Unconstrained Kernel-Based Optimisation with Reduced Support Vector Machines

Peter McMullan
Department of Computer Science
MSc Computational Finance
COMP0120: Numerical Optimisation
ucabpmc@ucl.ac.uk
Word count: 2927

April 22, 2024

1 The Underlying Optimisation Problem

1.1 a)

For the task at hand, a synthetic dataset was created using Scikit-Learn's `make_classification` function. 10,000 data points were created with 16 features and 2 classes. This was chosen as it creates a well-defined problem and is a balanced dataset (generates 5000 examples for each class), unlike other classification tasks which make it harder to guarantee convergence to an optimal solution. This dataset allows for clear application of Support Vector Machines (SVMs) given its non-separability. For the algorithm discussed in Section 2, this synthetic dataset is appropriate due to the large number of features and data points, with $m >> n$. The algorithm is used to estimate the weights and biases that draw an optimal hyperplane separating these two classes. The weights, biases, and hyperparameters were found via 10-fold cross validation.

1.2 b)

The features are not linearly separable, as seen in Figure 1. Therefore it will require a non-linear SVM to solve this classification task.

Kernel functions (the kernel 'trick') were used to find a solution to this problem. Such functions implicitly map the input data into a higher-dimensional feature space using dot products, therefore preserving pairwise relationships between data points. This allows the chosen SVM formulation to find non-linear decision boundaries in the original feature space by leveraging the linear separation properties in a higher-dimensional space. This kernel trick portrays an SVMs flexibility and computational efficiency by avoiding the explicit calculation of high-dimensional coordinates.

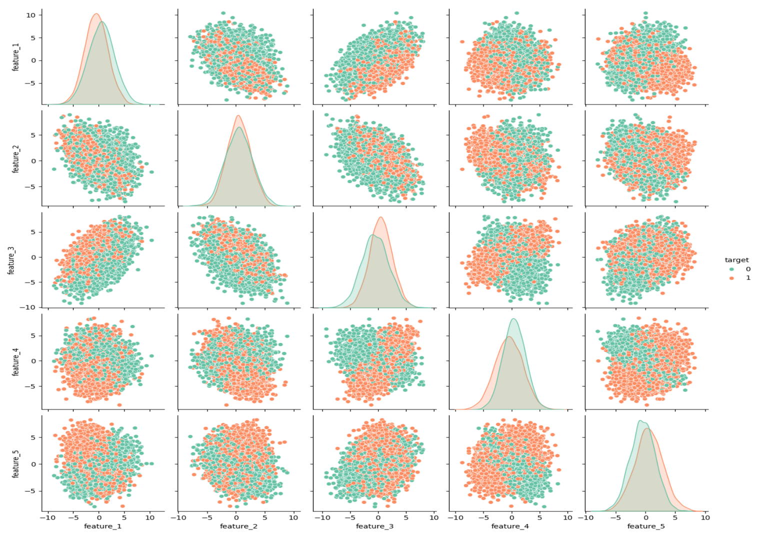


Figure 1: Visualisation of the synthetic data (first 5 features)

A **polynomial kernel** [10] was deployed in this paper given the relatively simplicity of the dataset (binary classification), and this also allows for better interpretability and generalisation (less chance of overfitting) versus other kernels. Polynomial kernels with low degrees generally require less memory and lower CPU than RBF and other kernels, so this helps in meeting the tasks specified objectives. This kernel is defined as;

$$K(x_i, x_j, b) = (x_i^T x_j + b)^d \tag{1}$$

where $K(\cdot)$ represents the polynomial kernel between vectors x_i and x_j, b is a constant bias term, and d is the degree of the polynomial ($d \geq 1$).

A non-standard loss function was developed, the margin-adaptive hinge loss function [2]. This is defined as:

$$L(y, f(x)) = \max(0, m(f(x)) - y \cdot f(x)) \tag{2}$$

where y is the binary classification label and $f(x)$ is the decision function of the SVM. The function $m(f(x))$ is a margin adaptation function defined as:

$$m(f(x)) = 1 + \beta \cdot \exp(-\delta \cdot |f(x)|) \tag{3}$$

In this context, β is a constant that controls the maximum additional margin that can be added for points classified with low confidence, and δ is a constant that controls how quickly the additional margin decreases as the classification confidence (i.e., $|f(x)|$) increases. Therefore, both hyperparameters control the margin's adaptation based on the confidence of the classification. From this expression it can be seen that as $\exp \to 0$, the loss function becomes standard hinge loss.

