Learning Data Mining

Riccardo Guidotti
University of Pisa and ISTI-CNR Pisa
guidotti@di.unipi.it

Anna Monreale
University of Pisa
annam@di.unipi.it

Salvatore Rinzivillo
ISTI-CNR Pisa
rinzivillo@isti.cnr.it

***Abstract*—In the last decade the usage and study of data mining and machine learning algorithms have received an increasing attention from several and heterogeneous fields of research. Learning how and why a certain algorithm returns a particular result, and understanding which are the main problems connected to its execution is a hot topic in the education of data mining methods. In order to support data mining beginners, students, teachers, and researchers we introduce a novel didactic environment. The Didactic Data Mining Environment (DDME) allows to execute a data mining algorithm on a dataset and to observe the algorithm behavior step by step to learn how and why a certain result is returned. DDME can be practically exploited by teachers and students for having a more interactive learning of data mining. Indeed, on top of the core didactic library, we designed a visual platform that allows online execution of experiments and the visualization of the algorithm steps. The visual platform abstracts the coding activity and makes available the execution of algorithms to non-technicians.**

I. Introduction

Nowadays, we are assisting to a more and more widespread adoption of data mining and machine learning algorithms in every discipline and in every aspect of everyday life [1]. Indeed, data mining algorithms are currently used in biology, medicine, business and marketing, social science, robotics, etc. but also in smart-phones and ordinary smart devices [2]. Such popularity is certainly due to the predictive and descriptive power that these algorithms have. Clustering methods are used to group the population and understand which are the typical subgroups. Associative rule learning is adopted for discovering interesting relations between variables. Classification algorithms are employed for identifying to which category an observation belongs to, on the basis of a training set of data containing observations whose category is known [3].

Undoubtedly, such pervasiveness has amplified the visibility of data mining and machine learning algorithms making this interesting but complex field one of the most widespread among universities, academies and high education centers. Given the exponential diffusion reached by data mining algorithms, several tools have been developed in order to make them usable by the largest possible number of users [4], [5]. Data mining libraries and platforms are nowadays available for beginners and practitioners with any background: mathematicians, physicists, engineers and computer scientists. Furthermore, some graphical tools have been developed for social scientists, biologists, economists, etc. Some of these tools are adopted for educational purposes. However, the main goal of all the existing tools is to allow the user to easily *use* data mining algorithms and not to *learn* how they work.

To overcome these limitations we propose DDME, a Didactic Data Mining Environment. The goal of this tool is twofold. The first one is to provide a support to *teachers* for the introduction and explanation of the data mining algorithms during their lectures. The second one is to offer to *students* an effective way to understand the details of the functionalities and behavior of the data mining algorithms, and to test their understanding and preparation. It is important to highlight that the DDME is not suitable for auto-learning; in other words, for students is not easy to understand the details of the algorithms without knowing some important theoretical concepts.

DDME provides a collection of data mining algorithms ranging from clustering to classification to association rules. Every algorithm allows the user to tune the parameters mainly affecting the algorithm behavior. DDME algorithms can be run on custom datasets as well as on random datasets that can be generated using functionalities offered by the library. The innovative aspect of DDME is that running a data mining algorithm in such environment provides the intermediate results for each step of the algorithm execution. These results are provided in an intelligible way using text or images, showing the value of a formula or a particular assignment. In this way the DDME user can learn how algorithms works and why a result is returned with respect to certain parameters or settings.

In particular, the Didactic Data Mining Environment currently provides: *K-Means* which is a prototype-based clustering algorithm, *DBSCAN* that is a density-based clustering algorithm, and *Single/Complete Linkage* approaches for hierarchical clustering. Moreover, it offers *Apriori* as association rule learner, and *Decision Tree Classifier* as classification algorithm. The DDME is organized in two layers: the Didactic Data Mining library (DDMlib), and the Didactic Data Mining visual platform (DDMvp). The DDMlib is a Python package that allows to easily build custom experiments using a few lines of an intuitive programming language. The execution of an algorithm shows its internal state and the choices selected for each step of the algorithm. The DDMvp is a web-based visual platform that abstracts from the coding complexity and allows the user to select a dataset and an algorithm with certain parameters, to run the algorithm and study the behavior of the algorithm in the specified conditions without writing any line of code. The DDMvp was provided as an open source and free tool to the students of the "Data Mining" course of the University of Pisa for the academic year 2017-2018. Their evaluation of the DDMvp collected by a survey show that it is a valuable and usable tool for learning data mining.

The idea of learning how data mining algorithms work in order to obtain a satisfactory level of algorithmic transparency is compliant to the actual General Data Protection Regulation (GDPR), recently adopted by the European Parliament[1]. A crucial point of the GDPR is that every algorithmic decision-making or profiling system must provide, to some extent, a "meaningful explanations of the logic involved in the black box" [6], [7]. Despite divergent opinions regarding the real scope of these clauses [8]–[10], everybody agrees that the need for such a principle is urgent, and that its implementation represents today a huge open scientific challenge. Therefore, providing to beginners a didactic environment to understand (part of) the applications they use in their everyday life can absolutely simplify the ways in which the logic involved in these applications must be explained.

The rest of the paper is organized as follows. In Section II we review some of the main tools available to use data mining algorithms and to visualize the results of their execution. Section III recalls basic aspects of the data mining algorithms available in the `DDME` by exploiting for the explanation contents of the `DDME` itself. Section IV provides the details of the Didactic Data Mining Environment. First of all, we highlight how the Python library is designed and how to use it. Then, we describe a web-based visual platform that abstracts from the coding complexity and allows the end user to setup, run and learn how a data mining algorithm works without writing any line of code. Section V discusses the survey results on the tool evaluation, and finally, Section VI concludes the paper underlining the advantages of the `DDME` and providing some insights on the future evolution the framework.

II. Data Mining Tools

The increasing relevance of data mining algorithms has fostered the development of different tools widely used in leading companies as well as in university and academia. These tools allow to easily perform data mining experiments and to apply data mining algorithms in different contexts. To the best of our knowledge, none of the tools currently available provides learning support to *teachers* and *students* like the `DDME`. However, since these tools are the most related to our environment, we propose a review of the most used libraries and visual tools for data mining. Our review will not cover all the existing resources but only those that, in our opinion, provide interesting facilities with a reasonable learning cost.

