Introduction to Machine Learning Work Adaptive Case-based reasoning exercise

Contents

1 Adaptive Case-based Reasoning exercise

1.1 Introduction

In this exercise, you will learn about case-based reasoning and maintenance with an adaptive case-based reasoning implementation in Python. You will apply these techniques 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 **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

In this 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 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 files in arff format and save the training or testing file in a data structure called `TrainCaseBase` or `TestCaseBase`, respectively (you should choose the data structure that best suits in your ACBR implementation).
   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)
```

Additionally, write a Python function that automatically repeats the process described in this initial 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.

2. Write a Python function or algorithm for classifying each instance from the `TestCaseBase` using the `TrainCaseBAse` to a classifier called `acbrAlgorithm(...)`(see reference [3]). You decide the parameters for this classifier and for each one of the phases of the algorithm. This ACBR function/algorithm will call each one of the phases of the ACBR. Write a function for each one of the phases of the ACBR. **Justify your implementation and add all the references you have considered for your decisions.** The paper that details the ACBR cycle is:

   [3] Adaptive case-based reasoning using retention and forgetting strategies. Knowledge-Based Systems. 24, 2 (March 2011), 230-247 by M. Salamó and M. López-Sánchez. 2011. DOI=10.1016/j.knosys.2010.08.003 http://dx.doi.org/10.1016/j.knosys.2010.08.003

   a. For the retrieval, `acbrRetrievalPhase(...)` function, you should consider the Euclidean distance. Adapt the Euclidean distance to handle all kind of attributes. Assume that the retrieval phase returns the `K` most similar instances (i.e., also known as cases) from the `TrainCaseBase` to the `current_instance`. The value of `K` will be setup in your evaluation to 3, 5, and 7.

   b. For the reuse, `acbrReusePhase(...),` you may consider to use the most similar retrieved case or alternatively to use a voting policy. The resulting case will be used to provide a solution to the `current_instance`.

   c. For the revision, `acbrRevisionPhase(...),` you do not need to implement it for the ACBR purpose. The revision phase will serve you for analyzing if the algorithm has correctly solved the new problem (i.e., `current_instance`). It will be used for the evaluation of the algorithm. In this phase you can store your results in a memory data structure or in a file.

   d. For the review phase, `acbrReviewPhase(...),` you will implement the update goodness measure and the oblivion by goodness forgetting strategy used in [3].

   e. For retaining, `acbrRetentionPhase(...),` you will implement two of the policies described in [3]: DD, DE, MG or LE. Additionally, you will implement two opposite retention strategies: NR (never retain), which does not change the case base, and AR (always retain), which increases its case base with the new testing cases. The last policy retains all the new solved cases, and you will store the solution with the real one. These two approaches will be called `acbrNoRetentionPhase()` and `acbrAlwaysRetentionPhase()`.