There are several reasons for incorporating this custom loss function. Firstly, the dataset features overlap between the two classes. A dynamic margin that expands for data points classified with lower confidence (near the boundaries) allows the model to be more cautious with these points, potentially improving class separation. Therefore, for points that the model classifies with high confidence (far from the decision boundary) the function reduces to the standard hinge loss, encouraging the model to maintain or even enhance its confidence level. By adjusting β and δ, the model's sensitivity to points near the decision boundary can be fine-tuned, offering a way to directly address the dataset's specific challenges without deviating too far from the proven effectiveness of the hinge loss. This allows for more flexibility versus the standard hinge loss.

Using the custom loss function for this classification, the soft-margin primal formulation is defined as;

$$\begin{aligned} &\underset{\mathbf{w},b,\xi}{\text{minimize}} && \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \\ &\text{subject to} && y_i(f(x)) - \beta\exp(-\delta|f(x)|) \geq 1 - \xi_i, \\ &\text{and} && \xi_i \geq 0, \\ & && \forall i = 1,\dots,m, \end{aligned} \tag{4}$$

where $\mathbf{w}$ is the weight vector perpendicular to the hyperplane, ξ_i are the slack variables for each data point, C is the regularization parameter controlling the trade-off between maximizing the margin width and minimizing the margin violations, $\mathbf{x}$ are the input features, and y_i are the class labels ([-1, 1] for binary classification) for ith data point.

The duel problem becomes;

$$\begin{aligned} &\max_{\boldsymbol{\lambda}}\sum_{i=1}^{m}\lambda_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \lambda_i \lambda_j K(\mathbf{x}_i,\mathbf{x}_j) + \sum_{i=1}^{m}\lambda_i\beta_i \cdot \exp(-\delta_i \cdot |f(\mathbf{x}_i)|) \\ &\quad - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j\beta_i\beta_j\delta_i\delta_j K(\mathbf{x}_i,\mathbf{x}_j)\exp(-\delta_i \cdot |f(\mathbf{x}_i)|)\cdot\exp(-\delta_j \cdot |f(\mathbf{x}_j)|) \\ &\text{subject to: } \lambda_i \geq 0, \sum_{i=1}^{m}\lambda_i y_i = 0,\ \forall i = 1,\dots,m, \end{aligned} \tag{5}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers and $K(\cdot)$ is the kernel function noted previously (assuming bias (b) $=$ 0). The above duel equates to the standard SVM duel [4] with two additional terms, which are similar but include the additional loss function terms. Defining the duel as such still includes both bounded and equality constraints and doesn't simply the function in a meaningful way. Therefore it is not used in this paper's implementation as convexity is not guaranteed. Therefore a different approach is taken, as outlined below.

1.3 c)

Two adjustments are made to the original primal outlined in Equation 4. First, the slack variable ξ is squared. The slack variable measures the degree to which each data point violates the margin. Squaring this term removes the non-negative constraint. Minimizing this shrinks the values towards zero, which corresponds to less violation and a "tighter" fit to the data that respects the margin more strictly. By using the $L2$ norm, this also encourages smoothness and convexity in the optimization problem and makes the function differentiable everywhere; This is critical for employing the Newton's method to guarantee convergence, which will be discussed in Section 2.1. The bias term, b, is also squared. This variable determines each boundary planes location relative to the origin and squaring this is done for the same reasons as with the slack variable outlined above. Therefore, the primal formulation becomes;

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{\text{minimize}} \quad & \frac{1}{2}(\mathbf{w}^T\mathbf{w} + b^2) + C\xi^T\xi \\ \text{subject to} \quad & y_i(f(x_i)) - \beta\exp(-\delta|f(x_i)|) \geq 1 - \xi_i, \qquad \forall i = 1, \dots, m, \end{aligned} \tag{6}$$

where a solution to this is often sighted for standard hinge loss using the max function (denoted $()_+$ here): $\xi_i = (1 - y_i(f(x_i)) + \beta\exp(-\delta|f(x_i)|)_+$. Therefore this can replace the formulation of the primal to become;

$$\underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}(\mathbf{w}^T\mathbf{w} + b^2) + C\|(1 - y(f(x)) + \beta\exp(-\delta|f(x)|))_+\|_2^2 \tag{7}$$

Moving from Equation 6 to Equation 7, ξ is replaced with its optimal value derived from the max function. This elimination makes the constraints implicit rather than explicit, making it an unconstrained optimization problem. The objective function now only involves $\mathbf{w}$ and b (along with the hyperparameters to be tuned), and the constraint is defined within the objective through the max operation. This is essential since the Newton method cannot deal with constraints.