Python Scikit-learn. The most famous Python package that provides a large set of the most important data mining and machine learning algorithms is `scikit-learn`[2]. `Scikit-learn` algorithms are provided with a quite efficient implementation and are relatively easy to use. An algorithm must be instantiated with a certain parameter setting. After that, the algorithm can be run on a dataset (typically using the *fit* method). Results are accessible as attributes of the algorithm instance. In the `DDME` we use this very common usage pattern. However, differently from `DDME`, `scikit-learn` algorithms only provide the final results without showing intermediate steps. Moreover, `scikit-learn` does not offer any association rule mining algorithm. A Python library that offers programming facilities and a visual platform allowing the user to observe what happens along different time instants is `NDlib` [11]. Indeed, `NDlib` provides access to network diffusion simulation models. Yet in Python, it is worth to mention the *Tensorflow-Playground* that provides a tiny neural network library for educational visualization[3]. Tensorflow-Playground visually shows how a deep neural network works by using appropriate colors assignment for every internal node.

R is another open-source tool and programming language offering various implementations of many data mining and machine learning algorithms, besides statistical data visualizations methods. Differently from Python it does not have a reference library like `scikit-learn` containing various algorithms but every package is specialized for a specific task (e.g. `cluster` for clustering, `arules` for association rules, `party` for decision trees, etc.). Also these libraries do not show why algorithms take certain decisions.

RapidMiner[4] is an open-source platform that provides an integrated environment for data preparation, machine learning, text mining, and predictive analysis [12]. It offers a GUI to design and execute analytical workflows without writing code. The workflows are called *processes* and are composed by different *operators*. Each operator executes a single task within the process, and the output of each operator represents the input of the next one. The engine can be called from other programs or used as a Java API. RapidMiner is used for business applications, research, education, and rapid prototyping.

WEKA[5] is an open-source data mining platform developed in Java [13]. It offers different implementations of data mining algorithms. Also WEKA permits to specify dataflows using connected visual components. WEKA is more oriented towards classification and regression and less towards descriptive statistics and clustering methods. The implemented algorithms of WEKA can be used as API from other programs.

KNIME[6] is an open-source data mining platform developed in Java and based on the visual programming paradigm where the user can model *workflows*, i.e., connected nodes that process and transport data [14]. This platform enables simple integration of new algorithms and tools.

Orange[7] is a data mining tool developed in Python and can be used either by Python scripting, or by a visual programming interface [15]. The available algorithms in this platform are limited with respect to RapidMiner or KNIME.

None of the nodes, workflows or processes of the aforementioned platforms show in any way how the algorithms work and which are intermediate results and critical points useful for learning the algorithms' behaviors.

[1] http://ec.europa.eu/justice/data-protection/

[2] http://scikit-learn.org/

[3] https://playground.tensorflow.org/

[4] https://rapidminer.com/

[5] https://www.cs.waikato.ac.nz/~ml/weka/

[6] https://www.knime.com/

[7] https://orange.biolab.si/

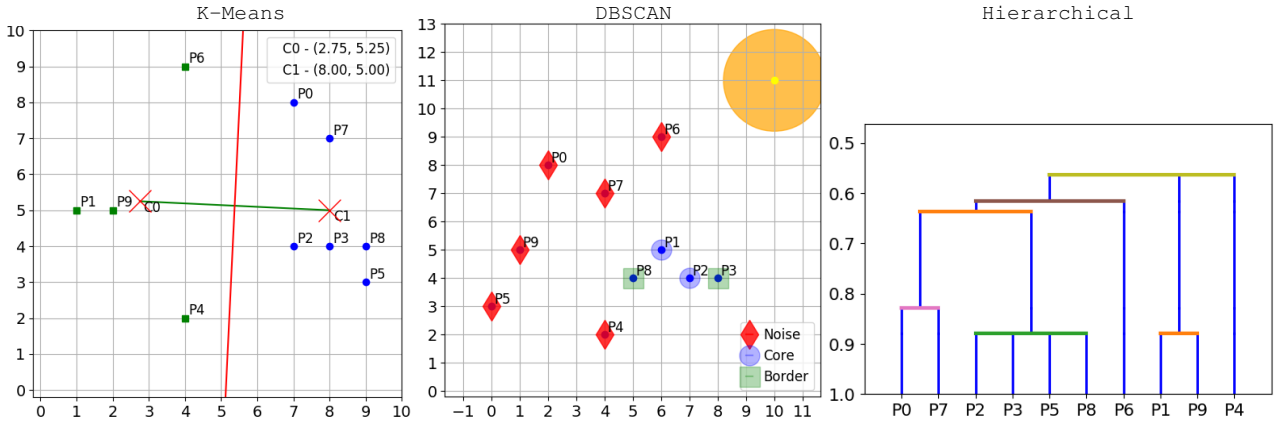

Fig. 1. Execution of the DDME for clustering algorithms: K-Means (left), DBSCAN (center), Hierarchical (right).

III. Data Mining Algorithms

Data Mining is the most important part of the KDD process (Knowledge Discovery in Databases) [16]. The goal of this process is to transform raw data into useful information.

Data Mining blends data analysis methods with sophisticated algorithms for processing large amount of data. The most important data mining tasks are: *clustering*, *association rule mining* and *classification*. They are all covered by the DDME. In this section we recall the main concepts of the algorithms available on the DDME exploiting some images of the DDME itself in order to present how they work.

A. Clustering

Cluster analysis divides data into groups (clusters) of similar points. This subdivision in groups is only based on information found in the data that describes the objects and their relationships. The main goal of this task is to obtain groups where the points within a group are similar and different from the points in other groups [16]. The greater the similarity within a group and the greater the difference between groups, the better the clustering. We can distinguish different types of clustering. A common distinction is between *partitional* and *hierarchical* clustering. The first one is a division of the data points into non-overlapping subsets, while the second one is a set of nested clusters organized as a tree. If points are assigned to more than a cluster, then we obtain a *partial* clustering instead of a *complete* clustering. The DDME provides both partitional and hierarchical clustering algorithms, but only deal with complete clustering algorithms. In the following we provide relevant details highlighted by the DDME for the clustering algorithms implemented.

K-Means [17], [18] is a *prototype-based* clustering algorithm that represents each centroid as the average of all points in the cluster. A *centroid* is a prototype representing the points in its cluster. Given a user-specified number of clusters k, this algorithm randomly selects k points as initial centroids. Then, each point is assigned to the closest centroid based on a distance measure. Finally, the centroids are updated iteratively based on the points assigned to the clusters. The algorithm stops when the centroids do not change. In Figure 1 (left) we report an example of the last step for a run of the K-Means algorithm on the DDME. The ten points are partitioned in two groups (divided by the red line) and the centroids $C0, C1$ are the prototypes of the two partitions.

