isx]-2,i], columns=distances,
index=kalgos)
```

6.2.1.5 all_kalgos_int

```
comparison_plots.all_kalgos_int = pd.DataFrame(columns=['k', 'Distance (Silhouette Score)',
'Distance (Clustering)', 'sil_score', 'kalgo'])
```

6.2.1.6 ax

```
comparison_plots.ax
```

6.2.1.7 bbox_to_anchor

```
comparison_plots.bbox_to_anchor
```

6.2.1.8 borderaxespad

```
comparison_plots.borderaxespad
```

6.2.1.9 c

```
comparison_plots.c
```

6.2.1.10 cluster

```
dictionary comparison_plots.cluster = kalgoclass[c](d, s, seed)
```

6.2.1.11 clustered_data

```
comparison_plots.clustered_data = np.zeros(len(cluster.data))
```

6.2.1.12 clusters

```
comparison_plots.clusters
```

6.2.1.13 d

```
comparison_plots.d
```

6.2.1.14 data

```
comparison_plots.data
```

6.2.1.15 datasets

```
list comparison_plots.datasets = ["iris", "wine", "diabetes", "housevotes"]
```

6.2.1.16 distances

```
comparison_plots.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

6.2.1.17 fig

```
comparison_plots.fig = plt.figure(figsize=(15, 10))
```

6.2.1.18 figsize

```
comparison_plots.figsize
```

6.2.1.19 hue

```
comparison_plots.hue
```

6.2.1.20 I1

```
comparison_plots.I1 = Indices(predicted, labels)
```

6.2.1.21 index

```
comparison_plots.index
```

6.2.1.22 index_ext_eval

```
list comparison_plots.index_ext_eval = ["ARI", "NMI", "Completeness Score", "Homogeneity Score"]
```

6.2.1.23 index_int_eval

```
list comparison_plots.index_int_eval = ["Silhouette Score"]
```

6.2.1.24 index_score

```
comparison_plots.index_score = I1.index_internal(index=index_int_eval[0], points=cluster.↩
data.tolist(), metric=di)
```

6.2.1.25 index_scores

```
comparison_plots.index_scores = np.zeros_like(index_ext_eval, dtype=float)
```

6.2.1.26 k

```
comparison_plots.k
```

6.2.1.27 kalgoclass

```
dictionary comparison_plots.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,
'kmedoids': kmedoidsClustering}
```

6.2.1.28 kalgos

```
list comparison_plots.kalgos = ['kmeans', 'kmedians', 'kmedoids']
```

6.2.1.29 kind

```
comparison_plots.kind
```

6.2.1.30 labels

```
dictionary comparison_plots.labels = cluster.labels.tolist()
```

6.2.1.31 legend

```
comparison_plots.legend
```

6.2.1.32 loc

```
comparison_plots.loc
```

6.2.1.33 num_of_classes

```
list comparison_plots.num_of_classes = [3,3,2,2]
```

6.2.1.34 predicted

```
comparison_plots.predicted = clustered_data.tolist()
```

6.2.1.35 results

```
comparison_plots.results = Results("./results")
```

6.2.1.36 s

```
comparison_plots.s
```

6.2.1.37 seed

```
int comparison_plots.seed = 42
```

6.2.1.38 stuff

```
comparison_plots.stuff
```

6.2.1.39 style

```
comparison_plots.style
```

6.2.1.40 title

```
comparison_plots.title
```

6.2.1.41 x

```
comparison_plots.x
```

6.2.1.42 y

```
comparison_plots.y
```

6.3 dbscan Namespace Reference

Classes

- class DBSCANClustering

  implements DBSCAN Clustering
  uses the scikit-learn DBSCAN implementation

6.4 dbscan_heuristic Namespace Reference

Classes

- class DBSCANHeuristic

  implements the DBSCAN heuristic proposed in the original DBSCAN paper:

6.5 heuristic_web Namespace Reference

Variables

- page_title
- page_icon
- key
- col1
- col2
- dataset = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes', 'housevotes'])
- dictionary cluster_dist_desc
- cluster_dist = col1.selectbox('Choose an awesome distance measure',list(cluster_dist_desc.keys()))
- k = col2.slider("Choose a nice value for k", min_value=1, max_value=20, step=1, value=4)
- submit_button = st.form_submit_button(label='Calculate kdist Graph')
- heu = DBSCANHeuristic()
- kdist = heu.kdist(k)
- reverse
- df
- nearest = alt.selection(type='single', nearest=True, on='mouseover', fields=['points'], empty='none')
- yaxis = alt.Y("dist", axis=alt.Axis(title=f"{k}-dist"))
- line
- selectors = alt.Chart(df).mark_point().encode(x='points', opacity=alt.value(0)).add_selection(nearest)
- points = line.mark_point(color="red").encode(opacity=alt.condition(nearest, alt.value(1), alt.value(0)))
- text = line.mark_text(align='left', dx=5, dy=-5, color="red").encode(text=alt.condition(nearest, "label:N", alt.↵ value(' '))).transform_calculate(label=f'"distance: " + format(datum.dist, ".2f")')
- textp = line.mark_text(align='left', dx=5, dy=-15, color="red").encode(text=alt.condition(nearest, "label:N", alt.value(' '))).transform_calculate(label=f'format((1 - (datum.points-1) / {len(kdist)}) ∗ 100, ".2f") + "% core points"')
- rules = alt.Chart(df).mark_rule(color='gray').encode(x="points").transform_filter(nearest)
- use_container_width

6.5.1 Variable Documentation

6.5.1.1 cluster_dist

```
heuristic_web.cluster_dist = col1.selectbox('Choose an awesome distance measure',list(cluster←
_dist_desc.keys()))
```

6.5.1.2 cluster_dist_desc

```
dictionary heuristic_web.cluster_dist_desc
```

Initial value:

```
=  {’euclidean’: ’d(x,y) = \sqrt{\sum_{i=1}^{n}(|x_i-y_i|)^2}’,
                        ’manhattan’: ’d(x,y) = \sum\limits_{i=1}^{n}|x_i - y_i|’,
                        ’chebyshev’: ’d(x,y) = \max(|x_i - y_i|)’,
                        ’cosine’: ’d(x,y) = \\frac{\\arccos(\\frac{\sum_{i=1}^{n} x_i
      y_i}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}})}{\pi}’}
```

6.5.1.3 col1

```
heuristic_web.col1
```

6.5.1.4 col2

```
heuristic_web.col2
```

6.5.1.5 dataset

```
heuristic_web.dataset = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes',
'housevotes'])
```

6.5.1.6 df

```
heuristic_web.df
```

Initial value:

```
=  pd.DataFrame(
    [[i+1, kdist[i]] for i in range(len(kdist))],
    columns=["points", "dist"])
```

6.5.1.7 heu

```
heuristic_web.heu = DBSCANHeuristic()
```

6.5.1.8 k

```
heuristic_web.k = col2.slider("Choose a nice value for k", min_value=1, max_value=20, step=1,
value=4)
```

6.5.1.9 kdist

```
heuristic_web.kdist = heu.kdist(k)
```

6.5.1.10 key

```
heuristic_web.key
```

6.5.1.11 line

```
heuristic_web.line
```

Initial value:

```
=  alt.Chart(df).mark_line(point=True).encode(x="points", y=yaxis).properties(
                     title=f"DBSCAN Heuristic k={k}, {cluster_dist} distance")
```