1.4 d)

This is a quadratic programming problem with piece-wise linear components, meaning it is not a standard problem. The gram matrix, consisting of polynomial kernel evaluations between each pair of data points, should be at least positive semi-definite (p.s.d), which may also be an issue. Furthermore, using the custom loss function defined means the hyperparameters (β and δ) must be tuned, which adds to the complexity and computational time of the problem. Also, the optimisation problem must be convex and tractable given this loss function. In addition, using a polynomial fit should lead to less memory, unless the optimal solution is found at a high degree. The choice of degree (d=2) could lead to underfitting of the solution. Finally, the use of the max function means

Equation 7 is not twice differentiable (hinge loss is non-differentiable at x=1), meaning the Newton method cannot be applied. This also means that if a large amount of data is centered around zero, numerical stability may be an issue. Also, one well-sighted issue with this method is that not every initial point is feasible [1]. The outlined problems are overcome through the Reduced Support Vector Machine outlined in Section 2.

2 Optimisation Method and Convergence Theory

2.1 a)

The optimisation algorithm deployed was the Reduced Support Vector Machine (RSVM) [8]. Due to the high-dimensional nature of this data, with $m \gg n$ and 16 features, the RSVM algorithm helps to solve this using a randomly selected subset (1-10%) of the data to train the model. This aids in finding an optimal separating hyperplane while reducing computational time and memory requirements significantly, which is especially important for the large dataset used in this problem. RSVM exhibits strict convexity for any arbitrary kernel, including the polynomial one used in this paper. Due to the use of Newton's method, the RSVM algorithm has therefore proven to be globally quadratically convergent. However, to ensure the objective function is twice differentiable (in order to apply Newton), the max function must be modified (discussed below). One drawback of this algorithm is that it involves the inversion of the Hessian at each iteration, meaning computational efficiency can be compromised even when a data subset is taken. Memory efficiency is not a major issue as the Jacobian and Hessian can be deleted after each iteration.

2.2 b)

The max function in Equation 7 is replaced with;

$$p(x, \alpha) = x + \frac{1}{\alpha} \log(1 + e^{-\alpha x}), \quad \alpha > 0. \tag{8}$$

which is the integral of the sigmoid function, with x as the loss function defined previously. The solution to Equation 6 is obtained by solving problem 7 including $p(x, \alpha)$ with α approaching infinity. For all $\alpha > 0$, we have the following inequality:

$$\|x_\alpha - \hat{x}\|_2 \leq \frac{m}{2} \left(\frac{(\log 2)^2}{\alpha} + 2\rho \frac{\log 2}{\alpha} \right), \tag{9}$$

where $\rho = \max_{1 \leq i \leq m} |Ax - b_i|$. This provides an upper bound for the convergence of x_α to the optimal $\hat{x}$ as α tends to infinity (Proof: see [7]). This new formulation provides a smooth function, ensuring the loss function is twice differentiable so the Newton method can be applied. The objective function becomes;

$$\underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}(\mathbf{w}^T\mathbf{w} + b^2) + C\|p(1 - \bar{y}(f(\bar{x})) - \beta \exp(-\delta|f(\bar{x})|), \alpha)\|_2^2 \tag{10}$$

where the kernel function is implemented on $f(x)$, $\bar{x}$ replaces x, and $\bar{y}$ replaces y in Equation 7 to denote subsets of the data taken for RSVM.

The following pseudo-code outlines the steps taken in finding an optimal solution;

Algorithm 1 Newton–Armijo Algorithm for Reduced SVM

```
Choose $w_0$, where $\mathbf{w}$ is a vector of coefficients including $b$.
Stop if max iterations reached, if $||p_k|| \leq$ tol, or $\nabla\phi_\alpha(w_k) = 0$.
Else compute $w_{k+1}$ as follows:
    1. Newton direction: Determine direction $p_k$:
           $p_k = -\nabla^2\phi_\alpha(w_k)^{-1}\nabla\phi_\alpha(w_k).$        (11)
    2. Armijo step size: Use backtracking line search to choose a step size $\lambda_i$ such that;
           Sufficient-Decrease: $\phi_\alpha(w_k + \lambda_k p_k) \leq \phi_\alpha(w_k) + c_1\lambda_k \nabla f_k^T p_k$        (12)
       where $\phi_\alpha(\cdot)$ is the objective function value at the current iterate $w_k$, $p_k$ is the search direction,
       $\lambda_k$ is the step size, and $c_1 \in (0,1)$ is a parameter for the sufficient decrease condition.
    3. Step towards the optimal values: Update the current iterate;
           $\mathbf{w_{k+1}} = \mathbf{w_k} + p_k\lambda_k$        (13)
```

This algorithm is adapted from the Smooth SVM formulation [7] but for the RSVM a subset of the data is taken as noted previously.

Positive Definite Kernel

The kernel (K) is positive-definite if;

$$\sum_{i=1}^{n}\sum_{j=1}^{n} K(x_i, x_j)c_i c_j \geq 0, \quad \forall x_i, x_j \in \text{dom}(f) \text{ and } c_i, c_j \in \mathbb{R} \tag{14}$$

where x_i, x_j are data points. The best case comes where K (the polynomial kernel) is positive definite. Since it's a sum of products of the inputs and a non-negative constant raised to an even power, it satisfies Mercer's theorem [3] for kernel functions as it corresponds to an inner product in some feature space. For d=2, it's equivalent to the squared Euclidean inner product augmented by a constant bias $b > 0$, which keeps the kernel positive definite provided the original space is Euclidean.

