K-Fold Cross Validation

In [1]:
```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import mglearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
iris = load_iris()
logreg = LogisticRegression()
```

In [3]:
```
scores = cross_val_score(logreg, iris.data, iris.target)
print("Cross-validation scores: {}".format([f"{score:.3f}" for score in scores]))
```

```
Cross-validation scores: ['0.967', '1.000', '0.933', '0.967', '1.000']
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

By default, cross_val_score performs 5-fold cross-validation, returning five accuracy values.

In [4]:
```
scores = cross_val_score(logreg, iris.data, iris.target, cv=3)
print("Cross-validation scores: {}".format([f"{score:.3f}" for score in scores]))
```

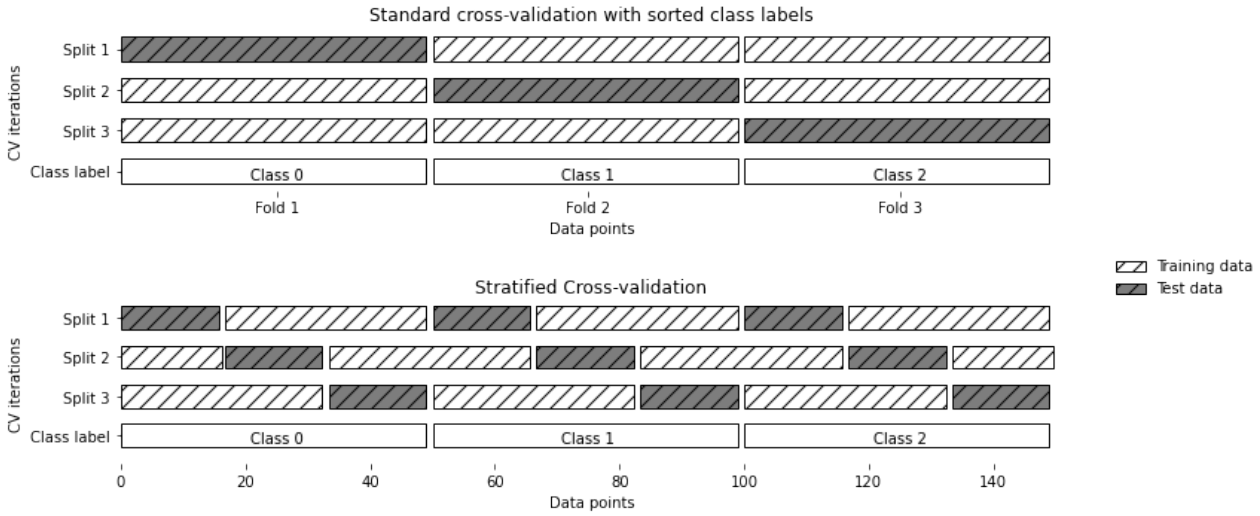
```
Cross-validation scores: ['0.980', '0.960', '0.980']
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [5]:
```
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

```
Average cross-validation score: 0.97
```

Using the mean cross-validation we can conclude that we expect the model to be around 97% accurate on average.

In [6]:
```
mglearn.plots.plot_stratified_cross_validation()
```



It is usually a good idea to use stratified k-fold cross-validation instead of k-fold cross-validation to evaluate a classifier, because it results in more reliable estimates of generalization performance

More control over cross-validation

In [7]:
```
from sklearn.model_selection import KFold
```

In [8]:
```
kfold = KFold(n_splits=3, shuffle=True, random_state=0)

print("Cross-validation scores:\n{}".format(
 cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

```
Cross-validation scores:
[0.98 0.96 0.96]
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Leave-one-out Cross Validation

In [9]:
```
from sklearn.model_selection import LeaveOneOut
```

In [10]:
```
loo = LeaveOneOut()
scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
```

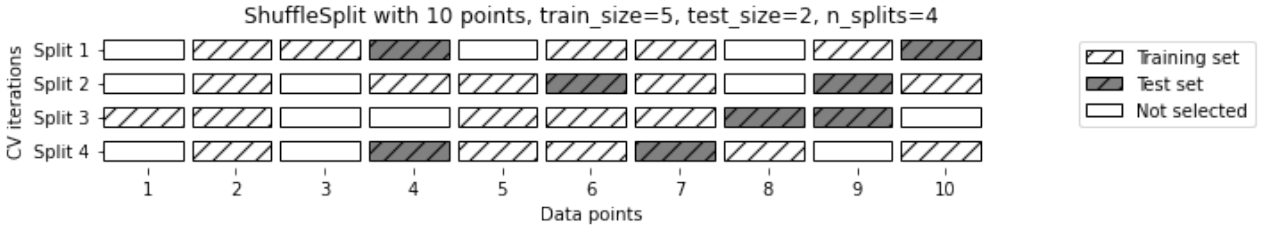
```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [11]:
```
print("Mean accuracy: {:.2f}".format(scores.mean()))
```

```
Mean accuracy: 0.97
```

