Support Vector Machine Report

Peijun Ruan 11610214

School of Computer Science and Engineering
Southern University of Science and Technology
Email: 11610214@mail.sustc.edu.cn

1. Preliminaries

1.1. Introduction

In machine learning,support vector machines(**SVM**) are supervised learning models that using algorithm to analyze data used for classification and regression analysis.Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other.[1]

1.2. Application

SVM can be used to solve various real world problems:

- SVM can be used in text and hypertext categorization.
- Hand-written characters can be recognized using SVM.
  etc.

In this project, we will be given a train set and a test set and a time limit, our goal is to train a model in limited time and print the classification result of test set.

2. Methodology

2.1. Model Design

There are many method to train a support vector machine model.One algorithm is Pegasos which using the idea of sub gradient descent, another one is Sequential minimal optimization(SMO).In this project, I implement both of these two algorithm.

For Pegasos,each loop we randomly choose a sample for gradient descent,and for certain number of loops,we get the final result[3].

For SMO,each loop we choose a pair of alpha and update them until the iteration times larger than setting value or the model convergence[2].

2.2. Representation

2.2.1. Notation.

- Pegasos
  - S: the data set
  - λ: the parameter for gradient calculation
- SMO
  - α: the vector of lagrangian multiplier
  - T: iteration times
  - error: the list of error of all training samples
  - kernel(K): the kernel trick used in algorithm
  - conv: the convergence judgment

2.2.2. Data Structure. All the data will first store in list and then transfer into numpy array for further calculation convenience.All the vectors during the algorithm training are also numpy.array type.

2.3. Functions

Here show the main functions I implement in this project.

- SMO: the implementation of SMO algorithm
- Pegasos: the implementation of Pegasos algorithm
- predict: the function that predicts result for given data set
- call_e: calculate the error of a single sample
- kernel: the kernel function that map (x,y) to ϕ(x,y)
- load_data: loading data from the given path.
- cal_L_H: calculate the up and low border of parameter
- call_w: calculate the w vector for model

2.4. Detail of Algorithm

Here shows the psudocode of SMO and Pegasos.

Algorithm 1 SMO

```
Input: data set $S$,data labels $L$,kernel $K$,iteration times $T$,convergence judgment $E$
Output: $w, b$
  $\alpha$ ←zero vector
  $itercount, b \leftarrow 0$
  while itercount < T do
      $itercount \leftarrow itercount + 1$
      $\alpha_{temp} \leftarrow \alpha$
      for i in range(S.shape[0]) do
          $error_i \leftarrow$ call_e(i)
          if sample i does not obey KKT condition then
              j← randint(i,S.shape[0])
              $error_j \leftarrow$ call_e(j)
              $\alpha_i, \alpha_j \leftarrow \alpha[i], \alpha[j]$
              L,H ← cal_L_H
              if L==H then
                  continue
              end if
              $\theta \leftarrow$ 2*K(i,j)-K(i,i)-K(j,j)
              $\alpha[j] -= L[j] * (error_i - error_j)/\theta$
              if $\alpha[j] > H$ then
                  $\alpha[j] \leftarrow H$
              else if $\alpha[j] < L$ then
                  $\alpha[j] \leftarrow L$
              end if
              $\alpha[i] += L[i] * L[j] * (\alpha_j - \alpha[j])$
              update $b$
          end if
      end for
      if $\alpha - \alpha_{temp} < E$ then
          break
      end if
  end while
  $w \leftarrow$ call_w return $w, b$
```

Algorithm 2 Pegasos

```
Input: data set $S$,data labels $L$,$\lambda$
Output: $w, b$
  $t \leftarrow 0$
  $w \leftarrow zero\ vector$
  for i in range(100) do
      for j in range($S$.shape[0]) do
          $t \leftarrow t + 1$
          $\theta \leftarrow$ 1/(λ*t)
          p ← w*S[j]
          if L[j]*p¡1 then w←w-$\theta$*(λ*w-L[j]*S[j])
          elsew←w-$\theta$*λ*w
          end if
      end for
  end for
  b←calculate b return $w, b$
```

3. Empirical Verification

3.1. Design

Since I implement two algorithms to train the SVM model, we can easily compare the accuracy of the final result and the training time between two models.

Here we just simply define the accuracy as the number of correct marks over the number of all test samples.

3.2. Data

We are only given one data set but we need a train set and a test set,so we can divide the total set into proper two part,one for training and the other for testing. I choose 90% of data for training and rest for testing.

3.3. Hyperparameters

There are many hyperparameters you need to pay attention to.Here show the strategy I choose the value of them.

- SMO:
  - C :the tolerance of soft margin, default value is 1.0
  - $Kernel$:the kernel trick used in calculation, default using RBF kernel
  - $error$:the lower border of error that exit the loop, default value is 0.001
  - σ :the radius length of RBF kernel, default value is 1
  - $degree$:the degree of polynomial kernel, default value is 2(quadratic kernel)
  - T :the maximum iteration times to finish training, default value is 1000
- Pegasos:
  - λ :the parameter for gradient calculation. default value is 2

3.4. Test Environment

CPU:i5-7300HQ 2.50GHz
RAM:8.0GB
Number of Process: only one process

3.5. Performance

To compare the performance, we just compare the accuracy and training time.I use the default value for all the hyperparameters at first and then change the hyperparameters to check the changes.

3.5.1. Default.

For training time cost: Pegasos costs 3.26s and SMO costs 1.56s

For accuracy:Pegasos 98% SMO 94%

3.5.2. Hyperparameters Changes.

$\sigma = 2, C = 5, \lambda = 3$:

For training time cost: Pegasos costs 3.26s and SMO costs 2.4s

For accuracy:Pegasos 99.5% SMO 95.5%

$\sigma = 1, C = 5, \lambda = 3$:

For training time cost: Pegasos costs 3.13s and SMO costs 3s

For accuracy:Pegasos 98% SMO 94.5%

$\sigma = 2, C = 1, \lambda = 3$:

For training time cost: Pegasos costs 3s and SMO costs 1.27s

For accuracy:Pegasos 98% SMO 97.5%

$\sigma = 4, C = 1, \lambda = 10$:

For training time cost: Pegasos costs 3.4s and SMO costs 1.46s

For accuracy:Pegasos 99.5% SMO 95%

3.6. Analysis

From the simple compare we find that the speed of SMO is faster than Pegasos.If I change the C to 5,the time cost of SMO become larger,and that's because C determine how many error samples model can tolerate during training, the bigger C is, the more samples SMO has to deal with,which result for the increasing training time.

Besides, the accuracy of SMO and Pegasos changes due to the changes of hyperparameters.For Pegasos.We notice that $\lambda = 3$ gives a better result than $\lambda = 1$.If we adjust λ to 10, the accuracy is higher and stable(because the result has fluctuation).For SMO,we may find that if we increase the C,the accuracy may be lower than C=1,and $\sigma = 2$ has a best accuracy among $\sigma = 1, 2, 4$.So it does not mean the larger σ is,the better result is,finding a proper σ is important.

However,even Pegasos has a faster training speed, I did not add kernel trick for it, it can not deal with some complex data sets which is not a linear classification set well.

References

[1] Support vector machine. (2018). Retrieved from https://en.wikipedia.org/wiki/Support_vector_machine

[2] 序列最小优化算法（SMO）浅析. (2018). Retrieved from https://www.jianshu.com/p/eef51f939ace

[3] svm随机次梯度下降算法-pegasos - shiqi,bao的博客- CSDN博客. (2018). Retrieved from https://blog.csdn.net/sinat_27612639/article/details/70037499