TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Assignment Cover Sheet

Full Name: Tao Shi

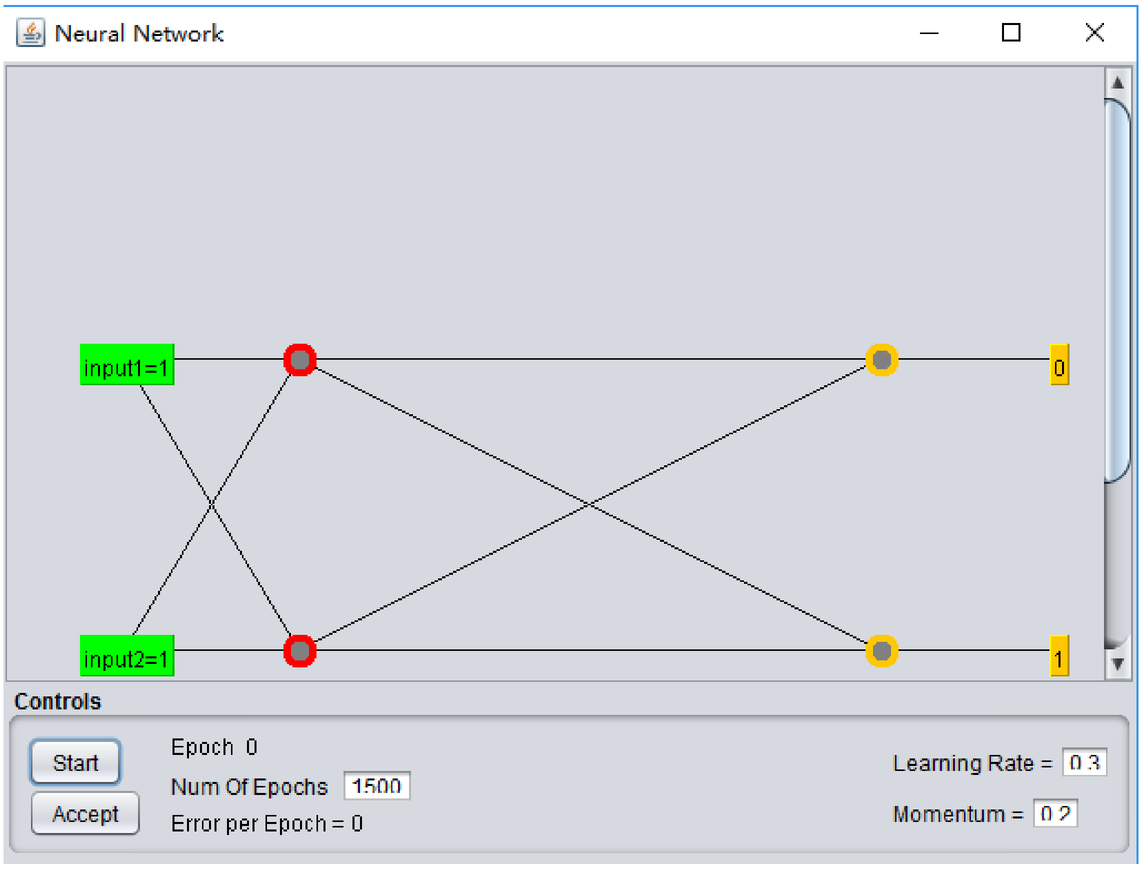
Student ID: 300409943 **Course:** COMP422

Assignment No.: 2 **Due Date:** 16 October at 11:59pm

Part 1: XOR Problem

1.1 Multilayer feed forward neural network

I use the classifier named as MultilayerPerceptron in Weka to create multilayer feed forward neural network for this task, the classifier applies back propagation algorithm to train the neural network. Please find the converted file xor.arff in the corresponding folder submitted.
The number of input nodes (attributes) is 2, and the number of output nodes (classes) is 2 as well.
The following default parameters of WEKA are implemented firstly:
Hidden layer: 1
Number of hidden nodes: 2
Learning rate: 0.3
Momentum: 0.2
Epochs: begin with 500
The training of neural network terminates after specified times of epochs. Until I set epoch as 1500 the training converges, and the accuracy of the neural network is 100%.
The visualized architecture of the neural network is as follows:



Epoch 0
Num Of Epochs 1500
Error per Epoch = 0
Learning Rate = 0.3
Momentum = 0.2

And the running result in Weka is as follows:

=== Run information ===

```
Scheme:       weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 1500 -V 0 -S 0 -E 20 -H a
-G -R
Relation:     xor
Instances:    4
Attributes:   3
              input1
              input2
              output
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Sigmoid Node 0
    Inputs    Weights
    Threshold    -0.8542972256665955
    Node 2    -3.984448827968652
    Node 3    2.7849299479658245
Sigmoid Node 1
    Inputs    Weights
    Threshold    0.8517254785281319
    Node 2    3.978704553822409
    Node 3    -2.7790829304304228
Sigmoid Node 2
    Inputs    Weights
    Threshold    -4.0014626037226915
    Attrib input1=1    -3.32602248274816
    Attrib input2=1    3.6164031745388887
Sigmoid Node 3
    Inputs    Weights
    Threshold    1.5154537065483
    Attrib input1=1    -2.145222399379249
    Attrib input2=1    2.1874736644111965
Class 0
    Input
    Node 0
Class 1
    Input
    Node 1



Time taken to build model: 2.36 seconds

=== Evaluation on training set ===
Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances           4               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.2195
Root mean squared error                  0.2308
Relative absolute error                 43.8985 %
Root relative squared error             46.1642 %
Total Number of Instances                4

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area
Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

 a b   <-- classified as
 2 0 | a = 0
 0 2 | b = 1
```

Because the training dataset is also used as test dataset in this problem, the reported accuracy is also the accuracy of the test dataset.
Then I try to experiment the following different parameters:
Learning rate: 0.8
Momentum: 0.5
After only 500 epochs, the training converges, and the accuracy of the neural network is 100%. And the running result in Weka is as follows:

```
=== Run information ===

Scheme:       weka.classifiers.functions.MultilayerPerceptron -L 0.8 -M 0.5 -N 500 -V 0 -S 0 -E 20 -H a -G
-R
Relation:     xor
Instances:    4
Attributes:   3
              input1
              input2
```

```
                    output
Test mode:      evaluate on training data

=== Classifier model (full training set) ===

Sigmoid Node 0
      Inputs      Weights
      Threshold      -2.4313992413026897
      Node 2      -5.670620722373995
      Node 3      5.315600584007766
Sigmoid Node 1
      Inputs      Weights
      Threshold      2.4309375783862874
      Node 2      5.669201246896655
      Node 3      -5.314477478296033
Sigmoid Node 2
      Inputs      Weights
      Threshold      -4.465498686474746
      Attrib input1=1      -4.096852286745907
      Attrib input2=1      4.2262153255924835
Sigmoid Node 3
      Inputs      Weights
      Threshold      3.106919071153539
      Attrib input1=1      -3.427275296481757
      Attrib input2=1      3.43889372847344
Class 0
      Input
      Node 0
Class 1
      Input
      Node 1


Time taken to build model: 2.27 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances               4                    100        %
Incorrectly Classified Instances             0                      0        %
Kappa statistic                              1
```

```
Mean absolute error                    0.0736
Root mean squared error                  0.0743
Relative absolute error               14.7259 %
Root relative squared error            14.8562 %
Total Number of Instances                4

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall    F-Measure   MCC       ROC Area   PRC Area
Class
               1.000     0.000     1.000       1.000     1.000       1.000     1.000      1.000      0
               1.000     0.000     1.000       1.000     1.000       1.000     1.000      1.000      1
Weighted Avg.    1.000     0.000     1.000       1.000     1.000       1.000     1.000      1.000

=== Confusion Matrix ===

 a b   <-- classified as
 2 0 | a = 0
 0 2 | b = 1
```

The experience demonstrates that it will speed the training process to increase the values of learning rate and momentum. The reason is that appropriately increasing learning rate and momentum can help weights of nodes converge. Concretely, the training process of back propagation in multilayer feed forward neural network is as follows:

```
1. Let $\eta$ be the learning rate;
2. Set all weights to smaller random values;
3. Until total error is small enough or the specified epochs is reached (in this case), repeat:
  -For each input example:
    •Feed forward pass to get predicted outputs;
    •Compute $\beta_z = d_z - o_z$ for each output node;
    •Compute $\beta_j = \sum_k w_{j \to k} o_k (1 - o_k) \beta_k$
    •Compute the weight changes $\Delta w_{i \to j} = \eta o_i o_j (1 - o_j) \beta_j$
  -Add up weight changes for all input examples;
  -Change weights.
```

1.2 Genetic Programming

The package of ECJ is applied in this problem.
I suppose the terminal set is x1 and x2 (function X1 and X2 in ECJ), and set {+, -, *, % (i.e. protected division, which returns 0 if denominator is 0)} as the function set, because the problem is not complicated.

