**Machine Learning from Data – IDC**

**HW2 – Decision Tree**

**\*\*\* This assignment can be submitted in pairs.**

In this assignment you will implement a Decision Tree algorithm as learned in class.

In addition to constructing the tree using the training data, your code should be able to address the following tasks.

1. Choosing an impurity measure.

In order to choose the most suitable impurity measure you will construct 2 decision trees, one that uses Entropy as the impurity measure and the other that uses Gini. After building each tree using the training set, you should calculate the error on the validation set and then choose the measure that gave you the better validation error.

1. Chi square pruning.

Your tree should be able to execute pruning with different Chi-square p-value cutoff values.

When you want to decide whether to prune or not, you need to calculate the Chi square statistic as learned in the recitation and compare this number to the relevant one from the Chi square table – according to the p-value cutoff (=alpha risk) and the degrees of freedom. In this work we will use, for a node S and an attribute A, the relevant number of values of A minus 1, as the degree of freedom (note that this is not the total number of values in A minus 1 as not all of these original values are relevant when considering the pruning of S). If the Chi square statistic is smaller than the number from the table you should prune.

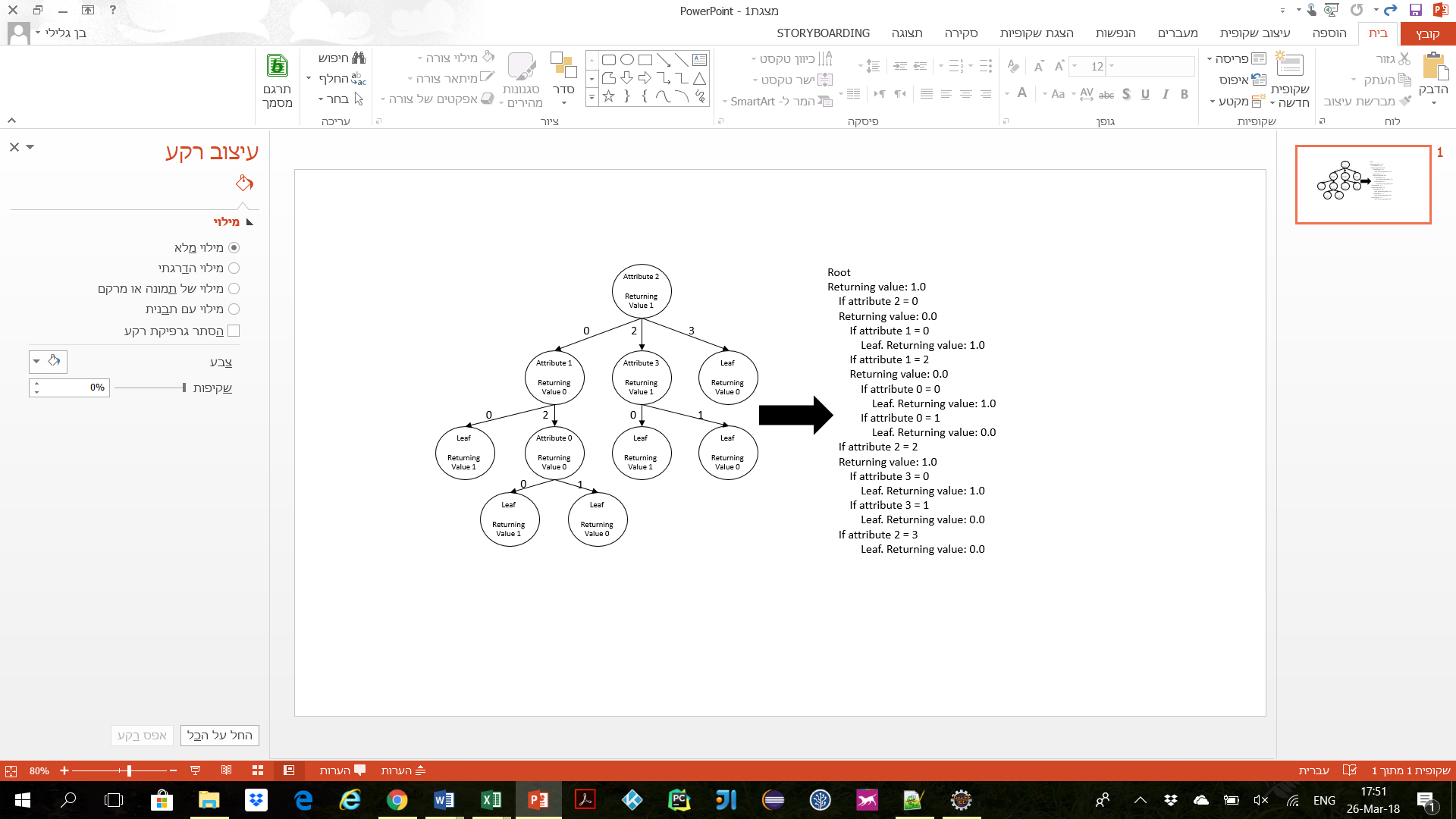
\* See the Chi square chart at the end.

