URL: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-and-model-selection As shown in the tutorial, create a notebook that replicates and implements at least four different cross-validation methods. Then pick two cross-validation methods to compare the performance of your best-performing SVM, Decision tree, AdaBoost, and Random Forest models on the breast cancer data from HW5.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, ShuffleSplit, RepeatedKFold, cross_val_score, GridSearchCV
from sklearn import datasets, svm

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
```

In [2]:
```
%matplotlib inline

# Turn off warnings completely for the Notebook
import warnings
warnings.filterwarnings('ignore')
```

Part 1. Replicates four types of cross-validation methods

In [3]:
```
# Load iris data
X, y = datasets.load_iris(return_X_y=True)
X.shape, y.shape
```

Out[3]:
```
((150, 4), (150,))
```

In [4]:
```
# split train-test subsets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0, stratify=y)
```

In [5]:
```
# fit SVM model
clf = make_pipeline(StandardScaler(), svm.SVC())
clf.fit(X_train, y_train)
clf.score(X_test,y_test)
```

Out[5]:
```
0.9777777777777777
```

In [6]:
```
# cross-validation computing
# Method 1. computing the score 5 consecutive times (with different splits each time)
scores_1 = cross_val_score(clf, X, y, cv=5)
scores_1
```

Out[6]:
```
array([0.96666667, 0.96666667, 0.96666667, 0.93333333, 1.        ])
```

In [7]:
```
print('%.2f accuracy with a standard deviation of %.2f '%(scores_1.mean(), scores_1.std()))
```

```
0.97 accuracy with a standard deviation of 0.02
```

In [8]:
```
# Method 2. ShuffleSplit
n_samples = X.shape[0]
cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
scores_2 = cross_val_score(clf, X, y, cv=cv)
scores_2
```

Out[8]:
```
array([0.97777778, 0.93333333, 0.95555556, 0.93333333, 0.97777778])
```

In [9]:
```
print('%.2f accuracy with a standard deviation of %.2f '%(scores_2.mean(), scores_2.std()))
```

```
0.96 accuracy with a standard deviation of 0.02
```

In [10]:
```
# Method 3. use an iterable yielding (train, test) splits
def custom_cv_2folds(X):
     n = X.shape[0]
     i = 1
     while i <= 2:
         idx = np.arange(n * (i - 1) / 2, n * i / 2, dtype=int)
         yield idx, idx
         i += 1

custom_cv = custom_cv_2folds(X)
scores_3 = cross_val_score(clf, X, y, cv=custom_cv)
print('%.2f accuracy with a standard deviation of %.2f '%(scores_3.mean(), scores_3.std()))
```

```
0.99 accuracy with a standard deviation of 0.01
```

In [11]:
```
# Method 4. Repeated K-Fold
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=123)
scores_4 = cross_val_score(clf, X, y, cv=rkf)
print('%.2f accuracy with a standard deviation of %.2f '%(scores_4.mean(), scores_4.std()))
```

```
0.94 accuracy with a standard deviation of 0.01
```

Part 2. pick two cross-validation methods to compare the performance of your best-performing SVM, Decision tree, AdaBoost, and Random Forest models on the breast cancer data from HW5

In [12]:
```
# load data
wbc = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data', header=None)
```

In [13]:
```
# check for any NA
wbc.isna().sum().sum()
```

Out[13]:
```
0
```

In [14]:
```
wbc.head()
```

Out[14]:

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	25	26	27	28	29	30	31
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 32 columns

In [15]:
```
# split as features and target
X, y = wbc.iloc[:, 2:], wbc.iloc[:, 1]
X.shape, y.shape
```

Out[15]:
```
((569, 30), (569,))
```

In [16]:
```
# Divide the data into train (80%) and test (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, stratify=y, test_size=0.2)
```

In [17]:
```
clf1 = DecisionTreeClassifier(random_state=123)
clf2 = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), random_state=123)
clf3 = RandomForestClassifier(random_state=123)
clf4 = svm.SVC(random_state=123)

