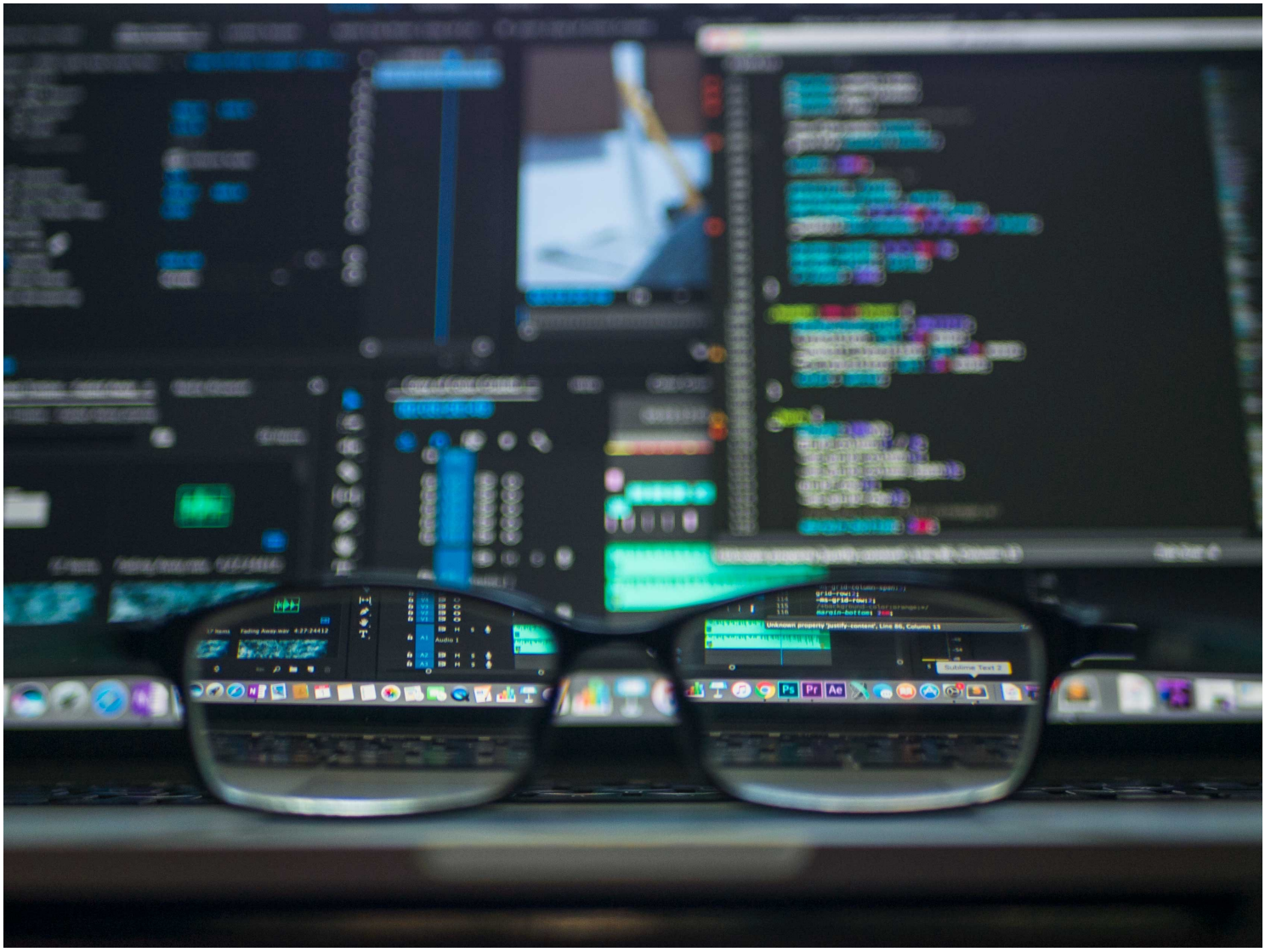

Photo by Kevin Ku from Pexels

An Introduction to Discretization Techniques for Data Scientists

Feature Engineering: 4 Discretization Techniques to Learn.

Rohan Gupta Follow
Dec 6, 2019 · 5 min read ★

Discretization is the process through which we can transform continuous variables, models or functions into a discrete form. We do this by creating a set of contiguous intervals (or bins) that go across the range of our desired variable/model/function.

