Optimization Techniques - Project 2

Saad Ejaz - 323511

June 2022

Problem 15

Contents

1 Problem Statement

We are provided with a dataset on **Acute Inflammations** that needs to be classified using a binary SVM classifier. The data will be divided into training and testing sets. The training set will be divided in two parts, which will be used as follow:

1. A linear SVM classifier (using the Primal Method) will be trained on one set to linearly separate the data with a soft margin. This will result in weights for a hyperplane that divide the training data into two classes.

2. The second part will be used to train a non-linear hypersurface using the Kernel method. An RBF (Radial Basis Function) kernel will be used.

Both problems use quadratic programming from the optimization toolbox of MATLAB. The function used is `quadprog`, which solves the following quadratic optimization problem:

$$\min_x \frac{1}{2}x^T Hx + f^T x \text{ such that } \begin{cases} A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub \end{cases}$$

2 The Dataset

2.1 Description of Features and Output

The dataset[1] was created by a medical expert to test the expert system, which will perform the presumptive diagnosis of two diseases of the urinary system.
The dataset has 120 records with 6 features that can be seen in Table **??**. Each record has two decisions associated with each (both binary). For this project, only the first decision will be treated as the output class i.e. the detection of the inflammation of urinary bladder. The other decision attribute is ignored.

#	**Type**	**Description**	**Range**
$a1$	Feature	Temperature of patient	$35\,°C - 42\,°C$
$a2$	Feature	Occurence of nausea	[yes, no]
$a3$	Feature	Lumbar pain	[yes, no]
$a4$	Feature	Urine pushing (continuous need for urine)	[yes, no]
$a5$	Feature	Micturition pains	[yes, no]
$a6$	Feature	Burning of urethra, itch, swelling of urethra outlet	[yes, no]
$d1$	Label	Inflammation of urinary bladder	[yes, no]

Table 1: Attributes of the Acute Inflammations Dataset

2.2 Data Loading and Preprocessing

The downloaded data is converted to an `xlsx` (with `yes/no` converted to `0/1`) file which is read as a table by MATLAB before converting into a matrix. The following pre-processing steps are applied:

1. Normalizing the temperature data using its mean and variance

2. The binary variables are converted from [0, 1] to [-1, 1]. This is necessary for good performance of SVM.

3. The dataset is split into training and testing set, with 52 records in each. Both the training and testing set contain an equal number of positive and negative records i.e., 26.

4. The training set is split equally into two parts: `training_set_a` and `training_set_b` to be used for determining hyperplane and hypersurface respectively.

The MATLAB code used to do this is shown below:

```
% setting seed for repeatability of results
rng(3);

disp('Loading data from XLSX file');
data = readtable('diagnosis.xlsx');
data = table2array(data);

% regularization of temperature data
data(:, 1) = (data(:, 1) - mean(data(:, 1)))/std(data(:, 1));

% determining number of records and features
num_records = size(data, 1);
num_features = size(data, 2) - 1;

% fixing the labels to be 1 (TRUE) or -1 (FALSE)
labels = data(:, end);
labels(labels == 0) = -1;
data(:, end) = labels;

% modifying binary variables from [0, 1] to [-1, 1]
data(data == 0) = -1;

% separtating the binary classes
positive_indices = ismember(data(:, end), ones(num_records, 1))
    ;
negative_indices = ismember(data(:, end), -1 * ones(num_records
    , 1));
```

```
26 true_data = data(positive_indices, :);
27 false_data = data(negative_indices, :);
28
29 disp(' ');
30 disp('Data pre-processed. Example row of data: ');
31 disp('Features: ');
32 disp(data(10, 1: end - 1));
33 disp('Label: ' + num2str(data(10, end)));
34
35 disp(' ');
36 disp('Creating train and test set');
37
38 % choosing number of data records from each binary class to
       create the train set
39 num_choose = 26;
40 train_set = [true_data(1: num_choose, :);
41              false_data(1: num_choose, :)];
42 train_set = train_set(randperm(size(train_set, 1)), :);
43
44 % create the test set similar to the train set
45 test_set = [true_data(num_choose + 1: num_choose * 2, :);
46              false_data(num_choose + 1: num_choose * 2, :)];
47 test_set = test_set(randperm(size(test_set, 1)), :);
48
49 % randomizing the train set before dividing for linear and non-
       linear classifier
50 train_set = train_set(randperm(size(train_set, 1)), :);
51
52 % dividing into two sets
53 train_set_a = train_set(1: num_choose, :);
54 train_set_b = train_set(num_choose + 1: num_choose * 2, :);
55
56 disp('Divided training set into two parts: ' + num2str(
       num_choose) + ...
57      ' records for linear SVM and RBF kernel SVM each');
```

3 Linear SVM using the Primal Method

3.1 Formulation of the Optimization Problem

Giving a training dataset of length n, SVM tries to find a maximum-margin hyperplane that best divides the data. Any hyperplane can be written as a set of points satisying:

$$w^T x - b = 0$$

where w is the weight vector of cardinality equal to the number of features of the dataset, and b is the bias.

