1. Define variables before you use them [5 points]

2. Primal and dual forms of hard and soft margin SVMs [20 points]

3. Concept of support vectors (two types) [10 points]

4. Discussions on why max margin is good [5 points]

5. Experiments

(a) compare your w, b obtained via solving the primal problem, with the w, b reconstructed by the dual variables obtained via solving the dual problem (both the results and the reconstruction formulation) [5 points]

(b) check duality gap of yours (both the result and the formulation) [5 points]

(c) compare your w, b, α with those of libsvm [5 points]

(d) compare training and testing errors of your code and libsvm [5 points]

(e) code [40 points]

1 INTRODUCTION

Support Vector Machines (SVMs) have attracted growing interests in statistical machine learning, because of its simple theoretical concepts and high performance in practical application. SMVs can be applied in handwriting digits recognition, 3D objects detection, facial recognition, text categorization and many other pattern recognition areas. For datasets with certain features, SVMs have demonstrated superior performance than other algorithms, such as small data volume, non-linear, or even high dimensional space. The SV classifiers are based on the construction of hyperplanes, where class decisions are made upon the output of threshold function T(r), for the given set of training samples *xi∈Xn, μi∈θn, i=1...M, S = μ1 x1 +μ2x2 +μ3x3+……+ μ0*.  We hope to find the decision function *f(x; θ)* to evaluate the dot product of *xТ* and *θ where r=θ.dot(xТ) .* So the class decision can be made for  *if r>0; or when .* In this assignment we start from a simplified version of SVM where the objective function is constructed as: y=wx+b.

The optimal separating plane predicts data labels by calculating wx+b. The model construction begins by assuming that separates all data points of one class from those of the other class. The best hyperplane for an SVM means the one with the largest margin between the two classes. Margin means the maximal width of the slab parallel to the hyperplane that has no interior data points.

The support vectors are the data points that are closest to the separating hyperplane; these points are on the boundary of the slab. The following figure illustrates these definitions, with + indicating data points of type 1, and – indicating data points of type –1.

A close up of a person

Description automatically generated

A close up of a map

Description automatically generated

For hard-margin SVMs, the goal is to minimize the function distances of with respect to the constraint variables in S. This can be expressed as: min(wTw), subject to y∙(wx + b)-1 >= 0. Whereas in soft-margin SVMs, we reconstructed w, b by introducing the lagrangian factor C and xi, so as to minimize the relaxed distance margin of min(wTw + C∙sum(xi)).

2 EXPERIMENTS

After applying both hard and soft-margin SVM algorithm upon the libsvm datasets a2a.train and a2a.test, we have obtained some test results and compare the prediction outcomes between two methods. Method 1 is the libsvm built-in nu-SVC algorithm and Method 2 is our own cvx algorithm codes. We listed both hyperparameters test results in the following table and address their performance advantages and disadvantages.

|  |  |  |
| --- | --- | --- |
| Method | Nu-SVC | CVX |
| System Performance | optimization finished, #iter = 796  nu = 0.461826  obj = -971.667542, rho = 0.585499  nSV = 1068, nBSV = 1028  Total nSV = 1068 | number of iterations = 10  residual of dual infeasibility  certificate X = 8.01e-18  reldist to infeas. <= 2.70e-17  Total CPU time (secs) = 0.29  CPU time per iteration = 0.03  termination code = 2  DIMACS: 0.0e+00 0.0e+00 4.7e+00 0.0e+00 -1.0e+00 4.4e-02 |
| Accuracy | Accuracy = 83.9781% (25442/30296) (classification) |  |
| w ; b values  Hard margin | w = 0.461826; b =0.585499 | w=0; b=0.0044; |
| w ; b values  Soft margin | w = 0.461826; b =0.585499 | w=0; b=0.0044; |
| C, xi | N/A | C=0.1, xi=-0.0044 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 862 X 2  C4L | C | w | b | xi | Iterations | Pri\_obs | Dual\_obs | gap | cputpi | cvx\_optval |
|  | 1 | 1.0e-06 \*  0.9045  -0.3245 | -0.2958 | 1.0e-11 \*  0.6347 | 25 | 5.47571986e-09 | -7.34545753e-09 | 1.35e-08 | 0.08 | +5.47572e-09 |
|  | 0.1 | 1.0e-06 \*  0.6507  -0.2650 | -0.2944 | 1.0e-10\*  0.4719 | 29 | 4.07053680e-09 | -5.54772878e-09 | 9.73e-09 | 0.08 | +4.07054e-09 |
|  | 0.01 | 1.0e-06 \*  0.6442  -0.2381 | -0.2928 | 1.0e-09 \*  0.3920 | 26 | 3.38128954e-09 | -4.63421704e-09 | 8.07e-09 | 0.08 | +3.38129e-09 |
|  | 0.001 | 1.0e-06 \*  0.5381  -0.1926 | -0.2921 | 1.0e-08 \*  0.2899 | 28 | 2.50064823e-09 | -3.40495541e-09 | 5.95e-09 | 0.08 | +2.50065e-09 |

