Report for HW2

Yang, Yunfan **Professor Tu**

yuy130@ucsd.edu

Abstract

In this report, I am going to explore structure prediction that is designed to understand the relationship between labels. I will study fix-point algorithm, SVMstruct, and M3N algorithm on OCR and POS datasets.

1 Introduction

Binary classifiers and multiclass classifiers seem to be useful on simple data. In real world, there are a lot of relational data. Traditional binary classifiers and multiclass classifiers ignore the relationship between objects. In fact, understanding the relationship between objects is extremely helpful to solve real world classifying problems. In this report, I will study fix-point algorithm and SVMstruct algorithm on OCR and POS datasets.

2 Methodology

2.1 Technology

Each experiment is written in matlab. For the fix-point algorithm, I use the source code from the original paper and did some changes. For the SVMstruct algorithm, I use SVMstruct as my SVM solver.

2.2 Data Preprocessing

OCR and POS datasets are from the paper, fixed point model for structured labeling. Both datasets contain structured data. OCR dataset contains 128 features and each sample contains 6 8 characters in average. The dataset is divided in to 10 folds. The input feature is a vector representation of 16x8 handwritten image. POS dataset contains 446054 features. POS dataset is divided in to training size of 500, 1000, 2000, 4000, 8000 sentences. The data obtained from the paper has already been processed and thus only needs to be read in matlab in a proper form.

3 Experiment

Fix-point:

The fix-point algorithm is obtained from prof Tu's website. Thus, only minor changes need to made to generate result for this report. For simplicity, I only trained first 5000 rows from OCR dataset and first 10% rows from POS dataset. For OCR datset, I train all 10 folds and use 5 iterations.

SVMstruct:

I obtain SVMstruct algorithm as my SVM solver. I use 1-slack without feature caching as my optimizing algorithm. Then, I define three functions, constraintCB, lossCB, and featureCB to perform training. Finally,

Table 1: error rate of fix-point on POS dataset on different window size

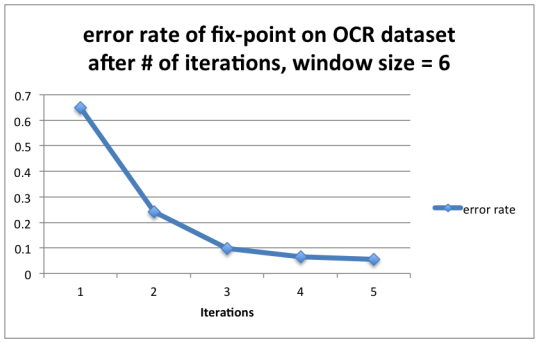
windows size	500	1000	2000	4000	8000
2	0.25	0.16	0.14	0.11	0.09
4	0.25	0.16	0.14	0.11	0.09
6	0.25	0.17	0.14	0.11	0.09
8	0.25	0.17	0.14	0.11	0.09
10	0.25	0.17	0.14	0.10	0.09

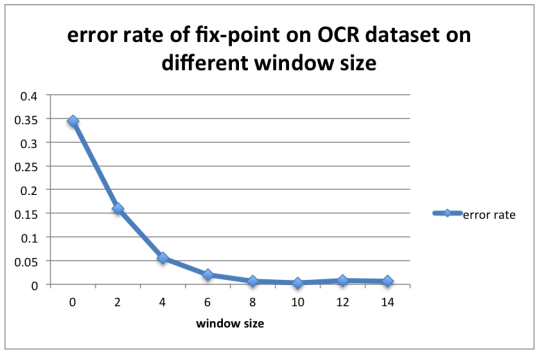
I define a predict to perform testing. During training stage, I apply gibbs sampling to speed up the process. Gibbs sampling increase training phase by server order of magnitude so that I am be able to train POS dataset. To train POS dataset efficiently, I also apply multiple optimizing techniques including using sparse vectors. After using sparse vectors during constructing features, I am able to cut down the time to build each feature from 2 seconds to 0.001 seconds. Also, I chop each sample to size of 2 so that efficient training is able to conduct. For the OCR dataset, I chop data to size of 3. Each feature is encoded using formula from option 3 from lecture slides 3.

