This assignment makes you more familiar with decision trees and random forests. You will apply the concepts from the lecture, implement and experiment with them to see decision and regression trees *in action*. The assignment consists of two IPython notebooks and some questions to be answered with pen and paper. Those questions at the end are supposed to prepare you for the final exam as well as encourage you to think about some properties of trees and forests.

1. **Simple Regression Trees**
   In this exercise, you implement a regression tree and a random forest for a one dimensional problem. With that implementation, you explore the impact of some of the important hyperparameters.

   (a) Work through the 'regression_trees.ipynb' notebook found on ILIAS.

   > **Solution:** See the 'regression_trees_solution.ipynb' for a reference implementation.

   (b) What is the computational complexity of the decision tree you implemented? To ease the notation, assume that N, the number of datat points, is a power of 2, every split halves the observations exactly, and that $min\_leaf = 1$. How does the complexity change, if the best loss for N data points could be computed in $\mathcal{O}(N)$?

   > **Solution:** Computing the loss for any split of N data points is of order $\mathcal{O}(N)$, which can be achieved for any loss function. As we have to look at at most $\mathcal{O}(N)$ possible split points, the complexity of finding the best possible split points is $\mathcal{O}(N^2)$. This complexity also holds for computing the optimal splits for two sets of size $\frac{N}{2}$, or four of size $\frac{N}{4}$, ... In particular, this means all computations at each each level of the tree are $\mathcal{O}(N^2)$. If splitting halves the data every time there are exactly $\log_2(N)$ levels in the tree. This leads to a complexity of $\mathcal{O}(N^2 \log_2(N))$.
   >
   > For the squared loss function (and maybe a few others) one can do substantially better than that. By computing the loss of all possible splits sequentially, one can obtain the loss of all possible split points in $\mathcal{O}(N)$. The idea is to start out with a split where all but one datapoint go into the right child. Computing the loss for that takes $\mathcal{O}(N)$. By moving one datapoint at a time over to the left child, one can compute the associated loss in $\mathcal{O}(1)$ operations. In that case, the total complexity of growing a one dimensional regression tree would be the familiar $\mathcal{O}(N \log_2(N))$.

2. **Decision Trees and Ensembles**
   Here, you explore ensembling via bagging for different types of trees and study the out-of-bag error, a useful estimate of the validation error that comes at low cost when bagging is used.

   (a) Work through the 'tree_ensembles.ipynb' notebook found on ILIAS.

   > **Solution:** See the 'tree_ensembles_solution.ipynb' for a reference implementation.

3. **The CART Algorithm**
   This question aims to give you a better understanding of the algorithm used to *grow* decision trees.

   (a) Explain *in words* the CART algorithm for growing a classification tree as presented in the lecture. Give the definition of Entropy, expressed in terms of $p(v_k)$ for $k = 1, \ldots, K$ (see also the slides).

Solution: The CART algorithm splits the data hierarchically following the same basic steps for every split. First, it computes the best possible split for every feature based on a loss function, often the information gain:

$$I = N \cdot H(V) - N_l \cdot H(V_l) + N_r \cdot H(V_r)\,.$$

Here,

$$H(V) = -\sum_{k=1}^{K} p(v_k) \log_2 p(v_k)$$

is called the entropy of the set of class labels v_k from the current data. It then selects and stores the feature and associated split value with the highest information gain. The data is split accordingly and the algorithm is applied recursively on both partitions to form the two subtrees. The recursion stops, if the node is pure (all labels are identical), if all data points have identical features, when the split would lead to a leaf with less than a given parameter entries, when the total number of leaves in the tree reaches a given limit, or if the tree reaches a defined maximum depth.

(b) How would you handle the following three distinct cases that can happen when deciding on a splitpoint in a node:

- All instances have the same class label, but various attribute values
- All instances have the same value for all attributes, but various class labels
- The instances have various class labels and various attribute values

Solution: As already discussed above, the first case means that no further splitting would lead to any information gain, as the entropy is already zero. Therefore, the data is split no further and the node is considered a leaf.

The second case can happen if there is no *ground truth* and the class label is drawn from a distribution, or there is *noise*. As all attributes are identical, there is no further split possible, and the node has to be considered a leaf. If the tree should only predict a label, majority voting can be applied and only the corresponding label is stored. If the prediction of the tree are the probabilities of the classes, the frequencies of all data points in the leaf are the empirical estimate for that.

In the last case, the standard CART procedure is executed: consider all possible split points for all attributes and pick the one with the highest information gain. Unless a different stopping criterion is met (e.g., maximum depth) , split the data accordingly and recurse into the two child nodes.