The fitness function is overall classification accuracy, which is the number of training examples classified correctly (#correct), over the total number of training examples (#total). This can be expressed as:

$$\text{fitness} = \frac{\#correct}{\#total}$$

The most default parameter values of ECJ are applied in the problem except the rates of reproduction, crossover and mutation:
Population = 1024. Generations = 51, tournament size = 7, ramped half-and-half method.
In term of max depth (program size): the crossover and mutation are 17, grow for subtree mutation is 5 (as many as the minimal depth), and the halfbuilder is 6(minimal depth is 2 by the way).
The modified reproduction rate is set as 5%, crossover rate: 90%, mutation rate: 5%.
In the problem, the stopping criteria is at generation 51 or the standardized error is 0, and the outfile offers the Best Individual of Run as follows:
1. (x2 - x1) * (x2 - x1)
2. ((x2 - x2) + (x2 - x1)) * ((x2 - x1) * (x2 + x1))
3. (((x1 * x2) * (x2 + x1)) - ((x2 + x2) + (x1 - x2))) % (((x1 - x2) + (x2 % x1)) % ((x2 - x1) - (x2 * x1)))

1.3 Comparison and analysis

The two methods both achieve 100% accuracy, since the problem has very small search space. Any classifier that can solve non-linearly-separable problems will generate the solution of classification.

While the symbolic description of the model about the relation of input and output generated by GP technique is straightforward and interpretative compared with the black box of neural networks.

Part 2: Digit Recognition

2.1 Neural network

I still use the classifier named as MultilayerPerceptron in Weka to create neural network for the task 4-7, whose noise ratios are 15%, 20%, 30% and 40% respectively.
Please find the converted file digits15.arff, digits20.arff, digits30.arff and digits40.arff in the corresponding folder submitted.
The number of input nodes (attributes) is 49, and the number of output nodes (classes) is 10.
The following default parameters of WEKA are implemented firstly:
Hidden layer: 1

Number of hidden nodes: 29, calculated by (Number of Attributes + Number of Classes)/2
Learning rate: 0.3
Momentum: 0.2
Epochs: 500
The training of neural network terminates after 500 epochs, and the accuracies of the four tasks on training data are all close to 100%.
The accuracies of 10 runs on test data are in the following table.

No. of run	Random split seed	Task 4	Task 5	Task 6	Task 7
Run1	1	94.2%	89.2%	83.4%	73.6%
Run2	2	93.2%	90.4%	84.2%	80%
Run3	3	95.8%	90%	84.4%	76.6%
Run4	4	94.6%	88%	87%	76.4%
Run5	5	93.6%	89.2%	83.6%	74.4%
Run6	6	93.6%	90%	85%	76.4%
Run7	7	94%	88.6%	86.2%	77.2%
Run8	8	92.2%	89.2%	88%	75.8%
Run9	9	93.2%	89.4%	85.6%	73.2%
Run10	10	93.8%	90%	86%	77%
Average of 10 runs		**93.82%**	**89.40%**	**85.34%**	**76.06%**

2.2 Nearest neighbour method

I use the classifier named as IBk (Instance-based learning) in Weka to realize the nearest neighbor method for the same tasks.
The method is based on Euclidean distance, and the unseen instance is classified as the class of the nearest neighbor. The accuracies of 10 runs in Weka are in the following table.

No. of run	Random split seed	Task 4	Task 5	Task 6	Task 7
Run1	1	93.8%	88%	76%	70.4%
Run2	2	93.4%	90.4%	75.6%	68.4%
Run3	3	94.6%	89.2%	75.6%	72.6%
Run4	4	94%	90.2%	77%	68%
Run5	5	93.2%	87.8%	78.2%	68.8%
Run6	6	92.2%	89.4%	77.6%	70.4%
Run7	7	93.6%	89.4%	81%	69.4%
Run8	8	92.8%	85.6%	75.6%	68%
Run9	9	93.4%	86.6%	78.6%	67%
Run10	10	92.8%	84.6%	77.8%	69%
Average of 10 runs		**93.38%**	**88.12%**	**77.30%**	**69.20%**

If k is set as more than 1, the accuracies of *task 7* are in the following table.

No. of run	Random split seed	k=1	K=3	K=7	k=11
Run1	1	70.4%	74%	75.4%	76.8%
Run2	2	68.4%	72.6%	75.4%	78.2%
Run3	3	72.6%	76%	76.6%	77.8%
Run4	4	68%	70.6%	75%	76.6%
Run5	5	68.8%	70.2%	74.4%	76.2%
Run6	6	70.4%	76.2%	77.4%	78.2%
Run7	7	69.4%	74.8%	77.2%	78.4%
Run8	8	68%	73%	75.8%	77.6%
Run9	9	67%	71.4%	74.4%	74%
Run10	10	69%	73%	76%	78.2%
Average of 10 runs		**69.20%**	**73.18%**	**75.76%**	**77.20%**

2.3 Comparison and analysis

Method	Task 4	Task 5	Task 6	Task 7
Neural network	**93.82%**	**89.40%**	**85.34%**	**76.06%**
Nearest neighbour method	**93.38% (k=1)**	**88.12% (k=1)**	**77.30% (k=1)**	**77.2% (k=11)**

In terms of the average accuracies of two methods in above table, the performances of the two methods are similar when the noise ratio is relatively low, such as task 4 (15%) and task 5 (20%). However, the performance of nearest neighbor method will become worse than that of neural network when the noise ratio increases. However, it is notable that if the k is increased, the accuracy will increase as well. When k=11, the accuracy of k-nearest neighbor method is similar to that of neural network again.

Part 3: Symbolic Regression Problem

The package of ECJ is also applied in this problem.
I suppose the terminal set is numerically valued attribute and random number r (function X and RegERC in ECJ). The X is the independent variable of the problem and the random number is used to provide flexibility for the building of GP tree.
I set {+, -, *, % (protected division, which returns 0 if denominator is 0), sine and If conditional statement} as the function set, in my program the relevant functions are {Add, Sub, Mul, Div, Sin and If}. The sine function can offer the calculation ability of trigonometry that is a very popular arithmetical operation. Notable that if condition function is powerful to cope with piecewise functions, which is also popular in mathematical calculation. The function has three children, the first one is the evaluation according to the condition, if the condition meets, the function returns the value of the second child, otherwise returns the value of the third child. In this problem, I set the condition is "greater than 0".

In order to create training data set, the program generate 1000 double numbers randomly, and the expected results are calculated according to the given formula.

The fitness function can be described as the mean squared error (MSE), the error in the problem is the difference between the ProgOut calculated by the program evolved by GP and output variable y according to each input variable x given in the question. The less the MSE is, the better the performance of the program is.

I create 201 fitness cases, the X value is from -50 to 50, the interval is 0.5, the corresponding output is calculated from the given equation.

The most default parameter values of ECJ are applied in the problem except the rates of reproduction, crossover and mutation:
Population = 1024. Generations = 51, tournament size = 7, ramped half-and-half method.
In term of max depth (program size): the crossover and mutation are 17, grow for subtree mutation is 5 (as many as the minimal depth), and the halfbuilder is 6(minimal depth is 2 by the way).
And reproduction rate is set as 10%, crossover rate: 80%, mutation rate: 10%. The experiment shows better performance.
In the problem, the stopping criteria is at generation 51 or the standardized error is 0, and the outfile offers the Best Individual of Run as follows, they all can successfully predict all the fitness cases:

```
if(if(x, 0.6025661876208679 - -0.7339139880349161, if(x, 0.6025661876208679 -
-0.7339139880349161, -0.8599332234285835)) - (-0.521594796843289 % x),
(sin(x) - sin(-0.521594796843289 % x)) - (-0.521594796843289 % x), (x * x) + ((x
+ (x - -0.8599332234285835)) - (-0.8482800686656553 - (0.6025661876208679 -
-0.7339139880349161))))
```

```
if(-0.014337125428467878 + x, (0.9253746739004061 %
(-0.014337125428467878 + x)) + sin((x - x) + x), (x * x) + ((x -
-0.8312655177381636) + (((0.6145060062096901 + x) - -0.8312655177381636) -
(sin(-0.3960442501602297) - if(x, x, 0.380304822286029)))))
```

```
if(-0.014337125428467878 + x, (0.9253746739004061 %
(-0.014337125428467878 + x)) + sin(x), if(-0.014337125428467878 + x, x + ((if(x,
sin(0.6432070971229926) + sin(-0.8640961374388396), if(0.381891976231137 %
x, -0.3313464663759331 + -0.9092375574774394, sin(x))) - if(if(x, x, x), x,
if(-0.21853974034389423, x, x))) * (-0.014337125428467878 + x)), (x * x) + ((x -
-0.8312655177381636) + (((0.6145060062096901 + x) - -0.8312655177381636) -
(sin(-0.3960442501602297) - (0.24175159597525298 +
0.10659374314499037))))))
```

Conclusions:

The complicated problem proves the excellent performance of GP. By choosing appropriate functions and parameters, GP will automatically evolve the model structure and the corresponding coefficients without domain knowledge.

The choice of functions and parameters plays key role in solving the problem. If there is no sine or if condition function, GP will not evolve perfect regression results. On the other hand, even though the appropriate functions and parameters are decided, GP cannot guarantee evolve perfect regression results in every run.

Part 4: Function Optimisation

The package of PSO coded by Doctor Bing is applied in this problem.

4.1 Rosenbrock's function

Based on the default parameters, common settings and experimental performance, I choose the following appropriate values:
c1 = c2 = 1.49618
w = 0.7298
population size = 1000
fitness function = f1(x), the smaller the better
particle encoding: xi = (xi1, xi2, xi3, …, xi20)
topology type: ring topology (i.e. lbest), and set the particle is influenced by its local 10 neighbours.
stopping criterion: 1000 iterations
After repeat the experiments 30 times, the mean and standard deviation of my results are as follows:

```
Best result of 30 runs is:0.03315546598256223
Average of best results of 30 runs is:3.3196208199541966
Standard Deviation of best results of 30 runs is:3.1248627317750524
```

4.2 Griewanks's function

I apply empirically appropriate values for the following parameter:
c1 = c2 = 1.49618
w = 0.7298
population size = 1000
fitness function = f1(x), the smaller the better
particle encoding:
xi = (xi1, xi2, xi3, …, xi20) when D = 20
xi′ = (xi1′, xi2′, xi3′, …, xi50′) when D = 50
topology type: ring topology (i.e. lbest), and set the particle is influenced by its local 10 neighbours.
stopping criterion: 1000 iterations
When D = 20 and after repeat the experiments 30 times, the mean and standard

deviation of my results is as follows:

```
Best result of 30 runs is:0.0
Average of best results of 30 runs is:0.0
Standard Deviation of best results of 30 runs is:0.0
```

When D = 50 and after repeat the experiments 30 times, the mean and standard deviation of my results is as follows:

```
Best result of 30 runs is:1.0410709405661578E-8
Average of best results of 30 runs is:2.6412838723584762E-8
Standard Deviation of best results of 30 runs is:9.832554184484756E-9
```

4.3 Conclusions

According to the experiment, PSO can offer excellent performance on searching the minimum of the two functions without mathematical or statistical assumption.
To achieve the excellent performance, the appropriate parameters need to be determine carefully:
1. I increase the numbers of population size and iteration greatly, which is useful to improve the performance by observation;
2. By changing the topology type from star to ring, the particles move to the local best positions rather than the globally best position, then the search space become larger. It improves the performance in this problem although it might miss the global optimal.
3. The mean and standard deviation are smaller when D = 20, comparing with D = 50. That’s because it is more difficult to find the minimum when there are more variables in each vector.

Part 5: Feature Construction

5.1 Before feature construction

I apply the classifier named as NaiveBayes and J48 (decision tree) in Weka to measure the classification performance, the results using 10-fold cross-validation are as follows:

	Balance Scale	**Wine Recognition**
Naive Bayes	90.88%	96.6292%
Decision Tree	76.8%	93.8202%

5.2 The principle of GP-based feature construction

I use K-CV for wrapper features construction to avoid bias. Concretely, 10-fold CV in the outer loop and 3-fold CV in the inner loop.
In the outer loop, the original data is split into 10 folds with equal size, each time 9 folds as training set are used to construct features, and the other 1-fold as test set. Then repeat the process 10 times, and the average accuracy is calculated as final performance.

In the inner loop, the 3-fold CV is implemented by the package of ECJ. The training data of outer loop is split into 3 folds with equal size, each time 2 folds as training set are used by GP algorithm to construct features, and the other 1-fold as test set. Then repeat the process 3 times, and the average accuracy is calculated as the performance of the classifier (GP tree), see the following fitness function:

$$\frac{1}{3}\sum_{i=1}^{3}\frac{\#corret\ of\ ith\ fold\ data}{\#total\ of\ ith\ fold\ data}$$

5.3 Feature construction about Balance Scale problem

The best individual is as follow, whose accuracies of training data and test data are both 100% (% is protected division, which returns 0 if denominator is 0):

(- x3 (% (* x1 x2) x4))

After feeding the constructed feature to original classifiers, the following table shows the comparison of performance:

	Before feature construction	**After feature construction**
Naive Bayes	90.88%	99.84%
Decision Tree	76.8%	100%

5.4 Feature construction about Wine Recognition problem

The result from GP-based feature construction is unsatisfactory, so I choose the following relatively good program to transform the data (% is protected division, which returns 0 if denominator is 0).

```
(% (+ x4 (% (- (+ x1 x6) (% x12 x7)) x7))
    (- (+ x1 x12) (% x4 (- (- (+ x1 (- (+ x1
       (* x11 (- (+ x1 (* x11 (+ x4 (* x6 x12))))
          (% (- (- (- (+ x1 (+ x1 x6)) (% (% x12 x7)
             (% x12 x7))) 2.0) 2.0) (* x6 x12))))) (-
       (- (+ x1 (+ x1 x6)) (% (% x12 x7) (+ x1 x6)))
       2.0))) (% (% (- (+ x1 x6) (% (+ x1 x6) x7))
       x7) (+ 2.0 (+ 2.0 (- (* x6 x12) 2.0)))))
```

After feeding the constructed feature to original classifiers, the following table shows the comparison of performance:

	Before feature construction	After feature construction
Naive Bayes	96.6292%	97.7528%
Decision Tree	93.8202%	92.6966%

5.5 Conclusions

In most situations, the performance is enhanced in both classifiers and both problems by feature construction, especially the Balance Scale problem, the classifiers learning form 4 original features are even not as good as the single new constructed feature that makes the two classifiers achieve nearly 100% accuracies. The Wine Recognition problem with 13 features already has good accuracies before feature construction, while adding the new constructed features, the performance of Naïve Bayes classifier is improved further.

The experiment proves the power of the GP-based feature construction algorithm. When the problem is not too complicated, the single constructed feature can find out the mathematical and logical relationship between many original features without domain knowledge to reduce the dimensionality. On the other hand, the flexible method can construct multiple features from a single tree, which is also helpful to improve the performance of the classifiers.

Part 6: Feature Selection

6.1 Wisconsin Breast Cancer problem

Firstly, I apply the classifier named as NaiveBayes in Weka to measure the classification performance using all the features and 5 top single-ranked features, the result by 10-fold cross-validation is as follows:

	WEKA tool	Feature list	Accuracy
No ranking algorithm		All	92.9701%
Information Gain	InfoGainAttributeEval	23, 24, 21, 28 and 8	94.2004%
Pearson's correlation coefficient	CorrelationAttributeEval	28, 23, 8, 21 and 3	94.0246%

According to the results in the above table, the performance of the classifier using 5 top ranked features is better than using all the features, which means not all the features are relevant to the class and irrelevant features may reduce the performance.
Then I apply wrapped approach to select 5 features, also use NaiveBayes in Weka as the wrapper classifier. The top 5 ranked features are feature12, 15, 19, 22, 23 and 25, the result by 10-fold cross-validation is as follows:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          549               96.4851 %
Incorrectly Classified Instances         20                3.5149 %
Kappa statistic                           0.9247
Mean absolute error                         0.075
Root mean squared error                       0.1769
Relative absolute error                  16.0429 %
Root relative squared error              36.5911 %
Total Number of Instances                  569

=== Detailed Accuracy By Class ===

                 TP  Rate   FP  Rate   Precision   Recall    F-Measure   MCC        ROC  Area   PRC  Area
Class
                 0.975      0.052      0.969        0.975      0.972       0.925      0.988       0.991       1
                 0.948      0.025      0.957        0.948      0.953       0.925      0.988       0.984       2
Weighted Avg.      0.965      0.042      0.965        0.965      0.965       0.925      0.988       0.989

=== Confusion Matrix ===

   a    b    <-- classified as
 348    9 |    a = 1
  11 201 |    b = 2
```

In this problem, the wrapped approach selects the different features from the filter approach, and better performance.

6.2 Sonar problem

Firstly, I apply the classifier named as NaiveBayes in Weka to measure the classification performance using all the features and 5 top single-ranked features, the result by 10-fold cross-validation is as follows:

	WEKA tool	**Feature list**	**Accuracy**
No ranking algorithm		All	67.7885%
Information Gain	InfoGainAttributeEval	11, 12, 9, 10 and 13	70.1923%
Pearson's correlation coefficient	CorrelationAttributeEval	11, 12, 49, 10 and 45	65.8654%

According to the results in the above table, the performance of the classifier using 5 top ranked features sometimes is better than using all the features (information gain

as feature ranking algorithm), sometimes not (Pearson's correlation coeddicient as feature ranking algorithm), which means the selected feature ranking algorithm influences the performance of the classifier built from the features.
Then I apply wrapped approach to select 5 features, also use NaiveBayes in Weka as the wrapper classifier. The top 5 ranked features are feature 12, 17, 18, 24 and 39, the result by 10-fold cross-validation is as follows:

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          152               73.0769 %
Incorrectly Classified Instances         56               26.9231 %
Kappa statistic                           0.4584
Mean absolute error                         0.3291
Root mean squared error                       0.4356
Relative absolute error                  66.1113 %
Root relative squared error              87.3005 %
Total Number of Instances                 208

=== Detailed Accuracy By Class ===

                 TP  Rate   FP  Rate   Precision   Recall    F-Measure   MCC        ROC  Area   PRC  Area
Class
                 0.701      0.243      0.716       0.701     0.708       0.458     0.796       0.789       1
                 0.757      0.299      0.743       0.757     0.750       0.458     0.796       0.783       2
Weighted Avg.      0.731      0.273      0.731       0.731     0.731       0.458     0.796       0.786

=== Confusion Matrix ===

  a   b    <-- classified as
 68 29 |   a = 1
 27 84 |   b = 2
```

