Assignment 1 Automated ML

George Doukeris s3536149, Antoni Czernek s4000595

October 2023

1 Introduction

This is the report of a test that we did for the purpose on an assignment on the course "Automated Machine Learning". In this assignment, we learned how to implement Bayesian Optimization, in particular Sequential Model-based Optimization (SMBO). SMBO is a powerful technique that is used for the purpose of Automated Machine Learning - to find the ideal hyperparameters with which the model produces the best results. This assignment has two purposes, to implement the SMBO and to test it on a few different datasets. For algorithms, we decided to use SVM (Support Vector Machines) and Adaboost.

2 Algorithms

2.1 SVM

The Support Vector Machine (SVM) algorithm is a classification technique that finds the best possible line or hyperplane to separate different classes in a dataset. It does this by maximizing the margin, which is the distance between the decision boundary and the nearest data points from each class. SVMs use a clever mathematical technique called the kernel trick to handle complex, non-linear relationships in the data. This allows them to effectively work in high-dimensional spaces. The focus of SVM is on support vectors, which are the data points closest to the decision boundary.

2.2 Adaboost

Adaboost (Adaptive Boosting) is an ensemble learning method that aims to enhance the accuracy of machine learning models. It works by training a series of simple models (weak learners). Initially, all data points are assigned equal importance. The first weak learner is trained to perform slightly better than random chance. Adaboost then assesses the weak learner's performance and assigns an importance weight, called an alpha value, based on its error rate. The algorithm adjusts the weights of the data points, focusing more on those that were previously misclassified.

3 Results

After we implemented SMBO we tested its performance and compared it to RandomSearch and GridSearch on two datasets: The first one is the *Electricity*, which is a dataset about electricity transfers between neighboring states in Australia. It includes 7 numeric features 1 nominal feature and 1 Class (target) which is also nominal. The second dataset that we found is the qsar-biodeg. The QSAR biodegradation dataset was built in the Milano Chemometrics and QSAR Research Group. It includes 6 numeric features and 1 Class (target) which is nominal. We have chosen these datasets because of the amount of numerical type input and the fact that they are both binary classification problems. The primary dataset we tested is the Electricity set, as it contains much more examples than the qsar-biodeq.

3.1 GridSearch

We noticed that setting the grid of the parameters for a GridSearch algorithm is very important to get good results, but it is very time-consuming to find the specific grid that would work the best, and if we put a huge amount of values in the gird it would take a lot of run-time to fit. Setting the parameter grid for this algorithm usually requires a number of

runs, as at first, we need to get an overall understating of how different parameters perform on a dataset, and then we can make the grid denser to find a good solution.

3.2 RandomizedSearch

RandomizedSearch could be faster than GridSearch in finding the best solution, but as it goes by random, the chances of finding the exact best solution are low. In our tests we set the distribution from which the parameters are drawn the same as is used in Bayesian Optimization, the number of fits is very similar as we want to give the same amount of run-time budget to all of the algorithms.

3.3 Bayesian Optimization

Bayesian Optimization algorithm in its approach solves the issues mentioned earlier, by finding the most likely hyperparameter solution that would increase the result. One of the conclusions that we arrived at is that it also requires some tuning to get the best results. For example, changing the range of the hyperparameters, or increasing the number of initial configurations helps the algorithm give better accuracy.

3.4 Performance of the algorithms

We noticed that sometimes the GridSearch or RandomSearch would give us better results than the Bayesian Optimization. We assume the reason is that the Bayesian algorithm sometimes stays in the local minimum and does not expect any improvement from other configurations. Even though the SMBO algorithm would get very similar results to the GridSearch and RandomizedSearch without a lot of fine-tuning, most of the time it gets better results than the other two. The performance of the algorithms on the qsar data set showed us that when a problem has a low amount of data, the SMBO struggles to get good results, as some changes in the parameters do not make big changes in the results (or even stay the same). We assume that could be fixed instead of relying on the accuracy we could be minimizing a loss function in the SMBO.