> **Continuous data** is *Measured,* while **Discrete data** is *Counted.*

|| I || Why Discretization is Important

Mathematical problems with continuous data have an infinite number of DoF. Such a problem would entail having limited degrees of freedom (DoF) since our calculations cannot go on forever. Data Scientists require using Discretization for a number of reasons. Many of the top contributions on Kaggle use discretization for some of the following reasons:

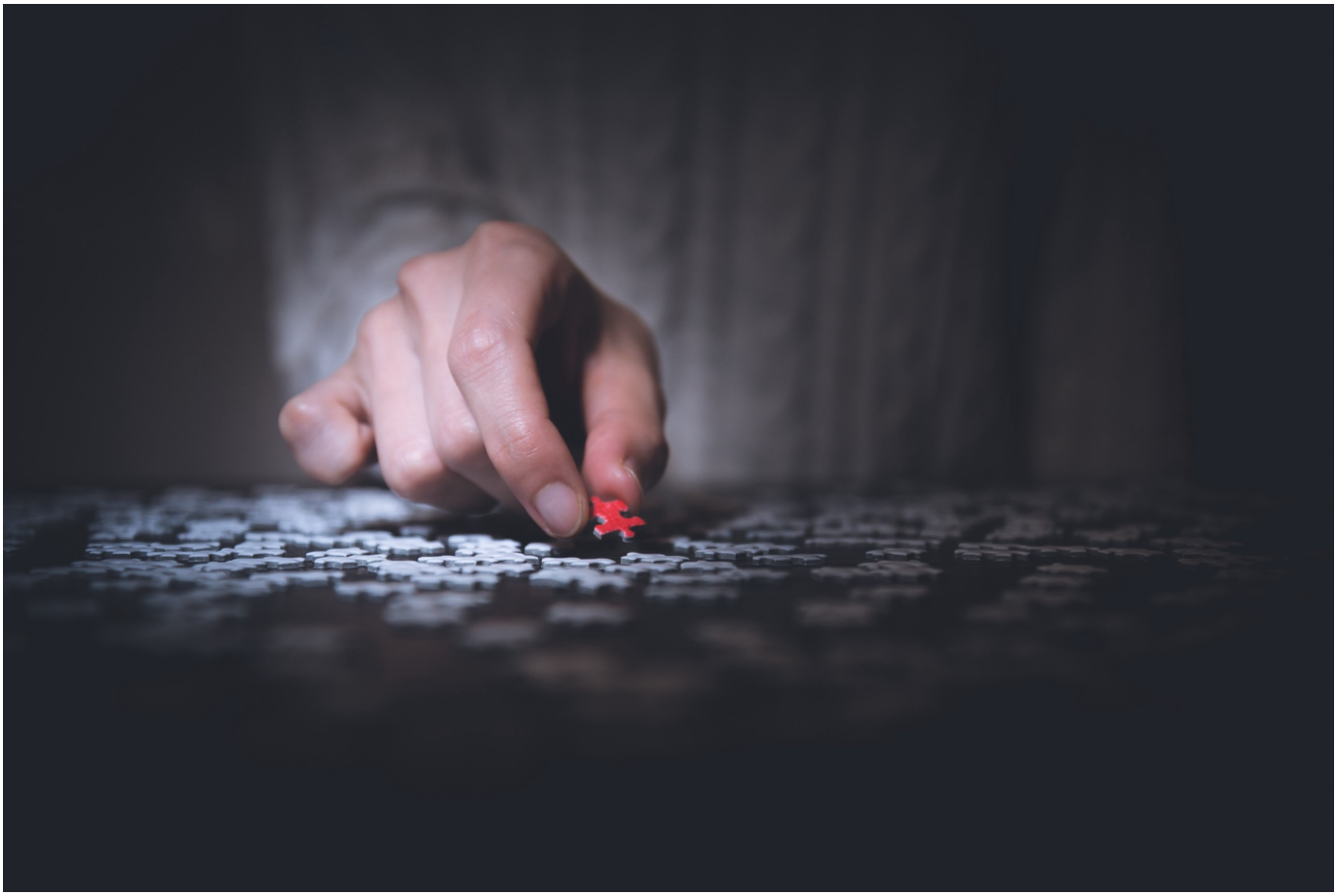

Photo by Ryoji Iwata on Unsplash

Fits the problem statement

Often, it is easier to understand continuous data (such as weight) when divided and stored into meaningful categories or groups. For example, we can divide a continuous variable, weight, and store it in the following groups :
Under 100 lbs *(light),* ***between 140–160 lbs*** *(mid), and* ***over 200 lbs*** *(heavy)*

We would consider the structure useful if we see no objective difference between variables falling under the same weight class.
In our example, weights of *85 lbs* and *56 lbs* convey the same information (the object is light). Therefore, discretization helps make our data easier to understand if it fits the problem statement.

