C2_W4_lecture

June 2, 2020

1 AI4M Course 2 Week 4 lecture notebook

1.1 Outline

Section **??**

Section **??**

Section **??**

One-hot encode categorical variables

[1]:
```
import pandas as pd
```

1.1.1 Which features are categorical?

[2]:
```
df = pd.DataFrame({'ascites': [0,1,0,1],
                   'edema': [0.5,0,1,0.5],
                   'stage': [3,4,3,4],
                   'cholesterol': [200.5,180.2,190.5,210.3]
                  })
df
```

[2]:

	ascites	edema	stage	cholesterol
0	0	0.5	3	200.5
1	1	0.0	4	180.2
2	0	1.0	3	190.5
3	1	0.5	4	210.3

In this small sample dataset, 'ascites', 'edema', and 'stage' are categorical variables - ascites: value is either 0 or 1 - edema: value is either 0, 0.5 or 1 - stage: is either 3 or 4

'cholesterol' is a continuous variable, since it can be any decimal value greater than zero.

1.1.2 Which categorical variables to one-hot encode?

Of the categorical variables, which one should be one-hot encoded (turned into dummy variables)?

- ascites: is already 0 or 1, so there is not a need to one-hot encode it.
  - We could one-hot encode ascites, but it is not necessary when there are just two possible values that are 0 or 1.
  - When values are 0 or 1, 1 means a disease is present, and 0 means normal (no disease).
- edema: Edema is swelling in any part of the body. This data set's 'edema' feature has 3 categories, so we will want to one-hot encode it so that there is one feature column for each of the three possible values.
  - 0: No edema
  - 0.5: Patient has edema, but did not receive diuretic therapy (which is used to treat edema)
  - 1: Patient has edeam, despite also receiving diuretic therapy (so the condition may be more severe).
- stage: has values of 3 and 4. We will want to one-hot encode these because they are not values of 0 or 1.
  - the "stage" of cancer is either 0, 1,2,3 or 4.
  - Stage 0 means there is no cancer.
  - Stage 1 is cancer that is limited to a small area of the body, also known as "early stage cancer"
  - Stage 2 is cancer that has spread to nearby tissues
  - stage 3 is cancer that has spread to nearby tissues, but more so than stage 2
  - stage 4 is cancer that has spread to distant parts of the body, also known as "metastatic cancer".
  - We could convert stage 3 to 0 and stage 4 to 1 for the sake of training a model. This would may be confusing for anyone reviewing our code and data. We will one-hot encode the 'stage'. -You'll actually see that we end up with 0 representing stage 3 and 1 representing stage 4 (see the next section).

1.1.3 Multi-collinearity of one-hot encoded features

Let's see what happens when we one-hot encode the 'stage' feature.

We'll use pandas.get_dummies

[3]:
```
df_stage = pd.get_dummies(data=df,
                          columns=['stage']
                         )
df_stage[['stage_3','stage_4']]
```

[3]:

	stage_3	stage_4
0	1	0
1	0	1
2	1	0
3	0	1

What do you notice about the 'stage_3' and 'stage_4' features?

Given that stage 3 and stage 4 are the only possible values for stage,
If you know that patient 0 (row 0) has stage_3 set to 1, what can you say about that same patient's value for the stage_4 feature? - When stage_3 is 1, then stage_4 must be 0 - When stage_3 is 0, then stage_4 must be 1

This means that one of the feature columns is actually redundant. We should drop one of these features to avoid multicollinearity (where one feature can predict another feature).

[4]:
```
df_stage
```

[4]:

	ascites	edema	cholesterol	stage_3	stage_4
0	0	0.5	200.5	1	0
1	1	0.0	180.2	0	1
2	0	1.0	190.5	1	0
3	1	0.5	210.3	0	1

[5]:
```
df_stage_drop_first = df_stage.drop(columns='stage_3')
df_stage_drop_first
```

[5]:

	ascites	edema	cholesterol	stage_4
0	0	0.5	200.5	0
1	1	0.0	180.2	1
2	0	1.0	190.5	0
3	1	0.5	210.3	1

Note, there's actually a parameter of pandas.get_dummies() that lets you drop the first one-hot encoded column. You'll practice doing this in this week's assignment!

1.1.4 Make the numbers decimals

We can cast the one-hot encoded values as floats by setting the data type to numpy.float64. - This is helpful if we are feeding data into a model, where the model expects a certain data type (such as a 64-bit float, 32-bit float etc.)

[6]:
```
import numpy as np
```

[7]:
```
df_stage = pd.get_dummies(data=df,
                          columns=['stage'],
                         )
df_stage[['stage_4']]
```

[7]:

	stage_4
0	0
1	1
2	0
3	1

[8]:
```
df_stage_float64 = pd.get_dummies(data=df,
                columns=['stage'],
                dtype=np.float64
               )
df_stage_float64[['stage_4']]
```

[8]:

	stage_4
0	0.0
1	1.0
2	0.0
3	1.0

1.1.5 This is the end of this practice section.

Please continue on with the lecture videos!