The alternative case comes where K is positive semi-definite (p.s.d). If we drop the requirement that b is non-negative, the polynomial kernel can still be p.s.d because the quadratic form $\alpha^T K\alpha$ will be non-negative for any vector α. This is because $(x^T x)^2$ is always non-negative, and the cross terms that include b will be non-negative if they are dominated by the quadratic terms.

2.3 c) and d) - Convergence type and rate

The Newton method is expected to strongly converge provided a feasible point is chosen and the function is strictly convex (leading to global convergence). By construction, this optimisation formulation is convex and twice differentiable for any arbitrary kernel. Under the assumption that the Hessian is positive definite in the neighborhood $\mathcal{N}$ of the solution w^* and is Lipschitz continuous, the iterations $w_{k+1} = w_k + p_k\lambda_k$ converges quadratically (globally) for the Newton Method;

$$\|w_k + p_k - w^*\| \leq L\|\nabla^2 f(w^*)^{-1}\|\|w_k - w^*\|^2$$

Using this inequality inductively it can be concluded that if a feasible point is chosen, then the iterations converge to the solution, and the rate of convergence is quadratic (Proofs available in [7] [9]). It can be confirmed that such a result applies to this problem, since the objective function is strictly convex. A function f is strictly convex if for $x \neq y$;

$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha)f(y), \quad \forall \alpha \in (0,1). \tag{15}$$

Lipschitz continuity ensures the rate of change of the Hessian matrix is bounded by a constant L times the distance between points in the domain of the function. Further, a function whose Hessian matrix is positive definite at all points in its domain is said to be strongly convex. Under such conditions, the Newton step is guaranteed to be a descent direction, and the sequence of iterates will converge to the minimiser, provided a suitable step size is chosen at each iteration (dictated by the Armijo condition).

One problem with the Newton method is that if the Hessian is not positive definite then p_k may not be a descent direction. A function is convex if its Hessian matrix $\nabla^2 f(w)$ is positive definite for all $w \in \mathbb{R}^n$, implying that all eigenvalues of the Hessian are non-negative. Positive definiteness of $\nabla^2 f(w)$ implies that

$$\nabla f(w)^T p_k = -\nabla f(w)^T \nabla^2 f(w)^{-1} \nabla f(w) < 0$$

unless $\nabla f(w) = 0$, so the Newton step is a descent direction unless w is optimal [1]. We must therefore prove the stated optimisation formulation is convex to ensure the Newton direction can be applied and that it converges globally to the optimal solution. For the formulated problem, the loss function is standard hinge loss, which is well-sighted as a convex function [11], plus an exponential term, which is convex and non-decreasing. $p(x, \alpha)$ ensures smoothness and differentiability of this function. The convexity of this loss term is confirmed through plotting, as seen in Figure 2.

The term $\mathbf{w}^T\mathbf{w}$ equates to the sum of squares of the components of $\mathbf{w}$, which is a simple quadratic form with the identity matrix as its coefficient matrix. Because the identity matrix is positive

