Python Tutorial 8

March 10, 2022

This tutorial is for Dr. Xin Tong's DSO 530 class at the University of Southern California in Spring 2022. It contains two parts: the first part is CV for classification problems and the second part is CV for regression problems.

1 Cross-validation for classification

In this part, we demonstrate k-Fold cross-validation using classification error and AUC with an *auto* classification example. First, we will present a quite generic way which you can use for CV beyond the `sklearn` packages. Then we will present a quick way to do CV with a `sklearn` package.

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

Auto = pd.read_csv('auto.csv')
Auto.head()
```

[1]:
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0    70
1  15.0          8         350.0         165    3693          11.5    70
2  18.0          8         318.0         150    3436          11.0    70
3  16.0          8         304.0         150    3433          12.0    70
4  17.0          8         302.0         140    3449          10.5    70

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
```

`displacement` represents a vehicle's engine displacement. First, we transform it into a binary variable `displacement_binary` in two steps: 1) we calculate the mean of `displacement`; 2) we compare each value of `displacement` with the mean of `displacement`. If it is smaller than or equal to the mean, we label it as `small`. Otherwise, we label it as `big`. Then we use `displacement_binary` as the responses to do the classification.

[2]:
```
small_index = Auto["displacement"] <= np.mean(Auto["displacement"])
Auto.loc[small_index,"displacement_binary"] = 'small'
```

```
Auto.loc[~small_index,"displacement_binary"] = 'big'
```

Note that we still need to add a column name `displacement_big` to represent `displacement_binary` and make it numeric if we want to use `smf.logit` to do logistic regression.

[3]:
```
Auto["displacement_big"] = np.where(Auto["displacement_binary"] == 'big', 1, 0)
```

[4]:
```
Auto
```

[4]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0         130    3504          12.0    70
1    15.0          8         350.0         165    3693          11.5    70
2    18.0          8         318.0         150    3436          11.0    70
3    16.0          8         304.0         150    3433          12.0    70
4    17.0          8         302.0         140    3449          10.5    70
..    …          …           …          …      …           …   …
387  27.0          4         140.0          86    2790          15.6    82
388  44.0          4          97.0          52    2130          24.6    82
389  32.0          4         135.0          84    2295          11.6    82
390  28.0          4         120.0          79    2625          18.6    82
391  31.0          4         119.0          82    2720          19.4    82

     origin                       name displacement_binary  displacement_big
0         1  chevrolet chevelle malibu                 big                 1
1         1          buick skylark 320                 big                 1
2         1         plymouth satellite                 big                 1
3         1              amc rebel sst                 big                 1
4         1                ford torino                 big                 1
..      …                        …                 …               …
387       1            ford mustang gl               small                 0
388       2                 vw pickup               small                 0
389       1              dodge rampage               small                 0
390       1                ford ranger               small                 0
391       1                  chevy s-10               small                 0

[392 rows x 11 columns]
```

We use 10-fold CV to compare two logistic regression models that use different predictors.

First, we use `mpg` and `horsepower` as predictor variables and use `displacement_big` as the response variable.

[5]:
```
from sklearn.model_selection import KFold ## for regression
from sklearn.model_selection import StratifiedKFold ## recommended for
↪classification
kfolds = StratifiedKFold(n_splits = 10, random_state = 1, shuffle = True)## a
↪random state is set for reproducibility purpose
```

```
## you can try to remove `random_state = 1, shuffle = True` and see what␣
↪happens next
```

[6]:
```
print(kfolds)
```

```
StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
```

To show the details while implementing `kfolds.split`, in the following execution, we print out the `train_index` and `test_index` of the first loop for you to understand it.

[7]:
```
for train_index, test_index in kfolds.split(Auto, Auto['displacement_big']):
    print("trian_index:{}\n\ntest_index;{}".format(train_index, test_index))
    break
```