In this problem, the wrapped approach selects the different features from the filter approach, also better performance.

6.3 Conclusions

The experimental result demonstrates the performance of wrapper approach outweighs the performance of filter method. Because the wrapper method includes a classification algorithm in the evaluation procedure, and the goodness of a feature subset is evaluated by the classification performance. Usually the combination of features is evaluated and selected. While the filter method ignores the performance of the selected features on a classification algorithm. Usually the importance (goodness) of each feature is measured individually and ranked in the descending order of their

importance. Lastly the m top (most important) features are selected, and their interaction is not considered.
In summary, the wrapper method offers higher classification accuracy but higher computational cost.

Part 7: Data Processing for Classification

7.1 Experimental design method

I apply 10-fold Cross Validation to split the dataset, for there are only 155 instances available. The procedure of the method is as follows:
1. Split the whole dataset to 10 folds with equal size;
2. Use 1-fold as test set, and the other 9 folds as training set;
3. Repeat 10 times to make sure each fold has a chance to be the test set
4. Average the 10 test performances (e.g. accuracy rates) as the result.

7.2 Analysis of potential problems

There are three main kinds of potential problems in the raw data.
1. Data format
(a) In order to apply WEKA to solve the problem, the raw data (hepatitis.data) has to be transferred to arff format.
(b) The tool named as ARFF-Viewer in WEKA is helpful to the format conversion.
(c) Please find converted file hepatitis.arff.
(d) After the step, the data is prepared for the KDD tool.
2. Necessity of discretization
(a) According to the file hepatitis.names, there are 19 features about hepatitis, including continuous/real data and discrete data.
The 13 discrete features are SEX, STEROID, ANTIVIRALS, FATIGUE, MALAISE, ANOREXIA, LIVER BIG, LIVER FIRM, SPLEEN PALPABLE, SPIDERS, ASCITES, VARICES and HISTOLOGY, which can be applied directly.
The left 6 continuous features are AGE, BILIRUBIN, ALK PHOSPHATE, SGOT, ALBUMIN and PROTIME, which need to be discretized to improve learning speed and accuracy.
(b) I apply the filter "unsupervised-Discretize" in WEKA to solve the problem. Concretely I select equal-width discretization on the feature AGE and set N as 7, then the width of intervals is about 10 ((78-7)/10≈10). On the other hand, I select equal-depth discretization (equal-frequency in WEKA) on other 5 continuous features and set N as 7 as well.
(c) Please find converted file hepatitis-discrete.arff.
(d) After discretization, i.e. the data reduction mechanism, the large domain of numeric values is diminished to a subset of categorical values (max 7 in my experiment). As a result, the data is available to more DM algorithms which can only deal with discrete attributes and improve performance in learning speed and accuracy.

3. Missing data
(a) There are 167 missing data related to 15 features in the raw data, the missing data is completely at random. The missing rates of the 15 features are as follows (decreasing sequence): PROTIME (43%), ALK PHOSPHATE (19%), ALBUMIN (10%), LIVER FIRM (7%), LIVER BIG(6%), BILIRUBIN(4%), SPLEEN PALPABLE, SPIDERS, ASCITES, VARICES, SGOT(3%), STEROID, FATIGUE, MALAISE and ANOREXIA (1%)
(b) For the 10 features whose missing rates are less than 5%, I apply the deletion approach (instance filter RemoveWithValues in WEKA) to omit the instances (sum 12). While for the other 5 features whose missing rates are greater than 5%, I apply the mode imputation approach (feature filter ReplaceMissingValue in WEKA) to fill missing values with most frequent value.
(c) Please find converted file hepatitis-impute.arff.
(d) After the preprocessing, the left 143 instances without missing data are generated.

7.3 Classification algorithm selection

As the classification for Hepatitis diagnosis, I apply decision tree as classification algorithm, for it is very straightforward to understand and interpret, which can be combined with relevant expert knowledge to help the doctors diagnose the Hepatitis.

7.4 Result

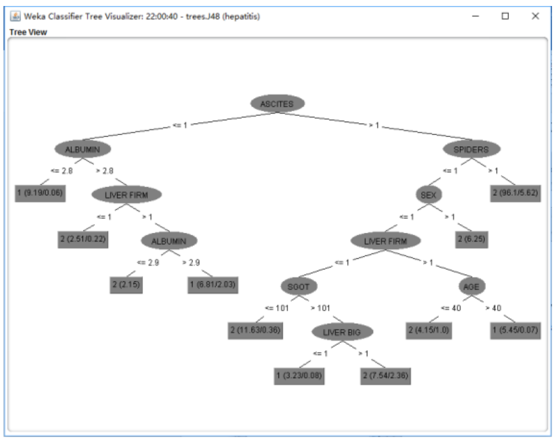
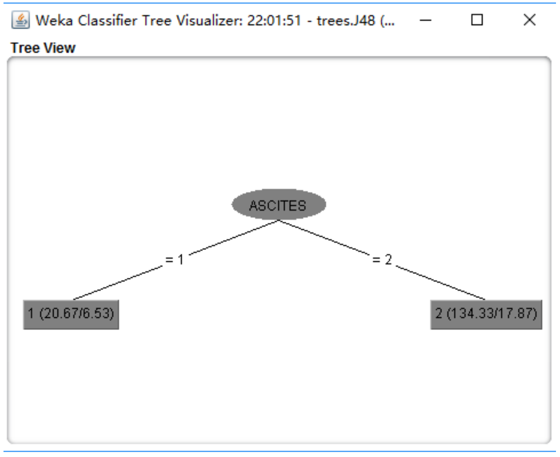
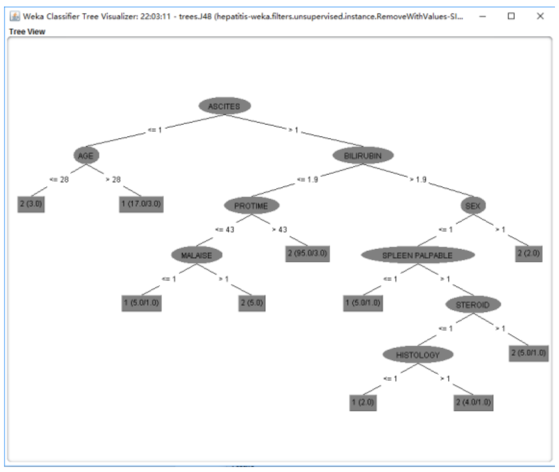
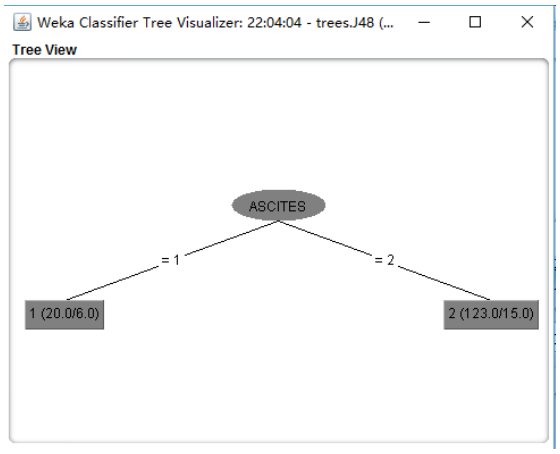
I apply J48 and 10-fold Cross Validation in WEKA to perform the classification, and the accuracies are as follows:

Dataset	Number of instances	Accuracy
Raw data	155	74.8387%
Data after discretization	155	80.6452%
Data after imputation	143	74.8252%
Data after discretization and imputation	143	85.3147%

In terms of above table, discretization can improve the performance comparing with raw data, while only filling the missing does not work obviously. Note that the combination of discretization and imputation achieve best performance among the four kinds of situation, which demonstrate the contribution of imputation (mainly because of omitting some instances with missing data). In addition, the discrete data also accelerates the build of model (decision tree), the time taken to build model is 0 second in WEKA, comparing with 0.13 second using raw data. If there are large number of instances and features, the learning time will be an issue.

In conclusion, the quality of input data can drastically affect the learning performance.

The decision trees learnt from above four datasets are as follows:

Raw data	**Data after discretization**
	
Data after imputation	**Data after discretization and imputation**
	

The visualization of decision trees demonstrates that discrete data can produce shorter and more compact model, which is helpful to simplify the classification in practical work, e.g. the feature ASCITES is the key indicator in Hepatitis diagnosis.