1. Calculate tree height based on the validation data.

When you are checking the validation error you should also check the tree height. For each test instance, calculate the length of the path it goes through until classified. The classification height of the instance will be the height of the leaf \ internal node that classified the instance. The tree max height will be the maximum amongst the classification heights of all test instances and the tree average height will be the average of the classification heights of all test instances.

1. Print the tree.

One of the advantages of the decision tree classification algorithm is the interpretability. That is – we can easily explain a classification decision using the path that the instance traverses in the tree. But, in order to do so, we should be able to visualize the tree. An alternative approach is to represent the tree structure using 'if statements'. In this work you will print the tree in 'if statement' as shown in the following example:



Your tasks in this HW are:

1. Choosing an impurity measure:
   1. Construct a tree with Entropy as the impurity measure using the training set. Calculate the average error on the validation set.
   2. Construct a tree with Gini as the impurity measure using the training set. Calculate the average error on the validation set.
   3. Choose the impurity measure that gave you the lowest validation error. Use this impurity measure for the rest of the tasks.
2. For each p-value cutoff value do the following:
   1. Construct a tree and prune it according to the current cutoff value.
   2. Calculate training & validation errors.
   3. Calculate the tree average & max heights according to the validation set as described above.

\* For this task consider the following p-value cutoffs:  
{1 (no pruning), 0.75, 0.5, 0.25, 0.05, 0.005}.

1. Select the cutoff that resulted in the best validation error.
   1. Calculate the test error for the tree corresponding to this configuration.
   2. Print the corresponding tree to the console as described above.
2. Plot the training and validation error rates vs the p-value cutoff on the same graph (two different lines in two different colors) in Excel using the ‘Scatter with Smooth Lines and Markers’ graphing utility.

The console code output should look like this:

*Validation error using Entropy: XXX*

*Validation error using Gini: XXX*

*----------------------------------------------------*

*Decision Tree with p\_value of: XXX*

*The train error of the decision tree is XXX*

*This part should return for each p\_value.*

*Max height on validation data: XXX*

*Average height on validation data: XXX*

*The validation error of the decision tree is XXX*

*----------------------------------------------------*

*Best validation error at p\_value = XXX*

*Test error with best tree: XXX  
Representation of the best tree by ‘if statements’*

In order to perform the tasks above you need to first install WEKA:

1. See instruction in HW1.

Prepare your Eclipse project:

1. Create a project in eclipse called HomeWork2.
2. Create a package called HomeWork2.
3. Move the DecisionTree.java and MainHW2.java that you downloaded from the Moodle into this package.
4. Add WEKA to the project:
   1. See instruction in HW1.

Your goal is to predict whether a breast cancer tumor has a recurrence based on parameters of the patient and of the tumor. In order to do so you will implement the decision tree described above. For making your code more readable you will use several mandatory methods. Only in the first 2 methods (classifyInstance, buildClassifier) we supply an input and output signature that you must follow (those methods are override methods, and you must implement them accordingly), and in the rest you can implement the methods according to your discretion (we added input \ output descriptions for reference, but you can change them if you like):

1. double classifyInstance: Return the classification of the instance.
   1. Input: Instance object.
   2. Output: double number, 0 or 1, represent the classified class.
2. void buildClassifier: Builds a decision tree from the training data. buildClassifier is separated from buildTree in order to allow you to do extra preprocessing before calling buildTree method or post processing after.
   1. Input: Instances object.
3. void buildTree: Builds the decision tree on given data set using either a recursive or queue algorithm.
   1. Input: Instances object (probably the training data set or subset in a recursive method).
4. calcAvgError: Calculate the average error on a given instances set (could be the training, test or validation set). The average error is the total number of classification mistakes on the input instances set divided by the number of instances in the input set.
   1. Input: Instances object.
   2. Output: Average error (double).
5. calcGain: calculates the gain (giniGain or informationGain depending on the impurity measure) of splitting the input data according to the attribute.
   1. Input: Instances object (a subset of the training data), attribute index (int).
   2. Output: The gain (double).
6. calcEntropy: Calculates the Entropy of a random variable.
   1. Input: A set of probabilities (the fraction of each possible value in the tested set).
   2. Output: The Entropy (double).
7. calcGini: Calculates the Gini of a random variable.
   1. Input: A set of probabilities (the fraction of each possible value in the tested set).
   2. Output: The Gini (double).
8. calcChiSquare: Calculates the chi square statistic of splitting the data according to the splitting attribute as learned in class.
   1. Input: Instances object (a subset of the training data), attribute index (int).
   2. Output: The chi square score (double).

In addition to the methods described above you are more than welcome to add more methods to the required ones if it helps in making the code better organized.

The Decision Tree object holds a Node object. Think how to use this object in order to construct the tree.

You should hand in a DecisionTree.java, MainHW2.java and hw2.xlsx files. The grader will use this as well as the console output to test your work.

All of these files should be placed in a hw\_2\_##id1##\_##id2##.zip folder with the id numbers of both members of the team.

\*\*\* Submitting in groups on Moodle does not work. Please only submit one zip folder per pair

