Advanced Machine Learning

CIS550

Spring ‘24

Lab Homework 3

<u>Submitted by :</u>

Name : Manas Vishal (01971464)

Email : mvishal@umassd.edu

Title : Encoding of ordinal and non-ordinal categorical data

Encoding the data

When data researchers take data, it can often be non-ordinal which means it might not have any numerical value associated with it. For those type of data, machine learning engineers and data scientists have to encode them properly and assign numerical values to it. This is called encoding of data.

About the data

In this exercise I am working with automobile dataset which I got from here : [Automobile - UCI Machine Learning Repository]

The data has following attributes :

Attribute: Attribute Range

1. symboling: -3, -2, -1, 0, 1, 2, 3.
2. normalized-losses: continuous from 65 to 256.
3. make: alfa-romero, audi, bmw, chevrolet, dodge, honda, isuzu, jaguar, mazda, mercedes-benz, mercury, mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota, volkswagen, volvo
4. fuel-type: diesel, gas.
5. aspiration: std, turbo.
6. num-of-doors: four, two.
7. body-style: hardtop, wagon, sedan, hatchback, convertible.
8. drive-wheels: 4wd, fwd, rwd.
9. engine-location: front, rear.
10. wheel-base: continuous from 86.6 120.9.
11. length: continuous from 141.1 to 208.1.
12. width: continuous from 60.3 to 72.3.
13. height: continuous from 47.8 to 59.8.
14. curb-weight: continuous from 1488 to 4066.
15. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
16. num-of-cylinders: eight, five, four, six, three, twelve, two.
17. engine-size: continuous from 61 to 326.
18. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
19. bore: continuous from 2.54 to 3.94.
20. stroke: continuous from 2.07 to 4.17.
21. compression-ratio: continuous from 7 to 23.
22. horsepower: continuous from 48 to 288.
23. peak-rpm: continuous from 4150 to 6600.
24. city-mpg: continuous from 13 to 49.
25. highway-mpg: continuous from 16 to 54.
26. price: continuous from 5118 to 45400.

Fig. 1 Data and its attributes

```python
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

```

[1] Python

Fig. 2 Importing necessary modules and setting parameters for pandas

The dataset is downloaded by using the .get function from requests module.

```python
f_zip = 'https://archive.ics.uci.edu/static/public/10/automobile.zip'
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()
```

[2] Python

Fig. 2 Downloading and extracting the data locally

The extracted data is unstructured in .data and .names files so we need to pack it in dataframe using the column names given.

```python
column_names = ['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']
data_list =[]
data = ["imports-85.data"]

for file in data:
    df_car = pd.read_csv(file, names = column_names)
    data_list.append(df_car)
```

[3] Python

Fig. 3 Generating a dataframe using the data and column names

We examine the data by checking its shape and head

```
[4]  1  df_car.shape
     ✓ 0.0s                                   Python
```

```
(205, 26)
```

```
[5]  1  df_car.head(5)
     ✓ 0.0s                                   Python
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	eng loca
0	3	?	alfa-romero	gas	std	two	convertible	rwd	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	
3	2	164	audi	gas	std	four	sedan	fwd	
4	2	164	audi	gas	std	four	sedan	4wd	

Fig. 4 Printing off the data and its values on screen

To get more information about the dataframe we use the info function

```python
df_car.info()
```

[5] Python

```
<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 
...
 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
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Fig. 5 Data Info

I trim the data that I will not be using by reassigning the dataframe into a new variable and using the column names that I want to work on.

```
df_car.columns
```

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

```
df_car_new = df_car[[ 'aspiration', 'num-of-doors','drive-wheels','num-of-cylinders']]
```

```
df_car_new.head()
```

	aspiration	num-of-doors	drive-wheels	num-of-cylinders
0	std	two	rwd	four
1	std	two	rwd	four
2	std	two	rwd	six
3	std	four	fwd	four
4	std	four	4wd	five

Fig. 6 Trimming the data

Now we can work on this subset of data for our lab. We can encode the following data.

We can see the num-of-cylinders field is **text but does have an ordinal value** so we will have to make it ordered by assigning numerical values to it.

While for aspiration and drive-wheels, the data is not ordinal. They have to be mapped to a numerical value somehow.

