Contents

Project 10 - Description

(M) is a SVR-type approach of your choice (in particular, with one or more kernels of your choice).

(A1) is an algorithm of the class of level bundle methods, applied to either the primal or the dual formulation of the SVR.

(A2) is a general-purpose solver applied to an appropriate formulation of the problem.

Use of an off-the-shelf solver for the Master Problem of the bundle method is allowed.

Introduction

SVR and Level Bundle Method

This project focuses on the development and implementation of an SVR (Support Vector for Regression) capable of learning from a dataset in the form of "feature x target," where "target" must be a vector of dimensions `n x 1`, in accordance with the definition of SVR. In addition to the basic implementation, a significant part of the SVR will leverage the Level Bundle Method for optimizing the dual function.

For the Master Problem of the **SVR with LBM**, the MATLAB function `quadprog` was used. This function is primarily designed for solving quadratic objective functions with linear terms.

Instead, for the **general purpose SVR (Oracle)** we simply solved the dual function described in the following paragraphs, using a high efficiency solver.

Support Vector for Regression (SVR)

Support Vector Regression (SVR) aims to find a function $f(X)$ that approximates the training data while minimizing a given loss function. The primal optimization problem is formulated as follows:

$$\begin{aligned} \min_{w,b,\xi,t^*} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{s.t.:} \quad & \\ & y_i - \langle w, x_i\rangle - b \leq \varepsilon + \xi_i, \\ & \langle w, x_i\rangle + b - y_i \leq \varepsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0 \end{aligned}$$

where:

- w and b define the regression hyperplane
- ξ_i, ξ_i^* are slack variables that account for deviations beyond the margin ε
- C is a regularization parameter

By applying Lagrange multipliers and transforming the problem into its dual formulation, we obtain:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} y_i\alpha_i - \varepsilon\sum_{i=1}^{n}|\alpha_i| - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j K(x_i, x_j) \\ \text{subject to:} \quad & \\ & \sum_{i=1}^{n}\alpha_i = 0, \\ & -C \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n. \end{aligned}$$

where $K(x_i, x_j)$ is a kernel function that allows the method to handle non-linear relationships by implicitly mapping the input data to a higher-dimensional space.

Once the dual problem is solved, the **support vectors** are identified as the data points corresponding to nonzero Lagrange multipliers. Specifically, the support vectors are found by selecting the indices (i, j) that satisfy the condition:

$$\{(i, j) \mid |\alpha_i| > \tau\}$$

where τ is a small positive threshold to account for numerical precision. These support vectors are the most influential data points in defining the regression function, as they determine the final predictive model.

Overview of the Level Bundle Method

The Level Bundle Method (LBM) is an optimization approach that refines solutions iteratively by leveraging cutting-plane techniques and a level constraint. It is particularly useful in non-differentiable optimization problems, such as those encountered in support vector regression.

Problem Formulation

Since the SVR function defined before is **non-differentiable**, we apply the **Level Bundle Method** to approximate it iteratively. Especially we find a new solution by solving the LBM objective function defined as follow:

$$\alpha_{k+1} = \arg\min_{\alpha} \left\{ \frac{1}{2}\|\alpha - \hat{\alpha}_k\|^2 \,\middle|\, \hat{f}_k(\alpha) \leq f_k^{\text{level}},\ \alpha \in X \right\}$$

where:

- α represents the vector of dual variables in the **Support Vector Regression (SVR)** problem.
- $\hat{\alpha}_k$ is the best solution found so far at iteration k.
- f_k^{level} is the current level used to restrict the search space within an acceptable region.
- X is the set of original constraints of the SVR dual problem.
- $\hat{f}_k$ is defined as follow:

$$\hat{f}_k(x) := \max_{j \in \mathcal{B}_k} \{f(x_j) + \langle \xi_j, x - x_j \rangle\}$$

Where:

$$f(x) = \frac{1}{2}x^\top Kx + \varepsilon \sum_{i=1}^{n} |x_i| - y^\top x$$

$$\xi = Kx + \epsilon \cdot \text{sign}(x) - y$$

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0. \end{cases}$$

However, from MATLAB documentaion, the `quadprog` general function is designed to solve quadratic optimization problems in the following standard form:

$$\begin{aligned} \min_x \quad & \frac{1}{2}x^\top Hx + f^\top x \\ \text{s.t.} \quad & Ax \leq b, \\ & A_{\text{eq}}x = b_{\text{eq}}, \\ & lb \leq x \leq ub. \end{aligned}$$

Therefore, it is necessary to reformulate the **Level Bundle Method** problem so that it is compatible with the form required by `quadprog`.

Quadratic Norm Expansion

The objective function of the Level Bundle Method can be rewritten explicitly as follows:

$$\frac{1}{2}\|\alpha - \hat{\alpha}_k\|^2 = \frac{1}{2}\left(\alpha^\top\alpha - 2\hat{\alpha}_k^\top\alpha + \hat{\alpha}_k^\top\hat{\alpha}_k\right) = \frac{1}{2}\alpha^\top\alpha - \hat{\alpha}_k^\top\alpha + \frac{1}{2}\hat{\alpha}_k^\top\hat{\alpha}_k.$$

Since the constant term $\frac{1}{2}\hat{\alpha}_k^\top\hat{\alpha}_k$ does not affect the minimization, we can omit it. The function to minimize then becomes:

$$\frac{1}{2}\alpha^\top\alpha - \hat{\alpha}_k^\top\alpha.$$

Reformulation into `quadprog` format

By comparing this expression with the standard objective function solved by `quadprog`:

$$\frac{1}{2}x^\top Hx + f^\top x,$$

we obtain the following parameters:

- $H = I$ (identity matrix, since the quadratic term is $\frac{1}{2}x^\top x$).
- $f = -\hat{x}_k$ (since the linear term is $-\hat{\alpha}_k^\top\alpha$).

Substituting these values, we obtain:

$$\frac{1}{2}x^\top Hx + f^\top x = \frac{1}{2}x^\top x - \hat{x}_k^\top x.$$

That is exactly the function that we want to minimize.

Constraints

The Level Bundle Method algorithm is subject to the following constraints:

$$\begin{aligned} &\text{s.t. :} \\ &\quad \hat{f}_k(\alpha) \leq f_k^{\text{level}} \\ &\quad \sum_{i=1}^{n} \alpha_i = 0 \\ &\quad -C \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n \end{aligned}$$

Thus, they must be rewritten into a form solvable by `quadprog`, just as we did for the objective function.

Bundle Cuts Constraint

Given the following constraint:

$$\hat{f}_k(\alpha) \leq f_k^{\text{level}}$$

We can implement it through the cutting plane approach, where the function is approximated via a collection of tangent hyperplanes. These hyperplanes are defined by the subgradients of the function and translate into linear constraints of the form:

$$f(\alpha_k) + \langle \xi_k, \alpha - \alpha_k \rangle \leq f^{\text{level}}$$

Where ξ_j are the subgradients computed at those points and $f(\hat{\alpha_k})$ is the real objective value evaluated.

This formulation must be rewritten in the form $Ax \leq b$. Rewriting the constraint:

$$f(\alpha_k) + \langle \xi_k, \alpha - \alpha_k \rangle \leq f^{\text{level}} \iff \langle \xi_k, \alpha \rangle - f^{\text{level}} \leq \langle \xi_j, \alpha_k \rangle - f(\alpha_k)$$

With:

$$A = \begin{bmatrix} \xi_k^\top & -1 \end{bmatrix}, \quad b = \langle \xi_k, \hat{\alpha_k} \rangle - f(\hat{\alpha_k})$$

Lower Bound Estimation in the Level Bundle Method

In the classical Level Bundle Method, at each iteration a convex overestimator of the dual function is constructed using subgradients. This overestimator can then be used to compute a lower bound on the dual function. Such a lower bound is valuable for guiding the choice of the level parameter, helping the algorithm

converge more efficiently toward the optimal solution. Given the constraint function:

$$\hat{f}_k(\alpha) = \max_{i \in B_k} \{f(z_i) + \langle \xi_i, \alpha - z_i \rangle\}$$

This piecewise-linear and convex function overestimates the true dual objective function, and can be used to compute a global lower bound:

$$f_{\text{lower}} = \min_{\alpha \in X} \hat{f}_k(\alpha)$$

To compute this lower bound in practice, we solve the following convex optimization problem by introducing an auxiliary scalar variable t:

$$\begin{array}{ll} \min_{\alpha,t} & t \\ \text{s.t.} & f(z_i) + \langle \xi_i, \alpha - z_i \rangle \leq t, \quad \forall i \in B_k \\ & \sum_{i=1}^{n} \alpha_i = 0 \\ & -C \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n \end{array}$$

Then we can evaluate the level by using:

$$f^{\text{level}} = f_{\text{lower}} + \theta * (f_{\text{upper}} - f_{\text{lower}})$$

Where f_{upper} is the current best solution found at iteration k.

Can the lower bound diverge to infinity? No, because the dual objective function is convex and continuous, and the optimization is carried out over a feasible region defined by box constraints (which impose $-C \leq \alpha_i \leq C$) and the sum-to-zero constraint.

Equality Constraint

$$\sum_{i=1}^{n} \alpha_i = 0.$$

This can be defined directly as:

$$A_{\text{eq}} = \left[\mathbf{1}^\top\right], \quad b_{\text{eq}} = 0.$$

Bound Constraints

$$-C \leq \alpha_i \leq C,$$

These can be defined as:

$$lb = [-C, -\infty],$$
$$ub = [C, f_{\text{level}}].$$

Does the Level Bundle Method Converge?

The introduction of the absolute value in the objective function, while introducing non-differentiability, does not compromise the convexity of the function, since the absolute value is itself a convex function. As a result, the dual problem, when viewed as a minimization problem, remains convex: this ensures that **any local minimum found is also a global minimum**.

The expected efficiency of the method is sub-linear ($O(\frac{1}{k})$ with k = number of iterations), which is typical for non-smooth optimization algorithms.

Finally **the method converges** if and only if, in most iterations, the condition:

$$\hat{f}_k(\alpha_{k+1}) \geq f(\alpha_{\text{best}})$$

is satisfied cause this ensures that the solution progressively approaches to the true optimum. This is achieved by solving the quadratic problem (Master Problem) defined previously, that constructs a global over-estimator of the objective function.

Additionally, the careful selection of the f^{level} parameter prevents the algorithm from diverging beyond the optimal solution.

General Notes and Considerations

As previously described, this project aims to implement an SVR that leverages the Level Bundle Method for optimization. However, the goal is not to achieve a fully optimized SVR in terms of hyperparameters or generalization performance on the dataset cause this would require additional tuning techniques such as grid search or k-fold cross-validation, which are beyond the scope of this project.

Nevertheless, we will present the error achieved using the Mean Squared Error (MSE) metric, along with the selected hyperparameters. Additionally, we will compare our implementation against SVR general solver (A1), which will be used as an oracle.

Conclusion

We have demonstrated how the objective function of the **Level Bundle Method** can be rewritten in the standard form required by `quadprog`. In the following sections, we will present the implementation of the algorithm following the steps outlined so far.