DBSCAN [19] is a density-based technique that is able to find clusters with any shape by discovering dense areas surrounded by areas with low density, typically formed by noise points. The algorithm measures the density of a certain region with respect to a ε parameter (indicating "the radius"), and then defining clusters as regions that exceed a certain density threshold modeled as minimum number of points min_pts. The final clusters are obtained by connecting neighboring dense regions. More in detail, as shown in Figure 1 (center), DBSCAN identifies three different types of points:

- *Core* points which are in the interior of a cluster. A point is a core point if the number of points within a given ε-neighborhood around the point exceeds the specified minimum number of points min_pts threshold.
- *Border* points which fall within the ε-neighborhood of a core point (i.e., their distance to a core is lower than ε).
- *Noise* points, i.e., any point that is neither a core point nor a border point.

In summary, DBSCAN works as follows. First it labels all points as core, border, or noise points. Then, it eliminates noise points and puts an edge between all core points that are within ε of each other and make each group of connected core points into a separate cluster. Finally, it assigns each border point to one of the clusters of its associated core points.

Hierarchical Clustering [20], [21]. In the literature, there are two of strategies for this king of clustering: *agglomerative* and *divisive*. The first one is a bottom-up approach which starts considering each point as a cluster and then, merges the pairs of *closest* clusters to generate the hierarchy. The divisive strategy, instead, is a top-down approach which starts with a situation where all points belong to one cluster, and then, splits recursively the clusters for creating the hierarchy. The DDME provides the two most famous agglomerative algorithms: Single and Complete Linkage. In order to identify

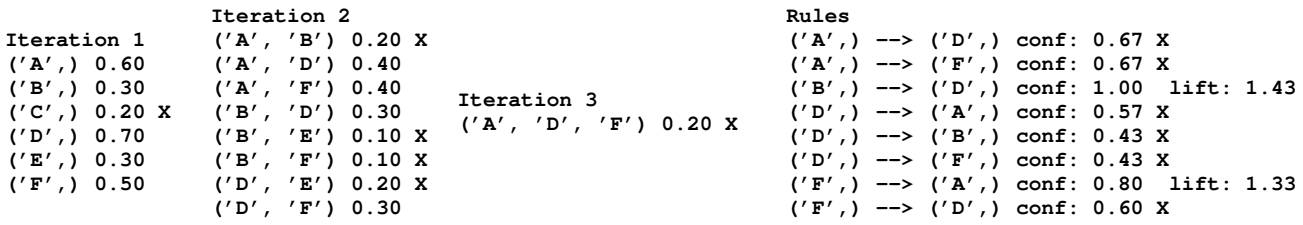

Fig. 2. Execution of the DDME for association rules: Apriori $min\_sup = 0.3$, $min\_conf = 0.8$

the closest clusters for generating the hierarchy they use a different approach. *Single Linkage* defines cluster proximity as the proximity between the *closest* two points in different clusters, while *Complete Linkage* defines cluster proximity as the proximity between the *farthest* two points in different clusters. The returned clusters are represented with a *dendrogram*, i.e., a tree-like diagram which shows both the cluster-subcluster relationships and the order in which the clusters were merged/split. An example of dendogram for the *Single Linkage* version is reported in Figure 1 (right).

B. Association Rule Mining

Association analysis allows to discover the most interesting patterns describing relationships between features in the data in an efficient manner [16].

The relationships that are hidden in the data can be expressed as a collection of *association rules*. Association rules are derived from *frequent itemsets*.

Let $B = \{b_1, \ldots, b_N\}$ be a set of N transactions (or baskets) and $I = \{i_1, \ldots, i_D\}$ a set of D items, a basket b_i is a subset of items such that $\emptyset \subset b_i \subseteq I$. A set of items which are *frequent* in B is called *itemset* or *pattern*. An itemset X is frequent if its *support* is higher than a min_sup parameter. The support over B of an itemset X is defined as $supp_B(X) = |\{b_i \in B | X \subseteq b_i\}|/|B|$. The problem of finding the *frequent itemset* from a dataset of transactions B requires to find in a set of transactions all the itemsets having support greater or equal than min_sup. The search space of itemsets that need to be explored to find the frequent itemsets is exponentially large ($2^D - 1$). Thus, the set of all possible itemsets forms a lattice structure and using a brute force problem makes the problem intractable for large datasets.

Apriori [22] is the most famous algorithm for finding frequent itemsets. Apriori proposes an effective way to eliminate candidate itemsets without counting their support. This algorithm is based on the principle that if an itemset is frequent, then all of its subsets must also be frequent. This principle is used for pruning candidates during the itemset generation. For example in Figure 2, yet from the DDME, itemset $('C',)$ has a support lower than $min\_sup = 0.3$. As consequence, it is not considered in the itemset generation in next iteration.

An association rule is an implication rule of the form $X \rightarrow Y$, where X and Y are disjoint itemsets. X represents the antecedent of the rule while Y the consequent. The confidence of a rule expresses how frequently items in Y appear in transactions containing X, i.e., $conf(X \rightarrow Y) = \frac{supp_B(X \cup Y)}{supp_B(X)}$. Let min_sup and min_conf be the support and confidence parameters, the problem of association rule mining consists in finding all the rules $X \rightarrow Y$ such that $supp_B(X) \geq min\_sup$ and $conf(X \rightarrow Y) \geq min\_conf$. In Figure 2 two rules have the confidence higher than min_conf=0.8.

C. Classification

Given a dataset of records whose class membership is known, classification is the task of identifying to which class a new record belongs to. Classification requires two sequential steps: *(i)* learning a model that assigns each record in the *training set* to a class label, *(ii)* using the model to classify new unknown records belonging to the *test set* [16]. Thus, each record is a tuple $\langle X, y \rangle$ where X contains the attribute values and y the *class label* (also known as category or target). The performance of a classifier are evaluated by counting the test records correctly and incorrectly classified by the model. This information is obtained from the *confusion matrix*. Typical evaluation metrics for predictive models derived by the confusion matrix and also returned by the DDME are *accuracy*, *precision*, *recall* and *F1-score*. In literature, there is a large set of different algorithms to solve the classification problem [16].

Decision Tree Classifier [23]–[25], that is one of the most famous, is present in the DDME. It builds a *decision tree* (see Figure 3 from DDME for an example) containing two type of nodes: *internal nodes* representing a test on one of the attributes (e.g., whether a variable has a value lower than, equals to or grater than a threshold), and *leaf nodes* representing a class label. The paths from the root to the leaves leads to the class label to be associated to the test instance. The most popular implementations of decision tree classifiers, such as ID3 [23] (used in the DDME), C4.5 [24], and CART [25] are based on Hunts algorithm. The learning algorithm starts with a single root node associated with all the training records. The tree is expanded using an attribute determined by a *splitting criterion*. For each value of the selected attribute the algorithm creates a child node and distributes the records of the parent node to the children based on the different values. This step is recursively applied to each child node until all records with the same class are associated to a node. The measures used to select the best splitting attribute are *Entropy*, *Gini Index* and *Misclassification Error*. The DDME supports the last two.

