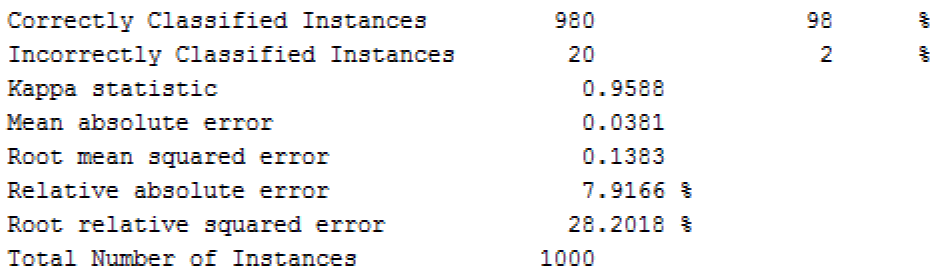
Roman Grigorii

Homework 4

Problem 1.

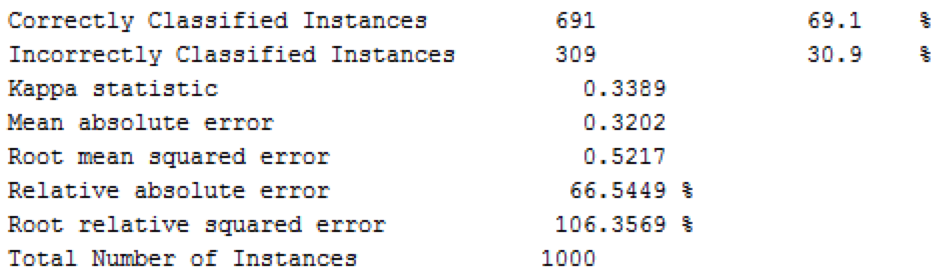
I created a data with 49 nominal attributes, 48 of which were names of soccer players and could hold two values, absent or present, and the last attribute with a nominal value of ‘won’ or ‘lost’ based on whether the team has won a soccer game when the said players have or haven’t shown up to the game. I forced a trend into my data, which made sure that only a few star players made a significant contribution to the game’s outcome. The way I did this was by converting the Boolean values marking the presence of a player, to a 16 bit number, where the first player corresponded to the most significant bit of that number. This used up 16 players, but ignored the rest of 32. So in the end, I have generated a table of players and their games, where the first 16 players had any importance to winning any given game, where there first player had twice the contribution to the game winning chance than the second, and the second had twice contribution than the third, and so on. The last 32 players had no contributions even if they were present at the game. I allowed for 5% of the games to be randomly won or lost due to attributes not accounted for in the data.

The result was, as expected, that the decision trees did very well on this type of data learning and the k-nearest algorithm did not do too well. I believe, this is due to the fact that decision tree algorithm very easily grouped the attributes that had the most significance to the team’s performance to be at the very top of the tree, and thus was able to build a very nice model, making the attributes that were insignificant not contribute to the classification process. The k-nearest algorithm however, built a multidimensional classification space that was complicated by the number of attributes that were not important and ended up misclassifying a lot of instances.



```
Correctly Classified Instances         980               98      %
Incorrectly Classified Instances        20                2      %
Kappa statistic                          0.9588
Mean absolute error                      0.0381
Root mean squared error                  0.1383
Relative absolute error                  7.9166 %
Root relative squared error             28.2018 %
Total Number of Instances             1000
```

Figure 1 J48 model classification



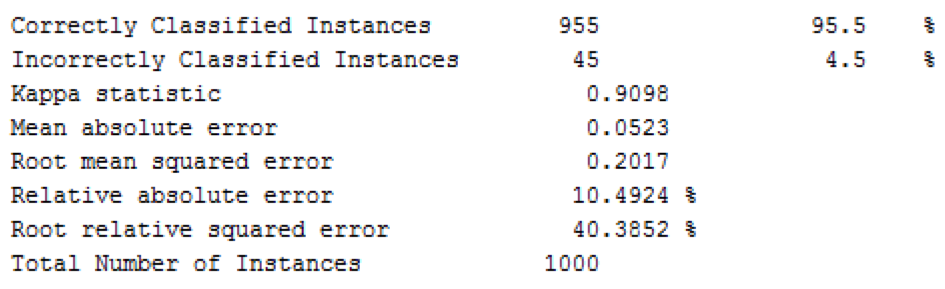
```
Correctly Classified Instances         691               69.1    %
Incorrectly Classified Instances       309               30.9    %
Kappa statistic                          0.3389
Mean absolute error                      0.3202
Root mean squared error                  0.5217
Relative absolute error                 66.5449 %
Root relative squared error            106.3569 %
Total Number of Instances             1000
```

Figure 2 IBk model classification

Problem 2.

