Support Vector Mechine

Qiang Hu, Gengshang Dong, Pinzhao Li, Zhili Yang

June 3, 2023

Contents

1 Contribution

Qiang Hu	Background, Motivation, Problem formulation and code of SMO
Gengshang Dong	Lagrangian and KKT of Hard Margin, Soft Margin and contrast
Pingzhao Li	Nonlinear separable case, Kernel function and thoery of SMO
Zhili Yang	Application and code of Soft Margin, Linear Indivisible and Multiple Classification.

2 Background and Motivation

2.1 Background

Support Vector Machine (SVM) is a classic supervised learning algorithm used for solving classification problems. The background of SVM can be traced back to the development stages of the 1960s and 1970s, as well as the subsequent theoretical and algorithmic breakthroughs in the 1980s and 1990s.

In the 1960s and 1970s, the focus of research in statistical learning theory and machine learning was primarily on linear classification methods such as Perceptron and Linear Regression. However, these methods had certain limitations when it came to solving nonlinear classification problems.

In the 1980s, statistician Vladimir Vapnik and his colleagues began researching the theoretical foundations of machine learning and proposed the theory of statistical learning. This theory provided mathematical analysis of the probabilistic guarantees and generalization capabilities of learning problems.

In 1989, Vladimir Vapnik and Alexey Chervonenkis published a paper titled "On a Complexity of Learning," introducing the concept of VC dimension (Vapnik-Chervonenkis dimension), which measures the representational capacity of a learning algorithm.

In 1992, Vladimir Vapnik and others published a paper titled "A Training Algorithm for Optimal Margin Classifiers," formally presenting the basic ideas and optimization algorithms of Support Vector Machine (SVM). They introduced the concept of maximum margin classifiers, which construct classifiers by maximizing the margin between samples, and employed techniques such as Lagrange multipliers and convex optimization to solve the optimization problem.

Subsequently, in 1995, Corinna Cortes and Vladimir Vapnik published a paper titled "Support-Vector Networks," further refining the theory and algorithms of SVM. This paper delved into the core ideas, characteristics, and generalization performance of SVM and introduced the kernel function approach, enabling SVM to handle nonlinear classification problems.

2.2 Motivation

The core idea of SVM is to transform a classification problem into a convex optimization problem. By maximizing the margin, SVM seeks a hyperplane that can separate the samples and maximize the distance between different classes, pushing the samples as far away from the hyperplane as possible. This idea allows SVM to focus only on the samples that are closest to the decision boundary, as they have a decisive impact on the position of the boundary. As a result, compared to other classification algorithms, SVM exhibits stronger robustness.

3 Linearly separable case

3.1 Problem transformation

Let's assume that we have positive and negative sample points X_i and their corresponding labels $y_i(= \pm 1)$. We need to train a hyperplane using these sample points for classification. According to the SVM concept, we aim to maximize the margin between positive and negative samples.

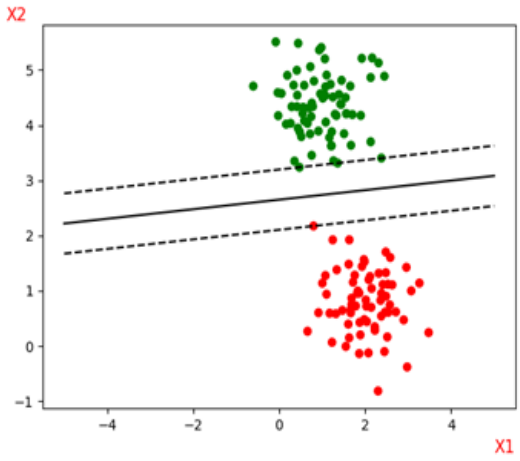

Figure 1: SVM Visualization

From the diagram, let's assume the margin is $2d$ and the equation of the plane at the center of the maximum margin region is:

