model-evaluation_exercise

May 31, 2019

```
In [ ]: import matplotlib.pyplot as plt
        %matplotlib inline
```

0.1

```
In [ ]: from sklearn.datasets import make_blobs
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import train_test_split

        #
        X, y = make_blobs(random_state=0)
        #
        X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
        #
        logreg = LogisticRegression().fit(X_train, y_train)
        #
        print("  : {:.2f}".format(logreg.score(X_test, y_test)))
```

0.1.1 (Cross Validation)

```
In [ ]: import mglearn
        mglearn.plots.plot_cross_validation()
```

scikit-learn

```
In [ ]: from sklearn.model_selection import cross_val_score
        from sklearn.datasets import load_iris
        from sklearn.linear_model import LogisticRegression

        iris = load_iris()
        logreg = LogisticRegression()

        scores = cross_val_score(logreg, iris.data, iris.target)
        print("  : {}".format(scores))
```

```
In [ ]: scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
        print("  : {}".format(scores))
```

```
In [ ]: print("   : {:.2f}".format(scores.mean()))
```

0.1.2

0.1.3 (Stratified) k-

```
In [ ]: from sklearn.datasets import load_iris
        iris = load_iris()
        print("Iris :\n{}".format(iris.target))
```

```
In [ ]: mglearn.plots.plot_stratified_cross_validation()
```

0.1.4

```
In [ ]: from sklearn.model_selection import KFold
        kfold = KFold(n_splits=5)
```

```
In [ ]: print("  :\n{}".format(
              cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

```
In [ ]: kfold = KFold(n_splits=3)
        print("  :\n{}".format(
              cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

```
In [ ]: kfold = KFold(n_splits=3, shuffle=True, random_state=0)
        print("  :\n{}".format(
            cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

```
In [ ]: len(iris.target)
```

LOOCV(Leave-One-Out cross-validation)

```
In [ ]: from sklearn.model_selection import LeaveOneOut
        loo = LeaveOneOut()
        scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
        print("   : ", len(scores))
        print(" : {:.2f}".format(scores.mean()))
```

```
In [ ]: mglearn.plots.plot_shuffle_split()
```

```
In [ ]: from sklearn.model_selection import ShuffleSplit
        shuffle_split = ShuffleSplit(test_size=.5, train_size=.5, n_splits=10)
        scores = cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
        print("  :\n{}".format(scores))
```

```
In [ ]: from sklearn.model_selection import GroupKFold
        #
        X, y = make_blobs(n_samples=12, random_state=0)
        #
        #     .
        groups = [0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3]
        scores = cross_val_score(logreg, X, y, groups, cv=GroupKFold(n_splits=3))
        print("  :\n{}".format(scores))
```

```
In [ ]: mglearn.plots.plot_group_kfold()
```

0.1.5 (Grid Search)

```
In [ ]: #
        from sklearn.svm import SVC
        X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
                                                            random_state=0)
        print("  : {}      : {}".format(
              X_train.shape[0], X_test.shape[0]))

        best_score = 0

        for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
            for C in [0.001, 0.01, 0.1, 1, 10, 100]:
                #     SVC
                svm = SVC(gamma=gamma, C=C)
                svm.fit(X_train, y_train)
                #   SVC
                score = svm.score(X_test, y_test)
                #
                if score > best_score:
                    best_score = score
                    best_parameters = {'C': C, 'gamma': gamma}

        print(" : {:.2f}".format(best_score))
        print(" : {}".format(best_parameters))
```

```
In [ ]: mglearn.plots.plot_threefold_split()
```

```
In [ ]: from sklearn.svm import SVC
        #  +
        X_trainval, X_test, y_trainval, y_test = train_test_split(
            iris.data, iris.target, random_state=0)
```

```
# +
X_train, X_valid, y_train, y_valid = train_test_split(
    X_trainval, y_trainval, random_state=1)
print("  : {}     : {}     :"
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        #     SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        #   SVC
        score = svm.score(X_valid, y_valid)
        #
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

#
#
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("   : {:.2f}".format(best_score))
print(" : ", best_parameters)
print("    : {:.2f}".format(test_score))
```

```
In [ ]: import numpy as np
        for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
            for C in [0.001, 0.01, 0.1, 1, 10, 100]:
                #     SVC
                svm = SVC(gamma=gamma, C=C)
                #
                scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
                #
                score = np.mean(scores)
                #
                if score > best_score:
                    best_score = score
                    best_parameters = {'C': C, 'gamma': gamma}
        #
        svm = SVC(**best_parameters)
        svm.fit(X_trainval, y_trainval)
```

```
In [ ]: mglearn.plots.plot_cross_val_selection()
In [ ]: mglearn.plots.plot_grid_search_overview()
```

```
In [ ]: param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
                      'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
        print(" :\n{}".format(param_grid))
```

```
In [ ]: from sklearn.model_selection import GridSearchCV
        from sklearn.svm import SVC
        grid_search = GridSearchCV(SVC(), param_grid, cv=5, return_train_score=True)
```

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
                                                            random_state=0)
```

```
In [ ]: grid_search.fit(X_train, y_train)
```

```
In [ ]: print("  : {:.2f}".format(grid_search.score(X_test, y_test)))
```

```
In [ ]: print(" : {}".format(grid_search.best_params_))
        print("   : {:.2f}".format(grid_search.best_score_))
```

```
In [ ]: print("  :\n{}".format(grid_search.best_estimator_))
```