Michael Wee

1. Decision Trees and ID3.

(a)

	A=true	A=false
positive	2	2
negative	2	1

	B=true	B=false
positive	1	3
negative	1	2

We want to compare the mutual information between the label and the attributes, A and B.

$I(label; A) = H(label) - H(label|A) = 0.985 - 0.965 = \mathbf{0.02}$

The relevant computations are below:

$$\begin{aligned}
H(label) &= \frac{4}{7}\log_2\frac{7}{4} + \frac{3}{7}\log_2\frac{7}{3} \approx 0.985 \\
H(label|A = true) &= P(pos|A = true)log_2(pos|A = true) + P(neg|A = true)log_2(neg|A = true) \\
&= \frac{2}{4}\log_2\frac{4}{2} + \frac{2}{4}\log_2\frac{4}{2} = 1 \\
H(label|A = false) &= P(pos|A = true)log_2(pos|A = false) + P(neg|A = false)log_2(neg|A = false) \\
&= \frac{2}{3}\log_2\frac{3}{2} + \frac{1}{3}\log_2\frac{3}{1} \approx 0.918 \\
H(label|A) &= P(A = true)H(label|A = true) + P(A = false)H(label|A = false) \\
&= \frac{4}{7}(1) + \frac{3}{7}(0.918) \approx 0.965
\end{aligned}$$

On the other hand,

$I(label; B) = H(label) - H(label|B) = 0.985 - 0.979 = \mathbf{0.006}$

The relevant computations are below:

$$\begin{aligned}
H(label) &= \frac{4}{7}\log_2\frac{7}{4} + \frac{3}{7}\log_2\frac{7}{3} \approx 0.985 \\
H(label|B = true) &= P(pos|B = true)log_2(pos|B = true) + P(neg|B = true)log_2(neg|B = true) \\
&= \frac{1}{2}\log_2\frac{2}{1} + \frac{1}{2}\log_2\frac{2}{1} = 1 \\
H(label|B = false) &= P(pos|B = true)log_2(pos|B = false) + P(neg|B = false)log_2(neg|B = false) \\
&= \frac{3}{5}\log_2\frac{5}{3} + \frac{2}{5}\log_2\frac{5}{2} \approx 0.971 \\
H(label|A) &= P(A = true)H(label|A = true) + P(A = false)H(label|A = false) \\
&= \frac{2}{7}(1) + \frac{5}{7}(0.971) \approx 0.979
\end{aligned}$$

Comparing **0.02** and **0.006**, we will choose to split on A since it gives us more bits. The differentiating factor is the difference between $H(label|A = false)$ and $H(label|B = false)$. The former has a lower specific entropy of 0.918 which signifies more extreme partitions (2:1 vs. 3:2). Intuitively, you would want to split on the attribute that can cause a bigger split in your data. 2:1 is a bigger split than 3:2.

In ID3, the inductive bias reflects this since ID3 strongly prefers a split that generates a partition with one very extreme (2:1) and one very mixed part(2:2) to a partition with 2 fairly well-sorted parts (1:1 and 3:2).

(b) The given data can be represented as follows.

An easy way of reading this is when $A = True$ or 1, it produces a split of 2 positive : 2 negative. When $A = False$ or 0, it produces a split of 2 positive : 1 negative. The same goes for B, C and D. From hereon, we'll just shorten this and say that A yields a $(2:2)$ and $(2:1)$ split.

	True		False	
	1 or (+)	0 or (−)	1 or (+)	0 or (−)
A	2	2	2	1
B	1	1	3	2
C	1	1	3	2
D	3	2	1	1

1b_tree.jpg

The first split is on A because B, C and D all give the same mutual information which is less than that of A. Just as A (2:2 and 2:1) was chosen over B (1:1 and 3:2) in the previous problem, A will also yield a greater mutual information value than C and D because both C

and D split the data the same way B does (1:1 and 3:2).

Going down the $A = 1$ branch, we can then either split on B or C because both attributes yield a (0:1) and (2:1) split when they assume a true or false value, respectively. Let's say we chose to split on B.

Now, when $A = 1$ and $B = 1$, then we're done because the label is just 0 so we create a leaf down the branch. When $A = 1$ an $B = 0$, then we will choose to split on C rather than D because C gives a (0:1 and 2:0) while D gives us a (1:0 and 1:1 split). Again, ID3 prefers more extreme partitions.

Splitting on C, we get that when $C = 1$, the label should be 0 and if $C = 0$, the label should be 1.

Going back up the tree now, when $A = 0$ then we can again split on either B or C since they create (1:0 and 1:1) splits as opposed to D which gives a (2:1 and 0:0) split. Let's say we chose to split on B.

When $A = 0$ and $B = 1$, then we just create a leaf node with label 1.Otherwise, when $A = 0$ and $B = 0$, we get inconsistent data so we just choose the majority label at the previous node. The majority label at the previous node is 1, so we created a leaf with value 1.

(c) After branching on $A = 0$, instead of branching on $B = 0$, just prune that subtree and make it a leaf with label 1 since the two children of that B node give the same label 1 anyway. The reason we can find a simpler tree can be explained by ID3 being a greedy algorithm. ID3 knows it should branch on A first because that gives us more information. After which, it chooses to branch on B or C for the same reason (more mutual information between the attribute and the label). What we realize is that ID3 can't look ahead one step. It knows branching on one attribute gives more information, but it doesn't know if it's going to encounter inconsistency one step further. Thus, because it can't look ahead, the tree becomes susceptible to redundant leaves because of the inconsistent data.

2. ID3 with Pruning.

(a) The average cross validated training performance for the non-noisy dataset is 1.0. The average cross validated test performance for the non-noisy dataset is 0.87.

The average cross validated training performance for the noisy dataset is 0.98. For the test performance, it's 0.78.

In our code, we created the function `tenfoldCrossValidate` that implements the ten-fold cross validation and is called in the `main` function. To calculate the average cross validated training performance, we created the function `performance`.

(b) For pruning, we created the recursive function `prune` which takes in a decision tree `dtree` to prune, a training `dataset` on which the majority class label will be determined, and the `valSet` or the validation set used for the pruning decision.

The high-level description of the function `prune` is it does a depth-first search checking to see if a certain node is not a leaf. If it's another subtree and not a leaf, call the function `prune` again on this subtree. This time, the training data will be split using the `splitData` function so that only the subset of the training data that reaches that branch gets passed to the second iteration of `prune`. Likewise, the `valSet` is also split and passed to the second iteration of `prune`. Each time `prune` finishes iterating through each non-leaf node of a subtree that was passed to `prune`, we used the `MajorityLearner()` to draw out the most popular target value in the subset of the training data. Then, we replace the subtree with a leaf with the most popular target value. Then, we compare the performance of the pruned subtree and the original subtree on the subset of the `valSet` that reaches the subtree being pruned. If the pruned tree gets a lower performance error on that subset, then we prune the subtree. Otherwise, we keep the subtree.

i. Graph of cross-validated training and test performance for full and non-noisy dataset with varying validation set sizes.

   `2bifigure.jpg`

ii. There is a sweet spot for validation set size that peaks performance on the test set. For our algorithm on this particular training set, we found this size to be in the neighbourhood of 19 (the specific value that maximizes on our arbitrary validation size is 19). Following this sweet spot, the test performance declines and then dips and rises. It makes sense that there is a sweet spot as there is an implicit tradeoff between the size of training data and size of validation set data. More data to be trained on leads to better hypotheses but it is important that the validation set has enough data to be a reasonable reflection of the training set.

On the training set, training performance gradually increases with the size of the validation set. With less training data, the hypothesis is likely to learn less complex trees. A peculiarity of this data set is that simpler trees do better provided the right attributes are being split on which may be a cause of this.

iii. Yes, validation set pruning does improve the cross-validated test performance over ID3. ID3 originally has test performance of 0.87 which improves to a peak of 0.92 test performance with validation set pruning.

iv. Yes, overfitting is an issue for these data. ID3 is susceptible to overfitting because there is no regularization and it fits training sets with performance 1.0 on the non noisy dataset. Since pruned trees which go through regularization of complexity to resist overfitting offer better performance on the test set (test performance 0.92 versus 0.87 for ID3), this indicates that the original tree was overfitted.

3. Boosting

(a) We perform a simple modification on the information gain function by changing every instance of counting by 1 to summing up weights that satisfy the specific condition. Where in the old information gain we count the number of data points that satisfy a particular attributes value for a particular label and use denote that with label $N_{x_k=j,c}$, we now sum the weights of the data points that satisfy the value and label. Likewise, instead of N denoting the total number of data points, we now have W, the sum of all of the weights for each data point. We replace $N_{x_k=j}$ with $W_{x_k=j}$ by summing over the weights of the data points that match the attribute value j for attribute k. This is the natural analogue of counting by weight; we still partition the data by attribute and label the same way. We see that it would still work the same way for all weights equal to 1 as adding the weights would be the same as counting.

In the toy example given, the weighted entropy is 1. We get a sum with two terms where the total weight is 1 and the total weight of the data points with label 0 is 0.5 and the total weight of the data points with label 1 is also 0.5. After taking logs, we get a sum of 1.

Computation: $H(labels) = \frac{0.5}{1} log_2 \frac{1}{0.5} + \frac{0.5}{1} log_2 \frac{1}{0.5} = 1.$

New information gain criterion $= H(labels) - H(labels|x_k)$

$$= (\sum_{c=1}^{N} \frac{W_c}{W} log_2 \frac{W}{W_c}) - \sum_{j=1}^{J} \frac{W_{x_k=j}}{W} H(labels|x_k = j)$$

$= (\sum_{c=1}^{N} \frac{W_c}{W} log_2 \frac{W}{W_c}) - \sum_{j=1}^{J} \frac{W_{x_k=j}}{W} (\sum_{c=1}^{C} \frac{W_{x_k=j,c}}{W_{x_k=j}} log_2 \frac{W_{x_k=j}}{W_{x_k=j,c}})$ where $W_c = \sum_{i=1}^{N} w_n \mathbb{I}(y_n = c)$ and $W = \sum_{n=1}^{N} w_n$

i. Table of Cross-Validated Test Performance (Boosting Rounds 10 and 30, Max Depth of 1 and 2)

`boostingtable.png`

The above table shows different test performance for different depths and rounds on the noisy and non noisy dataset. Comparing the columns with different depths shows that a depth of 1 does better overall than a depth of 2. Other than tying its depth 2 counterpart on 30 rounds of boosting on the noisy dataset, depth 1 trees do better than depth 2 trees on noisy and noisy for 10 and 30 rounds.

This makes sense as the training performance on the non noisy dataset starts out at 1.0 on just the first tree trained. It turns out that the decision tree algorithm generally doesnt make trees of depth greater than 2 even without setting a maximum depth parameter. Having training performance of 1.0 is a strong indication of overfitting and thus it makes sense that introducing some regularization (in this case, limiting tree depths to 1) would improve performance on the test set. Introducing this regularization would negate the extra splits that ID2 does on the spurious or peculiar patterns in the training data.

The performance of the resulting classifier after boosting does better overall on non noisy data than on noisy data. This, of course, is a natural result as noise in general throws off learning by providing an inconsistent data set. Also, on the noisy dataset we see that test performance decreases slightly from 0.82 to 0.81 from 10 to 30 rounds of boosting. This displays the vulnerability of boosting to noise, one of its serious practical limitations. If there is inconsistent data, boosting will increase the weight of these data points since they are difficult to explain with its current set of rules. Boosting will try hard to classify these correctly after many rounds which may affect its accuracy on true data.

On the non noisy data we see the trend that increasing the rounds of boosting from 10 to 30 with trees of depth 1 increases test performance. This displays the gains that boosting makes on non noisy data sets. There is no change on the non noisy dataset for trees of depth 2, but recall that this is because these trees are overfitted with training performance of 1.0, so boosting does not actually act.

ii. Graph of Cross-Validated Test Performance with Boosting Rounds from 1 to 30, Depth 1.

```
3aBfigure.jpg
```

On the non-noisy dataset the test performance increases with the number of rounds until it hits 7 rounds or so. After this point, the test performance is just about steady with slightly variations of 0.01 or 0.02. After 22 or so rounds the test performance becomes more stable. Overall, for number of rounds greater than 7, the test performance has a noticeable improvement over just one round of boosting. There are very slight variations, but as the number of rounds gets large we can expect an improvement in test performance. This is expected as boosting decreases variance as different weak learners will be able to pick up on different true patterns in the data but does not change bias as the learned weights will generally outvote any spurious patterns that are learned on. Since the data is not noisy, training data is consistent and the learning algorithm can accurately learn patterns from features and the correctly provided class labels. Therefore as the number of rounds increases test performance should increase or stay in the same ballpark even ask more weak classifiers are added. Since these weak classifiers do better than 0.5 on their own, they will outvote poor rules, which is why test performance should not decrease significantly but increase with a highly weighted hypothesis or stay fairly stable.

On the noisy dataset the test performance has a step downward of about 0.04 in performance between rounds 3 and 5 and another decrease of about 0.05 between rounds 12 and 13. As the number of boosting rounds increase, the test performance dips up and down around the baseline value with just one boosting round, mostly staying below. On average, boosting does not yield improvements in performance with more rounds over just one round on the noisy dataset. Noise adds inconsistency to data which causes many problems in general, but these are amplified by the particular nature of AdaBoosts algorithm. As AdaBoost learns patterns on data, it will continually increase the weights of the classes it gets wrong. However, inconsistent data is of the nature that data points corresponding to a given set of features may be labeled correctly. As AdaBoost learns rules from the correctly classified data, it will continually be stumped by inconsistent data which look like the patterns it learned but has the wrong label. In subsequent rounds it will increase the on these consistent data and eventually devise spurious rules to correctly classify them. The more noise, or the greater the magnitude of incorrectly labeled outliers the more this propagates. AdaBoost does not guarantee better test performance as the number of rounds increases for noisy datasets.

iii. Boosting on the non-noisy dataset will get no worse with more boosting rounds. Meanwhile, validation set pruning has a sweet spot of 19 data points for validation set size that maximizes test performance. This is the best balance of allocating data to train on and data for the validation set. ID3 without pruning will give us the base tree that we sought to improve on. We will first compare the performance of the best cases of all three and then their performance in general. Plots of all three lie above in other sections of this writeup.

With validation set size 19, which is the size that yields maximum test performance, we get test performance of 0.92. Without pruning we get test performance of 0.87. With boosting we get peak performance with 18 boosting rounds with test performance of 0.93. We see that both pruning and boosting gives superior performance to just ID3 without pruning of approximately 0.05 to 0.06 points. This makes total sense ID3 without pruning tends to overfit to training data which is what we are trying to overcoming with pruning while boosting offers superior performance with weak learners.

If we look at general trends of the three methods, we see that for large boosting rounds (around 30), test performance stabilizes at 0.92, which also happens to be the peak performance for validation set pruning. However, looking at the graph of pruning, we see that there is only one set size that reaches this peak value. Pruning performance then can decline from the peak by as much as 0.10 depending on what sizes are being used so pruning performance is sensitive to validation set size. The peak performance tied to validation set size may not generalize well because it is based on an arbitrary chosen size that is sensitive to what is in arbitrarily chosen validation sets which may not generalize to real data. However, boosting on non noisy data with large numbers of rounds should not do worse as the number of rounds increases because spurious patterns will be voted away so we can comfortably set a large number of boosting rounds to achieve its peak performance.

iv. Cross-validated Training and Test Performance, Depth 1, rounds [1,15]

`3aDfigure.jpg`

For both noisy and non noisy data the final classifiers training performance is higher than the test performance by a fairly significant margin (normally more than 0.08 points for large numbers of rounds). For larger boosting rounds, training performance tends to increase gradually on both datasets while the test performance seems to oscillate slightly around a seemingly fixed performance value.

4. Tree Analysis

The tree we chose appeared in 9 out of 10 folds in cross validation with an average test performance of 0.92. It splits on Test 2, which corresponds to uniformity of cell shape. On this data set it turns out that uniformity of cell shape in itself is a good predictor of having breast cancer or not. If the uniformity of cell shape takes on a value less than or equal to 3 the person is likely to be benign; if the uniformity of cell shape takes on a value greater than or equal to 4 the person is likely to have breast cancer.

This tree was generated with validation set pruning on the original tree generated by ID3. We used a validation set size of 19 which yields the peak performance on this dataset. Looking through the dataset manually we see that splitting on attribute 2 almost always correctly predicts breast cancer. This is confirmed by the 0.92 average performance on the test set for which this particular tree contributes the performance on 9 out of 10 folds. We use a reasonably sized validation set of 19 that is likely to have a reasonable estimate of patterns in the data. The pruning algorithm decides to prune away the children of this initial split, which were decided by maximum information gain, because it leads to overfitting on the training dataset. This suggests that once we factor in information about Attribute 2, splits on any other attribute tend to overfit to the peculiarities of the dataset. It may be that there are other splits below Attribute 2 that could further refine the diagnosis but are not found because the dataset is relatively small with just 100 data points compounded with the fact that we split it further for cross validation and validation set pruning. A medical professional can verify if this Attribute captures the best the true pattern or if additional data would be useful in further splitting below Attribute 2 to get the small remainder of cases correct.