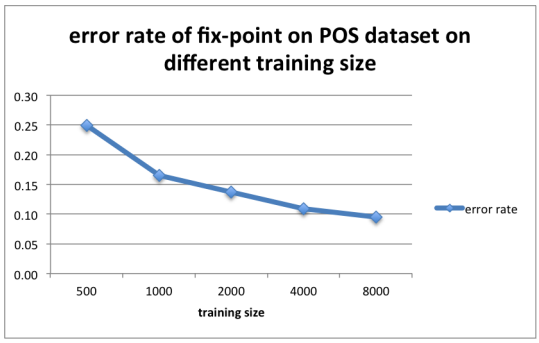
M3N

My M3N setup is mostly the same as SVMstruct besides in M3N algorithm, I use a graph to define each feature and thus difference between each feature can be better represented. I apply gibbs sampling during training stage and chop data to equal length of 3.

3.1 Comparison






error rate of fix-point on POS dataset on different training size
0.30
0.25
0.20
0.15
0.10
0.05
0.00
500
1000
2000
4000
8000
training size
error rate

The error rate of SVMstruct on OCR dataset using gibbs sampling and option 3 is 7.04%. The error rate of SVMstruct on OCR data set without using gibbs sampling and with option 3 is 15.49%. The error rate of SVMstruct on OCR dataset with gibbs sampling and with option 2 is 7.04%. The error rate of SVMstruct on OCR dataset wihout using gibbs sampling and with option 2 is 7.04%.
From my experiment, it shows that using gibbs sampling during training phase doesn't affect result at all. In fact, in option 3, it even generates better result. I think this might due to the overfitting casued by too many comparisons. By applying gibbs sampling, the training time reduces by several magnitude. However, the prediction time still doesn't change since each sample needs to compete with all possible hypothesis.
By applying gibbs sampling and multiple optimizing techniques, I am able to cut down the time to compute each POS sample from 2 seconds to 0.001 seconds that enables me to train POS dataset in a reasonable amount of time. When training size is 500, the accuracy I got is 95.8%. When training size if 1000, the accuracy I got is 90.9%. The accuracy seems a bit surprising and I think this might due to I only test on a subset of the total samples and chop each sample to size of 2.
The fix-point algorithm seems to generate very decent result. On OCR dataset, from the above graph, we can see that as window size increases, the error rate decreases and finally it converges to around 0.6%. When window size is 6, the error rate of fix-point algorithm on OCR dataset decreases after a few rounds of iterations and finally converges.
On POS dataset, it seems like that window size doesn't affect error rate as at all. In fact, the error rate of fix-point algorithm on POS dataset relies on training size. The larger training size yields better result.
On OCR dataset, M3N algorithm has the best performance, it reached 0 error rate. The reason is probably because M3N algorithm uses more features to distinguish each sample. M3N algorithm applies a graph to distinguish each sample. Although it largely increases sample size, it dramatically increases classification accuracy.

4 Conclusion

SVMstruct algorithm is very expensive since each feature needs to compete with all possible hypothesis. By applying gibbs sampling, exponential runtime is cut down to polynomial runtime. However, testing stage still takes too long since each sample needs checking with all possible hypothesis. Due to the nature of SVMstruct, some better algorithms need to be used on a sample with large size of labels. M3N algorithm is one possible approach that can both cut down training time and testing time to polynomial time. In contrast, fix-point algorithm is a better algorithm that is able to converge after only a few iterations without enumerating all possible hypothesis. M3N algorithm is very similar to SVMstruct. M3N algorithm uses a smarter way to define feature that increases classification accuracy by a lot.

Acknowledgments

We want to expressly acknowledge the teaching and instruction provided by Prof. Zhuowen Tu. Additionally we want to highlight our exceptional TA, Saining Xie and Daniel Maryanovsky, whose mentoring and patience was instrumental for our success throughout this project and the course at large.

