IMPLEMENTATION AND STUDY OF THE SUPPORT VECTOR MACHINE

Luu Minh Thong Tran
Student ID: 14522561
Undergraduate
luu.m.tran@student.uts.edu.au

1 Introduction

Support Vector Machine (SVM) is a supervised machine learning model founded in 1964 (Chervonenkis, 2013). The goal of the model is to find a hyperplane that maximizes the margin while minimizing the loss. According to Cover's theorem, the higher the dimension, the higher the chance that the samples are linearly separated. This is one of the fundamental theories of Support Vector Machine as it uses a trick called the kernel trick to learn the hyperplane at a higher dimensional space than the input dimensional space. However, during the training and testing phase, each data point is not necessarily to be directly mapped but rather to be dotted product with other data points. This implies that a kernel matrix is computed before the training process using a defined kernel function, and the kernel product would be reused to save computing power. With the kernel trick, SVM can learn non-linear separation with a less chance of overfitting as it also lowers the generalization error.

With the ability to learn non-linear separation and low generalization error, SVM used to be one of the most used supervised machine learning models before deep learning is popularized. One of its applications is in natural language, as was used in text categorization (Joachims, 1998). Nowadays, natural language processing is done using deep learning which requires a huge amount of computing power. This demonstrates the power of SVM back in the early stages of AI.

This journal would discuss the foundation of the Support Vector Machine including the loss, margin, objective functions, hypothesis spaces, kernel trick, and Sequential Minimizing Optimization algorithm (SMO). Then, the report would implement the SVM into Python code from scratch. Finally, it would be tested on a toy dataset to demonstrate the model.

Motivation

Because SVM is complicated, I am curious about the behavior of the model and motivated to investigate further. I found it very challenging to study the model but because of that, it motivated my curiosity to dive deep into the model and understand it intuitively. Therefore, in this journal, I will explain SVM through my theoretical intuition.

2 Support Vector Machine components theory

Setup

This section provide a brief introduction of learning task of a linear supervised classification machine learning model.

For a linear model, the task is to find a relationship that differentiate 2 or more classes given the input samples and the target class. To do it, it's objective is to learn a hyperplane with parameter w and b where the predicted class will be determined as

$$f(x) = w^T x + b$$

The parameter w controls the rotation of the plane and the bias (b) controls the elevation of the plane. The goal is to learn the optimized w and b that the loss function or the criterion is minimized.

2.1 Margin

The primary objective supervised classification machine learning model is estimating the input class by learning the relationship between the input and their class. However, when the model sees new unseen samples, different hyperplane with the same training accuracy might give a different test accuracy. This is the generalization ability of the model and to improve the generalization ability, it is assumed that the hyperplane that is the furthest from both classes would maximize the generalization. The distance from the hyperplane to the closest samples is called the margin of the classifier. Because the vector from the hyperplane to the closest sample is parallel with w, hence the margin is the norm of a scaled w. The margin hence defined as:

$$\hat{\gamma} = \min_i \frac{y_i \left(w^T x_i + b\right)}{\|w\|}$$

Now if w and b are scaled by the factor c, the hyperplane still produce the same classification. This is because scaling does not affect the angle rotated by the hyperplane. Hence by setting c to make the numerator to be 1, the margin becomes:

$$\hat{\gamma} = \frac{1}{\|w\|}$$

The objective is to maximize the margin which is the same of minimizing the reciprocal of the margin. This is well-known for the name L1- regularization. However, Support Vector Machine used L2- regularization which similar to L1-regularization but squared as maximize squared margin would also maximize the margin (convenient for differentiation by squared and halve it). In addition to minimizing the L2-regularizer, the model also needs to correctly classifies other points. Therefore, the full objective function of the margin becomes:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1, \quad \forall i.$$

2.2 Hinge Loss

The idea behind the Hinge Loss is that it will ignore the loss of any correctly classified data points if it lies outside the margin. To fully explain the hinge loss, start by considering the Logistic loss for a single sample:

$$l(y, \hat{y}) = -\left(y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})\right)$$

where y is the label $y \in \{0, 1\}$ and $\hat{y}$ is the predicted probability.To investigate the behavior of this loss, consider 2 scenarios when the hyperplane correctly classifies the sample and incorrectly classifies the sample.

Scenario	Behaviour
Incorrectly classify and far away from the hyperplane	Loss approaches infinity as the point is further away.
Correctly classify and far away from hyperplane	Loss approaches 0 as the point is further away from the hyperplane.