. . .

Photo by William Daigneault on Unsplash

Interprets features

Continuous features have a smaller chance of correlating with the target variable due to infinite degrees of freedom and may have a complex non-linear relationship. Thus, it may be harder to interpret an such a function. After discretizing a variable, groups corresponding to the target can be interpreted.

. . .

Photo by Franck V. on Unsplash

Incompatible with models/methods

Certain models may be incompatible with continuous data, for example, alternative decision-tree models such as a Random-Forest model is not suitable for continuous features.

Feature engineering methods, for example any entropy-based methods may not work with continuous data, thus we would discretize variables to work with different models & methods.

. . .

Photo by Mariano Werneck on Unsplash

Signal-to-Noise Ratio

When we discretize a model, we are fitting it to bins and reducing the impact of small fluctuation in the data. Often, we would consider small fluctuations as noise. We can

reduce this noise through discretization. This is the process of "smoothing", wherein each bin smoothens fluctuations, thus reducing noise in the data.

. . .

|| II || Approaches to Discretization

- Unsupervised:
  — Equal-Width
  — Equal-Frequency
  — K-Means
- Supervised:
  — Decision Trees

|| III || Equal-Width Discretization

Separating all possible values into '**N**' number of bins, each having the same width. Formula for interval width:

Width = (maximum value - minimum value) / N
** where N is the number of bins or intervals.*

On python, you would want to import the following for discretization:

```
from sklearn.preprocessing import KBinsDiscretizer
from feature_engine.discretisers import EqualWidthDiscretiser
```

Set up the Equal-Width Discretizer in the following way:

```
discretizer = EqualWidthDiscretiser(bins=10, variables = ['var1',
'var2'])

#OR

discretizer = KBinsDiscretizer(n_bins=10, encode='ordinal',
strategy='uniform')
```

- Equal Width doesn't improve the value spread
- It can handle outliers
- Can be combined with categorical encodings

. . .

|| IV || Equal-Frequency Discretization

Separating all possible values into 'N' number of bins, each having the same amount of observations. Intervals may correspond to quantile values.

On python, you would want to import the following for discretization:

```
from sklearn.preprocessing import KBinsDiscretizer
from feature_engine.discretisers import EqualFrequencyDiscretiser
```

Set up the Equal-Frequency Discretizer in the following way:

```
discretizer = KBinsDiscretizer(n_bins=10, encode='ordinal',
strategy='quantile')

#OR

discretizer = EqualFrequencyDiscretiser(q=10, variables = ['var1',
'var2'])
```

- Equal Frequency does improve the value spread
- It can handle outliers
- Can be combined with categorical encoding

. . .

|| V || K-Means Discretization

We apply K-Means clustering to the continuous variable, thus dividing it into discrete groups or clusters.

On python, you would want to import the following for discretization with K-means:

```
from sklearn.preprocessing import KBinsDiscretizer
```

Set up the K-means Discretizer in the following way:

```
discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal',
strategy='kmeans')
```

- K-Means doesn't improve the value spread
- It can handle outliers, however a centroid bias may exist.
- Can be combined with categorical encoding

. . .

|| VI || Discretization with Decision Trees

We use a decision tree to identify the optimal number of bins. When the model makes a decision, it assigns an observation for each node. These observations are then classified into discrete output for our variable.

On python, you would want to import the following for discretization with decision trees:

```
from sklearn.model_selection import train_test_split
from feature_engine.discretisers import DecisionTreeDiscretiser
```

Your Discretizer should be set up in the following way:

```
# cross-validation number (cv)
# how to evaluate model performance (scoring)
# the variables we want to discretise (variables)
# whether it is a target for regression or classification
# and the grid with the parameters we want to test

treeDisc = DecisionTreeDiscretiser(cv=10, scoring='accuracy',
                                   variables=['var1', 'var2'],
                                   regression=False,
                                   param_grid={'max_depth': [1,2,3],
'min_samples_leaf':[10,4]})
```

- Decision Tree does not improve the value spread
- It can handle outliers well as trees are robust to outliers.
- Creates monotonic relationships

. . .

|| VII || Next Steps

After discretizing variables, you can do either of the following:

- Build decision tree algorithms and directly use the output of discretization as the number of bins. The decision trees can find non-linear relationships between the discretized variable and the target variables.
- Use a linear model, while the bins do not have a linear relationship with the target variable. Improve the model by treating bins as categories with some sort of encoding.

. . .

Thanks for the read!

Follow me for more content on Data Science.

Data Science Machine Learning Python Pandas Data

About Help Legal