Shuffle-split Cross-Validation

In [12]:
```
mglearn.plots.plot_shuffle_split()
```



In [13]:
```
from sklearn.model_selection import ShuffleSplit
```

In [14]:
```
shuffle_split = ShuffleSplit(test_size=.8, train_size=.2, n_splits=10)
scores = cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
```

In [15]:
```
print("Cross-validation scores:\n{}".format([f"{score:.3f}" for score in scores]))
```

```
Cross-validation scores:
['0.942', '0.933', '0.950', '0.950', '0.942', '0.875', '0.933', '0.900', '0.925', '0.98
3']
```

Grid Search

In [16]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(
 iris.data, iris.target, random_state=0)
```

In [18]:
```
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]:

        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)

        score = svm.score(X_test, y_test)

# if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
```

In [19]:
```
svm = SVC(**best_parameters)
svm.fit(X_train, y_train)
test_score = svm.score(X_test, y_test)
```

In [20]:
```
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

```
Best score on validation set: 0.97
Best parameters:  {'C': 100, 'gamma': 0.001}
Test set score with best parameters: 0.97
```

Grid Search with Cross-Validation (1)

In [21]:
```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:

        svm = SVC(gamma=gamma, C=C)

        scores = cross_val_score(svm, X_train, y_train, cv=5)
        score = np.mean(scores)

        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
```

In [22]:
```
svm = SVC(**best_parameters)
svm.fit(X_train, y_train)
test_score = svm.score(X_test, y_test)
```

In [23]:
```
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

```
Best score on validation set: 0.97
Best parameters:  {'C': 100, 'gamma': 0.001}
Test set score with best parameters: 0.97
```

Grid Search with Cross-Validation (2)

In [24]:
```
from sklearn.model_selection import GridSearchCV
```

In [25]:
```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}

grid_search = GridSearchCV(SVC(), param_grid, cv=5)
```

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(
 iris.data, iris.target, random_state=0)
```

In [27]:
```
grid_search.fit(X_train, y_train)
```

Out[27]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                         'gamma': [0.001, 0.01, 0.1, 1, 10, 100]})
```

In [28]:
```
print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
```

```
Test set score: 0.97
```

In [29]:
```
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'C': 10, 'gamma': 0.1}
Best cross-validation score: 0.97
```

Analyzing the result of cross-validation

In [30]:
```
results = pd.DataFrame(grid_search.cv_results_)
results.head(3)
```

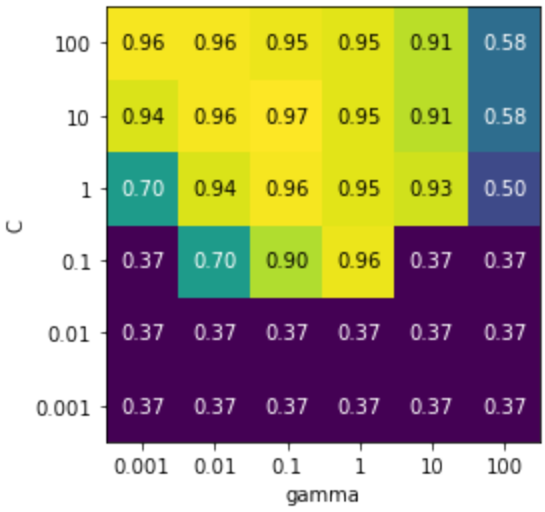
Out[30]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	params	split
0	0.001595	0.000798	0.000405	0.000497	0.001	0.001	{'C': 0.001,	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	params	split
							'gamma': 0.001}	
1	0.000797	0.000399	0.000599	0.000489	0.001	0.01	{'C': 0.001, 'gamma': 0.01}	
2	0.000792	0.000396	0.000598	0.000488	0.001	0.1	{'C': 0.001, 'gamma': 0.1}	

In [31]:
```
scores = np.array(results.mean_test_score).reshape(6, 6)

mglearn.tools.heatmap(scores, xlabel='gamma', xticklabels=param_grid['gamma'],
 ylabel='C', yticklabels=param_grid['C'], cmap="viridis")
```

Out[31]: <matplotlib.collections.PolyCollection at 0x237f78219a0>



The color encodes the cross-validation accuracy, with light colors meaning high accuracy and dark colors meaning low accuracy. You can see that SVC is very sensitive to the setting of the parameters.

Confusion Matrix

In [32]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
```

