In [4]:
```python
from sklearn import svm, datasets

from sklearn.model_selection import GridSearchCV

# ML
from sklearn.neighbors import KNeighborsClassifier

# import libraries for model validation
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import LeavePOut
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import StratifiedShuffleSplit

from sklearn.model_selection import cross_val_score
```

In [5]:
```python
iris = datasets.load_iris()
```

In [8]:
```python
iris.data[:10]
```

Out[8]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [13]:
```python
iris.target[:100]
```

Out[13]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [12]:
```python
iris.target_names
```

Out[12]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [14]:
```python
iris.target
```

Out[14]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [10]:
```python
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [23]:
```
X = iris.data
y = iris.target
```

In [24]:
```
# instantiate the KNN classifier
# {'metric': 'minkowski', 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
clf = KNeighborsClassifier()
```

In [25]:
```
# get the KNN parameters
clf.get_params()
```

Out[25]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

In [26]:
```
# get the KNN parameters
clf.get_params().keys()
```

Out[26]:
```
dict_keys(['algorithm', 'leaf_size', 'metric', 'metric_params', 'n_jobs', 'n_neigh
bors', 'p', 'weights'])
```

In [27]:
```
n_neighbors = [3, 5, 7, 9, 11, 13, 15, 19, 23, 29]
algos       = ['ball_tree', 'kd_tree']
dist_metric = ['minkowski']
p_root      = [1, 2, 3, 4, 5]
weights     = ['uniform', 'distance']
leaf_size   = [15, 30, 40, 50, 60,70]
```

In [28]:
```
# define the parameters dict
parameters = dict(
                n_neighbors= n_neighbors,
                #algorithm= algos,
                metric= dist_metric,
                p= p_root,
                weights= weights,
                #leaf_size= leaf_size
            )

print(parameters)
```

```
{'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 19, 23, 29], 'metric': ['minkowski'],
'p': [1, 2, 3, 4, 5], 'weights': ['uniform', 'distance']}
```

In [29]:
```
# define splits
n_splits = 5

kf  = KFold(n_splits =n_splits, shuffle=True, random_state=100)
#skf = StratifiedKFold(n_splits =n_splits, random_state=100)
skf = StratifiedKFold(n_splits =n_splits)

#sf  = ShuffleSplit(n_splits =n_splits, test_size=0.2, random_state=100)
#ssf = StratifiedShuffleSplit(n_splits =n_splits, test_size=0.2, random_state=100)
```

In [30]:
```
# instantiate the grid search CV
grid = GridSearchCV(estimator = clf,
                   param_grid = parameters,
                   scoring    = 'accuracy',
                   cv=kf,
                   verbose=1)
```

In [31]:
```python
# fit the data to the grid object
grid.fit(X, y)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[31]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=KNeighborsClassifier(),
             param_grid={'metric': ['minkowski'],
                         'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 19, 23, 29],
                         'p': [1, 2, 3, 4, 5],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy', verbose=1)
```

In [ ]:

In [32]:
```python
print('Estimator: \n',    grid.best_estimator_)
print('Best params : \n', grid.best_params_)
print(grid.classes_)
print(grid.best_score_)
```

```
Estimator:
 KNeighborsClassifier(n_neighbors=13, p=3)
Best params :
 {'metric': 'minkowski', 'n_neighbors': 13, 'p': 3, 'weights': 'uniform'}
[0 1 2]
0.9866666666666667
```

---

what does it mean ... "depending on the dataset the accuracies will be different?"

Accuracy depends on sepration of the data samples (training)

- separation can be achieved
  - choosing right predictors
  - choosing enough training samples
  - choosing the appr ML algo
    - configuring the ML also with optimal values

In [ ]: