**Machine Learning from Data – IDC**

**HW4 – K-Nearest-Neighbor**

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

In this assignment you will implement the K-Nearest-Neighbor algorithm and a cross validation error estimate. The assignment has 2 phases:

1. Finding the best hyper parameters (K – number of neighbors, l-p distance measure, majority method) using 10-folds cross validation, for 2 different datasets ("glass" & "cancer").

In this phase you need to go over all hyper parameters combination, and for each combination to calculate the average error over all 10 folds.

The possible values for the hyper parameters are:

* 1. K – {1, 2, …, 20}
  2. l-p distance – {infinity, 1, 2, 3}
  3. Majority – {"uniform", "weighted"}

For each dataset you will choose the combination with the lower average error and print this combination and the average cross validation error.

In addition, for the cancer dataset only you will print the average Precision & Recall among the 10 folds. This dataset has 2 classes – recurrence-events, no-recurrence-events which converted to 0.0, 1.0 appropriately by WEKA. For this homework the 0 class (recurrence-events) will be the positive class and the 1 class (no-recurrence-events) will be the negative class.

Reminder:

* 1. Precision
  2. Recall

1. In this phase you will examine how the number of folds, and the edited method influence on the running time. You will use the glass dataset, and the hyper parameters that you find in the first phase.

You will run several cross validation, each time with different number of folds and different edit method. For each number of folds you will output the result for every possible edit method.

The possible values for the hyper parameters are:

* 1. {the size of the glass dataset, 50, 10, 5, 3}

\* When the number of fold equal to the dataset size you will get Leave-One-Out cross validation.

The output should contain the average error and the average time for classify all the instances in each fold, the total time for classify (which is the average multiple by the number of folds) and the amount of instances that used during the cross validation on the training set (for example, if I have 10 instances and I'm using 5-folds the amount will be 40 – 2 in each folds, 4 folds used to train and the training process was occurred 5 time, once for each fold = 2\*4\*5 = 40).

In order to do so you need to first install WEKA:

1. See instruction in HW1.

Prepare your Eclipse project:

1. Create a project in eclipse called HomeWork4.
2. Create a package called HomeWork4.
3. Move the Knn.java and MainHW4.java that you downloaded from the Moodle into this package.
4. Add WEKA to the project:
   1. See instruction in HW1.

In the first phase you will implement the 'regular' kNN algorithm (without editing method = EditMode.None). Your main method should trigger the cross validation search in order to find the best hyper parameters for each dataset. The following methods are mandatory methods, but you can add additional parameters to those methods and you can add additional methods and additional properties to the MainHW4 & Knn classes:

1. double classifyInstance: Return the classification of the instance.
   1. Input: Instance object.
   2. Output: double number, represent the classified class.
2. void buildClassifier: Builds a kNN from the training data. The method is already implemented using switch statement on the enum EditMode. This enum set the edit mode to one of its possibilities (None, Forwards, Backwards). You should implement each one of the helper methods noEdit, editedForward and editedBackward (the last 2 describe later).
   1. Input: Instances object.
3. void noEdit: Store the training set in the m\_trainingInstances without editing.
   1. Input: Instances object.

\* This method is already implemented.

1. calcAvgError: Calculate the average error on a given instances set. The average error is the total number of classification mistakes on the input instances set and divides that by the number of instances in the input set.
   1. Input: Instances object.
   2. Output: Average error (double).
2. calcConfusion: Calculate the Precision & Recall on a given instances set.
   1. Input: Instances object.
   2. Output: double array of size 2. First index for Precision and the second for Recall.
3. crossValidationError: Calculate the cross validation error = average error on all folds.
   1. Input: Instances object
   2. Output: Average fold error (double)
4. findNearestNeighbors:Find the K nearest neighbors for the instance being classified.
   1. Input: an instance
   2. Output: finds the K nearest neighbors (and perhaps their distances)
5. getClassVoteResult:Calculate the majority class of the neighbors
   1. Input: a set of K nearest neighbors
   2. Output: the majority vote on the class of the neighbors
6. getWeightedClassVoteResult: Calculate the weighted majority class of the neighbors. In this method the class vote is normalized by the distance from the instance being classified. Instead of giving one vote to every class, you give a vote of .
   1. Input: a set of K nearest neighbors (and perhaps their distances)
   2. Output: the majority vote on the class of the neighbors, where each neighbor's class is weighted by the neighbor’s distance from the instance being classified.
7. distance:
   1. Input: two instances
   2. Output: the input instances’ distance according to the distance function that your algorithm is configured to use.
8. lPDistance:
   1. Input: two instances
   2. Output: the l-p distance between the two instances

note: p can be a variable of your class or you can set p some other way

1. lInfinityDistance:
   1. Input: two instances
   2. Output: the l-infinity distance between two instances

Points for thought:

1. In order to get a good result in the cross validation, you should shuffle your data, think where you need to do it.
2. How and where you should iterate over all hyper parameters combinations
3. Where to put the relevant properties.

In the end of this phase you should print the cross validation error (the output of crossValidationError) of your algorithm with the best parameters for the glass and cancer datasets and for cancer dataset you should print the average Precision & Recall as well.

In the second phase you will add to your algorithm the 'edited kNN' ability. As we saw in class the edited kNN is a greedy algorithm that 'prune' the training set. After you'll implement the edited kNN you will calculate some measurement on each number of fold and on each kNN algorithm (regular, forwards, and backwards). You need to output for each number of fold the following:

1. The edited method (None, Forwards, Backwards).
2. The average error of the cross validation.
3. The average elapsed time of the classification of 1 fold in the cross validation.
4. The total elapsed time for the classification in the cross validation.
5. The total training instances that used in the classification during the cross validation.

Remember: before splitting the dataset for the cross validation, you need to shuffle the data. Think where you need to do it.

In order to calculate the running time use the java method System.nanoTime(). Think where you should use this method and remember that we want to measure only the classification time.

You need to implement the following method:

1. void editedForward: Store the training set in the m\_trainingInstances using the forwards editing.
   1. Input: Instance object.
2. void editedBackward: Store the training set in the m\_trainingInstances using the backwards editing.
   1. Input: Instance object.

Your whole output should be like this:

Cross validation error with K = <K\_value>, p = <P\_value>, majority function = <weighted or uniform> for glass data is: <cross\_vaidation\_error>

Cross validation error with K = <K\_value>, p = <P\_value>, majority function = <weighted or uniform> for cancer data is: <cross\_vaidation\_error>

The average Precision for the cancer dataset is: <Precision\_value>

The average Recall for the cancer dataset is: <Recall\_value>

----------------------------

Results for <number\_of\_folds> folds:

----------------------------

Cross validation error of None-Edited knn on glass dataset is <error> and the average elapsed time is <average\_elapsed\_time\_in\_nano\_seconds>

The total elapsed time is: <total\_elapsed\_time\_in\_nano\_seconds>

The total number of instances used in the classification phase is: <number of training instances>

Cross validation error of Forwards-Edited knn on glass dataset is <error> and the average elapsed time is <average\_elapsed\_time\_in\_nano\_seconds>

The total elapsed time is: <total\_elapsed\_time\_in\_nano\_seconds>

The total number of instances used in the classification phase is: <number of training instances>

Cross validation error of Backwards-Edited knn on glass dataset is <error> and the average elapsed time is <average\_elapsed\_time\_in\_nano\_seconds>

The total elapsed time is: <total\_elapsed\_time\_in\_nano\_seconds>

The total number of instances used in the classification phase is: <number of training instances>

The yellow part should reoccur for each number of folds.

In addition to the code output, add to your txt file answers to the following questions:

1. In general, what is the influence of the number of folds on the running time?
2. Is there a connection between your answer for the first question to the number of instances that used in the classification (the number that you printed)? If yes, what is the connection? If not, explain why not?

You should hand in a Knn.java, MainHW4.java and hw4.txt files which the grader will use to test your implementation. All of these files should be placed in a hw\_4\_##id1##\_##id2##.zip folder with the id of both of the members of the group.

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