Implementation

Dataset

Initially, we used simple functions such as sine, exponential, and step functions with added noise. These synthetic datasets allowed us to test and verify the correctness of our SVR implementation in terms of predictions and performance. Subsequently, we moved our evaluation to larger and more complex real-world datasets for thoroughly assess the true performance and robustness of our SVR model under challenging conditions.

SVR Dual Function implementation

Firstly, we define the SVR dual function, which computes both the function value `f` and the corresponding subgradient `g`:

```
function svr_dual_functon(x, y, epsilon, K):
Input:
    x           % input vector
    y           % target vector
    epsilon     % scalar value
    K           % kernel matrix

Output:
    f           % objective function value
    g           % gradient

f = 0.5 * TRANSPOSE(x) * (K * x) + epsilon * SUM(ABS(x)) - TRANSPOSE(y) * x
g = K * x + epsilon * SIGN(x) - y
```

Before training, the features are normalized.

(A2) SVR general-purpose solver

For the implementation of a generic SVR solver, we used the dual problem by solving it with a quadratic solver (MOSEK). In pseudo-code:

```
Constraints = [-C <= alpha <= C, sum(alpha) == 0]

Objective = svr_dual_functon(x, y, epsilon, K)

alpha = optimize(Objective, Constraints)
```

Once the execution is completed, we extract the **support vectors** from the solution.

```
% Identify indices of support vectors
% Select indices where the absolute value of alpha is greater than a threshold (tol)
sv_indices = indices where |alpha[i]| > tol

% Compute the bias term for prediction
bias = mean( Y[i] - sum over j of K[i][j] * alpha[j] ), for all i in sv_indices
```

Performance Evaluation

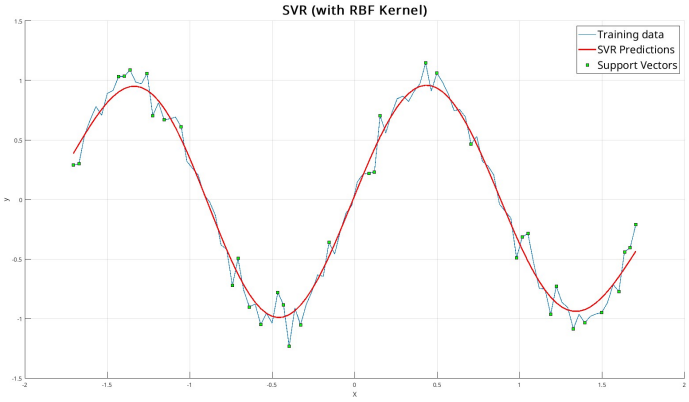
We tested our SVR model on both simple synthetic datasets (e.g., a noisy sine function) and large-scale real-world datasets. The synthetic data allowed us to verify the implementation's correctness, while the real data assessed its practical performance. The results demonstrate that while the solver generally works correctly, its speed significantly decreases when applied to large data volumes.

The following sections will showcase results on benchmark functions (such as noisy sine waves), which demonstrate the SVR implementation's correctness, while the high-dimensional real-world datasets will be addressed in the final chapter.

Synthetic data

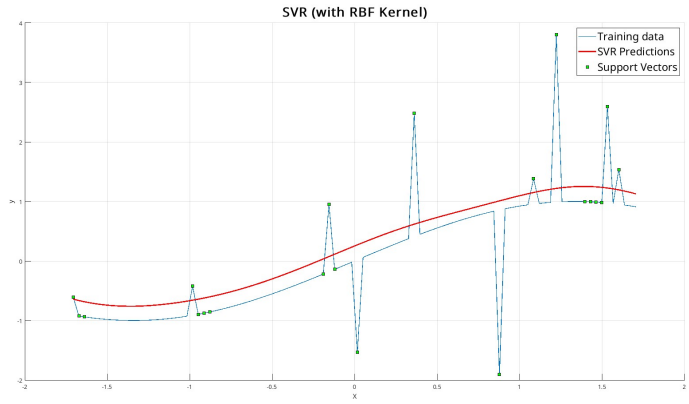
- **Sine function**

$$Y = \sin(3X) + 0.1 \cdot \mathcal{N}(0, 1).$$



- **Outliers test**

$$Y = \sin(X) + \underbrace{2 \cdot \mathcal{N}(0,1)}_{10 \text{ random points}} \quad .$$



(A1) Level Bundle Method Implementation

Previously, we mathematically formulated the objective function and constraints of the **Level Bundle Method (LBM)** in a format compatible with `quadprog`. We now translate these formulations into code, ensuring a direct correspondence between the mathematical expressions and their implementation.

Hessian Matrix and Linear Coefficients

The objective function of the optimization problem is defined as:

$$H = I, \quad f = -\hat{x}_k.$$

In pseudo-code, this is implemented as:

```
H = BLOCK_DIAG(IDENTITY(n), 0) % the extra 0 is beacuse we have f_level
f = CONCAT(-alpha_hat, 0)
```

Inequality Constraints

The linear constraints are represented in matrix form as:

$$A = \begin{bmatrix} \xi_k^\top & -1 \end{bmatrix}, \quad b = \langle \xi_k, \hat{\alpha_k} \rangle - f(\hat{\alpha_k})$$

```
A = CONCAT_COLUMNS(TRANSPOSE(bundle.g), -ONES(m, 1))
b = SUM_ROWS(bundle.g * bundle.alpha) - bundle.f
```

Equality Constraints

The equality constraints are given by:

$$A_{\text{eq}} = \begin{bmatrix} \mathbf{1}^\top \end{bmatrix}, \quad b_{\text{eq}} = 0.$$

```
Aeq = CONCAT(ONES(1, n), 0)
beq = 0
```

