EE6227: Programming Assignment 2

Wei Zhifeng G2002825F

A. Algorithm Description and Implement

i. Random Forests

Random Forest is a kind of ensemble classifier, it contains n decision trees that trained either independently or sequentially. And its output is gained by weighting votes from all these decision trees.

The main idea of training a random forest is to perturb the training dataset and inject some randomness in each decision tree, aggregate them in a suitable way to gain a better output.

The first step of training a random forest is to generate different training data sets. We use bagging method to extract data from training data. In other words, we sample from the original training data with replacement, if we want to train n trees, we perform n times bagging to obtain n different data sets.

```
for i = 1:nTree
    B_indx{i} = randsample(size(trainX, 1), size(trainX, 1), true);
    %sample Ntimes from N trainingcases with replacement
end
```

After having n data sets, we begin to train n decision tree with corresponding dataset. In each trees, we randomly choose mtry features to split the data in each node. Normally, mtry is the square root of total features.

```
for i = 1 : nTrees

   %for parameter tuning
   %TDindx = randsample(M, numel(Labels), true);


   TDindx = B_indx{i}; %use the indices in the bags. Random sampling with replacement

   tic            %starts the clock

    Random_ForestT = cartree_train(Data(TDindx, :), Labels(TDindx), TDindx, options.mtry);

    Trainingtime_temp=toc;
    Trainningtime=Trainningtime+Trainingtime_temp;


    Random_ForestT.method = method;

    Random_ForestT.oobe = 1;

    Random_Forest(i) = Random_ForestT;
end
```

When the node is not a pure node, we randomly pick mtry features as criterion to split the data at this node. And we use axis_parallel_cut method to split the data. Every time, we use one feature to split the data and calculate its gini_impurity. Comparing these gini_impurity, we pick the maximum one to split the data and build two child nodes.

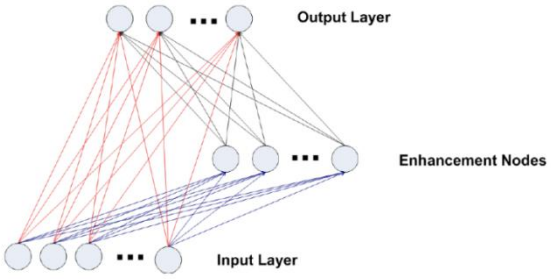
```
if numel(currentDataIndx)>2*minparent

    node_var = randperm(M);
    node_var = node_var(1:mtry);%randomly pick mtry feature

    X = Data(currentDataIndx,node_var);%pick feature value

    [bestCutVar,bestCutValue] =axis_parallel_cut(Labels(currentDataIndx),X,minleaf);
    %divide using gini_impurity
    if bestCutVar~=-1
        b=zeros(mtry+1,1);
        b(bestCutVar)=1;
        b(end)=bestCutValue;
    end
```

When all the nodes are a pure node, we have complete the training of a decision tree. And we repeat the preceding procedure to train n decision trees and aggregate them together to form a random forest.

If we want to use the random forest to predict, just input the data to every decision tree and select the label that appear most time in these trees.

ii. Random Vector Functional Link(RVFL)

Random Vector Functional Link is a special neural network, whose weight in hidden layer nodes are appropriately and randomly generated and keep fixed during the training procedure. Its input layer nodes are also connected to output layer nodes directly. The structure is shown as the picture below.



In the hidden layer nodes(enhancement nodes), the feature is computed as the formula below

$$h(x) = g(wx + b)$$

g denotes a non-linear activate function, and w is the fixed weight we randomly defined previously. b is the bias for the enhancement layer.

In the output layer, the input features are concatenated with the output of hidden layer to improve the performance. The formula is shown below

$$f(x) = \beta * d$$

d denotes the combined features and β denotes the weight in output layers.

The d is shown as the picture shown below.

$$\begin{bmatrix} h_1(x_1) & \cdots & h_k(x_1) & x_{11} & \cdots & x_{1m} \\ h_1(x_2) & \cdots & h_k(x_2) & x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ h_1(x_n) & \cdots & h_k(x_n) & x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

Therefore, to optimize this network, we should solve the following problem

$$\min_{\beta} \|\beta * d - Y\| + \gamma * \|\beta\|^2$$

Matric d represents the combined features; matric Y represents the expected output; γ is the regularization parameter.

