KDE Dataset Generation

Version 1.0.1

Purpose

The purpose of this project is to generate a new dataset from an original dataset. The new dataset can be smaller and/or bigger than the original one. Also, the new dataset must be equivalent to the original one, which means that the original information must be kept in the new one.

We assume that the dataset "essence" is contained in the attributes and categories distributions, so we will estimate `Probability Density Functions` (`PDFs`) from the original dataset using `Kernel Density Estimation` (`KDE`). The new dataset will be built from the original `PDFs` .

Test Dataset Generation

We will build a custom dataset to simplify the analysis, development and testing. This dataset will be considered as the original.

We need a dataset with only one attribute and two categories. Also, the attribute distributions grouped by each category must show a little overlap between them.

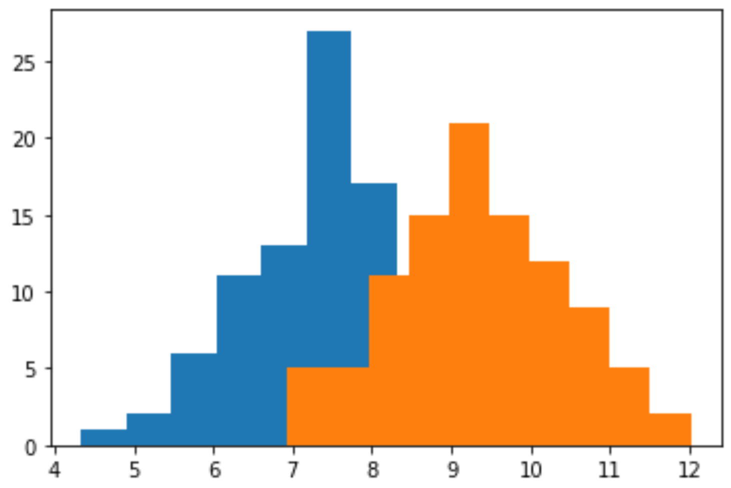
We will generate the custom dataset with 200 instances (100 instances per category).

Finally, we will plot the dataset distribution per category.

In [1…

```
from kde_dataset_generator import dataset

# Dataset params
size = 200
n_categories = 2
seed = 8

# Generate df
df = dataset.generate_univariate_dataset(size, n_categories, s

# Plot df
print("Figure 1: Dataset distribution by category")
dataset.plot_univariate(df)
```

```
Figure 1: Dataset distribution by category
```



Looking at the chart we can conclude that the custom dataset was properly generated. We can even see the little overlap between distributions.

KDEs

We need to get the `KDEs` from the dataset distributions to estimate `PDFs` .

We will always generate as many `KDEs` as categories are. We will take instances associated to a specific category and build the `KDE` from them. Each instance will keep all the attributes, that means `KDEs` are multidimensional (one dimension per attribute).

For this case, we will calculate two `KDEs` (1 for category `0` , 1 for category `1`).

In [2…

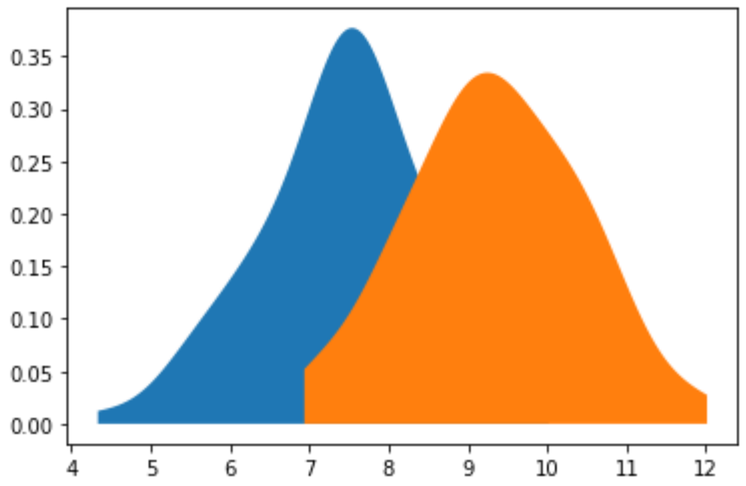
```python
from kde_dataset_generator import kde

# Caculate KDEs
kdes, ranges = kde.calculate_kdes(df)
```

To make sure `PDFs` estimation are right, we will plot the dataset distributions using the previous calculated `KDEs` .