Variable Bounds

The variables are subject to the following bounds:

$$lb = [-C, -\infty],$$
$$ub = [C, f_{\text{level}}].$$

```
lb = CONCAT(-C * ONES(n, 1), -INF)
ub = CONCAT( C * ONES(n, 1),  f_level)
```

Solving the Optimization Problem

Finally, we use `quadprog` to find the optimal solution. So the ending function is:

```
function mp_solve(alpha_hat, bundle, f_level, C):
    Input:
        alpha_hat   % current estimate of the solution (vector)
        bundle      % set of past points, function values, and gradients
        f_level     % current objective level (level bundle threshold)
        C           % box constraint (‖alpha‖ <= C)

    Output:
        alpha_opt   % vector of alpha dual variables

    n = LENGTH(alpha_hat)
    m = LENGTH(bundle.f)

    H = BLOCK_DIAG(IDENTITY(n), 0)
    f = CONCAT(-alpha_hat, 0)

    A = CONCAT_COLUMNS(TRANSPOSE(bundle.g), -ONES(m, 1))
    b = SUM_ROWS(bundle.g * bundle.alpha) - bundle.f

    Aeq = CONCAT(ONES(1, n), 0)
    beq = 0

    lb = CONCAT(-C * ONES(n, 1), -INF)
    ub = CONCAT( C * ONES(n, 1),  f_level)

    sol = SOLVE_QP(H, f, A, b, Aeq, beq, lb, ub)

    if sol
        alpha_opt = sol[1:n]
    else
        WARN("Best solution not found, keeping previous...")
        alpha_opt = alpha_hat

    return alpha_opt
```

Lower bound estimation for Level

Given the following optimization problem

$$\begin{array}{ll} \min_{\alpha,t} & t \\ \text{s.t.} & f(z_i) + \langle \xi_i, \alpha - z_i \rangle \leq t, \quad \forall i \in B_k \\ & \sum_{i=1}^{n} \alpha_i = 0 \\ & -C \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n \end{array}$$

The corrisponding pseudo-code can be:

```
function compute_lower_bound(bundle, C)
    % Input:
    %   bundle: structure containing fields alpha, g, f
    %   C: bounding parameter for variables
    % Output:
    %   lb: computed lower bound

    [n, m] = SIZE(bundle.alpha)  % n = rows, m = columns

    % Build inequality constraints
    A = [TRANSPOSE(bundle.g), -ONES(m, 1)]
    b = SUM(bundle.g .* bundle.z, 1)' - TRANSPOSE(bundle.f)

    % Define linear program
    objective = [ZEROS(n, 1); 1]
    A_eq = [ONES(1, n), 0]
    b_eq = 0
    lower_bounds = [-C * ONES(n, 1); -inf]
    upper_bounds = [ C * ONES(n, 1);  inf]

    % Solve linear program
    [~, lb, status] = SOLVE_LP(objective, A, b, A_eq, b_eq, lower_bounds, upper_bounds)

    return lb
```

Then we can compute the level with:

```
level = lb + theta * (f_best - lb)
```

LBM Algorithm: Pseudo-code Implementation

```
Input:
    K           % kernel matrix
    alpha       % current solution (dual vector)
    C           % upper bound on dual variables
    theta       % level parameter
    tol         % convergence tolerance
    max_iter    % maximum number of iterations


% Initialization
f, g = svr_dual_functon(alpha, y, epsilon, K)
f_best = f

INIT_BUNDLE(bundle, alpha, f, g)

for iter = 1 to max_iter:
    % Compute the lower bound
    lb = compute_lowerbound(bundle, C)

    % Compute the acceptance level
    level = lb + theta * (f_best - lb)

    % Solve the master problem to get the new point
    alpha_new = mp_solve(alpha, bundle, level, C)

    % Evaluate the function and subgradient at alpha_new
    f_new, g_new = svr_dual_functon(alpha_new, y, epsilon, K)

    % Check if the new point is acceptable
    if f_new < f_best:
        f_best = f_new

    % Update bundle
    ADD_TO_BUNDLE(bundle, alpha_new, f_new, g_new)

    % Check for convergence (distance between upper bound and lower bound)
    if |(f_best - lb) / f_best| < tol:
        break

return alpha  % Return the optimal solution
```

Performance Evaluation

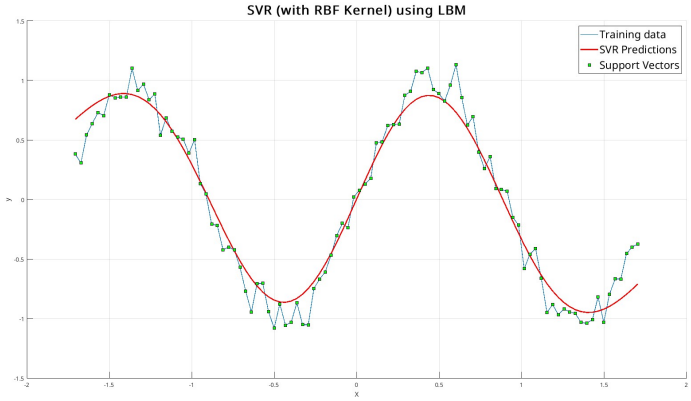
SVR Training with LBM on Synthetic Data

To evaluate the performance of our SVR (Support Vector Regression) model implemented with the Level Bundle Method (LBM), we conducted preliminary testing on predefined synthetic datasets. The results demonstrate that the model retains the generalization capability characteristic of classical SVR while maintaining comparable computational efficiency in terms of execution time.

Below, we present the predictive performance on the sine function as a representative example of the model's generalization capabilities. The remaining functions (omitted for brevity) demonstrate behaviors fully aligned with the standard SVR implementation.