Because any correctly classified samples contribute a loss that penalize the model although the loss is small which approaches 0. Therefore, by ignoring the loss of samples that are outside of the margin, the model would be focused only the samples that are lies inside the margin and points that are incorrectly classified. Consider a perceptron loss:

$$l(y, w, b) = \max\left(0, -y\left(w^T x + b\right)\right)$$

The perceptron does a similar function with the Hinge Loss as it ignores any correctly classified samples and only penalized the model on the incorrectly classified samples. However, Support Vector Machine also maximizes the margin which implies that the loss needs to penalize the model on the points inside the margin of 1. Therefore, by adding 1 to the second term in max function, the model would be penalized by the points that are correctly classified and lies within the margin, and the Hinge Loss is derived. The Hinge Loss for a single sample is:

$$l(y, w, b) = \max\left(0,\ 1 - y\left(w^T x + b\right)\right)$$

Following that, the average Hinge Loss for n samples is:

$$L(w, b) = \frac{1}{n}\sum_{i=1}^{n} \max\left(0,\ 1 - y_i\left(w^T x_i + b\right)\right)$$

2.3 Primal and dual problem

Primal problem is the initial optimization where the objective is to minimize the cost function respect to a constraint. Dual problem on the other hand is a maximizing problem of a transposed cost function where the variable of the dual problem is the constraint in primal problem. Support vector machine can solely optimize the cost function using the primal form. However, using dual form would benefit computation power and in case of SVM, using dual form can lead to the use of kernel trick which reduces significantly the time complexity.

2.4 Primal (Soft-margin) Support Vector Machine

By minimizing the sum of the regularization term and the loss function, it will simultaneously maximize the margin while accurately classify the samples. This gives the primal form of Support Vector Machine without constraint:

$$\min_{w,b}\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max\left(0,\ 1 - y_i\left(w^T x_i + b\right)\right)$$

The objective function consists of 2 terms: regularization term and the Hinge loss which has been discussed. The goal for this objective function is to minimize the hinge loss and the regularization term. However, if minimizing only the sum of 2 terms would make the model values both terms at the same significance. Therefore, a parameter C is introduced to adjust the contribution of the loss. If C is large, the objective function will penalize the model more on misclassifications (Weinberger, 2018). Otherwise, if C is small, the objective function will value the loss less which make the margin more contributed. During training, the parameter C is usually determined by cross-validation for the best value of C. Using this objective function, SVM can optimize without any problem. However, by transforming the objective function to produce a set constraint, it can be later manipulated into the dual form. The un-constrained objective function can be re-written as:

$$\min_{w,b,\xi}\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Where ξ_i and the constraint $y_i(w^T x_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0$ is equivalent to $\max\left(0, 1 - y_i(w^T x_i + b)\right)$. By introducing these constraints, the problem can be transformed into its dual form to utilize the kernel trick.

2.5 Dual form of Support Vector Machine

Before transforming from primal to dual form, it is crucial to view the weights as the contributions of all the input samples. Suppose the input sample X has the dimension of [n x m] where n is number of samples and m is number of features. The weight (w) on the other hand has the dimension of [m x 1] which m is number of features. This means that by letting $w = X^T\alpha$, α need to have the dimension of [n x 1]. This implies that α is a set of weights each sample give to contribute to w. The view of the weight is a contribution of the samples also seen from other model such as linear regression as it has the close-form solution of $w = (X^T X)^{-1} X^T y$ for the weight. Back to SVM, the weight became a weighted sum of all data sample which instead of optimizing the weight, the new goal is to optimize $\alpha_i \forall i$ Interestingly,there is a characteristic that every correctly classified sample and outside margin will have a weight of 0 which means that during inference, only support vector matters.

By substituting $w = \sum_{i=1}^{m} \alpha_i y_i x_i$ into the Lagrangian of the constrained soft-margin primal problem and taking derivatives with respect to w, b, and ξ, we obtain the dual formulation:

$$\max_{\alpha}\ \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j\, x_i^T x_j \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C,\ \sum_{i=1}^{m} \alpha_i y_i = 0$$

By transforming to dual problem, parameter C became the constraint, and the objective is to maximize the problem. Interestingly, the new objective function depends on the term $x_i^T x_j$which will lead to the kernel trick. Another interesting feature is that the dual is a concave quadratic maximization with linear constraints hence there is a global optimum and SMO solves it exactly via QP subproblems. This will be point out in the Sequential Minimal Optimization which is the algorithm to optimize α.