APPENDIX A

REFERENCES

Step1: understand the difference of Libsvm and cvxopt in terms of its variables, models, and applications

Reference 1:

https://au.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html

Reference 2:

http://cvxopt.org/userguide/coneprog.html#quadratic-programming

Reference 3:

https://www.m-asim.com/2018/10/19/how-to-compare-machine-learning-algorithms-in-python-with-scikit-learn/

Mathematical Formulation: Primal. This discussion follows Hastie, Tibshirani, and Friedman [1] and Christianini and Shawe-Taylor [2].

The data for training is a set of points (vectors) xj along with their categories yj. For some dimension d, the xj ∊ Rd, and the yj = ±1. The equation of a hyperplane is

f(x)=x′β+b=0

where β ∊ Rd and b is a real number.

The following problem defines the best separating hyperplane (i.e., the decision boundary). Find β and b that minimize ||β|| such that for all data points (xj,yj),

y

j

f(x

j

)≥1.

The support vectors are the xj on the boundary, those for which y

j

f(x

j

)=1.

For mathematical convenience, the problem is usually given as the equivalent problem of minimizing β. This is a quadratic programming problem. The optimal solution (

ˆ

β

,

ˆ

b

) enables classification of a vector z as follows:

class(z)=sign(z′

ˆ

β

+

ˆ

b

)=sign(

ˆ

f

(z)).

ˆ

f

(z) is the classification score and represents the distance z is from the decision boundary.

Mathematical Formulation: Dual. It is computationally simpler to solve the dual quadratic programming problem. To obtain the dual, take positive Lagrange multipliers αj multiplied by each constraint, and subtract from the objective function:

L

P

=

1

2

β′β−



j

α

j

(y

j

(x

j

′β+b)−1),

where you look for a stationary point of LP over β and b. Setting the gradient of LP to 0, you get

β

0

=



j

α

j

y

j

x

j

=



j

α

j

y

j

.

(1)

Substituting into LP, you get the dual LD:

L

D

=



j

α

j

−

1

2



j



k

α

j

α

k

y

j

y

k

x

j

′x

k

,

which you maximize over αj ≥ 0. In general, many αj are 0 at the maximum. The nonzero αj in the solution to the dual problem define the hyperplane, as seen in Equation 1, which gives β as the sum of αjyjxj. The data points xj corresponding to nonzero αj are the support vectors.

The derivative of LD with respect to a nonzero αj is 0 at an optimum. This gives

y

j

f(x

j

)−1=0.

In particular, this gives the value of b at the solution, by taking any j with nonzero αj.

The dual is a standard quadratic programming problem. For example, the Optimization Toolbox™ quadprog solver solves this type of problem.

Nonseparable Data

Your data might not allow for a separating hyperplane. In that case, SVM can use a soft margin, meaning a hyperplane that separates many, but not all data points.

There are two standard formulations of soft margins. Both involve adding slack variables ξj and a penalty parameter C.

The L1-norm problem is:

The L1-norm refers to using ξj as slack variables instead of their squares. The three solver options SMO, ISDA, and L1QP of fitcsvm minimize the L1-norm problem.

The L2-norm problem is:

subject to the same constraints.

In these formulations, you can see that increasing C places more weight on the slack variables ξj, meaning the optimization attempts to make a stricter separation between classes. Equivalently, reducing C towards 0 makes misclassification less important.

Mathematical Formulation: Dual. For easier calculations, consider the L1 dual problem to this soft-margin formulation. Using Lagrange multipliers μj, the function to minimize for the L1-norm problem is:

where you look for a stationary point of LP over β, b, and positive ξj. Setting the gradient of LP to 0, you get

The final set of inequalities, 0 ≤ αj ≤ C, shows why C is sometimes called a box constraint. C keeps the allowable values of the Lagrange multipliers αj in a “box”, a bounded region.

