Machine Learning from Data – IDC
HW3 – K-Nearest-Neighbor

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

In this assignment you will try to predict the price of a car using the K-Nearest-Neighbor algorithm and a cross validation error estimate.

<u>The assignment has 3 phases:</u>

1. <u>Feature Scaling</u> – Before running your kNN algorithm you will need to build a class to scale the features in the data. The scaling function we will use in this exercise is Standardization (z-score scaling).
   Meaning, for each feature value you'll need to perform the following:
   $$x' = \frac{x - mean(x)}{std(x)}$$
   Where mean(x) is the mean of all the values attained for this feature, in the training data. And std(x) is the standard deviation of all the values attained for this feature, in the training data.
   You are allowed and **advised** to use Weka libraries to preform the scaling.

2. <u>Hyper Parameters Search</u> – Next, you'll try to find the best hyper parameters (K – number of neighbors, Lp distance measure, weighting scheme) using 10-folds cross validation, for the original auto_price dataset and the scaled auto_price dataset.
   You'll need to iterate over all hyper parameter combinations, and for each combination, to calculate the average error over all 10 folds.
   The values to consider for the hyper parameters in this phase are:
   a. K – {1, 2, …, 20}
   b. Lp distance – {1, 2, 3, infinity}
   c. Weighting Scheme – {"uniform", "weighted"}
   At the end, for each dataset (original, scaled) you will choose the hyper parameters combination with the lowest average error and print this combination and the corresponding average cross validation error.

3. Number of folds and efficient distance check - In this phase you will examine how the number of folds, and how an efficient distance check influence the running time. You will use the scaled auto_price dataset, and the appropriate hyper parameters as found in the second phase.
   In this context, efficient distance check means you stop your distance calculations once your distance check is greater than the current k'th neighbor distance.
   Note that if you still haven't found k neighbors you don't perform this check.
   Remember that the Lp distance calculation is:

$$distance(x^1, x^2) = \sqrt[p]{\sum_{i=1}^{n} \left(|x_i^1 - x_i^2|\right)^p}$$

   So, if during the iteration for calculating this sum you reached a value which is bigger than the distance of the k'th neighbor to the power p then you stop iterating.
   For example, say p = 2, k=3 and you found d1=2, d2=5 and d3=10.
   Then while calculating a distance from a new instance, once your iteration sum is greater than 100=10^2 you stop iterating.
   You will run several cross validations, each time with a different number of folds and a regular and efficient distance check. For each number of folds you will output the result, including running times, with and without the distance check.
   The possible values for the hyper parameters in this phase are:
   a. Folds = {the size of the auto_price dataset, 50, 10, 5, 3}
   b. Distance check = {regular, efficient}

   When the number of folds is equal to the dataset size you are in effect running Leave-One-Out cross validation.
   The output should contain the average error, the average time for predicting all the instances in each fold, the total prediction time (which is the average multipled by the number of folds) and the distance check used.

Implementation Details:

1. For the first phase you will need to implement the FeatureScaler class provided to you in the exercise files.
   This class will be in charge of scaling your dataset in the desired fashion. You'll need to decide if you'll use this class inside the Knn class or in the main class.
   - Look in FeatureScaler.java for detailed implementation details.

2. In the second phase you will implement the Knn and MainHW3 classes. Inside the Knn.java file you also have a DistanceCalculator class. For this phase your DistanceCalculator class should perform a regular distance check. Your main method should iterate over all possible hyperparameters sets and trigger the cross-validation estimation in order to find the best ones, as mentioned before you will use 10-fold cross validation.
   In the end of this phase you should print the lowest cross-validation error (the output of crossValidationError) of your algorithm along with the parameters that attained it. This should be done for each dataset (regular, scaled).
   - Look in Knn.java and MainHW3.java for detailed implementation details.

3. In the third phase you will add to your algorithm the 'efficient distance check' capability. After you'll implement the efficient kNN you will calculate some measurements on each number of folds and on each kNN algorithm (regular, efficient). You need to output for each number of folds the following:
   a. The distance check method used (regular, efficient).
   b. The average error of the cross validation.
   c. The average elapsed time of the prediction of a single fold in the cross validation.
   d. The total elapsed time for the prediction in the cross validation.

   Remember: before splitting the dataset for the cross validation, you need to shuffle the data.
   In order to calculate the running time, use the java method System.nanoTime().
   Notice you should calculate the prediction time only, without the learning (which in this case should be negligible, but it's a good practice).
   - Look at the DistanceCalcularor in the Knn.java file for implementation details.

   Keep in mind we provide you with a flexible skeleton.
   Feel free to add parameters to methods, to add variables to classes and of course to add methods and classes as you see fit.
   * Do not change the supplied methods and classes names.

<u>Output Guidelines:</u>

Your whole output should be like this:

```
----------------------------
Results for original dataset:
----------------------------
Cross validation error with K = <K_value>, lp = <lP_value>, majority function
= <weighted or uniform> for auto_price data is: <cross_vaidation_error>


----------------------------
Results for scaled dataset:
----------------------------
Cross validation error with K = <K_value>, lp = <lP_value>, majority function
= <weighted or uniform> for auto_price data is: <cross_vaidation_error>


----------------------------
Results for <number_of_folds> folds:
----------------------------
Cross validation error of regular knn on auto_price 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>

Cross validation error of efficient knn on auto_price 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 yellow part should reoccur for each number of folds.

<u>Theoretical questions:</u>

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

1. In general, why do we expect feature scaling to have a positive effect on our kNN algorithm? Would we expect to have a positive effect of feature scaling in the context of decision tree algorithms?
2. In class we saw we can perform an edited kNN algorihtm which used either backward or forward kNN to filter out instances.
   Could we use this procedure for our dataset? If so explain how, if not explain why.

<u>Project setup:</u>

Install WEKA:

1. See instruction in HW1.

Prepare your Eclipse project:

2. Create a project in eclipse called HomeWork3.
3. Create a package called HomeWork3.

4. Move the Knn.java and MainHW3.java that you downloaded from the Moodle into this package.
5. Add WEKA to the project:
   a. See instruction in HW1.

You should hand in a Knn.java, FeatureScaler.java MainHW3.java and hw3.txt files which the grader will use to test your implementation. All of these files should be placed in a hw_3_##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 <u>one</u> zip folder per pair