2.6 Kernel Trick

When developing supervised linear family machine learning model, there is a crucial assumption made that is the hypothesis space is a set of linear hyperplanes. However, true function may not be linear and might be more complex than a linear hyperplane. Therefore, by considering other hypotheses space, it can capture more complex relationships though it not necessarily better. However, if the linear is a subset of the new hypotheses space, the empirical risk will never worse than ground error of model using linear hypotheses space. Therefore, to allow the model to learn more complex relationships, it is necessary to making the hypothesis space richer. To do this, input X is mapped from a low dimensionality to a high dimensional space. According to Cover's theorem, the higher the dimensionality, there is a higher chance of separability. This transformation is non-linear hence it will make the each hyperplane in the new dimensional space will correspond to a non-linear plane the original dimensional space. The mapping is defined as:

$$x_i \mapsto \phi(x_i)$$

Where $\phi(x_i)$ lives in a higher dimension than x_i . However, considering the complexity of the computation, it is extremely expensive. Consider the objective function:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j \, x_i^T x_j \quad \text{s.t.} \quad 0 \le \alpha_i \le C, \; \sum_{i=1}^{m} \alpha_i y_i = 0$$

The function takes the sum of all dot product of 2 data sample each iteration. Therefore, the time complexity will be $O(n * 2^d)$ where d is the dimension and n are number of samples. However, the dot product of 2 data sample does not change throughout the training process. Therefore, it can be stored in a matrix K where $K_{ij} = x_i^T x_j$. As $x_i \mapsto \phi(x_i)$, the matrix K is now transformed into $K_{ij} = \phi(x_i)^T \phi(x_j)$. It can be observed that K_{ij} is now a function x_i and x_j. Therefore, instead of defining $\phi(x)$, K_{ij} can be directly defined by a specific kernel function where:

$$K_{ij} = K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$$

This would reduce the time complexity down to $O(n^2)$ which is big improvement. There are many pre-defined kernel functions such as the polynomial kernel, RBF kernel or linear kernel where RBF is a universal approximator where it maps the input to an infinitely high dimension and it behave like a K-Nearest Neighbour model when looking at the decision boundary. Each kernel functions capture a particular relationship for example linear kernel is a linear hyperplane or polynomial captures a wavy relationship like a polynomial function. With the kernel trick, it made Support Vector Machine make use of the kernel method the most though other model can apply kernel method with out the trick. Furthermore, kernel method makes Support Vector Machine hypothesis space richer and flexible hence the true hypothesis space rely heavily on the kernel it uses.

2.7 Simplified Platt Sequential Minimal Optimization

The final component of SVM is its optimization algorithm. In other models such as logistic regression, artificial neural networks, they use gradient descent as the optimization algorithm which might guarantee convergence only if the hyperparameter which is the learning rate is carefully set. In SVM however, it uses the Sequential Minimal Optimization (SMO) algorithm to optimize the objective function. Unlike gradient descent, SMO guarantees convergence without hyperparameter tuning. This is due to the characteristics of the problem when it is concave in QP and followed a quadratic form (Platt, 1999).

Inspecting the constraint $\sum_{i=1}^{m} \alpha_i y_i = 0$, this means that if one of the α is update, one or more α needs to modify to respect the constraint. For example, if $\alpha_1 = 1, y_1 = 1, \alpha_2 = -1, y_2 = 1$ this will give the sum of 0. If α_1 is updated to be 0, then α_2 will also be updated to the 0 to respect the constraint. Note that if y_1 and y_2 have the same sign, if α_1 changes for a certain value, α_2 will changes for the corresponding value but in the opposite sign. In SMO, it picks 2 parameters α_i and α_j that is not optimized and optimize them in the way that they still respect the constraint as above (Page, n.d.). Aside from the constraint above, there is another constraint which is $0 \le \alpha_i \le C$ this means that there needs to be an upper and lower boundary so that if α_i changes, α_j still can change accordingly so that both still lie in the range of 0 to C. The boundary is given as:

$$L = \begin{cases} \max\{0, \alpha_j - \alpha_i\}, & y_i \neq y_j \\ \max\{0, \alpha_i + \alpha_j - C\}, & y_i = y_j \end{cases}, \quad H = \begin{cases} \min\{C, C + \alpha_j - \alpha_i\}, & y_i \neq y_j \\ \min\{C, \alpha_i + \alpha_j\}, & y_i = y_j \end{cases} \tag{1}$$

With L is lower bound and H is higher bound. Next, to compute the update α_i, it computes the minima and update α_i to that point. The clipped update α_i is given by:

$$\eta = 2K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j), \quad \alpha_i \leftarrow \alpha_i - \frac{y_j(E_j - E_i)}{\eta},$$
$$E_k = f(x_k) - y_k, \qquad f(x) = \sum_{\ell=1}^{n} \alpha_\ell \, y_\ell \, K(x_\ell, x) + b. \quad (2)$$

Where η is the stepping size. However, this is not the final updated value, it will need to be compare with the lower and the higher bound above to decide it's final updated value.

$$\alpha_i^{new} = \begin{cases} H, & \alpha_i^{new} > H, \\ L, & \alpha_i^{new} < L, \\ \alpha_i^{new}, & \text{otherwise.} \end{cases} \quad (3)$$

Then α_j will be updated as:

$$\alpha_j^{new} = \alpha_j^{old} + y_i y_j \left(\alpha_i^{old} - \alpha_i^{new}\right). \quad (4)$$