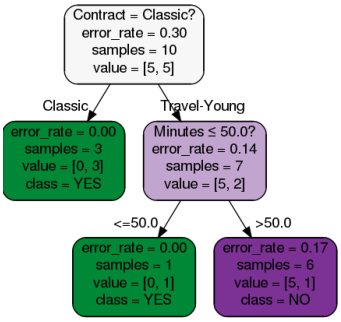

Fig. 3. Execution of the DDME for Decision Tree Classifier.

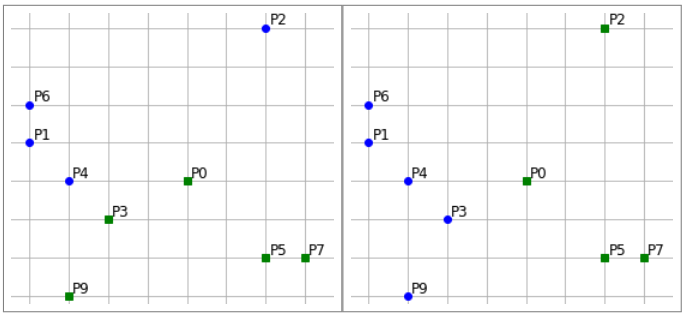

Fig. 4. K-Means with different initial centroids: $P1, P9$ (left), $P1, P2$ (right).

IV. Didactic Data Mining Environment

The study of data mining and machine learning methods is becoming more and more ubiquitous, and understanding how and why these complex algorithms work is mandatory for those wanting to use them. In this section we describe the details of the Didactic Data Mining Environment (DDME), that we propose as support for both *teachers* during the explanation of data mining algorithms, and *students* during their study and preparation. We arranged our environment in two modules: the DDM core Python *library* (DDMlib), and the DDM dynamic *visual platform* (DDMvp). In this section we describe and discuss the main features of each of such components, underlining the critical steps a beginner has to understand for learning how every algorithm works.

Note that, both the DDMlib and DDMvp are made available in different ways. We provide on GitHub the open-source code of both the Python based library[8] and the web application[9]. This enables an easy and free extension of the tool with new features, functionalities and algorithms. We also make available the web application DDMvp ready for being used by teachers and students from different level of education[10]. Moreover, since the tool on its own is not suitable for auto-learning, i.e., is does not to provide all the details necessary for understanding the algorithms, on GitHub[11] we also prepared comprehensive slides for each one on of the algorithm that explain the algorithms and together with the DDMvp offer a complete way to learn all the procedures.

A. DDME - Python Library

DDMlib (Didactic Data Mining library) is a Python package at the core of the DDME built upon the facilities offered by numpy, pandas and matplotlib libraries[12]. DDMlib is available for Python 3.x, and is currently hosted on GitHub. The available data mining algorithms are:

- Clustering: K-Means, DBSCAN, Hierarchical
- Association Rules: Apriori
- Classification: Decision Tree Classifier

[8]https://github.com/riccotti/DidacticDataMining
[9]https://github.com/rinziv/DDM
[10]http://kdd.isti.cnr.it/ddm/
[11]https://github.com/riccotti/DidacticDataMining/tree/master/slides
[12]http://www.numpy.org/, https://pandas.pydata.org/, https://matplotlib.org/

In the rest of the section we analyze how to create an experiment for understanding how an algorithm works. Moreover, we discuss the main parameters affecting the algorithm's behavior, and we analyze the critical steps explained by the DDME.

As anticipated above, DDMlib usage recalls the usage of the sklearn Python library. Given a *dataset*, the DDMlib requires to instantiate an *algorithm* with certain *parameters*, and then to *fit* the algorithm on the dataset. The dataset can be provided by the user, randomly generated or both depending on the algorithm category. The *fit* function visually explains through text and images the behavior of the algorithm besides enabling additional functionalities. Details for each algorithm are provided in the rest of the section.

Clustering Algorithms. The input dataset for clustering algorithms is a list of points with two coordinates[13]. As shown in the following, random datasets can be created using the function create_dataset. The user can specify the number of points (ten by default) and the minimum and maximum values (respectively zero and ten by default).

```
import didactic_datamining as ddm

dataset = ddm.create_dataset(npoints=10)
ddm.print_dataset(dataset)
"P0 [8 1]
 P1 [2 4]
 P2 [6 9]
 ..."
```

K-Means allows to set the number of clusters k (by default $k = 2$), the indexes of the initial centroids (random by default) and the distance function. The available distance functions are Euclidean (set by default) and Manhattan [26] distances. For each iteration of K-Means, the library shows *(i)* the assignment of the points to each cluster using different colors and shapes, *(ii)* the coordinates of the centroids, and *(iii)* the bisecting lines dividing the plane between pairs of centroids (see Figure 1 (left)). A crucial aspect of K-Means is to understand how the final result is affected by the choice of the initial centroids. This can be easily achieved with several runs using different user specified initial centroids. Figure 4 shows exactly this effect reporting the final clustering with different initial centroids: $P1, P9$ (left), $P1, P2$ (right).

[13]This constraint allows to simplify the visualization on a plane and a better comprehension of the basic concepts of the algorithm.

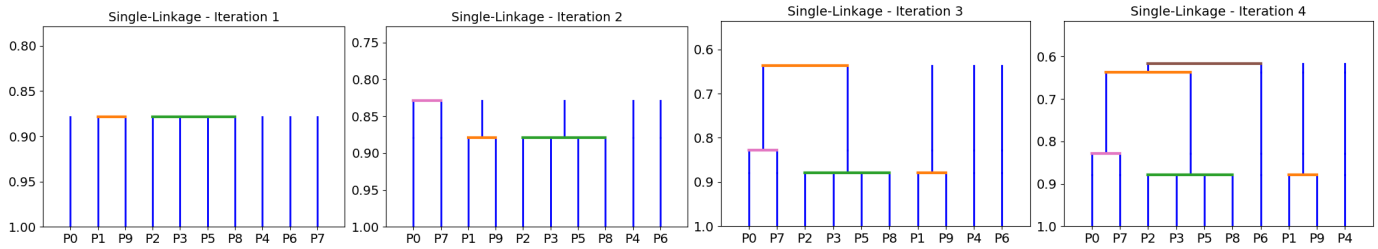

Fig. 5. Hierarchical steps: dendogram construction for each merging step.

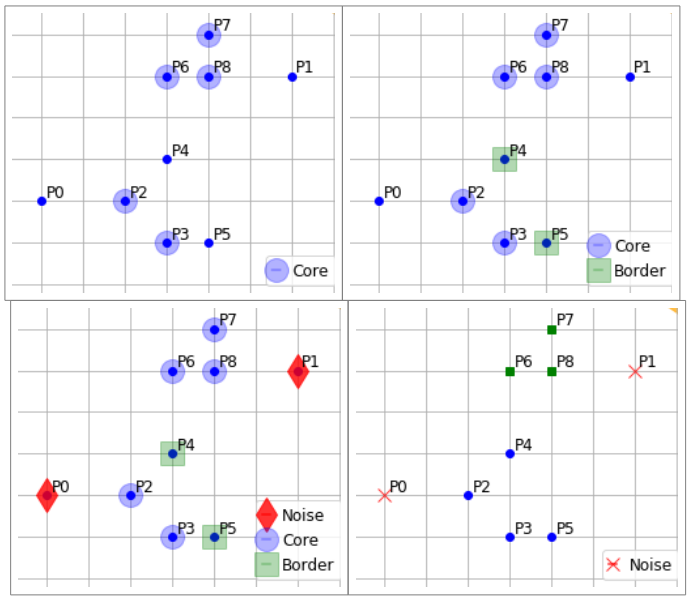

Fig. 6. DBSCAN steps: core, border, noise points and final clustering.

```
kmeans = ddm.DidatticKMeans(centroid_indexs=(0,1))
kmeans.fit(dataset)
```

DBSCAN allows to set the distance radius ε and the minimum number of points min_pts (respectively, 1.8 and 3 by default). The radius is shown visually on the top left of the plot (see Figure 1 (center)). For DBSCAN the library shows the various points categorization into *core*, *border* and *noise* (Figure 6) before presenting the final clustering assignment.

```
dbscan = ddm.DidatticDbscan(eps=1.8, min_pts=3)
dbscan.fit(dataset)
```

Hierarchical clustering offers a large set of options. It implements both Single and Complete linkage algorithms (single by default) and allows to observe the results either using distances (Euclidean by default or Manhattan) or similarities which is obtained as $(1-d)/max_d$, where d is a distance value and max_d is the maximum distance value in the matrix distance between all couples of points. For each merging step the procedure shows the following important aspects: *(i)* the distance/similarity matrix, *(ii)* the merging distance/similarity selected as the minimum (or maximum value depending on Single or Complete Linkage), *(iii)* the step-by-step dendogram showing which points/clusters have been merged. Figure 5 reports an example of the dendogram evolution which ends in the final results already shown in Figure 1 (right).

```
hier = ddm.DidatticHierarchical()
hier.fit(dataset, link_criteria='min')
```

Association Rules Algorithm. The input dataset for association rule algorithms is a list of transactions. Also in this case random datasets can be created using the function `create_transactional_dataset`. The user can specify the number of transactions, the number of items and the minimum and maximum transactions length. By changing these parameters can be created denser or sparser transactional datasets in which itemsets and rules may be discovered using different values of min_sup and min_conf.

```
transactions = ddm.create_transactional_dataset(
    num_transaction=10, num_items=6, max_len=4)
ddm.print_transactions(transactions)
"['A', 'B', 'F'],
 ['A', 'E'],
 ['D', 'F'],
 ..."
```

As shown in the following, the call to the `Apriori` *fit* function returns the itemsets that are considered as possible candidates together with their support. Candidate itemsets with a support lower than the min_sup threshold are marked with a cross `X`. The final result of `Apriori` are the valid itemsets with a support equal or greater than the min_sup threshold along the various steps. For the example in Figure 2 we have the following valid itemsets: $('A',)$, $('B',)$, $('D',)$, $('E',)$, $('F',)$, $('A','D',)$, $('A','F',)$, $('B','D',)$, $('D','F',)$.

```
apriori = ddm.DidatticApriori(min_sup=0.3)
apriori.fit(transactions)
"Apriori - Iteration 1
 ('A',) 0.40
 ('B',) 0.40
 ('C',) 0.30
 ...

 Apriori - Iteration 2
 ('A', 'B') 0.20 X
 ('A', 'C') 0.30
 ...

 Apriori - Iteration 3
 ('A', 'C', 'E') 0.20 X
 ('B', 'C', 'E') 0.40"
```

The `Apriori` module allows also to extract from the valid itemsets the association rules with a confidence greater or equal than *min_conf* as shown in the following. Besides the confidence value, for the valid rules is also reported the lift coefficient. Beginners can play with *min_sup* and *min_conf* observing which itemsets and rules become valid or not and how the search space changes according to them.

```
apriori.extract_rules(min_conf=0.8)
"('A',) --> ('C',) conf: 0.75 X
 ('B',) --> ('C',) conf: 1.00  lift: 1.11
 ('B',) --> ('E',) conf: 0.90  lift: 1.33
 ..."
```

The `Apriori` Python library only displays its steps through a textual visualization. A more appealing and possibly intuitive visualization of `Apriori` is realized in the `DDMvp`.

Classification Algorithm. As formalized in Section III, the input dataset for classification algorithms is set of records of the form $\langle X, y \rangle$ where X contains the attribute values and y the class label. In the `DDMlib` this data format is modeled using a `pandas` dataframe. Two dataframes are necessary to accomplish the classification task: a training set and a test set. The datafrems can be read from existing *csv* files (as in the code reported) or converted directly from a list of Python lists or from a list of Python dictionaries identically formatted. The dataframes appear like those displayed in Figure 7.

```
import pandas as pd
train = pd.read_csv('train.csv', sep=',')
test = pd.read_csv('test.csv', sep=',')
```

For the `Decision Tree Classifier` is possible to choose the splitting criterion function between Misclassification Error (by default) and Gini Index besides the minimum number of records in a leaf and in an internal node (respectively one and two by default). The learning phase that builds the tree is realized by the *fit* function that takes in input the train dataframe and the name of the class to predict (*'Churn'* in the example). This function, besides internally building the classification model, shows the necessary and performed calculus of the splitting criterion for each attribute and recursively for each node. At the end of the execution, it displays the learned decision tree as the one in Figure 3. In the following we report the calculus printed out by the *fit* function for the first split of the root node. For each attribute,

	Minutes	Contract	Sex	Churn
0	50	Classic	F	YES
1	60	Travel	M	NO
2	70	Travel	M	YES
3	80	Young	F	NO
4	90	Travel	F	NO
5	100	Classic	M	YES

	Contract	Sex	Minutes	Churn
0	Classic	F	30	YES
1	Travel	M	60	NO
2	Travel	F	100	NO
3	Travel	M	100	YES

...

Fig. 7. Dataframes: training and test sets for the Decision Tree Classifier.

i.e., *'Contract'*, *'Sex'* and *'Minutes'* in the example, and for all the possible splits, the algorithm calculates the *'Delta Gain'* indicating the improvement with respect to the node purity, and selects the split providing the highest improvement.

```
tree = ddm.DidatticClassificationTree()
tree.fit(train, target='Churn')
"Root Gain: 1 - 5/10 = 5/10

 Contract# ['Travel-Classic', 'Young']
   Travel-Classic 7:  NO, 3/7, YES, 4/7
      1 - 4/7 = 3/7
   Young 3:           NO, 2/3, YES, 1/3
      1 - 2/3 = 1/3
   Gain: (3/7 * 7/10) + (1/3 * 3/10) = 4/10
   Delta Gain: 5/10 - 4/10 = 1/10

 Contract# ['Classic', 'Young-Travel']
   Classic 3:       YES, 3/3
      1 - 3/3 = 0.0/3
   Young-Travel 7:  NO, 5/7, YES, 2/7
      1 - 5/7 = 2/7
   Gain: (0/3 * 3/10) + (2/7 * 7/10) = 2/10
   Delta Gain: 5/10 - 2/10 = 3/10
 ...
 Sex ['F', 'M']
   F 5:             NO, 3/5, YES, 2/5
      1 - 3/5 = 2/5
   M 5:             NO, 2/5, YES, 3/5
      1 - 3/5 = 2/5
   Gain: (2/5 * 5/10) + (2/5 * 5/10) = 4/10
   Delta Gain: 5/10 - 4/10 = 1/10

 Minutes# ['<=50', '>50']
   <=50 2:          YES, 2/2
      1 - 2.0/2 = 0.0/2
   >50 8: NO, 5/8,      YES, 3/8
      1 - 5/8 = 3/8
   Gain: (0/2 * 2.0/10) + (3/8 * 8/10) = 3/10
   Delta Gain: 5/10 - 3/10 = 2/10

 Minutes# ['<=60', '>60']
   <=60 3:          NO, 1/3, YES, 2/3
      1 - 2/3 = 1/3
   >60 7:           NO, 4/7, YES, 3/7
      1 - 4/7 = 3/7
   Gain: (1/3 * 3/10) + (3/7 * 7/10) = 4/10
   Delta Gain: 5/10 - 4/10 = 1/10
 ...

 --> Split By Contract#Young-Travel&Classic

 ..."
```

After that the tree has been learned using the *fit* function, we are enabled by the `Decision Tree Classifier` to use the *predict* and the *evaluate* functions. The *predict* function takes in input the test dataframe and predicts a class label for each record in the test set. The *predict* function assigns the label to a record X by following the splitting condition of the tree according to the values of the attributes in X. This function returns the list of predicted labels for each record. We add this list to the test dataframe in the attribute *'Predicted'* in order to compare them with the real values y.

```
test['Predicted'] = tree.predict(test)
```

Once a test set have been classified by the `Decision Tree Classifier`, we can evaluate its performance using

the *evaluate* function. This function compares the *'Predicted'* attribute values in the test dataframe with the values of the target class *'Churn'*. It reports the *confusion matrix* from which accuracy, precision, recall and F1-score are computed.

```
tree.evaluate(test)
"R\P    |NO     |YES    |
 NO     |3      |1      |
 YES    |2      |2      |

 Precision: 0.67
 Recall: 0.5
 F1-measure 0.57
 Accuracy: 0.63"
```

B. DDME - Visual Platform

The DDMvp is a web visual interactive platform that enables the user to study how the data mining algorithms behave without the need of writing any line of code but yet with the possibility of playing with all the parameters available in the DDMlib library. In particular, the user is enabled to explore the result of the *fit* function for each family of algorithms implemented in the DDMlib library with some additional intuitive visual functionalities.

From a conceptual point of view, the resulting model learned by an algorithm of the DDMlib can be described as a discrete sequence of updates: starting from the input dataset, that can be either a set of points, transactions or records, the mining algorithm begins from an initial configuration of the model and then updates it through several iterations. The DDMvp provides functionalities to present to the user the model learned from the dataset and, through user-interaction, allows to explore the different steps that were evaluated to create the final results.

For each algorithm implemented in the DDMlib, the visual platform DDMvp proposes the following general schema: a graphical widget to select a step iteration i of the mining process, and a visualization of the status of the model at step i, possibly showing the relation of the model with the original dataset similarly to the images produced by the DDMlib. Even though this general schema is instantiated for each algorithm of the DDMlib, in order to enhance the algorithms peculiarities to study, it is implemented in the DDMvp through a specialized visualization for the different algorithms.

To facilitate the diffusion and the use of the library, the visual interface is built for the web. It consists of a single-page web applications developed in HTML, SVG, and Javascript, using the most recent libraries for DOM (Document Object Model) manipulation, like D3.js and Vue.js. In the rest of this section we describe the detailed layout of two completely different algorithms[14], namely K-Means and Apriori.

K-Means. As accurately discussed in Section IV, the K-Means algorithm iteratively updates k centroids and distributes the points in the dataset to the closest centroid. Given a set of k centroids, we can provide a geometrical interpretation of this assignment by dividing the plane into Voronoi cells [27] starting from the position of the centroids. The resulting Voronoi tessellation is a complete partition of the space into convex polygons: all the points associated with centroid j are contained in the polygon generated by j.

As shown in Figure 8, we exploit the interpretation through Voronoi tessellation for presenting to the user the configuration of the the K-Means algorithm at a specific iteration i. We recall that, in order to simplify visualization and relative distance perception for the user, the input dataset consists of two-dimensional points that are visualized into a Cartesian space. During the execution of the K-Means algorithm, at each iteration i, we need two show two pieces of information: *(i)* the position of the centroids at step i, and *(ii)* the association of each data point to its centroid. For an effective visualization of this information we map each centroid to a color in a categorical color scheme and its position in the space is highlighted with a larger marker[15]. To represent the assignment of each data point p to its centroid c_i at step i, we assign to p the color of c_i with an opacity of 40%. Exploiting the geometrical interpretation based on Voronoi cells, we also draw the cell corresponding to the centroid filled with the color of the corresponding centroid and an opacity of 10%.

An example of this interactive evolution is reported in Figure 8. We can immediately notice that the interactive visualization of DDMvp contributes to improve more and more the comprehension of the behavior of K-Means algorithm, that was already made enough clear by DDMlib. The figure shows a series of screenshots of the visualization. However, the web platform implements a transition of visual appearance of the different elements, to better highlight what properties are changing from one state to the successive one.

Apriori. This method to extract frequent itemsets from a list of transactional tuples, is based on the principle that if a given item is not frequent, any other itemset containing this item can not be frequent. The visual interface to present the algorithm is aimed at explaining this principle. The interface follows the general schema of the DDMvp platform: the iteration slider allows the user to select the step of the execution plan and to evaluate how the status of the algorithm is updated. For this specific algorithm, however, at each iteration i we need also reference to the status of previous iterations. For this reason, the visualization explaining the algorithm is built incrementally. When the user choose a specific iteration i, the visual presentation shows all the steps up to iteration i, in order to explain the effect of pruning at a given time.

The visualization presents three different objects: the tuples of the input dataset, the itemsets built at each time step, and the rules extracted from the frequent itemsets. To simplify the perception of the rationale behind the algorithm, each item is represented as visual box associated with a categorical color. The color of each item is kept coherent through all the other steps. This helps also the visual evaluation of support of single items, since it is more effective for the observer to find the occurrences of a specific item. During the execution of the algorithm a set of items is formed by combining single items

[14]We do not discuss all the others for the sake of space.

[15]In our design a circle two times bigger than the dataset points

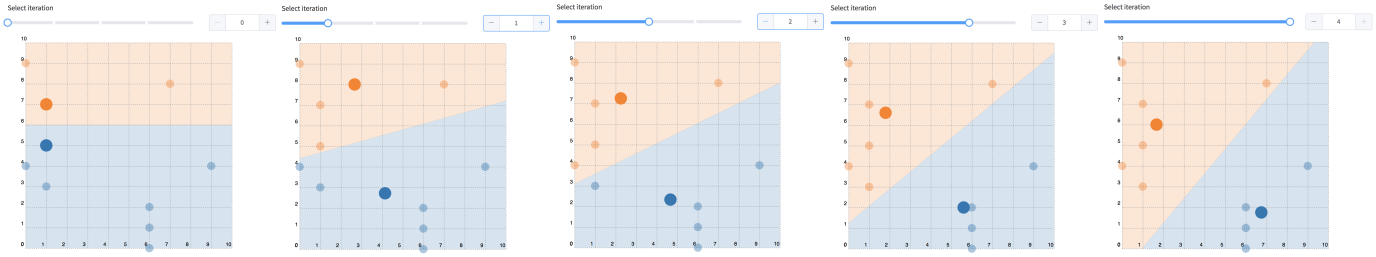

Fig. 8. K-Means clustering complete run on the DDMvp.

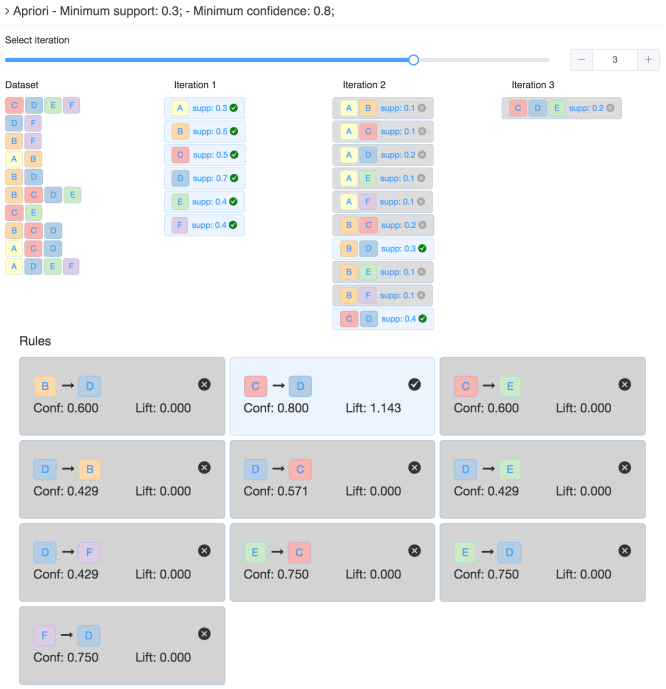

Fig. 9. Apriori run on the DDMvp: 3rd iteration (top), rules (bottom).

(see Section IV for details). For each iteration i we show a list of candidate itemset computed up to step i. For each candidate itemset, the visualization reports the list of contained items (using the same visual metaphor described above) and a measure of support for the item. The visual appearance of each itemset is themed accordingly to the measure being above (or below) a minimum threshold support. If the candidate itemset is frequent, the visual widget is presented with a background color and a symbol indicating the validity of the candidate. On the contrary, if the itemset is not frequent, it is presented with the same visual schema, but with desaturated colors. In this last case, the single items are presented with their original colors, to allow the observer to trace the frequency of the candidate item that is being discarded. At the successive iteration the disabled itemsets are not used to generate the new candidates. When no more candidates can be generated the sequence of iterations is interrupted and a set of rules is generated.

This last step is showed at the end of the visual widget and represents the set of rules extracted from the frequent itemsets. Each candidate rule presents a visual representation of head and body of the rule, recalling the arrow notation of association rules. Each item has a visual representation consistent with the other widgets. For each rule a set of measures are reported, namely confidence and lift. If a rule does not meet the minimum threshold for confidence, it is presented in disabled mode, with desaturated colors. Otherwise, the rules above the threshold are highlighted with a saturated color background for the box and the evaluation of lift measure.

The didactic support of DDME. The DDME, especially with its visual platform, is useful for supporting teachers during the explanation of DM algorithms by dynamically running the algorithms on examples. For instance, with the execution flow in Figure 8 a teacher can enrich her explanation showing the evolution of centroids and the convergence of K-Means. Instead, with respect to Apriori, using the execution flow in Figure 9, a teacher can show in each iteration i how to compute the frequent itemsets with length i and how to derive the rules. It is important to note that, to ease a visual support in the understanding of the algorithms, visual executions are enriched also in the library with functionality like the *circle* to represent the ε-*radius* in DBSCAN (Figure 1), or the mapping of points to the color representing the assigned cluster in K-Means.

V. Academic Survey

The DDMvp was provided as an open source and free tool to the students of the "Data Mining" course of the master degree at the Department of Computer Science of the University of Pisa for the academic year 2017-2018. We collected the student's opinions on the DDMvp for evaluating the utility and usability of the tool for learning data mining.

