Introduction to Machine Learning
Work 3
K-Nearest Neighbor exercise

Contents

1 K-Nearest Neighbor exercise

1.1 Introduction

In this exercise, you will learn about lazy learning. In particular, you will work on analyzing several similarity functions and weighting methods in a K-Nearest Neighbor algorithm. You will apply K-Nearest Neighbor 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 Articial 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.

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** 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. 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]. Next you have an example of how to normalize one attribute of your data and how to get your original data back:

```
bla = 100.*randn(1,10)

minVal = min(bla);
maxVal = max(bla);

norm_data = (bla - minVal) / ( maxVal - minVal )
your_original_data = minVal + norm_data.*(maxVal - minVal)
```

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.**

4. For the similarity function, you should consider the Hamming, Euclidean, Cosine, and another EXTRA (you decide which one) distance functions. 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 the `current_instance`. The value of `K` will be setup in your evaluation to 1, 3, 5, and 7.

   a. To decide the solution of the `current_instance`, you may consider using two policies: the most similar retrieved case and a voting policy.

   b. For evaluating the performance of the KNN algorithm, we will use the percentage of correctly classified instances. To this end, at least, you should store the number of cases correctly classified, the number of cases incorrectly classified. 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 (Hamming, Euclidean, Cosine, and the one you will choose), different values for the `K` parameter, and two policies for deciding the solution of the `current_instance`. 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. Extract conclusions by analyzing **two large** enough data sets. **At least one of these data sets will contain numerical and nominal data**.

5. Modify the kNN algorithm so that it includes a weighted similarity, you will call this algorithm as `weightedKNNalgorithm(...)`. The weights will be extracted using weighting or feature selection algorithms.
   a. Modify the similarity functions in the kNN so that it implements a weighted function. Each weight denotes if an attribute is considered or not for the similarity. A weight value

of 1.0 denotes that the attribute will be used by the similarity. By contrast, a weight value of 0.0 shows that the attribute is useless and it is not going to be used.

The 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.

b. Analyze the results of the `weightedKNNalgorithm` 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.

6. Modify the weightedKNN algorithm so that it only uses a subset of the most relevant features, you will call this algorithm as `selectionKNNalgorithm(...)`. The selection will be extracted using weighting with an appropriate policy to discard features or feature selection algorithms.
    a. Modify the similarity functions in the kNN so that it includes a feature selection function. The feature selected will have a weight value of 1.0 and the features not selected will have a weight value of 0 in the similarity function.

    You may choose **two algorithms** (filter or wrapper, as you wish). Use them as a preprocessing step. You may choose a feature selection algorithm or a feature weighting with an appropriate policy to decide which features should be maintained and which ones should be discarded in the similarity computation. You can use the implementations that exist in Python for your feature weighting implementations.

    b. Analyze the results of the `selectionKNNalgorithm` in front of the previous `weightedKNNalgorithm` 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.

1.3 Work to deliver

In this work, you will implement K-Nearest Neighbor algorithm with weighting and with feature selection. 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 use your code in Python to extract the performance of the different combinations. Performance will be measured in terms of classification accuracy and efficiency. 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 no weighting or selection). 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.

2. Once you have decided the best KNN combination. You will analyze it in front of using this combination with two feature selection algorithms. The idea is to extract conclusions of which feature selection algorithm is the best one.

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

- Which is the best value of K at each dataset?
- Did you find useful the use of a voting scheme for deciding the solution of the current_instance?
- Which is the best similarity function for KNN?
- Did you find differences in performance among the KNN and the weighted KNN?
- According to the data sets chosen, which feature selection algorithm provides you more advice for knowing the underlying information in the data set?
- In the case of the feature selection KNN, how many features where removed? Which are the features selected for each one the feature selection algorithms?
- Which criterion have you used to decide the features that should be removed?

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 Racó by December, 10th, 2017.

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%	-