In [3…

```python
print("Figure 2: Dataset distributions by category using KDEs"
kde.plot_univariate(df, kdes)
```

```
Figure 2: Dataset distributions by category using KDEs
```



The distributions represented in the charts are the same, so we conclude that `KDEs` are right.

First method

To build the new dataset this method will generate instances randomly and label the new instances using previous `KDEs` .

The possible attributes values of each instance must be contained in intervals (one interval per attribute) which have been calculated from the original values of the attributes.

Once a new instance is generated we will estimate its density using each `KDE` . We will set the label associated to the `KDE` that gives biggest density. We will repeat this process until reach the amount of instances per category which have been defined previously. Instances associate to a category which is full will be discard.

The minimun amount of instances per category is a parameter which depends of the dataset. To optimize this parameter we will need an another study. We will test different values until getting a value that allows to describe correctly each category distribution. We will check it comparing the charts.

We are going to define attributes intervals.

In [4…
```
# Get attributes intervals
attributes_intervals = dataset.attributes_intervals(df)
```

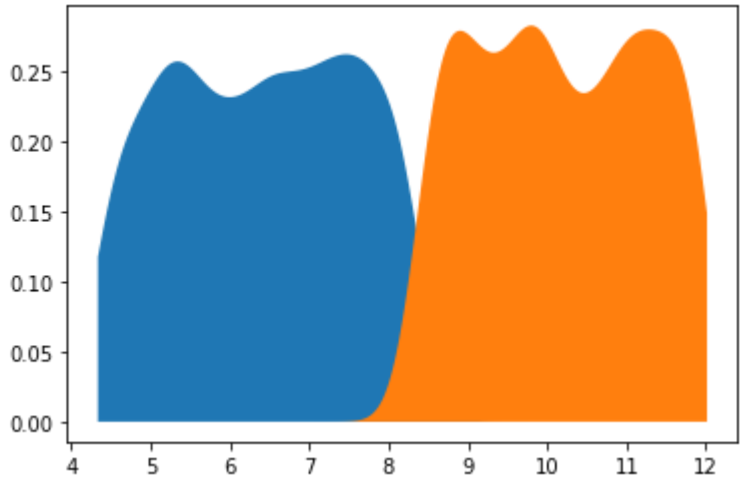
We are going to generate instances and label it until reach the target amount of new instances per category.

```
In [5…  # Define amount of new instances per category
        n_instances = 1000
        # Build nw dataset
        new_df = kde.random_generation(n_instances, kdes, attributes_i
```

Now we will plot the new dataset distribution and compare it with the original dataset distributions.

```
In [6…  # Calculate new kdes of the new dataset
        new_kdes = kde.calculate_kdes(new_df)[0]

        # Plot new dataset
        print("Figure 3: New Dataset generated using first method")
        kde.plot_univariate(new_df, new_kdes, ranges)
```

```
Figure 3: New Dataset generated using first method
```



The new dataset distributions is significantly different.

We can see that the random instances are well labeled. But if we compare each distributions with the originals, we will see that are not the same.

The reason behind of these differences may be that this method gives the same probability of appearing to all attribute values but in the distributions some values must have more propability of appearing than others.

The current version of this method does not converge.

Second method

In this method, we will generate relevant instances for each distribution, we will estimate density (relevancy) for each instance and we will build a dataset which each relevant instance is correctly represented.

We will generate each distribution in a independant way. For each distribution, we will generate relevant instances which be called support. For each instance, we will estimate density using the original `KDE` . The chosen `KDE` will be the associated one to the distribution (groped by category) which is currently being proccesed. This density determines which instances are more significant according to the original distribution so we will build a dataset which more relevant instances will be more represented. To achieve this we will introduce more occurrences of the relevant instances.

There are different ways to generate the support. This topic would require a study to find an efficient method.