The gradient equation for b gives the solution b in terms of the set of nonzero αj, which correspond to the support vectors.

You can write and solve the dual of the L2-norm problem in an analogous manner. For details, see Christianini and Shawe-Taylor [2], Chapter 6.

fitcsvm Implementation. Both dual soft-margin problems are quadratic programming problems. Internally, fitcsvm has several different algorithms for solving the problems.

For one-class or binary classification, if you do not set a fraction of expected outliers in the data (see OutlierFraction), then the default solver is Sequential Minimal Optimization (SMO). SMO minimizes the one-norm problem by a series of two-point minimizations. During optimization, SMO respects the linear constraint



=0, and explicitly includes the bias term in the model. SMO is relatively fast. For more details on SMO, see [3].

For binary classification, if you set a fraction of expected outliers in the data, then the default solver is the Iterative Single Data Algorithm. Like SMO, ISDA solves the one-norm problem. Unlike SMO, ISDA minimizes by a series on one-point minimizations, does not respect the linear constraint, and does not explicitly include the bias term in the model. For more details on ISDA, see [4].

For one-class or binary classification, and if you have an Optimization Toolbox license, you can choose to use quadprog to solve the one-norm problem. quadprog uses a good deal of memory, but solves quadratic programs to a high degree of precision. For more details, see Quadratic Programming Definition (Optimization Toolbox).

For pattern recognition we first generate linearly separable, 2-class data using 2-dimensional Gaussians. We hope to find a hyperplane which can separate the datasets as below and we hope to maximise the margin as large as possible.

To define SVM concepts mathematically, we introduce the data plot:

{z\_c,t\_c},c=1,....m, and z\_c in D^r, t\_c in {-1,+1}

To find the hyperplane f(x)=w^Tx-b so that when f(x)<0, y\_i=-1, and when f(x)>0, y\_i=1. The decision functions can be established by calculating f(x) for the training data and labeling new examples (x,y) with the value of f(x).

For very large data sets a feasible approach is to randomly choose a subset of the data set, conduct grid-search on them, and then do a better-region-only grid-search on the complete data set.

\*guide.pdf p.8\*

All SVM formulations supported in LIBSVM are quadratic minimization problems.

LIBSVR

\*libsvm.pdf p.2\*

Two implementation techniques to reduce the running time for minimizing SVM quadratic problems: shrinking and caching.

\*libsvm.pdf p.2\*

ν-support vector regression (ν-SVR)

All LIBSVM’s training and testing algorithms are implemented in the ﬁle svm.cpp. The two main sub-routines are svm train and svm predict. The training procedure is more sophisticated, so we give the code organization in Figure 1. From Figure 1, for classiﬁcation, svm train decouples a multi-class problem to two-class problems (see Section 7) and calls svm train one several times. For regression and one-class SVM, it directly calls svm train one. The probability outputs for classiﬁcation and regression are also handled in svm train. Then, according to the SVM formulation, svm train one calls a corresponding sub-routine such as solve c svc for C-SVC and solve nu svc for ν-SVC. All solve \* sub-routines call the solver Solve after preparing suitable input values. The sub-routine Solve minimizes a general form of SVM optimization problems; see (11) and (22). Details of the sub-routine Solve are described in Sections 4-6.

\*libsvm.pdf p.4\*

Comparison of standard forms, dual problems, and optimal solutions for SVM formulations

Summary of SVM formulations in LIBSVM

Ref1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|Ref2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|Ref3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|Ref4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|Ref5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

C-Support Vector Classiﬁcation\_\_\_|ν-Support Vector Classiﬁcation\_\_\_\_\_\_\_|Distribution Estimation (One-class SVM)\_\_\_\_\_\_\_\_\_\_|\epsilon-Support Vector Regression (\epsilon-SVR)|ν-Support Vector Regression (ν-SVR)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

\*libsvm.pdf p.3-7\*

Performance Measures: Accuracy, MSE (mean squared error) and r2 (squared correlation coeﬃcient).

All LIBSVM’s training and testing algorithms are implemented in the ﬁle svm.cpp.

The two main sub-routines are svm train and svm predict.

Then, according to the SVM formulation, svm train one calls a corresponding sub-routine such as solve c svc for C-SVC and solve nu svc for ν-SVC.

