Introduction to Machine Learning
Work 3
Lazy learning exercise

Contents

1 Lazy learning exercise

1.1 Introduction

In this exercise, you will learn about lazy learning. In particular, you will work on a comparative study in the user of similarity measures, feature selection methods and instance reduction methods in a k-Nearest Neighbor algorithm. You will apply lazy learning to a classification task. It is assumed that you are familiar with the concept of cross-validation. If not, you can read this paper:

[1] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the International Joint Conferences on Artificial Intelligence IJCAI-95. 1995.

Briefly, an s-fold cross validation (s = 10 in your case) divides a data set into s equal-size subsets. Each subset is used in turn as a test set with the remaining (s-1) data sets used for training. The data sets with a predefined 10-fold cross validation is provided in Campus Virtual.

For the validation of the different algorithms, you need to use a T-Test or another statistical method. Next reference is a **mandatory reading proposal** (in Campus Virtual) on this topic:

[2] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. J. Mach. Learn. Res. 7 (December 2006), 1-30.

This article details how to compare two or more learning algorithms with multiple data sets.

1.2 Methodology of the analysis

As in the previous work assignment, you will analyze the behavior of the different algorithms by comparing the results in a pair of well-known data sets (**medium and large size**) from the UCI repository. In that case, you will also use the class as we are testing several supervised learning algorithms. In particular, in this exercise, you will receive the data sets defined in .arff format but divided in **ten training and test sets** (they are the *10-fold cross-validation* sets you will use for this exercise).

This work is divided in several steps:

1. Read the source data for training and testing the algorithms. Improve the parser developed in previous works in order to use the class attribute, too. Now, you need to read and save the information from a training and their corresponding testing file in a `TrainMatrix` and a `TestMatrix`, respectively. Recall that you need to normalize all the numerical attributes in the range [0..1]. For representing the case base, the most used by its simplicity and applicability is a flat structure. The cases are represented as a feature vector approach by means of a set of attribute-value pairs. The `TrainMatrix` and the `TestMatrix` contain the set of cases for training and testing, respectively.

2. Write a Python function that automatically repeats the process described in previous step for the 10-fold cross-validation files. That is, read automatically each training case and run each one of the test cases in the selected classifier.
3. Write a Python function for classifying, using a kNN algorithm, each instance from the `TestMatrix` using the `TrainMatrix` to a classifier called `kNNAlgorithm(...)`. You decide the parameters for this classifier. **Justify your implementation and add all the references you have considered for your decisions.**

   Let us assume that we have a training dataset D made up of $(x_i)_{i\in[1,n]}$ training samples (where $n = |D|$). The examples are described by a set of features F and any numeric features have been normalized to the range $[0,1]$. Each training example is labelled with a class label $y_j \in Y$. Our objective is to classify an unknown example *q*. For each $x_i \in D$ we can calculate the distance between *q* and x_i as follows:

$$d(q, x_i) = \sum_{f\in F} w_f\ \delta(q_f, x_{if})$$

   This is the summation over all the features in F with a weight for each feature.

   a. There are large range of possibilities for the distance metric, δ. In this work, you should consider the Minkowski (with r = 1, r = 2) and another EXTRA (you decide which one). Adapt these distances to handle all kind of attributes (i.e., numerical and categorical). Assume that the kNN algorithm returns the `K` most similar instances (i.e., also known as cases) from the `TrainMatrix` to *q*. The value of `K` will be setup in your evaluation to 1, 3, 5, and 7.

   b. There are a variety of ways in which k nearest neighbors can be used to determine the class of the query q, you may consider using three policies: Majority class, Inverse Distance Weighted votes, and Shepard's work.

      i. **Majority class** is the simplest method. This approach assigns the majority class among the nearest neighbors to *q*. Technically, a method for breaking ties should also be specified. You should decide the method in case of ties.
      ii. **Inverse Distance Weighted votes** assigns more weight to the nearer neighbors in deciding the class of *q*. The simplest version is to take a neighbor's vote to be the inverse of its distance to *q*. In case of ties, you should decide the method for breaking ties. You must have a clear winner solution.

$$Vote(y_j) = \sum_{c=1}^{k} \frac{1}{d(q, x_c)^p}\ 1(y_j, y_c)$$

The vote assigned to class y_j by neighbor x_c is 1 divided by the distance to that neighbor (i.e., $1(y_j, y_c)$ returns 1 if the class labels match and 0 otherwise. In the equation, *p* would normally be 1 but values greater than 1 can be used to further reduce the influence of more distant neighbors.

iii. **Sheppard's work** uses an exponential function rather than the inverse distance.