- **Sine function**

$$Y = \sin(3X) + 0.1 \cdot \mathcal{N}(0, 1).$$



Training SVR with LBM on the Abalone Dataset

After validating the SVR's correctness, we evaluated its performance on complex real-world datasets like Abalone. This dataset contains 4177 samples with 8 features each, making it a perfect candidate as it introduces challenges such as:

- A high number of constraints
- Increased memory consumption
- Prolonged training time due to repeated calls to the solver

High Number of Constraints The main issue encountered in the implementation of the **Level bundle method** is the uncontrolled growth of the bundle size at each iteration.

Every cycle of the algorithm adds new constraints to the system, leading to an **exponential increase in the number of conditions** to be handled. This

progressive expansion of the bundle has two critical effects:

1. **Memory overload**: The bundle data structure, especially with high-dimensional datasets, consumes resources in a drastically non-linear manner, making allocation unsustainable for large data volumes.
2. **Performance degradation of quadprog**: MATLAB's quadratic programming solver becomes progressively slower—sometimes even leading to total stalls—due to the need to process constraint matrices with thousands of rows.

To address this issue, we introduced the **bundle truncation** technique: once a certain threshold is exceeded, the oldest constraints are removed from the bundle, leaving only the most recent k constraints. This significantly improved both memory efficiency and computational performance.

```
if LENGTH(bundle.alpha) > max_constraints
    REMOVE_FIRST_CONSTRAINT(bundle.alpha)
    REMOVE_FIRST_CONSTRAINT(bundle.f)
    REMOVE_FIRST_CONSTRAINT(bundle.g)
end
```

Memory Consumption Despite the introduction of bundle truncation, memory consumption remained significantly high. This was mainly due to the allocation of large matrices. For instance, given that the dataset contains 4177 features, the matrix H passed to `quadprog` has a size of 4178×4178, which is excessive for an identity matrix.

To mitigate this issue, we decided to switch to **sparse matrices**, which drastically reduced memory usage, as only nonzero elements are stored. Additionally, this optimization ensured that the `quadprog` solver could terminate within a reasonable time frame. So the new `mp_solve` function can be written as:

```
H = SPARSE(BLOCK_DIAG(IDENTITY(n), 0))
f = SPARSE(CONCAT(-alpha_hat, 0))

A = SPARSE(CONCAT_COLUMNS(TRANSPOSE(bundle.g), -ONES(m, 1)))
b = SPARSE(SUM_ROWS(bundle.g * bundle.alpha) - bundle.f)

Aeq = SPARSE(CONCAT(ONES(1, n), 0))
beq = 0

lb = SPARSE(CONCAT(-C * ONES(n, 1), -INF))
ub = SPARSE(CONCAT( C * ONES(n, 1),  f_level))
```

Choosing the Solver: Is Quadprog Really the Best Option? The existing implementation relies on **quadprog**, which suffers from slow training times (~50 seconds) even on simple objective functions due to its generic, non-optimized approach.

To address this inefficiency, we switched to **MOSEK**, a high-performance solver specialized for large-scale optimization. Leveraging advanced algorithms and hardware-aware optimizations, MOSEK achieves a ~**4x speedup** (reducing training time to ~12 seconds) while maintaining the same accuracy—making it ideal even for moderately complex problems.

So, in the end, for both the **Oracle** and our **SVR with Level Bundle Method (LBM)**, we use **MOSEK** to ensure consistent efficiency.

Conclusions: Achieved Results

This chapter presents a comparative analysis between an SVR using the Level Bundle Method (LBM) and a classical SVR implementation, referred to as the *Oracle*. Experiments were conducted across diverse datasets, with similar hyperparameters to ensure fairness. The goal is to evaluate under which conditions Level Bundle Method (LBM), achieves solution quality and computational performance comparable to the Oracle.

Oracle solution validity was verified using MOSEK logs, focusing on solution status (e.g., OPTIMAL, NEAR_OPTIMAL or UNKNOWN) and the duality gap (considered reliable if below 1e-4).

In general, we consider the comparison as a win if the relative gap, between Oracle and SVR with Level Bundle Method (LBM), is less than 1e-4.

Introduction

The dataset used are: Abalone, White Whine, Red Whine and Airfoil :

Dataset	Inputs	Features
Abalone	4,177	8
White Wine	4,898	11
Red Wine	1,599	11
Airfoil	1,503	5

The parameters for the Level Bundle Method used are:

- **tol**

  Tolerance for stopping criterion, indicating the distance between the lower and upper bound of the Level Bundle Method.

- **theta**

  Controls the step balance between the current lower bound and the best-known solution.

- **max_constraints**

  Maximum number of cutting planes (constraints) maintained in the bundle.

Each dataset is analyzed individually, reporting the parameters used for both the Oracle and Level Bundle Method (LBM), and presenting two key plots:

- **Relative Gap Plot**:
  - *Y-axis*: Relative gap between Oracle and LBM (logarithmic scale)
  - *X-axis*: Normalized number of iterations
  - *Purpose*: To assess how closely the LBM-based SVR approaches the Oracle's solution at each normalized iteration.