The solution of the above problem can be summarized as below:

- In the prime space(if the number of training samples is larger than total feature):

$$\beta = (\gamma I + d^T d)^{-1} d^T Y$$

- In the dual space(if the number of training samples is larger than total feature):

$$\beta = d^T (\gamma I + d^T d)^{-1} Y$$

The following matlab code realizes the procedure of training a RVFL network.

```
TrainingTime=0;
if isequal(size(target,1), size(input,1))==0
    error('Error: input and target sizes dismatch')
else
    tic;
    target=targetCreate(target);
    [net.normparameters.minn, net.normparameters.maxx, input]=normD(input);
    inputneuronnumber=size(input, 2);%input features
    %generate the hiden layer weight
    net.hiddenlayerweights=rand(inputneuronnumber,enhancementnodesneuronnumber);
    %output of hiden layer
    %enhancementnodesoutput=logsig(input*net.hiddenlayerweights);
    enhancementnodesoutput=logsig(input*net.hiddenlayerweights);
    %concatenate input and hidden layer
    hiddenlayerout=[input, enhancementnodesoutput]; % direct and enhacement nodes
    %cakculate the β
    c=1/length(unique(target));
    dd=hiddenlayerout.'*hiddenlayerout
    if size(input,1)>size(input,2)
        net.outputlayerweights=pinv(c*eye(size(dd,1),size(dd,2))+dd)*hiddenlayerout.'*target
    else
        net.outputlayerweights=hiddenlayerout.'*pinv(c*eye(size(dd,1),size(dd,2))+dd)*target
    end
```

B. Parameters Description and Tuning

i. Random Forests

In Random Forests algorithm, there are two important parameters for tuning. One is the ntree, which denotes the number of trees in a forest; the other one is mtry. Mtry is the number of features randomly chosen to split in each non-leaf node, a suitable mtry could raise the final accuracy.

The range of mtry is from 1 to the total features that a input variable has. In our tuning experiments, we run exhaustive trial to test these mtry to find a suitable mtry for this dataset.

```
%% Tuning
% Parameters to tune (You can also experiment with different values)
mtry_range = 1:size(trainX,2);

best_acc = 0;   % Initialisation

% Requried for consistency
s = RandStream('mcg16807','Seed',0);
RandStream.setGlobalStream(s);
%拆分数据集
cv_part = cvpartition(trainY,'KFold',4);    % Create indices for training/validation subsets

% Test every configuration
for p1 = 1:numel(mtry_range)

    TestModelParameters = ModelParameters;
    TestModelParameters.mtry = mtry_range(p1);
```

In the previous code, we try every mtry to test its performance, and use the best one to train our random forest.

Because we don't have enough training data, and in order not to over fit these training data. We use k-fold algorithm to tuning this parameter to tune mtry.

We divided the training data to 4 parts. Each time ,we take 1 part as validation data,other 3 parts as training data. Average these four times validation accuracy as the final training accuracy to find the better mtry.

```
cv_part = cvpartition(trainY,'KFold',4);    % Create indices for training/validation subsets

% Test every configuration
for p1 = 1:numel(mtry_range)

    TestModelParameters = ModelParameters;
    TestModelParameters.mtry = mtry_range(p1);

    val_acc = zeros(4,1);   % Initialisation
    for k = 1:4
        % Collect training/validation sets
        val_trainX = trainX(cv_part.training(k),:);
        val_trainY = trainY(cv_part.training(k),:);
        val_testX = trainX(cv_part.test(k),:);
        val_testY = trainY(cv_part.test(k),:);
```

```
        % Data Normalisation
        mean_X = mean(val_trainX,1);
        std_X = std(val_trainX);
        std_X(std_X==0) = 1e-4;                 % For numerical stability
        val_trainX = bsxfun(@rdivide,val_trainX-repmat(mean_X,size(val_trainX,1),1),std_X);
        val_testX = bsxfun(@rdivide,val_testX-repmat(mean_X,size(val_testX,1),1),std_X);
        %bsxfun 对两个数组应用按元素运算  归一化
        % Training and Testing
        val_acc(k) = RF(val_trainX,val_trainY,val_testX,val_testY,TestModelParameters);

    end
```

```
    % Average the validation accuracy
    ValAcc = mean(val_acc);

    % Check if current configuration is the best
    if ValAcc > best_acc
        best_acc = ValAcc;
        best_mtry = mtry_range(p1);
    end
end

% Ue the best settings
ModelParameters.mtry = best_mtry;
```

The tuning code is shown as above pictures.

We run this algorithm in 2 datasets, and the result in shown as follow table.

Dataset	Best_mtry	Feature_num	Val_acc	Test acc
cardiotocography-3clases	17	21	0.934705882352941	0.953051643192488
image-segmentation	8	18	0.968073593073593	0.976190476190476

According to this result,we could find that this method has a good performance, therefore, we use this method in the following experiments.

