b-it Bonn-Aachen International Center for Information Technology

Introduction to Machine Learning Tutorial

Categorical Data

Handling **categorical variables** is another integral aspect of Machine Learning. Categorical variables are basically the variables that are **discrete** and **not continuous**.

Example — **colour of an item** is a **discrete variable** whereas its **price is a continuous variable.**

Categorical Data

Categorical variables are further divided into 2 types:

- **Ordinal categorical variables** — These variables can be ordered.

Example: Size of a T-shirt. We can say that M<L<XL.

- **Nominal categorical variables** — These variables can't be ordered.

Example: colour of a T-shirt. We can't say that Blue < Green as **it doesn't make any sense to compare the colours** as they don't have any relationship.

This means both feature types have to be **processed differently**.

Ordinal Categorical Data

Let‘s look at it with an example:

```
df_cat = pd.DataFrame(data = [
                            ['green','M',10.1,'class1‘],
                            ['blue','L',20.1,'class2‘],
                            ['white','M',30.1,'class1']])

df_cat.columns = ['colour','size','price','classlabel']
```

Ordinal Categorical Data

```
df_cat = pd.DataFrame(data = [
                              ['green','M',10.1,'class1`],
                              ['blue','L',20.1,'class2`],
                              ['white','M',30.1,'class1']])

df_cat.columns = ['colour','size','price','classlabel']
```

Here the columns ‘size’ and ‘classlabel’ are **ordinal categorical variables** whereas ‘**colour**’ is a **nominal categorical variable**.

Ordinal Categorical Data

```
df_cat = pd.DataFrame(data =[
                              ['green','M',10.1,'class1`],
                              ['blue','L',20.1,'class2`],
                              ['white','M',30.1,'class1']])

df_cat.columns = ['colour','size','price','classlabel']
```

There are 2 simple and neat techniques to transform ordinal categorical variables:

- Using the **map()** function
- Using the **LabelEncoder**

Map() Function

```
size_mapping = {'M':1,'L':2}
df_cat['size'] = df_cat['size'].map(size_mapping)
```

What was done?

- Creation of a dictionary to map M and L to a number, 1 and 2 respectively.
- Here M will be replaced with 1 and L with 2.

LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
df_cat['classlabel'] =
class_le.fit_transform(df_cat['classlabel'].values)
```

What happened?

- The library LabelEncoder is imported
- An instance of the LabelEncoder is created
- The LabelEncoder is fitted to the data and and then transforms the classlabels feature (column): class1 will be represented by 0 and class2 by 1.
- Additional hint: `fit_transform` is applied to all values of the dictionary that belong to „classlabel“.

Common Mistakes

The **biggest mistake** that is often made is that users do not properly differentiate between **ordinal and nominal categorical variables**.

So, if you use the same `map()` function or `LabelEncoder` with nominal variables then the model will think that there is some sort of relationship between the nominal categorical variables.

So if we use the `map()` funtion to map the colours as in the following example:

```
col_mapping = {'Blue':1,'Green':2}
```

The algorithm will assume that according to the model **Green > Blue**, which is a **wrong assumption**.

So, the model will give you results considering this relationship, although you will get **wrong results using this method even without realising it at first.**

Correct Handling of Nominal Variables

The correct way of handling **nominal categorical variables** is to use **One-Hot Encoding**.

The easiest way to use One-Hot Encoding is to use the **get_dummies() function**.

One-Hot Encoding

