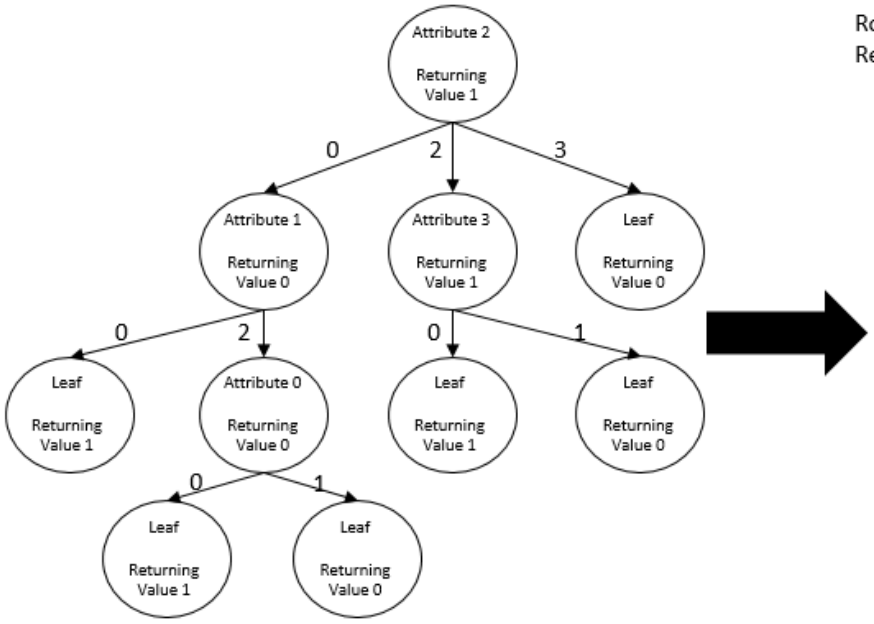
Machine Learning from Data – IDC
HW2 – Decision Tree

***** 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.
2. 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.
3. 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.
4. 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:




```
Root
Returning value: 1.0
  If attribute 2 = 0
  Returning value: 0.0
    If attribute 1 = 0
      Leaf. Returning value: 1.0
    If attribute 1 = 2
    Returning value: 0.0
      If attribute 0 = 0
        Leaf. Returning value: 1.0
      If attribute 0 = 1
        Leaf. Returning value: 0.0
  If attribute 2 = 2
  Returning value: 1.0
    If attribute 3 = 0
      Leaf. Returning value: 1.0
    If attribute 3 = 1
      Leaf. Returning value: 0.0
  If attribute 2 = 3
      Leaf. Returning value: 0.0
```

Your tasks in this HW are:

1. Choosing an impurity measure:
   a. Construct a tree with Entropy as the impurity measure using the training set. Calculate the average error on the validation set.
   b. Construct a tree with Gini as the impurity measure using the training set. Calculate the average error on the validation set.
   c. 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:
   a. Construct a tree and prune it according to the current cutoff value.
   b. Calculate training & validation errors.
   c. 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}.
3. Select the cutoff that resulted in the best validation error.
   a. Calculate the test error for the tree corresponding to this configuration.

b. Print the corresponding tree to the console as described above.

4. 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
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'

This part should return for each p_value.

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:
   a. 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.
   a. Input: Instance object.
   b. 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.
   a. Input: Instances object.
3. void buildTree: Builds the decision tree on given data set using either a recursive or queue algorithm.
   a. 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.
   a. Input: Instances object.
   b. 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.
   a. Input: Instances object (a subset of the training data), attribute index (int).
   b. Output: The gain (double).
6. calcEntropy: Calculates the Entropy of a random variable.
   a. Input: A set of probabilities (the fraction of each possible value in the tested set).
   b. Output: The Entropy (double).
7. calcGini: Calculates the Gini of a random variable.
   c. Input: A set of probabilities (the fraction of each possible value in the tested set).
   d. Output: The Gini (double).
8. calcChiSquare: Calculates the chi square statistic of splitting the data according to the splitting attribute as learned in class.
   a. Input: Instances object (a subset of the training data), attribute index (int).
   b. 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

