splitting

October 29, 2023

A

1 Implementare una procedura di Data Splitting per il calcolo dei p-value onesti.

Domanda: Supponendo di voler sapere quanto guadagna un battitore in base alle sue statistiche, vogliamo calcolare dei p-value onesti per verificare la significatività dei parametri.

Nel csv Hitters.csv ci sono i dati di circa 300 battitori e 19 statistiche più lo stipendio.

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

[ ]:
```
hitters = pd.read_csv(
    "Hitters.csv"
)
print(hitters.shape)
hitters.head()
```

```
(322, 20)
```

[ ]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	\
0	293	66	1	30	29	14	1	293	66	1	30	
1	315	81	7	24	38	39	14	3449	835	69	321	
2	479	130	18	66	72	76	3	1624	457	63	224	
3	496	141	20	65	78	37	11	5628	1575	225	828	
4	321	87	10	39	42	30	2	396	101	12	48	

	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
0	29	14	A	E	446	33	20	NaN	A
1	414	375	N	W	632	43	10	475.0	N
2	266	263	A	W	880	82	14	480.0	A
3	838	354	N	E	200	11	3	500.0	N
4	46	33	N	E	805	40	4	91.5	N

1.1 Preparazione dei dati

Abbiamo 3 variabili categoriche:

- League

- Division
- NewLeague

Verifichiamo quante classi ci sono in queste variabili e creiamo variabili dummies di conseguenza, escludendone una come riferimento.

[ ]:
```
print(f"League: {list(hitters['League'].unique())}")
print(f"Division: {list(hitters['Division'].unique())}")
print(f"NewLeague: {list(hitters['NewLeague'].unique())}")
```

```
League: ['A', 'N']
Division: ['E', 'W']
NewLeague: ['A', 'N']
```

Adesso che sappiamo le categorie possiamo aggiungere le dummies.

Essendo solo due classi a variabile categorica possiamo anche decidere di rinominare la colonna, ad esempio: **League -> League_A** e mettere 1 se League è A e 0 altrimenti. Il ragionamento può essere iterato anche alle altre variabili.

[ ]:
```
hitters.rename(columns={'League': 'League_A', 'Division':␣
  ↪'Division_E','NewLeague': 'NewLeague_A'}, inplace=True)
hitters["League_A"] = np.where(hitters["League_A"] == "A", 1, 0)
hitters["Division_E"] = np.where(hitters["Division_E"] == "E", 1, 0)
hitters["NewLeague_A"] = np.where(hitters["NewLeague_A"] == "A", 1, 0)

categories = ["League_A","Division_E","NewLeague_A"]
```

[ ]:
```
hitters.dropna(inplace=True)
print(hitters.shape)
hitters.reset_index(inplace=True,drop=True)
hitters.head()
```

```
(263, 20)
```

[ ]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
0	315	81	7	24	38	39	14	3449	835	69	321
1	479	130	18	66	72	76	3	1624	457	63	224
2	496	141	20	65	78	37	11	5628	1575	225	828
3	321	87	10	39	42	30	2	396	101	12	48
4	594	169	4	74	51	35	11	4408	1133	19	501

	CRBI	CWalks	League_A	Division_E	PutOuts	Assists	Errors	Salary
0	414	375	0	0	632	43	10	475.0
1	266	263	1	0	880	82	14	480.0
2	838	354	0	1	200	11	3	500.0
3	46	33	0	1	805	40	4	91.5
4	336	194	1	0	282	421	25	750.0

	NewLeague_A

```
0            0
1            1
2            0
3            0
4            1
```

Adesso dividiamo il dataset e lo standardizziamo

[ ]:
```
import statsmodels.api as sm

y = hitters["Salary"]
X = hitters.loc[:,hitters.columns != y.name]
categories_df = hitters.loc[:, hitters.columns.isin(categories)]
X = (X - X.mean())/X.std()
X[categories] = categories_df
```