- **Convergence Plot**:
  - *Y-axis*: Objective values
  - *X-axis*: Time in seconds
  - *Purpose*: To evaluate how quickly the LBM-based SVR progresses toward the Oracle's solution over time (function evaluation at time t)

Abalone

Parameter	Value
Kernel	RBF(sigma=0.4)
C	1
Epsilon	1e-6

Oracle

Iterations	MSE	Min	Time (s)
14	4.2092	-6087.4994	27.4832

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -6.0874994035e+03   nrm: 1e+00    Viol.  con: 6e-09    var: 0e+00
  Dual.    obj: -6.0874994815e+03   nrm: 2e+01    Viol.  con: 0e+00    var: 8e-06
  Gap primal-dual: 1.2813e-08
```

SVR with Level Bundle Method (LBM)

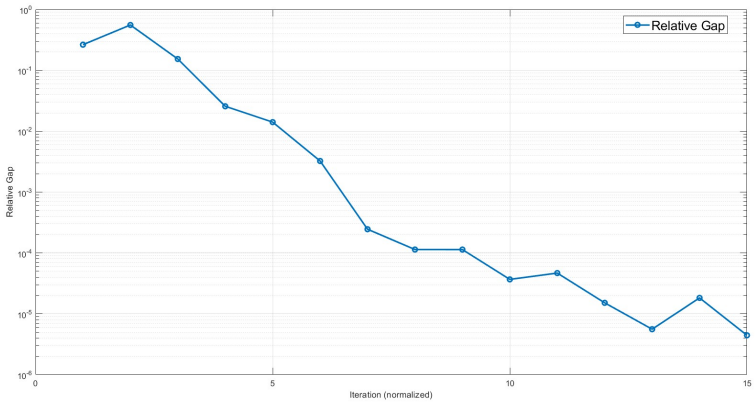
- **tol**: `2e-5`
- **theta**: `0.6`

- **max_constraints**: `100`

Iterations	MSE	Min	Time (s)
86	4.2122	-6087.4778	34.6167

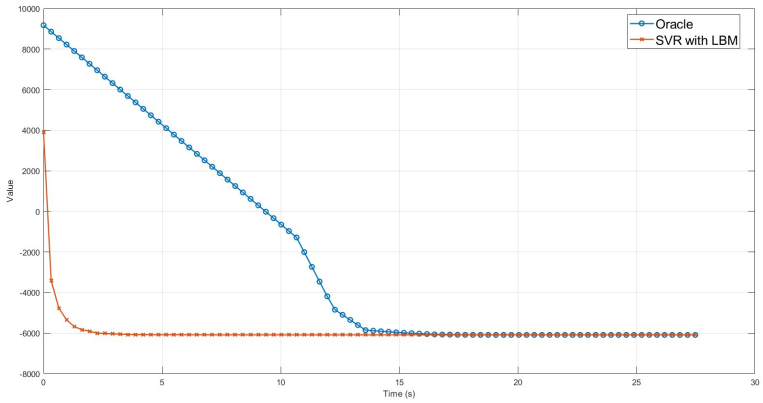
Relative Gap

Lower-Upper Bound	Oracle-LBM (Minimum)
1.973186e-05	3.5475e-06



The plot shows a clear decreasing trend in the relative gap between the Oracle and LBM methods across normalized iterations. Although LBM requires a significantly higher number of iterations (86) compared to the Oracle (14), the normalized view allows for a direct comparison of their progression.

The gap reduces steadily overall, with a noticeable drop in the middle and final stages. Interestingly, there is a brief increase in the gap before the final convergence, indicating a temporary divergence before the LBM method ultimately reaches the minimum.



Observing the performance over time, it is evident that the LBM method exhibits a much steeper initial decrease in the objective value compared to the Oracle. However, after reaching a plateau, it requires more iterations to converge fully to the minimum.

The Oracle, on the other hand, reaches the minimum slightly faster in terms

of total runtime (27.5 seconds versus 34.6 seconds for LBM). Nevertheless, the difference in runtime is relatively small, and the additional overhead of preserving bundle information appears to have limited impact on performance in this dataset, which is characterized by a large number of inputs and features.

White Wine

Parameter	Value
Kernel	RBF(sigma=0.6)
C	1
Epsilon	1e-6

Oracle

Iterations	MSE	Min	Time (s)
12	0.068958	-1257.6789	28.9418

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -1.2576789125e+03   nrm: 1e+00    Viol.  con: 1e-10    var: 0e+00
  Dual.    obj: -1.2576789122e+03   nrm: 6e+00    Viol.  con: 0e+00    var: 2e-07
  Gap primal-dual: 2.3853e-10
```

SVR with Level Bundle Method (LBM)

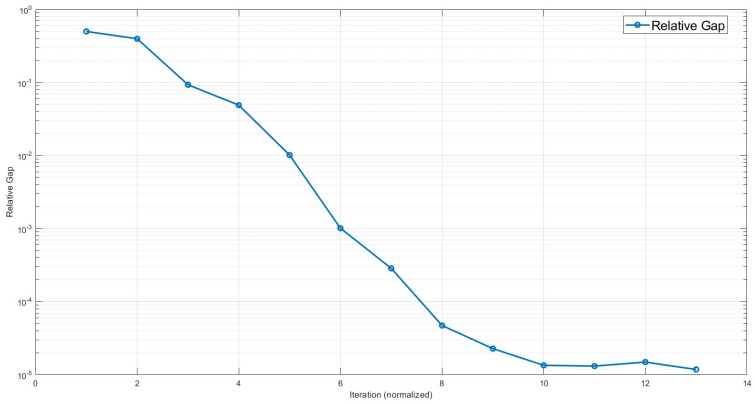
- **tol**: `2e-5`
- **theta**: `0.7`

- **max_constraints**: `100`

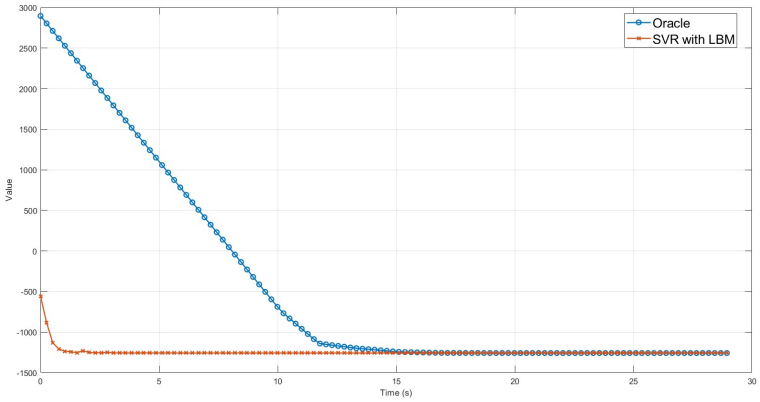
Iterations	MSE	Min	Time (s)
114	0.068975	-1257.6693	63.3961