6.5.1.12 nearest

```
heuristic_web.nearest = alt.selection(type='single', nearest=True, on='mouseover', fields=['points'],
empty='none')
```

6.5.1.13 page_icon

```
heuristic_web.page_icon
```

6.5.1.14 page_title

```
heuristic_web.page_title
```

6.5.1.15 points

```
heuristic_web.points = line.mark_point(color="red").encode(opacity=alt.condition(nearest,
alt.value(1), alt.value(0)))
```

6.5.1.16 reverse

```
heuristic_web.reverse
```

6.5.1.17 rules

```
heuristic_web.rules = alt.Chart(df).mark_rule(color='gray').encode(x="points").transform_↩
filter(nearest)
```

6.5.1.18 selectors

```
heuristic_web.selectors = alt.Chart(df).mark_point().encode(x='points', opacity=alt.value(0)).add↩
_selection(nearest)
```

6.5.1.19 submit_button

```
heuristic_web.submit_button = st.form_submit_button(label='Calculate kdist Graph')
```

6.5.1.20 text

```
heuristic_web.text = line.mark_text(align='left', dx=5, dy=-5, color="red").encode(text=alt.↩
condition(nearest, "label:N", alt.value(' '))).transform_calculate(label=f'"distance:  " +
format(datum.dist, ".2f")')
```

6.5.1.21 textp

```
heuristic_web.textp = line.mark_text(align='left', dx=5, dy=-15, color="red").encode(text=alt.↩
condition(nearest, "label:N", alt.value(' '))).transform_calculate(label=f'format( (1 - (datum.↩
points-1) / {len(kdist)}) * 100, ".2f") + "% core points"')
```

6.5.1.22 use_container_width

```
heuristic_web.use_container_width
```

6.5.1.23 yaxis

```
heuristic_web.yaxis = alt.Y("dist", axis=alt.Axis(title=f"{k}-dist"))
```

6.6 indices Namespace Reference

Classes

- class Indices

    calculates Indices for computed cluster labels uses the scikit library

6.7 kmeans Namespace Reference

Classes

- class kmeansClustering

    Class implementing k-Means Clustering
    uses the pyclustering k-means implementation
    centers can be initialised using the k++ or the random initialiser.

6.8 kmedians Namespace Reference

Classes

- class kmediansClustering

    implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser

6.9 kmedoids Namespace Reference

Classes

- class kmedoidsClustering

  implements k-Medians Clustering
  uses the scikit-learn-extra k-medoids implementation
  centers are set using the k++ initialiser if not set differently

6.10 result_calculation Namespace Reference

Variables

- list kalgos = ['kmeans', 'kmedians', 'kmedoids']
- dictionary kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids'↩ : kmedoidsClustering}
- list distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list datasets = ["iris", "wine", "diabetes", "housevotes"]
- int seed = 42
- results = Results("./results")
- s
- c
- d
- k
- dictionary alg = kalgoclass[c](d, s, seed)
- clusters
- centers
- minpts
- m
- eps

6.10.1 Variable Documentation

6.10.1.1 alg

```
result_calculation.alg = kalgoclass[c](d, s, seed)
```

6.10.1.2 c

```
result_calculation.c
```

6.10.1.3 centers

```
result_calculation.centers
```

6.10.1.4 clusters

```
result_calculation.clusters
```

6.10.1.5 d

```
result_calculation.d
```

6.10.1.6 datasets

```
list result_calculation.datasets = ["iris", "wine", "diabetes", "housevotes"]
```

6.10.1.7 distances

```
list result_calculation.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
```

6.10.1.8 eps

```
result_calculation.eps
```

6.10.1.9 k

```
result_calculation.k
```

6.10.1.10 kalgoclass

```
dictionary result_calculation.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,
'kmedoids': kmedoidsClustering}
```

6.10.1.11 kalgos

```
list result_calculation.kalgos = ['kmeans', 'kmedians', 'kmedoids']
```

6.10.1.12 m

```
result_calculation.m
```

6.10.1.13 minpts

```
result_calculation.minpts
```

6.10.1.14 results

```
result_calculation.results = Results("./results")
```

6.10.1.15 s

```
result_calculation.s
```

6.10.1.16 seed

```
int result_calculation.seed = 42
```

6.11 results Namespace Reference

Classes

- class Results

  class for easily saving and loading already calculated clustering results

  every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.

6.12 SessionState Namespace Reference

Classes

- class SessionState

Functions

- def get (**kwargs)

  Gets a SessionState object for the current session.

6.12.1 Function Documentation

6.12.1.1 get()

```
def SessionState.get (
            ** kwargs )
```

Gets a SessionState object for the current session.

```
Creates a new object if necessary.

Parameters
----------
**kwargs : any
    Default values you want to add to the session state, if we're creating a
    new one.

Example
-------
>>> session_state = get(user_name='', favorite_color='black')
>>> session_state.user_name
''
>>> session_state.user_name = 'Mary'
>>> session_state.favorite_color
'black'

Since you set user_name above, next time your script runs this will be the
result:
>>> session_state = get(user_name='', favorite_color='black')
>>> session_state.user_name
'Mary'
```

6.13 web_frontend Namespace Reference

Variables

- page_title
- page_icon
- session_state = SessionState.get(indices_data=pd.DataFrame())
- seeded = st.checkbox('Use precalculated results (with random seed for reproduction).', value=True)
- seed = None
- seaplots = st.checkbox('Use interactive altair plots over static seaborn plots.', value=True)
- resulthandler = Results("./code/results")
- col1
- col2
- dataset = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes', 'housevotes'])
- dictionary cluster_dist_desc
- cluster_dist = col1.selectbox('Choose an awesome distance measure', list(cluster_dist_desc.keys()))
- dictionary cluster_algo_class = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering, 'DBSCAN': DBSCANClustering}
- cluster_algo = col2.selectbox('Choose a lovely clustering algorithm', list(cluster_algo_class.keys()))
- dictionary cluster = cluster_algo_class[cluster_algo](cluster_dist, dataset, seed)
- epsilon = col2.slider("Choose a nice value for epsilon", min_value=0.1, max_value=20.0, step=0.1)
- minpts = col2.slider("Choose a minimal number of nearest points", min_value=1, max_value=20, step=1, value=5)
- eps
- clusters
- stuff
- k_value = col2.slider("Choose a nice value for k (number of clusters)", min_value=2, max_value=10, step=1, value=3)
- k
- clustered_data = np.zeros(len(cluster.data))
- list color_palette = ['black'] + sns.color_palette("husl", len(set(clustered_data))-1)
- weights
- perp = col1.slider("Perplexity for t-SNE", 5, 50, 25)
- marking_centroids = np.ones(cluster.data.shape[0])
- cluster_label = alt.Tooltip("c", title="Cluster ID")
- point_label = alt.Tooltip("i", title="Point ID")
- dfclusterdata = pd.DataFrame()
- projected_data_tsne = TSNE(random_state=42, perplexity=perp).fit_transform(cluster.data)
- fig
- ax
- x
- y
- hue
- palette
- legend
- False
- style
- size
- markers
- xaxis = alt.X("xt", axis=alt.Axis(title=None))
- yaxis = alt.Y("yt", axis=alt.Axis(title=None))
- altcolor = alt.Color("c", legend=None, scale=alt.Scale(domain=[0, 1 if max(clustered_data) == 0 else max(clustered_data)], scheme="turbo"))