Source Code

train svm struct gibbs 1000.m

```
% train_svm_struct using option3
% double(y(1) ~= y(2)); double(y(2) ~= y(3)) ;double(y(1) ~= y(3))
% optmized using gibbs sampling
function train_svm_struct
    % ------------------------------------------------------------------
    %                                                          Generate data
    % ------------------------------------------------------------------
    load letter1.data
    global x_train x_test y_train y_test parm model C;

    % only load first 5000 samples
    letter1 = letter1(1:5000,:);
    FOLD = 6;
    LABEL = 2;
    NEXT_ID = 3;
    CURR_ID = 5;
    C = 26;
    foldNum = 1;
    chop = 3;
    BM = 7;
    D = 128;

    x_train = {};
    y_train = {};
    x_test = {};
    y_test = {};
    temp_x = {};
    temp_y = [];

    for i = 1 :  size(letter1, 1)
        temp_x = [temp_x letter1(i, BM:end)'];
        temp_y = [temp_y letter1(i, LABEL) + 1];

        if(letter1(i, NEXT_ID) == -1)
            if letter1(i, FOLD) == foldNum
                %for training
                x_train = [x_train;{temp_x(:, 1: chop)}];
                y_train = [y_train;temp_y(:, 1: chop)];
            else
                %for testing
                x_test = [x_test;{temp_x(:, 1: chop)}];
                y_test = [y_test;temp_y(:, 1: chop)];
            end
            temp_x = {};
            temp_y = [];
        end
    end

    rand('seed', 1);
    perm = randperm(length(x_train));
    training_samples = 20;

    x_train = x_train(perm);
    x_train = x_train(1:training_samples,1:end);
    y_train = y_train(perm);
```

```
    y_train = y_train(1:training_samples,1:end);
    x_test = x_test(perm);
    y_test = y_test(perm);
    testing_samples = size(y_test, 1);
    % -------------------------------------------------------------------
    %                                                          Run SVM struct
    % -------------------------------------------------------------------
    parm.patterns = x_train ;
    parm.labels = y_train ;
    parm.lossFn = @lossCB ;
    parm.constraintFn  = @constraintCB ;
    parm.featureFn = @featureCB ;
    parm.dimension = C*D*chop+3;
    parm.verbose = 1 ;
    tic
    model = svm_struct_learn(' -c 1.0 -o 1 -v 1  ', parm) ;
    training_time = toc;
    w = model.w ;

    % predict
    tic
    correct = 0;
    for i = 1: length(x_test)
        if sum(predict(parm, model, x_test{i}) ~= y_test{i}) == 0
            correct = correct + 1;
        end
    end
    testing_time = toc;

    disp('acc: ')
    acc = correct/length(x_test)
    save('train_svm_struct_3_gibbs.mat', 'acc', 'training_time', '
       testing_time', 'training_samples', 'testing_samples');
end
% -------------------------------------------------------------------
%                                                    SVM struct callbacks
% -------------------------------------------------------------------

function ypredict = predict(param, model, x)
  max = -1;
  C = 26;
  for i = 1: C
      for j = 1: C
          for k = 1: C
              yh = [i j k];
              val = dot(featureCB(param, x, yh), model.w);
              if val > max
                  max = val;
                  ypredict = yh;
              end
          end
      end
  end
end

function delta = lossCB(param, y, ybar)
  delta = double(sum(y ~= ybar)) ;
end
```

```
function psi = featureCB(param, x, y)
  D = 128;
  window_size = 3;
  res = [];
  for i=1:window_size
        res = [res; zeros( D*(y(i) - 1), 1); x{i} ;zeros( D*(26-y(i)), 1)
           ];
  end
  psi = sparse([res ;double(y(1) ~= y(2)); double(y(2) ~= y(3)) ;double(y
     (1) ~= y(3))]);
end

function yhat = constraintCB(param, model, x, y)
% slack resaling: argmax_y delta(yi, y) (1 + <psi(x,y), w> - <psi(x,yi), w
   >)
% margin rescaling: argmax_y delta(yi, y) + <psi(x,y), w>
  max = -1;
  C = 26;
  iter = 3;
  % use gibbs sampling
  for i = 1: iter
      for j = 1: 3
          for k = 1: C
              yh = [1 1 1];
              yh(j) = k;
              val = dot(featureCB(param, x, yh), model.w) + sum(lossCB(
                 param, y, yh))/length(yh);
              if val > max
                  max = val;
                  yhat = yh;
              end
          end
      end
  end
end
```