(c) The table below shows information from previous computer sales. We want to to build a decision tree that for any new computer determines whether we should buy it. Execute the decision tree algorithm by hand. Use the definition of Entropy and Information Gain.

nr	memory	processor	rest	buy
1	much	fast	bad	yes
2	much	slow	bad	no
3	few	fast	good	yes
4	few	fast	bad	no

Solution: We have to consider all possible splits on all three attributes to determine the best split. All attributes are binary, so there is only one possible split per feature. The entropy of the total data is

$$H(V) = -\sum_{v\in\{\text{yes,no}\}} p(v)\cdot\log_2 p(v) = -\underbrace{p(\text{no})}_{=\frac{1}{2}}\cdot\underbrace{\log_2 p(\text{no})}_{=-1} - \underbrace{p(\text{yes})}_{=\frac{1}{2}}\cdot\underbrace{\log_2 p(\text{yes})}_{=-1} = 1\,.$$

This value is in fact the largest possible value for the entropy here. This makes sense as both labels are equally likely.

first split on memory If we split on memory first, the data will be split the following way:

memory = much		memory = few	
nr	buy	nr	buy
1	yes	3	yes
2	no	4	no

From that, we read the following values we need to compute the information gain:

$$N_l = 2 \qquad p_l(\text{no}) = \frac{1}{2} \qquad p_l(\text{yes}) = \frac{1}{2}$$
$$N_r = 2 \qquad p_r(\text{no}) = \frac{1}{2} \qquad p_r(\text{yes}) = \frac{1}{2}$$

The corresponding entropies follow as:

$$H_l = -\frac{1}{2}\cdot(-1) - \frac{1}{2}\cdot(-1) = 1$$
$$H_r = -\frac{1}{2}\cdot(-1) - \frac{1}{2}\cdot(-1) = 1$$

Finally, the information gain for this split is

$$I = 4\cdot 1 - 2\cdot 1 - 2\cdot 1 = 0\,.$$

This is no surprise given that both child nodes have exactly the same probabilities for the class labels as the data before splitting.

first split on processor If we split on processor first, the data will be split the following way:

processor = fast		processor = slow	
nr	buy	nr	buy
1	yes	2	no
3	yes		
4	no		

From that, we read the following values we need to compute the information gain:

$$N_l = 3 \qquad p_l(\text{no}) = \frac{1}{3} \qquad p_l(\text{yes}) = \frac{2}{3}$$
$$N_r = 1 \qquad p_r(\text{no}) = 1 \qquad p_r(\text{yes}) = 0$$

The corresponding entropies follow as:

$$H_l = -\frac{1}{3}\cdot(-1.585) - \frac{2}{3}\cdot(-0.585) = 0.918$$
$$H_r = -1\cdot 0 - \underbrace{0\cdot\log_2(0)}_{\stackrel{!}{=}0} = 0$$

Finally, the information gain for this split is

$$I = 4 \cdot 1 - 3 \cdot 0.918 - 1 \cdot 0 = 1.245\,.$$

Note that most of the information gain comes from the splitting away a single data point here.

first split on rest If we split on rest first, the data will be split the following way:

rest = bad		rest = good	
nr	buy	nr	buy
1	yes	3	yes
2	no		
4	no		

From that, we read the following values we need to compute the information gain:

$$N_l = 3 \qquad p_l(\text{no}) = \frac{2}{3} \qquad p_l(\text{yes}) = \frac{1}{3}$$
$$N_r = 1 \qquad p_r(\text{no}) = 0 \qquad p_r(\text{yes}) = 1$$

The corresponding entropies follow as:

$$H_l = -\frac{2}{3} \cdot (-0.585) - \frac{1}{3} \cdot (-1.585) = 0.918$$
$$H_r = -1 \cdot 0 - \underbrace{0 \cdot \log_2(0)}_{\stackrel{!}{=}0} = 0$$

Finally, the information gain for this split is

$$I = 4 \cdot 1 - 3 \cdot 0.918 - 1 \cdot 0 = 1.245\,.$$

This split achieves the same information gain as the previous one.