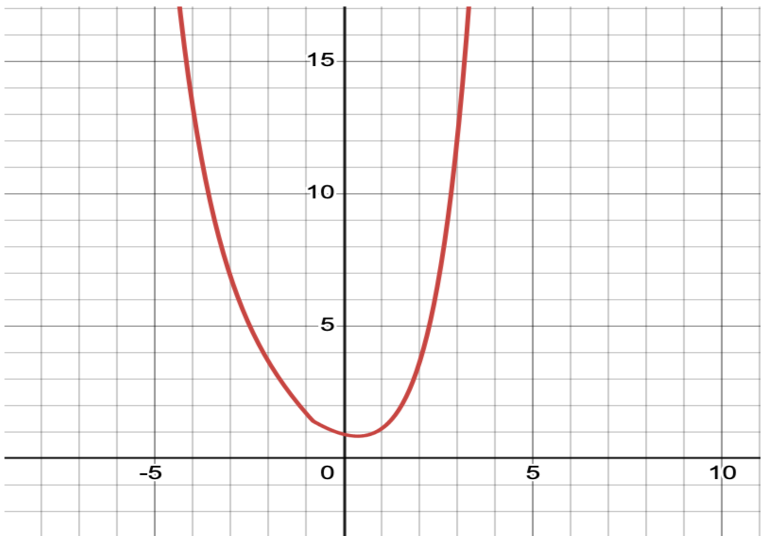


Figure 2: Loss Function for $\beta = 0.4, \delta = -0.8$

definite, the quadratic form $\mathbf{w}^T\mathbf{w}$ is convex. The addition of a b^2 constant does not change this. Finally, the squared-norm of a convex function is also convex [5] because the 2-norm is a convex function, and squaring is a non-decreasing operation over the domain of the norm (non-negative values). To show that the squared 2-norm is strictly convex, we need to show that Equation 15 holds. Expanding both sides with respect to $\|\mathbf{x}\|_2^2$ and $\|\mathbf{y}\|_2^2$ is trivial and we arrive at the inequality; $\alpha(1-\alpha)\|\mathbf{x}-\mathbf{y}\|_2^2 > 0$, which proves this is strictly convex given the reasons outlined. Since this function is strictly convex, the use of the Newton method means that any local minimum is the global minimum, and therefore we can say this algorithm is globally convergent, inline with the original RSVM paper[8].

3 Solution and discussion of the results:

3.1 a)

Using the described RSVM algorithm, an optimal hyperplane was found using a polynomial kernel with degree=2 where the solution comes in the form;

$$\mathbf{w} \cdot x + b = 0 \tag{16}$$

The problem was solved with the following parameters;

r = 0.1, v = 10, C = 1, β = 0.4, δ = -0.8, $\rho = 0.9$, $c_1 = 1e-4$,

where r is the ratio of data used (10%), v is the number of folds used during cross-fold validation, and ρ and c are the parameters used in backtracking line search methods. The max iterations was set to

1000 and the stopping tolerance was set to $1e^{-5}$. β, C and δ were found through cross-validation. Given the non-separability of the data and the 16-dimensions for which the separation is found, a representation of the separating hyperplane cannot be displayed. However, results of the classification task are outlined below and code used during the model building process is outlined in Section 4.

3.2 b)

Figure 3 plots the Exponential and Q-convergence for the algorithm to the optimal hyperplane outlined.

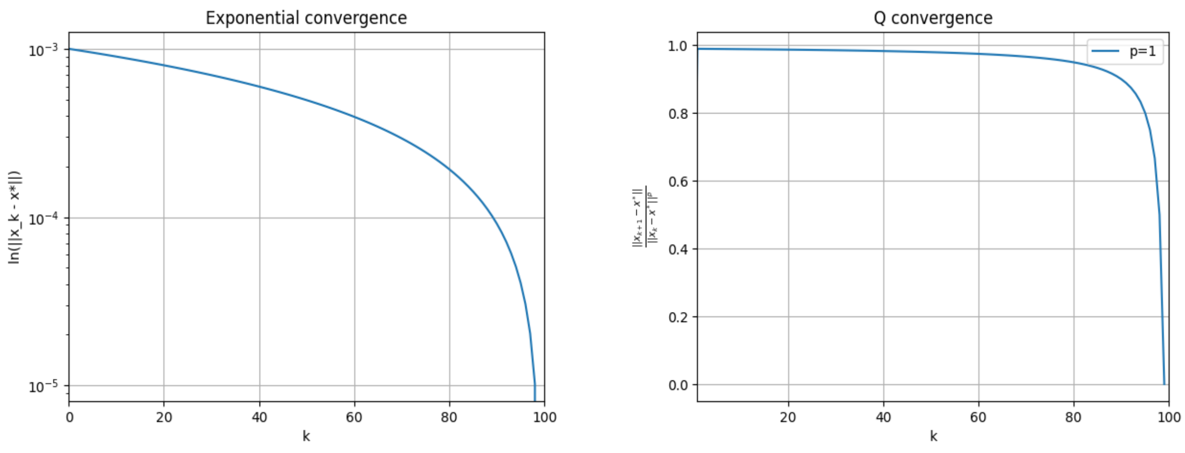


Figure 3: Convergence to optimal hyperplane for one fold

As outlined above, this algorithm was expected to converge quadratically. The Q-Convergence plot suggests linear convergence is present given that it falls below 1 towards 0 (for p=1). It can be seen that for both the Exponential and Q-convergence plots, they start very slowly, before quickly falling towards the optimal value in later iterations, as it finds a minima. From the exponential convergence plot, the steep drop early in the iterations and the sustained lower values suggest that the algorithm is quickly approaching the vicinity of the optimal solution. This convergence is faster than linear and therefore suggests that super-linear/quadratic convergence is present.

3.3 c)

The Hessian matrix must be positive definite to be invertible and converge quadratically via the Newton method. A positive definite Hessian indicates that the function being optimized is strictly convex near the current point, and therefore, a minimum can be found. A check was used during model running (see Section 4 code) to ensure the Hessian was correctly inverted during each iteration, and this was true throughout the model run. This confirmed that the formulation was continuously differentiable and nonsingular at the solution. Therefore, local convergence of this algorithm aligned with the theory. The convergence pattern of the algorithm also switches from a global convergence pattern (steady and linear) to a local pattern (acceleration near the solution). This is confirmed by Figure 3 above.

