Title: The Data - Type conversions

Course: Machine Learning

Instructor: Claudio Sartori

Master: Data Science and Business Analytics
Master: Artificial Intelligence and Innovation
Master: Finance and Financial Technologies

Academic Year: 2023/2024

Why do we need type conversion?

- Many algorithms require numeric features
  - categorical features must be transformed into numeric
  - ordinal features must be transformed into numeric, and the order must be preserved
- Classification requires a target with nominal values
  - a numerical target can be discretised
- Discovery of association rules require boolean features
  - a numerical feature can be discretised and transformed int a series of boolean features

Binarization of discrete attributes

Attribute d allowing V values $\Rightarrow$ V binary attributes.

Quality
Awful
Poor
Ok
Good
Great

Quality–Awful	*Quality–Poor*	*Quality–OK*	*Quality–Good*	*Quality–Great*
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Nominal to numeric

- One–Hot–Encoding
  - a feature with V unique values is substituted by V binary features each one corresponding to one of the unique values
  - if object x has value $v$in feature d then the binary feature corresponding to v has `True` for x, all the other binary features have value `False`
  - `True` and `False` are represented as 1 and 0, therefore can be processed by also by procedures working only on numeric data, as is the case for the estimators available in `scikit-learn`
- `sklearn.preprocessing.OneHotEncoder`

link to manual page

Ordinal to numeric

- The ordered sequence is transformed into consecutive integers
  - by default the lexicographic order is assumed
  - The user can specify the proper order of the sequence
- `sklearn.preprocessing.OrdinalEncoder`

link to manual page

Numeric to binary with threshold

- Not greater than the threshold becomes zero
- Greater than the threshold becomes one
- `sklearn.preprocessing.Binarizer`

link to manual page

Discretization/Reduction of the number of distinct values

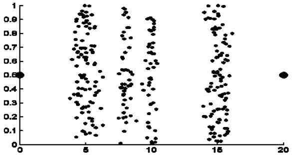
- Some algorithms work better with categorical data
- A small number of distinct values can let patterns emerge more clearly
- A small number of distinct values let the algorithms to be less influenced by noise and random effects

Discretization

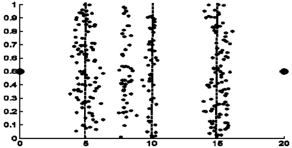
- Continuous ⇒ Discrete
  - thresholds
    - many options
  - binarization ⇒ single threshold
- Discrete with many values ⇒ Discrete with less values
  - guided by domain knowledge

Continuous ⇒ Discrete

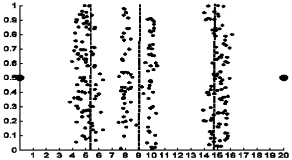
Boundaries on x axis – Unsupervised



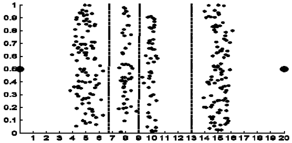
Data



Equal width



Equal frequency



K-means

Numeric to *k* values

- The numbers are discretised into a sequence of integers 0 to $k-1$
- Several strategies are available
  - `{'uniform','quantile','means'}`
- `sklearn.preprocessing.KBinsDiscretizer`

link to manual page

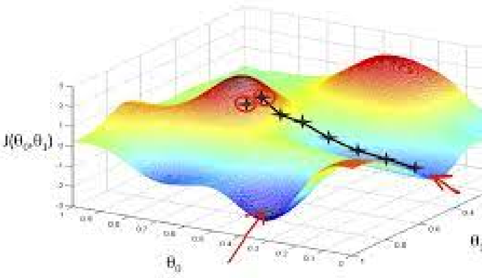
Why Data Transformation

- the features may have different scales
  - this can alterate the results of many learning techniques
  - some machine learning algorithms are sensitive to feature scaling while others are virtually invariant to it
- there can be outliers

Gradient descent

Machine learning algorithms that use *gradient descent* as an optimization technique require data to be scaled

- e.g. linear regression, logistic regression, neural network, etc.
- The presence of feature value X in the formula will affect the step size of the gradient descent
- The difference in ranges of features will cause different step sizes for each feature.
- Similar ranges of the various features ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features



Attribute transformation

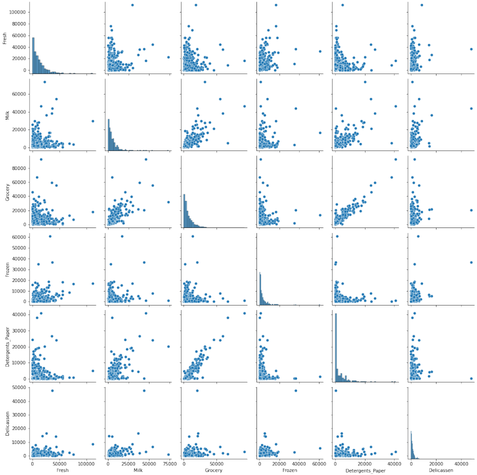
- Map the entire set of values to a new set according to a function
  - $x^k, \log(x), e^x, |x|$
  - in general they change the *distribution of values*
- Standardization: $x \to \frac{x-\mu}{\sigma}$
  - if the original values have a *gaussian* distribution, the transformed values will have a *standard* gaussian distribution $(\mu = 0, \sigma = 1)$
  - translation and shrinking/stretching, no change in distribution
- MinMax scaling (a.k.a. Rescaling): the domains are mapped to standard ranges

$$x \to \frac{x - x_{min}}{x_{max} - x_{min}} \quad \text{(0 to 1)} \qquad\qquad x \to \frac{x - \frac{x_{max}+x_{min}}{2}}{\frac{x_{max}-x_{min}}{2}} \quad \text{(-1 to 1)}$$