The sub-routine Solve minimizes a general form of SVM optimization problems; see (11) and (22). Details of the sub-routine Solve are described in Sections 4-6.

optimization for one linear constraint: C-SVC, -SVR, and One-class SVM

\*libsvm.pdf p.10-15\*

optimization for two linear constraint: νSVC and ν-SVR

\*libsvm.pdf p.15\*

A.1 Astroparticle Physics • Original sets with default parameters

$ ./svm-train svmguide1 $ ./svm-predict svmguide1.t svmguide1.model svmguide1.t.predict → Accuracy = 66.925% • Scaled sets with default parameters

$ ./svm-scale -l -1 -u 1 -s range1 svmguide1 > svmguide1.scale $ ./svm-scale -r range1 svmguide1.t > svmguide1.t.scale $ ./svm-train svmguide1.scale $ ./svm-predict svmguide1.t.scale svmguide1.scale.model svmguide1.t.predict → Accuracy = 96.15% • Scaled sets with parameter selection (change to the directory tools, which contains grid.py)

$ python grid.py svmguide1.scale ··· 2.0 2.0 96.8922

(Best C=2.0, γ=2.0 with ﬁve-fold cross-validation rate=96.8922%)

$ ./svm-train -c 2 -g 2 svmguide1.scale $ ./svm-predict svmguide1.t.scale svmguide1.scale.model svmguide1.t.predict → Accuracy = 96.875% • Using an automatic script

$ python easy.py svmguide1 svmguide1.t Scaling training data... Cross validation...

9

Best c=2.0, g=2.0 Training... Scaling testing data... Testing... Accuracy = 96.875% (3875/4000) (classification)

Using the same scaling factors for training and testing sets, we obtain much better accuracy.

$ ../svm-scale -l 0 -s range4 svmguide4 > svmguide4.scale $ ../svm-scale -r range4 svmguide4.t > svmguide4.t.scale $ python easy.py svmguide4.scale svmguide4.t.scale Accuracy = 89.4231% (279/312) (classification)

• RBF kernel with parameter selection

$ cat leu leu.t > leu.combined $ python grid.py leu.combined ··· 8.0 3.0517578125e-05 97.2222

(Best C=8.0, γ = 0.000030518 with ﬁve-fold cross-validation rate=97.2222%)

• Linear kernel with parameter selection

$ python grid.py -log2c -1,2,1 -log2g 1,1,1 -t 0 leu.combined ··· 0.5 2.0 98.6111

(Best C=0.5 with ﬁve-fold cross-validation rate=98.61111%)

Though grid.py was designed for the RBF kernel, the above way checks various C using the linear kernel (-log2g 1,1,1 sets a dummy γ).

LIBLINEAR is eﬃcient for large-scale document classiﬁcation. Let us consider a large set rcv1 test.binary with 677,399 instances.

$ time liblinear-1.21/train -c 0.25 -v 5 rcv1\_test.binary Cross Validation Accuracy = 97.8538% 68.84s

Consider the data http://www.csie. ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/covtype.libsvm.binary.scale. bz2. The number of instances 581,012 is much larger than the number of features 54. We run LIBLINEAR with -s 1 (default) and -s 2.

$ time liblinear-1.21/train -c 4 -v 5 -s 2 covtype.libsvm.binary.scale Cross Validation Accuracy = 75.67% 67.224s $ time liblinear-1.21/train -c 4 -v 5 -s 1 covtype.libsvm.binary.scale

14

Cross Validation Accuracy = 75.6711% 452.736s

Clearly, using -s 2 leads to shorter training time.

We have shown that the KKT condition of problem (22) implies Eqs. (25) and (26) according to yi = 1 and −1, respectively.

Now we consider the case of yi = 1. If there exists αi such that 0 < αi < C, then we obtain r1 = ∇if(α). In LIBSVM, for numerical stability, we average these values.

\*guide.pdf p.8-14\*

We found that if the number of iterations is large, then shrinking can shorten the training time. However, if we loosely solve the optimization problem (e.g., by using a large stopping tolerance ), the code without using shrinking may be much faster. In this situation, because of the small number of iterations, the time spent on all decomposition iterations can be even less than one single gradient reconstruction.