Since a backtracking line search algorithm was implemented, checking the step sizes is a suitable check. Steps of 1 should be accepted in the final iterations. If the algorithm is frequently reducing the step size significantly, this indicates that the search direction is not reliable, possibly because the chosen points are far from the solution. The implementation exhibited very small steps sizes, indicating a feasible point was not found. The gradients also decline strongly to begin, but increase slightly after the initial large decline (and the difference in the gradient norms is constant). This unfortunately suggests the algorithm does not converge and Lipschitz continuity is not confirmed (comparing gradients over iterations to check). Although it does not converge as expected, it may still produce good out-of-sample results (see Section 3.5) because it uses a subset of the data which significantly reduces the chance of overfitting.

3.4 d)

It took 1min 53secs for RSVM to perform 10-fold CV. For comparison, using the entire dataset (corresponding to the Smooth SVM algorithm) with 3-fold CV took 21min. Time complexity overall comes in $\mathcal{O}(N)$ (number of iterations per fold) and the matrix inversion comes with complexity of $\mathcal{O}(m^3)$. Overall the algorithm has complexity of $\mathcal{O}(mn + N\bar{m}^2)$ ($\bar{m}$ being the reduced dataset) [12]. Time grows almost linearly for the RSVM algorithm using increasing sizes of of data subsets, as seen in Figure 4. Although the Jacobian and Hessian needed to be calculated at each step, they could be deleted after each step meaning only one must be stored at a time. Therefore memory efficiency was not a concern (22.9mb per fold). The reduced dataset used in RSVM helped to reduce such memory demands.

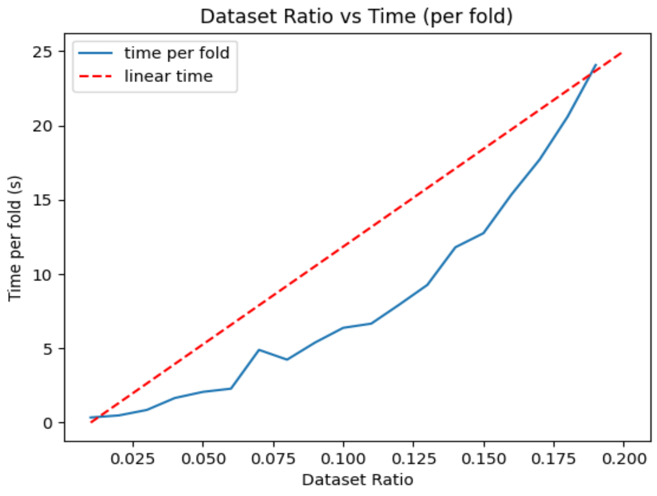


Figure 4: Runtime vs dataset ratio used

3.5 e)

The algorithm is successful in correctly identifying an optimal hyperplane for separating the data into its binary classification groups, with test set correctness of 95%, as measured by the mean absolute error: $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$. This is in comparison to the full dataset which showed a 96% accuracy. Python SVM packages exhibited similar results out-of-sample [6]. The confusion matrix, which conveys the number of correctly and incorrectly classified points for the two labels, is displayed in Figure 5. Overall, the RSVM method is clearly successful in finding an optimal hyperplane to separate this data. Further areas research could be to use a trust region method (with a modified Hessian to ensure positive definiteness) to form a quadratic model that is minimized.