Encoding the ordinal data

We start with first getting the info for the new data.

```
[9]  1  df_car_new.info()                                         Python

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   aspiration        205 non-null    object
 1   num-of-doors      205 non-null    object
 2   drive-wheels      205 non-null    object
 3   num-of-cylinders  205 non-null    object
dtypes: object(4)
memory usage: 6.5+ KB
```

Fig. 7 Subset of data

We can see from the screenshot above that all the columns have 205 values. Lets work on the num-of-doors field for now.

We print off the value counts for the num-of-doors field/feature.

```
[15]  1  df_car_new['num-of-doors'].value_counts()               Python
      ✓ 0.0s

num-of-doors
four    114
two      89
?         2
Name: count, dtype: int64
```

Fig. 8 Value counts of the num-of-doors feature

This feature has three values : four, two and unknown. We can create a simple mapper and we can assign numerics for all. I choose unknown to be 0.

```python
door_mapper=  {"two": 2,
"four": 4,
"?":0}
```

[16] ✓ 0.0s Python

```python
df_car_new['doors'] = df_car_new["num-of-doors"].replace(door_mapper)
```

[18] ✓ 0.0s Python

```python
df_car_new.head()
```

[19] ✓ 0.0s Python

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors
0	std	two	rwd	four	2
1	std	two	rwd	four	2
2	std	two	rwd	six	2
3	std	four	fwd	four	4
4	std	four	4wd	five	4

Fig. 9 Remapping to encode the dataframe using replace function and a remapper dictionary

We repeat the same process for another field **num-of-cylinders**

+ Code + Markdown

```python
df_car_new['num-of-cylinders'].value_counts()
```

[20] ✓ 0.0s Python

```
num-of-cylinders
four       159
six         24
five        11
eight        5
two          4
three        1
twelve       1
Name: count, dtype: int64
```

Fig. 10 Getting the values

```python
cylinder_mapper = {"two":2,
                   "three":3,
                   "four":4,
                   "five":5,
                   "six":6,
                   "eight":8,
                   "twelve":12}
```

Fig. 11 Remapping dictionary

```python
df_car_new['cylinders'] = df_car_new['num-of-cylinders'].replace(cylinder_mapper)
```

```python
df_car_new.head()
```

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors	cylinders
0	std	two	rwd	four	2	4
1	std	two	rwd	four	2	4
2	std	two	rwd	six	2	6
3	std	four	fwd	four	4	4
4	std	four	4wd	five	4	5

Fig. 12 Remapped-encoded dataframe

Encoding the non-ordinal data

In this dataframe, we have a few columns that are non-ordinal. We can use the get_dummies function of pandas to encode these features and add new binary features to dataframe.

For e.g., the drive-wheels feature can take 4wd, fwd,rwd as values. As mentioned in the lab, one can think of assigning some number to these features but by doing that we add extra order to the system which is not present at all. That's why we have to rely on binary for these type of data. They are represented well as binary features. This process is called one-hot encoding or dummying statistics.

```
[18]  1  df_car_new['drive-wheels'].value_counts()
                                                        Python
```

```
drive-wheels
fwd     120
rwd      76
4wd       9
Name: count, dtype: int64
```

Fig. 13 Value counts of drive-wheels feature

As expected from the attribute table, we have three possible values of drive-wheels

```
[19]  1  df_car_new = pd.get_dummies(df_car_new,columns=['drive-wheels'])
                                                        Python
```

Fig. 14 Using the get_dummies function to convert the feature to binary

```
1 df_car_new.head()
```

[20]

Python

	aspiration	num-of-doors	num-of-cylinders	doors	cylinders	drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd
0	std	two	four	2	4	False	False	True
1	std	two	four	2	4	False	False	True
2	std	two	six	2	6	False	False	True
3	std	four	four	4	4	False	True	False
4	std	four	five	4	5	True	False	False

Fig. 15 Encoded dataframe of non-ordinal data

We can see there are 3 new columns corresponding the 3 values of the feature we expected. These new columns are either True or False which means they are binary. In short we are unpacking the num-of-wheels column into three separate binary columns.