Hazard function

Let's say we fit the hazard function

$$\lambda(t, x) = \lambda_0(t)e^{\theta^T X_i}$$

So that we have the coefficients θ for the features in X_i

If you have a new patient, let's predict their hazard $\lambda(t, x)$

[9]:
```
import numpy as np
import pandas as pd
```

[10]:
```
lambda_0 = 1
coef = np.array([0.5,2.])
coef
```

[10]:
```
array([0.5, 2. ])
```

[11]:
```
X = pd.DataFrame({'age': [20,30,40],
                  'cholesterol': [180,220,170]
                 })
X
```

[11]:

	age	cholesterol
0	20	180
1	30	220
2	40	170

- First, let's multiply the coefficients to the features.
- Check the shapes of the coefficients and the features to decide which one to transpose

[12]:
```
coef.shape
```

[12]:
```
(2,)
```

[13]:
```
X.shape
```

[13]:
```
(3, 2)
```

It looks like the coefficient is a 1D array, so transposing it won't do anything.
- We can transpose the X so that we're multiplying a (2,) array by a (2,3) dataframe.

So the formula looks more like this (transpose X_i instead of θ

$$\lambda(t, x) = \lambda_0(t)e^{\theta X_i^T}$$

- Let's multiply θX_i^T

[14]:
```
np.dot(coef,X.T)
```

[14]:
```
array([370., 455., 360.])
```

Calculate the hazard for the three patients (there are 3 rows in X)

[15]:
```
lambdas = lambda_0 * np.exp(np.dot(coef,X.T))
print(lambdas.shape)
patients_df = X.copy()
patients_df['hazards'] = lambdas
patients_df
```

```
(3,)
```

[15]:

	age	cholesterol	hazards
0	20	180	4.886054e+160
1	30	220	4.017809e+197
2	40	170	2.218265e+156

1.1.6 This is the end of this practice section.

Please continue on with the lecture videos!

Permissible pairs with censoring and time

[16]:
```
import pandas as pd
```

[17]:
```
df = pd.DataFrame({'time': [2,4,2,4,2,4,2,4],
                   'event': [1,1,1,1,0,1,1,0],
                   'risk_score': [20,40,40,20,20,40,40,20]
                  })
```

```
df
```

[17]:

	time	event	risk_score
0	2	1	20
1	4	1	40
2	2	1	40
3	4	1	20
4	2	0	20
5	4	1	40
6	2	1	40
7	4	0	20

We made this data sample so that you can compare pairs of patients visually.

1.1.7 When at least one patient is not censored

- A pair may be permissible if at least one patient is not censored.
- If both pairs of patients are censored, then they are definitely not a permissible pair.

[18]:
```
pd.concat([df.iloc[0:1],df.iloc[1:2]],axis=0)
```

[18]:

	time	event	risk_score
0	2	1	20
1	4	1	40

[19]:
```
if df['event'][0] == 1 or df['event'][1] == 1:
    print(f"May be a permissible pair: 0 and 1")
else:
    print(f"Definitely not permissible pair: 0 and 1")
```

```
May be a permissible pair: 0 and 1
```

[20]:
```
pd.concat([df.iloc[4:5],df.iloc[7:8]],axis=0)
```

[20]:

	time	event	risk_score
4	2	0	20
7	4	0	20

[21]:
```
if df['event'][4] == 1 or df['event'][7] == 1:
    print(f"May be a permissible pair: 4 and 7")
else:
    print(f"Definitely not permissible pair: 4 and 7")
```

```
Definitely not permissible pair: 4 and 7
```

1.1.8 If neither patient was censored:

- If both patients had an event (neither one was censored). This is definitely a permissible pair.

[22]:
```
pd.concat([df.iloc[0:1],df.iloc[1:2]],axis=0)
```

[22]:
	time	event	risk_score
0	2	1	20
1	4	1	40

[23]:
```
if df['event'][0] == 1 and df['event'][1] == 1:
    print(f"Definitely a permissible pair: 0 and 1")
else:
    print(f"May be a permissible pair: 0 and 1")
```

```
Definitely a permissible pair: 0 and 1
```

1.1.9 When one patient is censored:

- If we know that one patient was censored and one had an event, then we can check if censored patient's time is at least as great as the uncensored patient's time. If so, it's a permissible pair as well

[24]:
```
pd.concat([df.iloc[6:7],df.iloc[7:8]],axis=0)
```

[24]:
	time	event	risk_score
6	2	1	40
7	4	0	20

[25]:
```
if df['time'][7] >= df['time'][6]:
    print(f"Permissible pair: Censored patient 7 lasted at least as long as␣
 ↪uncensored patient 6")
else:
    print("Not a permisible pair")
```

```
Permissible pair: Censored patient 7 lasted at least as long as uncensored
patient 6
```

[26]:
```
pd.concat([df.iloc[4:5],df.iloc[5:6]],axis=0)
```

[26]:
	time	event	risk_score
4	2	0	20
5	4	1	40

[27]:
```
if df['time'][4] >= df['time'][5]:
    print(f"Permissible pair")
else:
```

```
    print("Not a permisible pair: censored patient 4 was censored before␣
↪patient 5 had their event")
```

```
Not a permisible pair: censored patient 4 was censored before patient 5 had
their event
```

1.1.10 This is the end of this practice section.

Please continue on with the lecture videos!

[ ]: