Introduction to Machine Learning
Work 2
Classification with Lazy Learning and SVM

Course 2024-2025

Contents

1 Classification with Lazy learning and SVM

1.1 Introduction

In this exercise, you will use an **SVM algorithm** and you will learn about **lazy learning**. You will apply both machine learning algorithms to a classification task. In particular, in the lazy learning part, you will work on a comparative study in the use of similarity measures, feature weighting methods and instance reduction methods in a k-Nearest Neighbor algorithm. 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 are provided in Campus Virtual.** You cannot modify or perform a different cross-validation on the datasets.

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

You will analyze the behavior of the different algorithms by comparing the results in two well-known data sets (**with 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. Implement your code for reading the `arff` file in Python and store the information in memory. Be careful, some of the data sets contain numerical and categorical data and may also contain missing values. For this exercise you need to store the class of the data set. However, for future assignments you will be requested to not use the class. In the following address:
   https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.arff.loadarff.html

   You will find a code for reading and writing `arff` files in Python. You can implement your own parser (do not use any other library) or you can analyze the code based on `scipy.io` library in the abovementioned link, execute it, and modify it accordingly to your needs. If you need it, for

the numerical datasets, you can transform the continuous variables/attributes into categorical variables using function `pandas.qcut`.

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 instance base, the most used by its simplicity and applicability is a flat structure. The instances 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 instances 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 instance and run each one of the test instances in the selected classifier.

3. Write a Python function for classifying, using a kNN algorithm, each example from the `TestMatrix` using the `TrainMatrix` to a classifier called `kNNAlgorithm(...)`. You decide the parameters for this classifier. The code should be done from scratch, you will write your own code and you cannot use in this part any library or package. Note that instead of retrieving the most similar example, in the kNN algorithm, the similarity function will return the *K* most similar examples. **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 examples (i.e., also known as cases or instances) 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 solution of the query ***q***, you may consider using **three policies**: Majority class, Inverse Distance Weighting 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.

i. **Equal Weight**, all the features are equal and then assign a weight value of 1.0 to all the attributes.

ii. Weights can be extracted using weighting metrics. You may choose **two algorithms** (filter or wrapper, as you wish). Use them as a pre-processing 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. **Justify your implementation and add all the references you have considered for your decisions.**

d. For evaluating the performance of the K-IBL algorithm, we will use the classification **accuracy** (i.e., the average of correctly classified instances) 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 and the average efficiency 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 current_instance *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**. **Use statistical tests to decide which is the best option.**

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

4. Write a Python function for classifying, using an SVM algorithm, each example from the `TestMatrix` using a classifier called `svmAlgorithm(...)` that has been trained with the the `TrainMatrix`. In that case, you will use the SVM algorithm implemented in `scikit-learn` library. You have to setup appropriately the algorithm and use two different kernels (you decide which ones). Use statistical tests to decide which is the best option. **Justify your selection and add all the references you have considered for your decisions.**

   In addition, analyze the results of the `svmAlgorithm` in front of the best `kNNAlgorithm` implementation.

5. Departing from the **best kNN algorithm**. Modify the kNN algorithm so that it includes a pre-processing 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 instance reduction techniques. In this work, you will implement your own code for several preprocessing reduction techniques:
   a. Select one **condensed** reduction technique: Reduced Nearest Neighbor (**RNN**) or Fast Condensed Nearest Neighbor (**FCNN**) or Generalized Condensed Nearest Neighbor (**GCNN**) algorithm.
   b. Select one **edited** reduction technique: **RENN** or Relative Neighbourhood Graph Editing (**RNGE**) or Edited Nearest Neighbor Estimating Class Probabilistic and Threshold (**ENNTh)**
   c. Select one **hybrid** reduction technique: **IB2** 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.
   e. Analyze how the reduction of the training set affects the results obtained by the `svmAlgorithm` compared to the results obtained using the full training set.

Note that some of the reduction algorithms are detailed in:

[3] D. Randall Wilson and Tony R. Martinez. 2000. Reduction Techniques for Instance-BasedLearning Algorithms. Mach. Learn. 38, 3 (March 2000), 257-286. DOI: https://doi.org/10.1023/A:1007626913721 https://link.springer.com/content/pdf/10.1023%2FA%3A1007626913721.pdf

1.3 Work to deliver

In this work, you will use an SVM algorithm and implement a 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.

The idea is that you implement **your own code in Python 3.9 and PyCharm IDE** and you will use it to produce the results of the analysis, and 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. The storage is defined above (point 5.d). For the evaluation, you will use a T-Test or another statistical method [2].

From the accuracy, efficiency, and storage 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. First, you will analyze the KNN, with different parameters combination. You will analyze which is the most suitable combination of the different parameters analyzed. The one with the highest performance will be named as the best KNN algorithm. *Recall to perform a statistical analysis over the different configurations*.

2. You will analyze the SVM, with different parameters combination. You will analyze which is the most suitable parametrization for the algorithm. The analysis will be done in terms of accuracy and efficiency. *Recall to perform a statistical analysis over the different evaluations*.

3. 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 a statistical analysis*.

4. You will also analyze if the reduction in the training set produces different results on the SVM algorithm. *Recall to perform an appropriate statistical analysis*.

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 in the KNN algorithm?
- 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?
- Which is the best kernel for the SVM at each dataset?
- Did you find differences in performance among the k-NNL and the reduction k-NN?
- In the SVM algorithm, did you find differences when reducing the training set?
- 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 in the README.md and include the requirements.txt in your code folder**. 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 **November, 3rd, 2024**.

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

3 Report guidelines

I believe that it will be of great help for you some general **guidelines** for writing the report. It is not a complete list, it contains some comments and suggestions that you may consider in the assignments.

First of all, include a **front cover** with the name and surnames of the group members and a title of the work (this front cover is not part of the length of the report).

The font size should be 11 or 12, not smaller and recall that figures should be also large enough for easier readability.

Analyze the different parameters and use tables or plots to show the results of your evaluation, and justify your decision of the final parameters. Not just saying we have tested several parameters and the best one is X or Y.

Additionally, it is important to **justify your findings**, not to just plot the graph with no comments on it about your observations and your judgement of the behavior of the algorithm in your data. Recall that when you add a comment on a plot, you should also link the comment to the figure/table/plot you are talking about. For example, as shown in Figure X .... or this result indicates that ... (see Figure X).

For the reader of a report, it is difficult to compare graphs if they contain different ranges in their axes and if they are placed in different pages (i.e., more than one page in the middle). It is supposed that **your findings are based on the experiments and the results obtained**. You cannot extract a conclusion if your results do not support it.

Presenting the results in isolation for every one of the datasets helps you to **fix the parameters** and extract conclusions considering the properties of a particular dataset. However, an **overall evaluation** of the methods/algorithms tested over the different datasets is also important to denote the applicability of the method/algorithm to different domains.

If you have read and implemented improvements on the basic algorithms, please add the comment and the references that justify your decision too. The report should contain a section with **references** or bibliography. Remember to add references at the end of the report. (references are not part of the length of the report)

Apart from showing your results and describing them, you **should also answer the questions that are requested in the description of the work**.

Conclusions in a report are important, please, remember to add them in your reports. Not only the general conclusions about what you have done, the conclusions of your findings and your observations of the results.