– Part 1  
**Question 1.1**

Besides the number of instances, what is another main difference between train\_full.txt and train\_sub.txt?

The main difference is that train\_sub.txt is rather unbalanced dataset in comparison to train\_full.txt. From figure 1, it can be observed that all the labels are well represented within the range from 599 to 671 observations. This makes the dataset relatively balanced. On the other hand, when examining figure 2, we can see that the representation of each label is not uniform, ranging from 21 up to 187 observations. This makes this dataset relatively unbalanced.

Figure 1: Train\_full.txt:

Text

Description automatically generated

Figure 2: Train\_sub.txt

Text

Description automatically generated

**Question 1.2**

What kind of attributes are provided in the dataset (Binary? Categorical/Discrete? Integers? Real numbers?) What are the ranges for each attribute in train\_full.txt?

All the attributes are integers. The dependant variable (y vector) contains categorical/discrete strings. The ranges for each attributed, labelled from left to right using integers, e.g. 0, 1, 2 etc., can be found in table 1.

Table 1: ranges of each attribute for train\_full.txt

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Attribute index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |  |  |
| Min value | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 3 | 0 | 4 | 0 | 1 | 0 | 1 |
| Max value | 10 | 15 | 11 | 12 | 14 | 14 | 14 | 10 | 12 | 13 | 11 | 15 | 12 | 15 | 11 | 14 |

**Question 1.3**

Train\_noisy.txt is actually a corrupted version of train full.txt, where we have replaced the ground truth labels with the output of a simple automatic classifier. What proportion of labels in train\_noisy.txt is different than from those in train full.txt? (Note that the observations in both datasets are the same, although the ordering is different). Has the class distribution been affected? Specify which classes have a substantially larger or smaller number of examples in train\_noisy.txt compared to train\_full.txt.

Figure 3:

**Text, table

Description automatically generated with medium confidence**

Figure 4:

Chart, bar chart

Description automatically generated

# – Part 2

**Task 2.1**

We divided this problem into smaller tasks which we have solved by implementing the corresponding helper functions: *calculate\_entrophy, make\_opposite\_filter, calculate\_best\_info\_gain* and *split\_by\_best\_rule*. Those functions were then used in a recursive function: *induce\_tree*. The following paragraphs shed some light on the design decisions and why they were made.

**calculate\_entrophy**

A helper procedure those purpose is to calculate the entropy for a given slice of the dataset. It has time complexity, where N is the number of observations in the given slice of the dataset. It is used in *calculate\_best\_info\_gain* function.

**make\_opposite\_filter**

A helper producer that returns the inverted NumPy array of False and True values. It has time complexity, where N is the number of observations in the given array. It is also used in *calculate\_best\_info\_gain* function.

**calculate\_best\_info\_gain**

This is a procedure which iteratively keeps splitting the dataset by both features (column indices) and values (integers in each column). The splitting is binary and for each split, it calculates the information gained. Return is the feature index and the integer value, whose combination gave us the highest information gained.

The design decision was to use two loops. The First outer loop iterates over all features, the inner loop iterates over all unique values of a given feature column. The whole procedure has an upper bound of where N is the number of observations. In a case where the number of features is larger or equal to the number of observations, the upper bound would be , however, for our datasets where the highest number of features was 16, linear time complexity can be assumed.

We were careful to use NumPy methods to manipulate the dataset to be as efficient as possible. For instance, numpy.unique method was used to not only obtain the unique integer values of a given column, but also sort them in a convenient way which was used for the inner loop. This design decision does not waste a single iteration; the inner loop does not iterate over anything it does not need to, skipping integers that are not present and not iterating over a district integer more than once. Furthermore, the numpy.unique method is more efficient than simply sorting the array or looking for minimum and maximum values and using the range object.

This function is used in recursive function, *induce\_tree*.

**split\_by\_best\_rule**

This procedure takes as input the dataset, and the feature index and the integer value whose combination maximises the information gained. It returns two datasets which were split by the according to feature index and the integer value*.*

It has time complexity, where N is the number of observations in the input dataset. NumPy arrays and operations were used for the splitting to have it efficient.

**induce\_tree**

So far, all we have done was for one binary split. This procedure recursively repeats the operations above. The time complexity of the recursion itself is since we have binary splits, which reduces the database by half. Each recursive call calls procedures, discussed above, whose total time complexity is , unless the number of features exceeds the number of observations (does not occur in the datasets provided). The overall time complexity of *induce\_tree* is then .

**Task 2.2**

# – Part 3

**Question 3.1**

**Question 3.2**

**Question 3.3**

– Part 4  
**Task 4.1**

**Question 4.1**