As the splits on *processor* and *rest* achieve the same information gain, we can just pick one as our split criterion does not favor one over the other. Here, we will split on *processor*. As the right child node only contains one data point, we only need to consider splitting the left subtree. The entropy for the three remaining data points is 0.918

second split on memory If we split on memory first, the data will be split the following way:

processor = fast			
memory = much		memory = few	
nr	buy	nr	buy
1	yes	3	yes
		4	no

From that, we read the following values we need to compute the information gain:

$$N_l = 1 \qquad p_l(\text{no}) = 0 \qquad p_l(\text{yes}) = 1$$
$$N_r = 2 \qquad p_r(\text{no}) = \frac{1}{2} \qquad p_r(\text{yes}) = \frac{1}{2}$$

The corresponding entropies follow as:

$$H_l = -\underbrace{0 \cdot log_2(0)}_{\stackrel{!}{=}0} - 1 \cdot 0 = 0$$

$$H_r = -\frac{1}{2} \cdot (-1) - \frac{1}{2} \cdot (-1) = 1$$

Finally, the information gain for this split is

$$I = 3 \cdot 0.981 - 1 \cdot 0 - 2 \cdot 1 = 0.755\,.$$

second split on rest If we split on rest first, the data will be split the following way:

processor = fast			
rest = bad		rest = good	
nr	buy	nr	buy
1	yes	3	yes
4	no		

From that, we read the following values we need to compute the information gain:

$$N_l = 2 \qquad p_l(\text{no}) = \frac{1}{2} \qquad p_l(\text{yes}) = \frac{1}{2}$$

$$N_r = 1 \qquad p_r(\text{no}) = 0 \qquad p_r(\text{yes}) = 1$$

The corresponding entropies follow as:

$$H_l = -\frac{1}{2} \cdot (-1) - \frac{1}{2} \cdot (-1) = 1$$

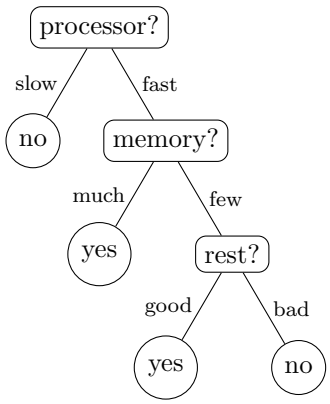
$$H_r = -1 \cdot 0 - \underbrace{0 \cdot \log_2(0)}_{\stackrel{!}{=}0} = 0$$

Finally, the information gain for this split is

$$I = 3 \cdot 0.918 - 2 \cdot 1 - 1 \cdot 0 = 0.755\,.$$

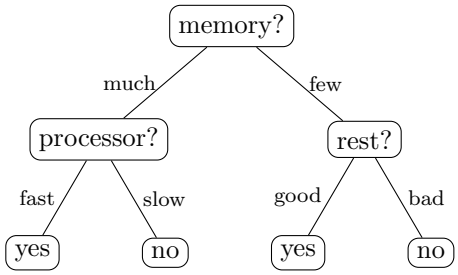
Again, this split achieves the same information gain as the previous one.

As before, we found two splits with the highest information gain, so we are free to pick any of the two features. We pick (for no real reason) *memory* as the attribute for our split.
We can see directly that the last node with more than one value contains the third and fourth row. The only feature they differ by is rest, and splitting will separate them into two pure child nodes/leaves.
Here is a visualization of the decision tree we constructed.



(d) Optimizing Entropy or Information Gain for every split does not always lead to the *best* tree. Can you find a perfectly balanced tree that has the same accuracy as the one in part (c)?

Solution: Looking at the tree from above, we could ask ourselves, is there a perfectly balanced tree that could split the data. The above tree represents the worst case scenario in which only a single data point is split off at node. To arrive at a balanced tree, we have to split using *memory*. From part (c), we know this is the worst possible split, but it is the only attribute that allows a 2-2 split. Splitting the data that way, we can easily see that the computers with memory = much can be separated by the *processor* attribute, and the ones with memory = few can be distinguished by *rest*. Here is an illustration of the tree:



Finding this tree with the CART algorithm is impossible, as the first split is clearly the worst possible in terms of information gain. When training a randomized tree, where the features used to split are subsampled to just one feature per split, this particular tree becomes feasible, but not likely.

The reasons optimizing criteria like the information gain greedily are preferred are simple: they are very fast (at training time) and the worst case does not often occur when using real world data. Considering all features for a single split is linear in the number of features, while considering all possible combinations of features for two consecutive splits is already quadratic, and the complexity depends exponentially on the depth considered. This makes finding the *perfect* tree impossible in practice. Luckily, trees found with the CART algorithms usually perform very well due to the low bias of trees in general. In our example, the balanced tree is only faster for predictions (every data point can be classified with only two decisions, while the tree from part (c) requires 2.25 decisions when averaged over the training data. We constructed the data to exhibit the worst case behavior, but larger datasets rarely have this adversarial structure.

