

Problem Set 4

1. Create dataset using code below. Let the results only depend on the first attribute.

```
import random
import csv

def create_dataset():
    attr_size = 50
    data_size = 1000
    train_set = [ [ 0 for j in range(attr_size) ] for i in range(data_size) ]
    for i in range(data_size):
        for j in range(attr_size):
            train_set[i][j] = random.uniform(j,j+1)
        if train_set[i][0] < 0.5:
            train_set[i].append(0)
        else:
            train_set[i].append(1)

    with open("./train.csv", "wb") as f:
        writer = csv.writer(f)
        writer.writerows(train_set)
```

Accuracy of J48:

The screenshot shows the Weka Classifier window with the 'Classify' tab selected. The classifier chosen is 'IBk' with parameters: -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-l". The test options are set to 'Cross-validation' with 10 folds. The classifier output shows a summary of performance metrics and a detailed accuracy by class table.

Classifier output

=== Summary ===

Correctly Classified Instances	998	99.8999 %
Incorrectly Classified Instances	1	0.1001 %
Kappa statistic	0.998	
Mean absolute error	0.001	
Root mean squared error	0.0316	
Relative absolute error	0.2002 %	
Root relative squared error	6.3278 %	
Total Number of Instances	999	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
0	0.998	0	1	0.998	0.999	0.999
1	1	0.002	0.998	1	0.999	0.999
Weighted Avg.	0.999	0.001	0.999	0.999	0.999	0.999

=== Confusion Matrix ===

a	b	<-- classified as
502	1	a = 0
0	496	b = 1

Accuracy of IBK:

The screenshot shows the Weka GUI with the 'Classify' tab selected. The classifier is 'IBk' with parameters '-K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-l'. The test options are set to 'Cross-validation' with 'Folds' set to 10. The classifier output shows the following summary:

Summary		
Correctly Classified Instances	662	66.2663 %
Incorrectly Classified Instances	337	33.7337 %
Kappa statistic	0.3251	
Mean absolute error	0.3377	
Root mean squared error	0.5802	
Relative absolute error	67.5424 %	
Root relative squared error	116.0345 %	
Total Number of Instances	999	

The detailed accuracy by class is as follows:

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
(Nom) 0	0.682	0.357	0.66	0.682	0.671	0.66
(Nom) 1	0.643	0.318	0.666	0.643	0.654	0.66
Weighted Avg.	0.663	0.338	0.663	0.663	0.663	0.66

The confusion matrix is:

```

a  b  <-- classified as
343 160 | a = 0
177 319 | b = 1

```

The difference in accuracy is $99.89\% - 66.26\% = 33.63\%$

2. Create dataset using code below. Let the results depend on the Covariance of attributes.

```

import random
import csv
import numpy as np

def create_dataset():
    attr_size = 2
    data_size = 1000
    train_set = [ [ 0 for j in range(attr_size) ] for i in range(data_size) ]
    for i in range(data_size):
        for j in range(attr_size):
            train_set[i][j] = random.uniform(0,30)
        if np.cov([train_set[i][0], train_set[i][1]]) > 50:
            train_set[i].append(0)
        else:
            train_set[i].append(1)

    with open("./2-train.csv", "wb") as f:
        writer = csv.writer(f)
        writer.writerows(train_set)

```

Multilayer Perceptron:

The screenshot shows the Orange3 software interface with the 'Classify' tab selected. The classifier chosen is 'NaiveBayes'. The 'Test options' section shows 'Cross-validation' selected with 'Folds' set to 10. The 'Classifier output' section displays the following statistics:

Metric	Value	Percentage
Correctly Classified Instances	924	92.4925 %
Incorrectly Classified Instances	75	7.5075 %
Kappa statistic	0.8489	
Mean absolute error	0.0915	
Root mean squared error	0.213	
Relative absolute error	18.6228 %	
Root relative squared error	42.9855 %	
Total Number of Instances	999	

Below the statistics, the 'Detailed Accuracy By Class' table is shown:

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
0	0.963	0.104	0.876	0.963	0.917	0.98
1	0.896	0.037	0.969	0.896	0.931	0.98
Weighted Avg.	0.925	0.066	0.929	0.925	0.925	0.98

The 'Confusion Matrix' is also displayed:

```

a  b  <-- classified as
417 16 | a = 0
 59 507 | b = 1
  
```

The 'Result list' on the left shows several entries, with '21:26:10 - functions.MultilayerPerceptron' selected.

Naïve Bayes:

The screenshot shows the Orange3 software interface with the 'Classify' tab selected. The classifier chosen is 'NaiveBayes'. The 'Test options' section shows 'Cross-validation' selected with 'Folds' set to 10. The 'Classifier output' section displays the following statistics:

Metric	Value	Percentage
Correctly Classified Instances	588	58.8589 %
Incorrectly Classified Instances	411	41.1411 %
Kappa statistic	0.1032	
Mean absolute error	0.4664	
Root mean squared error	0.4753	
Relative absolute error	94.9516 %	
Root relative squared error	95.9105 %	
Total Number of Instances	999	

Below the statistics, the 'Detailed Accuracy By Class' table is shown:

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
0	0.245	0.148	0.558	0.245	0.34	0.72
1	0.852	0.755	0.596	0.852	0.701	0.72
Weighted Avg.	0.589	0.492	0.579	0.589	0.545	0.72

The 'Confusion Matrix' is also displayed:

```

a  b  <-- classified as
106 327 | a = 0
 84 482 | b = 1
  
```

The 'Result list' on the left shows several entries, with '21:26:21 - bayes.NaiveBayes' selected.

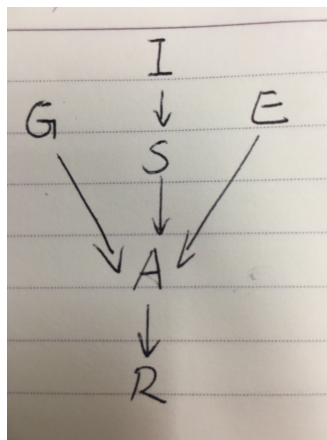
The difference in accuracy is $92.49\% - 58.85\% = 33.64\%$

3. Genetic Algorithm, Gradient Decent & Hill Climbing

1. Problems that have local minimum or maximum, e.g. N-Queue Problem, is more easily solved by Genetic Algorithm, but is not well-suit for Hill Climbing and Gradient Decent. The reason that N-Queue problem is well-suit for Genetic Algorithm is that successor state is generated by combining two best parent states, which will get the best solution. The reason that N-Queue problem is not well-suit for Hill Climbing and Gradient Decent is that N-Queue has local minimum state.
2. Problems like finding minimum value in a high dimensional function is more easily solved by Gradient Decent. Because we can easily find the direction that the function is descending most quickly using Gradient Decent, which helps us get the solution. However, Genetic Algorithm will be very slow in these problems and Hill Climbing is also time consuming because of the high dimension.
3. Problem like finding minimum value in a discrete function is more easily solved by Hill Climbing. Because we can easily find the maximum value searching follow the trend of the function. However, Genetic Algorithm will search states more randomly which makes it inefficient. Gradient Decent will perform bad because discrete function has no gradient.

4. Solutions:

1. As shown in figure below.



2. 1 for $P(I)$, $P(E)$ and $P(G)$; 2 for $P(S|I)$; 8 for $P(A|S,E,G)$ and 2 for $P(R|A)$. The minimum number of probabilities we need is 15.
3. There are six binary variables without independencies, the number of parameters we need is $2^6 - 1 = 63$.
4. Yes, E is independent of G if we are not given the value of any other variables. We know that $P(E,G,A) = P(E)*P(G)*(A|E,G)$. If we are not given the value of A, we will add them up with all possible value of A. Then we will get $P(E,G) = P(E)*P(G)$, which means E and G are independent.
5. No. Once we know the value of A, E is no longer independent of G. Now we have $P(E,G|A) = P(E,G,A)/P(A) = P(E)*P(G)*P(A|E,G)/P(A)$, which is not equal to $P(E)*P(G)$. Thus E is not independent of G given A.

5.
$$P(C) = P(C|A,B)*P(A)*P(B) + P(C|\neg A,B)*P(\neg A)*P(B) + P(C|A,\neg B)*P(A)*P(\neg B) + P(C|\neg A,\neg B)*P(\neg A)*P(\neg B) = 0.35 + 0.35 + 0.105 + 0.045 = 0.85.$$