import numpy as np

import matplotlib.pyplot as plt

from sklearn.svm import SVC

from sklearn.datasets import load\_breast\_cancer

from sklearn.preprocessing import MinMaxScaler

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

from sklearn.metrics import roc\_auc\_score

!pip install bayesian-optimization

from bayes\_opt import BayesianOptimization, UtilityFunction

import warnings

warnings.filterwarnings("ignore")

cancer = load\_breast\_cancer()

X = cancer["data"]

y = cancer["target"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,stratify = y,

                                        random\_state = 42)

scaler = MinMaxScaler()

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

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

# Define the black box function to optimize.

def black\_box\_function(C):

    # C: SVC hyper parameter to optimize for.

    model = SVC(C = C)

    model.fit(X\_train\_scaled, y\_train)

    y\_score = model.decision\_function(X\_test\_scaled)

    f = roc\_auc\_score(y\_test, y\_score)

    return f# Set range of C to optimize for.

# bayes\_opt requires this to be a dictionary.

pbounds = {"C": [0.1, 10]}

# Create a BayesianOptimization optimizer,

# and optimize the given black\_box\_function.

optimizer = BayesianOptimization(f = black\_box\_function,

                                 pbounds = pbounds, verbose = 2,

                                 random\_state = 4)

optimizer.maximize(init\_points = 5, n\_iter = 10)

print("Best result: {}; f(x) = {}.".format(optimizer.max["params"], optimizer.max["target"]))

Running the Python code above prints the following output:

| iter | target | C |

-------------------------------------

| 1 | 0.9979 | 9.674 |

| 2 | 0.9975 | 5.518 |

| 3 | 0.9979 | 9.73 |

| 4 | 0.9979 | 7.177 |

| 5 | 0.9979 | 7.008 |

| 6 | 0.9914 | 0.1023 |

**| 7 | 0.9981 | 8.506** |

| 8 | 0.9981 | 8.15 |

| 9 | 0.9981 | 8.327 |

| 10 | 0.9981 | 8.8 |

| 11 | 0.9981 | 8.671 |

| 12 | 0.9981 | 7.974 |

| 13 | 0.9979 | 6.273 |

| 14 | 0.9981 | 8.064 |

| 15 | 0.9981 | 8.911 |

Best result: {'C': 8.50571739015795}; f(x) = 0.9981132075471698.

From the results above, the optimizer managed to determine that using the hyper parameter value of C = 8.505 results in the best performing model!

VALIDATE

model1 = SVC(C = 8.505)

model1.fit(X\_train\_scaled, y\_train)

y\_score = model1.decision\_function(X\_test\_scaled)

f = roc\_auc\_score(y\_test, y\_score)

print(f)

0.99811

You might have realized that the optimizer outputs the search parameter as a continuous variable. This will lead to a problem if the parameter must be discrete.

As an example, let us assume that we also want to search for the best degree value of the SVC model, however degree must be an integer. In such a case we need more control over the optimization process.

Also, we did not specify the hyper parameter kappa of the acquisition function *a*(*x*) above, nor did we specify what type of acquisition function to use. In general the default settings should work in most cases, but for some cases it would be good to have a little more control over the optimizer.

Luckily, instead of using the simple workflow shown above, bayes\_opt also allows for a more controlled optimization process. In this case, we have to manually perform each optimization step in a for loop. Inside this for loop we can add additional code to perform other calculations if required, such as forcing the search parameters to be discrete.

# The black box is not here, it will be called later on.

optimizer = BayesianOptimization(f = None,

                                 pbounds = {"C": [0.01, 10],

                                            "degree": [1, 5]},

                                 verbose = 2, random\_state = 1234)

Specify the acquisition function (bayes\_opt uses the term utility function) to be the upper confidence bounds "ucb".

Set kappa = 1.96 to balance exploration vs exploitation.

xi = 0.01 is another hyper parameter required but is not used by "ucb".

Other acquisition functions such as the expected improvement "ei" will be affected by xi.

utility = UtilityFunction(kind = "ucb", kappa = 1.96, xi = 0.01)

**# We want to optimize both C and degree simultaneously.**

def black\_box\_function(C, degree):

    model = SVC(C = C, degree = degree)

    model.fit(X\_train\_scaled, y\_train)

    y\_score = model.decision\_function(X\_test\_scaled)

    f = roc\_auc\_score(y\_test, y\_score)

    return f# Optimization for loop.

for i in range(25):

    # Get optimizer to suggest new parameter values to try using the

    # specified acquisition function.

    next\_point = optimizer.suggest(utility)

    # Force degree from float to int.

    next\_point["degree"] = int(next\_point["degree"])

    # Evaluate the output of the black\_box\_function using the new parameter values.

    target = black\_box\_function(\*\*next\_point)

try:

        # Update the optimizer with the evaluation results.

        optimizer.register(params = next\_point, target = target)

    except:

        pass

print("Best result: {}; f(x) = {:.3f}.".format(optimizer.max["params"], optimizer.max["target"]))

**OUTPUT**

**Best result: {'C': 9.98421583, 'degree': 4.0}; f(x) = 0.998.**

Using the hyper parameter value of C = 9.984 and degree = 4 results in the best performing SVC model! 0.998

Y = optimizer.space.target

Cuantos = len(optimizer.space.target)

plt.figure(figsize = (15, 5))

plt.plot(range(1, 1 + Cuantos), Y, "-o")

plt.grid(True)

plt.xlabel("Iteration", fontsize = 14)

plt.ylabel("Black box function f(x)", fontsize = 14)

plt.xticks(fontsize = 14)

plt.yticks(fontsize = 14)

plt.show()

Chart, line chart

Description automatically generated

Bayesian optimization of C and degree of an SVC model over 25 iterations.

VALIDATE

model1 = SVC(C = 8.505, degree =4)

model1.fit(X\_train\_scaled, y\_train)

y\_score = model1.decision\_function(X\_test\_scaled)

f = roc\_auc\_score(y\_test, y\_score)

print(f)

0.99811