Mohammed J. Zaki

Dmcourse /

Assign3

Assign 3: Due Date: 22nd Oct, 2016, before midnight

The goal of this assignment is to implement a simplified version of the Sequential Minimal Optimization (SMO) algorithm by John Platt to train SVMs in the dual formulation.

The SVM maximization problem is as follows:

$$\max\ J(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the constraints $\sum_{i=1}^{n} \alpha_i y_i = 0$, and $0 \leq \alpha_i \leq C$ for $i = 1, 2, \ldots, n$.

The SMO algorithm solves the optimization problem for two values/points at a time, say α_i and α_j, keeping the other values α_k unchanged. This means it must mainta in the following constraint before and after the update:

$$\alpha_i y_i + \alpha_j y_j = const = \alpha_i^{'} y_i + \alpha_j^{'} y_j$$

where α_i denotes the new value and $\alpha_i^{'}$ the old value.

Assume that we update α_j first. Then, since the above invariant has to be maintained, we get the following bounds on the value of α_j, so that $L \leq \alpha_j \leq H$, where

- **case 1:** $y_i \neq y_j$

$$L = \max\ (0, \alpha_j^{'} - \alpha_i^{'})$$

$$H = \min\ (C, C - \alpha_i^{'} + \alpha_j^{'})$$

- **case 2:** $y_i = y_j$

$$L = \max\ (0, \alpha_i^{'} + \alpha_j^{'} - C)$$

$$H = \min\ (C, \alpha_i^{'} + \alpha_j^{'})$$

The update rule for α_j is given as

$$\alpha_j = \alpha_j^{'} + \frac{y_j(E_i - E_j)}{\kappa_{ij}}$$

where

$$\kappa_{ij} = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)$$

is the squared distance between the two points in feature space, and

$$E_k = h(\mathbf{x}_k) - y_k = \left(\sum_{j=1}^{n} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_k) + b\right) - y_k$$

is the difference between the predicted value and the true class for point $\mathbf{x}_k$.

One we have updated α_j using the above equation, we have to clip its value to the interval $[L, H]$, and then we can update the value of α_i as follows:

$$\alpha_i = \alpha_i' + y_i y_j (\alpha_j' - \alpha_j)$$

Note that even though E_k depends on b, when we compute $E_i - E_j$ above the b cancels out, so you do not need b when updating α's.

So the basic SMO algorithm is to iterate over the points in the dataset for the choice of j and to then select random points i, to create a possible pair of values (i, j) to update. After one such round, we can compare the value of the new α compared to the previous set of values α', stopping when the distance between these two falls below some threshold ϵ.

For computing b, once you have found the α's, using Eq. (21.33) in the book. Also for computing the accuracy, use Eq. (22.2) in the book.

Also, when choosing the points, we want to make sure that their α value is not already close to the limit, i.e., we want to make sure that $\alpha > 0$ and $\alpha < C$. To tackle small precision issues, we use a $tol = 10^{-5}$ value, and we make sure that $\alpha \geq tol$ and $\alpha \leq C - tol$. Finally, we use the *tryall* variable to make sure that the first time through the loops we try all pairs.

The complete algorithm in pseudo-code is given as follows:

Input: $kernel, Dataset: \{\mathbf{x}_i, y_i\}_{i=1}^{n}, C, \epsilon$
$\vec{\alpha} = (0, 0, \ldots, 0)$
$tol = 10^{-5}$
tryall = True
repeat
    $\vec{\alpha}_{prev} = \vec{\alpha}$
    for $j = 1, 2, \ldots, n$ **do**
        if tryall = False and ($\alpha_j - tol < 0$ or $\alpha_j + tol > C$) **then**
            skip to next j
        for $i = 1, 2, \ldots, n$ in random order such that $i \neq j$ **do**
            if tryall = False and ($\alpha_i - tol < 0$ or $\alpha_i + tol > C$) **then**
                skip to next i
            compute κ_{ij} based on kernel type (linear or quadratic)
            if $\kappa_{ij} = 0$ skip to next i
            $\alpha_j' = \alpha[j]$ and $\alpha_i' = \alpha[i]$
            compute L and H based on the two cases
            if $L = H$ skip to next i

compute E_i and E_j
$\alpha[j] = \alpha_j' + \frac{y_j(E_i - E_j)}{\kappa_{ij}}$
if $\alpha[j] < L$ **then** $\alpha[j] = L$
else if $\alpha[j] > H$ **then** $\alpha[j] = H$
$\alpha[i] = \alpha_i' + y_i y_j(\alpha_j' - \alpha[j])$
end for
end for
if tryall **then** tryall = False
until $\|\vec{\alpha} - \vec{\alpha}_{prev}\| \leq \epsilon$
compute bias b
print support-vectors i, α_i such that $\alpha_i - tol > 0$ and $\alpha_i + tol < C$
print bias
compute accuracy on training set
print training accuracy
CSCI6390 Only: print $\mathbf{w}$ for linear and quadratic kernel

What to turn in

Write a script named **assign3-LAST-FIRST.py** where **LAST** and **FIRST** are your last and first names, respectively. The script will be run as fo llows:
assign3.py FILENAME C eps [linear OR quadratic OR gaussian] spread
where FILENAME is the data file name, which will contain each point on a line, with comma separated attributes, and with the last attribute denoting the binary class; C is the regularization constant (a real number); linear, quadratic or gaussian d enotes the kernel to use; and eps is the ϵ value for convergence. You can assume that quadratic means homogeneous quadratic kernel (see ch ap 5). If the kernel is a gaussian, you should also specify the spread parameter on the command line.

For the FILENAME dataset, you can assume that each line contains one feature vector , with "," as the separator, and the last feature/attribute denotes the class, which can be -1 or +1.

Save your output to a text file **assign3-LAST-FIRST.txt**. It should contain the output of the print statements. Note that the accuracy is on the training set, i.e., after learning the α_i values, how many points are correctly classified divided by the total number of points.

Try your method on the following files:

assign3.py Attach:iris-virginica.txt 1 0.001 linear

assign3.py Attach:iris-versicolor .txt 1 0.001 linear (this dataset will not yield a good accuracy with linear kernel)

assign3.py Attach:iris-versicolor .txt 1 0.001 quadratic (with quadratic or gaussian we get good accuracy)

Finally, show your results on Attach:Concrete_Data_RNorm_Class.txt This is based on dataset as that at the UCI repository, but I have normalized the values to lie in the range 0 to 1. Try a "few" dif ferent values and report your results on the "best" combination of C & kernel (and spread if using gaussian).

Email your script and output file to datamining.rpi@gmail.com . The subject of the email should be **assign3-LAST-FIRST**.

Page last modified on October 21, 2016, at 03:32 PM