Table of Probabilities for the Chi-Squared Distribution

	Alpha Risk													
df	**0.995**	**0.990**	**0.975**	**0.95**	**0.9**	**0.75**	**0.5**	**0.25**	**0.1**	**0.05**	**0.25**	**0.01**	**0.005**	**0.001**
1	0.000039	0.000157	0.000982	0.00393	0.0158	0.102	0.455	1.323	2.706	3.841	1.323	6.635	7.879	10.828
2	0.010	0.020	0.051	0.103	0.211	0.575	1.386	2.773	4.605	5.991	2.773	9.210	10.597	13.816
3	0.072	0.115	0.216	0.352	0.584	1.213	2.366	4.108	6.251	7.815	4.108	11.345	12.838	16.266
4	0.207	0.297	0.484	0.711	1.064	1.923	3.357	5.385	7.779	9.488	5.385	13.277	14.860	18.467
5	0.412	0.554	0.831	1.145	1.610	2.675	4.351	6.626	9.236	11.070	6.626	15.086	16.750	20.515
6	0.676	0.872	1.237	1.635	2.204	3.455	5.348	7.841	10.645	12.592	7.841	16.812	18.548	22.458
7	0.989	1.239	1.690	2.167	2.833	4.255	6.346	9.037	12.017	14.067	9.037	18.475	20.278	24.322
8	1.344	1.646	2.180	2.733	3.490	5.071	7.344	10.219	13.362	15.507	10.219	20.090	21.955	26.124
9	1.735	2.088	2.700	3.325	4.168	5.899	8.343	11.389	14.684	16.919	11.389	21.666	23.589	27.877
10	2.156	2.558	3.247	3.940	4.865	6.737	9.342	12.549	15.987	18.307	12.549	23.209	25.188	29.588
11	2.603	3.053	3.816	4.575	5.578	7.584	10.341	13.701	17.275	19.675	13.701	24.725	26.757	31.264
12	3.074	3.571	4.404	5.226	6.304	8.438	11.340	14.845	18.549	21.026	14.845	26.217	28.300	32.909
13	3.565	4.107	5.009	5.892	7.042	9.299	12.340	15.984	19.812	22.362	15.984	27.688	29.819	34.528
14	4.075	4.660	5.629	6.571	7.790	10.165	13.339	17.117	21.064	23.685	17.117	29.141	31.319	36.123
15	4.601	5.229	6.262	7.261	8.547	11.037	14.339	18.245	22.307	24.996	18.245	30.578	32.801	37.697
16	5.142	5.812	6.908	7.962	9.312	11.912	15.338	19.369	23.542	26.296	19.369	32.000	34.267	39.252
17	5.697	6.408	7.564	8.672	10.085	12.792	16.338	20.489	24.769	27.587	20.489	33.409	35.718	40.790
18	6.265	7.015	8.231	9.390	10.865	13.675	17.338	21.605	25.989	28.869	21.605	34.805	37.156	42.312
19	6.844	7.633	8.907	10.117	11.651	14.562	18.338	22.718	27.204	30.144	22.718	36.191	38.582	43.820
20	7.434	8.260	9.591	10.851	12.443	15.452	19.337	23.828	28.412	31.410	23.828	37.566	39.997	45.315
21	8.034	8.897	10.283	11.591	13.240	16.344	20.337	24.935	29.615	32.671	24.935	38.932	41.401	46.797
22	8.643	9.542	10.982	12.338	14.041	17.240	21.337	26.039	30.813	33.924	26.039	40.289	42.796	48.268
23	9.260	10.196	11.689	13.091	14.848	18.137	22.337	27.141	32.007	35.172	27.141	41.638	44.181	49.728
24	9.886	10.856	12.401	13.848	15.659	19.037	23.337	28.241	33.196	36.415	28.241	42.980	45.559	51.179
25	10.520	11.524	13.120	14.611	16.473	19.939	24.337	29.339	34.382	37.652	29.339	44.314	46.928	52.620
26	11.160	12.198	13.844	15.379	17.292	20.843	25.336	30.435	35.563	38.885	30.435	45.642	48.290	54.052
27	11.808	12.879	14.573	16.151	18.114	21.749	26.336	31.528	36.741	40.113	31.528	46.963	49.645	55.476
28	12.461	13.565	15.308	16.928	18.939	22.657	27.336	32.620	37.916	41.337	32.620	48.278	50.993	56.892
29	13.121	14.256	16.047	17.708	19.768	23.567	28.336	33.711	39.087	42.557	33.711	49.588	52.336	58.301
30	13.787	14.953	16.791	18.493	20.599	24.478	29.336	34.800	40.256	43.773	34.800	50.892	53.672	59.703
40	20.707	22.164	24.433	26.509	29.051	33.660	39.335	45.616	51.805	55.758	45.616	63.691	66.766	73.402
50	27.991	29.707	32.357	34.764	37.689	42.942	49.335	56.334	63.167	67.505	56.334	76.154	79.490	86.661
60	35.534	37.485	40.482	43.188	46.459	52.294	59.335	66.981	74.397	79.082	66.981	88.379	91.952	99.607
70	43.275	45.442	48.758	51.739	55.329	61.698	69.334	77.577	85.527	90.531	77.577	100.425	104.215	112.317
80	51.172	53.540	57.153	60.391	64.278	71.145	79.334	88.130	96.578	101.879	88.130	112.329	116.321	124.839
90	59.196	61.754	65.647	69.126	73.291	80.625	89.334	98.650	107.565	113.145	98.650	124.116	128.299	137.208
100	67.328	70.065	74.222	77.929	82.358	90.133	99.334	109.141	118.498	124.342	109.141	135.807	140.169	149.449