kprototypes

Release 0.1.2

Johan Berdat

Jun 09, 2020

CONTENTS

CHAPTER

ONE

DEVELOPER INTERFACE

This part of the documentation covers the public interface of k-prototypes.

1.1 Main Interface

The main entry point follows similar conventions to Scikit-Learn, but it is not fully compatible (see design choices).

class kprototypes.**KPrototypes**(*n_clusters=8*, *initialization=None*, *numerical_distance=None*, *categorical_distance=None*, *gamma=None*, *n_iterations=100*, *random_state=None*, *verbose=0*)

K-Prototypes clustering.

The k-prototypes algorithm, as described in "Clustering large data sets with mixed numeric and categorical values" by Huang (1997), is an extension of k-means for mixed data.

This wrapper loosely follows Scikit-Learn conventions for clustering estimators, as it provide the usual `fit` and `predict` methods. However, the signature is different, as it expects numerical and categorical data to be provided in separated arrays.

See also:

fit(), *predict()*

initialization: callable
Centroid initialization function.

numerical_distance: callable
Distance function used for numerical features.

categorical_distance: callable
Distance function used for categorical features.

gamma: float32
Categorical distance weight.

n_clusters: int32
Number of clusters.

n_iterations: int32
Maximum number of iterations.

verbose: bool
Verbosity level (0 for no output).

true_gamma: float32
Categorical distance weight inferred from data, if gamma was not specified.

numerical_centroids: float32, n_clusters x n_numerical_features
Numerical centroid array.

categorical_centroids: int32, n_clusters x n_categorical_features
Categorical centroid array.

cost: float32
Loss after last training iteration.

fit (*numerical_values*, *categorical_values*)
Fit centroids.

Parameters

- **numerical_values** (*float32, n_samples x n_numerical_features*) – Numerical feature array.
- **categorical_values** (*int32, n_samples x n_categorical_features*) – Categorical feature array.

Returns **self**

Return type object

fit_predict (*numerical_values*, *categorical_values*)
Fit centroids and assign points to closest clusters.

Parameters

- **numerical_values** (*float32, n_samples x n_numerical_features*) – Numerical feature array.
- **categorical_values** (*int32, n_samples x n_categorical_features*) – Categorical feature array.

Returns **clustership** – Closest clusers.

Return type int32, n_samples

predict (*numerical_values*, *categorical_values*)
Assign points to closest clusters.

Parameters

- **numerical_values** (*float32, n_samples x n_numerical_features*) – Numerical feature array.
- **categorical_values** (*int32, n_samples x n_categorical_features*) – Categorical feature array.

Returns **clustership** – Closest clusers.

Return type int32, n_samples

1.2 Distance Measure

Common distance functions are provided, but any callable can be used, as long as broadcasting is properly done.

`kprototypes.check_distance`(*distance*)

Resolve distance function.

If `distance` is a string, only `"euclidean"`, `"manhattan"` and `"matching"` are accepted. If it is a callable, then is is return as-is. If it is `None`, it defaults to `"euclidean"`.

See also:

`euclidean_distance()`, *`manhattan_distance()`*, *`matching_distance()`*

`kprototypes.euclidean_distance`(*a*, *b*)

Squared euclidean distance.

This is the sum of squared differences for each feature pair, also known as squared L_2 norm.

Example

```
>>> a = np.array([0.0, 0.0, 0.0])
>>> b = np.array([1.0, 2.0, 3.0])
>>> euclidean_distance(a, b)
14.0
```

`kprototypes.manhattan_distance`(*a*, *b*)

Manhattan distance.

This is the sum of absolute differences for each feature pair, also known as L_1 norm.

Example

```
>>> a = np.array([0.0, 0.0, 0.0])
>>> b = np.array([1.0, 2.0, 3.0])
>>> manhattan_distance(a, b)
6.0
```

`kprototypes.matching_distance`(*a*, *b*)

Matching distance.

Each feature pair that does not match adds one to the distance. This distance measure is often used for categorical features.

Example

```
>>> a = np.array([1, 2, 3, 4, 5])
>>> b = np.array([1, 8, 3, 4, 0])
>>> matching_distance(a, b)
2
```

1.3 Initialization

Simple initialization functions are provided, but any callable can be used. Explicit centroids can also be provided, either to resume training or to use an external initialization process.

`kprototypes.check_initialization`(*initialization*)

Resolve initialization function.

If `distance` is a string, only `"random"` and `"frequency"` are accepted. If it is a callable, then is is return as-is. If it is `None`, it defaults to `"random"`.

Directly specifying centroids as a tuple of arrays is also accepted.

Returns **function** – Centroid initialization function.

Return type callable

See also:

random_initialization(), *frequency_initialization()*

`kprototypes.random_initialization`(*numerical_values*, *categorical_values*, *n_clusters*, *numerical_distance*, *categorical_distance*, *gamma*, *random_state*, *verbose*)

Random initialization.