The survey requests the evaluation of each data mining algorithm by providing a score from 1 (low) to 4 (high) and to answer the following three questions:

Q1: *How much did you use* DDMvp *for the exam preparation?*
Q2: *Is* DDMvp *useful for studying how the algorithms work?*
Q3: *Is* DDMvp *a useful tool for the exam preparation?*

We collected the evaluation of DDMvp from 34 students through a survey after the students passed the exam. Analyzing the answers to question Q1 we discovered that only 6% of these students *never used* the tool for their preparation, while about 20.6% used it *"few times"*, 44% *"occasionally"* and 29.4% used the tool *"a lot"*. The survey results for questions Q2 and Q3 are related only to students who have used the DDMvp (i.e., 32 students). Answers to question Q2 highlight that the DDMvp is considered useful for understanding how the data mining algorithms work step by step. Indeed, we have that 46.87% of students considers it *"very useful"*, 40.62% *"quite*

TABLE I
EVALUATION OF DATA MINING ALGORITHM IN DDMVP

Algorithm	Score			
	1	2	3	4
K-Means	0.0%	14.7%	52.9%	32.4%
DBSCAN	0.0%	14.7%	58.8%	26.5%
Hierarchical	2.9%	20.6%	44.1%	32.4%
Apriori	2.9%	23.5%	32.4%	41.2%
Decision Tree	2.9%	26.5%	41.2%	29.4%

useful" and only 12.5% *"useless"*. Instead, question Q3 allows us to verify the utility of the tool in terms of exam preparation. The answers to Q3 are in line with the results of Q2. We have 43.75% of the students who consider the tool *"very useful"* for the exam preparation, 40.62% *"quite useful"* and 15.63% *"useless"* who did not appreciate it for studying for the exam.

The evaluation of the different algorithms in terms of how intermediate results are presented step by step is reported in Table I. We observe that in general the evaluation is good. Indeed, only a very low number of students provided a very negative evaluation (score equals to 1), while for each data mining algorithm we received a positive evaluation (score equals to 3 or 4) from about 80-85% of the students in case of clustering algorithms, and from around 70% of students for Decision Tree Classifier and Apriori.

VI. CONCLUSION

We have presented the *Didactic Data Mining Environment* a two module-tool consisting of the `DDMlib` Python library and the `DDMvp` visual platform. The `DDME` allows to support data mining beginners and students, as well as teachers, in the study of data mining and machine learning algorithms. Indeed, executions of the algorithms in `DDME` displays in an intuitive and user-friendly way the internal status and steps performed by the algorithms to learn how and why a certain result is returned. The visual platform `DDMvp` abstracts the programmatic interface and makes available the execution of the algorithms also to non-technicians. We have proved the effectiveness of the `DDMvp` in the Data Mining Course for the academic year 2017-2018 at University of Pisa. A survey conducted on the students proved that the `DDME` was considered a valuable and useful tool not only for understanding the algorithms but also for studying and train their-self the exam.

As future work we would like to extend the set of available algorithms. In particular, we would like to extend the association rule algorithms with *FP-Growth* [28] and *ECLAT* [29] that use a different appraoch from Apriori for finding the frequent itemsets. On the other hand, for classification algorithms we would like to introduce simple models such as *K-NN* [16] and *Naive Bayes Classifier* [30], which, together with the Decision Tree Classifier, are at the basis of all the most complex classifiers such as SVM, Neural Networks, etc. Finally, we would like to provide an easy procedure for extending both the `DDMlib` and the `DDMvp`.

ACKNOWLEDGMENT

This work is partially supported by the European Community H2020 Program under the funding scheme "INFRAIA-1-2014-2015: Research Infrastructures" grant agreement 654024 *SoBigData*, http://www.sobigdata.eu.

REFERENCES

[1] M. Nanni, C. Thanos, F. Giannotti, and A. Rauber, "Big data analytics: towards a european research agenda," *ERCIM NEWS*, pp. 9–10, 2015.

[2] F. Provost and T. Fawcett, *Data Science for Business: What you need to know about data mining and data-analytics*. "O'Reilly Media.", 2013.

[3] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh *et al.*, "Top 10 algorithms in data mining," *KAIS*, vol. 14, no. 1, pp. 1–37, 2008.

[4] R. Mikut and M. Reischl, "Data mining tools," *WIREs: Data Mining and Knowledge Discovery*, vol. 1, no. 5, pp. 431–443, 2011.

[5] F. Serban, J. Vanschoren, J.-U. Kietz, and A. Bernstein, "A survey of intelligent assistants for data analysis," *CSUR*, vol. 45, p. 31, 2013.

[6] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, D. Pedreschi, and F. Giannotti, "A survey of methods for explaining black box models," *ACM computing surveys (CSUR)*, 2018.

[7] R. Guidotti, J. Soldani *et al.*, "Helping your docker images to spread based on explainable models," in *ECML-PKDD*. Springer, 2018.

[8] G. Comandè, "Regulating algorithms regulation? first ethico-legal principles, problems, and opportunities of algorithms," in *Transparent Data Mining for Big and Small Data*. Springer, 2017, pp. 169–206.

[9] C. Carter *et al.*, "The credit card market and regulation: In need of repair," *NC Banking Inst.*, vol. 10, p. 23, 2006.

[10] S. Wachter, B. Mittelstadt, and L. Floridi, "Why a right to explanation of automated decision-making does not exist in the general data protection regulation," *IDPL*, vol. 7, no. 2, pp. 76–99, 2017.

[11] G. Rossetti, L. Milli, S. Rinzivillo, A. Sirbu *et al.*, "Ndlib: Studying network diffusion dynamics," in *DSAA*. IEEE, 2017, pp. 155–164.

[12] M. Hofmann and R. Klinkenberg, *RapidMiner: Data mining use cases and business analytics applications*. CRC Press, 2013.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[14] M. R. Berthold *et al.*, "Knime-the konstanz information miner: version 2.0 and beyond," *KDD Newsletter*, vol. 11, no. 1, pp. 26–31, 2009.

[15] J. Demšar *et al.*, "Orange: data mining toolbox in python," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 2349–2353, 2013.

[16] P.-N. Tan, S. Michael, and K. Vipin, *Introduction to data mining*. Pearson Education India, 2006.

[17] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[18] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on MSAP*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[19] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[20] R. Sibson, "Slink: an optimally efficient algorithm for the single-link cluster method," *The computer journal*, vol. 16, no. 1, pp. 30–34, 1973.

[21] D. Defays, "An efficient algorithm for a complete link method," *The Computer Journal*, vol. 20, no. 4, pp. 364–366, 1977.

[22] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *VLDB*, vol. 1215, 1994, pp. 487–499.

[23] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[24] ——, "C4. 5: programs for machine learning," 2014.

[25] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[26] M. M. Deza and E. Deza, "Encyclopedia of distances," in *Encyclopedia of Distances*. Springer, 2009, pp. 1–583.

[27] F. Aurenhammer, "Voronoi diagramsa survey of a fundamental geometric data structure," *CSUR*, vol. 23, no. 3, pp. 345–405, 1991.

[28] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM sigmod record*, vol. 29. ACM, 2000, pp. 1–12.

[29] M. J. Zaki, "Scalable algorithms for association mining," *TKDE*, vol. 12, no. 3, pp. 372–390, 2000.

[30] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.