SMO will pick α_i by sequent and α_j by random, then they will be optimized accordingly in each iteration until the KTT condition is less than tolerable value. The KKT violation in SMO tells us how much a given α_i is breaking the optimality conditions. It is given as:

$$\begin{aligned} &\alpha_i \geq 0, \ \alpha_i \leq C, \\ &\alpha_i(y_i f(x_i) - 1) = 0, \\ &y_i f(x_i) - 1 \geq 0 \quad \text{if } \alpha_i = 0, \\ &y_i f(x_i) - 1 \leq 0 \quad \text{if } \alpha_i = C, \\ &y_i f(x_i) - 1 = 0 \quad \text{if } 0 < \alpha_i < C. \end{aligned} \quad (5)$$

Where $f(x_i) = \sum_{j=1}^{m} \alpha_j y_j K(x_j, x) + b$ Now, alpha is updated, however, there is another important parameter which is the bias. The bias also has upper and lower thresholds just like alpha. The upper threshold of the updated bias is defined as b_1 and lower thresholds is b_2.

$$b_1 = b - E_i - y_i(\alpha_i - \alpha_i^{old})K(x_i, x_i) - y_j(\alpha_j - \alpha_j^{old})K(x_i, x_j)$$

$$b_2 = b - E_j - y_i(\alpha_i - \alpha_i^{old})K(x_i, x_j) - y_j(\alpha_j - \alpha_j^{old})K(x_j, x_j) \quad (6)$$

Then, the bias will be updated as:

$$b = \begin{cases} b_1, & \text{if } 0 < \alpha_i < C, \\ b_2, & \text{if } 0 < \alpha_j < C, \\ \frac{1}{2}(b_1 + b_2), & \text{otherwise.} \end{cases} \quad (7)$$

This will check for if the alpha correspond to a sample is a support vector, it will update to that corresponding threshold. If both are not or are support vectors, then it will take the average of them.

3 High-level summary learning theory framework components of SVM

3.1 Data

The input for a SVM model is an array of vectors $[x_1, x_2, x_3, ..., x_m]$ vector represent a data sample and the number of feature is the dimension of the vector lives in:

$$x_i \in \mathbb{R}^n \quad \text{where n is the number of feature}$$

because it is an array of vectors lives in the same dimensional space. It can be said that the input for SVM model is an [m x n] matrix where m is the number of samples and n is the number of features.

The output on the other hand is the data label or class. Let z be the output, then $z \in \{-1, 1\}$.Since it is not a probabilistic model hence SVM does not product a probabilistic output which does not tell how confident the model is when predicting the sample.

3.2 Hypothesis space

The hypothesis space of SVM varies by the kernel it used. However, if kernel is not used, the hypothesis space is a hyperplane.

$$\mathcal{H} = \{ f(x) = \mathbf{w}^\top \mathbf{x} + b \mid \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R} \}$$

However, if SVM is kernelized. The hypothesis space in the mapped space is still a hyperplane yet the true hypothesis space would be different. For example, polynomial (which is the kernel use in the implementation) has the true hypothesis space of:

$$\mathcal{H}_{\text{poly}} = \left\{ f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i \left(\mathbf{x}_i^\top \mathbf{x} + c \right)^d + b \;\middle|\; \alpha_i \in \mathbb{R} \right\}$$

3.3 Criterion / Loss

Although the objective function is transformed to the dual form to use the kernel trick which make the loss barely visible. However, optimizing the dual form would be equivalent to optimizing the original objective function without constraint. The original objective function use a Hinge loss which is specially designed for the regularization term. The Hinge loss only penalize the model on the samples within the margin of 1 and the samples that is misclassified.

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} \max\left(0,\ 1 - y_i \left(w^T x_i + b\right)\right)$$

Hence, the objective function is to minimize the loss and also the regularization term.

$$\min_{w,b}\ \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \max\left(0,\ 1 - y_i \left(w^T x_i + b\right)\right)$$

3.4 Optimizer

To optimize the objective function (dual form). It use the Sequential Minimal Optimization algorithm to solve the best value of α and b. The algorithm although lengthy, it can be intuitively understood. In short, a_i is in quadratic form hence it can be computed for the optimized value in 1 go. However, because of the constraint where all α needs to added up to 1, another α need to be updated to meet that constraint. Moreover, because both 2 α values need to meet another constraint which is $0 \leq \alpha_i \leq C$, they need to set a upper and lower bound to not optimizing too far. The process will continue until converge by checking the KKT condition with a tolerant value or run out of maximum iteration.

4 Implementation

4.1 Theory to code Simple Support Vector Machine

The code will develop a classed called SimpleSVM which will transform all the theory discussed to code. I would explained which code is transformed and how it translate from a theoretical and mathematical formula or equation.