We can see how the algorithm works in the graphs, GridSearch is a grid on the hyperparameters and it sometimes misses the exact maximum. The RandomSearch goes randomly on the plain, and the chances of getting into the specific maximum are low, although it had a few good guesses. Bayesian Optimization clearly goes by expected improvement, as when it finds a good pair of parameters it will also check the neighborhood of that pair and look for a better result. We can see that clearly in the AdaBoost performance graph, where the purple dots are the runs from the Bayesian algorithm, black triangles are the runs from Randomized Search, and black "x" are the results of the GridSearch.

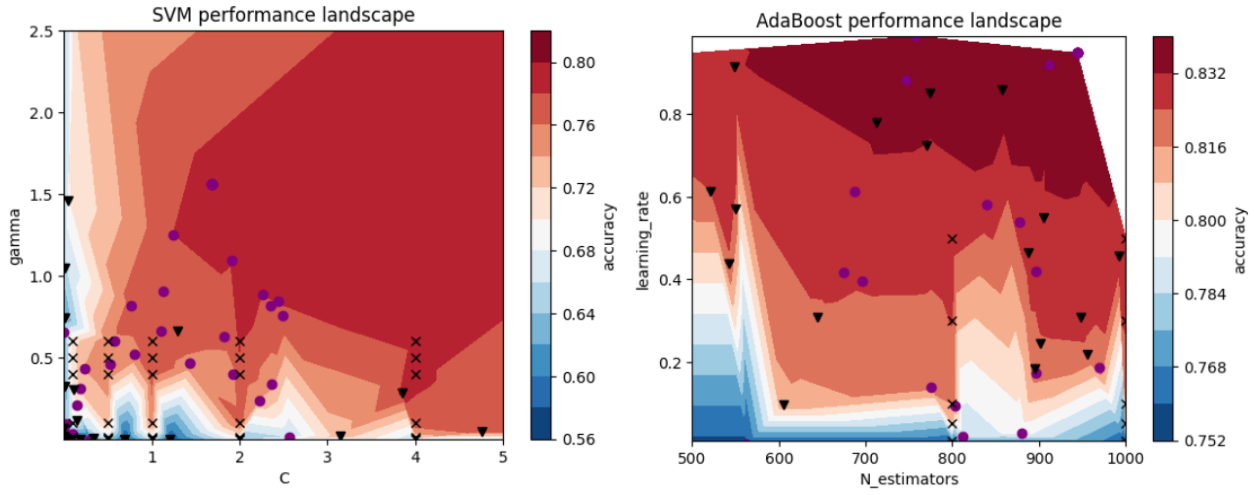

Figure 1: **Left:** Performance landscape of SVM on the Electricity Data set. **Right:** Performance landscape of AdaBoost Classifier on the Electricity Data set.

4 References

Scikit-learn, accessed October 11th 2023.

Dataset Electricity, accessed October 20th 2023.

Dataset qsar-biodeg, accessed October 11th 2023.

Paper on SMBO.

Paper on Practical Bayesian Optimization.

5 Appendix

Algorithm 1 Adaboost Algorithm

procedure ADABOOST($Data, T$)
  Initialize weights for data points: $w(i) = 1/n$ for $i = 1$ to n
  Initialize a list of weak classifiers, $h_t(x)$, for $t = 1, 2, ..., T$
  for $t = 1$ to T **do**
    Normalize the weights: $w(i) = w(i)/\sum w(j)$ ▷ Ensure sum of weights is 1
    Train a weak classifier using weighted training data: $h_t(x) = TrainWeakClassifier(Data, w)$
    Compute the error of the weak classifier: $\varepsilon_t = \sum w(i) \cdot (h_t(x_i) \neq y_i)$
    Compute the importance of the weak classifier: $\alpha_t = 0.5 \cdot \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
    for $i = 1$ to n **do**
      Update the weights: $w(i) = w(i) \cdot \exp\left(-\alpha_t \cdot y(i) \cdot h_t(x_i)\right)$
    end for
    Add the weak classifier and its importance to the ensemble: (h_t, α_t)
  end for
  return Ensemble of weak classifiers
end procedure
procedure CLASSIFY($x, Ensemble$)
  Initialize $score = 0$
  for each (h_t, α_t) in Ensemble **do**
    $score = score + \alpha_t \cdot h_t(x)$
  end for
  return sign($score$) ▷ Final prediction
end procedure