$$Vote(y_j) = \sum_{c=1}^{k} e^{-d(q,x_c)}\, 1(y_j\,, y_c)$$

c. There are different ways of weighting features in a kNN algorithm. You will analyze three different ways: Equal weight,
   i. **Equal Weight**, all the features are considered all of them are equal and then assign a weight value of 1.0 to all the attributes.
   ii. Weights can be extracted using a weighting metrics. You may choose **two algorithms** (filter or wrapper, as you wish). Use them as a preprocessing step. This means that you will only compute weights in the initial training case-base. For example, you can use *ReliefF*, *Information Gain*, or the *Correlation*, among others. There are several Python Libraries that also include most of the well-known metrics for feature weighting. You can use the implementations that exist in Python for your feature weighting implementations.

d. For evaluating the performance of the kNN algorithm, we will use the classification **accuracy** (i.e., the average of correctly classified cases) and the **efficiency** (i.e., average problem-solving time). To this end, at least, you should store the number of cases correctly classified, the number of cases incorrectly classified, and the problem-solving time. This information will be used for the evaluation of the algorithm. You can store your results in a memory data structure or in a file. Keep in mind that you need to compute the average accuracy over the 10-fold cross-validation sets.

At the end, you will have a kNN algorithm with several similarity functions (Minkowski r= 1, Minkowski r= 2, and the one you will choose), different values for the K parameter, three policies for deciding the solution of the query, *q,* and three different ways of assigning a weight to the features. You should analyze the behavior of these parameters in the kNN algorithm and decide which combination results in the **best KNN algorithm**.

You can compare your results in terms of classification accuracy and efficiency (in time). Extract conclusions by analyzing **two large** enough data sets. **At least one of these data sets will contain numerical and nominal data.**

4. Departing from the best kNN algorithm. Modify the kNN algorithm so that it includes a preprocessing for reducing the training set, you will call this algorithm as `reductionKNNAlgorithm(...)`. In general, kNN works with all the training set. However, the training set may be too large and it may also contain inconsistent and noisy cases. For this reason, it is important to decide which cases to store for use during the generalization. In the literature, techniques for obtaining a representative training set with a lower size are called reduction techniques, instance selection or prototype selection methods. In this work, you will implement several preprocessing reduction techniques:
    a. Select one condensed reduction technique: Selective Nearest Neighbor (**SNN**) or Fast Condensed Nearest Neighbor (**FCNN**) or Modified Condensed Nearest Neighbor (**MCNN**) algorithm.
    b. Select one edited reduction technique: **ENN** or Modified ENN (**MENN**) or **ENNTh**
    c. Select one hybrid reduction technique: **IB2** or **IB3** or **DROP2** or **DROP3**
    d. Analyze the results of the `reductionKNNAlgorithm` in front of the previous `kNNAlgorithm` implementation. To do it, setup both algorithms with the best combination obtained in your previous analysis. In this case, you will analyze your results in terms of classification **accuracy**, **efficiency** and **storage** (i.e, the average percentage of cases stored from the training to generalize the testing cases). Accordingly, the storage of the `kNNAlgorithm` is 100% since it does not reduce the training set.

1.3 Work to deliver

In this work, you will implement K-Nearest Neighbor algorithm with weighting and with reduction of the training set. You may select two data sets (large enough to extract conclusions) for your analysis. At the end, you will find a list of the data sets available.

You will implement your own code in Python and use it to extract the performance of the different combinations. Performance will be measured in terms of classification accuracy, efficiency, and storage. The accuracy measure is the average of correctly classified cases. That is the number of correctly classified instances divided by the total of instances in the test file. The efficiency is the average problem-solving time. For the evaluation, you will use a T-Test or another statistical method [2].

From the accuracy and efficiency results, you will extract conclusions showing graphs of such evaluation and reasoning about the results obtained.

In your analysis, you will include several considerations.

1. You will analyze the KNN (with different parameters). You will analyze which is the most suitable combination of the different parameters analyzed. The one with the highest accuracy. This KNN combination will be named as the best KNN. Recall to perform an statistical analysis.

2. Once you have decided the best KNN combination. You will analyze it in front of using this combination with three different reduction techniques. The idea is to analyze if reduction techniques may increase accuracy, efficiency and storage in a k-NN algorithm and to extract conclusions of which reduction technique is the best one, if there is any. Recall to perform an statistical analysis.

For example, some of questions that it is expected you may answer with yur analysis:

- Which is the best value of K at each dataset?
- Which k-NN algorithm is the best one at each dataset?
- Did you find useful the use of a voting scheme for deciding the solution of the query?
- Which is the best similarity function for the k-NN?
- Did you find differences in performance among the k-NNL and the reduction k-NN?
- According to the data sets chosen, which reduction method provides you more advice for knowing the underlying information in the data set?
- Do you think it will be much better to perform feature selection rather than reducing the training set?

Apart from explaining your decisions and the results obtained, it is expected that you reason each one of these questions along your evaluation.

Additionally, **you should explain how to execute your code**. Remember to add any reference that you have used in your decisions.

You should deliver a report as well as the code in Python in a PyCharm project in Campus Virtual by **December, 21st, 2020**.

Remember that the maximum size of the report is 20 pages, including description and graphs. Excluded are the cover page and the references.

2 Data sets

Below, you will find a table that shows in detail the data sets that you can use in this work. All these data sets are obtained from the UCI machine learning repository. First column describes the name of the domain or data set. Next columns show #Cases = Number of cases or instances in the data set, #Num. = Number of numeric attributes, #Nom = Number of nominal attributes, #Cla. = Number of classes, Dev.Cla. = Deviation of class distribution, Maj.Cla. = Percentage of instances belonging to the majority class, Min.Cla. = Percentage of instances belonging to the minority class, MV = Percentage of values with missing values (it means the percentage of unknown values in the data set). When the columns contain a '-', it means a 0. For example, the Glass data set contains 0 nominal attributes and it is complete as it does not contain missing values.

	Domain	#Cases	#Num.	#Nom.	#Cla.	Dev.Cla.	Maj.Cla.	Min.Cla.	MV
	Adult	48,842	6	8	2	26.07%	76.07%	23.93%	0.95%
	Audiology	226	-	69	24	6.43%	25.22%	0.44%	2.00%
	Autos	205	15	10	6	10.25%	32.68%	1.46%	1.15%
*	*Balance scale*	625	4	-	3	18.03%	46.08%	7.84%	-
*	*Breast cancer Wisconsin*	699	9	-	2	20.28%	70.28%	29.72%	0.25%
*	*Bupa*	345	6	-	2	7.97%	57.97%	42.03%	-
*	*cmc*	1,473	2	7	3	8.26%	42.70%	22.61%	-
	Horse-Colic	368	7	15	2	13.04%	63.04%	36.96%	23.80%
*	*Connect-4*	67,557	-	42	3	23.79%	65.83%	9.55%	-
	Credit-A	690	6	9	2	5.51%	55.51%	44.49%	0.65%
*	*Glass*	214	9	-	2	12.69%	35.51%	4.21%	-
*	*TAO-Grid*	1,888	2	-	2	0.00%	50.00%	50.00%	-
	Heart-C	303	6	7	5	4.46%	54.46%	45.54%	0.17%
	Heart-H	294	6	7	5	13.95%	63.95%	36.05%	20.46%
*	*Heart-Statlog*	270	13	-	2	5.56%	55.56%	44.44%	-
	Hepatitis	155	6	13	2	29.35%	79.35%	20.65%	6.01%
	Hypothyroid	3,772	7	22	4	38.89%	92.29%	0.05%	5.54%
*	*Ionosphere*	351	34	-	2	14.10%	64.10%	35.90%	-
*	*Iris*	150	4	-	3	-	33.33%	33.33%	-
*	*Kropt*	28,056	-	6	18	5.21%	16.23%	0.10%	-
*	*Kr-vs-kp*	3,196	-	36	2	2.22%	52.22%	47.78%	-
	Labor	57	8	8	2	14.91%	64.91%	35.09%	55.48%
*	*Lymph*	148	3	15	4	23.47%	54.73%	1.35%	-
	Mushroom	8,124	-	22	2	1.80%	51.80%	48.20%	1.38%
*	*Mx*	2,048	-	11	2	0.00%	50.00%	50.00%	-
*	*Nursery*	12,960	-	8	5	15.33%	33.33%	0.02%	-
*	*Pen-based*	10,992	16	-	10	0.40%	10.41%	9.60%	-
*	*Pima-Diabetes*	768	8	-	2	15.10%	65.10%	34.90%	-
*	*SatImage*	6,435	36	-	6	6.19%	23.82%	9.73%	-
*	*Segment*	2,310	19	-	7	0.00%	14.29%	14.29%	-
	Sick	3,772	7	22	2	43.88%	93.88%	6.12%	5.54%
*	*Sonar*	208	60	-	2	3.37%	53.37%	46.63%	-
	Soybean	683	-	35	19	4.31%	13.47%	1.17%	9.78%
*	*Splice*	3,190	-	60	3	13.12%	51.88%	24.04%	-
*	*Vehicle*	946	18	-	4	0.89%	25.77%	23.52%	-
	Vote	435	-	16	2	11.38%	61.38%	38.62%	5.63%
*	*Vowel*	990	10	3	11	0.00%	9.09%	9.09%	-
*	*Waveform*	5,000	40	-	3	0.36%	33.84%	33.06%	-
*	*Wine*	178	13	-	3	5.28%	39.89%	26.97%	-
*	*Zoo*	101	1	16	7	11.82%	40.59%	3.96%	-