4.1.1 Core Attribute

As discussed, the objective of the learning task is to optimize alpha and a bias term. Also, the input needs to be stored to compute the Kernel matrix which is another attribute. Another attribute is the label vector y which store values of the classes where each value in the set {-1, 1} (Binary class). Moreover, the penalty parameter C is another attribute. As discussed in the SMO part, the KKT needs to be less than a tolerant value which needs to be stored in the attribute also. In addition, as the problem can be large and complicated, the time for computation and the number of iterations may be large, therefore it needs to specify the maximum pass (goes through every alpha value) and maximum iteration.

```python
class SimpleSVM:
    def __init__(self, C=1.0, degree=3, gamma=None, coef0=1.0, tol=1e-3,
   max_passes=5, max_iter=10000, seed=0):
        self.C = C
        self.degree = degree
        self.gamma = gamma  # if None, uses 1 / n_features
        self.coef0 = coef0
        self.tol = tol
        self.max_passes = max_passes
        self.max_iter = max_iter
        self.rng = np.random.default_rng(seed)
        # learned params
        self.alphas = None
        self.b = 0.0
        self.X = None
        self.y = None
        self.K = None  # kernel matrix
```

4.1.2 Defining Kernel Function

In this implementation, polynomial kernel is used. It is defined as:

$$K(x_i, x) = \left(\gamma x_i^T x + r\right)^d$$

Where d is the degree of the polynomial and d = 1 equivalent to a linear kernel. γ and r are the coefficient and the free variables of the polynomial. Transforming this to code, it will be:

```python
    def _poly_kernel(self, X, Z):
        if self.gamma is None:
            gamma = 1.0 / X.shape[1]
        else:
            gamma = self.gamma
        return (gamma * (X @ Z.T) + self.coef0) ** self.degree
```

4.1.3 Fitting/Training

Setting up Attribute

```python
    def fit(self, X, y):
        X = np.asarray(X, dtype=float)
        y = np.asarray(y, dtype=float)
        assert set(np.unique(y)) <= {-1.0, 1.0}, "y must be in {-1, +1}"

        n_samples, n_features = X.shape
        self.X = X
        self.y = y
```

This will be taking in the input and the label y and ensure that the label y needs to be in -1, +1 otherwise it will be raising an error.

```python
        self.alphas = np.zeros(n_samples)
        self.b = 0.0
```

These two line initiates the learning parameter which is alphas (same dimension of the number of sample) and the bias.

```python
        # Precompute full kernel matrix
        self.K = self._poly_kernel(X, X)

        passes = 0
        iters = 0
```

Then it will compute the Kernel Matrix, in this case, it computes using the polynomial matrix. Then initialize the number of passes and iterations.

```
        def f(i):
            # decision value at training point i
            return np.sum(self.alphas * y * self.K[:, i]) + self.b
```

The function f(i) is the code version of the $f(x_i) = \sum_{j=1}^{m} \alpha_j y_j K(x_j, x) + b$ which mentioned in the SMO.

Sequential Minimal Optimization

This section might be lengthy as the algorithm was discussed earlier on. Hence, I will explain which theoretical knowledge correspond to the code.

```
        while passes < self.max_passes and iters < self.max_iter:
            num_changed_alphas = 0
            for i in range(n_samples):
                Ei = f(i) - y[i]
```

This will check for the number of passes (goes through all alphas) and the number of iterations if it still in the limit, the SMO will continue. The error will be f(i) subtract the label as discussed in the SMO.

```
                # KKT violation check
                if ((y[i]*Ei < -self.tol and self.alphas[i] < self.C) or
                    (y[i]*Ei > self.tol and self.alphas[i] > 0)):
```

This is the code version of the KKT condition check to see if that value of alpha is optimized or not.(Correspond to (5))

```
                    # pick j != i
                    j = i
                    while j      == i:
                        j = self.rng.integers(0, n_samples)
                    Ej = f(j) - y[j]

                    ai_old = self.alphas[i]
                    aj_old = self.alphas[j]
```

This will take 2 values α_i and α_j and compute the where α_j is randomly chosen. Then the error of α_j will be compute similar to α_i

```
                    # Compute L and H (box constraints)
                    if y[i] != y[j]:
                        L = max(0.0, aj_old - ai_old)
                        H = min(self.C, self.C + aj_old - ai_old)
                    else:
                        L = max(0.0, ai_old + aj_old - self.C)
                        H = min(self.C, ai_old + aj_old)
                    if L == H:
                        continue
```

This will compute the lower and the upper bound of the new update which again the code version of the boundary discussed in SMO algorithm. (Correspond to (1))

```
                    # Compute eta (second derivative along the line)
                    eta = 2.0 * self.K[i, j] - self.K[i, i] - self.K[j, j]
                    if eta >= 0:
                        # Non-positive definite along direction, skip (rare with
   poly/RBF)
                        continue
```