```
df_cat = pd.get_dummies(df_cat[['colour','size','price`]])
```

What happened?

- We have passed ‘size’ and ‘price’ along with ‘colour’ but the get_dummies() function.
- The functions is very smart: It will only consider the nominal values in the data set.
- In our case this will be the ‘colour’ feature.

One-Hot Encoding

What we essentially do in One-Hot Encoding is that we create ’n’ columns where n is the number of unique values that the nominal variable can take.

Example: In our case the colour can take the nominal values: Blue, Green and White. The **OneHotEncoder** will just create three new columns and these are:

- colour_blue,
- colour_green and
- colour_white

If the colour is green then the values of colour_blue and colour_white column will be 0 and value of colour_green column will be 1 .

One-Hot Encoding

Colour	Colour_blue	Colour_green	Colour_white
white	0	0	1
blue	1	0	0
green	0	1	0

What happened?

The colour that is ‘hot’ (e. g. in case of the white colour, the last column) gets the 1, all other colours in all other columns will be set to zero (= cold). One can state: The nominal colours have been translated into a matrix consisting of three columns. Each column can carry the values 0 or 1.

Multicollinearity

Multicollinearity occurs in our dataset when we have features which are strongly dependent on each other.

Example: In this case we have features -
colour_blue, colour_green and colour_white which are all dependent on each other and it can impact our model.

Multicollinearity

- The main impact it will have is that it can cause the **decision boundary to change** which can have a huge impact on the result of our model.

- In addition to that if we have multicollinearity in our dataset then we won't be able to use our weight vector to calculate the feature importance.

Let's get a little bit more information about **multicollinearity**.

Mathematics behind Multicollinearity

When predictor variables in the same regression model are correlated, they cannot independently predict the value of the dependent variable.

In other words, **they explain some of the same variance in the dependent** variable, which in turn **reduces their statistical significance**.

Collinearity can, however, be detected using the **Variance Inflation Factor (VIF)**.

Mathematics behind Multicollinearity

It can, however, be detected by the **Variance Inflation Factor (VIF)**.

$$\mathrm{VIF}_i = \frac{1}{1 - R^2{}_i}$$

where R^2 is the **coefficient of determination** (= the proportion of the variance in the dependent variable that is predictable from the independent variable(s).)

If VIF = 1 or 2 → no collinearity or multicollinearity
IF VIF = 20 → collinearity most likely

Multicollinearity

Multicollinearity exists **whenever two or more of the predictors in a regression model are moderately or highly correlated**.

There are two types of multicollinearity:

- **Structural multicollinearity** is a mathematical artifact caused by creating new predictors from other predictors — such as, creating the predictor x^2 from the predictor x.

- **Data-based multicollinearity**, on the other hand, is a result of a poorly designed experiment, reliance on purely observational data, or the inability to manipulate the system on which the data are collected.

Causes for Multicollinearity

- **Insufficient data.** In some cases, collecting more data can resolve the issue.
- **Dummy variables** may be incorrectly used. For example, the researcher may fail to exclude one category, or add a dummy variable for every category (e.g. spring, summer, autumn, winter).
- **Including a variable in the regression that is actually a combination of two other variables**. For example, including "total investment income" when total investment income = income from stocks and bonds + income from savings interest.
- **Including two identical (or almost identical) variables**. For example, weight in pounds and weight in kilos.

Multicollinearity

Data-based multicollinearity is the more troublesome of the two types of multicollinearity. Unfortunately it is the type we encounter most often!

Multicollinearity

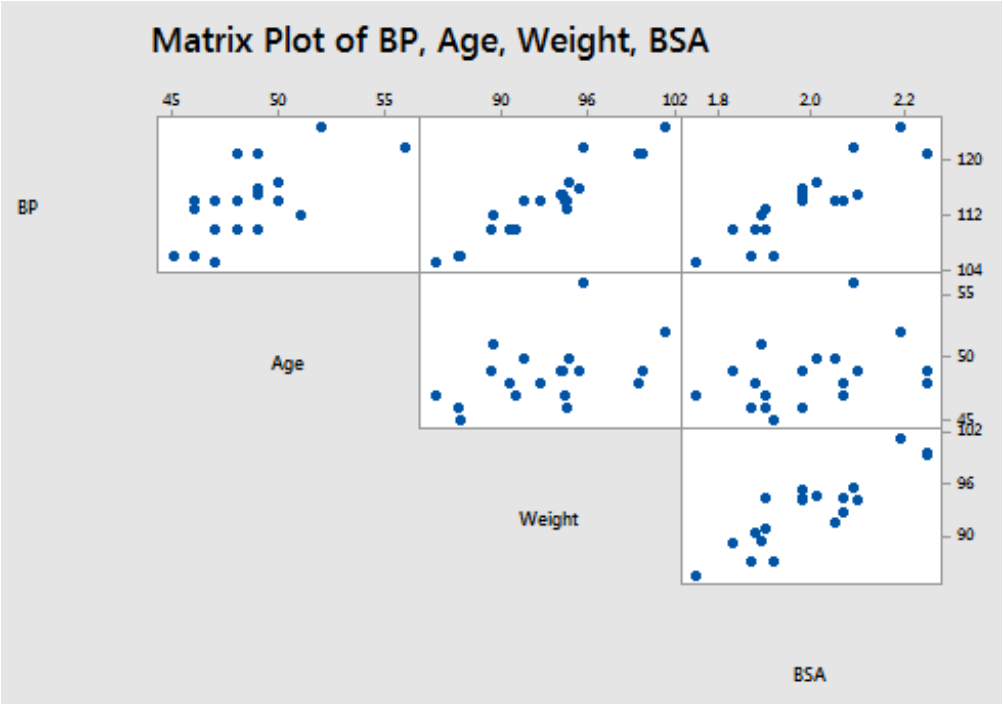
Let's take a quick look at an example in which data-based multicollinearity exists. Some researchers observed — notice the choice of word! —on 20 individuals with high blood pressure:

- blood pressure (y = *BP*, in mm Hg)
- age (x_1 = *Age*, in years)
- weight (x_2 = *Weight*, in kg)
- body surface area (x_3 = *BSA*, in sq m)
- duration of hypertension (x_4 = *Dur*, in years)
- basal pulse (x_5 = *Pulse*, in beats per minute)
- stress index (x_6 = *Stress*)

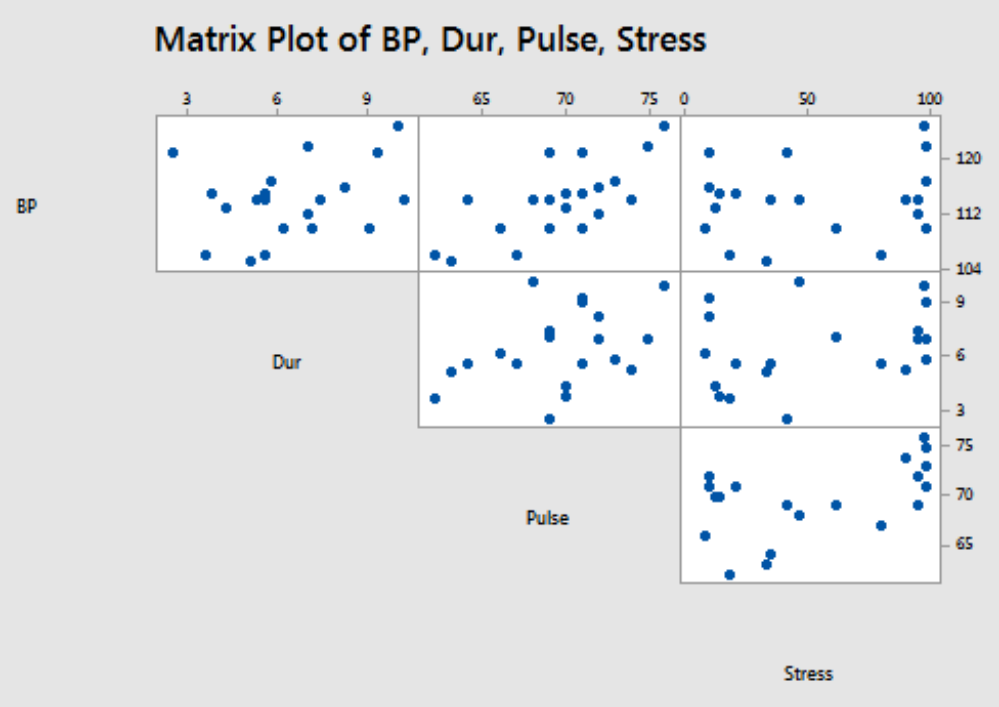
The researchers were interested in determining if a **relationship exists between blood pressure and age, weight, body surface area, duration, pulse rate and/or stress level.**

Multicollinearity

BSA: Body surface area



Multicollinearity



Blood pressure (BP) appears to be related fairly strongly to *Weight* and *BSA*, and hardly related at all to *Stress* level.

Multicollinearity

Let's additionally look into the correlation matrix:

Correlation: BP, Age, Weight, BSA, Dur, Pulse, Stress

	BP	Age	Weight	BSA	Dur	Pulse
Age	0.659					
Weight	0.950	0.407				
BSA	0.866	0.378	0.875			
Dur	0.293	0.344	0.201	0.131		
Pulse	0.721	0.619	0.659	0.465	0.402	
Stress	0.164	0.368	0.034	0.018	0.312	0.506

Blood pressure appears to be related fairly strongly to ***Weight*** (r = 0.950) and *BSA* (r = 0.866), and **hardly related at all to *Stress*** level (r = 0.164). And, ***Weight* and *BSA* appear to be strongly related** (r = 0.875), while *Stress* and *BSA* appear to be hardly related at all (r = 0.018). The **high correlation** among some of the **predictors suggests that data-based multicollinearity exists**.

Avoidance of Multicollinearity

- Dropping feature(s) that are collinear
- Linearly combining predictors, such as adding them together
- Feature extraction/ feature engineering

Example: The features of body weight and height are highly correlated. This means: The taller a person is, the heavier (the more weight) he/she is.

You could still use these original features by introducing a new feature that contains both original features, body weight and height: The **BMI (body mass index).**

$$\text{BMI} = \frac{weight\ [kg]}{(height\ [m])^2}$$

One-Hot Encoding

Colour	Colour_blue	Colour_green	Colour_white
white	0	0	1
blue	1	0	0
green	0	1	0

Example: In this case we have features - colour_blue,colour_green and colour_white which are all dependent on each other and it can impact our model.

How can we deal with multicollinearity in this case?

One-Hot Encoder

"The hack":

We can use **`drop_first=True`** in order to avoid the problem of multicollinearity.

```
df_cat =
pd.get_dummies(df_cat[['color','size','price']],drop
_first=True)
```

Here **`drop_first`** will drop the **first column** of colour. This means: colour_blue will be dropped and we will only have colour_green and colour_white.

One-Hot Encoder

```
df_cat = pd.get_dummies(df_cat[['colour','size','price']]
        , drop_first=True)
```

The important thing to note here is that we **don't lose any information** because **if colour_green and colour_white are both 0 then it implies that the color must have been blue**.

So we can infer the whole information with the help of only these 2 columns, hence the **strong correlation** between these three columns is **broken**.

Summary

In the past sessions we have focused on some essential **data pre-processing methods** in ML.

We centred on:

- **Data Scaling** (normalisation and standardisation) → apply this to the given data sets
- **Imputation** (handling missing data NaN or Na) → apply this to the given data sets
- **Converting categorical features into numbers** (e. g. map(), LabelEncoder, One-Hot Encoder) → you will apply this in the near future