$$W^T X + b = 0 \tag{1}$$

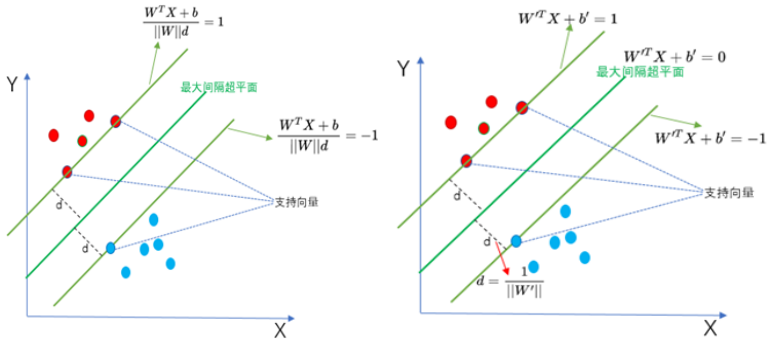

Figure 2: Problem Formulation

And the equations of the two planes defining the boundaries of the maximum margin region are:

$$\frac{W^TX+b}{\|W\|}=d,\quad \frac{W^TX+b}{\|W\|}=-d \tag{2}$$

Let:

$$W'=\frac{W}{\|W\|d},\quad b'=\frac{b}{\|W\|d} \tag{3}$$

After substituting W' and b' back, we can obtain the expressions for the maximum margin hyperplane and the maximum margin as:

$$W'^TX+b'=0,\quad 2d=\frac{2}{\|W'\|} \tag{4}$$

The constraint is that all sample points should be correctly separated by the hyperplane. From the diagram, when the sample point is a positive sample,

$$W'^TX_i+b'\geq 1 \tag{5}$$

And when the sample point is a negative sample,

$$W'^TX_i+b'\leq -1 \tag{6}$$

Multiply by the corresponding label value y_i, we can unify them as:

$$y_i(W'^TX_i+b')\geq 1 \tag{7}$$

In summary, we have obtained an optimization problem:

$$\max_{W',b'}\quad \frac{2}{\|\ W'\ \|}$$
$$s.t.\quad y_i(W'\cdot X_i+b')-1\geq 0,$$

i=1,2,$\cdots$,n

For ease of solving, we can equivalently transform it into the following form (avoiding the square root)[1]:

$$\min_{W',b'} \quad \frac{1}{2} \parallel W' \parallel^2$$
$$s.t. \quad y_i(W' \cdot X_i + b') - 1 \geq 0, \quad i = 1, 2, \cdots, n$$

Note that this is a typical Quadratic Programming (QP) problem.

3.2 Hard Margin

Let's start with a set of data points that we want to classify into two groups. We can consider two cases for these data: either they are linearly separable, or the separating hyperplane is non-linear. When the data is linearly separable, and we don't want to have any misclassifications, we use SVM with a hard margin.

Therefore our optimization problem would become:

$$\min_{w,b} \ \frac{1}{2} w^T w$$
$$\textbf{s.t.} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, 2, \cdots, n$$

It is guaranteed to have a global minimum. We can solve this by introducing Lagrange multipliers (α_i) and convert it to its dual problem:

$$L(w, b, \alpha_i) = \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1) \tag{8}$$

This is called the Lagrangian function of the SVM which is differentiable with respect to w and b.

$$\nabla_w L(w, b, \alpha) = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{9}$$

$$\nabla_b L(w, b, \alpha) = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{10}$$

By substituting them in the second term of the Lagrangian function, we'll get the dual problem of SVM:

$$\max_{\alpha} \ -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n} \alpha_i$$
$$\textbf{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

In the dual problem,we only need to solve the Lagrange multipliers α. Besides, the dual problem depending on the inner products of the training data

is applicable when extending linear SVM to learn non-linear boundaries. To continue solving the dual problem, we need to use Sequential Minimal Optimization(SMO)(3.3).

3.3 Soft Margin

However, in practice, completely linearly separable samples are rare. When a linear boundary is not feasible, or we want to allow some misclassifications in the hope of achieving better generality, we can opt for a soft margin for our classifier(Figure 3).