We are going to use an exhaustive method which consist in generaing instances to cover the whole distribution space. For example, in our dataset, the attribute interval for category `1` is [7,12] so we will generate equally spaced instances between 7 and 12.

The support adjustment to original distribution can be managed increasing or decreasing the amount of instances. We will call granularity to this parameter. To optimize the granularity we will need a specify study.

To build the support in multivariate dataset cases we will generate each attribute support with the above method and do a cartesian product between all attributes supports.

If we think about exahustive method in a practical way, we will take into account that this method has an unacceptable complexity. So we will need to design more efficient methods in the future.

We are going to generate the support.

In [7…
```
# Define granularity
granularity = 100

# Generate supports
new_df = kde.univariate_support(df, granularity)
```

We are going to estimate density for each instance. We will use the original `KDEs` to achieve this.

In [8…
```
# Calculate densities for each instance
densities = kde.estimate_density(new_df, kdes)
```

Now we know the density for each new instance. That means that we know the representation grade of each new instance according to the original dataset. The goal is to modify the new dataset to get a correct representation for each instance.

There are multiple ways to change instance representations in the dataset.

We are going to increase the amount of the same instance using its density. That means instances with higher density will be appear more than instances with lower density. We will link to the instance with lowest density to an amount of one instance. So, to calculate the amount of duplicate instances for each instance, we are going to divide the instance density between lowest density.

We can think that duplicates instances adding redundant information and so it is. But there is no problem because it is just a method of get the proper representation of the instances. On the other hand, granularity is managed by the instances amount in the support. Nevertheless, there are alternatives to manage instances represention which do not involve duplicate instances. For example, we can search instances neighbours and use them.

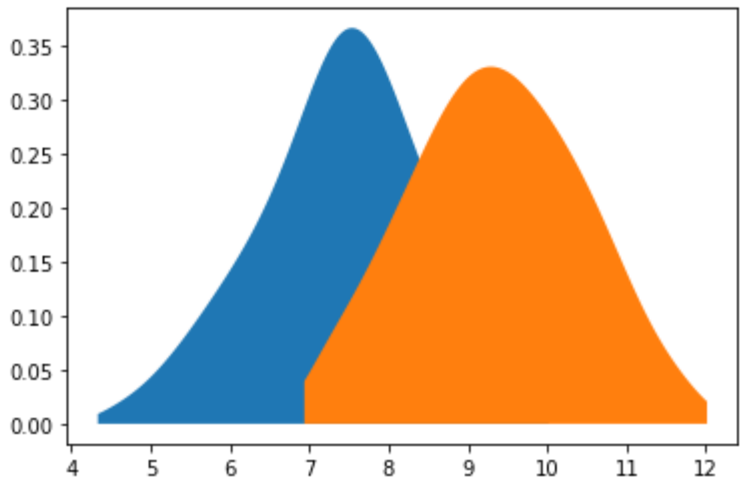
We are going to build the new dataset duplicating instances.

In [9…
```
# Adjust instance representations in the new dataset
new_df = kde.adjust_representation(densities)
```

Finally, we are going to plot new dataset and compare it with original dataset.

In [1…
```
# Plot new df
print("Figure 4: New dataset generated using second method")
new_kdes = kde.calculate_kdes(new_df)[0]
kde.plot_univariate(new_df, new_kdes)
```

```
Figure 4: New dataset generated using second method
```



If we compare the new dataset distributions with the original dataset distributions, we will see that they are the same so we can conclude that this method converges.

The new dataset size is about 2000 instances. The instances amount has increased significantly but it is not very important. This result is mainly due to the representation adjustment method used, which can easily change for another methods which can reduce the number of instances. Even if we can not find a method which give us a smaller dataset directly, we can design another `KDE` based method which can reduce dataset instances. For example, we can combine `KDE` with `MSE` to achieve this.

The most important part of the proposal method is that we can generate an equivalent dataset with different values compared to the original values.

Future changes

The original purpose is generate an equivalent dataset from the original dataset but with different values. For this, we will use `PDFs` which are estimated using `KDE`.

We are found a book section in which there are apparently at leat 4 methods to generate samples from a `PDF`. These methods would be able to more efficient than the second method so we will be interesting analyze them.