Relative Gap

Lower-Upper Bound	Oracle-LBM (Minimum)
1.946468e-05	6.8581e-06



The convergence of both methods over the iterations shows that they reach a plateau in the final stages, indicating a more stable — though still overall decreasing — trend toward the minimum, with progressively smaller differences in the obtained results.



As observed in the Abalone dataset, the Oracle exhibits a less steep decrease in terms of both runtime and objective value. However, the difference between the two methods is significant, with LBM showing approximately a 50% increase in runtime — from 28.9 to 63.4 seconds.

Red Wine

Parameter	Value
Kernel	RBF(sigma=0.55)
C	1
Epsilon	1e-6

Oracle

Iterations	MSE	Min	Time (s)
12	0.056638	-380.0152	2.6541

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -3.8001520940e+02   nrm: 1e+00    Viol.  con: 6e-11    var: 0e+00
  Dual.    obj: -3.8001514184e+02   nrm: 6e+00    Viol.  con: 0e+00    var: 3e-05
  Gap primal-dual: 1.7778e-07
```

SVR with Level Bundle Method (LBM)

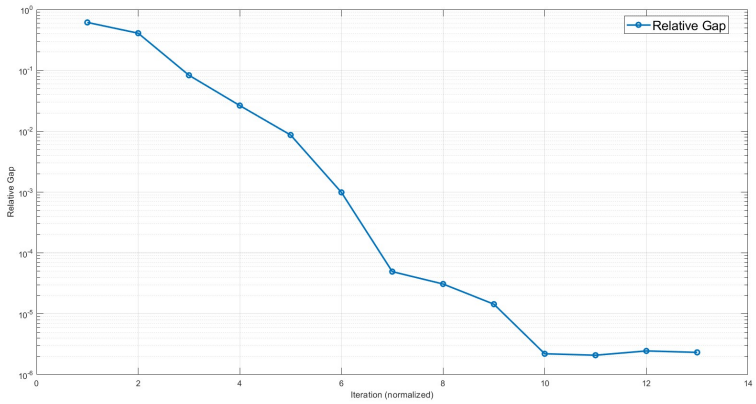
- **tol**: `1e-6`
- **theta**: `0.7`

- **max_constraints**: `100`

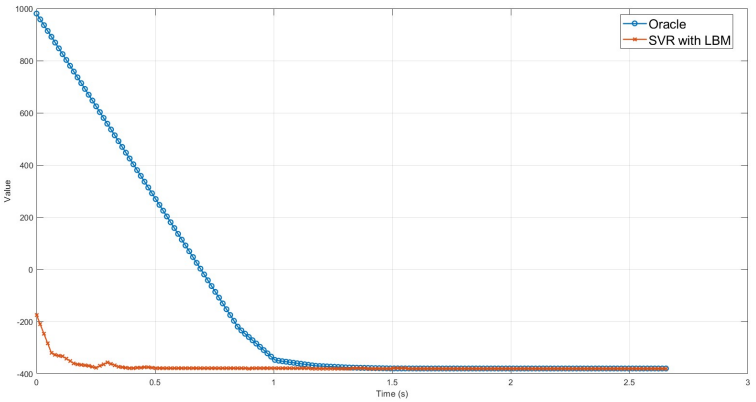
Iterations	MSE	Min	Time (s)
170	0.056647	-380.0144	61.7583

Relative Gap

Lower-Upper Bound	Oracle-LBM (Minimum)
9.820069e-07	2.1485e-06



Similar to the White Wine dataset, both methods reach a plateau in the final stages, reducing the relative gap between the minimum values achieved. Notably, in this case, the number of iterations is significantly higher: the Oracle converges within just 12 iterations, while the LBM requires 170 iterations to reach comparable results.



In terms of time-based performance, a significant difference is observed. Although LBM exhibits a faster initial phase (consistent with prior studies), its total computation time (61.8 seconds) is significantly higher compared to the Oracle (2.6 seconds). This discrepancy is attributed to LBM's larger number of iterations and the overhead of bundle storage.

Airfoil

Parameter	Value
Kernel	RBF(sigma=0.7)
C	1
Epsilon	1e-7

Oracle

Iterations	MSE	Min	Time (s)
13	10.24	-4635.7585	1.6597

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -4.6357585385e+03   nrm: 1e+00    Viol.  con: 1e-10    var: 1e-07
  Dual.    obj: -4.6357585894e+03   nrm: 1e+02    Viol.  con: 0e+00    var: 8e-06
  Gap primal-dual: 1.0980e-08
```

SVR with Level Bundle Method (LBM)