These line of code will compute the stepping size η which can also be referred as the second derivative. (Correspond to (2))

```
                    # Update alpha_j
                    self.alphas[j] = aj_old - (y[j] * (Ei - Ej)) / eta
```

Then it will compute the new updated value (not finalized).

```

                    # Clip to [L, H]
                    if self.alphas[j] > H:
                        self.alphas[j] = H
                    elif self.alphas[j] < L:
                        self.alphas[j] = L

                    if abs(self.alphas[j] - aj_old) < 1e-6:
                        # No significant change
                        self.alphas[j] = aj_old
                        continue
```

Then, it will take the boundaries computed earlier on and compute the final updated value. Note that if there is not a significant difference between the updated value and the old value, they will skip the update.(Correspond to (3))

```
                    # Update alpha_i (in tandem)
                    self.alphas[i] = ai_old + y[i]*y[j]*(aj_old - self.alphas[j])
```

Then α_i will be updated to respect the constraint. (Correspond to (4))

```
                    # Compute b1, b2 and update b
                    b1 = (self.b - Ei
                          - y[i]*(self.alphas[i] - ai_old)*self.K[i, i]
                          - y[j]*(self.alphas[j] - aj_old)*self.K[i, j])
                    b2 = (self.b - Ej
                          - y[i]*(self.alphas[i] - ai_old)*self.K[i, j]
                          - y[j]*(self.alphas[j] - aj_old)*self.K[j, j])

                    if 0 < self.alphas[i] < self.C:
                        self.b = b1
                    elif 0 < self.alphas[j] < self.C:
                        self.b = b2
                    else:
                        self.b = 0.5 * (b1 + b2)

                    num_changed_alphas += 1
```

Finally, the bias will be computed. (Correspond to (6),(7))

4.1.4 Inference/Projecting

As $w = \phi(X)^T\alpha$ and the testing samples also kernelized. Hence the hyperplane will be rewritten as

$$f(x) = \sum_{i=1}^{n_{\text{train}}} \alpha_i y_i \, \phi(x_i)^\top \phi(x) + b$$

Notice that $\phi(x_i)^\top\phi(x)$ is the kernel function. Hence it can be rewritten as:

$$f(x) = \sum_{i=1}^{n_{\text{train}}} \alpha_i y_i K(x_i, x) + b$$

Therefore, to infer, it is necessary to compute another kernel matrix of the training and the testing samples. However, it a lot of the values are 0 and in the range as only support vectors have non-zero value. The code for this is written as:

```
    def project(self, X):
        """
        Compute decision function f(x) for arbitrary X.
        """
        X = np.asarray(X, dtype=float)
        Kx = self._poly_kernel(self.X, X)  # shape (n_train, n_test)
        return (self.alphas * self.y) @ Kx + self.b
```

Hence, when predict, the sign of the projection is the predicted class.

```python
    def predict(self, X_test):
        return np.sign(self.project(X_test))
```

4.1.5 Getting Support Vectors

As discussed, a lot of alphas will be 0 only the support vectors matter hence for the purpose of visualization the which samples have non-zero alpha value.

```python
    def support_vectors_(self):
        return self.X[self.alphas > 1e-8], self.y[self.alphas > 1e-8], self.alphas
    [self.alphas > 1e-8]
```

This will return the support vectors (which 1e-8 means 0) with their label and alpha value.

4.2 Training Process

4.2.1 Input

The model will be trained on the Iris dataset and "Breast Cancer Benign vs Malignant" dataset. Firstly, the Iris dataset has 150 samples, 3 classes 4 attributes which corresponding to each characteristic of the flower such as the sepal length. Therefore, the input for the Iris dataset is a $[150 \times 4]$ matrix I where I_{ij} is the sample i attribute j. The label y on the other hand because of the model is binary class classification, its task is to differentiate between Iris-Setosa and other class. Therefore, the input for the label is a vector with 150 dimensions where $y_i = 1$ is Setosa class and -1 for non-Setosa class. All attributes of Iris dataset is numeric therefore there is no need to encode non-numeric values.

On the other hand, Breast Cancer dataset will have 569 samples with 2 class and 30 numeric attributes which corresponding to a characteristic of the patient such as there condition or age. This means that a sample will live in a 30-dimensional space which increase the complexity of the task because of the curse of dimensionality (points are far from each other). However, this can be a good example of the generalization ability of SVM as it learns the relationship that maximizes the margin. The input for the samples will be a $[569 \times 30]$ matrix I where I_{ij} is the sample i attribute j. The input for the target label is a 569-dimension vector where $y_i = 1$ for benign and -1 for malignant. Similar to Iris, all the attribute data type is numeric therefore no encoding is necessary.