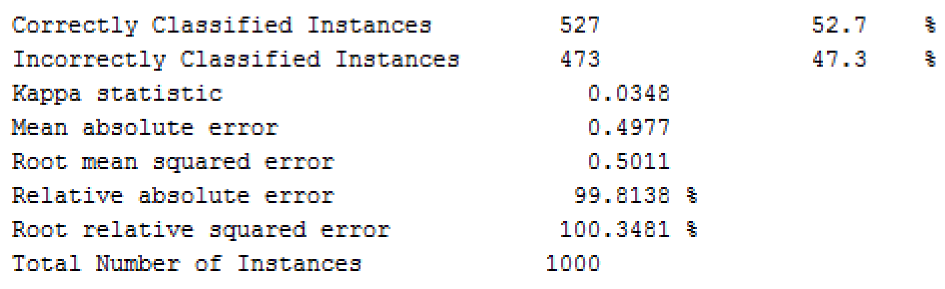
For this section, I created data based on the same situation as in the previous problem – attending players and their contribution to the winning of a game. The way I create the classification this time, however, was using an xor algorithm. Taking 16 players of the team, I xor the attendance of the first 8 with the following 8. I get an array of 8 values, which I divide into 2, and xor the first half with the second once again. From the output, I take the first entry and if it is 1 I let the game be won, and if it is 0 I let the game be lost. (I intended for half the team to not have any significance to the winning of a game) Based on this data Multilayer Perceptron performed over 40% better in classifying the examples correctly over the Naïve Bayes. I introduced a 95% error in the classification of data, meaning that 5% of the games were determined by attributes that were not considered. It should be noted, however, that without any error Multilayer perceptron performed with 100% accuracy.

I suspect that the results we are seeing is due to the fact that Multilayer Perceptions have been modified to handle cases with xor patterns in the data, where Naïve Bayes have not.



```
Correctly Classified Instances         955               95.5    %
Incorrectly Classified Instances        45                4.5    %
Kappa statistic                          0.9098
Mean absolute error                      0.0523
Root mean squared error                  0.2017
Relative absolute error                 10.4924 %
Root relative squared error             40.3852 %
Total Number of Instances             1000
```

Figure 3 Multilayer Perceptron



```
Correctly Classified Instances         527               52.7    %
Incorrectly Classified Instances       473               47.3    %
Kappa statistic                          0.0348
Mean absolute error                      0.4977
Root mean squared error                  0.5011
Relative absolute error                 99.8138 %
Root relative squared error            100.3481 %
Total Number of Instances             1000
```

Figure 4 Naive Bayes

Problem 3.

a) Genetic Algorithm

The case in which it is best:

Genetic Algorithm shines in problems for which we cannot compute descent direction of an optimization routine. Because it uses a random offset (mutation) to “shake” the solution up, a direction of steepest approach to the optimal solution is not being used. It also doesn’t require a function that is being optimized to be known. This could be the case in problems that only have only nominal inputs – in this case a gradient of the input space would be impossible to compute. In particular, a type of problem that would work better with genetic algorithm over other methods given, are the ones where the solution set has multiple local minima. With large enough mutations GAs can get out of local minima to track a larger minimizer in the output space because of the fact that it has a crossover technique.

Example: We can try to solve for daily routines a person can perform to increase their quality of life. If we let each of these events be represented by a nominal yes or no for if they do it or not, depending on whether a particular person has performed that action on a particular day, we can find which group of actions make the person the happiest. Assume there is a solution where a person is very happy playing video games all day and not perform any other activity in their life. Also, imagine that this person doesn’t like to step away from their computer for more than 2 hours unless they are sleeping, but once they are busy doing something else for that amount of time, they get very engaged enough to not want to go back to playing games for a while. GA has the potential of finding this pattern out by randomly mutation successful solutions to an optimal happiness index, and finding that a mix of work and play makes for happiest life.

Why Gradient Descent and Hill Climbing are worse for the job:

Gradient descent requires a gradient, and therefore would not work for a problem with only nominal inputs. For the given example, hill climbing algorithm could get stuck in the solution where the person plays video games all day if that is the starting condition, because as it samples whether or not a person would be happy performing another action besides playing games, they would find that they do not (assuming any other action requires less than 2 hours.) if they sample them one by one. It would never be able to find that only by performing several other actions that take over an hour long, the person will feel happier. Of course one can change their algorithm to account for this, but this means that on average sqrt(N(N-1)) of inputs will need to be sampled, where N is the number of algorithms, which can be a large number for a particularly large list of attributes, and also doesn’t promise for us to get out of local minima. While GA doesn’t guarantee global optimality either, by virtue of random change in multiple attributes at once it can find its solution more effectively.

b) Gradient Descent

Example: We want to find which weather conditions will return in lowest probability of rain tomorrow given numerical attributes such as data for wind speed, percent precipitation, inches of precipitation, temperature and so on. This would work best on Gradient Descent because we can easily set up a gradient and follow the scent direction to find an optimal solution which can be computer with numerical attributes.

Why Genetic Algorithm and Hill Climbing are worse for the job: Genetic algorithm requires binary input data, which means it needs to be nominal. Hill Climbing can somewhat work, if the numerical values are changed by a small value and it iteratively checks if the output changes. The issue with this approach is that it is unclear what these variations should be, since being too small they may not vary output enough, or being too large they can miss the optimal solution.

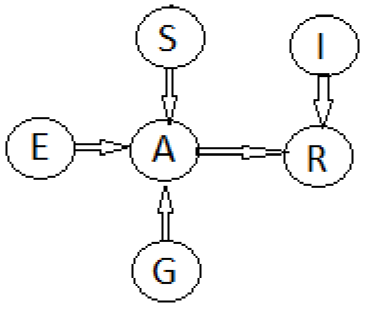
c) Hill Climbing

Example: I think the example I have given with soccer players attending their games and whether or not the team has lost, where the data heavily favors certain player's contribution over the others'. Hill climbing algorithms will easily pick up on this pattern, since the algorithm requires some improvement of the game outcome as it samples inputs in order to move toward the optimal team set-up.

Why Genetic Algorithm and Gradient Descent are worse for the job: Gradient descent requires numerical attributes to be effective. Genetic Algorithm would spend too much time moving in directions away from the optimal that are not helpful (at least at first few iterations) because the cross overs will occur with entire teams, and the algorithm will spend too much time working out the patterns among the players who do not significantly contribute to the team.

Problem 4.

a)



b)

G is a binary input, where we can either sleep well or not sleep well.

S is a binary input, where we can either study for the exam or not study.

A is a binary input, because we can either score it or not score it.

E is a binary input, because the exam can either be easy or hard.

I is a binary input, because we can either find the material interesting or not.

R is a binary input, because we can either recommend the class at the end of the year or not.

4 probabilities are needed to specify S E G and I tables, 8 to specify A since it can be written as a table of 16 probabilities (since A is conditionally dependent on S E and G which there are 8 combination for values of) ,

there are 8 combinations 8 of which can be inferred from the other 8, and finally 4 to specify R. This makes a total of **16 probabilities.**

c)

For N variables that take u number of values, the full joint distribution would requires k^N numbers. In this case we have, 2^6 = **64 independent variables.**

d)

YES They are independent. One can think of this as follows: say we know that the exam was easy. There is no way to find out whether the student has slept or not, because their grade also depends on whether they have studied which we do not know, and neither do we know what they got on the exam. So changing one of G or E will not let us find out what the other is.

e)

NO If we get an A on the exam, we had to have slept well and the exam would have had to have been relatively easy. Therefore G and E are not conditionally independent, because if A is true this means both G and E were true.

5) P(C) = .7*.5*1 + .5 * .7*1 + .3*.5*.7 + .3*.5*.3 = **.85**

In words: The probability of C occurring is the probability of C occurring given that A and B occur times the probability that either will occur, added to the probability of C occurring given that A occurs and B doesn't times the probability of A occurring and B not occurring, added to the probability of C occurring given that A doesn't and B does, times the probability of C given two of those events, and finally added to the probability of neither A nor B occurring times the probability of C occurring given those conditions.