# CSI4142 Fundamentals of Data Science Project Phase 4: Data Mining

Group 1

**Professor Yazan Otoum,**

TA: Lansu Dai

Jenson Wu #300166874
Victor Feng #300176400
Lootii Kiri #300189957

Date: April 9th 2024

Part A:

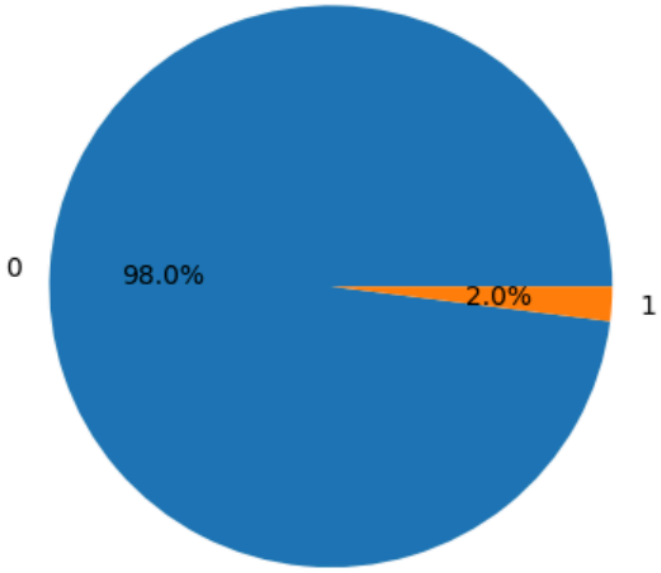



Figure 1 Class imbalance between canceled and non-cancelled flight

When looking at the dashboard that we created in phase 3 of the project, we realized that we had a huge class imbalance between canceled and non-cancelled flights, where only 2 % of all flights were canceled yearly(which meant that we had 500 000 data points for canceled flights). We decided to use undersampling for this phase because we felt that populating the minority will not produce realistic results and we also felt that 500 k for each class was sufficient to get an accurate classifier.

For the sampling technique, we decided to use stratified sampling to get the training(contain 66% of the data points) and the testing(contain 33% if the data) datasets, so that we got an even distribution of all attributes and class labels

Part B. Classification (Supervised Learning):

**Decision Tree:**

For the decision tree algorithm, we decided to do this in two ways to show our understanding and application of the algorithm essential for constructing decision trees. Instead of only using a decision tree library made accessible in python as seen in scikit-learn, we also implemented our own code that uses the logic of finding the Entropy of both categorical and numerical attributes of a given set. Our code also takes the entropy and calculates an information gain of all of the given attributes in relation to our class attribute.

```java
package com.example;

import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFCell;
//import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.apache.poi.xssf.usermodel.XSSFRow;

//import java.io.FileInputStream;
//import java.io.FileNotFoundException;
//import java.io.IOException;

public class DecisionTreeUtils {

    public static int countOccurrencesOfMonth(XSSFSheet sheet, Double month) {

        int numOfRows = sheet.getPhysicalNumberOfRows() - 1;
        int count = 0;

        for (int i = 1; i <= numOfRows; i++) {
            XSSFRow row = sheet.getRow(i);
            if (row.getCell(0).getNumericCellValue() == month) {
                count++;
            }
        }
        return(count);
    }


    public static int countOccurrencesOfMonthCancellationType(XSSFSheet sheet, Double month, Double binary) {

        int numOfRows = sheet.getPhysicalNumberOfRows() - 1;
        int count = 0;

        for (int i = 1; i <= numOfRows; i++) {
            XSSFRow row = sheet.getRow(i);
            XSSFCell lastCell = row.getCell(8);
            if (row.getCell(0).getNumericCellValue() == month && lastCell.getNumericCellValue() == binary) {
                count++;
            }
        }
        return(count);
    }