\*libsvm.pdf p.25\*

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>svm-train.exe a2a.train

\*

optimization finished, #iter = 796

nu = 0.461826

obj = -971.667542, rho = 0.585499

nSV = 1068, nBSV = 1028

Total nSV = 1068

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>dir

驱动器 C 中的卷是 OS

卷的序列号是 F27A-BA5A

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows 的目录

2019/08/22 11:19 <DIR> .

2019/08/22 11:19 <DIR> ..

2019/08/22 10:45 2,167,822 a2a.test

2019/08/22 10:43 162,053 a2a.train

2019/08/22 11:19 77,993 a2a.train.model

2019/08/22 11:04 258,048 libsvm.dll

2019/08/22 11:04 14,336 libsvmread.mexw64

2019/08/22 11:03 13,312 libsvmwrite.mexw64

2019/08/22 11:04 211,968 svm-predict.exe

2019/08/22 11:04 165,376 svm-scale.exe

2019/08/22 11:03 226,816 svm-toy.exe

2019/08/22 11:04 247,808 svm-train.exe

2019/08/22 11:04 28,160 svmpredict.mexw64

2019/08/22 11:04 68,608 svmtrain.mexw64

12 个文件 3,642,300 字节

2 个目录 423,082,975,232 可用字节

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>svm-predict.exe

Usage: svm-predict [options] test\_file model\_file output\_file

options:

-b probability\_estimates: whether to predict probability estimates, 0 or 1 (default 0); for one-class SVM only 0 is supported

-q : quiet mode (no outputs)

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>svm-predict.exe a2a.test a2a.train.model a2a.out

Accuracy = 83.9781% (25442/30296) (classification)

We can divide this process broadly into 4 stages. Each stage requires a certain amount of time to execute:

Loading and pre-processing Data – 30% time

Defining Model architecture – 10% time

Training the model – 50% time

Estimation of performance – 10% time

Soft Margin Binary SVM

- The primal form and dual form for both soft and hard margin SVM

- Concept of support vectors

- Why max margin is good

- Concepts of generalisation/test error

- Experimental results including comparison between your implementation and libsvm

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>svm-train.exe a2a.train

\*

optimization finished, #iter = 796

nu = 0.461826

obj = -971.667542, rho = 0.585499

nSV = 1068, nBSV = 1028

Total nSV = 1068

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>dir

驱动器 C 中的卷是 OS

卷的序列号是 F27A-BA5A

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows 的目录

2019/08/22 11:19 <DIR> .

2019/08/22 11:19 <DIR> ..

2019/08/22 10:45 2,167,822 a2a.test

2019/08/22 10:43 162,053 a2a.train

2019/08/22 11:19 77,993 a2a.train.model

2019/08/22 11:04 258,048 libsvm.dll

2019/08/22 11:04 14,336 libsvmread.mexw64

2019/08/22 11:03 13,312 libsvmwrite.mexw64

2019/08/22 11:04 211,968 svm-predict.exe

2019/08/22 11:04 165,376 svm-scale.exe

2019/08/22 11:03 226,816 svm-toy.exe

2019/08/22 11:04 247,808 svm-train.exe

2019/08/22 11:04 28,160 svmpredict.mexw64

2019/08/22 11:04 68,608 svmtrain.mexw64

12 个文件 3,642,300 字节

2 个目录 423,082,975,232 可用字节

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>svm-predict.exe

Usage: svm-predict [options] test\_file model\_file output\_file

options:

-b probability\_estimates: whether to predict probability estimates, 0 or 1 (default 0); for one-class SVM only 0 is supported

-q : quiet mode (no outputs)

C:\Users\wendy\Desktop\libsvm3.2.3\libsvm-3.23\windows>svm-predict.exe a2a.test a2a.train.model a2a.out

Accuracy = 83.9781% (25442/30296) (classification)

The SVM should maximize the distance between the two decision boundaries. Mathematically, this means we want to maximize the distance between the hyperplane defined by wTx+b=−1 and the hyperplane defined by wTx+b=1. This distance is equal to 2∥w∥. This means we want to solve maxw2∥w∥. Equivalently we want minw∥w∥2.

The SVM should also correctly classify all x(i), which means y(i)(wTx(i)+b)≥1,∀i∈{1,…,N}

Which leads us to the following quadratic optimization problem:

minw,bs.t.∥w∥2,y(i)(wTx(i)+b)≥1∀i∈{1,…,N}

This is the hard-margin SVM, as this quadratic optimization problem admits a solution iff the data is linearly separable.