1.2 Split del dataset

Solo a scopo d'esempio prendiamo la prima metà per il LASSO e la seconda metà per OLS con i parametri non nulli del LASSO

[ ]:
```
half_df = len(X) // 2

lasso_df = X.iloc[:half_df , :]
lasso_response = y.iloc[:half_df]

ols_df = X.iloc[half_df: , :]
ols_response = y.iloc[half_df:]
ols_df = sm.add_constant(ols_df)
```

Adesso utilizziamo il **LASSO** per selezionare solo le variabili significative. Omettimo il calcolo di λ (α in questo caso) ottenibile per CrossValidation Per tale scopo è possibile utilizzare sklearn.

[ ]:
```
#Dichiarazione dell'oggetto modello OLS
model = sm.OLS(lasso_response, lasso_df)

lasso = model.fit_regularized(
    method='elastic_net',
    alpha=5,
    L1_wt=1.0, #Il peso a 1 è equivalente al LASSO, se lo mettiamo a 0 equivale␣
 ↪a RIDGE
)
parameters = lasso.params
print(parameters)
```

```
AtBat            0.000000
Hits            77.305082
HmRun          -65.168593
Runs             0.000000
```

```
RBI              0.000000
Walks           99.593163
Years            0.000000
CAtBat           0.000000
CHits          107.214691
CHmRun         215.678132
CRuns            0.000000
CRBI           110.292574
CWalks        -203.887257
League_A       145.164193
Division_E     389.534130
PutOuts         76.148036
Assists          0.000000
Errors           0.000000
NewLeague_A    194.722584
dtype: float64
```

[ ]:
```
parameters = parameters[parameters != 0]
to_Use = parameters.index.tolist()
print(f"Parametri da usare: {to_Use}")
```

```
Parametri da usare: ['Hits', 'HmRun', 'Walks', 'CHits', 'CHmRun', 'CRBI',
'CWalks', 'League_A', 'Division_E', 'PutOuts', 'NewLeague_A']
```

[ ]:
```
ols_df = ols_df.loc[:, ols_df.columns.isin(to_Use)]
ols_df = sm.add_constant(ols_df)
ols = sm.OLS(ols_response, ols_df)
results = ols.fit()

results.summary()
```

[ ]:

Dep. Variable:	Salary	**R-squared:**	0.398
Model:	OLS	**Adj. R-squared:**	0.343
Method:	Least Squares	**F-statistic:**	7.219
Date:	Sun, 29 Oct 2023	**Prob (F-statistic):**	2.43e-09
Time:	16:09:19	**Log-Likelihood:**	-951.98
No. Observations:	132	**AIC:**	1928.
Df Residuals:	120	**BIC:**	1963.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
const	510.2050	58.021	8.793	0.000	395.327	625.083
Hits	-0.6886	46.670	-0.015	0.988	-93.091	91.714
HmRun	42.2921	46.344	0.913	0.363	-49.465	134.050
Walks	107.5083	52.318	2.055	0.042	3.923	211.093
CHits	215.0919	171.479	1.254	0.212	-124.425	554.608
CHmRun	-80.5488	148.519	-0.542	0.589	-374.607	213.509
CRBI	88.7700	248.818	0.357	0.722	-403.873	581.412
CWalks	-54.2604	115.034	-0.472	0.638	-282.019	173.498
League_A	-158.0517	124.150	-1.273	0.205	-403.860	87.757
Division_E	128.2603	61.447	2.087	0.039	6.600	249.920
PutOuts	49.6161	30.282	1.638	0.104	-10.340	109.572
NewLeague_A	73.6801	121.190	0.608	0.544	-166.267	313.627

Omnibus:	80.367	**Durbin-Watson:**	2.085
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	585.163
Skew:	1.996	**Prob(JB):**	8.58e-128
Kurtosis:	12.511	**Cond. No.**	19.6

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

2 References

1. Statsmodels
2. An Introduction to statistical learning