- tsnealt = alt.Chart(dfclusterdata).mark_circle().encode(x=xaxis, y=yaxis, tooltip=[cluster_label, point_label], color=altcolor).interactive()
- use_container_width
- projected_data_pca = PCA(random_state=42, n_components=2).fit_transform(cluster.data)
- pcaalt = alt.Chart(dfclusterdata).mark_circle().encode(x=xaxis, y=yaxis, tooltip=[cluster_label, point_label], color=altcolor).interactive()
- dataexpander = st.beta_expander("data")
- add_result = col1.button('Add')
- reset_tmp = col2.button('Reset')
- indices_data
- string val = "epsilon="+str(epsilon)+", np="+str(minpts)
- df = session_state.indices_data
- dictionary labels = cluster.labels.tolist()
- predicted = clustered_data.tolist()
- list precalc = [ ]
- list index_eval = ["ARI", "NMI", "Completeness Score", "Homogeneity Score", "Silhouette Score"]
- I1 = Indices(predicted, labels)
- score = I1.index_external(index_eval[i])
- list datasets = [ ]
- list results = [[1, "maximum reference value"]]
- list desc_list = [ ]
- desc = np.array(desc_list)
- stats = np.zeros(len(results))
- ys
- xs = np.arange(len(labels))
- float width = 0.5
- align
- color
- angles = np.linspace(0, 2∗np.pi, len(desc), endpoint=False)
- subplot_kw
- linewidth
- alpha

6.13.1 Variable Documentation

6.13.1.1 add_result

```
web_frontend.add_result = col1.button('Add')
```

6.13.1.2 align

```
web_frontend.align
```

6.13.1.3 alpha

```
web_frontend.alpha
```

6.13.1.4 altcolor

```
web_frontend.altcolor = alt.Color("c", legend=None, scale=alt.Scale(domain=[0, 1 if max(clustered_data)
== 0 else max(clustered_data)], scheme="turbo"))
```

6.13.1.5 angles

```
web_frontend.angles = np.linspace(0, 2*np.pi, len(desc), endpoint=False)
```

6.13.1.6 ax

```
web_frontend.ax
```

6.13.1.7 cluster

```
dictionary web_frontend.cluster = cluster_algo_class[cluster_algo](cluster_dist, dataset,
seed)
```

6.13.1.8 cluster_algo

```
web_frontend.cluster_algo = col2.selectbox('Choose a lovely clustering algorithm', list(cluster←
_algo_class.keys()))
```

6.13.1.9 cluster_algo_class

```
dictionary web_frontend.cluster_algo_class = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering,
'kmedoids': kmedoidsClustering, 'DBSCAN': DBSCANClustering}
```

6.13.1.10 cluster_dist

```
web_frontend.cluster_dist = col1.selectbox('Choose an awesome distance measure', list(cluster←
_dist_desc.keys()))
```

6.13.1.11 cluster_dist_desc

```
dictionary web_frontend.cluster_dist_desc
```

Initial value:

```
=  {'euclidean': 'd(x,y) = \sqrt{\sum_{i=1}^{n}(|x_i-y_i|)^2}',
                    'manhattan': 'd(x,y) = \sum\limits_{i=1}^{n}|x_i - y_i|',
                    'chebyshev': 'd(x,y) = \max(|x_i - y_i|)',
                    'cosine': 'd(x,y) = \\frac{\\arccos(\\frac{\sum_{i=1}^{n} x_i
      y_i}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}})}{\pi}'}
```

6.13.1.12 cluster_label

```
web_frontend.cluster_label = alt.Tooltip("c", title="Cluster ID")
```

6.13.1.13 clustered_data

```
web_frontend.clustered_data = np.zeros(len(cluster.data))
```

6.13.1.14 clusters

```
web_frontend.clusters
```

6.13.1.15 col1

```
web_frontend.col1
```

6.13.1.16 col2

```
web_frontend.col2
```

6.13.1.17 color

```
web_frontend.color
```

6.13.1.18 color_palette

```
list web_frontend.color_palette = ['black'] + sns.color_palette("husl", len(set(clustered_data))-1)
```

6.13.1.19 dataexpander

```
web_frontend.dataexpander = st.beta_expander("data")
```

6.13.1.20 dataset

```
web_frontend.dataset = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes',
'housevotes'])
```

6.13.1.21 datasets

```
list web_frontend.datasets = [ ]
```

6.13.1.22 desc

```
web_frontend.desc = np.array(desc_list)
```

6.13.1.23 desc_list

```
list web_frontend.desc_list = [ ]
```

6.13.1.24 df

```
web_frontend.df = session_state.indices_data
```

6.13.1.25 dfclusterdata

```
web_frontend.dfclusterdata = pd.DataFrame()
```

6.13.1.26 eps

```
web_frontend.eps
```

6.13.1.27 epsilon

```
web_frontend.epsilon = col2.slider("Choose a nice value for epsilon", min_value=0.1, max_↩
value=20.0, step=0.1)
```

6.13.1.28 False

```
web_frontend.False
```

6.13.1.29 fig

```
web_frontend.fig
```

6.13.1.30 hue

```
web_frontend.hue
```

6.13.1.31 I1

```
web_frontend.I1 = Indices(predicted, labels)
```

6.13.1.32 index_eval

```
web_frontend.index_eval = ["ARI", "NMI", "Completeness Score", "Homogeneity Score", "Silhouette
Score"]
```

6.13.1.33 indices_data

```
web_frontend.indices_data
```

6.13.1.34 k

```
web_frontend.k
```

6.13.1.35 k_value

```
web_frontend.k_value = col2.slider("Choose a nice value for k (number of clusters)", min_←↩
value=2, max_value=10, step=1, value=3)
```

6.13.1.36 labels

```
web_frontend.labels = cluster.labels.tolist()
```

6.13.1.37 legend

```
web_frontend.legend
```

6.13.1.38 linewidth

```
web_frontend.linewidth
```

6.13.1.39 markers

```
web_frontend.markers
```

6.13.1.40 marking_centroids

```
web_frontend.marking_centroids = np.ones(cluster.data.shape[0])
```

6.13.1.41 minpts

```
web_frontend.minpts = col2.slider("Choose a minimal number of nearest points", min_value=1,
max_value=20, step=1, value=5)
```

6.13.1.42 page_icon

```
web_frontend.page_icon
```

6.13.1.43 page_title

```
web_frontend.page_title
```

6.13.1.44 palette

```
web_frontend.palette
```

6.13.1.45 pcaalt

```
web_frontend.pcaalt = alt.Chart(dfclusterdata).mark_circle().encode(x=xaxis, y=yaxis, tooltip=[cluster_label,
point_label], color=altcolor).interactive()
```

6.13.1.46 perp

```
web_frontend.perp = col1.slider("Perplexity for t-SNE", 5, 50, 25)
```

6.13.1.47 point_label

```
web_frontend.point_label = alt.Tooltip("i", title="Point ID")
```

6.13.1.48 precalc

```
list web_frontend.precalc = []
```

6.13.1.49 predicted

```
web_frontend.predicted = clustered_data.tolist()
```

6.13.1.50 projected_data_pca

```
web_frontend.projected_data_pca = PCA(random_state=42, n_components=2).fit_transform(cluster.←
data)
```

6.13.1.51 projected_data_tsne

```
web_frontend.projected_data_tsne = TSNE(random_state=42, perplexity=perp).fit_transform(cluster.←
data)
```

