INF283 - Compulsory Assignment 2

By Mathias S. Grønstad – November 15, 2017

1. Calculate entropy gain for each feature

To deal with the missing data I tried to delete all of the rows containing a '?', but this would not be ideal since we scrap a lot of data (going from 8124 to 5644 rows), before even removing duplicates.
Instead I checked which feature occurred most often:
b 3776
? 2480
e 1120
r 192
c 556
It might be a better approach to replace the '?' with the most frequently occurring value, namely 'b'.

An even better approach would probably be to predict what the '?' should be based on all the available data/features and then replace the '?'s with the most probable values. I did this using the decision tree algorithm and got a 100% accuracy on predicting feature 11 on a test set of 25% of the data. In other words, I split the data into the data which had missing values and the data which did not. I did the training and testing on the first data, and then I classified the data which contained '?'. I then replaced all the '?' with the predicted values and recombined the data and wrote it to a .data file called newdata.data. Later I also did a decision tree algorithm to predict e/p using this new dataset, but the information gain was about the same as if I just replaced '?' with 'b'. The accuracy of the algorithm was also almost identical, also when using early stopping.

To keep all the rows, the '?' was replaced with 'b', the most frequently occurring value.

We also must remove the duplicates. There is no point in having duplicate values since they yield no extra information. In fact, they will cause a false goodness of accuracy since we would train and test on some of the same data points. The following line takes care of duplicates:

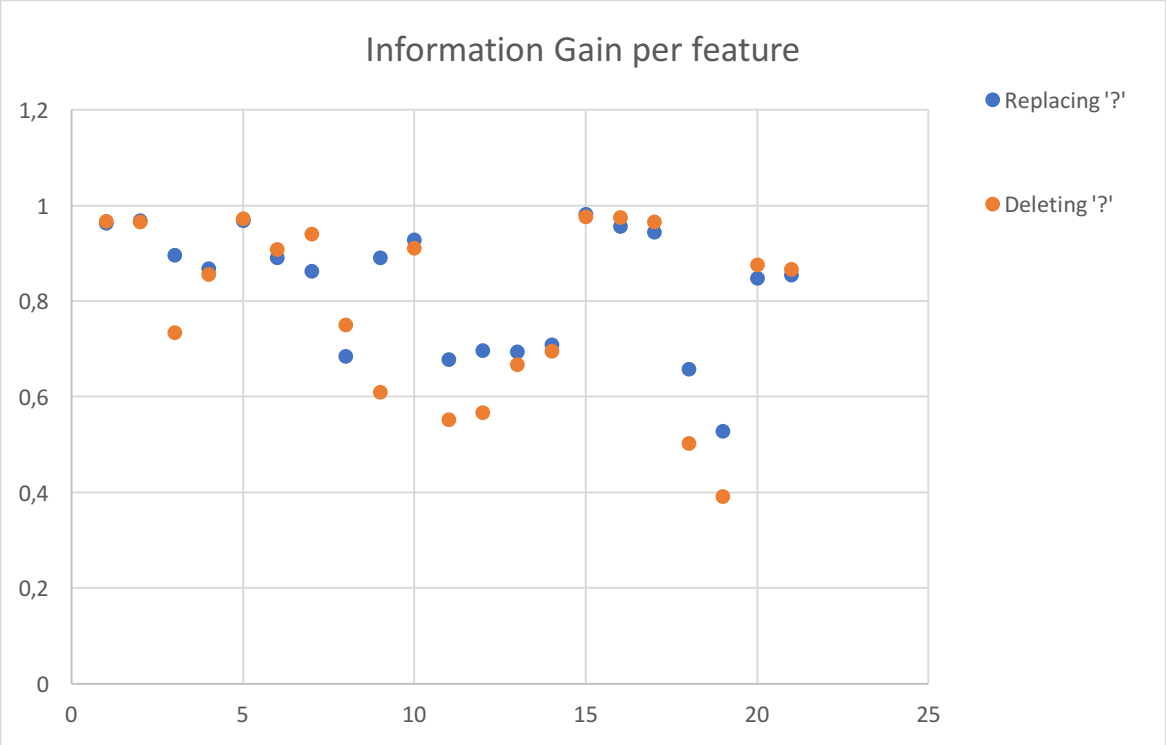
```
data = [data[i] for i in range(len(data)) if i == 0 or data[i] != data[i-1]]
```

After this we are left with 6572 data points (1552 duplicates removed). Next, I shuffle the data using:

```
data = random.sample(data, len(data))
```

This is to later train and test on random parts of the data (which seems to be in some order).

The features are set as a list [0,1, … 20] representing the names of the features. Now we're ready to calculate the information gain for each feature. The features are numbered (1-21) as per Table 1 in the assignment text:



	information gain		
feature	deleting '?' rows	replacing '? With 'b'	Predicting '?'
1	0,9678	0,9635	0,9635
2	0,9665	0,9680	0,9680
3	0,7350	0,8958	0,8958
4	0,8560	0,8683	0,8683
5	0,9727	0,9684	0,9684
6	0,9085	0,8907	0,8907
7	0,9400	0,8623	0,8623
8	0,7504	0,6850	0,6850
9	0,6104	0,8909	0,8909
10	0,9109	0,9286	0,8766
11	0,5524	0,6786	0,6786
12	0,5677	0,6965	0,6965
13	0,6670	0,6938	0,6938
14	0,6958	0,7090	0,7090
15	0,9771	0,9817	0,9817
16	0,9751	0,9568	0,9568
17	0,9657	0,9443	0,9443
18	0,5030	0,6578	0,6578

19	0,3916	0,5278	0,5278
20	0,8767	0,8478	0,8478
21	0,8667	0,8543	0,8543
sum	**16,6572**	**17,4699**	**17,4179**

As we see, the information gain is slightly better, about 5%, (higher) when we replace '?' instead of deleting the entire row. Replacing the '?' leaves us after (removing duplicates) with 6572 data points instead of 5244 which we do when deleting. We therefore prefer replacing to deleting, and the rest of the assignment will use the prior method.

Information gain tells us how much the entropy of the whole training set would decrease by choosing each particular feature for the next set. The ID3 algorithm calculates the information gain for each feature and picks the one with the highest value. In other words, it searches (in a greedy way) through the possible trees and gives the one with the highest information grain per step/stage.

2. ID3 Decision Tree

I make a tree from the data processed in part where I train it on 2/3 of the data. I found out that I can get about 100% accuracy by training on less than 50% of the data, but I don't see the point in that. I was surprised that the accuracy was 100%. The reason it works so well is that the data is quite uniform, meaning many of the data points have much in common.

Since we have 100% accuracy using just 50% of the data to train, I don't see the point in doing cross-validation. A cross validation scheme means we use some of the training set as a test-test set. This means splits the training data into groups called folds. Lets' say we split it into 4 folds, then we could train on fold 1,2,3 and validate on fold 4. Next, we would train on folds 2,3,4 and validate on 1 and so on.

The classifier does not over fit since it gives 100% accuracy on unseen data. Since we have 100% accuracy we don't get any false data, therefore the precision and accuracy are both 1. The precision and accuracy are given as:

Precision = #TP / (#TP + #FP) = 1 / (1 + 0) = 1
Accuracy = #TP / (#TP + #FP) = 1 / (1 + 0) = 1

3. Accuracy vs other measures

Classifying an edible as poisonous would count as a false negative, while classifying a poisonous as edible is a false positive. In this case a false positive is disastrous and we want to reduce the amount of these. This will come with the cost of giving an increased number of false negatives.

In general, we solve this by:

Penalizing the two types of errors - the penalty for making a false negative must have higher weight than that of a false positive. This means that false negatives are more costly than false positives. The algorithm therefore will create trees that favour false positives over false negatives. For example, we could introduce an error function adhering to some rules with certain weights:

Validation classification	**Penalty**
False positive	10
False negative	1
True positive	0
True negative	0

In our tree, we could use a validation set and give a score. If the validation shows a higher number of false negatives, we give it a lower score (or higher error). We do several different training schemes and pick the one which gives the best scores (lowest error). This means we get a tree that minimizes the number of false negatives.

4. Regularization

The simplest way to implement early stopping is to stop the recursion of a node if the ratio of edible/poisonous is below or above a certain threshold for all the leafs under that particular subtree. For example:

If e/p >100/1 we set the node to e
If e/p <1/100 we set the node to p

We just count the number of “e” in the classes list and divide by len(classes). If this p/e number is > threshold we set the node to “e”, if it’s <(1-treshold) we set it to “p”. If both tests fail, we recurse the subtree normally with the ID3 algorithm.

```
edible=0
threshold = 0.9
for c in classes:
   if c=="e":
      edible+=1
ep_ratio = edible/len(classes)
if(ep_ratio)>threshold:
   return "e"
elif(ep_ratio)<(1- threshold):
   return "p"
```

We could also implement early stopping by setting an information gain threshold. If the information gain of the next step is below the threshold we stop the recursion. This is quite similar to the above approach, but maybe less intuitive since information gain is more abstract than the edible/poisonous ratio.

I use the first approach (e/p ratio threshold), and get the following results:

Threshold	Accuracy	Tree depth
0.80	0.9160	3
0.85	0.9228	3
0.86	0.9480	6
0.87	0.9562	7
0.90	0.9904	8
0.95	0.9922	8
0.99	1.0000	8

It seems that accuracy is severely reduced when we set the threshold too low, below 0.9. Or in other words, when the depth is less than 8. The size of the tree does reduce with threshold, however, but it's the depth of the tree which is important for runtime. Therefore, this early stopping method does not seem to be very helpful since just one layer of depth sacrifices a significant portion of the accuracy.
However, this type of early stopping would be helpful if the algorithm builds a huge (deep) tree just to classify a single point. This can cause over fitting. We can try this out by training on 99% of the data, without early stopping, and see if we get such a scenario. Using more data increases the likelihood of having an odd point which creates a deep tree. When I tried this I still get 100% accuracy on the test data classification (66/66 correct). The tree also ends up being 8 layers deep.