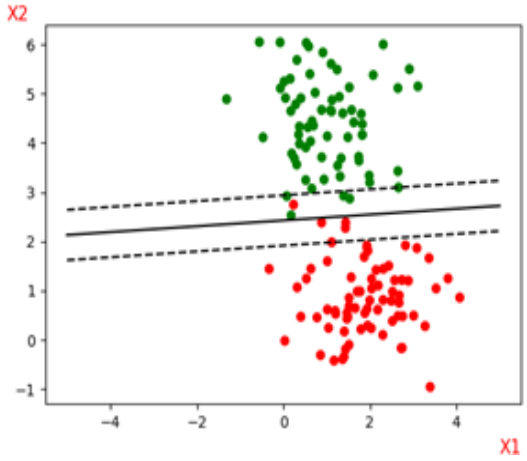

Figure 3: Allow some misclassifications to achieve better generality

For every sample point(x_i, y_i), we can introduce a slack variable $\xi_i \geq 0$. Guarantee the sum of function margin and the slack variable $\xi_i \geq 0$, then the constraints would be

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

the target function would be

$$\frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{N} \xi_i$$

where C is a constant greater than 0, which can be understood as the penalty of error sample, if C is infinite, slack variable ξ_i must be infinitesimal, so a linear SVM becomes a linear separable SVM; Only when C is finite, will some samples be allowed to not follow the constraint. According to this, we can adjust the degree of error by setting the value of C.

The loss of a misclassified point is called a slack variable and is added to the primal problem that we had for hard margin SVM. So the primal problem for the soft margin is similar to equation(10):

$$\min \ \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i$$

$$\textbf{s.t.} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i = 1, ..., n, \ \xi_i \geq 0$$

As we can see, the difference between the primal problem and the one for the hard margin is the addition of slack variables. The new slack variables (ξ_i) add flexibility for misclassifications of the model.

By applying lagrangian multiplier method to problem (15), we can get its dual problem[2]:

$$\max_{\alpha_i \geq 0}\{\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j)\}$$

$$s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0, 0 \leq \alpha_i \leq C, \quad i = 1, 2, \cdots, n$$

In the dual form, the difference is only the upper bound applied to the Lagrange multipliers. Meanwhile, according to its KKT condition, we can obtain:

$$\alpha_i = 0 \Rightarrow y_i(w \cdot x_i + b) \geq 1$$

$$\alpha_i = C \Rightarrow y_i(w \cdot x_i + b) \leq 1$$

$$0<\alpha_i<C \Rightarrow y_i(w \cdot x_i + b) = 1$$

The first formula means that if $\alpha_i = 0$, then the sample falls outside the two spaced lines. The second formula shows that if $\alpha_i = C$, then the sample may fall inside or above the two spacing lines, mainly depending on the sign of corresponding slack variable is equal to or greater than 0. The third formula shows that if $0 < \alpha_i < C$, then the sample must fall directly on the dividing line.

3.4 Hard Margin vs. Soft Margin

The difference between a hard margin and a soft margin in SVMs lies in the separability of the data. If our data is linearly separable, we go for a hard margin. However, if this is not the case, it won't be feasible to do that. In the presence of some data points that make it impossible to find a linear classifier, we would allow some of the data points be misclassified. In this case, a soft margin SVM is appropriate.

Sometimes, the data is linearly separable, but the margin is so small that the model tends to overfitting or being too sensitive to outliers. Also, in this

case, we can opt for a larger margin by using soft margin to help the model generalize better.

4 Nonlinear separable case and Solving

Certainly, so far our SVM has been relatively weak and can only handle linear cases. However, after obtaining the dual form and extending it to non-linear cases through the use of kernels, it becomes a very straightforward task.

4.1 Problem introduction

In fact, most of the time, data is not linearly separable, and in such cases, it is impossible to find a hyperplane that satisfies this condition. As we have discussed earlier, SVM handles linearly separable cases, but how does it deal with non-linear data? In the case of non-linear data, SVM employs a method called kernel trick. It involves selecting a kernel function K(,) and mapping the data to a higher-dimensional space to address the issue of linear inseparability in the original space.