```

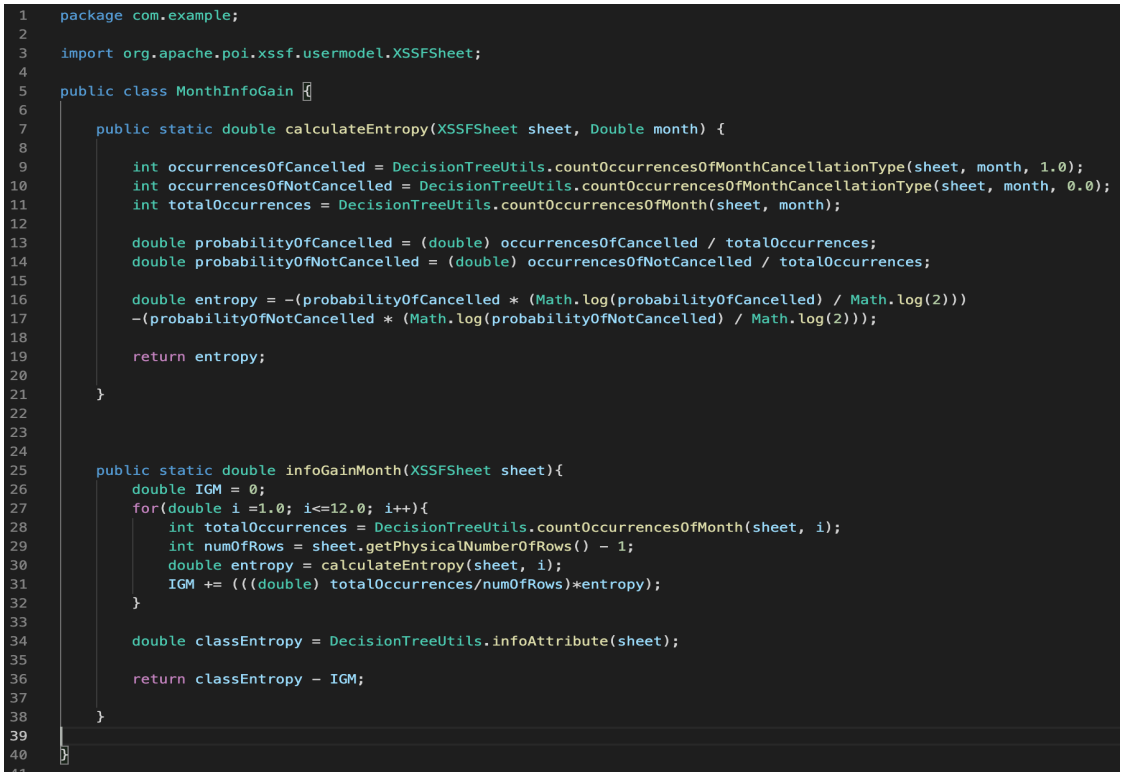
Image 1: functions to count the occurrence of a given month. As well as one that counts the occurrence of a given month that is either canceled(binary = 1.0) or not canceled(binary = 0.0).

```java
    public static double entropy(double positive, double negative){
        double total = positive + negative;
        double entropy = -((positive/total) * (Math.log(positive/total) / Math.log(2)))
        -((negative/total) * (Math.log(negative/total) / Math.log(2)));
        return entropy;
    }

    public static double infoAttribute(XSSFSheet sheet){
        int numOfRows = sheet.getPhysicalNumberOfRows() - 1;
        double zeros = 0.0;
        double ones = 0.0;
        for(int i=1; i<= numOfRows; i++){
            XSSFRow row = sheet.getRow(i);
            XSSFCell lastCell = row.getCell(8);
            if(lastCell.getNumericCellValue() == 0.0){
                zeros++;
            }
            if(lastCell.getNumericCellValue() == 1.0){
                ones++;
            }
        }
        double infoAttribute = entropy(ones,zeros);

        return infoAttribute;
    }
```