Ntree is another important parameter in random forest, too many trees require too much calculation time while less tree could not show the advantage of random forest. What's more, using more trees can prevent the overfitting in some way. But when the number of trees exceed a certain number, it might not improve the accuracy so much

We use the the k-fold method and set ntree range from 1 to 500 to find the best ntree parameter in a dataset, the result is shown as below

Dataset	Best_ntree	Val_acc	Test acc
cardiotocography-3clases	453	0.945823655324891	0.965304313198484
image-segmentation	492	0.97807359023483	0.979047196509476

ii. Random Vector Functional Link(RVFL)

RVFL is a simple neural network, which has few parameter to tune.

γ is the regularization parameter,when γ is zero, the solution becomes the Moore-Penrose pseudoinverse. When γ is not zero, it is the Ridge Regression.

In order test the effect that γ has on the performance of RVFL, we test these two situation in 8 datasets using k-fold algorithm, as the following table show:

Dataset	Test_acc	Val_acc	Best γ
cardiotocography-3clases	0.875586854460094	0.884117647058824	0
image-segmentation	0.896103896103896	0.904220779220779	0
molec-biol-splice	0.804075235109718	0.786442006269593	1/c
ozone_Train	0.982283464566929	0.968441814595661	1/c
semeion	0.852664576802508	0.806874864454565	1/c
spambase	0.890336590662324	0.892119565217391	1/c
steel-plates	0.890336590662324	0.709407216494845	0
waveform-noise	0.865000000000000	0.839500000000000	1/c

The testing code is shown as follow:

```
cv_part = cvpartition(trainlabel,'KFold',4);    % Create indices for training/validat
for j=1:2
for k = 1:4
        % Collect training/validation sets
        val_trainX = traindata(cv_part.training(k),:);
        val_trainY = trainlabel(cv_part.training(k),:);
        val_testX = traindata(cv_part.test(k),:);
        val_testY = trainlabel(cv_part.test(k),:);
        % Data Normalisation
        mean_X = mean(val_trainX,1);
        std_X = std(val_trainX);
        std_X(std_X==0) = 1e-4;                 % For numerical stability
        val_trainX = bsxfun(@rdivide,val_trainX-repmat(mean_X,size(val_trainX,1),1),1),s
        val_testX = bsxfun(@rdivide,val_testX-repmat(mean_X,size(val_testX,1),1),1),std_
        %bsxfun 对两个数组应用按元素运算  归一化
        % Training and Testing
        [tmp_net]=RVFLtrain(val_trainX,val_trainY,5,j-1);
        tmp_y=RVFLtest(val_testX,tmp_net);
        val_acc(i,k) = length(find(tmp_y==val_testY))/size(tmp_y,1);
end
         % Average the validation accuracy
    ValAcc(i) = mean(val_acc(i,:));

    % Check if current configuration is the best
    if ValAcc(i) > best_acc(i)
        best_acc(i) = ValAcc(i);
        best_j(i)=j-1;
```

According to the test result, we could see that these two kind of γ has similar accuracy. In fact, Ridge Regression performs a little better than Moore-Penrose pseudoinverse. Therefore, we set γ =1/c in the following experiments.

The number of hidden neurons might also influence the performance of RVFL. Cause more numbers of neurons will boost the model complexity. However, the increasing number of neurons could compensated the problem that bring by the randomization range. Scaling down the range could avoid saturating the neurons but might degenerating the discrimination power of the random feature. Scaling up the range to enhance the discrimination power might risk at saturating the neurons.

In order test the effect that the number of neurons has on the performance of RVFL, we test 3 situation in 8 datasets, as the following table show:

Dataset	Acc(num=5)	Acc(num=50)	Acc(num=500)
cardiotocography-3clases	0.880281690140845	0.873239436619718	0.880281690140845
image-segmentation	0.887445887445888	0.898268398268398	0.896103896103896
molec-biol-splice	0.805642633228840	0.816614420062696	0.777429467084639
ozone_Train	0.982283464566929	0.982283464566929	0.982283464566929
semeion	0.846394984326019	0.843260188087774	0.836990595611285
spambase	0.890336590662324	0.890336590662324	0.890336590662324
steel-plates	0.688946015424165	0.683804627249357	0.717223650385604
waveform-noise	0.871000000000000	0.865000000000000	0.841000000000000

According to the test result, we could find that different number of neurons has similar performance. In order to reduce the model complexity,we use neurons =5 in the following experiments.