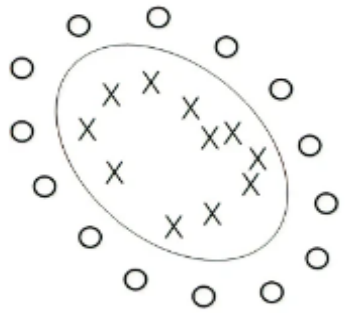

Figure 4: nonlinear separable case

Before encountering kernel functions, if we were to use the traditional approach, learning a nonlinear relationship with a linear learner would require selecting a nonlinear feature set and transforming the data into a new representation. This is equivalent to applying a fixed nonlinear mapping that maps the data to a feature space, and then using a linear learner in that feature space. Therefore, the hypothesis set considered would be of this type:

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) + b$$

Here, ϕ : X to F represents the mapping from the input space to a certain feature space. This means that constructing a nonlinear learner involves two

steps:Firstly, the data is transformed to a feature space F using a nonlinear mapping.Then, a linear learner is applied in the feature space for classification. Since the dual form is an important property of the linear learner, it implies that the hypothesis can be expressed as a linear combination of the training points. As a result, the decision rule can be represented using the inner product between the test point and the training points:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i \left\langle \phi\left(\mathbf{x}_i\right) \cdot \phi(\mathbf{x}) \right\rangle + b$$

4.2 Kernel trick

If there is a way to directly compute the inner product $<\phi(x_i), \phi(x)>$ in the feature space, just like in the original input space, it would be possible to merge the two steps and build a nonlinear learner. This approach, which allows for direct computation in the feature space, is known as the kernel trick method.

Kernel functions can simplify the computation of inner products in the feature space, and coincidentally, in our SVM, the data vectors always appear in the form of inner products. Comparing to the equation we wrote earlier, now our classification function becomes:

$$f(x) = sign\left(\sum_{i=1}^{n} \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b\right)$$

Here, the kernel function implicitly calculates the inner product $\langle \phi(x_i), \phi(x) \rangle$ without explicitly mapping the data points to the feature space. This allows us to work directly with the original input space, making the computation more efficient.

Comparing to the equation we wrote earlier, now our classification function becomes:

$$\sum_{i} \alpha_i y_i k(x_i, x) + b$$

Usually, people choose from a set of commonly used kernel functions (based on the problem and data, selecting different parameters essentially results in different kernel functions). For example:

The polynomial kernel, obviously, the example we mentioned earlier is a special case of the polynomial kernel (R = 1, d = 2). Although it may be cumbersome and unnecessary, the mapping corresponding to this kernel can actually be written out. The dimension of the resulting space is $\begin{pmatrix} (m+d) \\ d \end{pmatrix}$, where m is the dimension of the original space.

Gaussian kernel,$k(x_i, x) = \exp\left(-\frac{\|x_i - x\|^2}{2\sigma^2}\right)$ this kernel is the one mentioned earlier that maps the original space to an infinite-dimensional space. However,

if the parameter is chosen to be large, the weights on higher-order features decay rapidly, effectively representing a low-dimensional subspace (approximately, numerically speaking). Conversely, if the parameter is chosen to be small, it is possible to map any data to a linearly separable space. However, this may not necessarily be desirable as it can lead to severe overfitting issues. Overall, the Gaussian kernel exhibits significant flexibility through parameter tuning and is one of the most widely used kernel functions. The example shown in the figure below demonstrates the mapping of low-dimensional linearly inseparable data to a high-dimensional space using the Gaussian kernel:

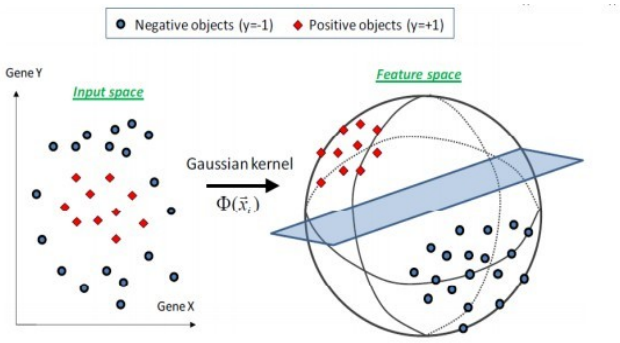

Figure 5: Gaussian kernal

Linear kernel, $k(x_i, x) = x_i \cdot x$ which is essentially the inner product in the original space. The main purpose of this kernel is to unify the "problem in the mapped space" and the "problem in the original space" in terms of their form.

After all the explanations above, readers may still not fully understand what a kernel function actually is. Let me summarize it briefly in the following three points:

In practice, we often encounter examples that are not linearly separable. In such cases, our common approach is to map the example features to a higher-dimensional space.

