CrashCoursesSklearnSupervised

December 6, 2023

#

Supervised Crash Course with sklearn

Abdulrahman ALLOUCHE / 6 December 2023

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.

https://scikit-learn.org/stable//

0.1 1. The four steps to make a prediction

- Select an **estimator** and its hyperparameters.
- Train the model using the **.fit** method.
- Evaluate the model using the **.score** method.
- Use the model to predict, using **.predict** method.

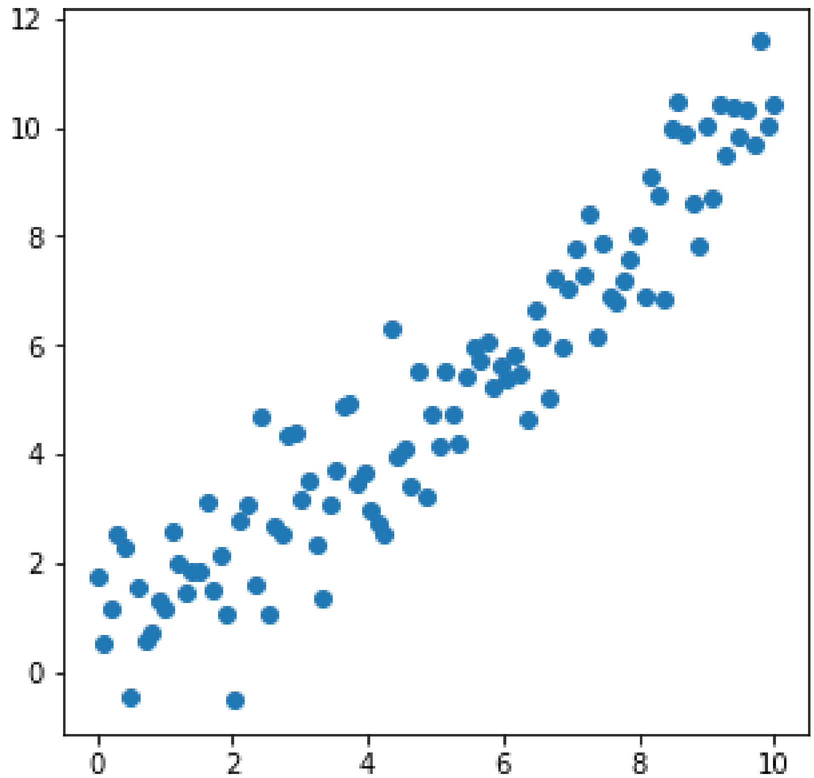
The liste of all available models are given here : https://scikit-learn.org/stable/user_guide.html

0.2 2. A simple regression problem

[88]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[55]:
```
np.random.seed(0)
m = 100 #creation of 100 samples
X = np.linspace(0, 10, m).reshape(m,1) # reshape to obtain m lines and one␣
↪column
y = X + np.random.randn(m, 1)

plt.figure(figsize=(5,5))
plt.scatter(X, y)
print("Xshape=",X.shape)
print("Yshape=",y.shape)
```

```
Xshape= (100, 1)
Yshape= (100, 1)
```



[56]:
```python
from sklearn.linear_model import LinearRegression
#The model are sorted by type. linear_model contains all linear models implemented in sklearn
```

[57]:
```python
model = LinearRegression()
model.fit(X, y) # train themodele
model.score(X, y) # evaluation using R^2
#model.get_params()
```

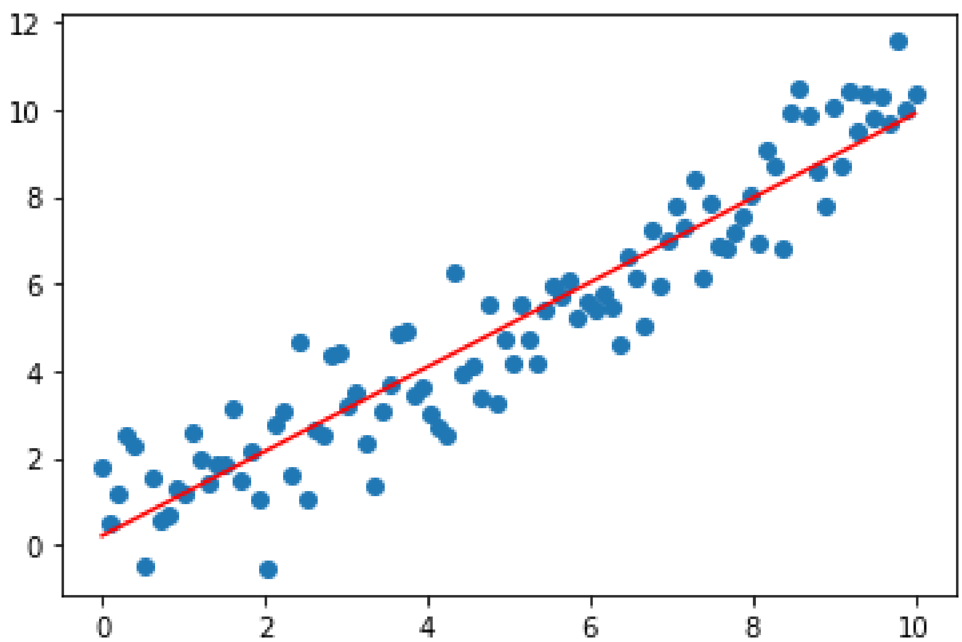
score function depends on the method used

