test

December 10, 2022

1 Lab 04

1.1 Cross Validation

1.1.1 1-2

1) Reuse the notebook from Lab 3 for the wine data. Make sure to

*** Reuse the same random seed throughout.**

*** Use nearest neighbors**

2) With using KFold to produce the data splits, implement cross validation. Make sure to store the predictions on each test fold and print the classification_report after having looped over all folds.

[ ]:
```
from sklearn.datasets import load_wine
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
import numpy as np

x, y = load_wine(return_X_y = True)  #split into features X and labels y
```

[ ]:
```
kf = KFold(n_splits=3, random_state=None, shuffle=True)

result_array = []
y_test_report = []
y_predict_report = []

for train_index, test_index in kf.split(x):
    x_train, x_test = x[train_index],x[test_index]
    y_train, y_test = y[train_index], y[test_index]
    y_test_report.extend(y_test)
    scaler = StandardScaler(copy=True)
    xTrain_scaled = scaler.fit_transform(x_train, y_train)
    minDis = KNeighborsClassifier(n_neighbors=7)
    minDis.fit(xTrain_scaled, y_train)
```

```
    xTest_scaled = scaler.transform(x_test)
    y_predict_report.extend(minDis.predict(xTest_scaled))
    result_array.append(minDis.score(xTest_scaled, y_test))

print('Average score: ', np.mean(result_array))

## print the test reports
print('The classification report:\n')
print(classification_report(y_test_report, y_predict_report))
```

1.1.2 3-4

3) Try with k=3 and k=10 folds.

4) In order to interpret the results (and fix possible issues), take a close look at the KFold visualization from the User Guide (not based on the wine data!):

[ ]:
```
kf = KFold(n_splits=10, random_state=None, shuffle=True)

result_array = []
y_test_report = []
y_predict_report = []

for train_index, test_index in kf.split(x):
    x_train, x_test = x[train_index],x[test_index]
    y_train, y_test = y[train_index], y[test_index]
    y_test_report.extend(y_test)
    scaler = StandardScaler(copy=True)
    xTrain_scaled = scaler.fit_transform(x_train, y_train)
    minDis = KNeighborsClassifier(n_neighbors=7)
    minDis.fit(xTrain_scaled, y_train)
    xTest_scaled = scaler.transform(x_test)
    y_predict_report.extend(minDis.predict(xTest_scaled))
    result_array.append(minDis.score(xTest_scaled, y_test))

print('Average score: ', np.mean(result_array))
## print the test reports
print('The classification report:\n')
print(classification_report(y_test_report, y_predict_report))
```

```
Average score:  0.9663398692810456
The classification report:

              precision    recall  f1-score   support

           0       0.95      1.00      0.98        59
           1       1.00      0.92      0.96        71
           2       0.94      1.00      0.97        48
```

```
    accuracy                           0.97       178
   macro avg       0.96      0.97      0.97       178
weighted avg       0.97      0.97      0.97       178
```

Setting the shuffle parameter is very important since the classes are already ordered dataset

1.2 Grid Search

1.2.1 1-3

1) Implement Grid Search in combination with cross validation.

*** Use the following parameters from the KNeighborsClassifier for the grid: n_neighbors and p . Select reasonable values for both.**

*** Implement a for loop to iterate over all combinations of the grid:**

2) Run the Grid Search and print the classification report for each parameter combination.

3) Which parameter combination performs best?