4.2.2 Output

The objective of the learning task is to optimize the objective function which is the dual form of SVM by using SMO to optimize learnable parameter alphas and b. The output will be an optimized function

$$f(x) = \sum_{i=1}^{n_{\text{train}}} \alpha_i y_i K(x_i, x) + b$$

Where $\alpha_i \ \forall i$ and b are optimized. The output for the prediction will be $z = \text{sign}(f(x))$ where $z \in \{1, -1\}$. Where in the Iris data set, $z = 1$ means it is labeled as Setosa and $z = -1$ means it is non-Setosa. On the other hand, with Breast Cancer dataset, $z = 1$ means it is labeled as benign and $z = -1$ means it labeled as malignant . To optimize these parameters the SVM object will call the fit method which discussed in the theory-to-code section. Then it will use the predict method to give the prediction.

4.2.3 Data preparation

Because the dataset itself has 3 class where Setosa has label of 0. Therefore, first step is to map 0 to 1 and others to -1. Breast Cancer has label {0,1} hence it needs to change label 0 to be -1. Then both the dataset is scaled using the Z-score normalization or the Standard Scaler given by:

$$Z = \frac{X - \mu}{\sigma}$$

Where X is the old data sample, Z is scaled value, μ is the mean and σ is the standard deviation. In other words, it transforms the data point into it's corresponding z-score. If a new data point is added, the new transformed data point may be different as the mean and standard deviation will differ. Then the dataset will be split into 2 sets, training and testing which the ratio of 70:30 for both dataset and the portion of the classes in training and testing will be the same. Now, data is prepared for training.

4.2.4 Training results and evaluation

Iris dataset

On this dataset, it achieves a test accuracy of 0.95 on training set and 0.97 on testing set. The accuracy is defined as the ratio of the correctly labeled prediction and the total number of predictions. The F1-score of the testing set is also high at 0.96 suggesting that the model can predict accurately on both class despite there is a class imbalance (class non-senota made of 2 other class accounts for 100 samples). The F1-score is a metric to measure the performance of the model on both class by taking the harmonic mean of the precision and the recall. Although the purpose of F1 score does not clearly reflect the objective from this dataset, the accuracy reflects the task as the task is to differentiate the Setosa from other species. From the decision boundary, it can be observed that the hyperplane lies exactly at the middle of 2 classes.

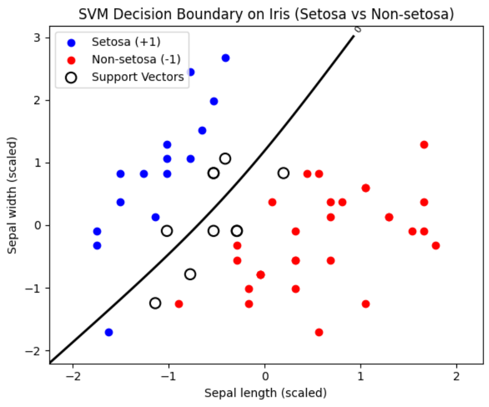

Figure 1: Decision Boundary of SVM on Iris dataset

Breast Cancer dataset

On the breast cancer data, training with parameter C = 1 without optimizing the hyperparameter, the model achieves 0.988 accuracy and 0.99 F1-score. This means that it did exceptionally well on both of the classes. In this dataset, F1 score truely reflects the task as it is important to correctly classify both the classes not just 1 (patient may be affected if diagnosed falsely). Interestingly, when implementing grid search for optimal C by further split the training set to train and validate set with ratio 60:10 to train and validate to see which C gives the best validation accuracy, the accuracy dropped to 0.95 and F1-score of 0.96. This is because it chooses the optimal C of 0.1 as the validation of C=0.1 and C=1 although equal 1 but 0.1 comes first making it is the chosen one.

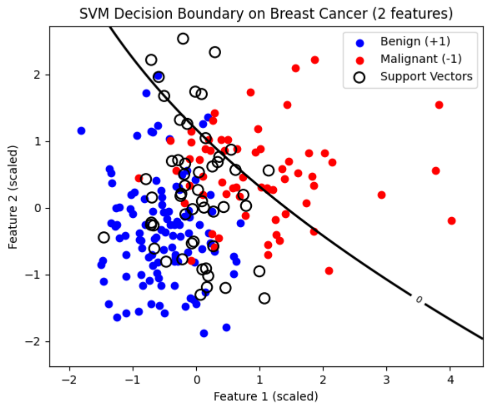

Figure 2: Decision Boundary of SVM on Breast Cancer dataset

5 Advantages and limitations of SVM

5.1 Advantages

Avoid overfitting by default

SVM has the regularization term and the hinge loss designed for maximize the margin which allows it to generalize on unseen data. However, if parameter C is set extremely high on purpose, it may vanish the regularization term and penalize the model only on the loss. Therefore, if C is set as default of 1, it can generally avoid overfitting.