One can relax the constraints by introducing so-called slack variables ξ(i). Note that each sample of the training set has its own slack variable. This gives us the following quadratic optimization problem:

minw,bs.t.∥w∥2+C∑i=1Nξ(i),y(i)(wTx(i)+b)≥1−ξ(i),ξ(i)≥0,∀i∈{1,…,N}∀i∈{1,…,N}

This is the soft-margin SVM. C is a hyperparameter called penalty of the error term. (What is the influence of C in SVMs with linear kernel? and Which search range for determining SVM optimal parameters?).

One can add even more flexibility by introducing a function ϕ that maps the original feature space to a higher dimensional feature space. This allows non-linear decision boundaries. The quadratic optimization problem becomes:

minw,bs.t.∥w∥2+C∑i=1Nξ(i),y(i)(wTϕ(x(i))+b)≥1−ξ(i),ξ(i)≥0,∀i∈{1,…,N}∀i∈{1,…,N}

Optimization

The quadratic optimization problem can be transformed into another optimization problem named the Lagrangian dual problem (the previous problem is called the primal):

maxαs.t.minw,b∥w∥2+C∑i=1Nα(i)(1−wTϕ(x(i))+b)),0≤α(i)≤C,∀i∈{1,…,N}

This optimization problem can be simplified (by setting some gradients to 0) to:

maxαs.t.∑i=1Nα(i)−∑i=1N∑j=1N(y(i)α(i)ϕ(x(i))Tϕ(x(j))y(j)α(j)),0≤α(i)≤C,∀i∈{1,…,N}

w doesn't appear as w=∑Ni=1α(i)y(i)ϕ(x(i)) (as stated by the representer theorem).

We therefore learn the α(i) using the (x(i),y(i)) of the training set.

(FYI: Why bother with the dual problem when fitting SVM? short answer: faster computation + allows to use the kernel trick, though there exist some good methods to train SVM in the primal e.g. see {1})

Making a prediction

Once the α(i) are learned, one can predict the class of a new sample with the feature vector xtest as follows:

ytest=sign(wTϕ(xtest)+b)=sign(∑i=1Nα(i)y(i)ϕ(x(i))Tϕ(xtest)+b)

The summation ∑Ni=1 could seem overwhelming, since it means one has to sum over all the training samples, but the vast majority of α(i) are 0 (see Why are the Lagrange multipliers sparse for SVMs?) so in practice it isn't an issue. (note that one can construct special cases where all α(i)>0.) α(i)=0 iff x(i) is a support vector. The illustration above has 3 support vectors.

Kernel trick

One can observe that the optimization problem uses the ϕ(x(i)) only in the inner product ϕ(x(i))Tϕ(x(j)). The function that maps (x(i),x(j)) to the inner product ϕ(x(i))Tϕ(x(j)) is called a kernel, a.k.a. kernel function, often denoted by k.

One can choose k so that the inner product is efficient to compute. This allows to use a potentially high feature space at a low computational cost. That is called the kernel trick. For a kernel function to be valid, i.e. usable with the kernel trick, it should satisfy two key properties. There exist many kernel functions to choose from. As a side note, the kernel trick may be applied to other machine learning models, in which case they are referred as kernelized.

Suppose we have training vectors Xi in real number R^l ,

with the constraints that numbers can be separable with a linear classifier for a binary classification problem(y are the labels, x are the features in R, i=1, 2, 3,...l).

We write our classifier as g(x)=y\_i(W^T\*Xi + b) and this produces either 1 or -1 to represent which side of the hyperplane does X falls into.

The value of objective function f(x)= W^T\*Xi + b could be :

W^T\*Xi + b>=0, then g(x)=1 (1)

W^T\*Xi + b<0, then g(x)=-1 (2)

The functional margin is then defined as to find the optimised slope and intercept of (w,b) so that the prediction errors are minimal.

Optimised separating hyperplane (OSH) can be obtained from the middle point of two parallel support hyperplanes,

which generate maximum margin in dividing the two classes of features. As demonstrated in the figure below:

We can also visualise the support hyperplanes as pushing two wood boards from OSH to each side until one nearest feature lands on the hyperplane, and making sure

as many features are correctly labeled.