train svm struct 3 gibbs.m

```
function train_svm_struct
    % ------------------------------------------------------------------
    %                                                          Generate data
    % ------------------------------------------------------------------
    load data/1000/posTrainData
    global x_train x_test y_train y_test parm model;

    % only load first 5000 samples
    letter1 = theData(1:500,:);
    LABEL = 2;
    NEXT_ID = 3;
    C = 41;
    foldNum = 1;
    chop = 2;
    BM = 7;
    D = 446054;
    x_train = {};
    y_train = {};
    x_test = {};
    y_test = {};
    temp_x = {};
```

```
temp_y = [];

for i = 1 :  size(letter1, 1)
    temp_x = [temp_x {letter1(i, BM:end)'}];
    temp_y = [temp_y full(letter1(i, LABEL) + 1)];

    if(letter1(i, NEXT_ID) == -1)
        %for training
        x_train = [x_train;{temp_x(:, 1: chop)}];
        y_train = [y_train;temp_y(:, 1: chop)];
        temp_x = {};
        temp_y = [];
    end
end

rand('seed', 1);
perm = randperm(length(x_train));
%training_samples = 15;
training_samples = size(x_train,1);

x_train = x_train(perm);
%x_train = x_train(1:training_samples,1:end);
y_train = y_train(perm);
%y_train = y_train(1:training_samples,1:end);

% ------------------------------------------------------------------
%                                                        Run SVM struct
% ------------------------------------------------------------------
parm.patterns = x_train ;
parm.labels = y_train ;
parm.lossFn = @lossCB ;
parm.constraintFn  = @constraintCB ;
parm.featureFn = @featureCB ;
parm.dimension = C*D*chop+1 ;
parm.verbose = 1 ;
tic
model = svm_struct_learn(' -c 1.0 -o 1 -v 1  ', parm) ;
training_time = toc;
w = model.w ;

% predict
load data/1000/posVerifyData

temp_x = {};
temp_y = [];
for i = 1 :  size(letter1, 1)
    temp_x = [temp_x {letter1(i, BM:end)'}];
    temp_y = [temp_y full(letter1(i, LABEL) + 1)];

    if(letter1(i, NEXT_ID) == -1)
        %for training
        x_test = [x_test;{temp_x(:, 1: chop)}];
        y_test = [y_test;temp_y(:, 1: chop)];
        temp_x = {};
        temp_y = [];
    end
end

rand('seed', 1);
```

```
    perm = randperm(length(x_test));

    x_test = x_test(perm);
    y_test = y_test(perm);

    testing_samples = size(y_test, 1);

    tic
    correct = 0;
    for i = 1: length(x_test)
        if sum(predict(parm, model, x_test{i}) ~= y_test{i}) == 0
            correct = correct + 1;
        end
    end
    testing_time = toc;

    disp('acc: ')
    acc = correct/length(x_test)
    save('train_svm_struct_gibbs_1000.mat', 'acc', 'training_time', '
       testing_time', 'training_samples', 'testing_samples');
end
% --------------------------------------------------------------------
%                                                    SVM struct callbacks
% --------------------------------------------------------------------

function ypredict = predict(param, model, x)
  max = -1;
  C = 41;
  for j = 1: C
      for k = 1: C
          yh = [j k];
          val = dot(featureCB(param, x, yh), model.w);
          if val > max
              max = val;
              ypredict = yh;
          end
      end
  end
end

function delta = lossCB(param, y, ybar)
  delta = double(sum(y ~= ybar)) ;
end

function psi = featureCB(param, x, y)
  window_size = 2;
  len = 446054;
  res = [];
  C = 41;
  D = 446054;
  chop = 2;
  for i=1:window_size
    res = [res; sparse( len*(y(i) - 1), 1); x{i} ;sparse( len*(C-y(i)), 1)
       ];
  end
  res = [res; y(1) ~= y(2)];
  psi = res;
end
```

```
function yhat = constraintCB(param, model, x, y)
% slack resaling: argmax_y delta(yi, y) (1 + <psi(x,y), w> - <psi(x,yi), w
   >)
% margin rescaling: argmax_y delta(yi, y) + <psi(x,y), w>
  max = -1;
  C = 41;
  iter = 3;
  % use gibbs sampling
  tic
  for i = 1: iter
      for j = 1: 2
          for k = 1: C
              yh = [1 1];
              yh(j) = k;
              val = dot(featureCB(param, x, yh), sparse(model.w)) + sum(
                 lossCB(param, y, yh))/length(yh);
              if val > max
                  max = val;
                  yhat = yh;
              end
          end
      end
  end
  toc
end
```