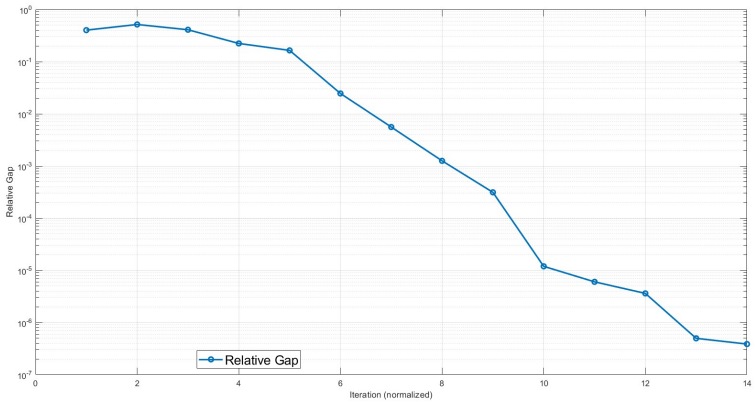
- **tol**: `1e-6`
- **theta**: `0.6`

- **max_constraints**: `100`

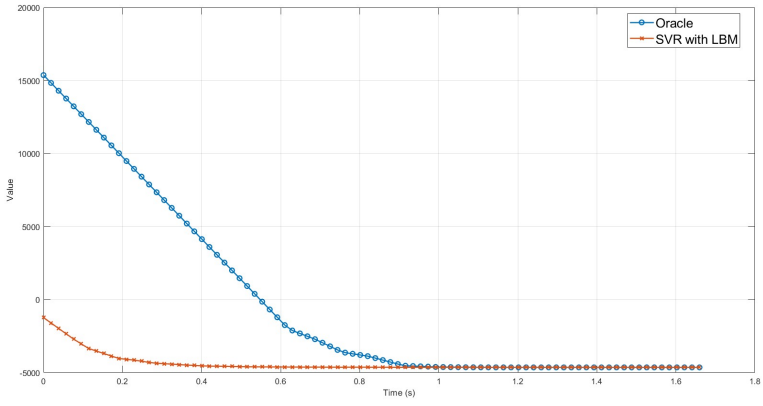
Iterations	MSE	Min	Time (s)
88	10.241	-4635.7580	14.6318

Relative Gap

Lower-Upper Bound	Oracle-LBM (Minimum)
9.674982e-07	1.1165e-07

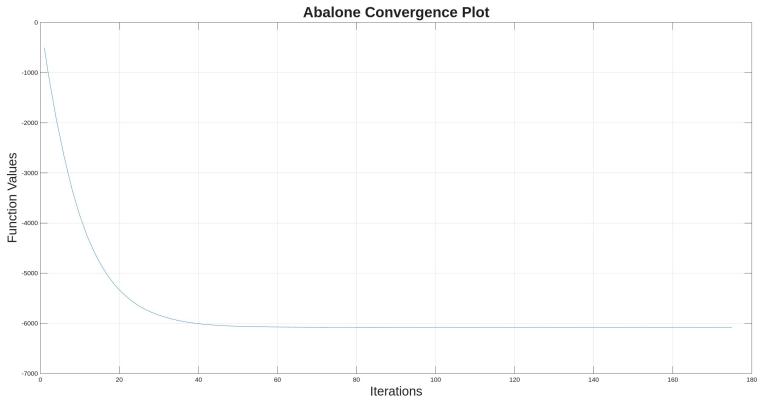


For the airfoil dataset, the relative gap plot highlights a more consistent descent, reaching a minimal gap between the models and yielding the best overall result compared to the other datasets in the final stages. The disparity in the number of iterations is still notable, with the oracle converging using only about 1/8 the iterations.



In terms of time, the overall behavior shown in this graph is consistent with previous findings; however, on this dataset, LBM exhibits a less steep initial decrease, which can be attributed to the algorithm taking larger steps in the early phase. The time difference is also less significant than for the Redwine dataset (which is the most similar case among those studied), with the oracle requiring 1.7 seconds compared to LBM, which takes only an additional 12 seconds.

Sublinear Convergence



As discussed in the theoretical section of the report, the algorithm is expected to exhibit sublinear convergence. This behavior is clearly reflected in the graph, where the gradual reduction in the gap over many iterations results in the characteristic curvature typical of sublinear convergence patterns.

Summary

Dataset	Model	Iterations	MSE	Min	Time (s)
Abalone	Oracle	14	4.2092	-6087.499	27.4832
Abalone	LBM	86	4.2122	-6087.477	34.6167
White Wine	Oracle	12	0.068958	-1257.678	28.9418
White Wine	LBM	114	0.06897	-1257.669	63.3961
Red Wine	Oracle	12	0.056638	-380.0152	2.6541
Red Wine	LBM	170	0.056647	-380.0144	61.7583
Airfoil	Oracle	13	10.24	-4635.758	1.6597
Airfoil	LBM	88	10.241	–4635.7580	14.6318

Relative Gap

Dataset	Lower-Upper Bound	Oracle-LBM
Abalone	1.973186e-05	3.5475e-06
White Wine	1.946468e-05	6.8581e-06
Red Wine	9.820069e-07	2.1485e-06
Airfoil	9.674982e-07	1.1165e-07

Across all datasets, the LBM-based SVR generally achieves comparable MSE and converges to the same minimum as the Oracle. However, computational time varies depending on dataset size and dimensionality.

- For **large, high-dimensional datasets** (e.g., *Abalone*, *White Wine*):
  - The time difference is small, as both methods require substantial computation.
  - LBM's overhead is less impactful, and convergence is similar.
- For **small or low-dimensional datasets** (e.g., *Airfoil*, *Red Wine*):
  - The Oracle is significantly faster (e.g., 1.7s vs. 14.7s for Airfoil, 2.6s vs. 61.8s for Red Wine).
  - LBM's overhead dominates, leading to notable inefficiency.

This shows that LBM is more competitive on complex problems, while its cost is harder to justify on simpler datasets.