[57]: 0.8881140743377214

[58]:
```python
Ypredict=model.predict(X)
plt.scatter(X, y)
plt.plot(X, Ypredict, c='red')
```

tracer pour comparer les predits et les scatters sont les vrai outputs

[58]: [<matplotlib.lines.Line2D at 0x7fefe4e9e880>]



0.3 3. A simple Classification problem

[59]:
```
titanic = sns.load_dataset('titanic')
titanic = titanic[['survived', 'pclass', 'sex', 'age']] # Select four columns
titanic.dropna(axis=0, inplace=True) # remove lines with nan
titanic['sex'].replace(['male', 'female'], [0, 1], inplace=True) # replace male␣
↪by 0 and female by 1
titanic.head()
```
prcque machine Learning utilise les nombre et non des strings

[59]:

y x

	survived	pclass	sex	age
0	0	3	0	22.0
1	1	1	1	38.0
2	1	3	1	26.0
3	1	1	1	35.0
4	0	3	0	35.0

[60]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[61]:
```
model = KNeighborsClassifier()
```

[62]:
```
y = titanic['survived'] # target
X = titanic.drop('survived', axis=1) # features
```
elimination des y

[63]:
```
model.fit(X, y) # train the modele
model.score(X, y) # evaluation using R^2
```

```
#model.get_params()
```

[63]: 0.8305322128851541

Survival Prediction

[64]:
```
def survival(model, pclass=1, sex=0, age=56):
  x = np.array([pclass, sex, age]).reshape(1, 3)   # 1 ligne et 3 cols
  print(model.predict(x))
  print(model.predict_proba(x))
  return model.predict(x)==1
```

[65]:
```
#survival(model)
survival(model,pclass=1,sex=0,age=57)
```

```
[0]
[[0.8 0.2]]
```

[65]: array([False])

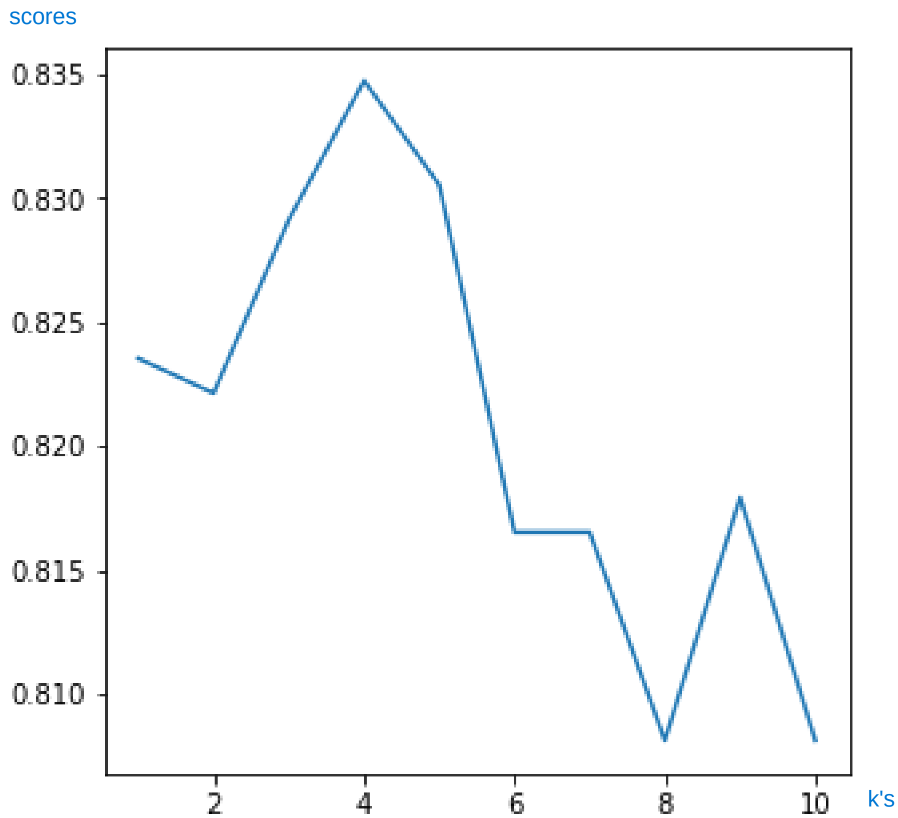
Best score by selection of n_neighbors

[66]:
```
score = []
best_k = 1      nombre de voisin
best_score = 0
best_model = None
for k in range(1, 30):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X, y)
    score.append(model.score(X, y))

    if best_score < model.score(X, y):
        best_k = k
        best_score = model.score(X, y)
        best_model = model

print("Best k=",best_k)
print("Best score =",best_score)
```

```
Best k= 4      par defaut k=5
Best score = 0.834733893557423
```

[67]:
```
plt.figure(figsize=(5,5))
#plt.plot(range(1,30), score )
n=10
plt.plot(range(1,n+1), score[0:n] )
```

[67]: [<matplotlib.lines.Line2D at 0x7fefe1610a30>]



[68]:
```
print("Survival ? ",survival(best_model,sex=0,age=57))
```

```
[0]
[[0.75 0.25]]
Survival ?  [False]
```