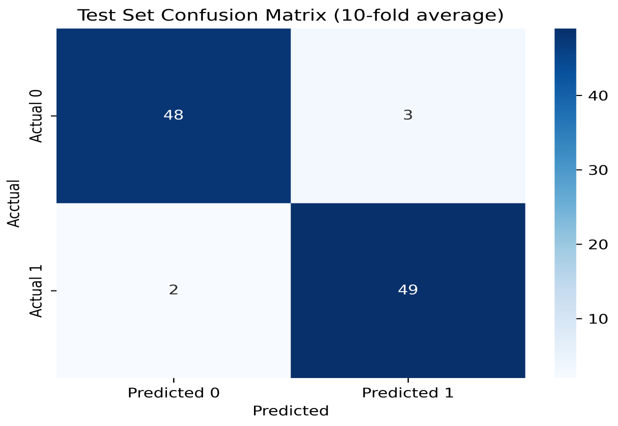
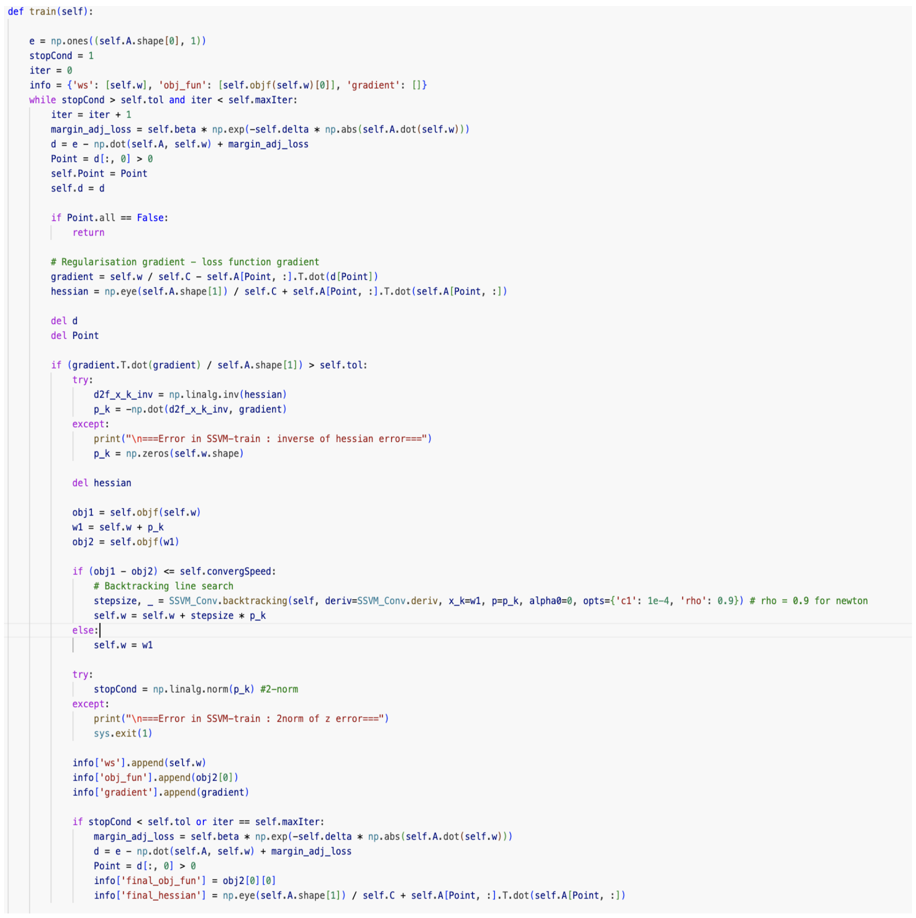


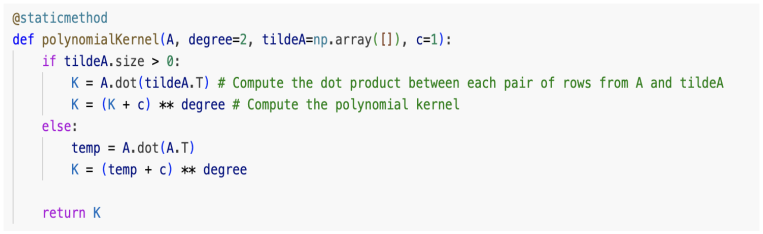
Figure 5: Average classification over 10-fold CV

