3.1-Data Preparation for Machine Learning

KeytoDataScience.com

Data preparation is a vital step in the machine learning pipeline. Just as visualization is necessary to understand the relationships in data, proper preparation or **data munging** is required to ensure machine learning models work optimally.

The process of data preparation is highly interactive and iterative. A typical process includes at least the following steps:

1. **Visualization** of the dataset to understand the relationships and identify possible problems with the data.
2. **Data cleaning and transformation** to address the problems identified. In many cases, step 1 is then repeated to verify that the cleaning and transformation had the desired effect.

In this Session you will learn the following:

- Recode character strings to eliminate characters that will not be processed correctly.
- Find and treat missing values.
- Set correct data type of each column.
- Transform categorical features to create categories with more cases and coding likely to be useful in predicting the label.
- Apply transformations to numeric features and the label to improve the distribution properties.
- Locate and treat duplicate cases.

`💬💬💬 Automotive DataSet Example 💬💬💬`

As a first example you will prepare the automotive dataset. Careful preparation of this dataset, or any dataset, is required before atempting to train any machine learning model. This dataset has a number of problems which must be addressed. Further, some feature engineering will be applied.

Table of Contents

- 1 Load the dataset
- 2 Recode names
- 3 Dropping Variables
- 4 Renaming Columns (single or multiple)
- 5 Sorting Data (single, multiple columns) in ascending and descending

- 6 Type Conversions(Convert Data types of columns)
- 7 Resetting Index
- 8 Handling Duplicates
- 9 Treat & Handling missing values
- 10 Create Dummies for a Categorical Variable
- 11 Feature engineering
- 12 Summary

1 Load the dataset

Load the packages required to run this notebook.

In [1]:

```
import pandas as pd
import numpy as np

%matplotlib inline
```

Load the dataset and print the first few rows of the data frame

In [2]:

```
# read the txt file
auto_prices = pd.read_table('Automobile price data _Raw_.txt', delimiter=',')

# print first five rows of dataframe
auto_prices.head(5)
```

Out[2]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...

5 rows × 26 columns

In [3]:

```
# check the info regarding dataframe
auto_prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
```

```
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       205 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               205 non-null    object
 19  stroke             205 non-null    object
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object
 22  peak-rpm           205 non-null    object
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              205 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

In [4]:

```
# print first five rows of dataframe that contains number
auto_prices.select_dtypes(include = ['number']).head(5)
```

Out[4]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
0	3	88.6	168.8	64.1	48.8	2548	130	9.0	21	27
1	3	88.6	168.8	64.1	48.8	2548	130	9.0	21	27
2	1	94.5	171.2	65.5	52.4	2823	152	9.0	19	26
3	2	99.8	176.6	66.2	54.3	2337	109	10.0	24	30
4	2	99.4	176.6	66.4	54.3	2824	136	8.0	18	22

In [5]:

```
# print first five rows of dataframe that contains number
auto_prices.select_dtypes(include = ['number']).columns
```

Out[5]:

```
Index(['symboling', 'wheel-base', 'length', 'width', 'height', 'curb-weight',
       'engine-size', 'compression-ratio', 'city-mpg', 'highway-mpg'],
      dtype='object')
```

In [6]:

```
# check all column names
auto_prices.columns
```

Out[6]:

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
```

```
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

In [7]:

```
# check shape of dataframe
auto_prices.shape
```

Out[7]:

```
(205, 26)
```

Subsetting the data for easier understanding

In [8]:

```
# Subsetting the Data
auto_prices=auto_prices[['make', 'peak-rpm', 'body-style', 'curb-weight',
                         'horsepower', 'num-of-cylinders', 'city-mpg', 'highway-mpg','p
```

In [9]:

```
# Create a New Column:
auto_prices['New_Col']=np.absolute(auto_prices['curb-weight'])
```

In [10]:

```
auto_prices.New_Col
```

Out[10]:

```
0      2548
1      2548
2      2823
3      2337
4      2824
       ...
200    2952
201    3049
202    3012
203    3217
204    3062
Name: New_Col, Length: 205, dtype: int64
```

We will now perform some data preparation steps.

↥ back to top

2 Recode names

Notice that several of the column names contain the '-' character. Python will not correctly recognize character strings containing '-'. Rather, such a name will be recognized as two character strings. The same problem will occur with column values containing many special characters including, '-', ',', '*', '/', '|', '>', '<', '@', '!' etc. If such characters appear in column names of values, they must be replaced with another character.

In [11]:

