Optimization Algorithms for Training Support Vector Machines

Kemiao Huang, 11610728, *Undergraduate, CSE*

***Abstract*— During the training of support vector machines, two classical methods are widely used, namely stochastic gradient descent (SGD) and sequential minimal optimization (SMO). In this project, those two algorithms are implemented based on Python to train SVM and predict labels and realize recognition.**

***Index Terms*— Stochastic gradient descent; Sequential Minimal Optimization; Support Vector Machine**

I. Preliminaries

THE support vector machine is a very useful model for computer recognition and classification. It is a self-supervised machine learning model with wide application field. This project aims to implement the training for SVM in an efficient way.

A. Problem Description

Given a training dataset with multidimensional space points and their own labels. SVM tries to separate the points with different labels by a max interval line or curve to give a best predict for the test points.

B. Problem Application

SVMs are helpful in text and hypertext categorization as their application can significantly reduce the need for labelled training instances in both the standard inductive and transductive settings. Classification of images and hand-written characters recognition can be performed using SVMs. In addition, it has been widely applied in the biological sciences.

II. Methodology

This section generally describes the two training methods and mathematical derivation for SVMs.

A. Notation

The important variable notations are shown in TABLE 1.

B. Data Structures

In this project, little special data structure is used. Overall, the most important structure is Numpy array. The mathematical matrix calculation is the main implementation for stochastic gradient descent algorithm and sequential minimal optimization algorithm.

C. Model Design

SVM model is trained by eigenvectors using stochastic gradient descent algorithm or sequential minimal optimization algorithm.

TABLE I
IMPORTANT VARIABLES USED IN THE REPORT

Variables	Descriptions
n	number of dimension of eigenvectors
w	weight matrix for SVC
b	element of linear function for SVC
X	input eigenvectors
y	input labels
γ	learning rate for SGD
ϵ	tolerance for soft margin algorithm
C	penalty parameter for soft margin algorithm
N	number of iterations

1) convex quadratic programming: The support vector classifier (SVC) is:

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

The objective function for convex quadratic programming (QP) is:

$$\min_{w,b} \frac{2}{||w||},$$

$$\text{s.t. } y_i((w \cdot x_i) + b) \geq 1, i = 1, \ldots, l$$

To solve the best w and b, SGD uses loss function $max(0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b))$ to evaluate the difference between the prediction of current state and the target value and then update the w of the reverse direction from the gradient according to the learning rate γ. [1]

2) Lagrange Duality: To solve the original in an easy way, using Lagrange duality to transform the equation is a good choice, especially for multidimension problems. [1] It will introduce the Lagrange multiplier and the condition for the transformation is KKT condition.

$$max_\alpha W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^i y^j \alpha_i \alpha_j \langle x^i, x^j \rangle$$

with following constraints:

$$0 \leq \alpha_i \leq C, i = 1, \cdots, m$$

$$\sum_{i=1}^m \alpha_i y^i = 0$$

To solve the dual form of the objective function, SMO has rise.

D. Details of Algorithms

1) Stochastic Gradient Descent: Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just

Algorithm 1 SGD

add one column at the front of the input eigenvector and the weight matrix.
generate random eigenvector and labels from input:
$ordered\_vector \leftarrow [1, \cdots, n]$
for $i \leftarrow 1$ to N **do**
  $randomize \leftarrow$ shuffle$(ordered_vector)$
  $random\_X \leftarrow X[randomize]$
  $random\_y \leftarrow y[randomize]$
  $loss \leftarrow 0$
  for x_i, y_i in $zip(random\_X, random\_y)$ **do**
    $loss \leftarrow loss + \max(0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b))$
    if $y_i * x_i \cdot x_i < 1$ **then**
      $w \leftarrow w - \gamma * (-y_i * x_i)$
    end if
    record loss for every iteration:
    print $i : loss$
  end for
end for
return w

recently in the context of large-scale learning [2]. Using stochastic way is just reduce the time for calculating the huge number of data.

To give a prediction after training, just add a column at the front of the test matrix and make dot product with w. The prediction result should be approximated to 1 or -1.

2) Sequential Minimal Optimization: Sequential Minimal Optimization, or SMO. Training a support vector machine requires the solution of a very large quadratic programming (QP) optimization problem. SMO breaks this large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. [3]

The kernel functions used in this project are linear, polynomial and Gaussian RBF.

To give the prediction, just put the value of w and b into the SVC and approximate it into 1 or -1.

III. Empirical Verification

This section shows the results for SVM project.

A. Dataset

Datasets such as scikit-learn dataset are simply used for testing the utility for my code. The library functions are mainly used for comparing the performance with my implementation.

B. Performance Measurement

The performance is measured by the time cost and the difference between prediction and ground truth.

C. Hyperparameters

1) SGD: The number of epochs for one big iteration is 2500. The learning rate is 0.01

2) SMO: The parameters are the same as the default parameters as scikit-learn's SVM.SVC. C = 1.0, ϵ = 0.001, degree for polynomial equation = 3.

D. Experimental Result

The results are shown in table 2.

Algorithm 2 SMO

initialize α to be all 0
while 1 **do**
  $\alpha' \leftarrow \alpha$
  for j from 1 to n **do**
    $i \leftarrow$ random integer other than j in [1, n]
    $x_i, x_j, y_i, y_j \leftarrow X[i], X[j], y[i], y[j]$
    $k_{ij} \leftarrow k(x_i, x_i) + k(x_j, x_j) - 2 * k(x_i, x_j)$
    if $k_{ij} = 0$ **then**
      continue
    end if
    $\alpha'_j, \alpha'_i \leftarrow \alpha[j], \alpha[i]$
    if $y_j \neq y_i$ **then**
      $L \leftarrow \max(0, \alpha'_j - \alpha'_i)$
      $H \leftarrow \min(C, C - \alpha'_j + \alpha'_i)$
    else
      $L \leftarrow \max(0, \alpha'_i + \alpha'_j - C)$
      $H \leftarrow \min(0, \alpha'_i + \alpha'_j)$
    end if
    $w \leftarrow X^T \cdot \alpha \times y$
    $b \leftarrow \text{mean}(y - X^T \cdot W^T)$
    $E_i \leftarrow \text{predict}(x_i) - y_i$
    $E_j \leftarrow \text{predict}(x_j) - y_j$
    $result \leftarrow \alpha'_j + y_j * (E_i - E_j)/k_{ij}$
    $result \leftarrow \max(\alpha[j], L)$
    $result \leftarrow \min(\alpha[j], H)$
    $\alpha[j] \leftarrow result$
    $\alpha[i] \leftarrow \alpha'_i + y_i * y_j * (\alpha'_j - \alpha[j])$
  end for
  check for convergence
  if normalize$(\alpha - \alpha' < \epsilon)$ **then**
    break
  end if
  final result for b and w
  $b \leftarrow \text{mean}(y - X^T \cdot W^T)$
  if kernel type = linear **then**
    $w \leftarrow X^T \cdot \alpha \times y$
  end if
end while
return w, b

TABLE II
SVM RESULT

	time cost(s)	mean error
SGD1	8.3	0.008
SGD2	5.3	0.012
SMO1	56.2	0
SMO2	9.4	0.24

E. Conclusion

Those two different ways have their own pros and cons. The advantages for SGD are its efficiency and ease of implementation while disadvantages are its need for many hyperparameters and iterations as well as sensitivity for feature scaling. [2]. SMO can easily introduce different types of kernel functions and it can be much more robust for bad training data. However, it still cost a huge amount of time to train.

References

[1] Y. Zhao, "Artificial intelligence lab: SVM," 2018.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[3] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," p. 21, April 1998.