6.13.1.52 reset_tmp

```
web_frontend.reset_tmp = col2.button('Reset')
```

6.13.1.53 resulthandler

```
web_frontend.resulthandler = Results("./code/results")
```

6.13.1.54 results

```
list web_frontend.results = [[1, "maximum reference value"]]
```

6.13.1.55 score

```
web_frontend.score = I1.index_external(index_eval[i])
```

6.13.1.56 seaplots

```
web_frontend.seaplots = st.checkbox('Use interactive altair plots over static seaborn plots.',
value=True)
```

6.13.1.57 seed

```
int web_frontend.seed = None
```

6.13.1.58 seeded

```
web_frontend.seeded = st.checkbox('Use precalculated results (with random seed for reproduction).',
value=True)
```

6.13.1.59 session_state

```
web_frontend.session_state = SessionState.get(indices_data=pd.DataFrame())
```

6.13.1.60 size

```
web_frontend.size
```

6.13.1.61 stats

```
web_frontend.stats = np.zeros(len(results))
```

6.13.1.62 stuff

```
web_frontend.stuff
```

6.13.1.63 style

```
web_frontend.style
```

6.13.1.64 subplot_kw

```
web_frontend.subplot_kw
```

6.13.1.65 tsnealt

```
web_frontend.tsnealt = alt.Chart(dfclusterdata).mark_circle().encode(x=xaxis, y=yaxis, tooltip=[cluster_label,
point_label], color=altcolor).interactive()
```

6.13.1.66 use_container_width

```
web_frontend.use_container_width
```

6.13.1.67 val

```
string web_frontend.val = "epsilon="+str(epsilon)+", np="+str(minpts)
```

6.13.1.68 weights

```
web_frontend.weights
```

6.13.1.69 width

```
web_frontend.width = 0.5
```

6.13.1.70 x

```
web_frontend.x
```

6.13.1.71 xaxis

```
web_frontend.xaxis = alt.X("xt", axis=alt.Axis(title=None))
```

6.13.1.72 xs

```
web_frontend.xs = np.arange(len(labels))
```

6.13.1.73 y

```
web_frontend.y
```

6.13.1.74 yaxis

```
web_frontend.yaxis = alt.Y("yt", axis=alt.Axis(title=None))
```

6.13.1.75 ys

```
web_frontend.ys
```

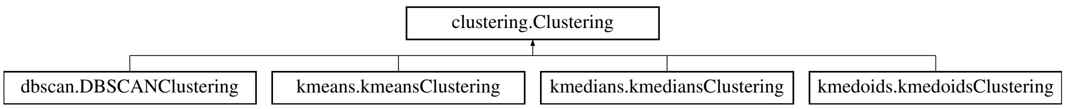
Chapter 7

Class Documentation

7.1 clustering.Clustering Class Reference

Base Class for all subsequent clustering algorithms
implements all functions needed for running the different
cluster algorithms.

Inheritance diagram for clustering.Clustering:



Public Member Functions

- def __init__ (self, metric, dataset, seed=None)

  constructor
- def pyc_metric (self, metric)

  returns a distance metric which is usable by the pyclustering algorithms
- def load_data (self)

  loads in a dataset, standardises it and sets it as self.data attribute
- def house_load (self, path, skip=1)

  loads the housevotes dataset and encodes it using One-Hot-Encoding
  democrats are labeled as 1, republicans as 0
- def cluster (self)

  does nothing in the meta class.

Public Attributes

- metric

    metric name as string or pyclustering distance_metric object
- dataset

    dataset name as string
- data

    data that gets clustered
- labels

    expected cluster values
- seed

    seed for initializer, None if no seed is used
- datadf

    dataset as pandas frame.

7.1.1 Detailed Description

Base Class for all subsequent clustering algorithms
implements all functions needed for running the different
cluster algorithms.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
def clustering.Clustering.__init__ (
              self,
              metric,
              dataset,
              seed = None )
```

constructor

Parameters

metric	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
dataset	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented in kmedoids.kmedoidsClustering, kmedians.kmediansClustering, kmeans.kmeansClustering, and dbscan.DBSCANClustering.

7.1.3 Member Function Documentation

7.1.3.1 cluster()

```
def clustering.Clustering.cluster (
              self )
```

does nothing in the meta class.

needs to be implemented in the inheriting cluster algorithm classes

7.1.3.2 house_load()

```
def clustering.Clustering.house_load (
              self,
              path,
              skip = 1 )
```

loads the housevotes dataset and encodes it using One-Hot-Encoding
democrats are labeled as 1, republicans as 0

Parameters

path	filepath to the dataset
skip	number of lines that get skipped when reading in a file

Returns

One-Hot-Encoded housevotes dataset and labels as array of 1s and 0s

7.1.3.3 load_data()

```
def clustering.Clustering.load_data (
              self )
```

loads in a dataset, standardises it and sets it as self.data attribute

7.1.3.4 pyc_metric()

```
def clustering.Clustering.pyc_metric (
              self,
              metric )
```

returns a distance metric which is usable by the pyclustering algorithms

Parameters

distance	metric string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"

Returns

pyclustering distance_metric object, None when distance is not supported

7.1.4 Member Data Documentation

7.1.4.1 data

`clustering.Clustering.data`

data that gets clustered

7.1.4.2 datadf

`clustering.Clustering.datadf`

dataset as pandas frame.

needed for web frontend later

7.1.4.3 dataset

`clustering.Clustering.dataset`

dataset name as string

7.1.4.4 labels

`clustering.Clustering.labels`

expected cluster values

7.1.4.5 metric

`clustering.Clustering.metric`

metric name as string or pyclustering distance_metric object

7.1.4.6 seed

`clustering.Clustering.seed`

seed for initializer, None if no seed is used

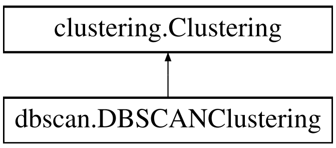
The documentation for this class was generated from the following file:

- clustering.py

7.2 dbscan.DBSCANClustering Class Reference

implements DBSCAN Clustering
uses the scikit-learn DBSCAN implementation

Inheritance diagram for dbscan.DBSCANClustering:



Public Member Functions

- def __init__ (self, metric, dataset, seed=None)

  constructor, seed can be given but is not used.
- def cluster (self, eps, minpts)

  clustering method.
- def package (self, labels)

  rearranges the result to a format similar to the one of the pyclustering algorithms allows for easier access in the streamlit interface

Public Attributes

- metric

  metric name as string
- dataset

  dataset name as string
- data

  data that gets clustered
- labels

  expected cluster values

7.2.1 Detailed Description

implements DBSCAN Clustering
uses the scikit-learn DBSCAN implementation

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def dbscan.DBSCANClustering.__init__ (
              self,
              metric,
              dataset,
              seed = None )
```

constructor, seed can be given but is not used.

its passing is allowed to simplfy the code in the web frontend

Parameters

metric	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
dataset	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from clustering.Clustering.

7.2.3 Member Function Documentation

7.2.3.1 cluster()

```
def dbscan.DBSCANClustering.cluster (
              self,
              eps,
              minpts )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric
params are the same as in the DBSCAN paper

Parameters

eps	Distance for the Eps-Neighbourhood
minPts	Minmal number of points in a cluster

Returns

formatted clustered data

7.2.3.2 package()

```
def dbscan.DBSCANClustering.package (
              self,
              labels )
```

rearranges the result to a format similar to the one of the pyclustering algorithms
allows for easier access in the streamlit interface

Parameters

labels	cluster labels DBSCAN assigns to a point

Returns

clusters as list of lists of indices of points and noise as list of indices of points

7.2.4 Member Data Documentation

7.2.4.1 data

```
dbscan.DBSCANClustering.data
```

data that gets clustered

7.2.4.2 dataset

```
dbscan.DBSCANClustering.dataset
```

dataset name as string

7.2.4.3 labels

```
dbscan.DBSCANClustering.labels
```

expected cluster values

7.2.4.4 metric

`dbscan.DBSCANClustering.metric`

metric name as string

The documentation for this class was generated from the following file:

- dbscan.py

7.3 dbscan_heuristic.DBSCANHeuristic Class Reference

implements the DBSCAN heuristic proposed in the original DBSCAN paper:

Public Member Functions

- def __init__ (self)

  constructor.
- def set_metric (self, metric)

  setter for the metric
- def set_dataset (self, dataset)

  sets and loads the dataset using the DBSCANClustering objects load_data() function
- def kdist (self, k)

  calculates all k-distances for the dataset
- def plot_kdist (self, kdist)

  plots the sorted kdist graph using matplotlib

Public Attributes

- clustering

  DBSCANClustering object for loading datasets.
- metric

  metric as string ("euclidean", "cosine", "manhattan", "chebyshev")
- k

  variable k used in the k-dist calculation

7.3.1 Detailed Description

implements the DBSCAN heuristic proposed in the original DBSCAN paper:

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, page 226–231. AAAI Press, 1996.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
def dbscan_heuristic.DBSCANHeuristic.__init__ (
              self )
```

constructor.

uses a DBSCANClustering object for loading the datasets

7.3.3 Member Function Documentation

7.3.3.1 kdist()

```
def dbscan_heuristic.DBSCANHeuristic.kdist (
              self,
              k )
```

calculates all k-distances for the dataset

Parameters

k	variable for the k-dist. Natural Number.

7.3.3.2 plot_kdist()

```
def dbscan_heuristic.DBSCANHeuristic.plot_kdist (
              self,
              kdist )
```

plots the sorted kdist graph using matplotlib

Parameters

k-dist	list containing the k-distances for every point of the dataset

7.3.3.3 set_dataset()

```
def dbscan_heuristic.DBSCANHeuristic.set_dataset (
              self,
              dataset )
```

sets and loads the dataset using the DBSCANClustering objects load_data() function

Parameters

dataset	string with name of the dataset used ("iris", "wine", "diabetes", "housevotes")

7.3.3.4 set_metric()

```
def dbscan_heuristic.DBSCANHeuristic.set_metric (
              self,
              metric )
```

setter for the metric

Parameters

metric	string containing name of the metric ("euclidean", "cosine", "manhattan", "chebyshev")

7.3.4 Member Data Documentation

7.3.4.1 clustering

```
dbscan_heuristic.DBSCANHeuristic.clustering
```

DBSCANClustering object for loading datasets.

7.3.4.2 k

```
dbscan_heuristic.DBSCANHeuristic.k
```

variable k used in the k-dist calculation

7.3.4.3 metric

```
dbscan_heuristic.DBSCANHeuristic.metric
```

metric as string ("euclidean", "cosine", "manhattan", "chebyshev")

The documentation for this class was generated from the following file:

- dbscan_heuristic.py

7.4 indices.Indices Class Reference

calculates Indices for computed cluster labels uses the scikit library

Public Member Functions

- def __init__ (self, cluster_calc, cluster_label)

  constructor
- def index_external (self, index)

  Function to calculate external index scores
  ARI, AMI, Homogeneity Score and Completeness Score @params index string with name of index used ("ARI", "AMI", "Homogeneity Score", "Completeness Score")
- def index_internal (self, index, points, metric)

  Function to calculate internal index scores

Public Attributes

- cluster_calc

  calculated clustering results
- cluster_label

  expected cluster results

7.4.1 Detailed Description

calculates Indices for computed cluster labels uses the scikit library

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
def indices.Indices.__init__ (
              self,
              cluster_calc,
              cluster_label )
```

constructor

Parameters

cluster_calc	calculated clustering results
cluster_label	expected cluster results

7.4.3 Member Function Documentation

7.4.3.1 index_external()

```
def indices.Indices.index_external (
              self,
              index )
```

Function to calculate external index scores
ARI, AMI, Homogeneity Score and Completeness Score @params index string with name of index used ("ARI", "AMI", "Homogeneity Score", "Completeness Score")

7.4.3.2 index_internal()

```
def indices.Indices.index_internal (
              self,
              index,
              points,
              metric )
```

Function to calculate internal index scores

Parameters

index	
points	
metric	

7.4.4 Member Data Documentation

7.4.4.1 cluster_calc

```
indices.Indices.cluster_calc
```

calculated clustering results

7.4.4.2 cluster_label

`indices.Indices.cluster_label`

expected cluster results

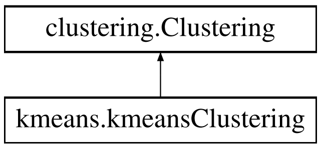
The documentation for this class was generated from the following file:

- indices.py

7.5 kmeans.kmeansClustering Class Reference

Class implementing k-Means Clustering
uses the pyclustering k-means implementation
centers can be initialised using the k++ or the random initialiser.

Inheritance diagram for kmeans.kmeansClustering:



Public Member Functions

- def __init__ (self, metric, dataset, seed=None)

    constructor
- def cluster (self, k, plusplus=True)

    clustering method.

Public Attributes

- metric

    metric name as pyclustering distance_metric object
- dataset

    dataset name as string
- data

    data that gets clustered
- labels

    expected cluster values
- seed

    seed for initializer, None if no seed is used

7.5.1 Detailed Description

Class implementing k-Means Clustering
uses the pyclustering k-means implementation
centers can be initialised using the k++ or the random initialiser.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__()

```
def kmeans.kmeansClustering.__init__ (
              self,
              metric,
              dataset,
              seed = None )
```

constructor

Parameters

metric	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
dataset	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from clustering.Clustering.

7.5.3 Member Function Documentation

7.5.3.1 cluster()

```
def kmeans.kmeansClustering.cluster (
              self,
              k,
              plusplus = True )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric

Parameters

k	number of clusters that are generated
plusplus	will use k++ initialiser if true

Returns

clusters as list of lists of indices of points and final cluster centers

7.5.4 Member Data Documentation

7.5.4.1 data

`kmeans.kmeansClustering.data`

data that gets clustered

7.5.4.2 dataset

`kmeans.kmeansClustering.dataset`

dataset name as string

7.5.4.3 labels

`kmeans.kmeansClustering.labels`

expected cluster values

7.5.4.4 metric

`kmeans.kmeansClustering.metric`

metric name as pyclustering distance_metric object

7.5.4.5 seed

`kmeans.kmeansClustering.seed`

seed for initializer, None if no seed is used

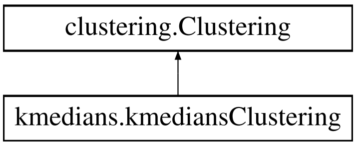
The documentation for this class was generated from the following file:

- kmeans.py

7.6 kmedians.kmediansClustering Class Reference

implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser

Inheritance diagram for kmedians.kmediansClustering:



Public Member Functions

- def __init__ (self, metric, dataset, seed=None)

  constructor
- def cluster (self, k, plusplus=True)

  clustering method.

Public Attributes

- metric

  metric name as pyclustering distance_metric object
- dataset

  dataset name as string
- data

  data that gets clustered
- labels

  expected cluster values
- seed

  seed for initializer, None if no seed is used

7.6.1 Detailed Description

implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__()

```
def kmedians.kmediansClustering.__init__ (
              self,
              metric,
              dataset,
              seed = None )
```

constructor

Parameters

metric	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
dataset	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from clustering.Clustering.

7.6.3 Member Function Documentation

7.6.3.1 cluster()

```
def kmedians.kmediansClustering.cluster (
              self,
              k,
              plusplus = True )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric

Parameters

k	number of clusters that are generated

Returns

clusters as list of lists of indices of points and final cluster medians

7.6.4 Member Data Documentation

7.6.4.1 data

```
kmedians.kmediansClustering.data
```

data that gets clustered

7.6.4.2 dataset

```
kmedians.kmediansClustering.dataset
```

dataset name as string

7.6.4.3 labels

`kmedians.kmediansClustering.labels`

expected cluster values

7.6.4.4 metric

`kmedians.kmediansClustering.metric`

metric name as pyclustering distance_metric object

7.6.4.5 seed

`kmedians.kmediansClustering.seed`

seed for initializer, None if no seed is used

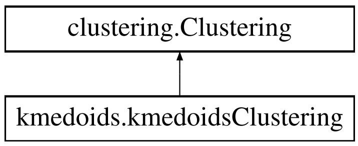
The documentation for this class was generated from the following file:

- kmedians.py

7.7 kmedoids.kmedoidsClustering Class Reference

implements k-Medians Clustering
uses the scikit-learn-extra k-medoids implementation
centers are set using the k++ initialiser if not set differently

Inheritance diagram for kmedoids.kmedoidsClustering:



Public Member Functions

- def __init__ (self, metric, dataset, seed=None)

  constructor
- def cluster (self, k, init="k-medoids++")

  clustering method.
- def package (self, labels)

  rearranges the result to a format similar to the one of the pyclustering algorithms allows for easier access in the streamlit interface

Public Attributes

- metric

    metric name as string
- dataset

    dataset name as string
- data

    data that gets clustered
- labels

    expected cluster values
- seed

    seed for initializer, None if no seed is used

7.7.1 Detailed Description

implements k-Medians Clustering
uses the scikit-learn-extra k-medoids implementation
centers are set using the k++ initialiser if not set differently

7.7.2 Constructor & Destructor Documentation

7.7.2.1 __init__()

```
def kmedoids.kmedoidsClustering.__init__ (
              self,
              metric,
              dataset,
              seed = None )
```

constructor

Parameters

metric	metric description as string. allowed: "euclidean", "manhattan", "chebyshev", "cosine"
dataset	dataset given as string. allowed: "diabetes", "iris", "wine", "housevotes"

Reimplemented from clustering.Clustering.

7.7.3 Member Function Documentation

7.7.3.1 cluster()

```
def kmedoids.kmedoidsClustering.cluster (
              self,
              k,
              init = "k-medoids++" )
```

clustering method.

Will execute clustering on the data saved in self.data with the metric given in self.metric

Parameters

k	number of clusters that are generated
init	initialisation parameter. Default: "k-medoids++"

Returns

clusters as list of lists of indices of points, final cluster centers

7.7.3.2 package()

```
def kmedoids.kmedoidsClustering.package (
              self,
              labels )
```

rearranges the result to a format similar to the one of the pyclustering algorithms allows for easier access in the streamlit interface

Parameters

labels	labels returned from the KMedoids algorithm

Returns

clusters formated similarly to the pyclustering algorithms

7.7.4 Member Data Documentation

7.7.4.1 data

```
kmedoids.kmedoidsClustering.data
```

data that gets clustered

7.7.4.2 dataset

`kmedoids.kmedoidsClustering.dataset`

dataset name as string

7.7.4.3 labels

`kmedoids.kmedoidsClustering.labels`

expected cluster values

7.7.4.4 metric

`kmedoids.kmedoidsClustering.metric`

metric name as string

7.7.4.5 seed

`kmedoids.kmedoidsClustering.seed`

seed for initializer, None if no seed is used

The documentation for this class was generated from the following file:

- kmedoids.py

7.8 results.Results Class Reference

class for easily saving and loading already calculated clustering results

every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.

Public Member Functions

- def __init__ (self, parentpath)

  constructor.
- def get_path (self, dataset, algorithm, metric, **kwargs)

  builds and returns the filepath to the json file fitting the given parameters
- def set_exists (self, dataset, algorithm, metric, **kwargs)

  checks if a file for a result defined by the parameters exists
- def load_set (self, dataset, algorithm, metric, **kwargs)

  loads results fitting the given parameters from a json file
- def save_set (self, dataset, algorithm, metric, clusters, centers, **kwargs)

  saves cluster results in a json file

Public Attributes

- parent

7.8.1 Detailed Description

class for easily saving and loading already calculated clustering results

every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.

Clustering results are saved as json files in their respective folders.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

```
def results.Results.__init__ (
            self,
            parentpath )
```

constructor.

needs the filepath to the parent directory where the json files are suposed to be saved

Parameters

parentpath	filepath to the parent directory

7.8.3 Member Function Documentation

7.8.3.1 get_path()

```
def results.Results.get_path (
              self,
              dataset,
              algorithm,
              metric,
             ** kwargs )
```

builds and returns the filepath to the json file fitting the given parameters

Parameters

dataset	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
algorithm	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
metric	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
∗∗*kwargs*	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

Returns

filepath to the correct json file

7.8.3.2 load_set()

```
def results.Results.load_set (
              self,
              dataset,
              algorithm,
              metric,
             ** kwargs )
```

loads results fitting the given parameters from a json file

Parameters

dataset	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
algorithm	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
metric	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
∗∗*kwargs*	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

Returns

loaded clustering results (clusters and centers)

7.8.3.3 save_set()

```
def results.Results.save_set (
              self,
              dataset,
              algorithm,
              metric,
              clusters,
              centers,
             ** kwargs )
```

saves cluster results in a json file

Parameters

dataset	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
algorithm	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
metric	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
∗∗*kwargs*	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

7.8.3.4 set_exists()

```
def results.Results.set_exists (
              self,
              dataset,
              algorithm,
              metric,
             ** kwargs )
```

checks if a file for a result defined by the parameters exists

Parameters

dataset	string with the name of the dataset ("iris", "wine", "diabetes", "DBSCAN")
algorithm	string with the name of the algorithm ("kmeans", "kmedians", "kmedoids", "DBSCAN")
metric	string with the name of the distance measure ("euclidean", "cosine", "chebyshev", "manhattan")
∗∗*kwargs*	algorithm specific parameters. Needs to be either "k" or "minpts" and "eps"

Returns

True if file exists, False if not

7.8.4 Member Data Documentation

7.8.4.1 parent

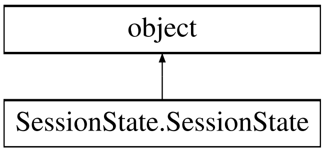
```
results.Results.parent
```

The documentation for this class was generated from the following file:

- results.py

7.9 SessionState.SessionState Class Reference

Inheritance diagram for SessionState.SessionState:



Public Member Functions

- def __init__ (self, **kwargs)
  A new SessionState object.

7.9.1 Constructor & Destructor Documentation

7.9.1.1 __init__()

```
def SessionState.SessionState.__init__ (
              self,
             ** kwargs )
```

A new SessionState object.

