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 program an instantiation of Bayesian Optimization, in particular Sequential Model-based Optimization (SMBO). SMBO is a powerful techique 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 two different scikit-learn algorithms and to find two different datasets to be used on the task. For algorithms we decided to implement 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. The algorithm targets to find the hyperplane that minimizes classification errors while maximizing the margin, with these support vectors playing a critical role. SVMs can handle both linearly separable and non-linearly separable data by utilizing different types of kernel functions. These kernels determine the similarity between data points and extend the applicability of SVM to a wide range of scenarios. In summary, SVMs are versatile classifiers known for their ability to handle complex data relationships.

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. This process iterates for a set number of times or until a stopping criterion is met. Each iteration introduces a new weak learner trained on the updated, weighted data. The final model is a combination of weak learners' predictions, with each learner's contribution weighted by its importance. Adaboost's strength is in its adaptive nature – it assigns greater emphasis to well-performing weak learners and less to those that perform poorly. By continually focusing on misclassified samples, Adaboost allows the ensemble to learn from its mistakes and improve overall performance. This iterative approach often results in a highly accurate final model well-suited for complex classification tasks.

3 Results

After we implemented the algorithms we tested them on two datasets: The first one is the blood-transfusion-service-center, which is a dataset created by the donor database of Blood Transfusion Service Center in Hsin-Chu City in Taiwan. It

includes 4 numeric features and 1 Class (target) which is 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 results were the following:

We have noticed that setting the grid for a GridSearch is very important to get the right 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 huge amount of run-time to fit. RandomizedSearch could be faster but as it goes by random chances of finding the exact best solution are very low. Bayesian Optimization algorithm in its approach solves that 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, for example, changing the range of the hyperparameters helps achieve better results, or increasing the number of initial configurations also helps the algorithm give better results (especially when we take a big interval of parameters). Sometimes the GridSearch or RandomSearch would give us better results than the Bayesian Optimization, but after some tuning, the algorithm would get at worst as good results as those algorithms. In conclusion, Bayesian Optimization gave us better or similar results with less run time.

We can see how the algorithm works in the graphs, the grid search 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 in a reasonable runtime are low. 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.

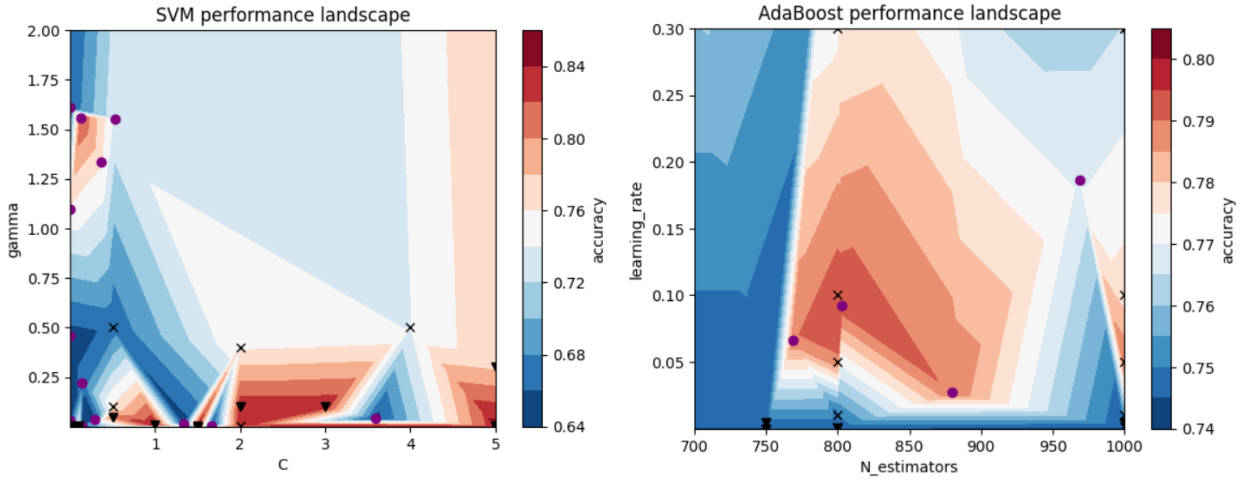

Figure 1: **Left:** Performance landscape of SVM on the QSAR biodegradation dataset (This section of the graph does not contain the best hyperparameters, but it clearly shows how the algorithm works). **Right:** Performance landscape of AdaBoost Classifier on the Blood Transfusion Service Center Data set

4 References

Scikit-learn, accessed October 11th 2023.

Dataset blood-transfusion-service, accessed October 11th 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