We can repeat the same thing for aspiration column. We could do the same thing as we did for above feature. But here, the data is either std or turbo, so we get to choose which feature gets to be mapped with True or False. Here, I will use the drop_first function and just choose turbo as true or false which means if the engine is turbo, it will get mapped to 1 or True.

```python
df_car_new['aspiration'].value_counts()
```

```
aspiration
std      168
turbo     37
Name: count, dtype: int64
```

```python
df_car_new = pd.get_dummies(df_car_new,columns=['aspiration'], drop_first=True)
```

```python
df_car_new.head()
```

	num-of-doors	num-of-cylinders	doors	cylinders	drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd	aspiration_turbo
0	two	four	2	4	False	False	True	False
1	two	four	2	4	False	False	True	False
2	two	six	2	6	False	False	True	False
3	four	four	4	4	False	True	False	False
4	four	five	4	5	True	False	False	False

```python
df_car_new[["aspiration_turbo"]][0:9]
```

	aspiration_turbo
0	False
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	True

Fig. 16 Encoded data for aspiration feature

Challenge Task

For the challenge task, I am adding three new columns : `'normalized-losses'`,`'make'`, `'fuel-type'`

First field of normalized-losses does not need any encoding as it is a continuous attribute. However, it has missing values so we need to remap those values to either zero or some other constant.

```
1  df_car_new = df_car[[ 'aspiration', 'num-of-doors','drive-wheels','num-of-cylinders','normalized-losses','make', 'fuel-type']]
```

[20] ✓ 0.0s Python

```
1  df_car_new.head()
```

[22] ✓ 0.0s Python

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	normalized-losses	make	fuel-type
0	std	two	rwd	four	?	alfa-romero	gas
1	std	two	rwd	four	?	alfa-romero	gas
2	std	two	rwd	six	?	alfa-romero	gas
3	std	four	fwd	four	164	audi	gas
4	std	four	4wd	five	164	audi	gas

Fig. 17 Adding three new columns

The make field has 22 available options.

```
1  df_car_new['make'].value_counts()
```

[24] ✓ 0.0s Python

```
make
toyota           32
nissan           18
mazda            17
mitsubishi       13
honda            13
volkswagen       12
subaru           12
peugot           11
volvo            11
dodge             9
mercedes-benz     8
bmw               8
audi              7
plymouth          7
saab              6
porsche           5
isuzu             4
jaguar            3
chevrolet         3
alfa-romero       3
renault           2
mercury           1
Name: count, dtype: int64
```

Here, we will use the get_dummies function to assign binary feature to the new dataframe.

```
1  df_car_new=pd.get_dummies(df_car_new,columns=['make'])
```
[16] ✓ 0.0s Python

```
1  df_car_new.info()
```
[25] ✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 28 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   aspiration          205 non-null    object
 1   num-of-doors        205 non-null    object
 2   drive-wheels        205 non-null    object
 3   num-of-cylinders    205 non-null    object
 4   normalized-losses   205 non-null    object
 5   fuel-type           205 non-null    object
 6   make_alfa-romero    205 non-null    bool  
 7   make_audi           205 non-null    bool  
 8   make_bmw            205 non-null    bool  
 9   make_chevrolet      205 non-null    bool  
 10  make_dodge          205 non-null    bool  
 11  make_honda          205 non-null    bool  
 12  make_isuzu          205 non-null    bool  
 13  make_jaguar         205 non-null    bool  
 14  make_mazda          205 non-null    bool  
 15  make_mercedes-benz  205 non-null    bool  
 16  make_mercury        205 non-null    bool  
 17  make_mitsubishi     205 non-null    bool  
 18  make_nissan         205 non-null    bool  
 19  make_peugot         205 non-null    bool  
...
 26  make_volkswagen     205 non-null    bool  
 27  make_volvo          205 non-null    bool  
dtypes: bool(22), object(6)
memory usage: 14.1+ KB
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
1  df_car_new[0:10]
```
[23] ✓ 0.0s Python

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	normalized-losses	fuel-type	make_alfa-romero	make_audi	make_bmw	make_chevrolet
0	std	two	rwd	four	?	gas	True	False	False	False
1	std	two	rwd	four	?	gas	True	False	False	False
2	std	two	rwd	six	?	gas	True	False	False	False
3	std	four	fwd	four	164	gas	False	True	False	False
4	std	four	4wd	five	164	gas	False	True	False	False
5	std	two	fwd	five	?	gas	False	True	False	False
6	std	four	fwd	five	158	gas	False	True	False	False
7	std	four	fwd	five	?	gas	False	True	False	False
8	turbo	four	fwd	five	158	gas	False	True	False	False
9	turbo	two	4wd	five	?	gas	False	True	False	False

Fig. 18 New dataframe with extra Boolean/binary features associated to the make column

For the fuel-type column, I use drop_first function to directly the column into fule-type_gas column which is a Boolean/binary feature..