4. **Majority Voting and Class Probabilities**

Show that for a binary classification problem with n_0 (n_1) observations of class 0 (1) in a given leaf:

(a) majority voting minimizes the absolute error over the set of examples in that leaf

Solution: For all the samples in one leaf, the error of any constant prediction is

$$Error = \frac{\#\text{wrongly labeled points}}{\#\text{points in leaf}}$$

As the size of the leaf is fixed, we minimize this by correctly predicting as many of the samples as we can, i.e. majority voting.

(b) the class probabilities $p_0 = n_0/(n_0 + n_1)$ and $p_1 = n_1/(n_0 + n_1)$ minimize the sum of squared error.

Solution:
Before we can start, we need to say a few words about the squared error in binary classification. While this loss is usually used for regression, it can be applied to classification as well. In this context it is often referred to as the Brier score (when normalized by the number of samples). It quantifies the difference between predicticted and actual probabilities. When considering different samples (in a leaf of tree, for example), we can write its definition as

$$SSE = \sum_{i=1}^{n_0+n_1} \sum_{j=0,1} (p_j - t_i)^2.$$

Here, we sum over all samples in the leaf and compute the difference between the target class (t_i) and the predicted class probability (p_0 and p_1) from the model:

$$SSE = \sum_{i=1}^{n_0+n_1} \left[(t_i - p_0)^2 + (t_i - p_1)^2\right]$$

Let us split this sum into two corresponding to the two class labels:

$$SSE = \sum_{i=1}^{n_0+n_1} \left\{\left[(1 - p_0)^2 + (0 - p_1)^2\right] \delta_{t_i,0} + \left[(0 - p_0)^2 + (1 - p_1)^2\right] \delta_{t_i,1}\right\}.$$

Taking into account that the probabilities are constant and the t_i's can only be 0 and 1, we find:

$$SSE = n_0 \left[(1 - p_0)^2 + (0 - p_1)^2\right] + n_1 \left[(0 - p_0)^2 + (1 - p_1)^2\right].$$

Substituting $p_1 = 1 - p_0$ yields

$$SSE = n_0 \left[(1 - p_0)^2 + (1 - p_0)^2\right] + n_1 \left[p_0^2 + p_0^2\right].$$

Expanding this, we find

$$SSE = 2(n_0 + n_1)p_0^2 - 4n_0p_0 + 2n_0.$$

We find the minimum by computing $\mathrm{d}(SSE)/\mathrm{d}p_0$ and set it equal to zero:

$$0 = \frac{\mathrm{d}(SSE)}{\mathrm{d}p_0} = 4(n_0 + n1)p_0 - 4n_0 \Longrightarrow p_0 = \frac{n_0}{n_0 + n_1}.$$

The expression for p_1 follows directly from the substitution $p_1 = 1 - p_0$ we used above.

5. **Number of Out-of-Bag Samples**
In the lecture, you were told that almost 37% of the data points do not occur in a given bootstrap sample. This question will show you why. Consider a bootstrap sample of size N where N is also the number of data points, i.e. we draw a sample the same size as the original data, but we pick data points randomly with replacement.

(a) What is the probability p_i that any given data point is NOT in a bootstrap sample?

Solution: The probability of the i^{th} data point to NOT be selected as the n^{th} sample in the bootstrap procedure is

$$p_{i,n} = 1 - \frac{1}{N}.$$

As we choose the samples i.i.d., the probability of NOT including the sample at all reads:

$$p_i = \prod_{n=1}^{N} \left(1 - \frac{1}{N}\right) = \left(1 - \frac{1}{N}\right)^N .$$

(b) As this probability is the same for all data points, and the random sampling is independent, the expected number of unique data points in the bootstrap sample will be $\mathbb{E}\left[N_{unique}\right] = N(1 - p_i)$. What is the limit $N \to \infty$ of the fraction N_{unique}/N?

Solution: As stated in the question, the expected value of unique samples is the sum of all $1 - p_i$ terms, i.e.

$$\mathbb{E}[N_{unique}] = \sum_{i=1}^{N} p_i = N - N\left(1 - \frac{1}{N}\right)^N$$

The fraction of data points used in each sample is simply

$$\mathbb{E}\left[\frac{N_{unique}}{N}\right] = 1 - \left(1 - \frac{1}{N}\right)^N .$$

This value converges to $1 - \mathrm{e}^{-1} \approx 0.632$ as N grows. This can be seen by recalling the definition for the exponential function:

$$\mathrm{e}^x = \lim_{N \to \infty} \left(1 + \frac{x}{N}\right)^N$$