The distance between separating hyperplane and two support hyperplane can be written as:

d = (||b+1|-|b||)/||w||=1/||w||, if b<=-1, or b>=0; (4)

d = (||b+1|+|b||)/||w||=1/||w||, if -1<b<0. (5)

margin = 2d = 2/||w||, therefore minimising w can create maximum margin.

Each side of equation (1) and (2) still exist when multiplying a constant,

therefore infinite groups of anwers may satisfy the conditions of (1) and (2).

To simplify the hypothesis, we times f(x) with g(x) and thus combine (1) and (2) to the following equation:

y\_i(W^T\*Xi + b)-1 >= 0, for yi=1, i=1,2,3,...,m (3)

To solve the primal optimization problem, we consider introducing Lagrange multipliers alpha for optimising w and b together as a joint set.

If we define (3) as g\_i(\theta)<=0, the primal optimum problem is turned into :

min\_{w,b}||w||^2/2 =f({\theta}) (6)

s.t. y\_i(W^T\*Xi + b)-1 >= C{\sum\_{i=1}^{n}}{\kesi}, i=1,2,3,...,m (7)

{\kesi} \geq or

\ge 0, i=1,2,...,l (8)

We can convert the above equation using Lagrange Multiplier Method, to find the w\*, alpha\*, beta\* that can minimise the L values:

when {y}\_i({w}^T{x}\_i+b)-1=0, {a}\_i>=0;(vectors on the support hyperplanes)

when {y}\_i({w}^T{x}\_i+b)-1>0, {a}\_i=0; (points away from the OSH zone)

To find minimum L value, we have summarized the Lagrange multiplier condition as:

w\*=Sum{1->N}{a}\_i{y}\_i{x}\_i

Sum{1->N}{a}\_i{y}\_i=0

{y}\_i({w}^T{x}\_i+b)>=1

Lagrange multiplier condition:

{a}\_i>=0

Complementary slackness

{a}\_i[{y}\_i({w}^T{x}\_i+b)-1]=0

The points that sit on the support hyperplanes are defined as support vectors.

when {y}\_i({w}^T{x}\_i+b)-1=0, {a}\_i=0;

After confirming the support vectors, we can further infer the classification of the new points with the following:

f(x)=({w}^T{x}\_i+b)=Sum{1->i}{a}\_i{y}\_i{x}\_i^Tx+b

b= {y}\_i-Sum{1->j}{a}\_j{y}\_j{x}\_j^T{x}\_i

比较一下LIBSVM在python和commandline以及matlab之间的差异并说明为什么

Compare the running results between python, windows commandline and matlab and explain why

Dataset in use should be MNIST or easy to use ML library

The goal of this assignment is to learn how to train SVM classifier on image data wth use of SVM from

We hope to minimize the Lagrangian factor \kesi towards 0 so that the precondition of g\_i(theta) <=0 can be satisfied.

When g\_i(theta) >0, the objective functional becomes non-convex and the optimisation problem cannot be resolved.

Please note the optimization problem has a convex quadratic objective and only linear constraints expressed as algebraic equations.

In real scenarios it is challenging to derive an efficient algorithm and solve the problem in very high dimensional spaces.

We can transform the above equations (6), (7), (8) through applying Lagrange multipliers method in (9). To find partial derivative of (6) + (7)

so that \frac {\vartheta f}{\vartheta x}= \frac {\vartheta f}{\vartheta y}=0. The minimum value of (6) + (7) can then be calculated when (9)=0.

h\_i({\theta})=L({\theta}, b, {\kesi} )=(6) +(7)=0 (9)

explained as to build separating hyperplane and minimise its errors with regularisation factor:

min(w, b, )

Draft\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_AB

Assignment 1

Support Vector Machine

This assignment attempts to explain basic principles for implementation of both hard and soft margin SVM algorithm in Python using the CVXOPT library, and comparing those with

the results from libsvms. We derive notations and inputs for the API from vectorised matrix, through formulations of the primal and dual optimization problems.

Background

SVM algorithms construct a set of hyperplanes for classification or regression tasks. A number of hyperplanes may satisfy the separation task therefore the best hyperplane

represents the largest margin between different classes of datasets. SVMs focus only on the points that are most hard to tell apart, whereas other classifiers may compute all the points.

Firstly SVMs calculate the distances between vectors and try to find two vectors that are nearest to each other. Through vector subtraction A-B,

a perpendicular plane is constructed to bisect AB as the optimal separating hyperplane.