```
Parameters
----------
**kwargs : any
    Default values for the session state.

Example
-------
>>> session_state = SessionState(user_name='', favorite_color='black')
>>> session_state.user_name = 'Mary'
''
>>> session_state.favorite_color
'black'
```

The documentation for this class was generated from the following file:

- SessionState.py

Chapter 8

File Documentation

8.1 clustering.py File Reference

contains the clustering base class

Classes

- class clustering.Clustering

    Base Class for all subsequent clustering algorithms implements all functions needed for running the different cluster algorithms.

Namespaces

- clustering

8.1.1 Detailed Description

contains the clustering base class

8.2 comparison_plots.py File Reference

Namespaces

- comparison_plots

Variables

- list comparison_plots.kalgos = ['kmeans', 'kmedians', 'kmedoids']
- dictionary comparison_plots.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering}
- list comparison_plots.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list comparison_plots.datasets = ["iris", "wine", "diabetes", "housevotes"]
- list comparison_plots.index_ext_eval = ["ARI", "NMI", "Completeness Score", "Homogeneity Score"]
- list comparison_plots.index_int_eval = ["Silhouette Score"]
- list comparison_plots.num_of_classes = [3,3,2,2]
- int comparison_plots.seed = 42
- comparison_plots.results = Results("./results")
- comparison_plots.all_kalgos = np.zeros((3,4, 9,len(index_ext_eval)))
- comparison_plots.all_kalgos_int = pd.DataFrame(columns=['k', 'Distance (Silhouette Score)', 'Distance (Clustering)', 'sil_score', 'kalgo'])
- comparison_plots.all_dist = np.zeros((4, 9,len(index_ext_eval)))
- comparison_plots.all_k = np.zeros((9,len(index_ext_eval)))
- comparison_plots.clusters
- comparison_plots.stuff
- comparison_plots.s
- comparison_plots.c
- comparison_plots.d
- comparison_plots.k
- dictionary comparison_plots.cluster = kalgoclass[c](d, s, seed)
- comparison_plots.clustered_data = np.zeros(len(cluster.data))
- dictionary comparison_plots.labels = cluster.labels.tolist()
- comparison_plots.predicted = clustered_data.tolist()
- comparison_plots.I1 = Indices(predicted, labels)
- comparison_plots.index_scores = np.zeros_like(index_ext_eval, dtype=float)
- comparison_plots.index_score = I1.index_internal(index=index_int_eval[0], points=cluster.data.tolist(), metric=di)
- comparison_plots.index
- comparison_plots.fig = plt.figure(figsize=(15, 10))
- comparison_plots.bbox_to_anchor
- comparison_plots.loc
- comparison_plots.borderaxespad
- comparison_plots.ax
- comparison_plots.figsize
- comparison_plots.data
- comparison_plots.x
- comparison_plots.y
- comparison_plots.hue
- comparison_plots.style
- comparison_plots.legend
- comparison_plots.all_kalgos_df = pd.DataFrame(all_kalgos[:,:,num_of_classes[isx]-2,i], columns=distances, index=kalgos)
- comparison_plots.kind
- comparison_plots.title

8.3 dbscan.py File Reference

implementation of the DBSCAN algorithm.

Classes

- class dbscan.DBSCANClustering

    implements DBSCAN Clustering
    uses the scikit-learn DBSCAN implementation

Namespaces

- dbscan

8.3.1 Detailed Description

implementation of the DBSCAN algorithm.

8.4 dbscan_heuristic.py File Reference

implementation of DBSCAN parameter estimation heuristic

Classes

- class dbscan_heuristic.DBSCANHeuristic

    implements the DBSCAN heuristic proposed in the original DBSCAN paper:

Namespaces

- dbscan_heuristic

8.4.1 Detailed Description

implementation of DBSCAN parameter estimation heuristic

8.5 heuristic_web.py File Reference

webfrontend for the DBSCAN heuristic impslemented using streamlit.

Namespaces

- heuristic_web

Variables

- heuristic_web.page_title
- heuristic_web.page_icon
- heuristic_web.key
- heuristic_web.col1
- heuristic_web.col2
- heuristic_web.dataset = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes', 'housevotes'])
- dictionary heuristic_web.cluster_dist_desc
- heuristic_web.cluster_dist = col1.selectbox('Choose an awesome distance measure',list(cluster_dist_desc.↩ keys()))
- heuristic_web.k = col2.slider("Choose a nice value for k", min_value=1, max_value=20, step=1, value=4)
- heuristic_web.submit_button = st.form_submit_button(label='Calculate kdist Graph')
- heuristic_web.heu = DBSCANHeuristic()
- heuristic_web.kdist = heu.kdist(k)
- heuristic_web.reverse
- heuristic_web.df
- heuristic_web.nearest = alt.selection(type='single', nearest=True, on='mouseover', fields=['points'], empty='none')
- heuristic_web.yaxis = alt.Y("dist", axis=alt.Axis(title=f"{k}-dist"))
- heuristic_web.line
- heuristic_web.selectors = alt.Chart(df).mark_point().encode(x='points', opacity=alt.value(0)).add_↩ selection(nearest)
- heuristic_web.points = line.mark_point(color="red").encode(opacity=alt.condition(nearest, alt.value(1), alt.↩ value(0)))
- heuristic_web.text = line.mark_text(align='left', dx=5, dy=-5, color="red").encode(text=alt.condition(nearest, "label:N", alt.value(' '))).transform_calculate(label=f'"distance: " + format(datum.dist, ".2f")')
- heuristic_web.textp = line.mark_text(align='left', dx=5, dy=-15, color="red").encode(text=alt.condition(nearest, "label:N", alt.value(' '))).transform_calculate(label=f'format((1 - (datum.points-1) / {len(kdist)}) ∗ 100, ".2f") + "% core points"')
- heuristic_web.rules = alt.Chart(df).mark_rule(color='gray').encode(x="points").transform_filter(nearest)
- heuristic_web.use_container_width

8.5.1 Detailed Description

webfrontend for the DBSCAN heuristic impslemented using streamlit.

uses altair charts for displaying the chart

8.6 indices.py File Reference

Evaluation Modul to compare clustering results.

Classes

- class indices.Indices

    calculates Indices for computed cluster labels uses the scikit library

Namespaces

- indices

8.6.1 Detailed Description

Evaluation Modul to compare clustering results.

8.7 kmeans.py File Reference

implementation of the k-means algorithm.

Classes

- class kmeans.kmeansClustering

  Class implementing k-Means Clustering
  uses the pyclustering k-means implementation
  centers can be initialised using the k++ or the random initialiser.

Namespaces

- kmeans

8.7.1 Detailed Description

implementation of the k-means algorithm.

8.8 kmedians.py File Reference

implementation of the k-medians algorithm.

Classes

- class kmedians.kmediansClustering

  implements k-Medians Clustering uses the pyclustering k-medians implementation centers are initialised using the random initialiser

Namespaces

- kmedians

8.8.1 Detailed Description

implementation of the k-medians algorithm.

8.9 kmedoids.py File Reference

implementation of the k-medoids algorithm.

Classes

- class kmedoids.kmedoidsClustering

    implements k-Medians Clustering
    uses the scikit-learn-extra k-medoids implementation
    centers are set using the k++ initialiser if not set differently

Namespaces

- kmedoids