In [33]:
```
cancer = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(
 cancer.data, cancer.target, random_state=0)

logreg = LogisticRegression(solver='liblinear').fit(X_train, y_train)
```

```
pred_logreg = logreg.predict(X_test)
print("logreg score: {:.2f}".format(logreg.score(X_test, y_test)))
```

```
logreg score: 0.96
```

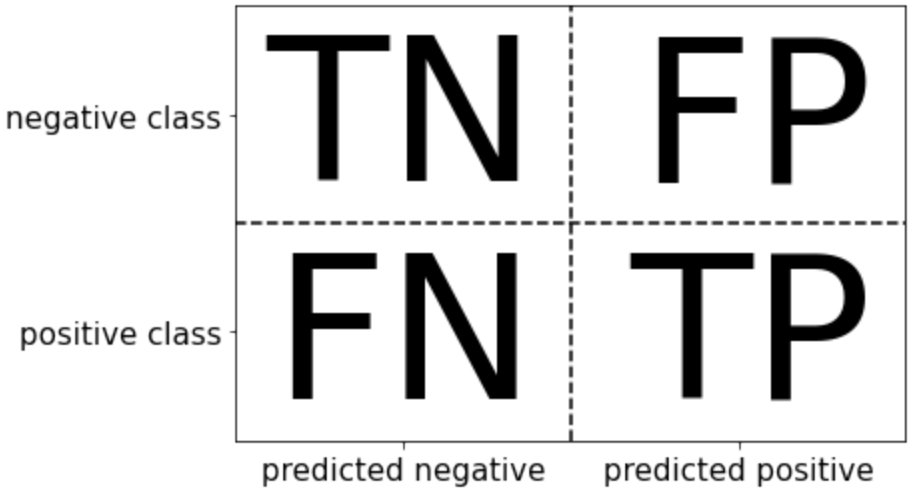
In [34]:

```
confusion = confusion_matrix(y_test, pred_logreg)
print("Confusion matrix:\n{}".format(confusion))
```

```
Confusion matrix:
[[52  1]
 [ 5 85]]
```

In [35]:

```
mglearn.plots.plot_binary_confusion_matrix()
```



$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad \text{Precision} = \frac{TP}{TP+FP} \quad \text{Recall} = \frac{TP}{TP+FN}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In [36]:

```
from sklearn.metrics import f1_score

print("f1 score logistic regression: {:.2f}".format(
 f1_score(y_test, pred_logreg)))
```

```
f1 score logistic regression: 0.97
```

Threshold

1. Precision-recall curves
2. ROC curves

In [37]:

```
#PRC
from sklearn.metrics import precision_recall_curve
```

```
precision, recall, thresholds = precision_recall_curve(
 y_test, pred_logreg)
```

In [38]:
```
# Find the best threshold (e.g., maximize precision and recall)
optimal_idx = np.argmax(precision + recall)  # Choose threshold that gives highest prec
optimal_threshold = thresholds[optimal_idx]
```

In [39]:
```
optimized_predictions = (pred_logreg >= optimal_threshold).astype(int)
```

In [40]:
```
print("f1 score logistic regression: {:.2f}".format(
 f1_score(y_test, optimized_predictions)))
```

```
f1 score logistic regression: 0.97
```

In [41]:
```
#ROC
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, pred_logreg)
```