Choose random points as cluster centroids.

Used in "Clustering large data sets with mixed numeric and categorical values" by Huang (1997), the original k-prototypes definition.

Returns

- **numerical_centroids** (*float32, n_clusters x n_numerical_features*) – Numerical centroid array.
- **categorical_centroids** (*int32, n_clusters x n_categorical_features*) – Categorical centroid array.

`kprototypes.frequency_initialization`(*numerical_values*, *categorical_values*, *n_clusters*, *numerical_distance*, *categorical_distance*, *gamma*, *random_state*, *verbose*)

Frequency-based initialization.

Choose centroids from points, based on probability distributions of each feature. The first centroid is selected at highest density point. Then, the remaining centroids are selected to be both far from current centroids and at dense locations.

This is an extension for mixed values of "A new initialization method for categorical data clustering" by Cao et al. (2009).

Returns

- **numerical_centroids** (*float32, n_clusters x n_numerical_features*) – Numerical centroid array.
- **categorical_centroids** (*int32, n_clusters x n_categorical_features*) – Categorical centroid array.

1.4 Data Preprocessing

As k-prototypes only accepts floating-point values for numerical data and integer values for categorical data, any other data type must be properly converted beforehand.

`sklearn.preprocessing.StandardScaler` is the equivalent for numerical values.

class `kprototypes.`**`CategoricalTransformer`**(***kwargs*)

Encode categorical values as integers.

Each column has its own vocabulary. Values are mapped from 0 to N - 1, where N is the size of the vocabulary.

Parameters

- **min_count** (*int, optional*) – Ignore values that appears less than a given number of times. Unknown values must be enabled as well.
- **allow_unknown** (*bool, optional*) – Add an additional value for unexpected or unknown values.
- **nan_as_unknown** (*bool, optional*) – Treat NaN as unknown, instead of allocating a dedicated index.

fit(*values*)

Build index.

fit_transform(*values*)

Build index and transform values.

inverse_transform(*indices*)

Convert indices back to values.

transform(*values*)

Transform values.

1.5 Low-Level Optimization Methods

At its core, k-prototypes is defined by these two methods.

`kprototypes.`**`fit`**(*numerical_values*, *categorical_values*, *numerical_centroids*, *categorical_centroids*, *numerical_distance*, *categorical_distance*, *gamma*, *n_iterations*, *random_state*, *verbose*)

Fit centroids.

This implementation follows the standard k-means algorithm, also referred to as Lloyd's algorithm. The optimization proceeds by alternating between two steps:

1. assignment step, where each sample is assigned to the closest centroid;
2. update step, where centroids are recomputed based on the assignment.

This approach differs from the original paper, where centroids are updated after each individual assignment.

Parameters

- **numerical_values** (*float32, n_samples x n_numerical_features*) – Numerical feature array.
- **categorical_values** (*int32, n_samples x n_categorical_features*) – Categorical feature array.
- **numerical_centroids** (*float32, n_clusters x n_numerical_features*) – Numerical centroid array.

- **categorical_centroids** (*int32, n_clusters x n_categorical_features*) – Categorical centroid array.
- **numerical_distance** (*callable*) – Distance function used for numerical features.
- **categorical_distance** (*callable*) – Distance function used for categorical features.
- **gamma** (*float32*) – Categorical distance weight.
- **n_iterations** (*int32*) – Maximum number of iterations.
- **random_state** (*numpy.random.RandomState*) – Random state used to initialize centroids.
- **verbose** (*int32*) – Verbosity level (0 for no output).

Returns

- **clustership** (*int32, n_samples*) – Closest clusers.
- **cost** (*float32*) – Loss after last iteration.

kprototypes.**predict**(*numerical_values*, *categorical_values*, *numerical_centroids*, *categorical_centroids*, *numerical_distance*, *categorical_distance*, *gamma*, *return_cost=False*)

Assign points to closest clusters.

Parameters

- **numerical_values** (*float32, n_samples x n_numerical_features*) – Numerical feature array.
- **categorical_values** (*int32, n_samples x n_categorical_features*) – Categorical feature array.
- **numerical_centroids** (*float32, n_clusters x n_numerical_features*) – Numerical centroid array.
- **categorical_centroids** (*int32, n_clusters x n_categorical_features*) – Categorical centroid array.
- **numerical_distance** (*callable*) – Distance function used for numerical features.
- **categorical_distance** (*callable*) – Distance function used for categorical features.
- **gamma** (*float32*) – Categorical distance weight.
- **return_cost** (*bool, optional*) – Whether to return cost.

Returns

- **clustership** (*int32, n_samples*) – Closest clusers.
- **cost** (*float32*) – Loss after last iteration, if `return_cost` is true.

PYTHON MODULE INDEX

k

INDEX