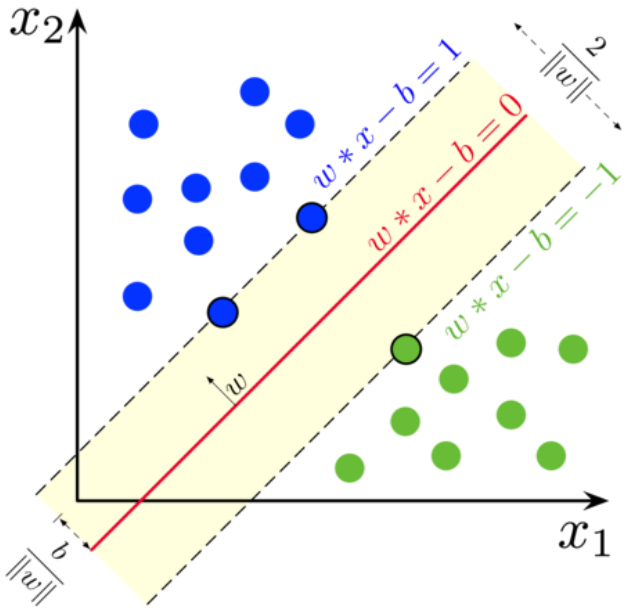

Figure 1: Maximum-margin Hyperplane in SVM

Since the hard-margin classifier is too strict (anomalous data makes it hard to find a possible solution because of the hard constraints), we use a soft-margin classifier based on the hinge loss to determine the optimal weights w and bias b. The task of the classifier then becomes to minimize the following loss function:

$$\lambda\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x - b))$$

where $\lambda > 0$ is the trade-off between increasing the margin size and ensuring that x_i lies on the right side of the margin.
We define a variable $\zeta_i = \max(0, 1 - y_i(w^T x - b))$. Thus, the classification problem converts to the following quadratic optimization problem (primal method):

$$\begin{aligned} \text{minimize} \quad & \frac{1}{n}\sum_{i=1}^{n} \zeta_i + \lambda \|w\|^2 \\ \text{subject to} \quad & y_i(w^T x_i - b) \geq \zeta_i \\ \text{and} \quad & \zeta_i \geq 0 \text{ for all } i \end{aligned}$$

3.2 Implementation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% primal method to find the best separating hyperplane
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp(' ');
disp('Solving primal optimization problem for the linear SVM...
    ');

% using quadratic programming to solve the soft-margin
    classifier based on hinge loss
bias_term = -10;
bias = bias_term * ones(size(train_set_a, 1), 1);
features = [train_set_a(:, 1: end - 1) bias];
labels = train_set_a(:, end);

% quadratic part
lambda = 1;
num_weights = num_features + 1;
num_zeta = size(train_set_a, 1);
num_dec_vars = num_weights + num_zeta;
H = zeros(num_dec_vars);
H(1: num_weights, 1: num_weights) = lambda/2 * eye(num_weights)
    ;
```

```
21
22 % linear part
23 f = 1/num_zeta * [zeros(num_weights, 1); ones(num_zeta, 1)];
24
25 % constraints
26 A = -[[labels .* features eye(num_zeta)]; [zeros(num_zeta,
       num_weights) eye(num_zeta)]];
27 b = -[ones(num_zeta, 1) zeros(num_zeta, 1)];
28
29 % solving the optimization problem
30 x = quadprog(H, f, A, b);
31 w = x(1: num_weights);
32
33 disp(' ');
34 disp('Optimization problem solved with hyperplane weights: ');
35 disp(w);
36
37 % inference on testing set
38 test_features = [test_set(:, 1: end - 1) bias_term * ones(size(
       test_set, 1), 1);];
39 test_labels = test_set(:, end);
40 results = test_features * w;
41
42 % evaluating accuracy
43 results(results >= 0) = 1;
44 results(results < 0) = -1;
45 results_a = results == test_labels;
46 test_accuracy_a = sum(results_a, 'all')/size(test_labels, 1) *
       100;
47
48 disp(' ');
49 disp('[Linear SVM] Accuracy on testing set: ' + num2str(
       test_accuracy_a) + '%');
```

3.3 Output and Results

The outuput of the code shown before is as follows:

```
Solving primal optimization problem for the linear
SVM...

Minimum found that satisfies the constraints.

Optimization completed because the objective function
is non-decreasing in feasible directions, to within
the value of the optimality tolerance, and constraints
are satisfied to within the value of the constraint
tolerance.