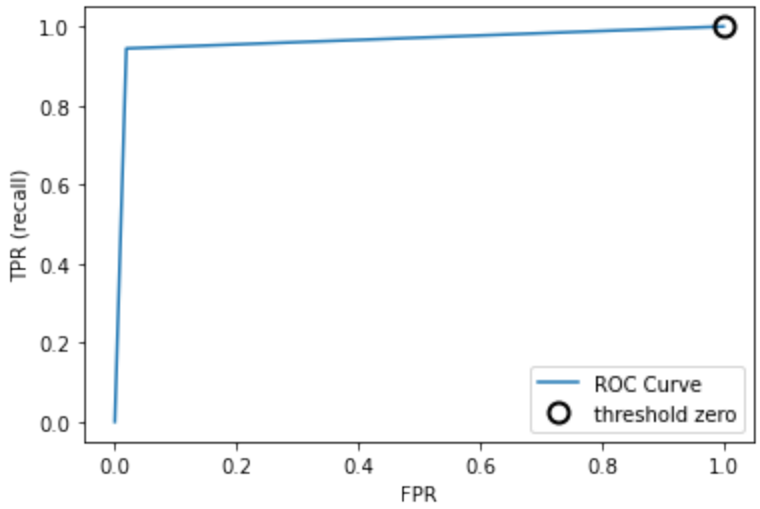
In [42]:
```
probs = logreg.predict_proba(X_test)[:, 1]
```

In [43]:
```
from sklearn.metrics import roc_auc_score
log_auc = roc_auc_score(pred_logreg, probs)

print("AUC for Logistics: {:.3f}".format(log_auc))
```

```
AUC for Logistics: 1.000
```

In [44]:
```
plt.plot(fpr, tpr, label="ROC Curve")
plt.xlabel("FPR")
plt.ylabel("TPR (recall)")
# find threshold closest to zero
close_zero = np.argmin(np.abs(thresholds))
plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
 label="threshold zero", fillstyle="none", c='k', mew=2)
plt.legend(loc=4)
```

Out[44]:
```
<matplotlib.legend.Legend at 0x237f76df1f0>
```



Metrics for Multiclass Classification

In [45]:
```
from sklearn.metrics import accuracy_score
```

In [47]:
```
from sklearn.datasets import load_digits
digits = load_digits()
```

In [48]:
```
X_train, X_test, y_train, y_test = train_test_split(
 digits.data, digits.target, random_state=0)
```

In [49]:
```
lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

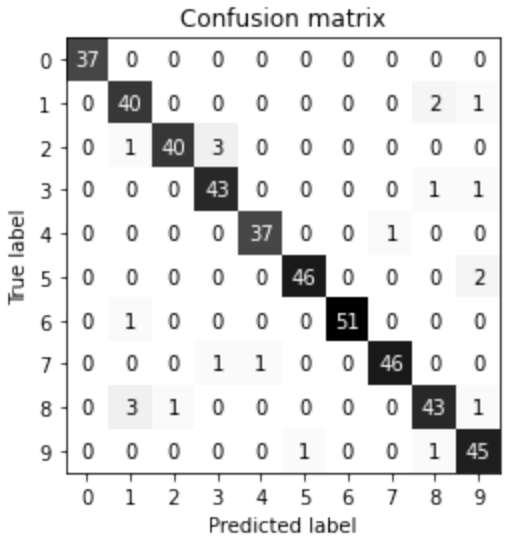
In [50]:
```
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

```
Accuracy: 0.951
Confusion matrix:
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 40  0  0  0  0  0  0  2  1]
 [ 0  1 40  3  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  1  1]
 [ 0  0  0  0 37  0  0  1  0  0]
 [ 0  0  0  0  0 46  0  0  0  2]
 [ 0  1  0  0  0  0 51  0  0  0]
 [ 0  0  0  1  1  0  0 46  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  0  0  1  0  0  1 45]]
```

The model has an accuracy of 95.3%.

In [53]:

```
scores_image = mglearn.tools.heatmap(
    confusion_matrix(y_test, pred), xlabel='Predicted label',
    ylabel='True label', xticklabels=digits.target_names,
    yticklabels=digits.target_names, cmap=plt.cm.gray_r, fmt="%d")

plt.title("Confusion matrix")
plt.gca().invert_yaxis()
```



In [56]:

```
from sklearn.metrics import classification_report


print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        37
           1       0.89      0.93      0.91        43
           2       0.98      0.91      0.94        44
           3       0.91      0.96      0.93        45
           4       0.97      0.97      0.97        38
           5       0.98      0.96      0.97        48
           6       1.00      0.98      0.99        52
           7       0.98      0.96      0.97        48
           8       0.91      0.90      0.91        48
           9       0.90      0.96      0.93        47

    accuracy                           0.95       450
```

macro avg	0.95	0.95	0.95	450
weighted avg	0.95	0.95	0.95	450