```
# check number of cylinders and their count
auto_prices['num-of-cylinders'].value_counts()
```

Out[11]:

```
four      159
six        24
```

```
five        11
eight        5
two          4
three        1
twelve       1
Name: num-of-cylinders, dtype: int64
```

In [12]:

```
# check number of cylinders and their count
auto_prices.num-of-cylinders.value_counts()

# this will give error, as column name has '-' between it
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_14600/390583709.py in <module>
      1 # check number of cylinders and their count
----> 2 auto_prices.num-of-cylinders.value_counts()
      3 
      4 # this will give error, as column name has '-' between it

~\AppData\Roaming\Python\Python39\site-packages\pandas\core\generic.py in __getattr__(se
lf, name)
   5485         ):
   5486             return self[name]
-> 5487         return object.__getattribute__(self, name)
   5488 
   5489     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'num'
```

In [13]:

```
auto_prices.columns=auto_prices.columns.str.replace('-','_')
# or
# auto_prices.columns = [str.replace('-', '_') for str in auto_prices.columns]
```

In [14]:

```
# check number of cylinders and their count
# new the same code will work fine, as it do not have any special character in column n
auto_prices.num_of_cylinders.value_counts()
```

Out[14]:

```
four      159
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: num_of_cylinders, dtype: int64
```

↥ back to top

3 Dropping Variables

In [15]:

```
auto_prices.drop("New_Col",axis = 1).head()
```

Out[15]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mpg

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mpg
0	alfa-romero	5000	convertible	2548	111	four	21	27
1	alfa-romero	5000	convertible	2548	111	four	21	27
2	alfa-romero	5000	hatchback	2823	154	six	19	26
3	audi	5500	sedan	2337	102	four	24	30
4	audi	5500	sedan	2824	115	five	18	22

In [16]:

```
auto_prices.columns
```

Out[16]:

```
Index(['make', 'peak_rpm', 'body_style', 'curb_weight', 'horsepower',
       'num_of_cylinders', 'city_mpg', 'highway_mpg', 'price', 'New_Col'],
      dtype='object')
```

In [17]:

```
auto_prices.columns.difference(['New_Col'])
```

Out[17]:

```
Index(['body_style', 'city_mpg', 'curb_weight', 'highway_mpg', 'horsepower',
       'make', 'num_of_cylinders', 'peak_rpm', 'price'],
      dtype='object')
```

↥ back to top

4 Renaming Columns (single or multiple)

In [18]:

```
#renaming column "RevolvingUtilization with Rev_Utilization" and "SeriousDlqin2yrs with
auto_prices.rename(columns={'aspiration':'Aspiration_', 'price':'Car_Price'}).head(10)
```

Out[18]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mpg
0	alfa-romero	5000	convertible	2548	111	four	21	27
1	alfa-romero	5000	convertible	2548	111	four	21	27
2	alfa-romero	5000	hatchback	2823	154	six	19	26
3	audi	5500	sedan	2337	102	four	24	30
4	audi	5500	sedan	2824	115	five	18	22
5	audi	5500	sedan	2507	110	five	19	25
6	audi	5500	sedan	2844	110	five	19	25
7	audi	5500	wagon	2954	110	five	19	25
8	audi	5500	sedan	3086	140	five	17	20
9	audi	5500	hatchback	3053	160	five	16	22

↥ back to top

5 Sorting Data (single, multiple columns) in ascending and descending

In [19]:
```
## Sorting the data
auto_prices.sort_values(by='city_mpg', ascending=False).head(7)
```

Out[19]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_m
30	honda	4800	hatchback	1713	58	four	49	
18	chevrolet	5100	hatchback	1488	48	three	47	
90	nissan	4800	sedan	2017	55	four	45	
45	isuzu	5400	sedan	1909	70	four	38	
32	honda	5500	hatchback	1837	60	four	38	
159	toyota	4500	hatchback	2275	56	four	38	
160	toyota	4800	sedan	2094	70	four	38	

In [20]:
```
auto_prices.sort_values(by = ["city_mpg","highway_mpg"], ascending=False).head(7)
```

Out[20]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_m
30	honda	4800	hatchback	1713	58	four	49	
18	chevrolet	5100	hatchback	1488	48	three	47	
90	nissan	4800	sedan	2017	55	four	45	
159	toyota	4500	hatchback	2275	56	four	38	
160	toyota	4800	sedan	2094	70	four	38	
19	chevrolet	5400	hatchback	1874	70	four	38	
20	chevrolet	5400	sedan	1909	70	four	38	

↥ back to top

6 Type Conversions(Convert Data types of columns)

Noted that, there are three columns in this dataset which do not have the correct type as a result of missing values. This is a common situation, as the methods used to automatically determine data type when loading files can fail when missing values are present.

In [21]:
```
# check the Dtype of the dataframe
```

