# Problem 1)

1. The attribute that will be picked up for splitting first is movie rating.   
   First of all, we’ll find the original entropy:

Out of 8 results, 4 results are true and 4 are false, so

Here are the calculations if the following attributes are picked up first (in terms of information gain):

So:

1. This is a bad attribute to split on for two reasons.

Firstly, if we split according to this attribute, then the tree is finished because the data suggests that the preference is completely dependent on the rating. For example, if rating is 9, 7, or 5, return true as the preference. However, if we split this way, we completely neglect the other two attributes which are important in the decision process.

Secondly, the data suggests that the only ratings that return true for the preference are 9, 7, and 5. This means the person would prefer a rating of 5 over a rating of 6 and 8. Likewise, he prefers a rating of 7 over a rating of 8, which is unlikely to be the real case.

For example, if the person prefers all long, Natalie Portman movies. If one of such movies happens to have a rating of 8, then the model will wrongly predict his preference.

1. Consider a node with data labeled with + and data labeled with -.

Without loss of generosity, assume(if the opposite case happens, we can reverse the label + and – to achieve the same case). For ease of demonstration, we ignore the denominator, since it’s the same for the training error before and after the split. Our training error would be as we’ll label it +.

Now, let’s split the node into two nodes and.

Say A has data labeled with + and data labeled with –.

Say B has data labeled with + and data labeled with –.

So, and

We have four cases:

1. . In this case, we classify A as + and B as +, . This is consistent as our change in error is .
2. In this case, we classify A as - and B as +, This is consistent as our change in error is
3. In this case, we classify A as + and B as -, This is consistent as our change in error is
4. This case is infeasible since, which implies that, a contradiction of what we established in the setup of the problem.

Therefore, for all feasible splits, the change in training error resulting from a split is always.

# Problem 2)

1. The table of stopping parameter with number of nodes and validation set misclassification is as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Stopping Parameter | 1 | 5 | 9 | 17 | 25 | 37 | 43 | 49 |
| Number of Nodes | 51 | 41 | 35 | 23 | 17 | 17 | 15 | 15 |
| Validation Set Misclassification | 0.054545 | 0.054545 | 0.054545 | 0.018182 | 0.054545 | 0.054545 | 0.018182 | 0.018182 |

The plot of training and testing misclassification error vs. number of nodes is as follows:



The shape of the training data set is as expected. We expect the error to decrease as node number increases (we have more splits). However, the shape of the testing data set should be slightly different theoretically. It should start high, decrease and increase again. This is because with low nodes, you are essentially over-generalizing the pattern so many features are not picked out. As you increase node count, this over-generalization disappears and eventually becomes overfit when you specify many rules that describe the training set but does not necessary accurately describe the test set.

Given the validation table, a reasonable pick for stopping parameter is 17, since it minimizes training error.

1. The table of stopping parameter with number of nodes and validation set misclassification is as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Stopping Parameter | 1 | 5 | 9 | 17 | 25 | 37 | 43 | 49 |
| Number of Nodes | 13 | 13 | 11 | 9 | 9 | 9 | 9 | 9 |
| Validation Set Misclassification | 0.018182 | 0.018182 | 0.018182 | 0.018182 | 0.018182 | 0.018182 | 0.018182 | 0.018182 |

The plot of training and testing misclassification error vs. number of nodes is as follows:



Given the results from the run, it seems to suggest that there is no preference for the stopping parameter. However, like the theoretical result I gave for part 2a, there should be a dip in the graph instead of a straight line. Perhaps we don’t have enough good data to see this pattern.

Note: The current algorithm does not split a node with [+++-] if any possible feature could only split it into [++] and [+-], where the stopping parameter is 1. However, there might be a possibility where if we split into [++] and [+-], the [+-] could eventually be split into [+] and [-] to reduce error. If we split either the misclassification stays or reduces, then we have the risk of overfitting the training data. But in that case, we get the following results:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Stopping Parameter | 1 | 5 | 9 | 17 | 25 | 37 | 43 | 49 |
| Number of Nodes | 71 | 67 | 59 | 45 | 45 | 39 | 37 | 37 |
| Validation Set Misclassification | 0.036364 | 0.036364 | 0.018182 | 0.018182 | 0.018182 | 0.018182 | 0.018182 | 0.018182 |

With a test/training misclassification error chart as:



In this case, a reasonable pick of stopping parameter would be 9, as it minimizes the validation error, and is the value that generates that minimum with least number of nodes.

1. The number of nodes in the post-pruned tree is 17, and its testing misclassification error is 0.046980.
2. The optimal stopping parameter found is 1 with a test error = 0.073826 with high probability. My preference depends on the properties of the training data set. For example if the training data set is small, then I would prefer using cross validation, but if it is large, then I would prefer using a single validation set. This is because, by using cross-validation, we don’t “waste” any data, that is, we maximized the usage of the data. In addition cross-validation nullifies the effects of small training sets because the randomness should somewhat cancel out. The con is that using cross-validation is much more computationally expensive. If we split it into ten groups, then the run-time is ten times as long.
3. The plot of testing misclassification error vs. number of nodes is as follows:



We notice that the training error is quite a lot higher than that of the tree with full training set. Also since the training set is smaller, the number of nodes is also reduced. As a result of reducing the number of nodes, the range of number of nodes is tighter.

This is as expected because we should expect the error to decrease as we have more data describing the pattern. Part a on the contrary tells us that error goes up as nodes increases. There is a fundamental difference between the generation of nodes between the two graphs. Part a generates nodes according to a fixed training data set; increasing nodes in part a means that we start making too many assumptions about the entire population from a constant sample size. However, for part e, the sample size is increasing, so we are actually finding out more about the population.

# Problem 3)

1. Using the attached program, we can compute the sample error for both algorithms:

|  |
| --- |
| import math  def main():  data = []  f = open('hyp\_test\_pred.txt', 'r')  for line in f:  s = eval('('+line+')')  data.append(s)  knnError = 0  dtError = 0  bothError = 0  totalSample = len(data)  for entry in data:  if entry[0] != entry[1]:  knnError += 1  if entry[0] != entry[2]:  bothError += 1  if entry[0] != entry[2]:  dtError += 1  meanKnn = float(knnError)/totalSample  meanDt = float(dtError)/totalSample  print "Average error for knn = %f" % meanKnn  print "Average error for dt = %f" % meanDt  print "knn exclusive number of errors = %d" % (knnError - bothError)  print "dt exclusive number of errors = %d" % (dtError - bothError)  print "Both algorithm number of errors = %d" % (bothError)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Sample Output:

|  |
| --- |
| $ python calculatePart3.py  Average error for knn = 0.190000  Average error for dt = 0.150000  knn exclusive number of errors = 36  dt exclusive number of errors = 20  Both algorithm number of errors = 40 |

Then we know that the sample error of kNN algorithm is , and the sample error of Decision Tree algorithm is . We approximate this problem as normal distribution, and find in the Z table that the probability of 0.025 corresponds to a Z value of 1.96.

The confidence interval for kNN algorithm with 95% confidence can be roughly approximated as:

That calculates to , which ranges from 0.152 to 0.228.

The confidence interval for DT algorithm with 95% confidence can be roughly approximated as:

That calculates to , which ranges from 0.115 to 0.185.

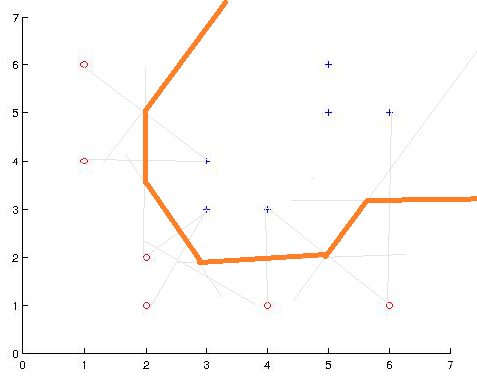
Since from the program we also know that kNN makes 36 exclusive mistakes, denoted as ; and DT makes 20 exclusive mistakes, denoted as . Assume that the algorithm performs equally good, then we know and are binomially distributed with .

We have our null hypothesis where is binomial distributed with .

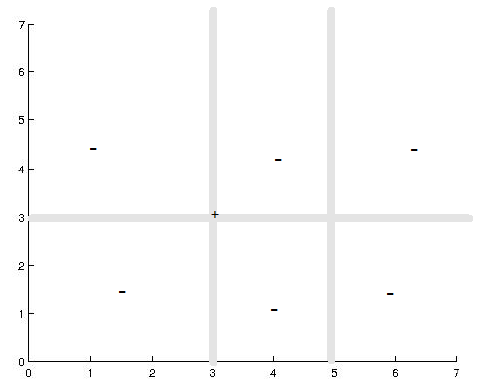
We calculate .

Since we know , we can reject the null hypothesis with 95% confidence. That is, we can say with 95% confidence that the error rate of decision tree is significantly different from that of kNN classifier.

1. The decision boundary of kNN looks like:



1. The decision boundary should correspond to the rule:



1. Looking at the 2D case, we can see that since the true labels follow a very simple and structured rule (if A1 or A2 == 3 or 5, return true, else false) the decision tree predictions much better. Therefore for the 5 dice version, I would prefer the decision tree predictions method much better as well. However, the KNN method could fare better in certain circumstances as well. For example, if we have rules like then the KNN method would definitely perform better. The preference is highly related to the structure of the labeling method, and I prefer decision tree for the 5D case for our problem simply because the problem is structured in a simple if-else manner.