```
trian_index:[  0   1   2   3   4   5   6   8   9  12  13  14  15  16  17  18  19
20
  21  22  24  25  26  27  28  29  30  31  32  33  34  35  36  38  39  40
  41  43  44  45  46  47  48  49  50  51  52  53  54  56  57  58  59  61
  62  63  64  65  66  67  68  69  70  71  72  73  74  76  77  78  79  81
  82  83  84  85  87  88  89  90  91  92  93  94  95  96  97  99 100 101
 102 103 104 105 106 107 108 109 110 111 112 113 114 115 118 119 120 121
 122 124 125 127 128 129 130 131 133 134 135 136 137 138 139 141 143 144
 145 146 147 148 149 150 151 152 153 154 155 156 157 159 160 161 162 163
 164 165 166 167 168 169 170 171 173 174 175 176 177 178 179 180 181 182
 183 184 185 186 187 188 190 191 192 194 195 196 197 198 199 200 201 202
 203 204 205 206 207 208 209 210 211 212 214 215 216 217 218 219 220 221
 222 223 224 225 226 227 228 230 231 232 233 234 236 237 238 240 241 242
 243 244 245 246 247 248 249 250 251 252 253 255 256 258 259 260 261 263
 264 265 266 267 268 269 270 271 272 273 275 276 277 278 279 281 282 283
 284 285 286 287 288 289 290 291 292 293 294 295 296 298 299 301 303 304
 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322
 323 324 325 326 327 328 329 331 332 334 336 337 338 339 340 341 342 343
 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362
 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380
 381 382 383 384 385 386 387 388 390 391]

test_index;[  7  10  11  23  37  42  55  60  75  80  86  98 116 117 123 126 132
140
 142 158 172 189 193 213 229 235 239 254 257 262 274 280 297 300 302 330
 333 335 344 389]
```

[8]:
```
cv_classification_errors_1 = []
cv_auc_1 = []
```

[9]:
```
import statsmodels.formula.api as smf
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

```
for train_index, test_index in kfolds.split(Auto,Auto['displacement_big']):
    # train the logistic model
    result = smf.logit('displacement_big ~ mpg + horsepower', data=Auto, subset␣
 ↪= train_index).fit()

    # select the test set according to test_index produced by kfolds.split
    X_test = Auto.loc[test_index,["mpg","horsepower"]]
    y_test = Auto.loc[test_index,"displacement_big"]

    # compute the probabilities of test data
    result_prob = result.predict(X_test)
    # select 0.5 as the threshold
    result_pred = (result_prob > 0.5)
    # compute the classification error
    classification_error = np.mean(result_pred != y_test)
    # add the computed classification error to "cv_classification_errors_1" to␣
 ↪store the result
    cv_classification_errors_1.append(classification_error)

    # calculate the auc
    fpr,tpr,threshold = roc_curve(y_test, result_prob)
    roc_auc = auc(fpr,tpr)
    # add the computed auc to "cv_auc_1" to store the result
    cv_auc_1.append(roc_auc)
```

```
Optimization terminated successfully.
         Current function value: 0.224324
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.240459
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.221340
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.232408
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.224961
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.227415
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.218369
         Iterations 9
Optimization terminated successfully.
```

```
         Current function value: 0.220679
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.202915
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.229121
         Iterations 8
```

Note that the outputs above are the default output of `smf.logit().fit()`.

[10]:
```
print("classification errors using 10-fold CV: {}\n".
 ↪format(cv_classification_errors_1))
print("mean of classification errors using 10-fold CV: {}\n".format(np.
 ↪mean(cv_classification_errors_1)))
```

```
classification errors using 10-fold CV: [0.125, 0.025, 0.1282051282051282,
0.05128205128205128, 0.1282051282051282, 0.05128205128205128,
0.15384615384615385, 0.10256410256410256, 0.07692307692307693,
0.10256410256410256]

mean of classification errors using 10-fold CV: 0.09448717948717947
```

[11]:
```
print("auc using 10-fold CV: {}\n".format(cv_auc_1))
print("mean of auc using 10-fold CV: {}\n".format(np.mean(cv_auc_1)))
```

```
auc using 10-fold CV: [0.9744245524296675, 1.0, 0.9545454545454546,
0.9893048128342246, 0.9679144385026738, 0.9705882352941176, 0.9598930481283422,
0.9572192513368984, 0.93048128342246, 0.981283422459893]

mean of auc using 10-fold CV: 0.9685654498953731
```

Then, we use `weight` and `acceleration` as predictor variables and use `displacement_big` as the response variable to do the logistic regression.

[12]:
```
kfolds
```

[12]:
```
StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
```

[13]:
```
cv_classification_errors_2 = []
cv_auc_2 = []
```

[14]:
```
import statsmodels.formula.api as smf
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

for train_index, test_index in kfolds.split(Auto,Auto['displacement_big']):
```

```
    # train the logistic model
    result = smf.logit('displacement_big ~ weight + acceleration', data=Auto,␣
↪subset = train_index).fit()

    # select the test set according to test_index produced by kfolds.split
    X_test = Auto.loc[test_index,["weight","acceleration"]]
    y_test = Auto.loc[test_index,"displacement_big"]

    # compute the probabilities of test data
    result_prob = result.predict(X_test)
    # select 0.5 as the threshold
    result_pred = (result_prob > 0.5)
    # compute the classification error
    classification_error = np.mean(result_pred != y_test)
    # add the computed classification error to "cv_classification_errors_1" to␣
↪store the result
    cv_classification_errors_2.append(classification_error)

    # calculate the auc
    fpr,tpr,threshold = roc_curve(y_test, result_prob)
    roc_auc = auc(fpr,tpr)
    # add the computed auc to "cv_auc_1" to store the result
    cv_auc_2.append(roc_auc)
```

```
Optimization terminated successfully.
         Current function value: 0.166688
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.176276
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.160031
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.173053
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.166417
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.151830
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.141794
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.168865
```

```
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.174350
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.159999
         Iterations 9
```

[15]:

```python
print("classification errors using 10-fold CV: {}\n".
 ↪format(cv_classification_errors_2))
print("mean of classification errors using 10-fold CV: {}".format(np.
 ↪mean(cv_classification_errors_2)))
```

```
classification errors using 10-fold CV: [0.1, 0.025, 0.15384615384615385,
0.02564102564102564, 0.05128205128205128, 0.10256410256410256,
0.1282051282051282, 0.05128205128205128, 0.02564102564102564,
0.1282051282051282]

mean of classification errors using 10-fold CV: 0.07916666666666666
```

[16]:

```python
print("auc using 10-fold CV: {}\n".format(cv_auc_2))
print("mean of auc using 10-fold CV: {}".format(np.mean(cv_auc_2)))
```

```
auc using 10-fold CV: [0.9872122762148338, 1.0, 0.9759358288770053, 1.0,
0.9893048128342246, 0.9545454545454546, 0.9679144385026738, 0.9973262032085561,
1.0, 0.9759358288770053]

mean of auc using 10-fold CV: 0.9848174843059754
```

Now we can put together the results of the above two models.

[17]:

```python
print("predictor varible: mpg, horsepower; response variable: displacement_big")
print("mean of classification errors using 10-fold CV: {}".format(np.
 ↪mean(cv_classification_errors_1)))
print("mean of auc using 10-fold CV: {}\n".format(np.mean(cv_auc_1)))

print("predictor varible: weight, acceleration; response variable:␣
 ↪displacement_big")
print("mean of classification errors using 10-fold CV: {}".format(np.
 ↪mean(cv_classification_errors_2)))
print("mean of auc using 10-fold CV: {}".format(np.mean(cv_auc_2)))
```

```
predictor varible: mpg, horsepower; response variable: displacement_big
mean of classification errors using 10-fold CV: 0.09448717948717947
mean of auc using 10-fold CV: 0.9685654498953731

predictor varible: weight, acceleration; response variable: displacement_big
mean of classification errors using 10-fold CV: 0.07916666666666666
```

```
mean of auc using 10-fold CV: 0.9848174843059754
```

With both cross-validation criteria, the logistic regression model with `weight` and `acceleration` as predictors is the better model.

1.1 a quick way to do CV for models in `sklearn`

We will redo the same example with the *Auto* data, using the `cross_val_score` function in `sklearn`. The default `scoring` is the accuracy (i.e., 1-classification error), but you can also choose others, such as `roc_auc`, for different applications.

[18]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

logistic_model = LogisticRegression(penalty='none', max_iter = 10000)
error_model_1_cv = cross_val_score(logistic_model, Auto[['mpg','horsepower']],␣
↪Auto['displacement_big'], cv=10)
error_model_2_cv = cross_val_score(logistic_model,␣
↪Auto[['weight','acceleration']], Auto['displacement_big'], cv=10)

print("Logisgic Regression: \n")
print("accuracies of 10-folds:",error_model_1_cv,"(mean classification error:
↪",1-np.mean(error_model_1_cv),")")
print("accuracies of 10-folds:",error_model_2_cv,"(mean classification error:
↪",1-np.mean(error_model_2_cv),")")
```

```
Logisgic Regression:

accuracies of 10-folds: [0.925      0.75       0.97435897 0.92307692 0.87179487
0.8974359
 0.8974359  1.         0.87179487 0.84615385] (mean classification error:
0.10429487179487162 )
accuracies of 10-folds: [0.925      0.95       1.         0.84615385 0.8974359
0.94871795
 0.92307692 0.94871795 0.87179487 0.92307692] (mean classification error:
0.07660256410256405 )
```

Let's implement the above code again (with the default `scoring` filled up) and see if you get different ressults.

[19]:
```
logistic_model = LogisticRegression(penalty='none', max_iter = 10000)
error_model_1_cv = cross_val_score(logistic_model, Auto[['mpg','horsepower']],␣
↪Auto['displacement_big'], cv=10, scoring = 'accuracy')
error_model_2_cv = cross_val_score(logistic_model,␣
↪Auto[['weight','acceleration']], Auto['displacement_big'], cv=10,scoring =␣
↪'accuracy')

print("Logisgic Regression: \n")
```

```
print("accuracies of 10-folds:",error_model_1_cv,"(mean classification error:
↪",1-np.mean(error_model_1_cv),")")
print("accuracies of 10-folds:",error_model_2_cv,"(mean classification error:
↪",1-np.mean(error_model_2_cv),")")
```

```
Logisgic Regression:

accuracies of 10-folds: [0.925      0.75       0.97435897 0.92307692 0.87179487
0.8974359
 0.8974359  1.         0.87179487 0.84615385] (mean classification error:
0.10429487179487162 )
accuracies of 10-folds: [0.925      0.95       1.         0.84615385 0.8974359
0.94871795
 0.92307692 0.94871795 0.87179487 0.92307692] (mean classification error:
0.07660256410256405 )
```

Obviously, we see the same results. This is because by default `cross_val_score` does not shuffle data. If we want to shuffle data, we need to specify how we want to divide the folds. Note that the `kfolds` in the following implementation was a specific 10-fold division we created at the beginning of the tutorial.

[20]:
```
logistic_model = LogisticRegression(penalty='none', max_iter = 10000)
error_model_1_cv = cross_val_score(logistic_model, Auto[['mpg','horsepower']],␣
↪Auto['displacement_big'], cv=kfolds, scoring = 'accuracy')
error_model_2_cv = cross_val_score(logistic_model,␣
↪Auto[['weight','acceleration']], Auto['displacement_big'], cv=kfolds,scoring␣
↪= 'accuracy')

print("Logisgic Regression: \n")
print("accuracies of 10-folds:",error_model_1_cv,"(mean classification error:
↪",1-np.mean(error_model_1_cv),")")
print("accuracies of 10-folds:",error_model_2_cv,"(mean classification error:
↪",1-np.mean(error_model_2_cv),")")
```

```
Logisgic Regression:

accuracies of 10-folds: [0.875      0.975      0.87179487 0.94871795 0.87179487
0.94871795
 0.84615385 0.8974359  0.92307692 0.8974359 ] (mean classification error:
0.09448717948717944 )
accuracies of 10-folds: [0.9        0.975      0.84615385 0.97435897 0.94871795
0.8974359
 0.87179487 0.94871795 0.97435897 0.87179487] (mean classification error:
0.07916666666666661 )
```

2 CV for regression problems

We will use the `displacement` as the response variable. Only the quick `sklearn` CV implementation will be covered here. We will use `r2` (the default) as the criterion for CV. Ohter options include `scoring = neg_mean_squared_error`, which is the negative mean squared error. In older Python versions, one can specify `scoring = mean_squared_error`, but the current version only accepts `scoring = neg_mean_squared_error`.

[21]:
```
from sklearn.linear_model import LinearRegression