```
auto_prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   make             205 non-null    object
 1   peak_rpm         205 non-null    object
 2   body_style       205 non-null    object
 3   curb_weight      205 non-null    int64
 4   horsepower       205 non-null    object
 5   num_of_cylinders 205 non-null    object
 6   city_mpg         205 non-null    int64
 7   highway_mpg      205 non-null    int64
 8   price            205 non-null    object
 9   New_Col          205 non-null    int64
dtypes: int64(4), object(6)
memory usage: 16.1+ KB
```

As seen in the below table

- peak_rpm
- horsepower
- price

are actually numerical variables. However due presence of non numerical values they are mentioned as 'object' type.

In [22]:

```
auto_prices.head(3)
```

Out[22]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mpg
0	alfa-romero	5000	convertible	2548	111	four	21	27
1	alfa-romero	5000	convertible	2548	111	four	21	27
2	alfa-romero	5000	hatchback	2823	154	six	19	26

First filter out the non-numeric value in the column

In [23]:

```
# Let's check for peak_rpm
auto_prices[pd.to_numeric(auto_prices['peak_rpm'], errors='coerce').isnull()]
```

Out[23]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mp
130	renault	?	wagon	2579	?	four	23	3
131	renault	?	hatchback	2460	?	four	23	3

As seen in the above table `peak_rpm` has `?` in 2 rows.

Let's convert non numeric values to **nan** so they can be easily replaced using numpy operation.

We'll be using `pd.to_numeric` and setting parameter `errors='coerce'`

In [24]:
```
cols = ['peak_rpm','horsepower','price']
```

In [25]:
```
# dtypes before conversion
auto_prices[cols].dtypes
```

Out[25]:
```
peak_rpm      object
horsepower    object
price         object
dtype: object
```

The code in the cell below iterates over a list of columns setting them to numeric. Execute this code and observe the resulting types.

In [26]:
```
# dtypes after conversion
for column in cols:
    auto_prices[column] = pd.to_numeric(auto_prices[column],errors='coerce')
auto_prices[cols].dtypes
```

Out[26]:
```
peak_rpm      float64
horsepower    float64
price         float64
dtype: object
```

In [27]:
```
auto_prices.head(3)
```

Out[27]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mpg
0	alfa-romero	5000.0	convertible	2548	111.0	four	21	27
1	alfa-romero	5000.0	convertible	2548	111.0	four	21	27
2	alfa-romero	5000.0	hatchback	2823	154.0	six	19	26

↥ back to top

7 Resetting Index

It is used to create a DF with the data *conformed* to a new index.

If we subset a Series or DataFrame with an index object,

the data is *rearranged* to obey this new index and missing values are introduced wherever the data was not present

In [28]:

```python
auto_prices.set_index("make").head(5)
```

Out[28]:

make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mpg
alfa-romero	5000.0	convertible	2548	111.0	four	21	27
alfa-romero	5000.0	convertible	2548	111.0	four	21	27
alfa-romero	5000.0	hatchback	2823	154.0	six	19	26
audi	5500.0	sedan	2337	102.0	four	24	30
audi	5500.0	sedan	2824	115.0	five	18	22

In [29]:

```python
auto_prices.reset_index().head(4) #create variable
```

Out[29]:

	index	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highwa
0	0	alfa-romero	5000.0	convertible	2548	111.0	four	21	
1	1	alfa-romero	5000.0	convertible	2548	111.0	four	21	
2	2	alfa-romero	5000.0	hatchback	2823	154.0	six	19	
3	3	audi	5500.0	sedan	2337	102.0	four	24	

↥ back to top

8 Handling Duplicates

- `df.duplicated()` Returns boolean Series denoting duplicate rows, optionally only considering certain columns
- `df.drop_duplicates()` Returns DataFrame with duplicate rows removed, optionally only considering certain columns

In [30]:

```python
auto_prices.loc[auto_prices.duplicated()].head(5)
```

Out[30]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_m
88	mitsubishi	5500.0	sedan	2403	116.0	four	23	

In [31]:

```python
auto_prices.drop_duplicates()
```

Out[31]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mp
0	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
1	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
2	alfa-romero	5000.0	hatchback	2823	154.0	six	19	2
3	audi	5500.0	sedan	2337	102.0	four	24	3
4	audi	5500.0	sedan	2824	115.0	five	18	2
...	...	...	...	...	...	...	...	
200	volvo	5400.0	sedan	2952	114.0	four	23	2
201	volvo	5300.0	sedan	3049	160.0	four	19	2
202	volvo	5500.0	sedan	3012	134.0	six	18	2
203	volvo	4800.0	sedan	3217	106.0	six	26	2
204	volvo	5400.0	sedan	3062	114.0	four	19	2

204 rows × 10 columns

In [32]:

```
auto_prices.drop_duplicates(keep='last')
```

Out[32]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mp
0	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
1	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
2	alfa-romero	5000.0	hatchback	2823	154.0	six	19	2
3	audi	5500.0	sedan	2337	102.0	four	24	3
4	audi	5500.0	sedan	2824	115.0	five	18	2
...	...	...	...	...	...	...	...	
200	volvo	5400.0	sedan	2952	114.0	four	23	2
201	volvo	5300.0	sedan	3049	160.0	four	19	2
202	volvo	5500.0	sedan	3012	134.0	six	18	2
203	volvo	4800.0	sedan	3217	106.0	six	26	2
204	volvo	5400.0	sedan	3062	114.0	four	19	2

204 rows × 10 columns

In [33]:
```
#To find the number of duplicated rows
auto_prices.duplicated().value_counts()
```

Out[33]:
```
False    204
True       1
dtype: int64
```

↥ back to top

9 Treat & Handling missing values

Missing values are a common problem in data set. Failure to deal with missing values before training a machine learning model will lead to biased training at best, and in many cases actual failure. The Python scikit-learn package will not process arrays with missing values.

There are two problems that must be deal with when treating missing values:

1. First you must find the missing values. This can be difficult as there is no standard way missing values are coded. Some common possibilities for missing values are:
   - Coded by some particular character string, or numeric value like -999.
   - A NULL value or numeric missing value such as a NaN.
2. You must determine how to treat the missing values:
   - Remove features with substantial numbers of missing values. In many cases, such features are likely to have little information value.
   - Remove rows with missing values. If there are only a few rows with missing values it might be easier and more certain to simply remove them.
   - Impute values. Imputation can be done with simple algorithms such as replacing the missing values with the mean or median value. There are also complex statistical methods such as the expectation maximization (EM) or SMOTE algorithms.
   - Use nearest neighbor values. Alternatives for nearest neighbor values include, averaging, forward filling or backward filling.

Carefully observe the first few cases from the data frame and notice that missing values are coded with a '?' character. Execute the code in the cell below to identify the columns with missing values.

In [34]:
```
(auto_prices.astype(object) == '?').any()
```

Out[34]:
```
make                 False
peak_rpm             False
body_style           False
curb_weight          False
horsepower           False
num_of_cylinders     False
city_mpg             False
highway_mpg          False
price                False
New_Col              False
dtype: bool
```

In [35]:
```
auto_prices.isnull().sum()
```

Out[35]:

```
make                0
peak_rpm            2
body_style          0
curb_weight         0
horsepower          2
num_of_cylinders    0
city_mpg            0
highway_mpg         0
price               4
New_Col             0
dtype: int64
```

In [36]:

```python
# Replace missing values with 0
auto_prices.fillna(0)
```

Out[36]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mp
0	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
1	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
2	alfa-romero	5000.0	hatchback	2823	154.0	six	19	2
3	audi	5500.0	sedan	2337	102.0	four	24	3
4	audi	5500.0	sedan	2824	115.0	five	18	2
...	...	...	...	...	...	...	...	
200	volvo	5400.0	sedan	2952	114.0	four	23	2
201	volvo	5300.0	sedan	3049	160.0	four	19	2
202	volvo	5500.0	sedan	3012	134.0	six	18	2
203	volvo	4800.0	sedan	3217	106.0	six	26	2
204	volvo	5400.0	sedan	3062	114.0	four	19	2

205 rows × 10 columns

In [37]:

```python
# Fill with median
auto_prices.fillna(auto_prices.median())
```

```
C:\Users\Prateek\AppData\Local\Temp/ipykernel_14600/297449977.py:2: FutureWarning: Dropp
ing of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecate
d; in a future version this will raise TypeError.  Select only valid columns before call
ing the reduction.
  auto_prices.fillna(auto_prices.median())
```

Out[37]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mp
0	alfa-romero	5000.0	convertible	2548	111.0	four	21	2

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mp
1	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
2	alfa-romero	5000.0	hatchback	2823	154.0	six	19	2
3	audi	5500.0	sedan	2337	102.0	four	24	3
4	audi	5500.0	sedan	2824	115.0	five	18	2
...	...	...	...	...	...	...	...	
200	volvo	5400.0	sedan	2952	114.0	four	23	2
201	volvo	5300.0	sedan	3049	160.0	four	19	2
202	volvo	5500.0	sedan	3012	134.0	six	18	2
203	volvo	4800.0	sedan	3217	106.0	six	26	2
204	volvo	5400.0	sedan	3062	114.0	four	19	2

205 rows × 10 columns

In [38]:

```
# dropping the observations
auto_prices.dropna()
```

Out[38]:

	make	peak_rpm	body_style	curb_weight	horsepower	num_of_cylinders	city_mpg	highway_mp
0	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
1	alfa-romero	5000.0	convertible	2548	111.0	four	21	2
2	alfa-romero	5000.0	hatchback	2823	154.0	six	19	2
3	audi	5500.0	sedan	2337	102.0	four	24	3
4	audi	5500.0	sedan	2824	115.0	five	18	2
...	...	...	...	...	...	...	...	
200	volvo	5400.0	sedan	2952	114.0	four	23	2
201	volvo	5300.0	sedan	3049	160.0	four	19	2
202	volvo	5500.0	sedan	3012	134.0	six	18	2
203	volvo	4800.0	sedan	3217	106.0	six	26	2
204	volvo	5400.0	sedan	3062	114.0	four	19	2

199 rows × 10 columns

↥ back to top

10 Create Dummies for a Categorical Variable

Let's convert body_style variable of Car into a Dummy Variable.

In [39]:

```
auto_prices['body_style']
```

Out[39]:

```
0      convertible
1      convertible
2        hatchback
3            sedan
4            sedan
          ...
200          sedan
201          sedan
202          sedan
203          sedan
204          sedan
Name: body_style, Length: 205, dtype: object
```

A **Dummy variable** takes only the value 0 or 1 to indicate the absence or presence of categorical variable.

For example, in first row we can check Convertible is 1 rest all of body_style are 0.

In [40]:

```
pd.get_dummies(auto_prices['body_style'], prefix="D").head(10)
```

Out[40]:

	D_convertible	D_hardtop	D_hatchback	D_sedan	D_wagon
0	1	0	0	0	0
1	1	0	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	1	0
5	0	0	0	1	0
6	0	0	0	1	0
7	0	0	0	0	1
8	0	0	0	1	0
9	0	0	1	0	0

↥ back to top

11 Feature engineering

In most cases, machine learning is not done with the raw features. Features are transformed, or combined to form new features in forms which are more predictive. This process is known as **feature engineering**. In many cases, good feature engineering is more important than the details of the machine learning model used. It is often the case that good features can make even poor machine learning models work well, whereas, given poor features even the best machine learning model will produce poor results. As the famous saying goes, "garbage in, garbage out".

Some common approaches to feature engineering include:

- **Aggregating categories** of categorical variables to reduce the number. Categorical features or labels with too many unique categories will limit the predictive power of a machine learning model. Aggregating categories can improve this situation, sometime greatly. However, one must be careful. It only makes sense to aggregate categories that are similar in the domain of the problem. Thus, domain expertise must be applied.
- **Transforming numeric variables** to improve their distribution properties to make them more covariate with other variables. This process can be applied not only features, but to labels for regression problems. Some common transformations include, **logarithmic** and **power** included squares and square roots.
- **Compute new features** from two or more existing features. These new features are often referred to as **interaction terms**. An interaction occurs when the behavior of say, the produce of the values of two features, is significantly more predictive than the two features by themselves. Consider the probability of purchase for a luxury mens' shoe. This probability depends on the interaction of the user being a man and the buyer being wealthy. As another example, consider the number of expected riders on a bus route. This value will depend on the interaction between the time of day and if it is a holiday.

➕➕➕ **We will cover feature engineering and transforming variables in further modules.**

↥ back to top

12 Summary

Good data preparation is the key to good machine learning performance.

Data preparation or data munging is a time interactive and iterative process.

Continue to visualize the results as you test ideas. Expect to try many approaches, reject the ones that do not help, and keep the ones that do.

In summary, test a lot of ideas, fail fast, keep what works. The reward is that well prepared data can improve the performance of almost any machine learning algorithm.

↥ back to top

Great Job!