  - translation and shrinking/stretching, no change in distribution

Attribute transformation – Example I

Data with skewed distribution

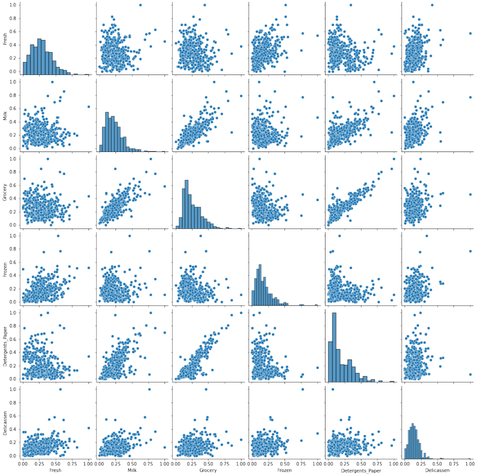


Attribute transformation – Example II

Python code

```
from sklearn.preprocessing
      import PowerTransformer
pt = PowerTransformer(method=’box-cox’)
X = pd.DataFrame(pt.fit_transform(X0)
                  , columns = X0.columns)
```

After the transformation the data are *less skewed*



Distance–based algorithms

- KNN, K–Means, SVM, . . .
- distances between points are used to determine their similarity

Example

Original data		
Student	**CGPA**	**Salary**
A	3	60
B	3	40
C	4	40
D	4.5	50
E	4.2	52

Scaled data		
Student	**CGPA**	**Salary**
A	-1.18431	1.520013
B	-1.18431	-1.100699
C	0.41612	-1.100699
D	1.21635	0.209657
E	0.736212	0.471728

Distances before and after scaling

$$distance(A, B) = \sqrt{(40 - 60)^2 + (3 - 3)^2} = 20$$
$$distance(B, C) = \sqrt{(40 - 40)^2 + (4 - 3)^2} = 1$$
$$distance(A_s, B_s) = \sqrt{(1.1 + 1.5)^2 + (1.18 - 1.18)^2} = 2.6$$
$$distance(B_s, C_s) = \sqrt{(1.1 - 1.1)^2 + (0.41 + 1.18)^2} = 1.59$$

Before the scaling the two distances seemed to be very different, due to the a big numeric difference in the `Salary` attribute, now they are comparable

Range–based scaling and standardization

operate on single features

- Range–based scaling stretches/shrinks and translates the range, according to the range of the feature (there are some variants)
  - good when we know that the data are not gaussian, or we do not make any assumption on the distribution
  - the base variant, the MinMax scaler, remaps to $0, 1$
- Standardization subtracts the mean and divides by the standard deviation
  - the resulting distribution has mean *zero* and *unitary* standard deviation
  - good when the distribution is gaussian
  - `StandardScaler`

Range–based scalers in Scikit-Learn

affine transformations: linear transformation plus translation

- `MinMaxScaler` – remaps the feature to $[0, 1]$
- `RobustScaler` – centering and scaling statistics is based on percentiles
    - not influenced by a few number of very large marginal outliers
    - the resulting range of the transformed feature values is larger than the one given by `MinMaxScaler` and `StandardScaler`

Normalization

- Normalization is mentioned sometimes with different meanings
  - frequently it refers to `MinMaxScaler`
- in `Scikit-learn` the `Normalizer` normalizes each data row to unit norm

Workflow

1. transform the features as required both for the train and test data
2. fit and optimize the model(s)
3. test
4. possibly, use the original data to plot relevant views (e.g. to plot cluster assignments)

Imbalanced data in classification

- The performance minority class (classes) has little impact on standard performance measures
- The optimised model could be less effective on minority class (classes)
- Some estimators allow to weight classes
- Some performance measures allow to take into account the contribution of minority class (classes)

Cost Sensitive learning

Already introduced in *Machine-Learning-03-classification*

- several classifiers have the parameter `class_weight`
- it changes the cost function to take into account the imbalancing of classes
- in practice it is equivalent to oversampling the minority class, (repeating random examples) in order to produce a balanced training set

Undersampling

- Obtains a balanced training set by randomly reducing the number of examples of the majority class
- Obviously part of the knowledge embedded in the training set is dropped out

Oversampling with SMOTE

Synthetic Minority Oversampling Technique – a type of data augmentation

let the minority class be c_{min}, synthesise new examples of class c_{min}

- choose from the *training set* random example x_r of class c_{min}
- find in the training set the k nearest neighbours of x_r whose class is c_{min}
- choose randomly one of the neighbours, say x_{rn} found above and *create* a new data element chosen randomly from the segment connecting x_r x_{rn} in the feature space $m = r * (x_r + x_{rn})/2$

Theory developed in SMOTE: Synthetic Minority Over-sampling Technique[Bowyer et al.(2011)Bowyer, Chawla, Hall, and Kegelmeyer]

Workflow for *undersampling/oversampling*

- resample the training set
- fit and optimise the estimator
- test the fitted estimator on the test set (untouched)

Bibliography I

- Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer.
  SMOTE: synthetic minority over-sampling technique.
  CoRR, abs/1106.1813, 2011.
  URL http://arxiv.org/abs/1106.1813.