Effective in high dimensional space

As demonstrated in the Breast Cancer dataset, the dataset has 30 feature which SVM still works well with. This is because SVM maximize the margin hence it can still handle large dimensional data.

Rich hypothesis space / Complex pattern

Due to the use of kernel, the model can learn more complex patterns by changing different kernel's function. Hence, it allows the model to have the flexibility to choose and try different hypothesis space by changing the kernel function.

5.2 Limitations

Pre-defined kernels

Kernel trick is one of the advantages of SVM, but it is a critical limitation also. True function may not lie within the chosen kernel space, leading to poor generalization. This may be counter intuitive as the SVM has a great generalization ability. However, this generalization ability is contingent on the alignment between the kernel-induced feature space and the true underlying data distribution.

Takes up memory spaces

By using the Kernel Trick, it trades the time complexity for space complexity. Although the time complexity significantly reduces by the kernel trick, additional memory needs to be used for the Kernel matrix. Furthermore, when testing, it also needs to compute the kernel matrix of the training and the testing data.

No probabilistic output

SVM is not a probabilistic model like logistic regression where the output also shows the probability or the confidence of the prediction. Personally, I think this is the most significant limitation of SVM as I used to have a chess outcome prediction where it needs the probability of the match to do further statistical operations.

6 Future Enhancement

I asked GitHub Copilot to suggest me some future enhancements that I can use to improve my model. It suggested that I can:

Create different kernels

I think that this is critical enhancements since I want to keep the simplicity of the model, I only implement the polynomial kernel. However, in future, I can use different kernel such as the RBF kernel to enrich the choices of hypothesis space.

Probabilistic output

As discussed in the limitation, SVM does not output a probabilistic prediction by default. However, it can be done using Platt scaling which can be implemented to give the probabilistic output.

Multi-class classification

Currently, my SVM only a binary classifier, hence it can only deal with 2 classes where for example Iris dataset has 3 classes in default. This can be achieved by using strategies such as 1 vs rest or one-vs-one. For example, 1-vs-rest strategy will build 3 classifiers for each species in Iris dataset to differentiate them from other species.

Cross validation

Cross validation will split the training set in to k folds where 1-fold will be the validating set. Each round, the validating fold will rotate through every fold and choose the fold that optimizes the validation accuracy. Currently, I only split into a static validating set for optimizing the hyperparameter but in future in can be done using cross validation.

7 Final remark and reflection

After studying the theory of SVM and implement it into code, I learned a lot of things. Before learning the theory of SVM, I only understand it only on a surface which I only understand that is map the input to a kernel space and it learn the hyperplane there. However, I did not understand how they did it and how does it is optimized. After studying suggested materials, I understand deeply the art behind SVM. Turns out, my intuition was partially wrong, the kernel trick does not actually map the samples to a higher dimensional space but rather use the relationship between the samples. Furthermore, it also leverage my understanding of other models because I have to compare with other model and I can understand more about the behavior of those model and what made SVM stands out from those. In addition, implementing from theory to code helped me to understand the code behind existing framework/libraries such as sci-kit learn. I believe that understand deeply the theory and able to translate it into code can help me understand the behavior of model and diagnose problem better when using frameworks such as sci-kit learn.

Finally, in my view, SVM can be very easy to explain the idea intuitively but hard to understand the core theory behind it. Those theories I found somehow counter-intuitive but after managing to understand it, I can see how it used to be the most used model before deep learning era.

8 References

Chervonenkis, A. Y. (2013). Early history of support vector machines. In B. Schölkopf, Z. Luo, & V. Vovk (Eds.), *Empirical inference* (pp. 13–20). Springer. https://doi.org/10.1007/978-3-642-41136-6_3

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine Learning*, *46*, 145–171. https://doi.org/10.1007/BFb0026683

Page, D. (n.d.). Svm by sequential minimal optimization (smo) [PDF slides]. https://pages.cs.wisc.edu/~dpage/cs760/MLlectureSMO.pdf

Platt, J. C. (1999). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods: Support vector learning* (pp. 185–208). MIT Press.

Weinberger, K. (2018). Lecture 9: Svm [Lecture notes]. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html

9 Appendix

Link to Google Colab file

https://colab.research.google.com/drive/1UplrqJ4dx14IhkkfVLX3rpmMPj4Q01tK?usp=sharing

Link to GitHub Repository - including source code for Manim presentation

https://github.com/LuuMTran/Machine-Learning-Assignment-A2-Support-Vector-Machine

Link to Overleaf documents

https://www.overleaf.com/read/ksbcrsrrkgwr#0bcda3