Image 2: function that calculates the entropy and method that calculates Info(Class). As in the information or entropy of the entire class which will be later used to subtract from attribute entropies to receive information gain.



```java
package com.example;

import org.apache.poi.xssf.usermodel.XSSFSheet;

public class MonthInfoGain {

    public static double calculateEntropy(XSSFSheet sheet, Double month) {

        int occurrencesOfCancelled = DecisionTreeUtils.countOccurrencesOfMonthCancellationType(sheet, month, 1.0);
        int occurrencesOfNotCancelled = DecisionTreeUtils.countOccurrencesOfMonthCancellationType(sheet, month, 0.0);
        int totalOccurrences = DecisionTreeUtils.countOccurrencesOfMonth(sheet, month);

        double probabilityOfCancelled = (double) occurrencesOfCancelled / totalOccurrences;
        double probabilityOfNotCancelled = (double) occurrencesOfNotCancelled / totalOccurrences;

        double entropy = -(probabilityOfCancelled * (Math.log(probabilityOfCancelled) / Math.log(2)))
        -(probabilityOfNotCancelled * (Math.log(probabilityOfNotCancelled) / Math.log(2)));

        return entropy;

    }



    public static double infoGainMonth(XSSFSheet sheet){
        double IGM = 0;
        for(double i =1.0; i<=12.0; i++){
            int totalOccurrences = DecisionTreeUtils.countOccurrencesOfMonth(sheet, i);
            int numOfRows = sheet.getPhysicalNumberOfRows() - 1;
            double entropy = calculateEntropy(sheet, i);
            IGM += (((double) totalOccurrences/numOfRows)*entropy);
        }

        double classEntropy = DecisionTreeUtils.infoAttribute(sheet);

        return classEntropy - IGM;

    }

}
```

image 3: since our “Month” attribute is categorical, it is handled differently then the numerical attributes. Here are functions that calculate the entropy and information gain for “Month”.

```java
    public static double[] countOfZeroAndOne(double[][] sortedArray, int number){
        int numOfRows = sortedArray.length -1;
        double zeroCountLessThan = 0;
        double oneCountLessThan = 0;
        double zeroCountGreaterThan = 0;
        double oneCountGreaterThan = 0;

        for(int j=0; j<=numOfRows; j++){
            if(j<=number  && sortedArray[j][1] == 0.0){
                zeroCountLessThan++;
            }
            if(j<=number  && sortedArray[j][1] == 1.0){
                oneCountLessThan++;
            }
            if(j>number  && sortedArray[j][1] == 0.0){
                zeroCountGreaterThan++;
            }
            if(j>number  && sortedArray[j][1] == 1.0){
                oneCountGreaterThan++;
            }
        }
        double[] values = {oneCountLessThan, zeroCountLessThan-1, oneCountGreaterThan, zeroCountGreaterThan};

        return values;
    }

    public static double[][] sortedAttributesClass(XSSFSheet sheet, int coloumn){
        int numOfRows = sheet.getPhysicalNumberOfRows() - 1;
        double[][] sorted = new double[numOfRows+1][2];
        for(int i=1; i<=numOfRows; i++){
            XSSFRow row = sheet.getRow(i);
            XSSFCell cellOfAttribute = row.getCell(coloumn);
            XSSFCell cellOfClass = row.getCell(8);
            sorted[i][0] = cellOfAttribute.getNumericCellValue();
            sorted[i][1] = cellOfClass.getNumericCellValue();
        }
        // Sort the 2D array based on the first element of each row
        Arrays.sort(sorted, Comparator.comparingDouble(a -> a[0]));

        return sorted;

    }
```

Image 4: Method to count the numbers of flights canceled (oneCount..) and not canceled (zeroCount..) for values less than and equal to a particular row and greater than. Another method that sorts the Attributes in order for numeric attributes.

```java
public static double numericalSplitEntropy(double[][] sortedList){

    double[] values = countOfZeroAndOne(sortedList, 1);
    values[0]--;
    values[2]++;
    double maxGain = 0.0;
    for(int i=1;i<sortedList.length; i++){
        double currentGain;

        if(sortedList[i][1] == 1.0){
            values[0]++;
            values[2]--;
        }
        if(sortedList[i][1] == 0.0){
            values[1]++;
            values[3]--;
        }
        double positiveValuesLessThan = values[0];
        double negativeValuesLessThan = values[1];
        double positiveValuesGreaterThan = values[2];
        double negativeValuesGreaterThan = values[3];

        double totalPos = positiveValuesGreaterThan + positiveValuesLessThan;
        double totalNeg = negativeValuesGreaterThan + negativeValuesLessThan;
        double attributeEntropy = DecisionTreeUtils.entropy(totalPos, totalNeg);

        double lessThanEntropy = DecisionTreeUtils.entropy(positiveValuesLessThan, negativeValuesLessThan);
        double greaterThanEntropy = DecisionTreeUtils.entropy(positiveValuesGreaterThan, negativeValuesGreaterThan);

        double total = totalNeg + totalPos;
        currentGain = attributeEntropy - (((positiveValuesLessThan+negativeValuesLessThan)/total) * lessThanEntropy)
        - (((positiveValuesGreaterThan + negativeValuesGreaterThan)/ total) *greaterThanEntropy);
        if(currentGain > maxGain){
            maxGain = currentGain;
        }

    }

    return maxGain;

}
```