However, if we map all linearly non-separable examples to a high-dimensional space without any consideration, the dimensionality can become overwhelmingly large. What should we do then?

This is where the kernel function comes into play. The value of a kernel function lies in the fact that it performs the feature transformation from a low-dimensional space to a high-dimensional space, but it calculates the essential classification effect on the low-dimensional space beforehand. It then manifests this effect in the high-dimensional space, thus avoiding complex calculations directly in the high-dimensional space, as mentioned earlier.

4.3 Sequential Minimal Optimization

After deriving various types of support vector machine (SVM) dual algorithms, we have many optimization algorithms to solve this problem. However, when the sample size is large, these algorithms can become very inefficient. To address this issue, researchers have proposed several fast implementation algorithms, and the SMO (Sequential Minimal Optimization) algorithm is one of them.

This is the dual problem that we originally intended to solve.

$$\begin{aligned} \alpha max \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, n \end{aligned}$$

By using the kernel function and duality, we obtain the problem that our SMO (Sequential Minimal Optimization) algorithm aims to solve.

$$\begin{aligned} \min \quad & \Psi(\alpha) = -\sum_{i=1}^{m} \alpha_i + \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s.t.} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, m \end{aligned}$$

SMO Algorithm Steps[3]:

1. Initialize the Lagrange multipliers α_i for all training samples to zeros or small random values.
2. Select two Lagrange multipliers α_i and α_j to optimize using a heuristic or predetermined strategy.
3. Choose the corresponding training samples x_i and x_j associated with the Lagrange multipliers α_i and α_j.
4. Compute the error for both samples using the current set of Lagrange multipliers.

$$\mathrm{E}_k = g(x_k) - y_k = (\sum_{l=1}^{N} \alpha_l y_l K(x_l, x_l) + b) - y_k, \quad k = i, j$$

5. Compute the bounds for the Lagrange multipliers α_i and α_j based on their current values and the constraint $\alpha_i y_i + \alpha_j y_j = constant$.

$$0 \leq \alpha_{i=i,j} \leq C$$

$$\alpha_i y_i + \alpha_j y_j = \zeta = -\sum_{l,j}^{N} \alpha_i y_i = constant$$

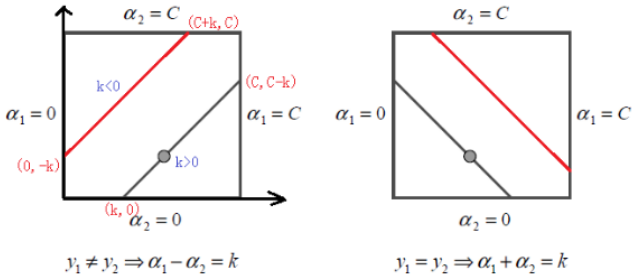

Figure 6: Bounds

The optimal solution must be on the line, thus:

$$\mathrm{L} \leq \alpha_j^{new} \leq H$$

$$\text{if} \quad \mathrm{y}_i \neq y_j, \quad L = \max(0, \alpha_j^{old} - \alpha_i^{old}), \quad H = \min(C, C + \alpha_j^{old} - \alpha_i^{old})$$

$$\text{if} \quad \mathrm{y}_i = y_j, \quad L = \max(0, \alpha_i^{old} + \alpha_j^{old} - C), \quad H = \min(C, \alpha_j^{old} + \alpha_i^{old})$$

6. Update the Lagrange multipliers α_i and α_j by solving the QP (Quadratic Programming) subproblem:

$$\begin{aligned} \min \quad & \mathrm{Q}(\alpha_i, \alpha_j) = \tfrac{1}{2}(\alpha_i^2 K_{ii} + \alpha_j^2 K_{jj}) - (\alpha_i \alpha_j) K_{ij} - (\alpha_i y_i) - (\alpha_j y_j) \\ & \text{subject to} \quad \alpha_i y_i + \alpha_j y_j = constant \\ & 0 \leq \alpha_i \leq C \\ & 0 \leq \alpha_j \leq C \end{aligned}$$

We can get the α_2 before clip:

$$\begin{aligned} \alpha_j^{new,unc} &= \alpha_j^{old} + \tfrac{y_j(E_i - E_j)}{\eta} \\ \eta &= K_{ii} + K_{jj} - 2K_{ij} \end{aligned}$$

7. Clip the updated Lagrange multipliers α_j to satisfy the bounds and calculate α_i.

$$\alpha_i^{new} = \alpha_i^{old} + y_i y_j (\alpha_j^{old} - \alpha_j^{new})$$

8. Update the threshold value b

$$\begin{aligned} \mathrm{b}_i^{new} &= y_i - \textstyle\sum_{l \neq i,j}^{N} \alpha_l y_l K_{li} - \alpha_i^{new} y_i K_{ii} - \alpha_j^{new} y_j K_{ji} \\ &= \text{-}\mathrm{E}_i - y_i K_{ii} (\alpha_i^{new} - \alpha_i^{old}) - y_j K_{ji} (\alpha_j^{new} - \alpha_j^{old}) + b^{old} \\ \mathrm{b}_j^{new} &= -E_j - y_i K_{ij} (\alpha_i^{new} - \alpha_i^{old}) - y_j K_{jj} (\alpha_j^{new} - \alpha_j^{old}) + b^{old} \end{aligned}$$

Update the threshold b based on the updated Lagrange multipliers and the error of the support vectors. Repeat steps 2-8 until convergence or a maximum number of iterations is reached. Obtain the optimized set of Lagrange multipliers i and the threshold b for classification.

Based on the above steps, we try to realize SMO by python and design a brief demo, which would be attached behind.

The SMO algorithm iteratively selects pairs of Lagrange multipliers to optimize, updates them using a QP subproblem, and adjusts the threshold for classification. This process continues until convergence, where the Lagrange multipliers satisfy the Karush-Kuhn-Tucker (KKT) conditions or a stopping criterion is met.

In summary, the basic idea of the SMO (Sequential Minimal Optimization) algorithm is to maximize the efficiency of the Chunking method proposed by Vapnik in 1982. The SMO algorithm selects only two Lagrange multipliers, α_i and α_j, for adjustment in each iteration, while keeping the other multipliers fixed. After obtaining the updated values of α_i and α_j, the algorithm improves the remaining multipliers using these updated values. Compared to conventional decomposition algorithms, although the SMO algorithm may require more iterations, each iteration involves a smaller computational cost. As a result, the algorithm exhibits good convergence speed and does not require storing the kernel matrix or performing matrix operations.

5 Coding

Next, we will introduce how to implement some SVM algorithms with computer programs. Thanks to Python's sklearn library, we don't need to write complex algorithms repeatedly in each experiment. Calling functions directly makes solving some problems much easier. Note, however, that the code in our report is incomplete, and details require opening the three accompanying Python files.

5.1 Soft Margin

Let's start with the implementation of soft margin. It is well known that soft margin can well cancel out the interference of noise points, in other words, it can accept a certain degree of error.

First, we need to read the data inside the file.

```
def npz_read(file_dir):
    npz = np.load(file_dir)
    data = npz['data']
    label_list = npz['label']
    npz.close()

    return data, label_list

file_name = 'Dataset/9.npz'
data, label_list = npz_read(file_name)
```

Then there is the training of the test set.

```
def split_train_test(data, label_list):
    xtrain, xtest, ytrain, ytest = train_test_split(data, label_list,
        test_size=0.3)
    return xtrain, xtest, ytrain, ytest
xtrain, xtest, ytrain, ytest = split_train_test(data, label_list)
```

This is followed by a direct call to the sklearn library method. Note the setting of the C value size. The smaller the value is, the softer it will be; otherwise, the harder it will be.

```
    linear_svm = svm.SVC(kernel='linear', C=1)
```

The result see in Figure C=1.

In fact, soft margin do not require that our data be linearly separable, so they tend to have a wider range of applications than hard margin.

5.2 Linear Indivisible and kernel Function

If the data set is completely linear and indivisible, and the accuracy of using the soft interval model is low, consider other options, such as the following figure:

Here are some common kernel choices:

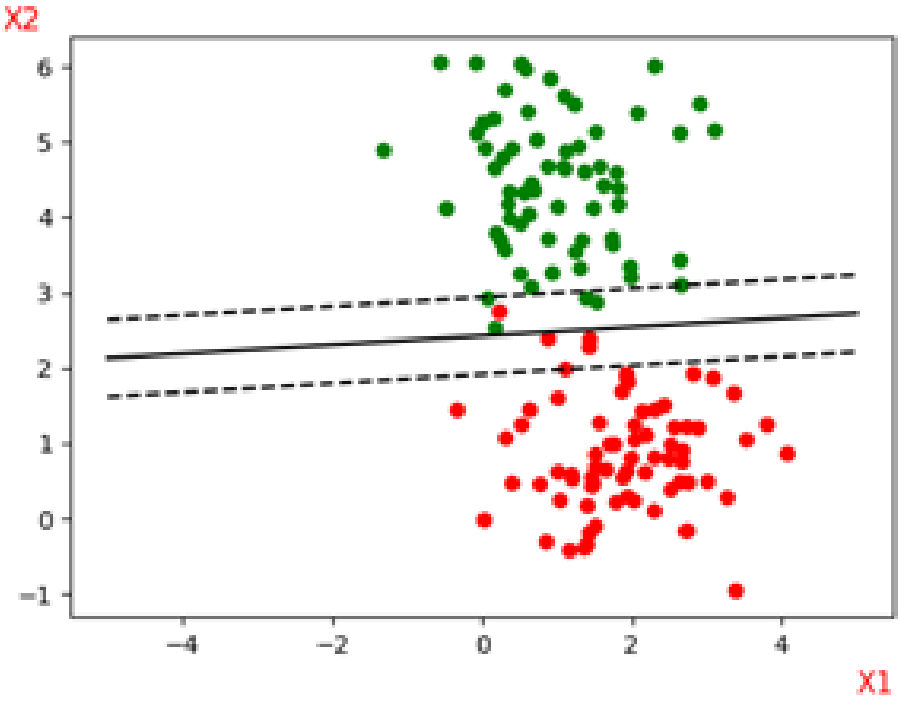

Figure 7: C=1

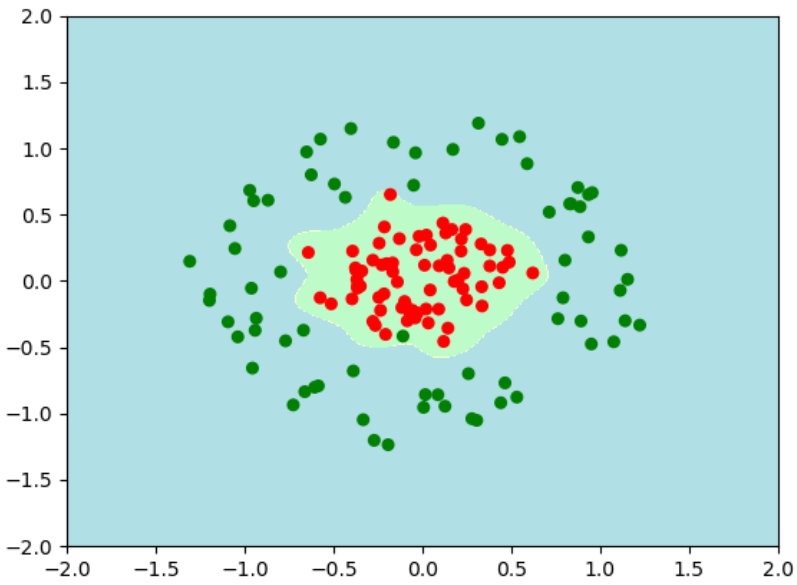

Figure 8: Linear Indivisible

Linear kernel: can only solve linear separable problems, easy to use, strong interpretation.

Polynomial kernel: makes linearly indivisible data linearly divisible by raising dimension. Can solve some nonlinear problems, the power number is too large to apply.

Gaussian kernel: Also called RBF kernel, or radial basis kernel, the function can be mapped to infinite dimensions (by Taylor series expansion) with only one argument. It is easy to overfit, poor interpretability and slow calculation speed.

Laplace kernel: The Laplace kernel is completely equivalent to the exponential kernel, with the only difference being that the former is less sensitive to parameters and is also a radial basis kernel function.

Sigmoid kernel: also known as hyperbolic tangent kernel. When Sigmoid function is used as kernel function, support vector machine is a multi-layer perceptron neural network.

For the data in the figure above, we select the Gaussian kernel function.

```
1 rbf_svc = RBFKernelSVC(gamma=10)
```

The Gauss kernel in the scikit-learn library is:

$$K(x, y) = e^{-\gamma||x-y||^2}$$

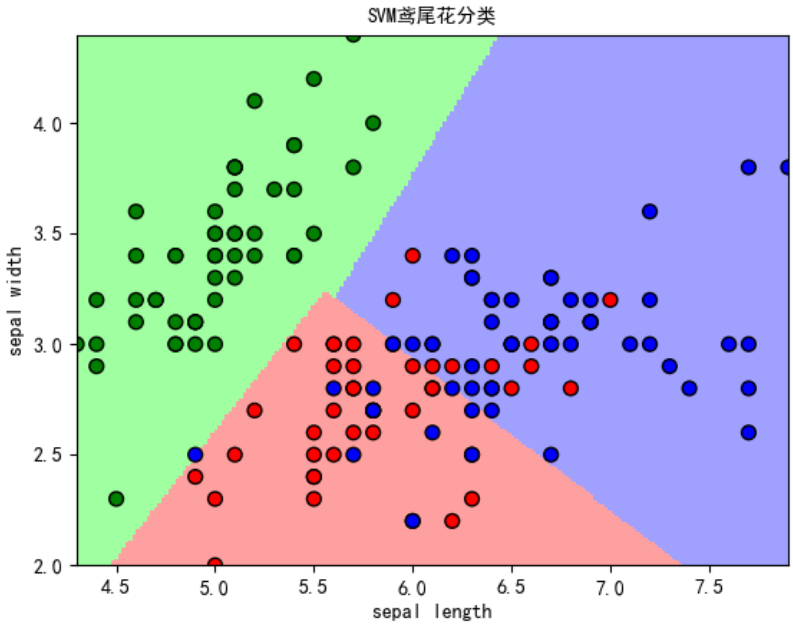

Figure 9: Iris

The value is the reciprocal of the Gaussian function sigama, and the larger the value, the narrower the Gaussian distribution, the more the model tends to overfit, and the smaller the value, the more the model tends to underfit.

5.3 Multiple Classification Problem

For multi-classification problems, there are generally two solutions: The first is the one-versus-rest method (OVR)

If we have four categories to divide (i.e., four labels), they are A, B, C, D. So when we extract the training set, extract it separately:The vector corresponding to A is taken as a positive set, and the vector corresponding to B, C and D is taken as a negative set;And so on

During the test, the samples are classified and predicted by these four classifiers, and the one whose result is a positive set is the prediction result we want.

The second one is the one-versus-one method.

The practice is to design a SVM between any two classes of samples, so k(k1)/2 k(k1)/ 2k(k1)/2 SVMS are required for k kk classes of samples. When an unknown sample is classified, the category with the most votes is the category of the unknown sample. This strategy is called voting method.

Let's say we have four categories A, B, C, and D. During the training, we selected the corresponding vectors of (A, B), (A, C), (A, D), (B, C), (B, D) and (C, D) as the training set, and then got six training results. During the test, we tested the corresponding vectors against the six results respectively, and then adopted the voting form, and finally got a set of results. When there are many categories, the number of models is large and the cost is high.

Iris in this example has 150 data samples, divided into 3 categories with 50 data in each category, and each data contains 4 attributes. The four attributes of calyx length, calyx width, petal length and petal width can be used to predict which of the three categories of Virginica (Setosa, Versicolour and versicolour) iris flowers belong to.

References

[1] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. A training algorithm for optimal margin classifiers. In Proceedings of the fifth annual workshop on Computational learning theory (COLT '92). Association for Computing Machinery, New York, NY, USA, 144–152. https://doi.org/10.1145/130385.130401

[2] Cortes, C., Vapnik, V. Support-vector networks. Mach Learn 20, 273–297 (1995). https://doi.org/10.1007/BF00994018

[3] Platt, John. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Advances in Kernel Methods-Support Vector Learning. 208.