Optimization problem solved with hyperplane weights:
   -0.1103
    0.3282
   -0.2069
    0.2573
    0.6385
   -0.0504
   -0.0111

[Linear SVM] Accuracy on testing set: 82.6923%
```

The accuracy of the computed classifier is approximately 82.7% which corresponds to 43 correct classification out of 52 records, with the following confusion matrix.

		True Class		
		Positive	Negative	Total
Predicted Class	Positive	21	4	25
	Negative	5	22	27
	Total	26	26	52

4 SVM using the Kernel Method

4.1 Formulation of the Optimization Problem

To learn a non-linear rule, we use a kernel to transform our data points x_i to a higher dimension and separate them there. The kernel we will use for this project is the *Guassian Radial Basis Function*, which is defined as follows:

$$k(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$$

$$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

We know that owing to the transformed space, the weight vector now corresponds to

$$w = \sum_{i=1}^{n} c_i y_t \varphi(x_i)$$

where the c_i are obtained by solving the following quadratic optimization problem:

$$\text{maximize} \quad \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i k(x_i, x_j) y_j c_j$$

$$\text{subject to} \quad \sum_{i=1}^{n} c_i y_i = 0$$

$$\text{and} \quad 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i$$

4.2 Implementation

The value for *sigma* for the RBF kernel is chosen as 0.25, while the value for λ is chosen as 0.005 for this project.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% kernel trick to find the best separating hypersurface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% parameters: lambda for soft margins, and sigma for RBF kernel
lambda = 0.005;
```

```
sigma = 0.25;

disp(' ');
disp('Solving using the kernel method with lambda = ' + num2str
    (lambda) + ...
      ' and sigma =  ' + num2str(sigma));

% features, labels, and number of records
num_weights = num_features + 1;
num_zeta = size(train_set_b, 1);
features = train_set_b(:, 1: end - 1);
labels = train_set_b(:, end);

% using RBF kernel
sqr = sum(features' .^ 2);
K = exp((2 * (features * features') - sqr' * ones(1, num_zeta)
    ...
         - ones(num_zeta, 1) * sqr)/(2 * sigma ^ 2));

% label vector included
% resulting in the quadratic part of the cost function
H = (labels * labels') .* K;

% linear part
f = -ones(num_zeta, 1);

% equality constraint
Aeq = labels';
beq = zeros(1, 1);

% bounds
lb = zeros(num_zeta, 1);
ub = 1/(2 * num_zeta * lambda) * ones(num_zeta, 1);

% solving the quadratic problem
c = quadprog(H, f, [], [], Aeq, beq, lb, ub);

disp(' ');
```

```
disp('Optimization problem solved with resulting decision
   variables values: ');
disp(c);

% for bias calculation
[~, ind] = max(c);
xj = features(ind, :);
b = exp((2 * features * xj' - sum(xj .^ 2) - sqr')/(2 * sigma ^
    2)) .* labels .* c;
b = sum(b) - labels(ind);

% inference on testing set
test_features = test_set(:, 1: end - 1);
test_labels = test_set(:, end);
num_test = size(test_set, 1);
sqr_t = sum(test_features' .^ 2);
results = exp((2 * test_features * features' - sqr_t' * ...
        ones(1, num_zeta) - ones(num_test, 1) * sqr)/(2 * sigma
            ^ 2));
results = results .* labels' .* c';
results = sum(results, 2) - b;

% evaluating accuracy
results(results >= 0) = 1;
results(results < 0) = -1;
results_b = results == test_labels;
test_accuracy_b = sum(results_b, 'all')/size(test_labels, 1) *
   100;

disp(' ');
disp('[Kernel Method] Accuracy on testing set: ' + num2str(
   test_accuracy_b) + '%');
```

4.3 Output and Results

The outuput of the code shown before is as follows:

```
Solving using the kernel method with lambda = 0.005
and sigma =  0.25

Minimum found that satisfies the constraints.

Optimization completed because the objective function
is non-decreasing in feasible directions, to within
the value of the optimality tolerance, and constraints
are satisfied to within the value of the constraint
tolerance.

Optimization problem solved with resulting decision
variables values:
    0.9033
    0.0000
    0.7222
    0.7222
    0.2962
    0.8714
    0.0000
    0.9431
    0.8661
    0.8659
    0.0000
    0.0000
    0.8659
    0.7222
    0.9033
    0.3002
    0.0000
    0.7222
    0.9396
    0.7222
    0.0000
    0.0000
```

```
0.7222
0.0000
1.0569
0.8714

[Kernel Method] Accuracy on testing set: 94.2308%
```

The accuracy of the computed classifier is approximately 94.2% which corresponds to 49 correct classification out of 52 records, with the following confusion matrix.

		True Class		
		Positive	Negative	Total
Predicted Class	Positive	26	3	29
	Negative	0	23	23
	Total	26	26	52

5 Analysis and Conclusion

By solving the linear SVM classification problem using the primal method, we received an accuracy of 82.7%, while using a non-linear kernel (RBF), we achieved an accuracy of 94.2%. The results of the non-linear classifier are slightly better than the linear one. It can also be noted that the non-linear method did not produce any false negatives, which is extremely important in a medical setting. However, the difference is not significant considering randomness and the small amount of data. Using more data to train will better improve the model and help decide which learning/training technique is superior.Moreover, introduction of a validation set will also help solve the problem of over-fitting, and can help tune λ and σ where they occur.

References

[1] Jacek Czerniak and Hubert Zarzycki. Application of rough sets in the presumptive diagnosis of urinary system diseases. In *Artificial intelligence and security in computing systems*, pages 41–51. Springer, 2003.