4 Appendix



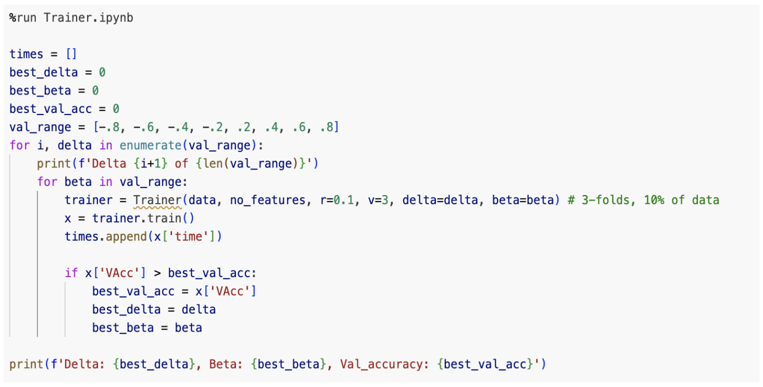
```python
def train(self):

    e = np.ones((self.A.shape[0], 1))
    stopCond = 1
    iter = 0
    info = {'ws': [self.w], 'obj_fun': [self.objf(self.w)[0]], 'gradient': []}
    while stopCond > self.tol and iter < self.maxIter:
        iter = iter + 1
        margin_adj_loss = self.beta * np.exp(-self.delta * np.abs(self.A.dot(self.w)))
        d = e - np.dot(self.A, self.w) + margin_adj_loss
        Point = d[:, 0] > 0
        self.Point = Point
        self.d = d

        if Point.all == False:
            return

        # Regularisation gradient - loss function gradient
        gradient = self.w / self.C - self.A[Point, :].T.dot(d[Point])
        hessian = np.eye(self.A.shape[1]) / self.C + self.A[Point, :].T.dot(self.A[Point, :])

        del d
        del Point

        if (gradient.T.dot(gradient) / self.A.shape[1]) > self.tol:
            try:
                d2f_x_k_inv = np.linalg.inv(hessian)
                p_k = -np.dot(d2f_x_k_inv, gradient)
            except:
                print("\n===Error in SSVM-train : inverse of hessian error===")
                p_k = np.zeros(self.w.shape)

            del hessian

            obj1 = self.objf(self.w)
            w1 = self.w + p_k
            obj2 = self.objf(w1)

            if (obj1 - obj2) <= self.convergSpeed:
                # Backtracking line search
                stepsize, _ = SSVM_Conv.backtracking(self, deriv=SSVM_Conv.deriv, x_k=w1, p=p_k, alpha0=0, opts={'c1': 1e-4, 'rho': 0.9}) # rho = 0.9 for newton
                self.w = self.w + stepsize * p_k
            else:
                self.w = w1

            try:
                stopCond = np.linalg.norm(p_k) #2-norm
            except:
                print("\n===Error in SSVM-train : 2norm of z error===")
                sys.exit(1)

            info['ws'].append(self.w)
            info['obj_fun'].append(obj2[0])
            info['gradient'].append(gradient)

            if stopCond < self.tol or iter == self.maxIter:
                margin_adj_loss = self.beta * np.exp(-self.delta * np.abs(self.A.dot(self.w)))
                d = e - np.dot(self.A, self.w) + margin_adj_loss
                Point = d[:, 0] > 0
                info['final_obj_fun'] = obj2[0][0]
                info['final_hessian'] = np.eye(self.A.shape[1]) / self.C + self.A[Point, :].T.dot(self.A[Point, :])
```

Figure 6: Optimisation algorithm



```python
@staticmethod
def polynomialKernel(A, degree=2, tildeA=np.array([]), c=1):
    if tildeA.size > 0:
        K = A.dot(tildeA.T) # Compute the dot product between each pair of rows from A and tildeA
        K = (K + c) ** degree # Compute the polynomial kernel
    else:
        temp = A.dot(A.T)
        K = (temp + c) ** degree

    return K
```

Figure 7: Polynomal Kernel, with TildaA representing the subset of data for RSVM



```
%run Trainer.ipynb

times = []
best_delta = 0
best_beta = 0
best_val_acc = 0
val_range = [-.8, -.6, -.4, -.2, .2, .4, .6, .8]
for i, delta in enumerate(val_range):
    print(f'Delta {i+1} of {len(val_range)}')
    for beta in val_range:
        trainer = Trainer(data, no_features, r=0.1, v=3, delta=delta, beta=beta) # 3-folds, 10% of data
        x = trainer.train()
        times.append(x['time'])

        if x['VAcc'] > best_val_acc:
            best_val_acc = x['VAcc']
            best_delta = delta
            best_beta = beta

print(f'Delta: {best_delta}, Beta: {best_beta}, Val_accuracy: {best_val_acc}')
```

Figure 8: Function call (with hyperparameter tuning via Grid-Search)

References

[1] Boyd and Vandenberghe. Convex optimisation. *Cambridge University Press*, 2009.

[2] Ali Karami-Mollaee Davood Zabihzadeh, Amar Tuama and Seyed Jalaleddin Mousavirad1. Low-rank robust online distance/similarity learning based on the rescaled hinge loss. *Springer Nature*, 2022.

[3] Ghojogh et. al. Reproducing kernel hilbert space, mercer's theorem, eigenfunctions, nystro m method, and use of kernels in machine learning: Tutorial and survey. *arXiv*, 2021.

[4] Hastie et al. Elements of statistical learning. *Springer*, 2009.

[5] Ryan Tibshirani et. al. Convex optimization. *CMU*, 2018.

[6] Aurelien Geron. Hands-on machine learning with scikit-learn, keras, and tensorflow. *O'Reilly*, 2023.

[7] Yuh-Jye Lee and Olvi Mangasarian. Ssvm: A smooth support vector machine for classification. *Springer*, 2000.

[8] Yuh-Jye Lee and Olvi Mangasarian. Rsvm: Reduced support vector machines. *SIAM*, 2001.

[9] Nocedal and Wright. Numerical optimisation. *Springer*, 1999.

[10] Vinge and McKelvey. Understanding support vector machines with polynomial kernels. *ResearchGate*, 2019.

[11] Mammadov Zhao and Yearwood. From convex to nonconvex: a loss function analysis for binary classification. *Core*, 2010.

[12] Shenglong Zhou. Sparse svm for sufficient data reduction. *arXiv*, 2021.