[ ]:
```python
from sklearn.model_selection import ParameterGrid

n_neighbours = [2, 10]
p = [1, 2]

result_acb = {}

for n_nei in n_neighbours:
    for p_ in p:
        kf = KFold(n_splits=10, random_state=None, shuffle=True)
        result_array = []
        result_acb[str(n_nei) + " / " + str(p_)] = {}
        result_acb[str(n_nei) + " / " + str(p_)]["Y_TEST"] = []
        result_acb[str(n_nei) + " / " + str(p_)]["Y_PREDICT"] = []
        result_acb[str(n_nei) + " / " + str(p_)]["Y_Score"] = []

        for train_index, test_index in kf.split(x):
            x_train, x_test = x[train_index],x[test_index]
            y_train, y_test = y[train_index], y[test_index]
            result_acb[str(n_nei) + " / " + str(p_)]["Y_TEST"].extend(y_test)
            scaler = StandardScaler(copy=True)
            xTrain_scaled = scaler.fit_transform(x_train, y_train)
            minDis = KNeighborsClassifier(n_neighbors=n_nei, p=p_ )
```

```
            minDis.fit(xTrain_scaled, y_train)
            xTest_scaled = scaler.transform(x_test)
            result_acb[str(n_nei) + " / " + str(p_)]["Y_PREDICT"].extend(minDis.
↪predict(xTest_scaled))
            result_acb[str(n_nei) + " / " + str(p_)]["Y_Score"].append(minDis.
↪score(xTest_scaled, y_test))


## print the test reports
for parameters_in in result_acb.keys():
    print('Grid parameters:')
    print(parameters_in)
    print('Average score: ', np.mean(result_acb[parameters_in]["Y_Score"]))
    print(classification_report(result_acb[parameters_in]["Y_TEST"],␣
↪result_acb[parameters_in]["Y_PREDICT"]))
    ␣
↪print('---------------------------------------------------------------------------')
```

```
Grid parameters:
2 / 1
Average score:  0.9663398692810456
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        59
           1       1.00      0.92      0.96        71
           2       0.98      1.00      0.99        48

    accuracy                           0.97       178
   macro avg       0.97      0.97      0.97       178
weighted avg       0.97      0.97      0.97       178

---------------------------------------------------------------------------
Grid parameters:
2 / 2
Average score:  0.9493464052287581
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        59
           1       1.00      0.87      0.93        71
           2       0.94      1.00      0.97        48

    accuracy                           0.95       178
   macro avg       0.95      0.96      0.95       178
weighted avg       0.95      0.95      0.95       178

---------------------------------------------------------------------------
Grid parameters:
```

```
10 / 1
Average score:  0.9830065359477125
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        59
           1       1.00      0.96      0.98        71
           2       1.00      1.00      1.00        48

    accuracy                           0.98       178
   macro avg       0.98      0.99      0.98       178
weighted avg       0.98      0.98      0.98       178

---------------------------------------------------------------------------
Grid parameters:
10 / 2
Average score:  0.9607843137254901
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        59
           1       1.00      0.90      0.95        71
           2       0.94      1.00      0.97        48

    accuracy                           0.96       178
   macro avg       0.96      0.97      0.96       178
weighted avg       0.96      0.96      0.96       178

---------------------------------------------------------------------------
```

neighbour = 10 and manhattahn distance gets the best result

1.3 Combining Grid Search and Cross Validation

1.3.1 1 - 4

1) Carefully read the documentation of GridSearchCV, which combines the mechanisms of the grid search and the cross validation.

2) Reuse the kNeighborsClassifier and the ParameterGrid (check for correct naming).

3) Set the cross validation splitting strategy to k=10 folds.

4) Evaluate the results using GridSearchCV 's built-in methods.

[ ]:
```
from sklearn.model_selection import GridSearchCV
parameters = {"n_neighbors":[2,10], "p":[1,2]}
kn = KNeighborsClassifier()
clf = GridSearchCV(kn, parameters, cv = 10)

clf.fit(x,y)
```

```
print(clf.best_estimator_.score)
print("best score: ",clf.score(x,y))
```

```
<bound method ClassifierMixin.score of KNeighborsClassifier(n_neighbors=10,
p=1)>
best score:  0.8370786516853933
```

As we see the result is the same as we got in the previous task

1.3.2 5-6

5) Change the parameter scoring to use the F1 score for evaluation.

6) Find out how to store/access the best model parametrization.

```
[ ]: from sklearn.model_selection import GridSearchCV
parameters = {"n_neighbors":[2,10], "p":[1,2]}
kn = KNeighborsClassifier()
# f1_weighted because we do not have only 0 and 1 as values
clf = GridSearchCV(kn, parameters, cv = 10, scoring = "f1_weighted")

clf.fit(x,y)
estimator = clf.best_estimator_
print("best value for n_neighbors parameter: ",estimator.
 ↪get_params()['n_neighbors'])
print("best value for p parameter: ",estimator.get_params()["p"])
```

```
best value for n_neighbors parameter:  10
best value for p parameter:  1
```