train M3N 3 gibbs.m

```
% train_M3N
% double(y(1) ~= y(2)); double(y(2) ~= y(3)) ;double(y(1) ~= y(3))
% optmized using gibbs sampling
function train_m3n
    % ---------------------------------------------------------------------
    %                                                          Generate data
    % ---------------------------------------------------------------------
    load letter1.data
    global x_train x_test y_train y_test parm model C;

    % only load first 5000 samples
    letter1 = letter1(1:5000,:);
    FOLD = 6;
    LABEL = 2;
    NEXT_ID = 3;
    CURR_ID = 5;
    C = 26;
    foldNum = 1;
    chop = 3;
    BM = 7;
    D = 128;

    x_train = {};
    y_train = {};
    x_test = {};
    y_test = {};
    temp_x = {};
    temp_y = [];

    for i = 1 :  size(letter1, 1)
        temp_x = [temp_x letter1(i, BM:end)'];
```

```
        temp_y = [temp_y letter1(i, LABEL) + 1];

        if(letter1(i, NEXT_ID) == -1)
            if letter1(i, FOLD) == foldNum
                %for training
                x_train = [x_train;{temp_x(:, 1: chop)}];
                y_train = [y_train;temp_y(:, 1: chop)];
            else
                %for testing
                x_test = [x_test;{temp_x(:, 1: chop)}];
                y_test = [y_test;temp_y(:, 1: chop)];
            end
            temp_x = {};
            temp_y = [];
        end
    end

    rand('seed', 1);
    perm = randperm(length(x_train));
    training_samples = 20;

    x_train = x_train(perm);
    x_train = x_train(1:training_samples,1:end);
    y_train = y_train(perm);
    y_train = y_train(1:training_samples,1:end);
    x_test = x_test(perm);
    y_test = y_test(perm);
    testing_samples = size(y_test, 1);
    % -------------------------------------------------------------------
    %                                                          Run SVM struct
    % -------------------------------------------------------------------
    parm.patterns = x_train ;
    parm.labels = y_train ;
    parm.lossFn = @lossCB ;
    parm.constraintFn  = @constraintCB ;
    parm.featureFn = @featureCB ;
    parm.dimension = C*D*chop+C*C*(chop-1);
    parm.verbose = 1 ;
    tic
    model = svm_struct_learn(' -c 1.0 -o 1 -v 1', parm) ;
    training_time = toc;
    w = model.w ;

    % predict
    tic
    correct = 0;
    for i = 1: length(x_test)
        if sum(predict(parm, model, x_test{i}) ~= y_test{i}) == 0
            correct = correct + 1;
        end
    end
    testing_time = toc;

    disp('acc: ')
    acc = correct/length(x_test)
    save('train_m3n_3_gibbs.mat', 'acc', 'training_time', 'testing_time',
       'training_samples', 'testing_samples');
end
% -------------------------------------------------------------------
```

```
%                                                    SVM struct callbacks
% ---------------------------------------------------------------------

function ypredict = predict(param, model, x)
  max = -1;
  C = 26;
  for i = 1: C
      for j = 1: C
          for k = 1: C
              yh = [i j k];
              val = dot(featureCB(param, x, yh), model.w);
              if val > max
                  max = val;
                  ypredict = yh;
              end
          end
      end
  end
end

function delta = lossCB(param, y, ybar)
  delta = double(sum(y ~= ybar)) ;
end

function psi = featureCB(param, x, y)
  D = 128;
  chop = 3;
  res = [];
  C = 26;
  for i=1:chop
        res = [res; zeros( D*(y(i) - 1), 1); x{i} ;zeros( D*(26-y(i)), 1)
           ];
  end
  graph = zeros(C*C*(chop-1), 1);
  graph(y(1) * C + y(2)) = 1;
  graph(y(2) * C + y(3)) = 1;
  psi = sparse([res ; graph]);
end

function yhat = constraintCB(param, model, x, y)
% slack resaling: argmax_y delta(yi, y) (1 + <psi(x,y), w> - <psi(x,yi), w
   >)
% margin rescaling: argmax_y delta(yi, y) + <psi(x,y), w>
  max = -1;
  C = 26;
  iter = 3;
  % use gibbs sampling
  for i = 1: iter
      for j = 1: 3
          for k = 1: C
              yh = [1 1 1];
              yh(j) = k;
              val = dot(featureCB(param, x, yh), model.w) + sum(lossCB(
                 param, y, yh))/length(yh);
              if val > max
                  max = val;
                  yhat = yh;
              end
          end
```

```
      end
  end
end
```

References

[1] Q. Li, J. Wang, D. Wipf, and Z. Tu, "Fixed-Point Model for Structured Labeling", 2013.