0.4 4. Model selection: choosing estimators and their parameters

[69]:
```
from sklearn.datasets import load_iris
iris = load_iris()
iris.target
```

[69]:
```
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])
```

y=0,1,2
trois class

[70]:
```
iris.data.shape
```

[70]: (150, 4)

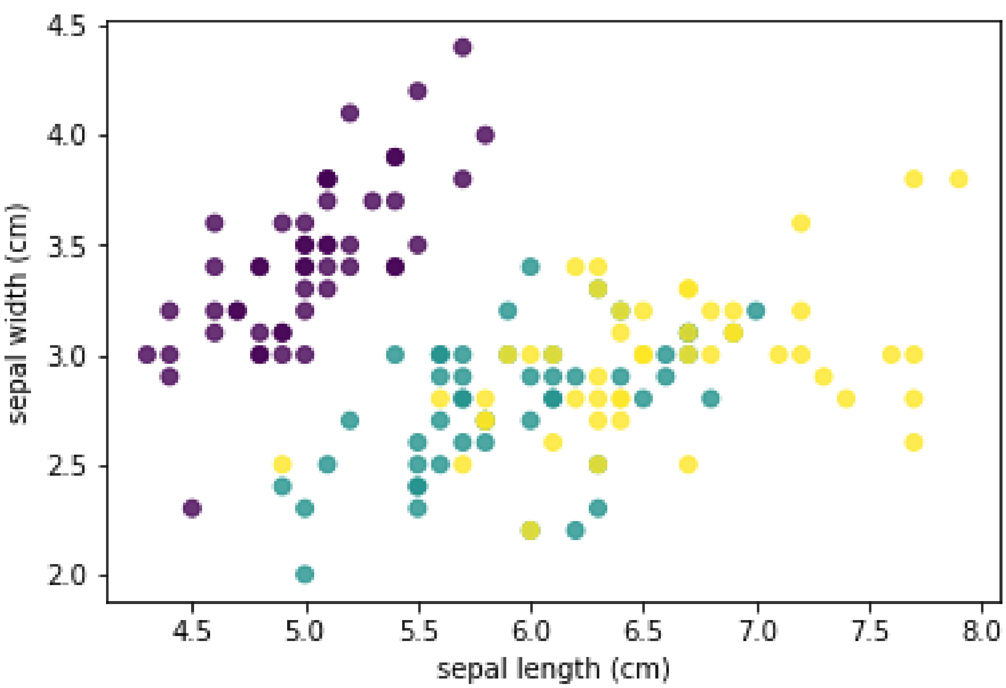
[71]:
```
iris.feature_names
#iris.data
```

[71]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[72]:
```
X = iris.data # variables
y = iris.target # target

plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8)
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
```

[72]: Text(0, 0.5, 'sepal width (cm)')

couleur= class #
3'dimension= couleur



0.4.1 4.1 Train Test Split

[73]:
```
from sklearn.model_selection import train_test_split
```

[74]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=5)

print('Train set:', X_train.shape)
print('Test set:', X_test.shape)
```

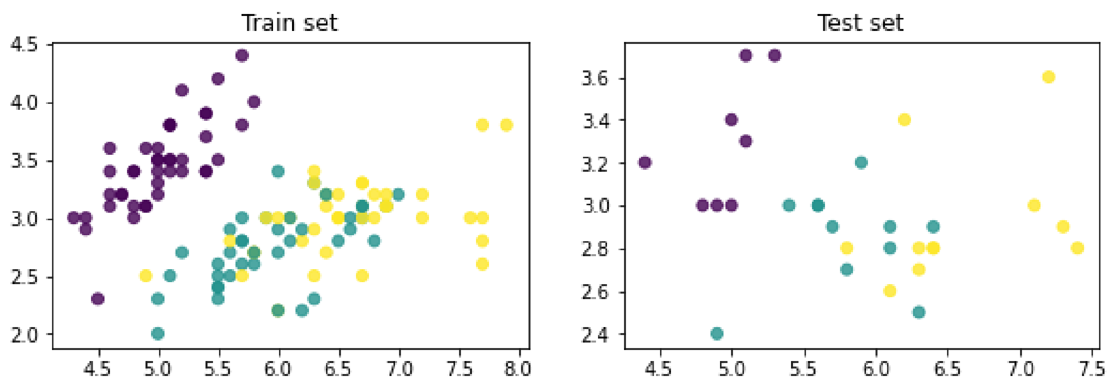
20% des datas est pris pour le test apres training

```
Train set: (120, 4)
Test set: (30, 4)
```

[75]:
```
plt.figure(figsize=(10, 3))
plt.subplot(121)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, alpha=0.8)
plt.title('Train set')
plt.subplot(122)
plt.scatter(X_test[:, 0], X_test[:, 1],c=y_test, alpha=0.8)
plt.title('Test set')
```

[75]: `Text(0.5, 1.0, 'Test set')`

le test est assez bien reparti, elle se fait au hasard



[76]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[77]:
```
model = KNeighborsClassifier(n_neighbors=3) # Try several n_neighbors values :␣
↪1, 2, 3 and see scores

model.fit(X_train, y_train)

print('train score:', model.score(X_train, y_train))
print('test score:', model.score(X_test, y_test))
```

```
train score: 0.975
test score: 0.9333333333333333
```

on remarque un over-fitting

0.4.2 4.2 Validation Set et Cross Validation

[78]:
```
from sklearn.model_selection import cross_val_score
```

[79]:
```
model = KNeighborsClassifier()
s=cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
print("scores=",s, "Mean=",s.mean())
```

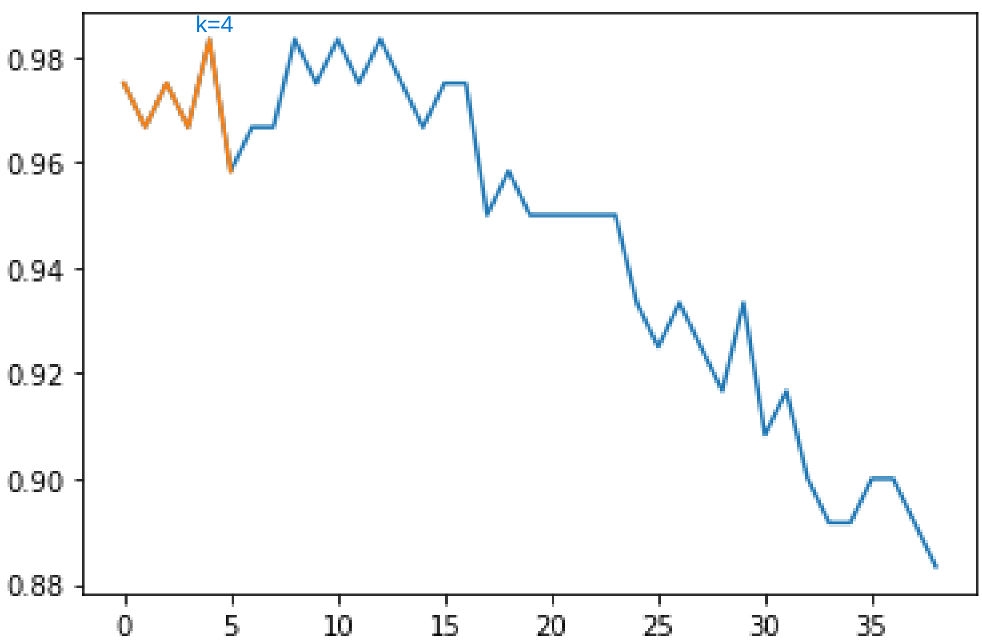
séparer le datas en 5 parties(cv)

```
scores= [1.         1.         1.         0.95833333 0.95833333] Mean=
0.9833333333333334
```

[83]:
```
val_score = []
for k in range(1, 40):
    score = cross_val_score(KNeighborsClassifier(k), X_train, y_train, cv=5).
 ↪mean()
    val_score.append(score)

plt.plot(val_score)
plt.plot(val_score[0:6])
```

5

[83]: `[<matplotlib.lines.Line2D at 0x7fefe043bc10>]`

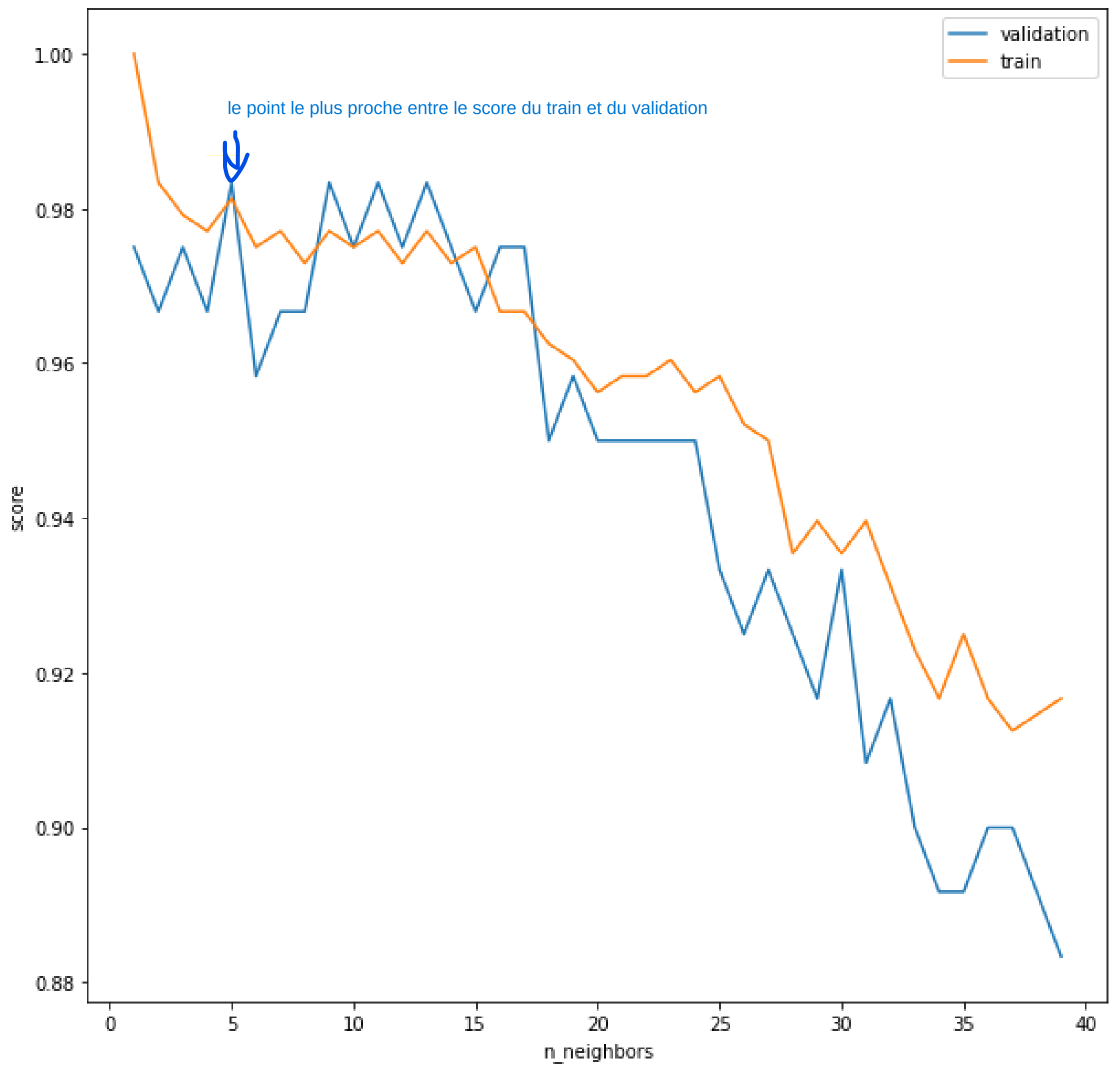


0.4.3 4.3 Validation Curve

[86]:
```
from sklearn.model_selection import validation_curve
```

[92]:
```
model = KNeighborsClassifier()
k = np.arange(1, 40)
train_score, val_score = validation_curve(estimator=model, X=X_train, y=y_train,
                                          param_name='n_neighbors',␣
 ↪param_range=k, cv=5)

plt.figure(figsize=(10,10))
plt.plot(k, val_score.mean(axis=1), label='validation')
plt.plot(k, train_score.mean(axis=1), label='train')

plt.ylabel('score')
plt.xlabel('n_neighbors')
plt.legend()
```

[92]: <matplotlib.legend.Legend at 0x7ff02a9fd970>



0.4.4 4.4 GridSearchCV

Cross-Validation(CV)

[94]:
```python
from sklearn.model_selection import GridSearchCV
```

[95]:
```python
param_grid = {'n_neighbors': np.arange(1, 20),
              'metric': ['euclidean', 'manhattan']}

grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)

grid.fit(X_train, y_train)
```

dictionary creation

metric: deux facon de calculer la distance

5 fit different

[95]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'metric': ['euclidean', 'manhattan'],
                         'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19])})
```

[96]:
```python
print(grid.best_score_)
print(grid.best_params_)
```

```
0.9833333333333334
{'metric': 'euclidean', 'n_neighbors': 5}
```

[97]:
```python
model = grid.best_estimator_
model.score(X_test, y_test) # On test data
```

[97]:
```
0.9333333333333333
```

[106]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, model.predict(X_test))
```

[106]:
```
array([[ 8,  0,  0],
       [ 0,  9,  2],
       [ 0,  0, 11]])
```

2xFaux 2

X_test = 30

0.4.5 4.5 Learning Curve

le score en fonction de la taille du datas set et non- du hyperparameter k

[107]:
```python
from sklearn.model_selection import learning_curve
```

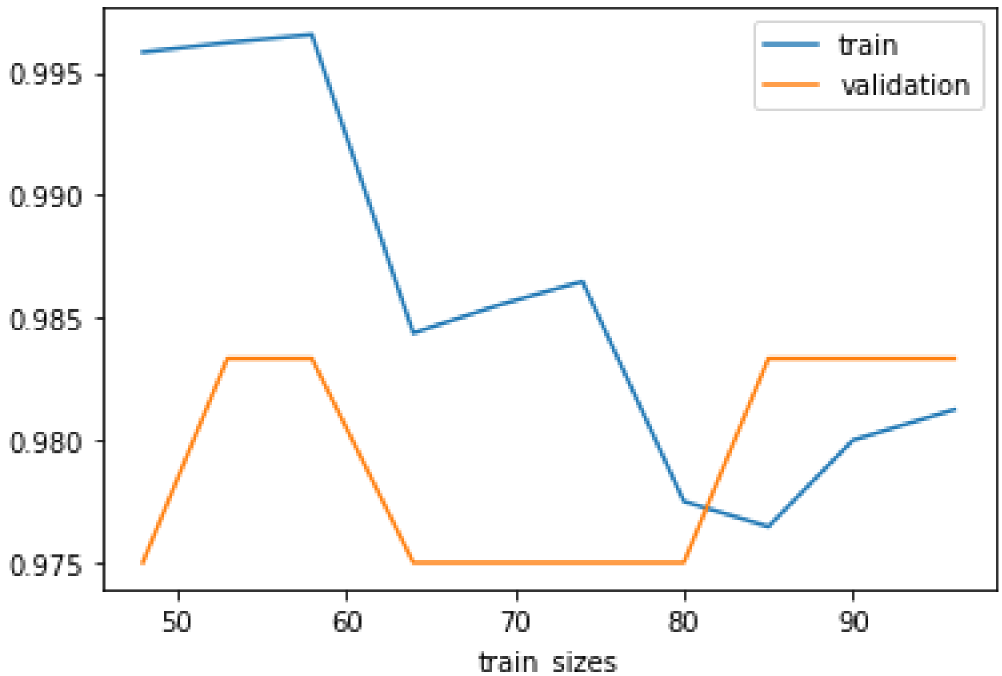
[109]:
```python
N, train_score, val_score = learning_curve(model, X_train, y_train,
                                           train_sizes=np.linspace(0.5, 1, 10),␣
 ↪cv=5)
```

[110]:
```python
print(N)
plt.plot(N, train_score.mean(axis=1), label='train')
plt.plot(N, val_score.mean(axis=1), label='validation')
plt.xlabel('train_sizes')
```

```
plt.legend()
```

[48 53 58 64 69 74 80 85 90 96]

[110]: <matplotlib.legend.Legend at 0x7ff02a9e35e0>



0.5 5. Pre-processing

0.5.1 5.1 Encoding

Encodage LabelEncoder et LabelBinarizer string => float

[111]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, LabelBinarizer # encoder for 1D
 ↪vector
```

[112]:
```
y = np.array(['cat', 'dog', 'cat', 'bird'])

encoder = LabelEncoder() # Select the encoder
encoder.fit_transform(y) # fit & transform
```

[112]: array([1, 2, 1, 0])

[113]:
```
encoder.inverse_transform(np.array([0, 0, 2])) # inverse transformet
```

[113]: array(['bird', 'bird', 'dog'], dtype='<U4')

[114]:
```
encoder = LabelBinarizer() # a vector for each value
encoder.fit_transform(y)
```

[114]:
```
array([[0, 1, 0],
       [0, 0, 1],
       [0, 1, 0],
       [1, 0, 0]])
```

tout class est representer par un vecteur

[115]:
```
#encoder.inverse_transform(np.array([[0,1,0], [0,0,1], [0,0,1]])) # inverse␣
 ↪transformet
```

Ordinal Encoding and OneHot Encoding

[116]:
```
import matplotlib.pyplot as plt
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder # for Tensor␣
 ↪like X
X = np.array( [['cat', 'hair'],
               ['dog', 'hair'],
               ['cat', 'hair'],
               ['bird', 'feathers']])

encoder = OrdinalEncoder()
encoder.fit_transform(X)
```

[116]:
```
array([[1., 1.],
       [2., 1.],
       [1., 1.],
       [0., 0.]])
```

[118]:
```
#encoder = OneHotEncoder()
#encoder.fit_transform(X)
#print(encoder.fit_transform(X))
```

0.5.2 5.2 Normalization

Many machine learning **algorithms perform better** when numerical input variables are scaled to a **standard range**. * Use MinMaxScaler as your default * Use RobustScaler if you have outliers and can handle a larger range * Use StandardScaler if you need normalized features

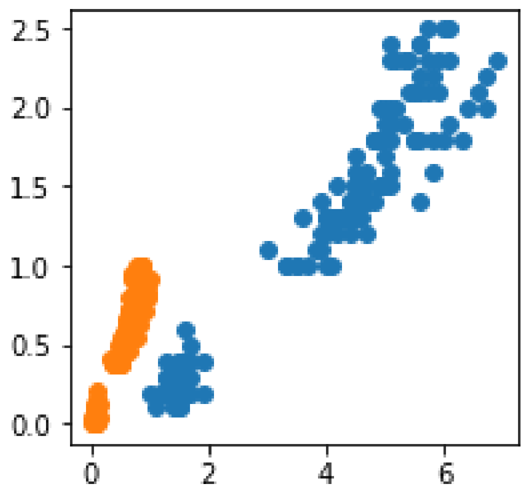
[119]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
```

MinMaxScaler MinMaxScaler subtracts the column mean from each value and then divides by the range.

$X_{scale} = \frac{x - x_{min}}{x_{max} - x_{min}}$

[120]:
```
X_minmax = MinMaxScaler().fit_transform(X)
plt.figure(figsize=(3,3))
plt.scatter(X[:, 2], X[:, 3])
plt.scatter(X_minmax[:, 2], X_minmax[:, 3])
```

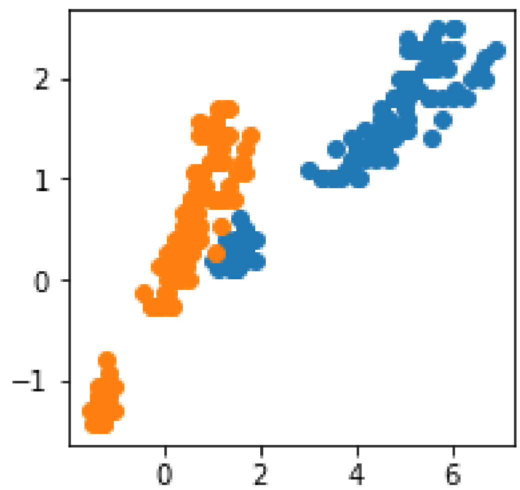
[120]: <matplotlib.collections.PathCollection at 0x7ff02a9064c0>



StandardScaler $z = \frac{x-\mu}{\sigma}$; $\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$; $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$

[121]:
```
X_stdscl = StandardScaler().fit_transform(X)
plt.figure(figsize=(3,3))
plt.scatter(X[:, 2], X[:, 3])
plt.scatter(X_stdscl[:, 2], X_stdscl[:, 3])
```

[121]: <matplotlib.collections.PathCollection at 0x7ff02a859d90>

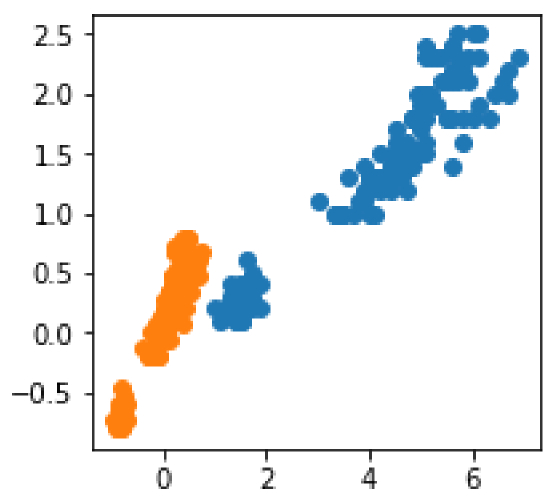


RobustScaler RobustScaler subtracts the column median and divides by the interquartile range.

$X_{scale} = \frac{x - x_{med}}{x_{75} - x_{25}}$

[122]:
```
X_robust = RobustScaler().fit_transform(X)
plt.figure(figsize=(3,3))
plt.scatter(X[:, 2], X[:, 3])
plt.scatter(X_robust[:, 2], X_robust[:, 3])
```

[122]:
```
<matplotlib.collections.PathCollection at 0x7ff02a842ac0>
```

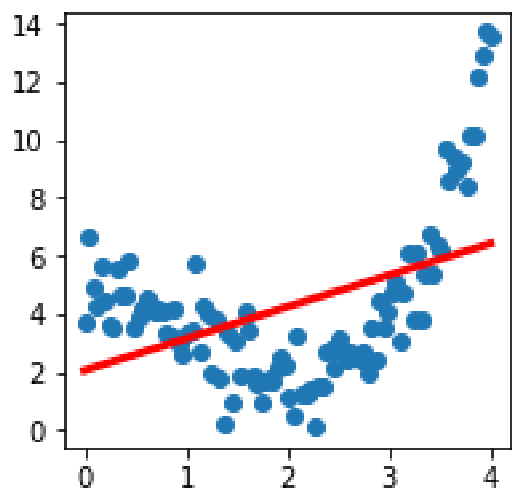


0.5.3 5.3 Polynomial Features

[153]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

[154]:
```
m = 100
X = np.linspace(0, 4, m).reshape((m, 1))
y = X**2 + 5*np.cos(X) + np.random.randn(m, 1)
plt.figure(figsize=(3,3))
model = LinearRegression().fit(X, y)
y_pred = model.predict(X)

plt.scatter(X, y)
plt.plot(X, y_pred, c='r', lw=3)
```

[154]:
```
[<matplotlib.lines.Line2D at 0x7fefc98126a0>]
```



[155]:
```
model.coef_
```
a param : a vector

[155]:
```
array([[1.09048616]])
```

[156]:
```
model.intercept_
```
b parameter : a vector

[156]:
```
array([2.05474357])
```

[157]:
```
model.score(X,y)
```

[157]:
```
0.20173223522197947
```

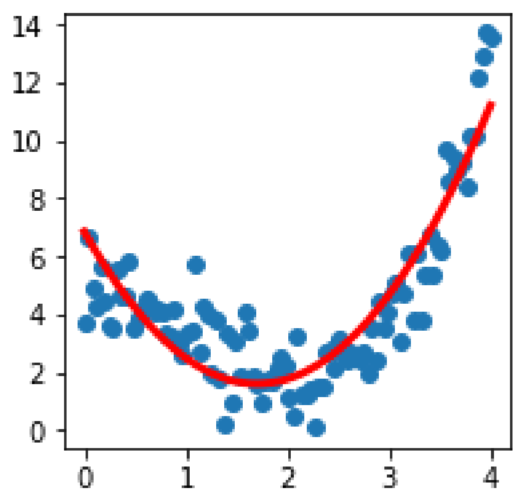
[158]:
```
#X
```

[159]:
```
#X_poly = PolynomialFeatures(2).fit_transform(X) # try with different ordrer
#X_poly
```

2 degree polynome

[160]:
```
X_poly = PolynomialFeatures(2).fit_transform(X) # try with different ordrer
model = LinearRegression().fit(X_poly, y)
y_pred = model.predict(X_poly)
plt.figure(figsize=(3,3))
plt.scatter(X, y)
plt.plot(X, y_pred, c='r', lw=3)
model.score(X_poly,y)
```

[160]:
```
0.8101960197652978
```



[161]:
```
print(X.shape)
print(X_poly.shape)
```

```
(100, 1)
(100, 3)
```

[162]:
```
model.coef_
```

[162]:
```
array([[ 0.        , -6.17253942,  1.8157564 ]])
```

[163]:
```
model.intercept_
```

[163]:
```
array([6.84785136])
```

0.5.4 5.5. Pipelines

une fonction qui permet de combiner/lier deux fonctions

[164]:
```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

[165]:
```
iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=100)
#X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,␣
↪test_size=0.3, shuffle=False)
#X_train=X
#y_train=y
```

[166]:
```
#model = LogisticRegression() # bad resul
#model = make_pipeline(StandardScaler(), LogisticRegression())
model = make_pipeline(PolynomialFeatures(), StandardScaler(),␣
↪LogisticRegression())

model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[166]:
```
0.9736842105263158
```

[171]:
```
model = make_pipeline(PolynomialFeatures(),
                      StandardScaler(),
                      LogisticRegression(solver="liblinear"))
params = {
    'polynomialfeatures__degree':[2, 3, 4],
    'logisticregression__penalty':['l1', 'l2']
    #'logisticregression__penalty':['l2']
}

grid = GridSearchCV(model, param_grid=params, cv=4)

grid.fit(X_train, y_train)
```

[171]:
```
GridSearchCV(cv=4,
             estimator=Pipeline(steps=[('polynomialfeatures',
                                        PolynomialFeatures()),
                                       ('standardscaler', StandardScaler()),
                                       ('logisticregression',
LogisticRegression(solver='liblinear'))]),
             param_grid={'logisticregression__penalty': ['l1', 'l2'],
                         'polynomialfeatures__degree': [2, 3, 4]})
```

[172]:
```
grid.score(X_test, y_test)
```

[172]: 0.9736842105263158

[173]:
```
print(grid.best_params_)
```

```
{'logisticregression__penalty': 'l1', 'polynomialfeatures__degree': 4}
```

[174]:
```
model = grid.best_estimator_
model.score(X_train, y_train) # On test data
```

[174]: 0.9642857142857143

[175]:
```
model.score(X_test, y_test)
```

[175]: 0.9736842105263158

[176]:
```
model.score(X_train, y_train)
```

[176]: 0.9642857142857143

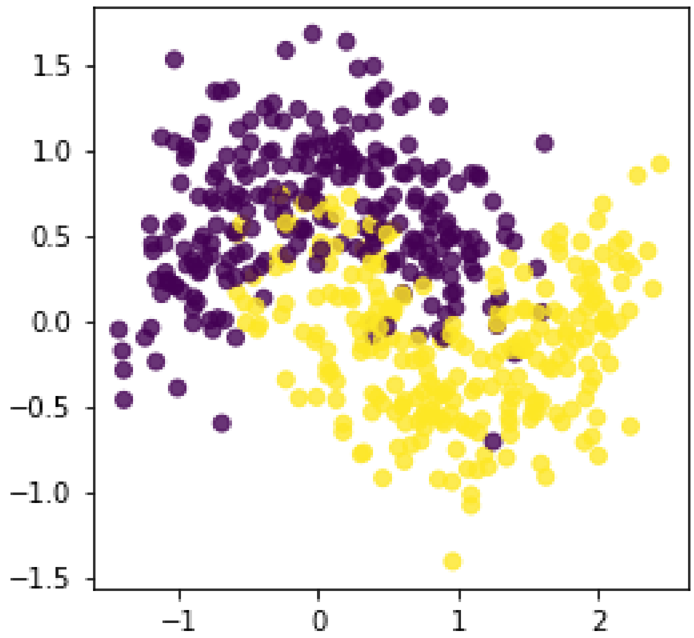
[177]:
```
#import pandas as pd
#print(grid.cv_results_)
#print(pd.DataFrame(grid.cv_results_))
```

0.6 7. Ensemble Learning

The sklearn.ensemble module includes ensemble-based methods for classification, regression and anomaly detection.

[178]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
```

[179]:
```
X, y = make_moons(n_samples=500, noise=0.3, random_state=0)
plt.figure(figsize=(4,4))
plt.scatter(X[:,0], X[:,1], c=y, alpha=0.8)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=0)
```



0.6.1 7.1. Voting Classifier

[182]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
```

[183]:
```
model_1 = LogisticRegression()
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = KNeighborsClassifier(n_neighbors=2)

model_4 = VotingClassifier([
                            ('SGD', model_1),
                            ('Tree', model_2),
                            ('KNN', model_3)],
                          voting='hard')

for model in (model_1, model_2, model_3, model_4):
    model.fit(X_train, y_train)
    print(model.__class__.__name__, model.score(X_test, y_test))
```

```
LogisticRegression 0.82
DecisionTreeClassifier 0.84
KNeighborsClassifier 0.84
VotingClassifier 0.85
```

score ici = Accuracy defini avant

0.6.2 7.2. Bagging

[184]:
```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
```

[185]:
```
model = BaggingClassifier(base_estimator=KNeighborsClassifier(),
                          n_estimators=100)
                                 prise de 100 model differents( de type Forest)

model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[185]: 0.875

[186]:
```
model = RandomForestClassifier(n_estimators=100)

model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[186]: 0.875

0.6.3 7.3. Boosting

[187]:
```
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
```

[188]:
```
model = AdaBoostClassifier(n_estimators=100)
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[188]: 0.88

[189]:
```
model = GradientBoostingClassifier(n_estimators=100)
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[189]: 0.87

0.6.4 7.4. Stacking

[190]:
```
from sklearn.ensemble import StackingClassifier
```

[191]:
```
model = StackingClassifier([('SGD', model_1),
                            ('Tree', model_2),
                            ('KNN', model_3)],
                            final_estimator=KNeighborsClassifier())

model.fit(X_train, y_train)
model.score(X_test, y_test)
```

[191]: 0.88

[ ]:

```

```

[ ]:

```

```