1.4 Homework

Extend the grid with a parameter for switching the scaling of the data on/off. Then, for each test run made so far, enter the cross validation results in your table. Those values are more robust and reliable than those obtained from a single run.

```
[ ]: x, y = load_wine(return_X_y = True)  #split into features X and labels y

parameters = {"n_neighbors":[2,10], "p":[1,2], "scale_with_mean":[False]}
kn = KNeighborsClassifier()
# f1_weighted because we do not have only 0 and 1 as values
clf = GridSearchCV(kn, parameters, cv = 10, scoring = "f1_weighted")

clf.fit(x,y)
estimator = clf.best_estimator_
print("best value for n_neighbors parameter: ",estimator.
 ↪get_params()['n_neighbors'])
print("best value for p parameter: ",estimator.get_params()["p"])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_8436/2862377156.py in <module>
      6 clf = GridSearchCV(kn, parameters, cv = 10, scoring = "f1_weighted")
      7 
----> 8 clf.fit(x,y)
      9 estimator = clf.best_estimator_
     10 print("best value for n_neighbors parameter: ",estimator.
 ↪get_params()['n_neighbors'])

~/.local/lib/python3.8/site-packages/sklearn/model_selection/_search.py in␣
 ↪fit(self, X, y, groups, **fit_params)
    889                 return results
    890 
--> 891             self._run_search(evaluate_candidates)
    892 
    893             # multimetric is determined here because in the case of a␣
 ↪callable

~/.local/lib/python3.8/site-packages/sklearn/model_selection/_search.py in␣
 ↪_run_search(self, evaluate_candidates)
   1390     def _run_search(self, evaluate_candidates):
   1391         """Search all candidates in param_grid"""
-> 1392         evaluate_candidates(ParameterGrid(self.param_grid))
   1393 
   1394 

~/.local/lib/python3.8/site-packages/sklearn/model_selection/_search.py in␣
 ↪evaluate_candidates(candidate_params, cv, more_results)
    836                     )
    837 
--> 838                 out = parallel(
    839                     delayed(_fit_and_score)(
    840                         clone(base_estimator),

~/.local/lib/python3.8/site-packages/joblib/parallel.py in __call__(self,␣
 ↪iterable)
   1041             # remaining jobs.
   1042             self._iterating = False
-> 1043             if self.dispatch_one_batch(iterator):
   1044                 self._iterating = self._original_iterator is not None
   1045 

~/.local/lib/python3.8/site-packages/joblib/parallel.py in␣
 ↪dispatch_one_batch(self, iterator)
    859                 return False
```

```
    860             else:
--> 861                 self._dispatch(tasks)
    862                 return True
    863

~/.local/lib/python3.8/site-packages/joblib/parallel.py in _dispatch(self, batch)
    777         with self._lock:
    778             job_idx = len(self._jobs)
--> 779             job = self._backend.apply_async(batch, callback=cb)
    780             # A job can complete so quickly than its callback is
    781             # called before we get here, causing self._jobs to

~/.local/lib/python3.8/site-packages/joblib/_parallel_backends.py in␣
 ↪apply_async(self, func, callback)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

~/.local/lib/python3.8/site-packages/joblib/_parallel_backends.py in␣
 ↪__init__(self, batch)
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573
    574     def get(self):

~/.local/lib/python3.8/site-packages/joblib/parallel.py in __call__(self)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)

    263                     for func, args, kwargs in self.items]
    264

~/.local/lib/python3.8/site-packages/joblib/parallel.py in <listcomp>(.0)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)

    263                     for func, args, kwargs in self.items]
    264

~/.local/lib/python3.8/site-packages/sklearn/utils/fixes.py in __call__(self,␣
 ↪*args, **kwargs)
    209     def __call__(self, *args, **kwargs):
    210         with config_context(**self.config):
--> 211             return self.function(*args, **kwargs)
```

```
    212
    213

~/.local/lib/python3.8/site-packages/sklearn/model_selection/_validation.py in _fit_and_score(estimator, X, y, scorer, train, test, verbose, parameters, fit_params, return_train_score, return_parameters, return_n_test_samples, return_times, return_estimator, split_progress, candidate_progress, error_score)
    667                 cloned_parameters[k] = clone(v, safe=False)
    668
--> 669         estimator = estimator.set_params(**cloned_parameters)
    670
    671     start_time = time.time()

~/.local/lib/python3.8/site-packages/sklearn/base.py in set_params(self, **params)
    238             key, delim, sub_key = key.partition("__")
    239             if key not in valid_params:
--> 240                 raise ValueError(
    241                     "Invalid parameter %s for estimator %s. "
    242                     "Check the list of available parameters "

ValueError: Invalid parameter scale_with_mean for estimator KNeighborsClassifier(n_neighbors=2, p=1). Check the list of available parameters with `estimator.get_params().keys()`.
```