Image 5: Finally we have the method that calculates and returns the information gain of a numerical attribute. It goes through the whole dataset, calculates each split points entropy, subtracts it from the class entropy for information gain and keeps track of the maximum one as it traverses the dataset. In the end it will return the largest info gain.

```java
public class App
{

    private static XSSFSheet readXLSXFile(String file) {
        try {
            XSSFWorkbook work = new XSSFWorkbook(new FileInputStream(file));
            XSSFSheet sheet = work.getSheet("in");
            return sheet;

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;

    }

    Run | Debug
    public static void main( String[] args ) {
        System.out.println( "Hello World!" );

        XSSFSheet sheet = readXLSXFile("/Users/lootiikiri/Desktop/CSI4142/Training_DataSet.xlsx");


        double[][] arr = ContinuousAttributesUtil.sortedAttributesClass(sheet, 1);
        double[][] a = ContinuousAttributesUtil.sortedAttributesClass(sheet, 2);

        System.out.println(ContinuousAttributesUtil.numericalSplitEntropy(arr));
        System.out.println(ContinuousAttributesUtil.numericalSplitEntropy(a));
        System.out.println(MonthInfoGain.infoGainMonth(sheet));


    }


}
```

```
0.0023154833669960656
0.0017042592922772715
0.04746434563934398
```

**Performance Measurement for all the algorithm**

| Classifier | Construction Time | Accuracy | Precision | Recall | F_Score |
|---|---|---|---|---|---|
| Decision Tree | 2sec | 63.6% | 64.23% | 61.37% | 62.77% |

| | | | | | |
|---|---|---|---|---|---|
| Random Forest | 1m 24sec | 64.21% | 63.61% | 66.4% | 64.98% |
| Gradient Boosting | 3m 28sec | 62.57% | 61.89% | 65.45% | 63.62% |

**Analysis**

1. **Construction Time**
   The construction time is very coherent to our understanding of how each of the classifiers are ensembled. First the decision was expected to have by far the lowest construction time because only 1 tree is being constructed for the classification, where the other classifier algorithms are using multiple subtrees and trees to perform the classification. The Random Forest classifier was also expected to have a lower construction time than the Gradient boosting because unlike Gradient boosting, it only utilize one decision tree to create multiple subtree that will perform different classification algorithm creating this forest type structure, where Gradient boosting create multiple decision tree that would gradually correct the mistakes made in the prior trees.

2. **Accuracy**
   The accuracy is not coherent with our understanding of the behavior of the classifier, but It does make sense given the context of our dataset. Hypothetically, the gradient boosting should have a superior accuracy to the decision tree because it uses multiple trees that will correct itself after each iteration of the decision tree. If you look at the recall, it has a superior recall to the decision tree, which implies that the gradient boosting is very good at predicting all of the relevant cases. The problem that may have caused this situation is the class imbalance that we had, where we utilize under sampling(removing non-cancelled flight from our dataset) in order to remove the class imbalance, which the data points that were removed may have been useful to accurately predicting the negative cases. Another potential issues with the gradient boosting is that since our datasets is being reduced, the value of each attribute may be very  sparse which could impact the accuracy of the classifier

3. **Precision, Recall, F-Score**
   The F-score and the recall was coherent with our understanding of the behavior of the classifier , but the precision of the gradient is substantially lower than we anticipated. Similar to the accuracy, this may have been caused by the class imbalance that we had and how we removed many negative cases(non-cancelled flights) in order to resolve the class imbalance.