pipe4 = Pipeline([['sc', StandardScaler()], ['clf', clf4]])
```

In [18]:
```
## Evaluating and tuning the ensemble classifier with GridSearchCV ===================
# get a basic idea of how we can access the individual parameters inside a GridSearch object:
pipe4.get_params()
```

Out[18]:
```
{'memory': None,
 'steps': [['sc', StandardScaler()], ['clf', SVC(random_state=123)]],
 'verbose': False,
 'sc': StandardScaler(),
 'clf': SVC(random_state=123),
 'sc__copy': True,
 'sc__with_mean': True,
 'sc__with_std': True,
 'clf__C': 1.0,
 'clf__break_ties': False,
 'clf__cache_size': 200,
 'clf__class_weight': None,
 'clf__coef0': 0.0,
 'clf__decision_function_shape': 'ovr',
 'clf__degree': 3,
 'clf__gamma': 'scale',
 'clf__kernel': 'rbf',
 'clf__max_iter': -1,
 'clf__probability': False,
 'clf__random_state': 123,
 'clf__shrinking': True,
 'clf__tol': 0.001,
 'clf__verbose': False}
```

In [19]:
```
# tune the parameters via a grid search
params_1 = {'criterion':['gini','entropy'],
            'max_depth':[5,10,20]}
params_2 = {'n_estimators':[50,500,1000],
             'learning_rate':[0.5, 1.0, 5.0],
             'algorithm':['SAMME.R','SAMME']}
params_3 = {'criterion':['gini','entropy'],
             'max_depth':[5,10,20],
             'min_samples_split':[20,50,100],
             'min_samples_leaf':[5,10],
             'oob_score':[True, False]}
params_4 = {'clf__C':[0.001, 0.1, 100.0],
             'clf__kernel':['linear', 'poly', 'rbf', 'sigmoid', 'precomputed'],
             'clf__gamma':['scale', 'auto'],
             'clf__shrinking':[True,False]}
```

In [20]:
```
# Define founction for best parameters search
def find_best_param(clf,param):
    grid = GridSearchCV(estimator=clf, param_grid=param, cv=10)
    grid.fit(X_train, y_train)

    print(grid.best_score_)
    print(grid.best_params_)
    print(grid.best_estimator_)
```

In [21]:
```
# best performing decision tree
find_best_param(clf1,params_1)
```

```
0.9561835748792271
{'criterion': 'gini', 'max_depth': 10}
DecisionTreeClassifier(max_depth=10, random_state=123)
```

In [22]:
```
clf1_b = DecisionTreeClassifier(criterion='gini', max_depth=10, random_state=123)
```

In [23]:
```
# best performing adaBoost classifier
find_best_param(clf2,params_2)
```

```
0.9517391304347826
{'algorithm': 'SAMME.R', 'learning_rate': 0.5, 'n_estimators': 50}
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.5,
                   random_state=123)
```

In [24]:
```
clf2_b = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(),
                            algorithm='SAMME.R', learning_rate=0.5, n_estimators=50,
                           random_state=123)
```

In [25]:
```
# best performing random forest classifer
find_best_param(clf3, params_3)
```

```
0.953816425120773
{'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 20, 'oob_score': True}
RandomForestClassifier(criterion='entropy', max_depth=5, min_samples_leaf=5,
                       min_samples_split=20, oob_score=True, random_state=123)
```

In [26]:
```
clf3_b = RandomForestClassifier(criterion='entropy', max_depth=5, min_samples_leaf=5,
                                min_samples_split=20, oob_score=True, random_state=123,
                                max_features='auto')
```

In [27]:
```
# best performing SVM
find_best_param(pipe4, params_4)
```

```
0.9802415458937197
{'clf__C': 0.1, 'clf__gamma': 'scale', 'clf__kernel': 'linear', 'clf__shrinking': True}
Pipeline(steps=[('sc', StandardScaler()),
                ['clf', SVC(C=0.1, kernel='linear', random_state=123)]])
```

In [28]:
```
pipe4_b = Pipeline(steps=[['sc', StandardScaler()],
                          ['clf', svm.SVC(C=0.1, kernel='linear', gamma='scale',
                                          shrinking=True, random_state=123)]])
```

In [29]:
```
clf_labels = ['Decision tree', 'adaBoost', 'RandForest', 'SVM']
```

In [30]:
```
# cross-validataion I === 10-fold cross validation

print('10-fold cross validation:\n')
for clf, label in zip([clf1_b, clf2_b, clf3_b, pipe4_b], clf_labels):
    scores = cross_val_score(estimator=clf,
                             X=X_train,
                             y=y_train,
                             cv=10)
    print("Accuracy with Standard Deviation: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
10-fold cross validation:

Accuracy with Standard Deviation: 0.96 (+/- 0.02) [Decision tree]
Accuracy with Standard Deviation: 0.95 (+/- 0.03) [adaBoost]
Accuracy with Standard Deviation: 0.95 (+/- 0.02) [RandForest]
Accuracy with Standard Deviation: 0.98 (+/- 0.02) [SVM]
```

In [31]:
```
# cross-validataion II === ShuffleSplit cross validation
n_samples = X_train.shape[0]
cv = ShuffleSplit(n_splits=10, test_size=0.3, random_state=0)
```

In [32]:
```
print('shuffle-split cross validation:\n')
for clf, label in zip([clf1_b, clf2_b, clf3_b, pipe4_b], clf_labels):
    scores = cross_val_score(estimator=clf,
                             X=X_train,
                             y=y_train,
                             cv=cv)
    print("Accuracy with Standard Deviation: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
shuffle-split cross validation:

Accuracy with Standard Deviation: 0.93 (+/- 0.02) [Decision tree]
Accuracy with Standard Deviation: 0.93 (+/- 0.02) [adaBoost]
Accuracy with Standard Deviation: 0.95 (+/- 0.02) [RandForest]
Accuracy with Standard Deviation: 0.98 (+/- 0.01) [SVM]
```

In [ ]: