**8-puzzel**

class Solution:

def solve(self, board):

dict = {}

flatten = []

for i in range(len(board)):

row = board[i]

for j in range(len(row)):

flatten.append(row[j])

flatten = tuple(flatten)

dict[flatten] = 0

if flatten == (0, 1, 2, 3, 4, 5, 6, 7, 8):

return 0

return self.get\_paths(dict)

def get\_paths(self, dict):

cnt = 0

while True:

current\_nodes = [x for x in dict if dict[x] == cnt]

if not current\_nodes:

return -1

for node in current\_nodes:

next\_moves = self.find\_next(node)

for move in next\_moves:

if move not in dict:

dict[move] = cnt + 1

if move == (0, 1, 2, 3, 4, 5, 6, 7, 8):

return cnt + 1

cnt += 1

def find\_next(self, node):

moves = {

0: [1, 3],

1: [0, 2, 4],

2: [1, 5],

3: [0, 4, 6],

4: [1, 3, 5, 7],

5: [2, 4, 8],

6: [3, 7],

7: [4, 6, 8],

8: [5, 7]

}

results = []

pos\_0 = node.index(0)

for move in moves[pos\_0]:

new\_node = list(node)

new\_node[move], new\_node[pos\_0] = new\_node[pos\_0], new\_node[move]

results.append(tuple(new\_node))

return results

ob = Solution()

matrix = [

[3, 1, 2],

[4, 7, 5], [6, 8, 0]]

print(ob.solve(matrix))

class Solution:

    def solve(self, board):

        dict = {}

        flatten = []

        for i in range(len(board)):

            row = board[i]

            for j in range(len(row)):

                flatten.append(row[j])

        flatten = tuple(flatten)

        dict[flatten] = 0

        if flatten == (0, 1, 2, 3, 4, 5, 6, 7, 8):

            return 0

        return self.get\_paths(dict)

    def get\_paths(self, dict):

        cnt = 0

        while True:

            current\_nodes = [x for x in dict if dict[x] == cnt]

            if not current\_nodes:

                return -1

            for node in current\_nodes:

                next\_moves = self.find\_next(node)

                for move in next\_moves:

                    if move not in dict:

                        dict[move] = cnt + 1

                    if move == (0, 1, 2, 3, 4, 5, 6, 7, 8):

                        return cnt + 1

            cnt += 1

    def find\_next(self, node):

        moves = {

            0: [1, 3],

            1: [0, 2, 4],

            2: [1, 5],

            3: [0, 4, 6],

            4: [1, 3, 5, 7],

            5: [2, 4, 8],

            6: [3, 7],

            7: [4, 6, 8],

            8: [5, 7]

        }

        results = []

        pos\_0 = node.index(0)

        for move in moves[pos\_0]:

            new\_node = list(node)

            new\_node[move], new\_node[pos\_0] = new\_node[pos\_0], new\_node[move]

            results.append(tuple(new\_node))

        return results

ob = Solution()

matrix = [

    [3, 1, 2],

    [4, 7, 5], [6, 8, 0]]

print(ob.solve(matrix))

**KNN**

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

names=['sepal-length','sepal-width','petal-length','petal-width','class']

dataset = pd.read\_csv ("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data")

x=dataset.iloc[:,0:-1]

y=dataset.iloc[:,-1]

print(x.head())

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.2)

classifier=KNeighborsClassifier(n\_neighbors=5).fit(xtrain,ytrain)

ypred=classifier.predict(xtest)

i=0

print('%-25s %-25s %-25s'%('original label','predicted label','correct/wrong'))

for label in ytest:

print('%-25s %-25s'%(label, ypred[i]),end="")

if label==ypred[i]:

print('%-25s'%('correct'))

else:

print('%-25s'%('wrong'))

i=i+1

print('\nConfusion Matrix:\n',metrics.confusion\_matrix(ytest,ypred))

print('\nclassification report:\n',metrics.classification\_report(ytest,ypred))

print('accuracy of the classifier is %0.2f'%metrics.accuracy\_score(ytest, ypred))

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

names=['sepal-length','sepal-width','petal-length','petal-width','class']

dataset = pd.read\_csv ("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data")

x=dataset.iloc[:,0:-1]

y=dataset.iloc[:,-1]

print(x.head())

xtrain,xtest,ytrain,ytest=train\_test\_split(x,y,test\_size=0.2)

classifier=KNeighborsClassifier(n\_neighbors=5).fit(xtrain,ytrain)

ypred=classifier.predict(xtest)

i=0

print('%-25s %-25s %-25s'%('original label','predicted label','correct/wrong'))

for label in ytest:

    print('%-25s %-25s'%(label, ypred[i]),end="")

    if label==ypred[i]:

        print('%-25s'%('correct'))

    else:

        print('%-25s'%('wrong'))

        i=i+1

print('\nConfusion Matrix:\n',metrics.confusion\_matrix(ytest,ypred))

print('\nclassification report:\n',metrics.classification\_report(ytest,ypred))

print('accuracy of the classifier is %0.2f'%metrics.accuracy\_score(ytest, ypred))

**Linear regression**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

np.random.seed (0)

X = 2\* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

model = LinearRegression()

model.fit(X, y)

X\_new = np.array([[0], [2]])

y\_pred = model.predict(X\_new)

plt.scatter(X, y, color='blue')

plt.plot(X\_new, y\_pred, color='red')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Linear Regression')

plt.show()

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

np.random.seed (0)

X = 2\* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

model = LinearRegression()

model.fit(X, y)

X\_new = np.array([[0], [2]])

y\_pred = model.predict(X\_new)

plt.scatter(X, y, color='blue')

plt.plot(X\_new, y\_pred, color='red')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Linear Regression')

plt.show()

**naïve bayers**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, confusion\_matrix

file\_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

iris\_data=pd.read\_csv(file\_path)

X=iris\_data.iloc[:, :-1].values

Y=iris\_data.iloc[:, -1].values

X\_train, X\_test, Y\_train, Y\_test =train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

scaler =StandardScaler()

X\_train =scaler.fit\_transform(X\_train)

X\_test =scaler.transform(X\_test)

naive\_bayes =GaussianNB()

naive\_bayes.fit(X\_train, Y\_train)

Y\_pred =naive\_bayes.predict(X\_test)

accuracy = accuracy\_score(Y\_test, Y\_pred)

print("Accuracy:", accuracy)

conf\_matrix =confusion\_matrix(Y\_test, Y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, confusion\_matrix

file\_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

iris\_data=pd.read\_csv(file\_path)

X=iris\_data.iloc[:, :-1].values

Y=iris\_data.iloc[:, -1].values

X\_train, X\_test, Y\_train, Y\_test =train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

scaler =StandardScaler()

X\_train =scaler.fit\_transform(X\_train)

X\_test =scaler.transform(X\_test)

naive\_bayes =GaussianNB()

naive\_bayes.fit(X\_train, Y\_train)

Y\_pred =naive\_bayes.predict(X\_test)

accuracy = accuracy\_score(Y\_test, Y\_pred)

print("Accuracy:", accuracy)

conf\_matrix =confusion\_matrix(Y\_test, Y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

**SVM**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.datasets import load\_iris

file\_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

iris\_data=pd.read\_csv(file\_path)

X= iris\_data.iloc[:, :-1].values

Y=iris\_data.iloc[:, -1].values

X\_train, X\_test, Y\_train, Y\_test=train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

scaler=StandardScaler()

X\_train=scaler.fit\_transform(X\_train)

X\_test=scaler.transform(X\_test)

svm\_classifier=SVC(kernel='linear', random\_state=42)

svm\_classifier.fit(X\_train, Y\_train)

Y\_pred=svm\_classifier.predict(X\_test)

accuracy=accuracy\_score(Y\_test, Y\_pred)

print("Accuracy:", accuracy)

conf\_matrix=confusion\_matrix(Y\_test, Y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.datasets import load\_iris

file\_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

iris\_data=pd.read\_csv(file\_path)

X= iris\_data.iloc[:, :-1].values

Y=iris\_data.iloc[:, -1].values

X\_train, X\_test, Y\_train, Y\_test=train\_test\_split(X, Y, test\_size=0.2, random\_state=42)

scaler=StandardScaler()

X\_train=scaler.fit\_transform(X\_train)

X\_test=scaler.transform(X\_test)

svm\_classifier=SVC(kernel='linear', random\_state=42)

svm\_classifier.fit(X\_train, Y\_train)

Y\_pred=svm\_classifier.predict(X\_test)

accuracy=accuracy\_score(Y\_test, Y\_pred)

print("Accuracy:", accuracy)

conf\_matrix=confusion\_matrix(Y\_test, Y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)