```
[12]
df_car_new['fuel-type'].value_counts()
✓ 0.0s                                                        Python
```

```
fuel-type
gas       185
diesel     20
Name: count, dtype: int64
```

```
[13]
df_car_new = pd.get_dummies(df_car_new,columns=['fuel-type'], drop_first=True)
✓ 0.0s                                                        Python
```

```
[14]
df_car_new.info()
✓ 0.0s                                                        Python
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   aspiration         205 non-null    object
 1   num-of-doors       205 non-null    object
 2   drive-wheels       205 non-null    object
 3   num-of-cylinders   205 non-null    object
 4   normalized-losses  205 non-null    object
 5   make               205 non-null    object
 6   fuel-type_gas      205 non-null    bool  
dtypes: bool(1), object(6)
memory usage: 9.9+ KB
```

```
[21]
df_car_new[-21:-1]
✓ 0.0s                                                        Python
```

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	normalized-losses	make	fuel-type_gas
184	std	four	fwd	four	94	volkswagen	False
185	std	four	fwd	four	94	volkswagen	True
186	std	four	fwd	four	94	volkswagen	True
187	turbo	four	fwd	four	94	volkswagen	False
188	std	four	fwd	four	94	volkswagen	True
189	std	two	fwd	four	?	volkswagen	True
190	std	two	fwd	four	256	volkswagen	True
191	std	four	fwd	five	?	volkswagen	True
192	turbo	four	fwd	four	?	volkswagen	False
193	std	four	fwd	four	?	volkswagen	True
194	std	four	rwd	four	103	volvo	True
195	std	four	rwd	four	74	volvo	True
196	std	four	rwd	four	103	volvo	True
197	std	four	rwd	four	74	volvo	True
198	turbo	four	rwd	four	103	volvo	True
199	turbo	four	rwd	four	74	volvo	True
200	std	four	rwd	four	95	volvo	True
201	turbo	four	rwd	four	95	volvo	True
202	std	four	rwd	six	95	volvo	True
203	turbo	four	rwd	six	95	volvo	False

Fig. 19 Encoding the non-ordinal data

Conclusion

In this lab, I learnt a lot about data types in data mining. Preprocessing of data is very important as if not done correctly, it can lead to incorrect or even irrelevant data analysis/prediction from machine learning. The data is acquired from different sources and can be packed in different ways. In our case, the dataset we worked on today was also packed non-ordinally for some features. It is very important to encode it properly to be passed in the machine learning architecture. We learnt ways to encode the non-ordinal as well as ordinal data.

I look forward to using these methods in future to encode raw dataset to feed into machine learning algorithms.

Edit : New update to the homework for returning 0 and 1's instead of Boolean in the challenge task

Passing an extra argument to the get_dummies function will return 0 and 1's instead of Boolean

```python
df_car_new = pd.get_dummies(df_car_new,
                            columns=['fuel-type'],
                            drop_first=True,dtype=int)
```

[44] ✓ 0.0s Python

```python
df_car_new.info()
```

[30] ✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   aspiration         205 non-null    object
 1   num-of-doors       205 non-null    object
 2   drive-wheels       205 non-null    object
 3   num-of-cylinders   205 non-null    object
 4   normalized-losses  205 non-null    object
 5   make               205 non-null    object
 6   fuel-type_gas      205 non-null    int64 
dtypes: int64(1), object(6)
memory usage: 11.3+ KB
```

```python
1 df_car_new[-21:-1]
```

[31] ✓ 0.0s Python

...

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	normalized-losses	make	fuel-type_gas
184	std	four	fwd	four	94	volkswagen	0
185	std	four	fwd	four	94	volkswagen	1
186	std	four	fwd	four	94	volkswagen	1
187	turbo	four	fwd	four	94	volkswagen	0
188	std	four	fwd	four	94	volkswagen	1
189	std	two	fwd	four	?	volkswagen	1
190	std	two	fwd	four	256	volkswagen	1
191	std	four	fwd	five	?	volkswagen	1
192	turbo	four	fwd	four	?	volkswagen	0
193	std	four	fwd	four	?	volkswagen	1
194	std	four	rwd	four	103	volvo	1
195	std	four	rwd	four	74	volvo	1
196	std	four	rwd	four	103	volvo	1
197	std	four	rwd	four	74	volvo	1
198	turbo	four	rwd	four	103	volvo	1
199	turbo	four	rwd	four	74	volvo	1
200	std	four	rwd	four	95	volvo	1
201	turbo	four	rwd	four	95	volvo	1
202	std	four	rwd	six	95	volvo	1
203	turbo	four	rwd	six	95	volvo	0

Fig. Updated challenge task