In a typical linear two-class problem, separation task can be solved by building linear classifier g({w^T}x\_{i}+b)= H\_{i}{g} with labels y and features x. g(z)=1 if z>=0, and g(z)=-1 otherwise.

y belongs to {-1,1} and "w, b" notation represents the slope and intercept terms. i = 1,..., m.

Suppose we have training vectors Xi in real number R^l ,

with the constraints that numbers can be separable with a linear classifier for a binary classification problem(y are the labels, x are the features in R, i=1, 2, 3,...l).

We write our classifier as g(x)=y\_i(W^T\*Xi + b) and this produces either 1 or -1 to represent which side of the hyperplane does X falls into.

The value of objective function f(x)= W^T\*Xi + b could be :

W^T\*Xi + b>=0, then g(x)=1 (1)

W^T\*Xi + b<0, then g(x)=-1 (2)

The functional margin is then defined as to find the optimised slope and intercept of (w,b) so that the prediction errors are minimal.

Optimised separating hyperplane (OSH) can be obtained from the middle point of two parallel support hyperplanes,

which generate maximum margin in dividing the two classes of features. As demonstrated in the figure below:

The distance between separating hyperplane and two support hyperplane can be written as:

d = (||b+1|-|b||)/||w||=1/||w||, if b<=-1, or b>=0; (4)

d = (||b+1|+|b||)/||w||=1/||w||, if -1<b<0. (5)

margin = 2d = 2/||w||, therefore minimising w can create maximum margin.

Each side of equation (1) and (2) still exist when multiplying a constant,

therefore infinite groups of anwers may satisfy the conditions of (1) and (2).

To simplify the hypothesis, we times f(x) with g(x) and thus combine (1) and (2) to the following equation:

y\_i(W^T\*Xi + b)-1 >= 0, for yi=1, i=1,2,3,...,m (3)

The primal optimisation problem is to find the optimal margin classifer that:

min\_{w,b} 1/2||w||\_2,

s.t. g\_{i}(w)=-y^{i}({w^T}x\_{i}+b)+1<=0, i = 1,..., m.

To solve the primal optimization problem, we consider introducing Lagrange multipliers alpha for optimising w and b together as a joint set.

If we define (3) as g\_i(\theta)<=0, the primal dual optimum problem is turned into :

min\_{w,b}||w||^2/2 =f({\theta}) (6)

s.t. y\_i(W^T\*Xi + b)-1 >= C{\sum\_{i=1}^{n}}{\kesi}, i=1,2,3,...,m (7)

{\kesi} \geq or \ge 0, i=1,2,...,l (8)

We hope to minimize the Lagrangian factor \kesi towards 0 so that the precondition of g\_i(theta) <=0 can be satisfied.

When g\_i(theta) >0, the objective functional becomes non-convex and the optimisation problem cannot be resolved.

Please note the optimization problem has a convex quadratic objective and only linear constraints expressed as algebraic equations.

In real scenarios it is challenging to derive an efficient algorithm and solve the problem in very high dimensional spaces.

We can transform the above equations (6), (7), (8) through applying Lagrange multipliers method in (9). To find partial derivative of (6) + (7)

so that \frac {\vartheta f}{\vartheta x}= \frac {\vartheta f}{\vartheta y}=0. The minimum value of (6) + (7) can then be calculated when (9)=0.

h\_i({\theta})=L({\theta}, b, {\kesi} )=(6) +(7)=0 (9)

Soft Margin Problem

The hard margin problem can be expressed as :

Min 1/2 ||w||^2 ;

With constraints:

yi(WTXi + b) > =1 ;

Whereas the Soft Margin problem is defined as:

Min 1/2 ||w||^2 + C \* Sum(ξi) {i=1,…, N}

With constraints:

Yi(WTXi + b) >=1-αi ;

αi >=0

The C value in this equation and ξ are the training errors in

cvx\_begin

variables w(n);

variables b;

variables C;

variables α(n);

variables β;

variables ξ(n)

minimize(1/2\*norm(w) + C\*sum(αi));

subject to

(αi[yi(β+WTxi)-1+ξi]=0, w, b, C, α, ‘MaxDegree’, 6)

ξi》=0

C > = αi》=0

C= μi + αi

cvx\_end

<https://blog.csdn.net/ldj1208/article/details/51893293>

<https://blog.csdn.net/Dominic_S/article/details/83002153?utm_source=app>