Algorithm 2 Sequential Minimal Optimization for SVM with RBF Kernel

procedure SMO-SVM-RBF($Data, C, tol, max\_iterations, \gamma$)
  Initialize α as a vector of zeros ▷ Lagrange multipliers
  Initialize $b = 0$ ▷ Threshold
  Initialize E as a vector of zeros ▷ Error cache
  for $iterations = 1$ to $max\_iterations$ **do**
    $num\_changed\_alphas = 0$
    for $i = 1$ to n **do**
      Calculate error for data point i:
      $E[i] = b - y[i] + \sum_{j=1}^{n} \alpha[j] \cdot y[j] \cdot K(x_i, x_j)$ ▷ Using RBF kernel
      if $(y[i] \cdot E[i] < -tol$ and $\alpha[i] < C)$ or $(y[i] \cdot E[i] > tol$ and $\alpha[i] > 0)$ **then**
        Randomly select $j \neq i$
        Calculate error for data point j:
        $E[j] = b - y[j] + \sum_{j=1}^{n} \alpha[j] \cdot y[j] \cdot K(x_j, x_i)$ ▷ Using RBF kernel
        Save old α values for i and j:
        $\alpha_{old_i} = \alpha[i]$
        $\alpha_{old_j} = \alpha[j]$
        Compute L and H, bounds on $\alpha[j]$:
        if $y[i] \neq y[j]$ **then**
          $L = \max(0, \alpha[j] - \alpha[i])$
          $H = \min(C, C + \alpha[j] - \alpha[i])$
        else
          $L = \max(0, \alpha[i] + \alpha[j] - C)$
          $H = \min(C, \alpha[i] + \alpha[j])$
        end if
        if $L = H$ **then**
          continue
        end if
        Compute η, the second derivative of the objective function with respect to $\alpha[j]$:
        $\eta = 2 \cdot K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j)$ ▷ Using RBF kernel
        if $\eta \geq 0$ **then**
          continue
        end if
        Update $\alpha[j]$:
        $\alpha[j] = \alpha[j] - \frac{y[j] \cdot (E[i] - E[j])}{\eta}$
        Clip $\alpha[j]$ to be within $[L, H]$
        if $|\alpha[j] - \alpha_{old_j}| < 0.00001$ **then**
          continue
        end if
        Update $\alpha[i]$:
        $\alpha[i] = \alpha[i] + y[i] \cdot y[j] \cdot (\alpha_{old_j} - \alpha[j])$
        Compute thresholds $b1$ and $b2$:
        $b1 = b - E[i] - y[i] \cdot (\alpha[i] - \alpha_{old_i}) \cdot K(x_i, x_i) - y[j] \cdot (\alpha[j] - \alpha_{old_j}) \cdot K(x_i, x_j)$ ▷ Using RBF kernel
        $b2 = b - E[j] - y[i] \cdot (\alpha[i] - \alpha_{old_i}) \cdot K(x_i, x_j) - y[j] \cdot (\alpha[j] - \alpha_{old_j}) \cdot K(x_j, x_j)$ ▷ Using RBF kernel
        Update b:
        if $0 < \alpha[i] < C$ **then**
          $b = b1$
        else if $0 < \alpha[j] < C$ **then**
          $b = b2$
        else
          $b = \frac{b1 + b2}{2}$
        end if
        $num\_changed\_alphas = num\_changed\_alphas + 1$
      end if
    end for
    if $num\_changed\_alphas == 0$ **then**
      break
    end if
  end for
  return α, b
end procedure