At the end, you will have ACBR with a retrieval function (Euclidean), a reuse function (that you decide), a review function and four retention policies (2 simple ones and 2 selected from [3]). You should analyze the behavior of these functions in the ACBR algorithm and decide which combination results in the best ACBR algorithm. Notice that you can analyze separately the retention and the oblivion so that you will obtain several combinations of your ACBR algorithm. For example, if you have chosen DD strategy you can also analyze DD-O (that is DD with oblivion). You can compare your results in terms of classification accuracy, efficiency, and final case-base size. Extract conclusions by analyzing **three large** enough data sets (i.e., one small, one medium, and one large). At least one of these data sets will contain numerical and nominal data.

f. Additionally, compare the ACBR to a non-adaptive case-based reasoner. That is, a simple CBR that does not apply on-line maintenance. For implementing this simple CBR you can use your initial ACBR with no revision and no oblivion, instead, only use the policies `acbrNoRetentionPhase()` and `acbrAlwaysRetentionPhase()`.

3. Modify the retrieval similarity function in the CBR so that it implements a weighted function called `weightedACBRRetrievalPhase()`. 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 metric or a feature selection algorithm. You may choose **two algorithms** (filter or wrapper, as you wish). Use them as a preprocessing step. For example, you can use ReliefF, Information Gain, or the Correlation, among others. There are several Python toolboxes that also include most of the well-known metrics for feature selection algorithms. You can use the implementations that exist in Python for your feature selection implementations.

   Analyze the results of the `weightedACBRalgorithm` in front of the previous `acbrAlgorithm` implementation. To do it, setup both algorithms with the best combination obtained in your previous analysis (step 2). In this case, you will only analyze your results in terms of classification accuracy.

The schedule for these steps is as follows:

- Week 1. Steps 1, 2a, 2b, 2c
- Week 2. Steps 2d, 2e, 3

1.3 Work to deliver

In this work, you will implement an adaptive case-based reasoning and you will compare it to a simple CBR. Additionally, you will implement an adaptive case-based reasoning with feature selection. You may select 3 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, efficiency and case-base size**. 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, efficiency and case-base size 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 ACBR (with no weighting or selection). You will analyze which is the most suitable combination of phases for the ACBR. The one with the highest accuracy. This ACBR combination will be named as the best ACBR.

2. Once you have decided the best ACBR combination. You will analyze it in front of a simple CBR. Optionally, you will also analyze the combination of the best ACBR 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:

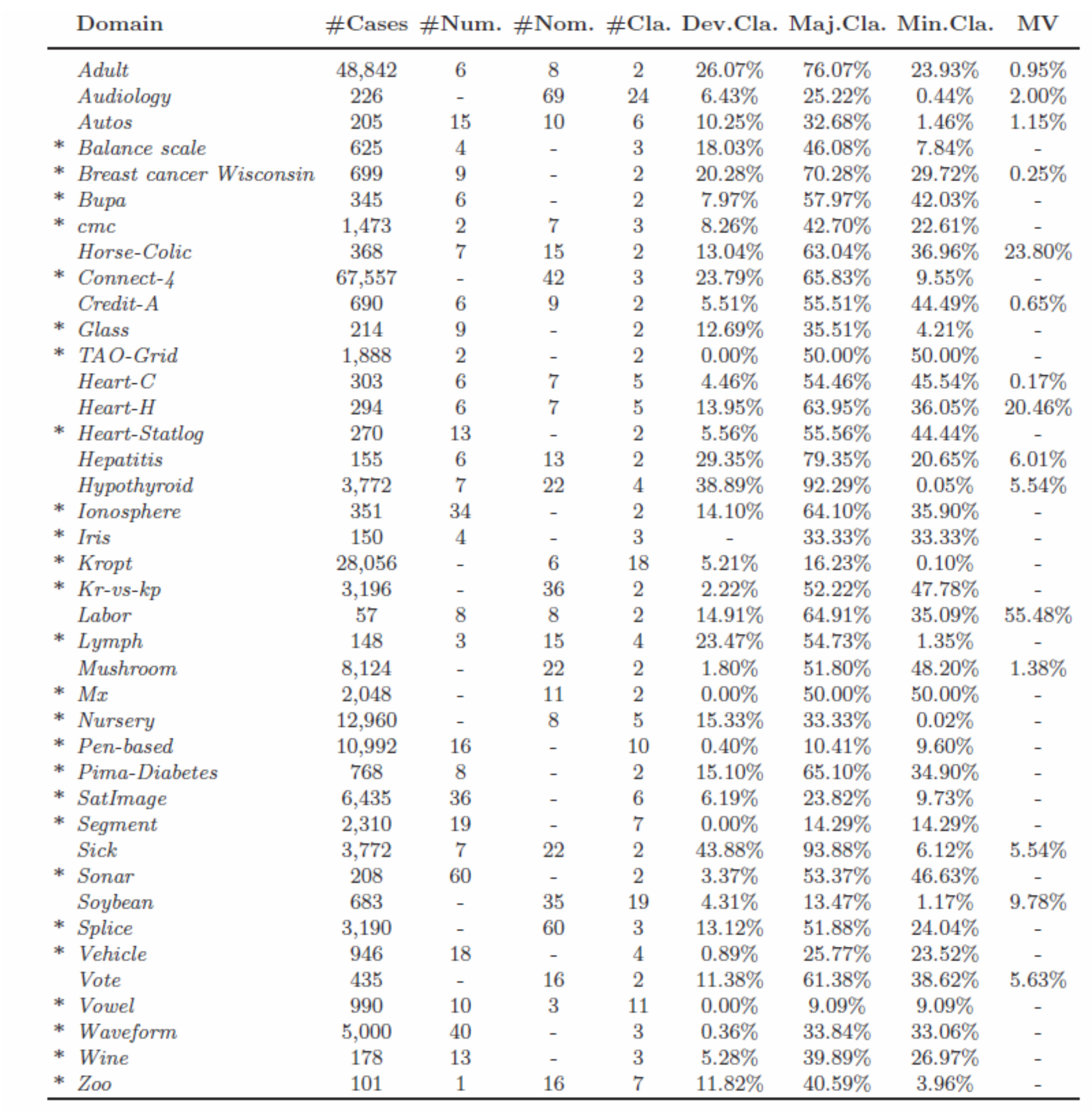
- Which is the best value of K for the retrieval and (optionally) for the reuse phase?
- Which is the best configuration for the ACBR?
- Did you find differences in performance (accuracy, time and case base size) among the ACBR and the simple CBR?
- (optional) Did you find differences in performance among the ACBR and the weighted ACBR?
- 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 with ACBR, 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 word or pdf document as well as the code in Python in Racó by January, 20th, 2017. The document will be written as a report that details your decisions on the implementation and the results obtained with your analysis.

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%	-