C. Final Result

In this section, we will evaluate the performance of Random Forest algorithm and RVFL algorithm base on 8 UCI open-source datasets. The datasets is shown as follow:

Training data:

Dataset	Num of samples	Num of features	Num of classes
cardiotocography-3clases	1700	21	3
image-segmentation	1848	18	7
molec-biol-splice	2552	60	3
ozone_Train	2028	72	2
semeion	1274	256	10
spambase	3680	57	2
steel-plates	1552	27	7
waveform-noise	4000	40	3

Test data:

Dataset	Num of samples	Num of features	Num of classes
cardiotocography-3clases	426	21	3
image-segmentation	462	18	7
molec-biol-splice	638	60	3
ozone_Train	508	72	2
semeion	319	256	10
spambase	921	57	2
steel-plates	389	27	7
waveform-noise	1000	40	3

i. Evaluation of RF

We use the previous UCI datasets to evaluate the Random Forest algorithm, the testing result is shown below:

Dataset	Val_acc	Test_acc	Test_time	Train_acc	Train_time
cardiotocography-3clases	0.945823655324891	0.965304313198484	0.0402697	0.975294117647059	1.15429870
image-segmentation	0.97807359023483	0.979047196509476	0.0222719	0.987012987012987	1.761447500
molec-biol-splice	0.975369665361965	0.965367965367965	0.0657639	0.986677115987461	3.042648600
ozone_Train	0.988073590361965	0.982283464566929	0.0612958	0.986686390532544	2.880692900
semeion	0.897010819665217	0.884012539184953	0.0687514	0.992935635792779	5.057426300
spambase	0.905303030521730	0.926167209554832	0.1688671	0.977173913043478	5.673265300
steel-plates	0.749716494845361	0.699228791773779	0.0283885	0.941365979381443	2.898149300
waveform-noise	0.853050750536121	0.797000000000000	0.1000839	0.968250000000000	13.38071670

From the result, we could find that Random forest has high accuracy, but it seems to have many time to train the trees.

ii. Evaluation of RVFL

We then use the previous UCI datasets to evaluate the Random Vector Functional Link(RVFL) algorithm, the testing result is shown below:

Dataset	Val_acc	Test_acc	Test_time	Train_acc	Train_time
cardiotocography-3clases	0.872941176470588	0.896713615023474	0.0067908	0.883529411764706	0.0751494
image-segmentation	0.905303030303030	0.891774891774892	0.0076045	0.905844155844156	0.0781588
molec-biol-splice	0.786833855799373	0.808777429467085	0.0051370	0.813871473354232	0.0781588
ozone_Train	0.967948717948718	0.982283464566929	0.0042550	0.968934911242604	0.0072802
semeion	0.808449655961042	0.849529780564263	0.0042162	0.961538461538462	0.0242737
spambase	0.897010869565217	0.890336590662324	0.0072676	0.897554347826087	0.0067334
steel-plates	0.719716494845361	0.709511568123393	0.0061098	0.761597938144330	0.0890862
waveform-noise	0.850750	0.866000	0.0083969	0.866250	0.0050514

We could find that RVFL also have a good performance in these datasets,what's more, its takes less time to train the model.

iii. Wilcoxon rank sum test

We use Wilcoxon rank sum test to check if there are significant difference between the accuracy in RF and RVFL

p	h	stats
0.396891996891997	0	76.5000000000000

The result shows that there is a nearly 40% percent chance of being equal on average accuracy. And the h is 0 also means there is no significant different between these two algorithm.

D. Conclusion

Comparing these two algorithm, we can find that both of these algorithm have a good classification ability. However, RF perform better in some datasets while RVFL is easily computed and also out perform in some datasets. Therefore, we should choose a better classifier according to the reality problem and consider the requirement.

E. Reference

[1]. Tin Kam Ho, "Random decision forests," Proceedings of 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 1995, pp. 278-282 vol.1, doi: 10.1109/ICDAR.1995.598994.

[2]. Le Zhang and P.N. Suganthan. 2016. A comprehensive evaluation of random vector functional link networks. Inf. Sci. 367, C (November 2016), 1094–1105. DOI:https://doi.org/10.1016/j.ins.2015.09.025

[3]. P. N. Suganthan, “Random Vector Functioanl Link (RVFL) Neural Networks,” Course Slides of EE6227, School of EEE, Nanyang Technological University, Singapore, 2020.

[4]. D. Husmeier, “Random Vector Functional Link (RVFL) Networks,” Neural Networks for Conditional Probability Estimation, Springer, London, pp. 87-97, 1999.

[5]. R. Katuwal, P. N. Suganthan, and M. Tanveer, “Random Vector Functional Link Neural Network based on Ensemble Deep Learning,” arXiv preprint, arXiv:1907.00350, 2019