8.9.1 Detailed Description

implementation of the k-medoids algorithm.

8.10 result_calculation.py File Reference

Namespaces

- result_calculation

Variables

- list result_calculation.kalgos = ['kmeans', 'kmedians', 'kmedoids']
- dictionary result_calculation.kalgoclass = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering}
- list result_calculation.distances = ["euclidean", "manhattan", "chebyshev", "cosine"]
- list result_calculation.datasets = ["iris", "wine", "diabetes", "housevotes"]
- int result_calculation.seed = 42
- result_calculation.results = Results("./results")
- result_calculation.s
- result_calculation.c
- result_calculation.d
- result_calculation.k
- dictionary result_calculation.alg = kalgoclass[c](d, s, seed)
- result_calculation.clusters
- result_calculation.centers
- result_calculation.minpts
- result_calculation.m
- result_calculation.eps

8.11 results.py File Reference

handler for saving and loading results.

Classes

- class results.Results

  class for easily saving and loading already calculated clustering results

  every dataset has a folder containing subfolders for every clustering algorithm containing more subfolders for every distance measure.

Namespaces

- results

8.11.1 Detailed Description

handler for saving and loading results.

8.12 SessionState.py File Reference

taken from `https://gist.github.com/tvst/036da038ab3e999a64497f42de966a92`.

Classes

- class SessionState.SessionState

Namespaces

- SessionState

Functions

- def SessionState.get (**kwargs)

  Gets a SessionState object for the current session.

8.12.1 Detailed Description

taken from `https://gist.github.com/tvst/036da038ab3e999a64497f42de966a92`.

Please refer to this gist and its original author

Hack to add per-session state to Streamlit.

8.12.1.1 Usage

import SessionState
session_state = SessionState.get(user_name='', favorite_color='black') session←
_state.user_name

''

session_state.user_name = 'Mary' session_state.favorite_color

'black'

Since you set user_name above, next time your script runs this will be the result:

session_state = get(user_name='', favorite_color='black') session_state.user_←
name

'Mary'

8.13 web_frontend.py File Reference

webfrontend for project.

Namespaces

- web_frontend

Variables

- web_frontend.page_title
- web_frontend.page_icon
- web_frontend.session_state = SessionState.get(indices_data=pd.DataFrame())
- web_frontend.seeded = st.checkbox('Use precalculated results (with random seed for reproduction).', value=True)
- web_frontend.seed = None
- web_frontend.seaplots = st.checkbox('Use interactive altair plots over static seaborn plots.', value=True)
- web_frontend.resulthandler = Results("./code/results")
- web_frontend.col1
- web_frontend.col2
- web_frontend.dataset = col1.selectbox('Choose a beautiful dataset',['iris', 'wine', 'diabetes', 'housevotes'])
- dictionary web_frontend.cluster_dist_desc
- web_frontend.cluster_dist = col1.selectbox('Choose an awesome distance measure', list(cluster_dist_desc.←
keys()))
- dictionary web_frontend.cluster_algo_class = {'kmeans': kmeansClustering, 'kmedians': kmediansClustering, 'kmedoids': kmedoidsClustering, 'DBSCAN': DBSCANClustering}
- web_frontend.cluster_algo = col2.selectbox('Choose a lovely clustering algorithm', list(cluster_algo_class.←
keys()))
- dictionary web_frontend.cluster = cluster_algo_class[cluster_algo](cluster_dist, dataset, seed)

- web_frontend.epsilon = col2.slider("Choose a nice value for epsilon", min_value=0.1, max_value=20.↩ 0, step=0.1)
- web_frontend.minpts = col2.slider("Choose a minimal number of nearest points", min_value=1, max_↩ value=20, step=1, value=5)
- web_frontend.eps
- web_frontend.clusters
- web_frontend.stuff
- web_frontend.k_value = col2.slider("Choose a nice value for k (number of clusters)", min_value=2, max_↩ value=10, step=1, value=3)
- web_frontend.k
- web_frontend.clustered_data = np.zeros(len(cluster.data))
- list web_frontend.color_palette = ['black'] + sns.color_palette("husl", len(set(clustered_data))-1)
- web_frontend.weights
- web_frontend.perp = col1.slider("Perplexity for t-SNE", 5, 50, 25)
- web_frontend.marking_centroids = np.ones(cluster.data.shape[0])
- web_frontend.cluster_label = alt.Tooltip("c", title="Cluster ID")
- web_frontend.point_label = alt.Tooltip("i", title="Point ID")
- web_frontend.dfclusterdata = pd.DataFrame()
- web_frontend.projected_data_tsne = TSNE(random_state=42, perplexity=perp).fit_transform(cluster.data)
- web_frontend.fig
- web_frontend.ax
- web_frontend.x
- web_frontend.y
- web_frontend.hue
- web_frontend.palette
- web_frontend.legend
- web_frontend.False
- web_frontend.style
- web_frontend.size
- web_frontend.markers
- web_frontend.xaxis = alt.X("xt", axis=alt.Axis(title=None))
- web_frontend.yaxis = alt.Y("yt", axis=alt.Axis(title=None))
- web_frontend.altcolor = alt.Color("c", legend=None, scale=alt.Scale(domain=[0, 1 if max(clustered_data) == 0 else max(clustered_data)], scheme="turbo"))
- web_frontend.tsnealt = alt.Chart(dfclusterdata).mark_circle().encode(x=xaxis, y=yaxis, tooltip=[cluster_label, point_label], color=altcolor).interactive()
- web_frontend.use_container_width
- web_frontend.projected_data_pca = PCA(random_state=42, n_components=2).fit_transform(cluster.data)
- web_frontend.pcaalt = alt.Chart(dfclusterdata).mark_circle().encode(x=xaxis, y=yaxis, tooltip=[cluster_label, point_label], color=altcolor).interactive()
- web_frontend.dataexpander = st.beta_expander("data")
- web_frontend.add_result = col1.button('Add')
- web_frontend.reset_tmp = col2.button('Reset')
- web_frontend.indices_data
- string web_frontend.val = "epsilon="+str(epsilon)+", np="+str(minpts)
- web_frontend.df = session_state.indices_data
- dictionary web_frontend.labels = cluster.labels.tolist()
- web_frontend.predicted = clustered_data.tolist()
- list web_frontend.precalc = [ ]
- list web_frontend.index_eval = ["ARI", "NMI", "Completeness Score", "Homogeneity Score", "Silhouette Score"]
- web_frontend.I1 = Indices(predicted, labels)
- web_frontend.score = I1.index_external(index_eval[i])
- list web_frontend.datasets = [ ]
- list web_frontend.results = [[1, "maximum reference value"]]

- list web_frontend.desc_list = [ ]
- web_frontend.desc = np.array(desc_list)
- web_frontend.stats = np.zeros(len(results))
- web_frontend.ys
- web_frontend.xs = np.arange(len(labels))
- float web_frontend.width = 0.5
- web_frontend.align
- web_frontend.color
- web_frontend.angles = np.linspace(0, 2*np.pi, len(desc), endpoint=False)
- web_frontend.subplot_kw
- web_frontend.linewidth
- web_frontend.alpha

8.13.1 Detailed Description

webfrontend for project.

implemented using streamlit. displays parameter selection, clustering results, data and the evaluation module. Charts can be displayed using seaborn or altair

8.14 /home/nordegraf/Uni/8.Semester/DataScienceI/datascience1_group42/README.md File Reference

Index