kfolds_regresssion = KFold(n_splits = 10, random_state = 1, shuffle = True)
regresssion_model = LinearRegression()
r2_model_1_cv = cross_val_score(regresssion_model, Auto[['mpg','horsepower']],␣
 ↪Auto['displacement'], cv=kfolds_regresssion)
r2_model_2_cv = cross_val_score(regresssion_model,␣
 ↪Auto[['weight','acceleration']], Auto['displacement'], cv=kfolds_regresssion)


print("Linear Regression: \n")
print("r squared of 10-folds:",r2_model_1_cv,"(mean r squared:",np.
 ↪mean(r2_model_1_cv),")")
print("r squared of 10-folds:",r2_model_2_cv,"(mean r squared:",np.
 ↪mean(r2_model_2_cv),")")
```

```
Linear Regression:

r squared of 10-folds: [0.90009344 0.85287829 0.84442486 0.75050943 0.79788919
0.82361727
 0.81645006 0.83507344 0.82927405 0.83134877] (mean r squared:
0.8281558811130347 )
r squared of 10-folds: [0.92741689 0.83846578 0.93362245 0.88999302 0.8949359
0.88834162
 0.8851957  0.90013406 0.93578374 0.87790923] (mean r squared:
0.8971798397399839 )
```

To double check that `r2` is the default option, we implement:

[22]:
```
r2_model_1_cv = cross_val_score(regresssion_model, Auto[['mpg','horsepower']],␣
 ↪Auto['displacement'], cv=kfolds_regresssion, scoring = 'r2')
r2_model_2_cv = cross_val_score(regresssion_model,␣
 ↪Auto[['weight','acceleration']], Auto['displacement'],␣
 ↪cv=kfolds_regresssion,scoring = 'r2')


print("Linear Regression: \n")
print("r squared of 10-folds:",r2_model_1_cv,"(mean r squared:",np.
 ↪mean(r2_model_1_cv),")")
```

```
print("r squared of 10-folds:",r2_model_2_cv,"(mean r squared:",np.
↪mean(r2_model_2_cv),")")
```

```
Linear Regression:

r squared of 10-folds: [0.90009344 0.85287829 0.84442486 0.75050943 0.79788919
0.82361727
 0.81645006 0.83507344 0.82927405 0.83134877] (mean r squared:
0.8281558811130347 )
r squared of 10-folds: [0.92741689 0.83846578 0.93362245 0.88999302 0.8949359
0.88834162
 0.8851957  0.90013406 0.93578374 0.87790923] (mean r squared:
0.8971798397399839 )
```

Next we do CV by mean squared error.

[23]:
```
neg_mse_model_1_cv = cross_val_score(regresssion_model,␣
↪Auto[['mpg','horsepower']], Auto['displacement'],␣
↪cv=kfolds_regresssion,scoring = 'neg_mean_squared_error')
neg_mse_model_2_cv = cross_val_score(regresssion_model,␣
↪Auto[['weight','acceleration']], Auto['displacement'],␣
↪cv=kfolds_regresssion,scoring = 'neg_mean_squared_error')


print("Linear Regression: \n")
print("mean_squared_error of 10-folds:",-neg_mse_model_1_cv,"(mean MSE:",-np.
↪mean(neg_mse_model_1_cv),")")
print("mean_squared_error of 10-folds:",-neg_mse_model_2_cv,"(mean MSE:",-np.
↪mean(neg_mse_model_2_cv),")")
```

```
Linear Regression:

mean_squared_error of 10-folds: [1109.29730323 1863.84433989 1629.08614552
2841.27375213 2432.68725183
 1708.91787009 1741.54206087 1642.23330432 1681.31387899 1825.55161294] (mean
MSE: 1847.5747519803822 )
mean_squared_error of 10-folds: [ 805.91544066 2046.43253439  695.06442086
1252.79258885 1264.59386299
 1081.82362154 1089.27590701  994.40123803  632.40349771 1321.56170032] (mean
MSE: 1118.426481235808 )
```