Homework 3

Dustin Leatherman

April 19, 2020

Contents

1 Part 1 **1**

2 Part 2 **2**
2.1 Boosting - Using error as the weight 2
2.2 Boosting - $Error^2$ 4

3 Appendix **5**
3.1 Code 5

1 Part 1

> Make a Table. The first column of this table should contain values of s, the second should be $\epsilon median$. Note that the median of $\epsilon\{$ depends on s. See page 1 for the definition of ϵ

```
nrow = 200; Ndim = 1000; s = <sparsity>
# refactored the example algorithm into discrete functions to make this easier to calc
E = []
for i = 1:5
  Amat = randn(nrow,Ndim);
  Amat = Amat * 1/sqrt(nrow);
  x_true = gen_sparse_vec(Ndim, s);
  yvect = Amat*x_true;
  xvect = recover_sparse_vec(Ndim, Amat, yvect);
  error = get_error(x_true, xvect);
  E = [E; error];
end
```

```
median(E)
```

s	ϵ_{median}
20	1.5536
30	2.6475
35	2.6697
40	8.0505
60	76.876

2 Part 2

> Describe carefully at least 2 interesting scenarios that you have tried. What did you observe? How does the performance of the algorithm change depending on how you select the parameter ϵ in each iteration?

2.1 Boosting - Using error as the weight

The first method I tried was to use **Boosting** to update the weights. The idea behind boosting is to update the weights using the classifier with the lowest error. In my implementation, I check for the error between the previous vector and the new vector and choose the epsilon to be the lower of the two values.

This performed poorly across the board but became comparable in performance at s = 60.

s	ϵ_{median}
20	11.173
30	28.277
35	36.161
40	49.588
60	77.347

```
function xvect = recover_sparse_vec(Ndim, Amat, yvect)

  wvect = ones(Ndim,1);

  inv_wvect = zeros(Ndim,1);

  prediction_error = zeros(40,1);
  prev_xvect = [];
```

```
  for iter = 1:30
%%% make the Qmatrix
    for j = 1:Ndim
        inv_wvect(j) = 1/wvect(j);
    end

    Qmat = diag(inv_wvect);

    xvect = Qmat*Amat'*inv(Amat*Qmat*Amat')*yvect;

    if isempty(prev_xvect)
      eps = 0;
      old_err = 999999999;
    else
      new_err = get_error(prev_xvect, xvect)/100
      if new_err < old_err
        eps = new_err
        old_err = new_err
      end
    end
    %%% update the weights
    for j = 1:Ndim
        wvect(j) = 1/sqrt( xvect(j)*xvect(j) + eps);
    end
    prev_xvect = xvect;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%  trim the values of xvect that are below threshold value
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  threshold = 0.05;

  for k = 1:Ndim
    if abs(xvect(k)) <= threshold
        xvect(k) = 0;
    end
  end
```

```
end
```

2.2 Boosting - $Error^2$

Since Boosting with using the straight error yielded values that were too large, squaring the error values might get better results.

s	ϵ_{median}
20	0.020667
30	0.30059
35	1.6775
40	21.646
60	80.068

Lower sparsity vectors did much better compared to the original algorithm but it began to fall apart around s = 40.

```
function xvect = recover_sparse_vec(Ndim, Amat, yvect)

  wvect = ones(Ndim,1);

  inv_wvect = zeros(Ndim,1);

  prediction_error = zeros(40,1);
  prev_xvect = [];
  for iter = 1:30
%%% make the Qmatrix
    for j = 1:Ndim
        inv_wvect(j) = 1/wvect(j);
    end

    Qmat = diag(inv_wvect);

    xvect = Qmat*Amat'*inv(Amat*Qmat*Amat')*yvect;

    if isempty(prev_xvect)
      eps = 0;
      old_err = 999999999;
    else
      new_err = get_error(prev_xvect, xvect)/100
      if new_err < old_err
```

```
        eps = new_err^2
        old_err = new_err
      end
    end
    %%% update the weights
    for j = 1:Ndim
        wvect(j) = 1/sqrt( xvect(j)*xvect(j) + eps);
    end
    prev_xvect = xvect;
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%  trim the values of xvect that are below threshold value
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  threshold = 0.05;

  for k = 1:Ndim
    if abs(xvect(k)) <= threshold
        xvect(k) = 0;
    end
  end
end
```

3 Appendix

3.1 Code

```
%%    Algorithm to solve Problem (P1)
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%    Make the 20-sparse vector x_true
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function sparse_vec = gen_sparse_vec(Ndim, s)
  sparse_vec = zeros(Ndim,1);

  for k = 201:(201 + s - 10 - 1)
    sparse_vec(k) = 4;
  end

  for k = 251:260
    sparse_vec(k) = -4;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%  Solve the optimization problem:
%%%%      Given yvect and Amat, recover the 20-sparse vector
%%%%  SNEAKY ALGORITHM
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xvect = recover_sparse_vec(Ndim, Amat, yvect)

  wvect = ones(Ndim,1);

  inv_wvect = zeros(Ndim,1);

  prediction_error = zeros(40,1);
  prev_xvect = []
  for iter = 1:30
%%% make the Qmatrix
    for j = 1:Ndim
        inv_wvect(j) = 1/wvect(j);
    end

    Qmat = diag(inv_wvect);

    xvect = Qmat*Amat'*inv(Amat*Qmat*Amat')*yvect;

    xvect(j) * xvect(j)
```

```
    if isempty(prev_xvect)
      eps = 0;
      old_err = 0;
    else
      new_err = get_error(prev_xvect, xvect)
      if new_err < old_err
        eps = 1/new_err;
      end
    end
    %%% update the weights
    for j = 1:Ndim
        wvect(j) = 1/sqrt( xvect(j)*xvect(j) + eps);
    end
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%  trim the values of xvect that are below threshold value
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  threshold = 0.05;

  for k = 1:Ndim
    if abs(xvect(k)) <= threshold
        xvect(k) = 0;
    end
  end
end

function error = get_error(x_true, xvect)
%%% how good is the prediction
  error = 100*norm(x_true - xvect)/norm(x_true);
end
```